Intrinsic Gradient Networks

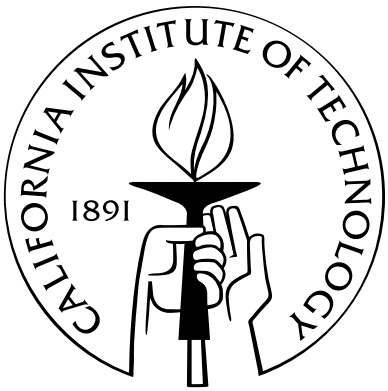
Thesis by

Jason Tyler Rolfe

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2012

(Defended June 9, 2011)

Acknowledgements

This thesis would not have been possible without the help of my advisers: Matthew Cook (my primary adviser for this thesis), Rodney Douglas, Yann LeCun, Pietro Perona, Thanos Siapas, and Erik Winfree. The members of my thesis committee, Jehoshua Bruck, Matthew Cook, Christof Koch, Pietro Perona, and Erik Winfree, provided valuable guidance. Matthew Cook, Rodney Douglas, Eric Downes, Richard Hahnloser, Florian Jug, Sepp Kollmorgen, Christoph Krautz, Yann LeCun, Sylvia Schroeder, and David Soloveichik read various versions of this thesis and contributed numerous helpful comments. During the course of my graduate education, I was financially supported by Thanos Siapas at Caltech, Rodney Douglas at the Institute of Neuroinformatics of ETH Zürich and the University of Zürich, and by a grant from the Mr. and Mrs. Rolfe Fund for Aimless Children.

I trow I hung on that windy Tree
nine whole days and nights,
stabbed with a spear, offered to Odin,
myself to mine own self given,
high on that Tree of which none hath heard
from what roots it rises to heaven.

None refreshed me ever with food or drink,
I peered right down in the deep;
crying aloud I lifted the Runes
then back I fell from thence.[1]

[1] Hávamál, The Words of Odin the High One, from Bray, O. (1982). The elder or poetic edda: Commonly known as Saemund's edda. New York: AMS Press (Original work published in 1908). Edited by D. L. Ashliman.

Abstract

Artificial neural networks are computationally powerful and exhibit brain-like dynamics. Unfortunately, the conventional gradient-dependent learning algorithms used to train them are biologically implausible. The calculation of the gradient in a traditional artificial neural network requires a complementary network of fast training signals that are dependent upon, but must not affect, the primary output-generating network activity. In contrast, the network of neurons in the cortex is highly recurrent; a network of gradient-calculating neurons in the brain would certainly project back to and influence the primary network. We address this biological implausibility by introducing a novel class of recurrent neural networks, *intrinsic gradient networks*, for which the gradient of an error function with respect to the parameters is a simple function of the network state. These networks can be trained using only their intrinsic signals, much like the network of neurons in the brain.

We derive a simple equation that characterizes intrinsic gradient networks, and construct a broad set of networks that satisfy this characteristic equation. The resulting set of intrinsic gradient networks includes many highly recurrent instances for which the gradient can be calculated by a simple, local, pseudo-Hebbian function of the network state, thus resolving a long-standing contradiction between artificial and biological neural networks. We demonstrate that these networks can learn to perform nontrivial tasks like handwritten digit recognition using only their intrinsic signals. Finally, we show that a cortical implementation of an intrinsic gradient network would have a number of characteristic computational, anatomical, and electrophysiological properties, and review experimental evidence suggesting the manifestation of these properties in the cortex.

Contents

Chapter 1

Introduction

Artificial neural networks are both a popular model of neural dynamics (Zipser & Andersen, 1988; Koch, 1999; Poirazi et al., 2003) and a powerful computational architecture (Bishop, 1995; Simard et al., 2003; LeCun et al., 2004; Ciresan et al., 2010). However, before an artificial neural network can be used to perform a computational task, its parameters must be trained to minimize an error function which measures the desirability of the network's outputs for each set of inputs. The most popular and effective algorithms for minimizing the error function of an artificial neural network depend upon the gradient of the error function (Bishop, 2006). The standard algorithm for calculating the gradient of an error function in an artificial neural network, including recurrent neural networks (Almeida, 1987; Pineda, 1987; Pearlmutter, 1989, 1995), is the backpropagation algorithm (Werbos, 1974; Rumelhart et al., 1986). When the backpropagation algorithm is used to train the network via gradient descent, artificial neural networks can achieve state-of-the-art performance on difficult computational tasks similar to those performed by the brain (Simard et al., 2003; Ciresan et al., 2010).

Despite these many strengths, it is generally believed that the backpropagation algorithm is not biologically plausible (Parker, 1985; Grossberg, 1987; Crick, 1989; Stork, 1989; Thorpe & Imbert, 1989; Zipser & Rumelhart, 1993). As suggested by its name, the backpropagation algorithm requires feedback messages to be propagated backwards through the network. These feedback messages, used to calculate the gradient, are directly dependent on the feedforward messages, and so must evolve on the same time scale as the network's inputs. Nevertheless, the feedback messages must not directly affect the original feedforward messages of the network; any influence of the feedback messages on the feedforward messages will disrupt the calculation of the gradient. Direct biological implementations of the backpropagation algorithm are thus not consistent with experimental evidence from the cortex, regardless of whether the feedback messages are carried by a secondary retrograde signaling mechanism or a separate neuronal feedback network, since the cortex neither shows signs of a sufficiently fast reverse signaling mechanism (Harris, 2008), nor segregates feedforward and feedback signals (Douglas & Martin, 2004).

Regardless of the details of the implementation, the biological implausibility of traditional artificial neural networks trained using backpropagation can be traced to two basic properties of the cortex. First, the network of cortical neurons is highly recurrent (Felleman & Van Essen, 1991; Douglas & Martin, 2004). In particular, experimental evidence implies that a directed path consisting of some number of successive synaptic connections links each cortical neuron to every other cortical neuron. As a result, each cortical neuron has the capacity to indirectly influence the activity of every other neuron. If we think of each neuron as a node and each synaptic connection as a directed edge in a graph, then the cortex has a *strongly connected* topology.[1]

Second, the cortex trains its parameters to perform nontrivial tasks efficiently and reliably using a simple, local function of the intrinsic signals within the network of neurons in the brain (Malenka & Bear, 2004). Synaptic strengths and other modifiable neuronal parameters are altered based upon the action potentials and other physical signals that are present at the sites where these parameters are manifested. Distant signals cannot directly affect a parameter in the absence of a direct projection to the site of the parameter.

Together, these two properties imply that the cortex is able to learn efficiently using only local signals within a recurrently interconnected network. That is, the cortical neural network appears to use a single interdependent set of messages for both computation and learning. Neural networks trained with traditional backpropagation algorithms, in contrast, require training signals that depend upon but do not directly affect the other half of the signals in the network. More generally, we are not aware of any existing deterministic neural network model that satisfies these two criteria.[2] In the absence of some other biologically plausible training algorithm, traditional artificial neural networks cannot be an accurate model of cortical computation.

We would thus like to derive a biologically plausible neural network architecture based upon the robust recurrence and efficient local learning observed in the cortex. However, these two properties

[1] The notion of a strongly connected cortical topology should not be confused with the no-strong-loops hypothesis (Crick & Koch, 1998). The no-strong-loops hypothesis is based upon the assumption that a loop of driving connections, each of which can independently induce its target neurons to fire strongly, will necessarily cause activity to oscillate uncontrollably. It thus posits a distinction between driving and modulatory connections, such that modulatory connections cannot independently induce strong firing, and asserts that the cortex and thalamus contain no directed loops of driving connections. In contrast, in claiming that the cortex has a strongly connected topology, we hold that every pair of neurons lies in a directed loop of some sort, potentially consisting of both driving and modulatory connections. A strongly connected network can easily avoid strong loops if, for instance, all feedback connections are modulatory rather than driving.

The networks we construct in section 3.4 have symmetric feedforward and feedback projections, and qualitatively appear to contain strong loops in the sense of Crick & Koch (1998). However, these networks do not exhibit uncontrolled oscillations, in apparent contradiction to the assumption that loops of driving connections necessarily induce uncontrolled oscillations. Strictly speaking, all connections in these example networks are modulatory, since the feedforward and feedback inputs to a unit are combined multiplicatively, and the output is zero if either of these inputs is zero. Nevertheless, the feedforward and feedback connections induce receptive fields much like traditional driving connections. As we discuss in section 4.2.1, these solely "modulatory" dynamics are consistent with the gross electrophysiological properties of the cortex. Intrinsic gradient networks can also be constructed with classically driving connections that obey a strict interpretation of the no-strong-loops hypothesis, but we do not explore such networks in detail in this thesis.

[2] As we shall discuss in section 1.3, while algorithms based on reinforcement learning can be used to train recurrent networks using only locally available signals, they are much less efficient than direct gradient descent, and so cannot plausibly accommodate the massive size of and complicated error function learned by the cortex.

alone do not uniquely specify a neural network architecture, nor do they provide any obvious hints to guide the search for a satisfying architecture. We thus make an additional assumption to narrow our search; based upon biological evidence and computational considerations, we assume that learning requires the approximate calculation of the gradient of an error function. We restrict our attention to networks in which the gradient can be calculated completely at a single network state, rather than stochastic networks in which learning is a function of some statistic of the network activity over time, since it can take a long time to accurately estimate the statistics of large networks. Using these assumptions to focus our search, we ask whether there exist highly recurrent neural networks for which the gradient of an error function defined in terms of the network's activity can be calculated via a simple, local function of the intrinsic network activity.

In approaching this difficult question, we first investigate a slight generalization that is more mathematically tractable. Specifically, we characterize the class of (not necessarily highly recurrent) neural networks for which the gradient of an error function defined in terms of the network's activity is a simple (but not necessarily local) function of the intrinsic network activity. We call this novel class of networks *intrinsic gradient networks.* In contrast to traditional artificial neural networks, intrinsic gradient networks do not require a separate, implicit set of backpropagation dynamics to train their parameters. Some intrinsic gradient networks may not be highly recurrent or may have a non-local gradient-calculating function. Nevertheless, by first identifying a large set of intrinsic gradient networks, we will easily be able to construct a wide variety of highly recurrent, locally trainable instances, which do not suffer from the biological implausibility of traditional artificial neural networks.

1.1 An algorithmic-level model

Intrinsic gradient networks inhabit a space somewhere between biophysically accurate compartment models of spiking neuron dynamics and traditional machine learning techniques. Unlike biophysically accurate models, which model the brain at Marr's implementational level (Marr, 1982), intrinsic gradient networks are intended to capture the algorithm used by the brain without reference to its implementation in terms of ion channels or membrane potentials. Thus, intrinsic gradient networks sacrifice biophysical specificity for computational power. Like other algorithmic-level models, intrinsic gradient networks are compatible with a range of different physical implementations, and make only broad predictions about brain structure and function. In return, intrinsic gradient networks can perform computations similar to those carried out by the brain, such as pattern recognition, whereas even state-of-the-art biophysically accurate models are generally not able to perform nontrivial input-output transformations (Markram, 2006; Izhikevich & Edelman, 2008).[3]

[3]Some models, such as those of Deco & Rolls (2004) and Lundqvist et al. (2006), can perform nontrivial computations while retaining biophysical specificity at the level of individual neurons, but their computational capabilities

On the other hand, the performance of intrinsic gradient networks on machine learning tasks like pattern recognition does not yet match the performance of computational-level models like support vector machines (Cortes & Vapnik, 1995) or feedforward sigmoidal neural networks trained with backpropagation (Rumelhart et al., 1986; Simard et al., 2003; Ciresan et al., 2010). This lack of state-of-the-art performance is unsurprising for an algorithmic-level model. Unlike the computational-level models favored by machine learning, which address a given problem by any means necessary, intrinsic gradient networks strive to capture the particular algorithm used by the brain, and so are limited by the architectural features of the brain. In particular, intrinsic gradient networks can be highly recurrent and locally trainable like the cortex, whereas traditional machine learning techniques almost uniformly use feedforward architectures and learning algorithms dependent upon complex, non-local computations.

Algorithmic-level neural models like intrinsic gradient networks are thus not directly comparable to either implementation-level models or computation-level models. The appropriate comparison is rather to other algorithmic-level models, such as Hopfield networks (Hopfield, 1982), Boltzmann machines (Ackley et al., 1985), adaptive resonance theory (Carpenter & Grossberg, 1987), SOAR (Newell, 1990), recurrent neural networks with contrastive Hebbian learning (O'Reilly, 1996), ACT-R (Anderson & Lebiere, 1998), HMAX (Riesenhuber & Poggio, 1999; Serre et al., 2005), liquid state machines (Maass et al., 2002), and belief propagation on factor graphs (Lee & Mumford, 2003). In following this middle path, we hope that the combination of computational and biological constraints will allow us to come closer to the brain's algorithm than an approach that only makes use of one of these sets of constraints.

An obvious hallmark of the cortical algorithm is that it performs tasks like pattern recognition astonishingly well, significantly outperforming conventional machine learning techniques. As the examples of intrinsic gradient networks presented in this thesis do not achieve state-of-the-art performance on pattern recognition tasks, let alone approach human-level capabilities, we have clearly not identified the cortical algorithm with any precision. However, as we shall show in this thesis, the space of intrinsic gradient networks is enormous, and we have empirically investigated only a handful of tiny examples. We hope that by conforming to the structural features of the cortex, we will be able to derive the space of algorithms within which the cortical algorithm lies. Even if we have been successful, further work will be required to single out the cortical algorithm within this space.

---

fall short of more abstract models (reviewed in Lansner, 2009). The dynamics of these networks are qualitatively similar to a large class of intrinsic gradient networks. It thus seems likely that the particular dynamics of intrinsic gradient networks will be realizable with similar biophysical models, while retaining the desirable learning properties of intrinsic gradient networks.

1.2 Organization of the thesis

In this thesis, we develop a simple mathematical characterization of intrinsic gradient networks. We then use this characterization to construct a large set of intrinsic gradient networks, including locally trainable, highly recurrent networks, and show that these intrinsic gradient networks can be used to perform nontrivial computations. Finally, we explore the connections between intrinsic gradient networks and the cortex.

We begin by developing the theory of intrinsic gradient networks in chapters 1 and 2. We conclude this introductory chapter in section 1.3 by briefly reviewing the literature on biologically plausible learning in neural networks. In section 2.1, we derive and analyze a simple equation that constitutes necessary and sufficient conditions for intrinsic gradient networks. In particular, we find in section 2.1.3 that intrinsic gradient networks must be of a restricted form, dependent upon both the simple functions that compute the gradient from the network activity at identifiable output states, and the functions that define such output states. In section 2.2, we consider the properties of the slack function, which effectively parameterizes the network's behavior prior to reaching an output state, and select a particularly parsimonious and powerful slack function for further examination. We examine the particular case where the gradient is an invertible linear function of an output state in section 2.3. Given this assumption, we find surprisingly simple solutions to the equations characterizing intrinsic gradient networks, including a large set of highly recurrent, locally trainable networks. In section 2.4, we construct the input dynamics of intrinsic gradient networks with common error functions, given the assumptions discussed in the previous sections.

We explore our theoretical characterization of intrinsic gradient networks through a number of examples in chapter 3. The relationship between intrinsic gradient networks and other well-known algorithms is discussed in section 3.1. In particular, we show that both belief propagation on acyclic factor graphs and recurrent backpropagation are degenerate instances of intrinsic gradient networks. In sections 3.2 and 3.3, we construct examples of highly recurrent, locally trainable intrinsic gradient networks with dynamics that are qualitatively similar to loopy belief propagation and sigmoidal artificial neural networks, respectively. We show that these intrinsic gradient networks can be used to solve a generalization of XOR, and apply them to handwritten digit recognition, in section 3.4.

The definition of intrinsic gradient networks was motivated by our desire to understand cortical computation, and in chapter 4 we return to biological considerations. In section 4.1, we examine the biological evidence and computational principles that motivate our interest in intrinsic gradient networks. In section 4.2, we identify some electrophysiological properties that the cortex would manifest if it were implementing an intrinsic gradient network, and evaluate their consistency with existing experimental evidence. We sketch a possible mapping of an intrinsic gradient network onto cortical anatomy in section 4.3. We find that the synaptic learning required in this biologically-

motivated implementation is nearly Hebbian, and that the requisite connection topology is consistent with that observed in the cortex.

We conclude in chapter 5. The appendix A contains full derivations of the results presented in the main text, as well as additional results of a more technical nature.

1.3 Prior work

Due to the success of artificial neural networks as both a computational architecture and a neural model, many prior efforts have been made to reconcile the calculation of the gradient, which is generally used to train neural networks, with the recurrent connectivity and intrinsic local trainability of the cortex. Williams & Zipser (1989) demonstrated that the gradient of an error function can be calculated via forward propagation through a network, but only at the cost of using a separate set of training signals for each parameter. The resulting training signals still must not feed back into the main network. Körding & König (2001) suggested that feedforward signals might be transmitted through low-frequency spiking in parallel with a separate backpropagation signal carried by action potential bursts. Their proposal requires that the basal and apical dendrites perform independent computations on the low-frequency spikes and the bursts, respectively. Hinton & McClelland (1988) and O'Reilly (1996) constructed recurrent networks that heuristically approximate the gradient using the difference between network activity at different stages of convergence, but no bounds on the accuracy of this approximation are known. Xie & Seung (2003) showed that the gradient can be calculated exactly in recurrent neural networks using a two-phase technique similar to that of Hinton & McClelland (1988) and O'Reilly (1996), so long as all forward projections have complementary, reciprocal back-projections with infinitesimal strength. However, the difference between the two resulting converged states is also infinitesimal, and so extremely sensitive to noise.

Roelfsema & van Ooyen (2005) demonstrated that careful choice of the output format and use of a global reinforcement signal can reduce the number of backpropagation signals required, but did not eliminate the need for feedback signals that do not affect the feedforward signals. Barto (1985); Williams (1992); Chialvo & Bak (1999); Seung (2003); and others proposed gradient approximation algorithms based upon a global reinforcement signal, which require no additional backpropagating messages. Such algorithms can even yield internal computations similar to those observed in the brain (Mazzoni et al., 1991), just as in neural networks trained via backpropagation (Zipser & Andersen, 1988), but tend to learn much slower than algorithms that calculate the gradient directly. Calculating the gradient in large, deterministic neural networks quickly, accurately, and using biologically accessible neural signals and biologically plausible connection topologies thus remains an open problem.

Rather than use deterministic dynamics, some authors have constructed stochastic networks for

which the gradient of an error function has a simple, closed-form expression in terms of the expected activities of the units of the network. Notable examples include Boltzmann machines, Helmholtz machines, and deep belief networks, all of which are traditionally trained with gradient descent on the negative log likelihood of the observed inputs (Ackley et al., 1985; Hinton et al., 1995, 2006). However, the calculation of expected activities is NP-hard in the loopy graphical models on which these networks are based (Cooper, 1990). Since the expected activities are required to compute the gradient, in practice, the gradient must be approximated rather than calculated exactly. Known efficient approximations depend upon adopting a restricted architecture in which recurrent interactions are limited (Smolensky, 1986), in contradiction to the highly recurrent structure of the brain (Felleman & Van Essen, 1991; Douglas & Martin, 2004). Even with such restricted architectures, only coarse approximations of the marginal probabilities are possible in polynomial time (Long & Servedio, 2010), and such approximations can mislead learning algorithms which are not appropriately matched to the inference technique, sometimes leading to surprisingly poor results (Kulesza & Pereira, 2008; Fischer & Igel, 2010). Indeed, deep belief networks generally use non-recurrent and thus biologically implausible mechanisms to fine-tune their parameters after initializing with partially recurrent dynamics (Hinton et al., 2006).

With carefully chosen architectural restrictions, these approximate, probabilistic algorithms can be rendered exact and deterministic. For instance, when using an acyclic (i.e., tree-structured) graphical model, a message passing technique called belief propagation can calculate the expected activities (and thus the gradient of the negative log likelihood) exactly and in linear time (Pearl, 1988; Kschischang et al., 2001). The corresponding network dynamics have the same form as a neural network with product units (Durbin & Rumelhart, 1989). As a result, in an acyclic network implementing belief propagation, with the negative log likelihood as an error function, the gradient of the error can be computed directly from the network activity, without the need for a complementary backpropagation network. Unfortunately, the requisite tree structure is a poor model of the brain's highly recurrent architecture (Felleman & Van Essen, 1991; Douglas & Martin, 2004). Moreover, tree-structured graphical models do not efficiently capture the statistical structure of many real-world phenomena, for which correlations generally depend on spatial locality in two or three dimensions, naturally inducing many loops.

Chapter 2

Theory

Intrinsic gradient networks are the novel class of neural networks for which the gradient of an error function defined in terms of the network's activity can be calculated simply from the intrinsic network activity. In section 2.1, we derive a system of nonlinear partial differential equations which characterizes intrinsic gradient networks in terms of four sets of functions: the output functions, which identify when the network has finished computing and has produced an output; the error function, which defines the desirability of the outputs; the slack function, which controls the behavior of the network before an output has been produced; and the training function, which helps calculate the gradient of the error function from the intrinsic network activity after an output has been produced. We find in section 2.2 that a particular slack function and a linear training function is required to construct modular intrinsic gradient networks, and discuss the implications of this slack function. We construct a large class of modular intrinsic gradient networks, satisfying these and a few additional restrictions, in section 2.3. Finally, in section 2.4 we construct intrinsic gradient networks with a variety of error functions.

2.1 A mathematical characterization of intrinsic gradient networks

We begin by describing the structure of intrinsic gradient networks. We then sketch a derivation of a simple equation that characterizes intrinsic gradient networks. In the remainder of this thesis, we will find solutions to this characteristic equation, explore their computational properties through some simple examples, and discuss their biological implications.

2.1.1 The structure of computation in a highly recurrent network

The structure of intrinsic gradient networks reflects the nature of computation in highly recurrent networks such as the cortex. Computation cannot occur instantaneously in a highly recurrent

network, since the inputs propagate through the network gradually, and generally must circulate through the network repeatedly before the computation is completed and the outputs are generated. If computation is based solely upon the intrinsic signals, the network itself must identify when the computation is finished and the outputs have been produced. We thus develop intrinsic gradient networks with reference to discrete, identifiable output states, although we do address the use of intrinsic gradient networks to model systems with continuous outputs in section 4.1.1.

We model the cortex as a time-varying vector of real-valued units $\vec{x}(t) \in \mathbb{R}^n$. We write $\vec{x}$ to denote $\vec{x}(t)$ at some arbitrary point in time t. The brain's ability to identify when a computation has been completed implies that there exists some vector of functions $\vec{Z}(\vec{x})$ such that $\vec{Z}(\vec{x}) = \vec{0}$ when a computation is finished and $\vec{x}$ contains the output, where $\vec{0}$ is a vector in which each element has the value 0. We will find it convenient to consider the vector of functions $\vec{F}(\vec{x}) = \vec{Z}(\vec{x}) + \vec{x}$, so that $\vec{F}(\vec{x}) = \vec{x}$ when $\vec{Z}(\vec{x}) = \vec{0}$ and a computation is complete. In particular, it will prove much more intuitive to characterize the subset of $\mathbb{R}^n$ within which $\vec{F}(\vec{x}) = \vec{x}$ than the equivalent subset within which $\vec{Z}(\vec{x}) = \vec{0}$. Nevertheless, any derivation in terms of $\vec{F}(\vec{x})$ can be used to construct a corresponding derivation in terms of $\vec{Z}(\vec{x})$. We refer to the functions F_i as *output functions*, since they determine when the output has been computed. Since the output states are the fixed points of $\vec{F}(\vec{x})$, we will often use the term *fixed points (of the network)* to refer to the output states where $\vec{F}(\vec{x}) = \vec{x}$. The biological plausibility of this formalism is explored in section 4.1.2.

Somewhat counterintuitively, although the output functions $\vec{F}(\vec{x})$ identify when the computation is complete, they define the computation performed by the network without regard to the network dynamics. In particular, the fixed points of $\vec{F}(\vec{x})$, which constitute the output states, need not be the same as the fixed points of the network dynamics. As an extreme case, consider network dynamics where $\vec{x}(t)$ evolves randomly in time. In this case, the cortical network effectively guesses random solutions to the problem of generating an acceptable output, and each of these random guesses is checked for consistency according to $\vec{F}(\vec{x}) = \vec{x}$ (or equivalently $\vec{Z}(\vec{x}) = \vec{0}$). The outputs of this random network will be the same as if the dynamics smoothly converged to a fixed point of $\vec{F}(\vec{x})$. Different network dynamics will vary in the speed with which they reach a fixed point of $\vec{F}(\vec{x})$, and some may never produce an output. Nevertheless, the functions F_i alone define the network's output, and so are properly the subject of learning if the error function is defined in terms of the network's output. As such, we will define intrinsic gradient networks in terms of the output functions F_i, and only consider the network dynamics secondarily.

The output functions are also necessary to characterize learning in intrinsic gradient networks because they define unique states in a potentially labile network. In an unstable network without an identified point at which to calculate the gradient of the error function from the network activity, either the *training function* $\vec{T}(\vec{x})$, which helps calculate the gradient from the network state, or the calculated gradient itself must change over time, even given constant input to the network. In a

neural context, this problem is not easily resolved by making the training function dependent on the time since stimulus onset, since such a reference point is difficult to define when various features of the stimulus change continuously and asynchronously. We address the potential variability of a training function that is dependent upon a dynamic network state by assuming that the calculated gradient is exact at the fixed points of the output functions $\vec{F}(\vec{x})$, but not excluding the possibility that suboptimal parameter updates might be performed based upon other network states. Indeed, optimal learning at the fixed points of the output functions can be seen as a necessary but not sufficient constraint for optimal learning at all points in the network's trajectory.

Defining the network outputs in terms of the fixed points of the output functions $\vec{F}(\vec{x})$ can help render the network robust to noise. At most, noise can knock the network from one fixed point into a different fixed point; it cannot change the identity of the fixed points. So long as all fixed points are valid outputs at which the computation is complete, noise can delay the output, but cannot corrupt it.

In principle, the gradient could be calculated based upon the network activity at different points in time than those at which the output is produced. That is, the functions used to identify the points at which the output is produced (what we might call output-output-functions $\vec{F}_{output}(\vec{x})$) might be different from the functions used to identify the points at which the parameters can be trained using the training function (which we might then call training-output-functions $\vec{F}_{training}(\vec{x})$). However, if multiple network configurations can be identified as output states for a given input, then the gradient of the error function with respect to the parameters will generally be different at each of these output configurations. The gradient at each such output state would be most parsimoniously calculated based upon a distinct, corresponding training state. It would likely be difficult to reliably select the right fixed point of the training-output-function, corresponding to the last selected fixed point of the output-output-function. To avoid this problem, we assume that the gradient is calculated based upon the network activity at the same point that constitutes the output, identified using a single set of output functions $\vec{F}(\vec{x})$.

2.1.2 The framework of intrinsic gradient networks

Consider a vector of differentiable functions $\vec{F}$, with $F_i : \mathbb{R}^n \to \mathbb{R}$, which characterize the output configurations of a network of real-valued units $\vec{x}(t) \in \mathbb{R}^n$, such that a state $\vec{x}$ is an output if and only if $\vec{F}(\vec{x}) = \vec{x}$. All of our results depend only on the network recognizing that a computation has been completed and that $\vec{x}$ constitutes an output when $\vec{F}(\vec{x}) = \vec{x}$. Any set of dynamics may be used for $\vec{x}(t)$, so long as $\vec{F}(\vec{x}) = \vec{x}$ is used to define the output states of the network.

Nevertheless, since poorly chosen dynamics are likely to find an acceptable output state very slowly, the dynamics of units $\vec{x}$ in the cortex would certainly be related to the output functions $\vec{F}$. A particularly simple and biologically plausible situation arises if the fixed points of the temporal

dynamics of the units $\vec{x}$ are the same as those of the vector of functions $\vec{F}(\vec{x})$. In this case, the network indicates the completion of a computation by settling into a stable activity pattern. A simple set of dynamics with the desired fixed points is

$$\vec{x}(t+1) = \vec{F}(\vec{x}(t))\,, \tag{2.1}$$

for which the output functions determine the output of each associated unit at the next time step, as well as identifying the final outputs of the network as a whole. Almost all of our results can be understood using the dynamics of equation 2.1. However, there are many sets of dynamics with fixed points that match the fixed points of $\vec{F}$. For instance, the dynamics of equation 2.1 can be rendered continuous according to

$$\frac{d\vec{x}(t)}{dt} = \frac{1}{\tau} \cdot \left(\vec{F}(\vec{x}(t)) - \vec{x}(t)\right)\,, \tag{2.2}$$

with $\tau \in \mathbb{R}^+$. In appendix A.5, we will show that the dynamics of equation 2.2 are provably convergent for a large subset of intrinsic gradient networks. Equation 2.2 can be further discretized to yield

$$\vec{x}(t+1) = \frac{1}{\tau} \cdot \vec{F}(\vec{x}(t)) + (1 - \frac{1}{\tau}) \cdot \vec{x}(t)\,, \tag{2.3}$$

which is equivalent to equation 2.1 when $\tau = 1$. We use the dynamics of equation 2.3 in the examples described in section 3.4.

We assume that the output functions $\vec{F}$ are collectively parameterized by a vector of real numbers $\vec{w}$. Strictly, we should write $\vec{F}(\vec{x}, \vec{w})$, but will often omit explicit mention of the dependence on $\vec{w}$ to avoid cluttering our notation. Just as each neuron in the cortex receives direct input from only a small fraction of the other cortical neurons, we generally assume that each output function F_i only depends on a subset of the units x_j.

The desirability of an output state, where $\vec{F}(\vec{x}) = \vec{x}$, is defined by an error function $E(\vec{x}, \vec{w})$. We restrict our attention to error functions $E(\vec{x}, \vec{w})$ defined in terms of the output states; that is, the fixed points of $\vec{F}(\vec{x})$. We wish to minimize $E(\vec{x}, \vec{w})$ at the fixed points of $\vec{F}(\vec{x}, \vec{w})$ by training the parameters $\vec{w}$. In practice, we will want to minimize the average value of the error function $E(\vec{x}, \vec{w})$ over an ensemble of different output states corresponding to various inputs to the system. Since the gradient operator is linear, the gradient of such a weighted sum of component error functions is equal to the weighted sum of the gradients of the components, and the ability to calculate the gradient of $E(\vec{x}, \vec{w})$ for each output state $\vec{x}$ separately implies that the gradient of such an average error function is also easily calculable. We thus further restrict our attention to error functions and gradients defined in terms of a single output state at a time.

Intuitively, this formalism is intended to model spike rates (or some other function of the spiking

activity, as discussed in section 4.1) in a network of neurons, with neuron outputs $\vec{x}$ and transfer functions $\vec{F}$. In this case, the output states are the fixed points of the network dynamics. We imagine that the elements of $\vec{x}$ on which $F_i(\vec{x})$ directly depends are physically connected to x_i, and the parameters of $F_i(\vec{x})$ are part of a single physical system that generates the dynamics of x_i. It is thus biologically plausible that these local signals could contribute to the training of the parameters of F_i in a biological neural network.

Our analysis treats inputs rather differently from how they are treated in a conventional formulation of supervised learning. As can be seen in equations 2.1 and 2.2, we do not model the inputs explicitly in our equations. Rather, our analysis will lead us (in section 2.1.4) to essentially discover that the parameters $\vec{w}$ of the system naturally fall into two categories. All parameters $\vec{w}$ affect the value of the error function $E(\vec{x}, \vec{w})$ at the fixed points of $\vec{F}(\vec{x}, \vec{w})$ by altering the location of those fixed points. However, the parameters w_i of one category also directly affect the value of the error function (the gradient of which is calculated by the training function $\vec{T}(\vec{x})$) at each potential fixed point $\vec{x}$. It is normal for an error function to depend directly on the inputs to a system, as well as on the outputs, since the desired output values change as a function of the input values. For this reason, we will refer to these parameters w_i which directly affect the error function $E(\vec{x}, \vec{w})$ as *input parameters.* We will not concern ourselves with whether their components of the gradient of the error function are computationally tractable, since inputs are not trainable parameters anyway. To avoid overly cluttered notation, we often write $E(\vec{x})$ instead of $E(\vec{x}, \vec{w})$, even though the error function is also dependent upon the input parameters.

The parameters w_i of the other category only affect the value of the error function insofar as they have an effect on the output states $\vec{x}$ of the network. We will refer to these as *internal parameters,* and will require that the gradient with respect to these parameters be easily computable. Even though the error function is not directly dependent on the internal parameters, the internal parameters affect the error function through their impact on the fixed points of the output functions $\vec{F}(\vec{x})$, and thus on the output states of the network. The distinction between input parameters and internal parameters will be discussed again in section 2.1.4, and we will provide examples in section 2.3.1.

In general, we will not directly address the dynamics $\vec{x}(t)$ of intrinsic gradient network units in this thesis. Rather, we focus on the output functions $\vec{F}(\vec{x})$, which specify the location of the output states, but do not directly determine the behavior of the network. The rate and reliability with which the dynamics reach a fixed point of $\vec{F}(\vec{x})$ is primarily a function of the global network dynamics themselves, rather than the location of the fixed points of $\vec{F}(\vec{x})$. While we often refer to the dynamics of equations 2.1, 2.2, and 2.3 to provide concrete examples of intrinsic gradient networks, we have little reason to believe that these particular dynamics will be optimal in practical applications. In an important exception to this focus on the output functions $\vec{F}(\vec{x})$ rather than the dynamics $\vec{x}(t)$, however, we do find provably convergent dynamics for a large set of intrinsic gradient

networks in appendix A.5.

2.1.3 Sketch of the derivation of the intrinsic gradient equation

We wish to derive restrictions on the set output functions $\vec{F}(\vec{x}, \vec{w})$ for which the gradient of an error function $E(\vec{x}^*, \vec{w})$ with respect to the internal parameters can be calculated simply at the fixed points $\vec{x}^*$ of $\vec{F}$, where $\vec{F}(\vec{x}^*) = \vec{x}^*$. Networks with output functions of this form are intrinsic gradient networks. Appendix A.1 contains two distinct, full derivations of necessary and sufficient criteria such that the gradient of an error function can be simply calculated from the network state at the fixed points of $\vec{F}(\vec{x})$. A sketch of the first of these derivations, including the definition of concepts and terms that will be used throughout the rest of this thesis, follows below.

The value of the error function $E(\vec{x}^*, \vec{w})$, which is defined in terms of the fixed points $\vec{x}^*$ of $\vec{F}(\vec{x}, \vec{w})$, depends on the location of the fixed point $\vec{x}^*$ at which it is evaluated. The output functions $\vec{F}(\vec{x}, \vec{w})$ determine the location of the fixed points $\vec{x}^*$, and are themselves dependent on the parameters $\vec{w}$. Therefore, the fixed points $\vec{x}^*$ are also a function of the parameters, although their dependence on the parameters is generally nontrivial. The gradient of the error function with respect to the internal parameters $\frac{dE(\vec{x}^*)}{d\vec{w}}$ must take this relationship between the fixed points and the internal parameters into account. As a simple example, consider the one-dimensional system with the output function $F(x) = \frac{w^2+x}{2}$ and the error function $E(x^*) = (x^* - c)^2$. In this case, $x^* = w^2$, so although $\frac{\partial E(x^*)}{\partial x^*} = 2 \cdot (x^* - c)$, to find the derivative with respect to the internal parameter w we must compute

$$\begin{aligned}\frac{dE(x^*(w))}{dw} &= \frac{\partial E(x^*(w))}{\partial x^*} \cdot \frac{dx^*(w)}{dw} \\ &= (2 \cdot [x^*(w) - c]) \cdot (2 \cdot w)\,.\end{aligned}$$

In general, the gradient of the error function with respect to the internal parameters is the inner product of the partial derivative of the error function with respect to the fixed point, and the total derivative of the fixed point with respect to the parameters.[1] Unfortunately, the total derivative of the fixed point with respect to the parameters is generally difficult to compute, since any alteration to the parameters induces changes in the units that resonate throughout a recurrently connected network.

Fortunately, we can derive conditions in which the gradient of the error function with respect to the internal parameters is easy to compute from the fixed points of the network. To find these

[1] By definition, the error function $E(\vec{x}, \vec{w})$ is not directly dependent on the internal parameters, so $\frac{\partial E(\vec{x}, \vec{w})}{\partial w_i} = 0$ if w_i is an internal parameter. Although $\frac{\partial E(\vec{x}, \vec{w})}{\partial w_j}$ need not be equal to zero for input parameters w_j, we are not interested in calculating the full derivative $\frac{dE(\vec{x}, \vec{w})}{dw_j}$ with respect to the input parameters, since inputs are not subject to training in traditional learning problems.

conditions, we take the derivative of the fixed point with respect to an internal parameter w', apply the chain rule, and solve for the derivative of the fixed point:

$$\begin{aligned}
\vec{x}^*(\vec{w}) &= \vec{F}(\vec{x}^*(\vec{w}), \vec{w}) \\
\frac{d\vec{x}^*(\vec{w})}{dw'} &= \frac{d\vec{F}(\vec{x}^*(\vec{w}), \vec{w})}{dw'} \\
&= \left(\frac{\partial \vec{F}(\vec{x}, \vec{w})}{\partial w'}\bigg|_{x^*} \right) + \left(\nabla^\top \vec{F}(\vec{x}, \vec{w})\big|_{x^*} \right) \cdot \frac{d\vec{x}^*(\vec{w})}{dw'} \\
&= \left(\mathbf{I} - \nabla^\top \vec{F}(\vec{x}, \vec{w})\big|_{x^*} \right)^{-1} \cdot \left(\frac{\partial \vec{F}(\vec{x}, \vec{w})}{\partial w'}\bigg|_{x^*} \right), \qquad (2.4)
\end{aligned}$$

where ∇ is the gradient with respect to $\vec{x}$ and $\nabla^\top \vec{F}(\vec{x}, \vec{w})$ is the Jacobian. In the equations below, all gradients and partial derivatives are evaluated at the fixed point x^*, although we do not indicate this explicitly to avoid cluttered notation. Since $\frac{\partial E(\vec{x}, \vec{w})}{\partial w'} = 0$ for the internal parameters,[2] we can apply equation 2.4 to the derivative of the error function with respect to internal parameter w' to find

$$\begin{aligned}
\frac{dE(\vec{x}^*(\vec{w}))}{dw'} &= \frac{\partial E(\vec{x})}{\partial w'} + \left(\nabla^\top E(\vec{x}) \right) \cdot \frac{dx^*(\vec{w})}{dw'} \\
&= \left(\nabla^\top E(\vec{x}) \right) \cdot \left(\mathbf{I} - \nabla^\top \vec{F}(\vec{x}, \vec{w}) \right)^{-1} \cdot \frac{\partial \vec{F}(\vec{x}, \vec{w})}{\partial w'} . \qquad (2.5)
\end{aligned}$$

Equation 2.5 can be divided into two parts: $\frac{\partial \vec{F}(\vec{x}, \vec{w})}{\partial w'}$, which is dependent upon the parameter w' for which we wish to calculate $\frac{dE(x^*)}{dw'}$, and is easy to compute; and $\left(\nabla^\top E(\vec{x}) \right) \cdot \left(\mathbf{I} - \nabla^\top \vec{F}(\vec{x}, \vec{w}) \right)^{-1}$, which is the same for all parameters, but hard to compute in that it requires the inversion of a matrix of size $n \times n$, where n is the number of units $|\vec{x}|$.

We require that the hard part of the gradient be computable from the fixed point $\vec{x}^*$ by some simple, easily calculated *training function* $\vec{T}(\vec{x}^*, \vec{w})$, and thus define

$$\vec{T}^\top(\vec{x}^*, \vec{w}) = \left(\nabla^\top E(\vec{x}) \right) \cdot \left(\mathbf{I} - \nabla^\top \vec{F}(\vec{x}, \vec{w}) \right)^{-1} , \qquad (2.6)$$

so

$$\frac{dE(\vec{x}^*(\vec{w}))}{dw'} = \vec{T}^\top(\vec{x}^*, \vec{w}) \cdot \frac{\partial \vec{F}(\vec{x}, \vec{w})}{\partial w'} . \qquad (2.7)$$

The training function $\vec{T}(\vec{x}^*, \vec{w})$ is the non-obvious component of the simple (and preferably local) function that computes the gradient of the error function from the intrinsic network activity at the fixed points of $\vec{F}$, referred to in the definition of intrinsic gradient networks. The training function

[2] Once again, the partial derivative of the error function E with respect to parameter w' is not necessarily zero if w' is an input parameter. However, the input parameters are not trainable, so we have no need to calculate the derivative of the error function with respect to the input parameters. It is thus safe to assume that $\frac{\partial E(\vec{x}, \vec{w})}{\partial w'} = 0$.

extracts the information required to compute the gradient from the fixed point. The real work of the gradient computation is performed by the network itself as it moves towards a fixed point x^* of $\vec{F}$; the function $\vec{T}(\vec{x}^*, \vec{w})$ merely decodes the output of the network after it reaches a fixed point.

In the simple one-dimensional example described above, $T(x^*, w) = 4 \cdot (x^* - c)$ and $\frac{\partial F(x,w)}{\partial w} = w$. We find in appendix A.1 that if any function can calculate the gradient of the error function from the intrinsic signals of the network at a fixed point, then a training function $\vec{T}(\vec{x}^*, \vec{w})$ satisfying equation 2.7 must necessarily exist. To reduce the complexity of our exposition, we generally consider training functions $\vec{T}(\vec{x}^*, \vec{w})$ that are independent of $\vec{w}$, although most of our results can easily be extended to training functions that are dependent on $\vec{w}$. As with $\vec{F}(\vec{x})$, we often write $\vec{T}(\vec{x})$ and omit explicit mention of the dependence on the parameters or the fixed point to avoid excessively cluttered notation.

We want to construct output functions $\vec{F}$ that are compatible with simple, local training functions $\vec{T}$, so that after an intrinsic gradient network reaches a fixed point, it is easy to compute the gradient of the error function $\frac{dE(x^*)}{d\vec{w}}$ and train the parameters using equation 2.7. We thus need to find a relationship between the training function $\vec{T}$ and the output functions $\vec{F}$ that is consistent with equation 2.7, but does not explicitly depend upon the difficult-to-calculate total derivative of the error function. Right-multiplying both sides of equation 2.6 by $\mathbf{I} - \nabla^\top \vec{F}(\vec{x}, \vec{w})$ and rearranging, we find that our definition of $\vec{T}(\vec{x}^*, \vec{w})$ is satisfied if and only if

$$\vec{T}(\vec{x}^*, \vec{w}) = \nabla E(\vec{x}) + \left(\nabla \vec{F}^\top(\vec{x}, \vec{w})\right) \cdot \vec{T}(\vec{x}^*, \vec{w}) \tag{2.8}$$

at fixed points $\vec{x}^*$ of $\vec{F}(\vec{x}, \vec{w})$, so we wish to find $\vec{T}$ and $\vec{F}$ such that this equation is satisfied for a given E. In particular, we choose simple, local $\vec{T}$ and then construct $\vec{F}$ to satisfy this equation. Note that $\nabla \vec{F}^\top(\vec{x})$ is the transposed Jacobian. The vector $\nabla E(\vec{x})$ only contains partial derivatives, and so is easy to evaluate analytically.

Whereas up until now we have only considered relationships at the fixed point, these equations can be generalized to all $\vec{x}$. If equation 2.8 holds at the fixed points, then there must exist some function $\vec{S}(\vec{a}, \vec{b}) : \{\mathbb{R}^n, \mathbb{R}^n\} \to \mathbb{R}^n$ that is zero when $\vec{a} = \vec{b}$, such that the following equation holds for all $\vec{x}$:

$$\vec{T}(\vec{x}) = \vec{S}(\vec{x}, \vec{F}(\vec{x})) + \nabla E(\vec{x}) + \left(\nabla \vec{F}^\top(\vec{x})\right) \cdot \vec{T}(\vec{x})\,. \tag{2.9}$$

We call equation 2.9 the *intrinsic gradient equation*, since it constitutes a necessary and sufficient condition for the gradient of an error function E (defined in terms of the fixed points of $\vec{F}$) to be calculable from the intrinsic network activity at the fixed points x^* of the output functions $\vec{F}(\vec{x})$. Specifically, for any $\vec{S}$, E, $\vec{F}$, and $\vec{T}$ satisfying the intrinsic gradient equation, with $\vec{S}(\vec{a}, \vec{b}) = 0$ when $\vec{a} = \vec{b}$, the gradient of the error function E can be calculated from a fixed point x^* using the function $\vec{T}$, according to equation 2.7. The intrinsic gradient equation defines $\vec{F}(\vec{x})$ for all $\vec{x}$ rather than

just the fixed points, thus allowing networks satisfying the intrinsic gradient equation to be more easily constructed. Nevertheless, the gradient is still only calculated by the training function $\vec{T}$ at the fixed points. We refer to the function $\vec{S}(\vec{a}, \vec{b})$ as a *slack function,* in analogy to slack variables in linear programming; it ensures that equation 2.8 holds at the fixed points, but allows deviation from equality when the system is not at a fixed point. In the simple example above, the intrinsic gradient equation is satisfied with $\vec{S}(\vec{a}, \vec{b}) = 0$ for all $\vec{a}$ and $\vec{b}$ and $T(x) = 4 \cdot (x - c)$, since

$$\begin{aligned} T(x) &= 0 + \frac{\partial E}{\partial x} + \frac{\partial F}{\partial x} \cdot T(x) \\ 4 \cdot (x - c) &= 0 + 2 \cdot (x - c) + \frac{1}{2} \cdot [4 \cdot (x - c)] \, . \end{aligned}$$

The intrinsic gradient equation (2.9) constitutes a nontrivial relationship between $\vec{S}$, E, $\vec{F}$, and $\vec{T}$, rather than simply a definition of $\vec{S}$ in terms of the other functions. Were we to use the intrinsic gradient equation to define the slack function $\vec{S}$ according to

$$\vec{S}(\vec{x}, \vec{F}(\vec{x})) = \vec{T}(\vec{x}) - \nabla E(\vec{x}) - \left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x}) \, ,$$

we would find that the resulting slack function does not always satisfy $\vec{S}(\vec{a}, \vec{b}) = 0$ when $\vec{a} = \vec{b}$ for most choices of E, $\vec{F}$, and $\vec{T}$. The intrinsic gradient equation (2.9) has a form similar to physical laws such as Maxwell's equations, the Schrödinger equation, or the Navier-Stokes equations. These well-known physical laws also define a class of acceptable solutions with certain desirable properties, even though actually constructing the full set of such solutions is extremely difficult. The intrinsic gradient equation is more abstract than these familiar examples from physics, however, in that it implicitly defines a set of nonlinear dynamics (through the output functions $\vec{F}$) which themselves govern the time evolution of the system, whereas the examples from physics directly characterize the allowed trajectories of the various systems. Moreover, the intrinsic gradient equation is nonlinear, like the Navier-Stokes equations but unlike most physical laws, and thus difficult to solve analytically.

2.1.4 The surprising relationship between inputs, parameters, and the error function

The full class of intrinsic gradient networks is broad, and we will be forced to make some assumptions about the slack function $\vec{S}$ and the training function $\vec{T}$ in order to draw strong conclusions about the form of the output functions $\vec{F}$. However, we can derive some important structural properties of intrinsic gradient networks solely from the form of the intrinsic gradient equation (2.9). Specifically, we can show that the inputs to the network take the form of a subset of the parameters $\vec{w}$ of the output functions $\vec{F}$. The particular values of these inputs are determined by the error function E.

The gradient of the error function E can be calculated from the internal signals of the converged

network if and only if the intrinsic gradient equation (2.9) is satisfied. Stated slightly differently, any function E that satisfies the intrinsic gradient equation for some functions $\vec{S}$, $\vec{F}$, and $\vec{T}$ (with $\vec{S}(\vec{a},\vec{b}) = 0$ when $\vec{a} = \vec{b}$) can be thought of as an error function, the gradient of which is calculable via equation 2.7 after the network reaches a fixed point of $\vec{F}$. The intrinsic gradient equation defines a relationship between these functions, but does not specify the direction in which constraints should be propagated to ensure that the relationship is satisfied. As a result, given a choice of slack function $\vec{S}$ and the training function $\vec{T}$, we can view the error function E for which the gradient is calculated as a function of $\vec{F}$ defined by the intrinsic gradient equation (2.9), rather than the other way around. We use the term *calculated error function* to refer to this error function, the gradient of which is actually calculated by the intrinsic gradient network.

In principle, though, the error function should define which outputs (a subset of the network state $\vec{x}^*$ at a fixed point of $\vec{F}$) are good and which are bad, and so must be imposed on the system *a priori* by the user. We use the term *desired error function* to refer to the error function that correctly judges the quality of the outputs. Since the desired error function is defined by the mind of the user, while the calculated error function is determined by the intrinsic gradient equation (2.9), these two error functions could be different in an incorrectly designed network. However, if the gradient calculated by the network according to equation 2.7 is to be used for learning, this calculated gradient must correspond to the desired error function, so the calculated error function must be equal to the desired error function (up to a constant).

Any parameters on which the calculated error function depends directly should not be changed by a learning algorithm intended to minimize the desired error function, since any alteration of these parameters would change the calculated error function minimized by the learning procedure. On the other hand, the desired error function is traditionally a function of both the inputs and the outputs of the network, so it is desirable for the calculated error function (which should be equal to the desired error function for all inputs) to be a function of the inputs. We thus identify parameters on which the calculated error function is directly dependent with the inputs to the network, and refer to them as *input parameters.* Parameters that do not directly affect the calculated error function are called *internal parameters,* and can be altered freely by the learning algorithm without disrupting the error function being minimized. The calculated error function, defined by the intrinsic gradient equation (2.9) as a function of both the input parameters and the network state $\vec{x}$, thus has the same form as an error function defined in the traditional manner on the inputs and the outputs of a system.

While the meaning of the parameters is clarified by considering the error function E as a function of the output functions $\vec{F}$, when constructing an intrinsic gradient network in practice, we generally proceed by inferring the output functions $\vec{F}$ from $\vec{S}$, $\vec{T}$, and a desired error function E. Specifically, given $\vec{S}$ and $\vec{T}$, we generally select a simple parameterized form for the desired error function E.

We then use these functions to derive a parameterized form for the output functions $\vec{F}$, such that the calculated error function maps simply onto the desired error function and the intrinsic gradient equation (2.9) is satisfied. The resulting relationship between the calculated error function and the desired error function both identifies the input parameters, and determines how they should be set to achieve any particular desired error function of the chosen parameterized form.

When an intrinsic gradient network constructed in this manner is used in a practical setting, the input parameters are presumably set directly by the outside environment, and determine the error function whose gradient is calculated by the network. It is thus important to select an input parameterization and corresponding desired error function such that direct manipulation of the input parameters yields a sensible error function. As we shall discuss in section 2.4, autoencoding error functions, which are minimized when the network output matches the input parameters, are particularly compatible with intrinsic gradient networks.

The partitioning of the parameters into input parameters and internal parameters is basically a structural constraint on all solutions to the intrinsic gradient equation (2.9). Stronger, functional conclusions about intrinsic gradient networks can be derived by making specific assumptions about the slack function $\vec{S}$ and the training function $\vec{T}$. However, this will require focusing our attention on a subset of all possible intrinsic gradient networks. We will pursue this less general but more powerful approach in sections 2.2 and 2.3.

2.2 Choosing a slack function

Our primary goal is to construct biologically plausible examples of intrinsic gradient networks. We thus want to find output functions $\vec{F}$ which satisfy the intrinsic gradient equation (2.9), with a given error function E and simple training functions $\vec{T}$. The error function E is part of the problem specification, since it determines the input-output mapping implemented by the trained network, so we must find instances of the other functions such that the intrinsic gradient equation is satisfied given the chosen error function. We approach this difficult objective by selecting a particular slack function $\vec{S}$ and training function $\vec{T}$, and then solving the intrinsic gradient equation for the output functions $\vec{F}$ given the chosen $\vec{S}$, $\vec{T}$, and E.

In section 2.1, we developed necessary and sufficient conditions for the intrinsic calculation of the gradient at a fixed point. The selection of a particular slack function $\vec{S}$ would appear to render our results in sections 2.2 and 2.3 less general. Once we choose a particular slack function, we can only find sufficient conditions for intrinsic gradient networks, given that choice of the slack function.

Moreover, it would seem difficult to select an appropriate slack function *a priori*, since the impact of the slack function $\vec{S}$ on the network dynamics is not especially intuitive. The slack function explicitly represents a degree of freedom of the system, rather than capturing a straightforward

restriction on the system's behavior. Specifically, the slack function formalizes what an intrinsic gradient network does when we are not really looking. Whereas the definition of intrinsic gradient networks only concerns the network activity at a fixed point, the slack function governs the behavior of the network away from the fixed point.

Nevertheless, the slack function has a variety of practical consequences which can be used to differentiate between the possibilities. In the rest of this thesis, we primarily focus our attention on the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$. This is the only slack function consistent with modular intrinsic gradient networks given reasonable assumptions on $\vec{T}$, as we sketch below and prove in appendix A.2. In section 3.1, we show that the set of intrinsic gradient networks based on our chosen slack function includes belief propagation on acyclic factor graphs (with an unconventional error function) and recurrent backpropagation; in sections 3.2 and 3.3 we construct highly recurrent intrinsic gradient networks consistent with our chosen slack function that are similar to belief propagation on loopy factor graphs and hierarchical sigmoidal neural networks.

Given the choice $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, the intrinsic gradient equation (2.9) becomes

$$\begin{aligned} \vec{T}(\vec{x}) &= \vec{T}(\vec{x}) - \vec{T}(\vec{F}(\vec{x})) + \nabla E(\vec{x}) + \left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x}) \\ \vec{T}(\vec{F}(\vec{x})) &= \nabla E(\vec{x}) + \left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x}) \,. \end{aligned} \tag{2.10}$$

As described in detail in appendix A.2, a restricted version of equation 2.10 can be derived by substituting the fixed point identity $\vec{x} = \vec{F}(\vec{x})$ into the left-hand side of equation 2.8, which itself only holds at the fixed points. As a result of this substitution, the constraints imposed by equation 2.8 on small groups of output functions F_i are defined solely in terms of their inputs, given some simple assumptions about the training function. The inputs to such a group of output functions at the fixed point will vary as the input parameters and internal parameters are changed. Equation 2.10, which applies to all points, results from requiring that each such group of output functions must be a viable component of an intrinsic gradient network for all possible inputs. Such a group of output functions constitutes a module, which can be plugged into any modular intrinsic gradient network. If the slack function is chosen to be something other than $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, then such universally applicable modules cannot exist given reasonable restrictions on the training function $\vec{T}$.

Intrinsic gradient networks with the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$ calculate the gradient in an intuitive manner. Since $\nabla \vec{F}^{\top}(\vec{x})$ is the transposed Jacobian, equation 2.10, in conjunction with the dynamics of equation 2.1, implies that running the network in the forwards direction is equivalent to performing backpropagation on a transformed version of the messages.

The slack function $\vec{S}$ also affects the behavior of an intrinsic gradient network in the regions of state space where the fixed point restriction is almost, but not exactly, satisfied. This is of practical concern, since the network's dynamics may only be able to identify points where $\vec{F}(\vec{x}) \approx \vec{x}$. If $\vec{T}(\vec{x})$

has relatively small first derivatives, then the training function yields a good approximation to the gradient at points near the fixed points. That is, if $\vec{F}(\vec{x}^*) = \vec{x}^*$, then $\vec{T}(\vec{x}) \approx \vec{T}(\vec{x}^*)$ when $\vec{x} \approx \vec{x}^*$. However, the network's dynamics may find $\vec{x}$ for which $\vec{F}(\vec{x}) \approx \vec{x}$, but where $\vec{x}$ is far from any true fixed point. In a neural implementation, such "approximate" fixed points may be impossible to distinguish from approximations to true fixed points. Fortunately, appendix A.1.3 shows that so long as $\vec{S}(\vec{a}, \vec{b}) \approx 0$ when $\vec{a} \approx \vec{b}$, intrinsic gradient networks calculate an approximate gradient for such approximate fixed points where $\vec{F}(\vec{x}) \approx \vec{x}$, regardless of the proximity to a true fixed point.

We further examine and consider generalizations of the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$ in appendix A.3. We also find in appendix A.8 that belief propagation on acyclic factor graphs with the negative log likelihood error function corresponds to an intrinsic gradient network with a slack function different than $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$. In the remainder of this thesis, however, we will focus on the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$ because it is consistent with modular intrinsic gradient networks, leads to sensible interpretations of the dynamics of $\vec{x}(t)$, and yields interesting solutions to the intrinsic gradient equation.

2.3 Linear training functions

If the training function $\vec{T}(\vec{x}, \vec{w})$ is linear in $\vec{x}$, then equation 2.10 consists of a system of linear partial differential equations in $\vec{F}$. Linear differential equations are much more tractable than nonlinear differential equations. Moreover, we prove in appendix A.2 that modular intrinsic gradient networks with linear parameterizations must have linear training functions $\vec{T}$. We thus focus our attention on linear training functions. While nonlinear $\vec{T}(\vec{x}, \vec{w})$ are possible, we do not consider them further in this thesis except in the case of belief propagation on acyclic factor graphs, which we discuss in section 3.1.1 and appendix A.8.

Perhaps surprisingly, the restriction to linear training functions $\vec{T}$ does not in principle limit the range of possible output functions. Given an intrinsic gradient network with a nonlinear training function $\vec{T}(\vec{x})$, we can construct a homologous intrinsic gradient network with twice as many units and a linear training function, for which the output functions of half the units are the same as those of the original network. The second half of the units in the homologous network explicitly implement the training function of the original network. The details of this construction are provided in appendix A.4.

We begin our analysis of intrinsic gradient networks with linear training functions by describing the way they give rise to input and internal parameters. We then make a few additional assumptions about the training function, and use them to find analytic solutions to the intrinsic gradient equation (2.9).

2.3.1 The manifestation of input and internal parameters

If the training function $\vec{T}(\vec{x},\vec{w})$ is linear in $\vec{x}$, with $\vec{T}(\vec{x},\vec{w}) = \mathbf{T}_{\vec{w}}\cdot\vec{x}$, and $\mathbf{T}_{\vec{w}}$ is a matrix parameterized by $\vec{w}$, the solutions of equation 2.10 are the sum of a solution to the inhomogeneous equation

$$\mathbf{T}_{\vec{w}} \cdot \vec{F}(\vec{x},\vec{w}) - \left(\nabla \vec{F}^{\top}(\vec{x},\vec{w})\right) \cdot \mathbf{T}_{\vec{w}} \cdot \vec{x} = \nabla E(\vec{x})\,, \tag{2.11}$$

and a linear combination of the solutions to the homogeneous equation

$$\mathbf{T}_{\vec{w}} \cdot \vec{F}(\vec{x},\vec{w}) - \left(\nabla \vec{F}^{\top}(\vec{x},\vec{w})\right) \cdot \mathbf{T}_{\vec{w}} \cdot \vec{x} = 0\,. \tag{2.12}$$

The parameters of the output functions $\vec{F}(\vec{x},\vec{w})$ thus consist of both the parameters of the solution to the inhomogeneous equation (2.11), and the parameters of the solutions to the homogeneous equation (2.12). While these two sets of parameters may overlap, we will generally consider solutions to the intrinsic gradient equation (2.9) for which the parameters of the inhomogeneous solution are disjoint from the parameters of the homogeneous solution.

Changes to the parameters of the homogeneous solution leave the left-hand side of equation 2.12 equal to zero. When a homogeneous solution is combined with an inhomogeneous solution, the right-hand side of equation 2.11 remains unchanged by a variation of the parameters of the homogeneous solution. Thus,

$$\nabla \left(E(\vec{x},\vec{w}_1) - E(\vec{x},\vec{w}_2)\right) = 0$$

if $\vec{w}_1$ and $\vec{w}_2$ differ only in the parameters of the homogeneous solution, and the impact of the outputs on the the error function cannot depend upon the parameters of the homogeneous solution (although a function of the parameters of the homogeneous solution can serve as a regularizer, independent of the network outputs). As a result, the parameters of the solution to the homogeneous equation (2.12) must be internal parameters rather than input parameters. Any solution to the homogeneous equation (2.12) can be used regardless of the value of $\nabla E(\vec{x})$, and the solutions to the homogeneous equation (2.12) constitute a consistent core of the network in the face of varying inputs. In contrast, the parameters of the solution to the inhomogeneous equation (2.11) may be input parameters.

More generally, on the basis of this analysis and appendix A.3, we see that the choice of the slack function $\vec{S}$ induces a particular relationship between the output functions $\vec{F}$, the training functions $\vec{T}$, and the error function E. When the slack function $\vec{S}(\vec{a},\vec{b})$ is linear in its second argument, the input parameters of $\vec{F}$ correspond to the solution to an inhomogeneous equation, whereas the internal parameters correspond to the solution to a homogeneous equation.

2.3.2 Additional assumptions on the training function

To make the intrinsic gradient equation (2.9) even more analytically tractable, we further restrict our attention to training functions $\vec{T}(\vec{x}, \vec{w})$ that are linear, invertible functions of $\vec{x}$, independent of $\vec{w}$, such that $\vec{T}(\vec{x}, \vec{w}) = \mathbf{T} \cdot \vec{x}$. The assumption that $\vec{T}(\vec{x}, \vec{w})$ is independent of $\vec{w}$ is made to simplify notation and exposition. The solutions we find work equally well if $\vec{T}(\vec{x}, \vec{w}) = \mathbf{T}_{\vec{w}} \cdot \vec{x}$. We additionally require that $\left(\mathbf{T}^{-1}\right)^{\top} \cdot \mathbf{T} = \mathbf{D}$, where $\mathbf{D}$ is a diagonal matrix with no elements equal to -1. This restriction will simplify equation 2.13 below and the subsequent analysis. If $\mathbf{T}$ is symmetric and invertible, for example, then this assumption is satisfied and $\mathbf{D} = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

We will often need to refer to the elements on the diagonal of $\mathbf{D}$. Since all other elements of $\mathbf{D}$ are 0, we denote the diagonal elements by D_i, rather than the more cumbersome $D_{i,i}$. Each non-zero element on the diagonal of $\mathbf{D}$ must be the multiplicative inverse of some other element on the diagonal of $\mathbf{D}$. The assumption that $\left(\mathbf{T}^{-1}\right)^{\top} \cdot \mathbf{T} = \mathbf{D}$ implies that $\mathbf{T} = \mathbf{T}^{\top} \cdot \mathbf{D}$, since $\left(\mathbf{T}^{-1}\right)^{\top} = \left(\mathbf{T}^{\top}\right)^{-1}$ for all $\mathbf{T}$. Thus, $\mathbf{T}$ is equal to the transpose of $\mathbf{T}$ with its columns scaled according to $\mathbf{D}$. Element $T_{i,j} = D_i \cdot T_{j,i}$, and complementarily $T_{j,i} = D_j \cdot T_{i,j}$, so $D_i = \frac{1}{D_j}$ except when $T_{i,j} = T_{j,i} = 0$. Similarly, $D_i \neq 0$ for any i. If $D_i = 0$, then $\mathbf{D}$ is not invertible, but the inverse of $\mathbf{D}$ is $\mathbf{T}^{-1} \cdot \mathbf{T}^{\top}$, which exists since $\mathbf{T}$ is invertible by assumption. These constraints on the D_i will induce corresponding constraints on the set of intrinsic gradient networks we are able to construct.

To find analytic solutions to the intrinsic gradient equation (2.9), we begin by combining the intrinsic gradient equation and the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$ (resulting in equation 2.10) with the assumption that the training function $\vec{T}(\vec{x})$ is linear, invertible, and independent of the parameters $\vec{w}$, with $\left(\mathbf{T}^{-1}\right)^{\top} \cdot \mathbf{T}$ diagonal. In section 2.3.4, we observe that $\mathbf{T} \cdot \vec{F}(x)$ must be proportional to a conservative vector field. A conservative vector field is the gradient of a scalar function, and in physics corresponds to a force like gravity that obeys a conservation law. The observation that $\mathbf{T} \cdot \vec{F}(x)$ is proportional to a conservative vector field can be understood intuitively by consideration of polynomial $\vec{F}(\vec{x})$, explored in detail in appendix A.7, and finally allows us to solve the intrinsic gradient equation directly. Since the output functions $\vec{F}(\vec{x})$ we construct based upon these assumptions satisfy the intrinsic gradient equation, their associated $\vec{T}(\vec{x})$ can be used to compute the gradient of the error function at their fixed points using equation 2.7. Finally, in appendix A.5 we construct a Lyapunov function for a subset of the intrinsic gradient networks, proving that they have convergent dynamics.

For easy reference, the assumptions made throughout the rest of this section are as follows:

$$
\begin{aligned}
&(i) \quad \vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b}) \\
&(ii) \quad \vec{T}(\vec{x}, \vec{w}) = \mathbf{T} \cdot \vec{x} \quad \text{where } \mathbf{T} \text{ is invertible} \\
&(iii) \quad \left(\mathbf{T}^{-1}\right)^{\top} \cdot \mathbf{T} = \mathbf{D} \quad \text{where } \mathbf{D} \text{ is a diagonal matrix and } \forall i \; D_i \neq -1 \,.
\end{aligned}
$$

2.3.3 Transformed output functions: $\vec{G}(\vec{x}) = \mathbf{T} \cdot \vec{F}(\vec{x})$

Given the assumption that the training function $\vec{T}$ is linear, invertible, and independent of $\vec{w}$, the intrinsic gradient equation (2.9) in conjunction with the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$ (i.e., equation 2.10) becomes

$$\mathbf{T} \cdot \vec{F}(\vec{x}) = \nabla E(\vec{x}) + \left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \mathbf{T} \cdot \vec{x}\,,$$

where $\mathbf{T}$ is a constant matrix. Defining

$$\vec{G}(\vec{x}) = \mathbf{T} \cdot \vec{F}(\vec{x})\,,$$

we obtain

$$\begin{aligned}
\vec{G}(\vec{x}) &= \nabla E(\vec{x}) + \left(\nabla \left(\vec{G}^{\top}(\vec{x}) \cdot \left(\mathbf{T}^{-1}\right)^{\top}\right)\right) \cdot \mathbf{T} \cdot \vec{x} \\
&= \nabla E(\vec{x}) + \left(\nabla \vec{G}^{\top}(\vec{x})\right) \cdot \left(\mathbf{T}^{-1}\right)^{\top} \cdot \mathbf{T} \cdot \vec{x} \\
&= \nabla E(\vec{x}) + \left(\nabla \vec{G}^{\top}(\vec{x})\right) \cdot \mathbf{D} \cdot \vec{x}\,. \qquad (2.13)
\end{aligned}$$

We can recover $\vec{F}(\vec{x})$ using $\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot \vec{G}(\vec{x})$. We sometimes refer to the vector of functions $\vec{G}(\vec{x})$ as the *transformed output functions*. We find it convenient to formulate the intrinsic gradient equation in terms of $\vec{G}(\vec{x})$ rather than $\vec{F}(\vec{x})$, since it allows us to largely abstract away the influence of the training function $\vec{T}(\vec{x})$.

Analogous to equations 2.11 and 2.12, equation 2.13 is linear in $\vec{G}$ (and thus $\vec{F}$, since we assume that $\vec{T}(\vec{x})$ is linear in $\vec{x}$), so the full solution for $\vec{G}$ is the sum of solutions to the homogeneous equation with $\nabla E(\vec{x}) = 0$, and a solution to the full inhomogeneous equation. As discussed above, we assume that the homogeneous and inhomogeneous solutions are independently parameterized. We interpret the inhomogeneous solution as the inputs to the network, since the error function can depend upon the parameters of the inhomogeneous solution. Complementarily, we interpret the homogeneous solution as the internal core of the network, since all solutions to the homogeneous equation are consistent with any error function. A solution to the homogeneous equation can remain constant in the face of changing inputs and their associated effects on the error function, and may be changed without altering the inputs or error function. In much of the rest of this thesis, we focus on solving the homogeneous intrinsic gradient equation with $\nabla E = 0$. However, we will address the full inhomogeneous intrinsic gradient equation and the input dynamics in section 2.4.

We have thus far made three important assumptions about the slack function $\vec{S}$ and the training function $\vec{T}$. The result of these assumptions, equation 2.13, is almost simple enough to attack directly. In the next section, we will make an observation about the structure of the output functions $\vec{F}$, at which point we will finally be able to find a large set of solutions to the intrinsic gradient

equation (2.9).

2.3.4 Conservative vector field formulation

Equation 2.13 is still too complicated to solve directly. Fortunately, we can show that

$$\vec{G}(\vec{x}) = (\mathbf{D} + \mathbf{I})^{-1} \cdot \nabla g(\vec{x}) \tag{2.14}$$

for some scalar function $g(\vec{x})$.[3] A vector function that is the gradient of a scalar function like $g(\vec{x})$ is traditionally called a conservative vector field. Although $\vec{G}(\vec{x})$ is not technically a conservative vector field due to the presence of the factor $(\mathbf{D} + \mathbf{I})^{-1}$, we refer to equation 2.14 as the *conservative vector field formulation*.

Plugging $\vec{G}(\vec{x}) = (\mathbf{D} + \mathbf{I})^{-1} \cdot \vec{H}(\vec{x})$ into equation 2.13 and noting that $(\mathbf{D} + \mathbf{I})$ is diagonal, so both it and its inverse are symmetric and easy to calculate, we find

$$\begin{aligned}
\vec{G}(\vec{x}) &= \nabla E(\vec{x}) + \left(\nabla \vec{G}^{\top}(\vec{x})\right) \cdot \mathbf{D} \cdot \vec{x} \\
(\mathbf{D} + \mathbf{I})^{-1} \cdot \vec{H}(\vec{x}) &= \nabla E(\vec{x}) + \left[\nabla \left(\vec{H}^{\top}(\vec{x}) \cdot [\mathbf{D} + \mathbf{I}]^{-1}\right)\right] \cdot \mathbf{D} \cdot \vec{x} \\
&= \nabla \left(E(\vec{x}) + \vec{H}^{\top}(\vec{x}) \cdot [\mathbf{D} + \mathbf{I}]^{-1} \cdot \mathbf{D} \cdot \vec{x}\right) - \mathbf{D} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \vec{H}(\vec{x}) \\
(\mathbf{D} + \mathbf{I}) \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \vec{H}(\vec{x}) &= \nabla \left(E(\vec{x}) + \vec{H}^{\top}(\vec{x}) \cdot [\mathbf{D} + \mathbf{I}]^{-1} \cdot \mathbf{D} \cdot \vec{x}\right) \\
\vec{H}(\vec{x}) &= \nabla \left(E(\vec{x}) + \vec{H}^{\top}(\vec{x}) \cdot [\mathbf{D} + \mathbf{I}]^{-1} \cdot \mathbf{D} \cdot \vec{x}\right) .
\end{aligned}$$

We thus see that $\vec{H}(\vec{x}) = \nabla g(x)$ for some scalar function $g(x) = \vec{H}^{\top}(\vec{x}) \cdot [\mathbf{D} + \mathbf{I}]^{-1} \cdot \mathbf{D} \cdot \vec{x}$, and

$$\nabla g(\vec{x}) = \nabla \left(E(\vec{x}) + \left[\nabla^{\top} g(\vec{x})\right] \cdot [\mathbf{D} + \mathbf{I}]^{-1} \cdot \mathbf{D} \cdot \vec{x}\right) .$$

Moreover, using the fundamental theorem of calculus, we can conclude that

$$g(\vec{x}) = c + E(\vec{x}) + \left[\nabla^{\top} g(\vec{x})\right] \cdot [\mathbf{D} + \mathbf{I}]^{-1} \cdot \mathbf{D} \cdot \vec{x}$$

for some constant scalar c. For the homogeneous part of equation 2.13, $E(\vec{x}) = 0$ and

$$g(\vec{x}) = c + \left[\nabla^{\top} g(\vec{x})\right] \cdot [\mathbf{D} + \mathbf{I}]^{-1} \cdot \mathbf{D} \cdot \vec{x} \, . \tag{2.15}$$

By analyzing the conservative vector field formulation, we find in appendix A.5 that a large class of intrinsic gradient networks have Lyapunov functions and are thus provably convergent. We also develop an interpretation of intrinsic gradient network dynamics in terms of probabilistic models.

[3] The matrix $(\mathbf{D} + \mathbf{I})^{-1}$ always exists since $D_i \neq -1$ for all i.

We show in appendix A.6 that

$$g(\vec{x}) = c + \sum_k \left[x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot \prod_{j \neq \psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}} \right) \right] \tag{2.16}$$

satisfies equation 2.15, so the corresponding $\vec{G}(\vec{x})$ defined by the conservative vector field formulation (2.14) satisfies the intrinsic gradient equation (2.9) with $\nabla E = 0$ for any set of indices $\psi(k)$ and any set of differentiable scalar functions $h_j^k(x)$ indexed by both j and k. The corresponding set of solutions for $\vec{F}(\vec{x})$ is

$$\begin{aligned} \vec{F}(\vec{x}) &= \mathbf{T}^{-1} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \nabla g(\vec{x}) \\ &= \mathbf{T}^{-1} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \nabla \left(c + \sum_k \left[x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot \prod_{j \neq \psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}} \right) \right] \right) . \end{aligned} \tag{2.17}$$

If the functions $h_j^k(x)$ are differentiable over a subset of their domain, then the intrinsic gradient equation (2.9) holds over the differentiable portion of the domain. In practice, so long as the non-differentiable subset of the domain is of measure zero, $h_j^k(x)$ will be differentiable at all encountered fixed points, and the gradient of the error function can be calculated based upon the intrinsic signals.

Since the intrinsic gradient equation (2.9) has no explicit dependence on the parameters, the functions $h_j^k(\vec{x})$ can be arbitrarily parameterized, and so should strictly be written as $h_j^k(\vec{x}, \vec{w})$. Similarly, $g(x)$ is also a function of the parameters, but we generally continue to write $g(\vec{x})$ to keep our expressions readable. As a particularly simple example compatible with the modular intrinsic gradient networks discussed in appendix A.2, each summand k of $g(x)$ can have a single multiplicative parameter w_k, yielding

$$\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \nabla \left(c + \sum_k \left[w_k \cdot x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot \prod_{j \neq \psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}} \right) \right] \right) , \tag{2.18}$$

where the resulting functions $h_j^k(x)$ are otherwise unparameterized.

Equations 2.16 and 2.17 are intended to capture solutions to the intrinsic gradient equation, rather than the direct specification of an algorithm. As a result, the order of operations is not strictly specified. For instance, it can be seen that if $h_j^k(x) = x^r$, then $\frac{x_j^{r \cdot \frac{D_j+1}{D_j}}}{x_{\psi(k)}^{r \cdot \frac{D_{\psi(k)}+1}{D_{\psi(k)}}}}$ can be used in place of $\left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}} \right)^r$. This is true even if $\frac{D_i+1}{D_i} \bmod 2 = 0$ for some i, in which case traditional

function composition would eliminate the sign of x_i.

It is also interesting to note that if all the functions $h_j^k(x)$ are polynomial and $\mathbf{T}$ is a pairwise permutation matrix so $D_i = 1$ for all i, then the total degree[4] of each summand of $g(x)$ is equal to two, and the total degree of each summand of every output function F_i is equal to one. That is, the output functions are in some sense linear. This is likely related to the fact that the gradient, which constitutes a linearization of the network, can be calculated from the intrinsic signals of the network.

Equations 2.14 and 2.16 are motivated by the result of the more complicated derivation in appendix A.7. There, we make the more conventional assumption that the functions $G_i(\vec{x})$ are polynomial series. Given this polynomial assumption, we develop relationships between the coefficients and exponents of the polynomials, and observe that all of the resulting solutions to equation 2.13 obey equations 2.14 and 2.16. In this section, we consider the conservative vector field formulation directly because it yields a larger set of solutions than are obtained with polynomial series, yet results in a simpler derivation. We further restrict our attention to the homogeneous part of the equation where $\nabla E = 0$, corresponding to the core of the network independent of the inputs, as described in section 2.3.1. Particular solutions to the full inhomogeneous equation with a chosen error function E can easily be found and added in to the network later.

We can extend these results to a wider range of slack functions S, as we discuss in detail in appendix A.3. Likewise, we have previously assumed that the training function $\vec{T}(\vec{x}, \vec{w})$ is independent of $\vec{w}$, but equation 2.17 applies equally well if $\vec{T}(\vec{x}, \vec{w})$ is dependent on $\vec{w}$; that is, $\vec{T}(\vec{x}, \vec{w}) = \mathbf{T}_{\vec{w}} \cdot \vec{x}$, and $\left(\mathbf{T}_{\vec{w}}^{-1}\right)^{\top} \cdot \mathbf{T}_{\vec{w}} = \mathbf{D}_{\vec{w}}$.

A truly staggering variety of solutions can be generated by choosing the arbitrary differentiable functions $h_j^k(\vec{x})$. For instance, these functions can be sigmoidal, corresponding to the sigmoidal firing rate of neurons in response to increasing sensory stimulation (Albrecht & Hamilton, 1982), or Gaussian, like radial basis functions. Moreover, the generalization of these solutions to larger networks and different connection topologies is straightforward. Given assumptions (i) and (ii), the intrinsic gradient equation (2.9) is linear in the output functions $\vec{F}$, so groups of output functions governing disjoint units in a collection of distinct intrinsic gradient networks can be composed freely to build larger networks with arbitrary connection topologies, including highly recurrent topologies. We will demonstrate in section 3.2 how pairwise permutation training functions $\vec{T}$ can be used to build such composite intrinsic gradient networks. Pairwise permutation training functions are simple and local, since they consist of a single signal. These intrinsic gradient networks thus satisfy both criteria for biologically plausible neural networks identified in section 1: they are highly recurrent, and can be trained using a simple, local function of the intrinsic signals within the network.

[4]The total degree, defined more carefully in appendix A.3, is the sum of the exponents of each factor in a single summand.

2.3.5 Simple examples

As an extremely simple example, consider the case where $h_j^k(\vec{x}) = 0$ for all j, k, and $\vec{x}$. In this case, $\vec{F}(\vec{x}) = 0$, which clearly satisfies the homogeneous part of equation 2.10.

Alternatively, assume that $h_j^k(\vec{x}) = 1$ for all j and k. In this case,

$$\begin{aligned}\vec{F}(\vec{x}) &= \mathbf{T}^{-1} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \left((\mathbf{D} + \mathbf{I}) \cdot \mathbf{D}^{-1} \cdot \vec{x}^{1/D} \right) \\ &= \mathbf{T}^{-1} \cdot \mathbf{D}^{-1} \cdot \vec{x}^{1/D},\end{aligned}$$

where $\vec{x}^{1/D}$ is the vector for which element i is x_i^{1/D_i}. If $\mathbf{T}$ is a pairwise permutation matrix, then $\mathbf{T}^{-1} = \mathbf{T}$ and $\mathbf{D} = \mathbf{I}$, so $\vec{F}(\vec{x}) = \mathbf{T} \cdot \vec{x}$.

If $h_{j'}^k(\vec{x}) = w_{\psi(k)}$ for some j' for each $\psi(k)$, and $h_j^k(\vec{x}) = 1$ for all other indices j, then we obtain the slightly more complicated solution

$$\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot \mathbf{D}^{-1} \cdot \mathbf{W} \cdot \vec{x}^{1/D},$$

where $\mathbf{W}$ is a diagonal matrix for which $W_{k,k} = w_k$. This incredibly simple set of output functions consists of pairs of reciprocally connected linear functions if $\mathbf{T}$ is a pairwise permutation matrix. It is intuitive that the gradient can be calculated in such a network using only the intrinsic activity, since both linearization and reversal of message direction leave the output functions unchanged. As a result, the backpropagation signals are identical to the original messages at the fixed points.

2.3.6 Analysis of the conservative vector field solution

By analyzing the conservative vector field formulation (2.14), it can be seen that the assumptions of section 2.3.2 imply that the training function matrix $\mathbf{T}$ determines the connectivity of the units $\vec{x}$. The term $\nabla g(\vec{x})$ gives rise to a vector of expressions, where entry i is produced by those terms of $g(\vec{x})$ that are a function of x_i. The output functions are then defined by $\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \nabla g(\vec{x})$. The matrix $(\mathbf{D} + \mathbf{I})^{-1}$ is diagonal, and so only scales the entries of $\nabla g(\vec{x})$. In contrast, the matrix $\mathbf{T}^{-1}$ exchanges and recombines the expressions generated by $\nabla g(\vec{x})$. If the entries of $\nabla g(x)$ are thought of as the atomic signals of the intrinsic gradient network, then $\mathbf{T}^{-1}$ specifies where each of these signals projects, and thus the connectivity of the network. In particular, if $\mathbf{T}$ is a pairwise permutation matrix, then all connections must be reciprocal; if $F_a(\vec{x}) \propto \frac{\partial g(\vec{x})}{\partial x_b}$, then $F_b(\vec{x}) \propto \frac{\partial g(\vec{x})}{\partial x_a}$.

Aside from the routing of signals, equation 2.13 almost entirely isolates the output functions $\vec{F}$ from the training function $\vec{T}$. For instance, all symmetric invertible matrices $\mathbf{T}$ yield the same matrix $\mathbf{D} = \left(\mathbf{T}^{-1}\right)^{\top} \cdot \mathbf{T} = \mathbf{I}$, and thus have an identical effect in equation 2.13. Since the training function $\vec{T}$ effectively specifies the routing of the atomic signals generated by $\nabla g(\vec{x})$, this independence is consistent with a set of independently parameterized modules, like those defined in appendix A.2,

which can be interconnected in a variable or programmable manner. This dissociation between $\vec{T}$ and $\vec{F}$ is unlikely to be a general feature of all solutions to the intrinsic gradient equation (2.9).

Although equation 2.17, coupled with assumptions (*i*) through (*iii*), describes a large set of intrinsic gradient networks, the form of equation 2.17 does not specify the overall computation performed by the network in an easily interpretable manner.[5] In contrast, the error function $E(\vec{x}, \vec{w})$ directly encodes the computation we want the network to perform. This error function specifies the desirability of every possible combination of the inputs, which consist of a subset of the parameters $\vec{w}$, and the outputs, which consist of a subset of the units $\vec{x}$ at a fixed point. Before an intrinsic gradient network can be used to perform this desired computation, it must be trained by minimizing the error function E, presumably using the gradient calculable via equation 2.7 from the intrinsic state of the network at a fixed point. This overall situation is directly analogous to a traditional artificial neural network. Feedforward artificial neural networks with at least one hidden layer are universal function approximators, so an artificial neural network with random weights does not intrinsically calculate anything in particular (Bishop, 1995). Rather, a neural network must be trained to minimize an error function, after which it performs an approximation of the computation specified by the error function.

One of the key ideas behind intrinsic gradient networks is that focusing on the fixed points simplifies the analysis. Given our chosen slack function $\vec{S}$ and a linear training function $\vec{T}$, the intrinsic gradient equation (2.9) becomes a system of linear partial differential equations in terms of the output functions $\vec{F}$. This system of linear partial differential equations specifies a system of nonlinear partial differential equations: the dynamics of the units $\vec{x}(t)$ as determined by the output functions $\vec{F}$. Thus, viewed as an analysis of existing network dynamics rather than a derivation of novel network dynamics, the intrinsic gradient equation renders a system of nonlinear partial differential equations into a linear system by restricting analysis to the fixed point. By considering this system of linear differential equations, rather than the original nonlinear system, we can see that the fixed points of the units $\vec{x}$ have a certain desired property: the gradient of the error function is computed by the training function via equation 2.7. Attempting to directly analyze the system of nonlinear partial differential equations that define the dynamics of the units would likely be much more difficult.

2.4 Error functions and input parameters

Since the intrinsic gradient equation (2.9) is linear in the output functions $\vec{F}$ given assumptions (*i*), (*ii*), and (*iii*) of section 2.3.2, it is possible to consider solutions to the full inhomogeneous intrinsic gradient equation separately from solutions to the homogeneous intrinsic gradient equation with

[5] However, if $\mathbf{T}$ is a symmetric positive-definite matrix and $g(\vec{x})$ is bounded above, we can characterize the output states in terms of the local maxima of a Lyapunov function, as in appendix A.5.

$\nabla E(\vec{x}) = 0$, as in section 2.3.1. In sections 3.1, 3.2, and 3.3, we will once again focus primarily on the solutions to the homogeneous equation. In this section, however, we construct output functions $\vec{F}$ consistent with some common error functions E, given assumptions (i), (ii), and (iii). These output functions correspond to the input dynamics of the network. Full intrinsic gradient networks are then formed by additively composing the output functions $\vec{F}$ generated by the error function E (corresponding to the input dynamics) with output functions $\vec{F}$ satisfying the homogeneous intrinsic gradient equation with $\nabla E(\vec{x}) = 0$ (corresponding to the internal dynamics), as we demonstrate in section 3.2.3.

While we will consistently use the term *error function* to refer to $E(\vec{x})$, the intrinsic gradient equation (2.9) merely ensures that the gradient of $E(\vec{x})$ with respect to the parameters can be calculated, without implying that $E(\vec{x})$ must be minimized. Indeed, in a number of the examples we will explore, it will be natural to use the gradient to maximize $E(\vec{x})$. In these cases, $E(\vec{x})$ will function more like an objective function, but we continue to refer to $E(\vec{x})$ as the error function to the avoid the confusion attendant upon using two words for one concept.

2.4.1 Linear error

The error function associated with the simplest possible set of non-zero output functions is $E_{lin}(\vec{x}) = \vec{c}^{\top} \cdot \vec{x}$, where $\vec{c}^{\top}$ is a constant row vector. Using this linear error function, $\nabla E_{lin}(\vec{x}) = \vec{c}$, and $\mathbf{T} \cdot \vec{F}(\vec{x}) = \vec{G}(\vec{x}) = \vec{c}$ is a solution to the full inhomogeneous intrinsic gradient equation, as can be seen from examination of equation 2.13. This solution yields constant output functions $\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot \vec{c}$. Correspondingly, a homogeneous intrinsic gradient network, such as a network satisfying equation 2.17, augmented with constant inputs $\vec{c}_{input}$ so that $\vec{F}(\vec{x}) = \vec{c}_{input}$, calculates the gradient of the error function $E(\vec{x}) = \vec{c}_{input}^{\top} \cdot \mathbf{T}^{\top} \cdot \vec{x}$.

The inputs at any fixed point can be thought of as constant, rather than dependent on the other units, since this transformation does not affect the existence of the fixed point. The resulting fixed inputs can be used to construct a linear error function with the same fixed point and gradient as a more complicated error function, potentially leading to a simpler analysis. Specifically, an intrinsic gradient network with an arbitrary error function $E(\vec{x})$ and fixed point x^* has the same gradient at $\vec{x}^*$ as a homologous intrinsic gradient network with error function $E_{linearized}(\vec{x}) = \vec{c}^{\top} \cdot \vec{x}$, where $\vec{c} = \nabla E(\vec{x})|_{x^*}$. We will use this technique to relate intrinsic gradient networks to stacked auto-associators and deep belief networks in section 3.1.5, and in the implementation of an example intrinsic gradient network in section 3.4.

2.4.2 Sum of squares error

The sum of squares error, defined by $E_{SS}(\vec{x}) = \frac{1}{2} \cdot \sum_i (x_i - c_i)^2$ is a common error function often used with conventional artificial neural networks (Bishop, 1995).[6] Using the sum of squares error, $\nabla E_{SS}(\vec{x}) = \vec{x} - \vec{c}$, so if $\mathbf{D} = \mathbf{I}$ as in the case of a pairwise permutation training function, a solution to the full inhomogeneous intrinsic gradient equation is

$$G_i(\vec{x}) = -c_i - x_i \cdot \log(|x_i|),$$

with $\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot \vec{G}(\vec{x})$, as can be seen from examination of equation 2.13.

An intuitive set of input parameters arise in intrinsic gradient networks designed to calculate the gradient of the negative sum of squares error, the maxima of which are the same as the minima of the sum of squares error. Using $E_{NSS}(\vec{x}) = -\frac{1}{2} \cdot \sum_i (x_i - c_i)^2$ and a linear training function with $\mathbf{D} = \mathbf{I}$, a solution to the full inhomogeneous intrinsic gradient equation is $G_i(\vec{x}) = c_i + x_i \cdot \log(|x_i|)$, with $\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot \vec{G}(\vec{x})$. The parameters c_i are input parameters, with values determined by the error function (now really an objective function). In particular, when using the negative sum of squares error and a pairwise permutation training function, the input consists of a set of constants equal to the desired output values, added to the units complementary (as defined by $\vec{T}$) to the outputs. The negative sum of squares error also requires that the term $x_i \cdot \log(|\vec{x}_i|)$ be added to the units complementary to the outputs, but this term is not affected by the optimal output value and so functions less like an input than like a regularizer.

In section 3.4, we will observe that the term $x_i \cdot \log(|x_i|)$ induced in $G_i(\vec{x})$ by the negative sum of squares error constitutes positive feedback for large x_i. When used in conjunction with otherwise desirable output functions and dynamics, this component of the inhomogeneous solution can cause the network to explode. Ideally, the input parameters would directly reflect the desired outputs, as with $E_{NSS}(\vec{x}) = -\frac{1}{2} \cdot \sum_i (x_i - c_i)^2$, and the regularizing component of the inhomogeneous solution would provide stabilizing negative feedback, as with $E_{SS}(\vec{x}) = \frac{1}{2} \cdot \sum_i (x_i - c_i)^2$. We can achieve this goal by progressively calculating the gradient in two passes, using two complementary intrinsic gradient networks.

Since the gradient is linear, we can split the error function into two parts, calculate the gradient of each part separately, and then add the component gradients back together. Moreover, the linearity of the gradient implies that $\nabla E(\vec{x}) = \frac{1}{c} \cdot \nabla (c \cdot E(\vec{x}))$, so we can calculate $\nabla E(\vec{x})$ by inverse-scaling the gradient of a scaled error function. Combining these two tricks, we can compute the gradient of $E_{NSS}(\vec{x})$ by separately calculating the gradient of $E_{wake}(\vec{x}) = \sum_i c_i \cdot x_i$ and $E_{sleep}(\vec{x}) = \frac{1}{2} \cdot \sum_i x_i^2$, and then computing $\nabla E_{NSS}(\vec{x}) = \nabla E_{wake}(\vec{x}) - \nabla E_{sleep}(\vec{x})$. Of course, since the gradient is calculated with respect to the fixed point, this technique is only applicable if the fixed points associated with

[6] The sum over i usually runs over only a subset of the units x_i.

the two components of the error function are approximately the same (taking into account the results of appendix A.1.3 regarding the gradient of approximate fixed points). We will further develop this approach in section 3.4.2.

2.4.3 Negative log likelihood error

The expected negative log likelihood $E_{NLL}(\vec{x}) = -\sum_i c_i \cdot \log(x_i)$ is another common error function when $0 \leq c_i, x_i \leq 1$ and $\sum_i c_i = \sum_i x_i = 1$, so both $\vec{c}$ and $\vec{x}$ can be interpreted as probability distributions.[7] The expected negative log likelihood differs from the Kullback-Leibler divergence, $D_{KL}(\vec{c}|\vec{x}) = \sum_i c_i \cdot \log\left(\frac{c_i}{x_i}\right)$, by an additive constant. As a result, the gradient of the expected negative log likelihood is identical to that of the Kullback-Leibler divergence. With the expected negative log likelihood error, $\frac{\partial E_{NLL}(\vec{x})}{\partial x_i} = -\frac{c_i}{x_i}$, so if $\mathbf{D} = \mathbf{I}$, the solution to the full inhomogeneous intrinsic gradient equation is

$$G_i(\vec{x}) = -\frac{1}{2} \cdot \frac{c_i}{x_i} \,,$$

with $\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot \vec{G}(\vec{x})$, as can be seen from examination of equation 2.13.

As in the case of the sum of squares error, intrinsic gradient networks that calculate the gradient of the expected log likelihood, $E_{LL}(\vec{x}) = \sum_i c_i \cdot \log(x_i)$, have more intuitive input parameters than those that calculate the gradient of the negative log likelihood. The maxima of the expected log likelihood are the same as the minima of the expected negative log likelihood. When using the expected log likelihood and a pairwise permutation training function, the solution to the full inhomogeneous intrinsic gradient equation is $G_i(\vec{x}) = \frac{1}{2} \cdot \frac{c_i}{x_i}$, so the input parameters are equal to the desired output values. The input itself is proportional to the desired output, and normalized by the actual output. This input is added to the units complementary (as defined by $\vec{T}$) to the outputs.

2.4.4 Generalized error

The linear error, sum of squares error, and expected negative log likelihood error are all instances of the class of functions that consist of a sum of terms, each of which is only a function of a single unit x_i. For such error functions, $\frac{\partial E(\vec{x})}{\partial x_i}$ is only a function of x_i. If $\mathbf{D} = \mathbf{I}$, the solution to the full inhomogeneous intrinsic gradient equation for such error functions is

$$G_i(\vec{x}) = -x_i \cdot \int \left(x_i^{-2} \cdot \frac{\partial E(\vec{x})}{\partial x_i} \cdot dx_i \right) . \tag{2.19}$$

Substituting equation 2.19 into equation 2.13, we find

$$-x_i \cdot \int \left(x_i^{-2} \cdot \frac{\partial E(\vec{x})}{\partial x_i} \cdot dx_i \right) = \frac{\partial E(\vec{x})}{\partial x_i} + \left(-\int \left(x_i^{-2} \cdot \frac{\partial E(\vec{x})}{\partial x_i} \cdot dx_i \right) - x_i \cdot \left(x_i^{-2} \cdot \frac{\partial E(\vec{x})}{\partial x_i} \right) \right) \cdot x_i \,,$$

[7] As in the sum of squares error, the sum over i usually runs over only a subset of the units x_i.

which is easily seen to be true.

The expected negative log likelihood error function can be difficult to interpret, since in most intrinsic gradient networks the units x_i are not necessarily bounded either above or below, let alone subject to the restriction $\sum_i x_i = 1$. Following standard practice in feedforward artificial neural networks, we might try to enforce the restriction $\sum_i x_i = 1$ directly in the error function. For instance, we could use the soft-max function $\text{softmax}_i(\vec{x}) = \frac{e^{x_i}}{\sum_j e^{x_j}}$ to define the error function $E_{NLL-SOFTMAX} = -\sum_i c_i \cdot \log(\text{softmax}_i(\vec{x}))$ (Bishop, 1995). However, each term of this modified negative log likelihood error function is a function of multiple units x_i, so the solution of equation 2.19 is not applicable.

This difficulty can be resolved by instead assuming that each output unit x_i parameterizes an independent Bernoulli random variable y_i, with $p(y_i = 1) = \text{logistic}(x_i)$ and $p(y_i = 0) = 1 - \text{logistic}(x_i)$, where $\text{logistic}(x) = \frac{1}{1+e^{-x}}$. We can then define the error function

$$E_{NLL-LOGISTIC}(\vec{x}) = -\sum_i c_i \cdot \log(\text{logistic}(x_i)) + (1 - c_i) \cdot \log(1 - \text{logistic}(x_i)) \,,$$

corresponding to the negative log likelihood of a set of such independent variables. This error function does consist of a sum of terms, each of which is only a function of a single unit x_i, so equation 2.19 can be applied. A simple calculation shows that $\frac{\partial E_{NLL-LOGISTIC}(\vec{x})}{\partial x_i} = \text{logistic}(x_i) - c_i$, so

$$G_i(\vec{x}) = -x_i \cdot \int \frac{\text{logistic}(x_i) - c_i}{x_i^2} \cdot dx_i \,.$$

Unfortunately, the required integral has no simple analytic solution.

A full vector of output functions consists of the sum of one of these inhomogeneous solutions, reflecting the input dynamics, and a homogeneous solution representing the internal dynamics. We have already seen examples of simple homogeneous solutions in section 2.3.5. We will explore more complex homogeneous solutions, and show how they are combined with an inhomogeneous solution induced by the error function, in sections 3.2 and 3.3.

Chapter 3

Examples

The intrinsic gradient equation (2.9) can be difficult to understand directly. Moreover, the set of solutions we have found for the intrinsic gradient equation, encapsulated in equation 2.17, is extremely diverse. This heterogeneity follows from the parameterization of equation 2.17 by a set of arbitrary differentiable functions $h_j^k(x)$. In this chapter, we explore some particular choices for the functions $h_j^k(x)$. In section 3.1, we find $h_j^k(x)$ that induce network dynamics identical to belief propagation on acyclic factor graphs and recurrent backpropagation, and discuss the relationship between intrinsic gradient networks and other familiar algorithms. In section 3.2, we consider polynomial $h_j^k(x)$ that result in novel, highly recurrent, locally trainable intrinsic gradient networks with dynamics similar to belief propagation on loopy factor graphs. We investigate sigmoidal $h_j^k(x)$, and the associated (novel, highly recurrent, locally trainable) dynamics reminiscent of a hierarchical sigmoidal neural network in section 3.3. Finally, we apply these two novel intrinsic gradient networks to the XOR problem and handwritten digit recognition in section 3.4.

3.1 Relationship to existing algorithms

We shall show that both belief propagation on acyclic (singly connected; tree-structured) factor graphs and recurrent backpropagation satisfy the intrinsic gradient equation (2.9), given appropriately chosen training functions $\vec{T}(\vec{x})$ and the conventional error functions $E(\vec{x})$. Belief propagation on acyclic factor graphs and recurrent backpropagation are thus intrinsic gradient networks. The existence of these familiar instances of intrinsic gradient networks confirms that intrinsic gradient networks can be effective in practice.

The training function we find for belief propagation on acyclic factor graphs with the negative log likelihood error function is nonlinear, unlike the other examples of intrinsic gradient networks we consider. Correspondingly, its slack function is not the conventional $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$ of section 2.2. Moreover, the associated intrinsic gradient network is not compatible with an arbitrary connection topology. The intrinsic gradient equation (2.9) is only satisfied if there are no loops in

the belief propagation network.

The training function we construct for recurrent backpropagation, in contrast, is trivial, with half of the component functions set equal to zero. This artificial aspect of the training function reflects the contrived structure and biologically implausible nature of recurrent backpropagation.

We also identify qualitative similarities between intrinsic gradient networks and Hopfield networks (Hopfield, 1982), Boltzmann machines (Ackley et al., 1985), and deep belief networks (Hinton et al., 2006). Like Hopfield networks, the outputs of intrinsic gradient networks are defined in terms of their fixed points, and the initial activities of the units can serve as inputs. Similar to Boltzmann machines, intrinsic gradient networks can be understood as inducing a probability distribution over their output configurations. The error functions minimized by intrinsic gradient networks are similar to those minimized by deep belief networks. These homologies help build our intuitive understanding of the operation of intrinsic gradient networks.

3.1.1 Belief propagation on an acyclic factor graph

Factor graphs are a parsimonious form of probabilistic model, in which the full probability distribution is factored into many local components (Kschischang et al., 2001). These local components are multiplied together and normalized to construct the full distribution. Factor graphs generally have a set of observed variables, the distribution over which is intended to model some external data set, and a set of hidden variables, which facilitate the construction of the desired distribution over the observed variables. Belief propagation is a simple message-passing algorithm for calculating the marginal probabilities of random variables in such a probabilistic model, so long as the dependencies between the random variables and the local factored components of the distribution do not form any cycles. Factor graphs and belief propagation are defined in full in appendix A.8.

We show in appendix A.8 that when the negative log likelihood of the observed variables is used as the error function, belief propagation on an acyclic factor graph constitutes an intrinsic gradient network satisfying equation 2.8. That is, belief propagation on an acyclic factor graph satisfies the intrinsic gradient equation (2.9), but with a slack function different from $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$. However, belief propagation is a degenerate instance of intrinsic gradient networks, in that the intrinsic gradient equation is only satisfied if the network is not recurrent.

The training function $\vec{T}$ required to realize this result is somewhat complicated. Although the internal belief propagation messages are of the form of equation 2.17 (and thus consistent with the assumptions of section 2.3.2), the ratio between the belief propagation messages and the output of the required training function changes throughout the network, so $\mathbf{T}$ is not a pairwise permutation and $\left(\mathbf{T}^{-1}\right)^{\top} \cdot \mathbf{T} \neq \mathbf{I}$. Additionally, the negative log likelihood error function traditionally associated with belief propagation is not consistent with the combination of the belief propagation inputs and a linear training function. The training function must precisely scale the intrinsic activity,

corresponding to the normalization required to produce a probability distribution in a factor graph, and resulting in a nonlinearity. Interestingly, given an unconventional error function related to the likelihood, belief propagation on an acyclic factor graph satisfies the intrinsic gradient equation (2.9) with linear $\mathbf{T}$ and $\vec{S}(\vec{a},\vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$. The full construction of the requisite training functions and a demonstration that they satisfy the intrinsic gradient equation can be found in appendix A.8.

3.1.2 Recurrent backpropagation

Recurrent backpropagation, including the special case of traditional backpropagation on feedforward networks, satisfies the intrinsic gradient equation (2.9) with the slack function $\vec{S}(\vec{a},\vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$ discussed in section 2.2. In recurrent neural networks trained with recurrent backpropagation, the units are divided into two disjoint but complementary groups: forward-propagation units and backpropagation units. The forward-propagation units form a traditional recurrent neural network, potentially projecting to any other unit, and can serve as outputs that directly affect the value of the error function. The backpropagation units only project to other backpropagation units, and cannot affect the value of the error function directly. The forward-propagation units are recurrently connected, the backpropagation units are recurrently connected, and the forward-propagation units can project to the backpropagation units, but the backpropagation units cannot project to the forward-propagation units; in this sense, recurrent backpropagation is a degenerate instance of intrinsic gradient networks.

Each forward-propagation unit $x_{i_{ff}}$, is associated with a distinct backpropagation unit $x_{i_{fb}}$, with $T_{i_{ff}}(\vec{x}) = x_{i_{fb}}$. That is, $x_{i_{fb}}$ computes that hard part of the gradient for $x_{i_{ff}}$. The symbols ff and fb serve as subscripts on the index i; i_{ff} and i_{fb} are the distinct indices of a corresponding forward-propagation unit and backpropagation unit.

In the traditional formulation of recurrent backpropagation, the output function (often called the activation function in this context) of a forward-propagation unit is a sigmoidal function of a weighted sum of some subset of the forward-propagation units, as in equation 3.1 (Almeida, 1987; Pineda, 1987). The output function of the matching backpropagation unit ($x_{i_{fb}}$ for forward-propagation unit $x_{i_{ff}}$) is a complementarily weighted sum of the backpropagation units associated with the forward-propagation units to which the original forward-propagation unit projects, scaled by the derivative of the sigmoid of those forward-propagation units, as in equation 3.2. The complementarity of the backpropagation weights is such that $w_{i_{ff}j_{ff}} = w_{i_{fb}j_{ff}} = w_{j_{fb}i_{fb}}$ for all i and j, where w_{ab} is the weight of the connection to unit a from unit b. To simplify notation, we write w_{ij} for all of these weights. Since there are no projections from backpropagation units to forward-propagation units, $w_{i_{ff}j_{fb}} = 0$ for all i and j. That is, for each forward-propagation unit $x_{i_{ff}}$ and corresponding

backpropagation unit $x_{i_{fb}}$,

$$F_{i_{ff}}(\vec{x}) = f\left(\sum_j w_{ij} \cdot x_{j_{ff}} + c_i\right) \text{ and} \tag{3.1}$$

$$F_{i_{fb}}(\vec{x}) = \frac{\partial E(\vec{x})}{\partial x_{i_{ff}}} + \sum_j f'\left(\sum_k w_{jk} \cdot x_{k_{ff}} + c_j\right) \cdot w_{ji} \cdot x_{j_{fb}}\,, \tag{3.2}$$

where $f(x)$ is a sigmoid function such as hyperbolic tangent, c_i is constant external input, and the dynamics of $\vec{x}(t)$ converge to a fixed point $\vec{x}^*$ such that $\vec{x}^* = \vec{F}(\vec{x}^*)$. The dynamics of equations 2.1 and 2.2 are commonly used. When $w_{ij} = 0$ for all $i \geq j$, the dynamics of equation 2.1 perform traditional backpropagation on a feedforward network.

We shall now show that equations 3.1 and 3.2 directly satisfy equation 2.10, which is equivalent to the intrinsic gradient equation (2.9) with the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$. Consider the training function defined by

$$\begin{aligned} T_{i_{ff}}(\vec{x}) &= x_{i_{fb}} \\ T_{i_{fb}}(\vec{x}) &= 0 \end{aligned} \tag{3.3}$$

for all i. Using this training function, equation 3.2 can be rewritten as

$$\begin{aligned} T_{i_{ff}}(\vec{F}(\vec{x})) &= \frac{\partial E(\vec{x})}{\partial x_{i_{ff}}} + \sum_j f'\left(\sum_k w_{jk} x_{k_{ff}} + c_j\right) \cdot w_{ji} \cdot T_{j_{ff}}(\vec{x}) \\ &= \frac{\partial E(\vec{x})}{\partial x_{i_{ff}}} + \sum_j \frac{\partial F_{j_{ff}}(\vec{x})}{\partial x_{i_{ff}}} \cdot T_{j_{ff}}(\vec{x}) \\ &= \frac{\partial E(\vec{x})}{\partial x_{i_{ff}}} + \sum_j \left(\frac{\partial F_{j_{ff}}(\vec{x})}{\partial x_{i_{ff}}} \cdot T_{j_{ff}}(\vec{x}) + \frac{\partial F_{j_{fb}}(\vec{x})}{\partial x_{i_{ff}}} \cdot T_{j_{fb}}(\vec{x})\right)\,, \end{aligned} \tag{3.4}$$

where the first line results from applying equation 3.3 to equation 3.2, the second line follows from equation 3.1, and the third line holds because $T_{j_{fb}}(\vec{x}) = 0$ for all j, as in equation 3.3.

We can also show that

$$T_{i_{fb}}(\vec{F}(\vec{x})) = \frac{\partial E(\vec{x})}{\partial x_{i_{fb}}} + \sum_j \left(\frac{\partial F_{j_{ff}}(\vec{x})}{\partial x_{i_{fb}}} \cdot T_{j_{ff}}(\vec{x}) + \frac{\partial F_{j_{fb}}(\vec{x})}{\partial x_{i_{fb}}} \cdot T_{j_{fb}}(\vec{x})\right)\,. \tag{3.5}$$

All terms in equation 3.5 are equal to zero. For the first term and the last term, $T_{i_{fb}}(\vec{x}) = 0$ for all i_{fb} by definition in equation 3.3; for the second term, $\frac{\partial E(\vec{x})}{\partial x_{i_{fb}}} = 0$ for all i_{fb} because the error function is not directly dependent upon the backpropagation units; and for the third term, $\frac{\partial F_{j_{ff}}(\vec{x})}{\partial x_{i_{fb}}} = 0$ for all i_{fb} by equation 3.1. Equations 3.4 and 3.5 together are equivalent to equation 2.10, with the output functions defined by equations 3.1 and 3.2, and the training function defined by equation 3.3.

As a result, recurrent backpropagation satisfies the intrinsic gradient equation (2.9), with the slack function $\vec{S}(\vec{a},\vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$.

3.1.3 Hopfield networks

Hopfield networks consist of a network of discrete units with the dynamics

$$x_i(t+1) = \begin{cases} 1 & \text{if } c_i + \sum_j w_{ij} \cdot x_j(t) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

(Hopfield, 1982), or continuous units with the dynamics

$$\frac{dx_i}{dt} = -\frac{x_i}{\tau} + c_i + \sum_j w_{ij} \cdot g(x_j)\,, \tag{3.7}$$

where $g(x)$ is a sigmoidal function (Cohen & Grossberg, 1983; Hopfield, 1984). The input to a Hopfield network is often encoded in the initial state of the units $\vec{x}$; the network converges from each initial state to a potentially distinct fixed point, which constitutes the output of the network for that input. A particular configuration of the units can be made a fixed point of the dynamics by training the parameters $\vec{w}$ in a Hebbian manner after fixing the activities of the units to the desired configuration.

While output functions consistent with the dynamics of equations 3.6 or 3.7 do not satisfy the intrinsic gradient equation (2.9), intrinsic gradient networks are similar to Hopfield networks in that the outputs of both are defined in terms of the fixed points. Furthermore, as we will see in section 4.3, gradient descent in a large class of intrinsic gradient networks consists of a pseudo-Hebbian update. Whereas Hebbian training in Hopfield networks is heuristic and liable to introduce spurious fixed points, these pseudo-Hebbian updates at the fixed points of appropriate intrinsic gradient networks directly minimize the error function. Furthermore, whereas Hebbian training of Hopfield networks requires that all units be observed, intrinsic gradient networks can accommodate hidden units.

Hopfield networks are often constructed such that their input consists solely of the initial configuration of their units. In contrast, we have shown in sections 2.1.4 and 2.4 that the input to an intrinsic gradient network must include a set of input parameters, which control the output functions $\vec{F}$ and thus the fixed points (and usually the dynamics) of the network. Interestingly, these input parameters can take a form analogous to the term c_i in equations 3.6 and 3.7, as discussed in section 2.4. Moreover, the output of an intrinsic gradient network can also be affected by the initial state. Just as in a Hopfield network, the initial network state can determine which of the possible fixed points is selected, within the set determined by the input parameters. The initial state of the network thus constitutes an input to an intrinsic gradient network.

Input consisting of the initial state of the units is naturally and unavoidably present when an intrinsic gradient network is presented with a time-varying input. At time t, the intrinsic gradient network converges to an attractor based upon its state at time $t - \Delta t$. As a result, an attractor easily reached from that at time $t - \Delta t$ is adjusted to minimize the error function at time t. This mechanism is especially compatible with slowly varying input parameters; if the attractors also change slowly, the network is trained to produce not just single fixed points based upon constant inputs, but trajectories from dynamic inputs. While this phenomenon seems unlikely to train an intrinsic gradient network to reproduce observed trajectories in the absence of appropriate input, it should make the observed trajectories easier to recognize and follow when they are presented.

3.1.4 Boltzmann machines

Boltzmann machines are a stochastic generalization of Hopfield networks (Ackley et al., 1985). They consist of a network of discrete, stochastic units $\vec{x}$ with the dynamics

$$P\left[x_i(t+1) = 1\right] = \frac{1}{1 + e^{-\left(c_i + \sum_j w_{ij} \cdot x_j(t)\right)}}$$
$$P\left[x_i(t+1) = 0\right] = 1 - P\left[x_i(t) = 1\right] ,$$

which constitutes Gibbs sampling on the probability distribution

$$P(\vec{x}) = \frac{e^{-E(\vec{x})}}{\sum_{\vec{x}'} e^{-E(\vec{x}')}}$$

where

$$E(\vec{x}) = -\sum_i c_i \cdot x_i - \sum_{i<j} w_{ij} \cdot x_i \cdot x_j .$$

Boltzmann machines are similar to intrinsic gradient networks in that the gradient of the negative log likelihood, a common error function for probabilistic models, can be calculated based solely on the intrinsic activity (Geman & Geman, 1984; Ackley et al., 1985). However, whereas intrinsic gradient networks have discrete output states defined by a deterministic fixed-point equation, Boltzmann machines use the stochastic dynamics of Gibbs sampling and converge to a distribution over their possible network states. Minimization of the negative log likelihood in a Boltzmann machine maximizes the amount of time that the stochastic Gibbs sampling dynamics spend at a set of desired states after convergence to the equilibrium probability distribution, and calculating the gradient requires the average activity over this equilibrium distribution. Unfortunately, the probability distributions underlying Boltzmann machines are in general analytically intractable, and difficult to sample from efficiently and accurately (Long & Servedio, 2010).

Intrinsic gradient networks can be thought of as a variation on Boltzmann machines for which

the equilibrium distributions are weighted sums of Dirac delta functions.[1] Since the output functions $\vec{F}$ of an intrinsic gradient network only specify the output states ($\vec{x}^*$ such that $\vec{F}(\vec{x}^*) = \vec{x}^*$), the dynamics are only loosely constrained. In particular, the dynamics may be stochastic, as in a Boltzmann machine, thereby inducing a probability distribution over the fixed points. The fixed points of an intrinsic gradient network each contribute one term to the weighted sum of Dirac delta functions that constitutes the probability distribution over the output states. Unlike Boltzmann machines, however, even intrinsic gradient networks with stochastic dynamics have a simple, unambiguous method for determining when the system has converged: the output can be read out and training performed when the system has reached a fixed point of the output functions. In contrast, it is impossible to determine whether a sample from a Boltzmann machine is drawn independently from the underlying distribution simply by looking at the sample.

Even using deterministic dynamics, an intrinsic gradient network may have many fixed points associated with a given error function (corresponding to a particular set of input parameters). The fixed point to which the network converges using deterministic dynamics is determined by the initial configuration of the units ($\vec{x}(t)$ at $t = 0$). As a result, there is still an implicit probability distribution over the fixed points, induced by the probability distribution over the starting configuration of the units.

So long as the error function E can be interpreted as a function of the likelihood of the input given the network state $\vec{x}$, an intrinsic gradient network calculates the gradient of that function of the likelihood for the current fixed point. For instance, the sum of squares error corresponds to the log likelihood if the probability distribution of the inputs given a network configuration $\vec{x}$ is a Gaussian centered around the outputs. However, the gradients naturally calculated by intrinsic gradient networks only take into account the position the current fixed point; they ignore any change in the probability of reaching the current fixed point, or any effect on the other fixed points.

Although simpler in some ways than the probability distributions of Boltzmann machines, the weighted sum of Dirac delta functions has the advantage that, provided that there are relatively few delta spikes (i.e., fixed points), they can all be trained directly. In contrast, in a Boltzmann machine, all configurations generally have non-zero probability and can constitute outputs of the network, so the probability of all configurations must be actively managed. Indeed, a local minimum of the negative log likelihood in a Boltzmann machine may assign substantial probability to undesired configurations, so long as the probabilities of the desired configurations are large. For example, a Boltzmann machine trained to maximize the likelihood of handwritten digits may learn to assign significant probability to a figure that looks like a "3" on the top but an "8" on the bottom. A

[1] As we discuss in appendix A.5, in many intrinsic gradient networks, the fixed points are stationary points of a probability distribution induced by the output functions. In particular, the fixed points correspond to the locally maximal *a posteriori* configurations of this probability distribution. The probability distribution maximized at the fixed points constrains, but is different from, the probability distribution over the fixed points (i.e., the output states) discussed here.

Boltzmann machine would experience pressure to eliminate such an undesired configuration only to the extent that it consumes probability that could otherwise be assigned to the exemplars. If such a spurious figure were a fixed point of an intrinsic gradient network, it would be corrected directly.

3.1.5 Deep belief networks and stacked auto-associators

A variety of effective techniques have recently been developed for training deep networks, such as deep belief networks and stacked autoencoders (Hinton et al., 2006; Bengio et al., 2007; Vincent et al., 2008). The increased efficacy of these techniques compared to conventional stochastic gradient descent is due to a stage of unsupervised pre-training on the inputs alone, prior to supervised training on the inputs and outputs (Larochelle et al., 2009; Erhan et al., 2010). Like these techniques for conventional deep networks, the gradient calculated by intrinsic gradient networks can be used to perform unsupervised training on the inputs. In particular, when using an error function that constitutes a distance or divergence measure between the network's outputs and some vector of ideal values, such as the sum of squares and negative log likelihood error functions described in section 2.4, minimization of the error function induces the network to reconstruct these ideal values on its outputs. As discussed in sections 2.1.4 and 2.4, the error function $E(\vec{x})$, and thus these ideal values, are also reflected in the input parameters. For example, when using the sum of squares error function (and given the assumptions described in section 2.3), the ideal values manifest as constant additive offsets in the output function of the complementary input units. As a result, the minimization of such distance-measure error functions constitutes unsupervised training.

Additionally, hierarchical intrinsic gradient networks with a pairwise permutation training function naturally train their internal layers to reconstruct their inputs, similar to stacked autoencoders (Bengio et al., 2007; Vincent et al., 2008). Consider each layer of a hierarchical intrinsic gradient network (with a pairwise permutation training function) separately. At a fixed point, the inputs to each layer are effectively constant. As discussed in section 2.4.1, an intrinsic gradient network with a pairwise permutation training function and constant inputs computes the gradient of the inner product between the vector of constant inputs and the complementary outputs. Therefore, each layer of such a hierarchical intrinsic gradient network effectively computes the gradient of the inner product between its output to the adjacent layers and its input from the adjacent layers at the fixed point. Ascent of this gradient differs from the training algorithm used by stacked autoencoders primarily in the use of the additive inverse of the inner product, instead of the sum of squares error or negative log likelihood, as the error function.

3.2 Belief-propagating intrinsic gradient networks

Factor graphs are a popular type of probabilistic model (Kschischang et al., 2001), defined in detail in appendix A.8. Belief propagation is a set of network dynamics which can be used to perform exact inference on acyclic (singly connected) factor graphs, and approximate inference on factor graphs with loops (Murphy et al., 1999). Since the marginal probabilities of groups of random variables are sufficient to calculate the gradient of the negative log likelihood in a factor graph, acyclic factor graphs can be trained easily and efficiently using belief propagation. Correspondingly, we have shown in section 3.1.1 and appendix A.8 that belief propagation on an acyclic factor graph is an intrinsic gradient network.

However, belief propagation on an acyclic factor graph is not recurrent, whereas the network of cortical neurons is highly recurrent (Felleman & Van Essen, 1991; Douglas & Martin, 2004). As a result, belief propagation on acyclic factor graphs is not a biologically plausible model of cortical computation. While belief propagation on loopy factor graphs is recurrent, it can produce grossly inaccurate approximations to the marginal probabilities, especially when there are small, strong loops (Yedidia et al., 2005). Belief propagation on a loopy factor graph is therefore not an intrinsic gradient network, and is not biologically plausible because it cannot be used to reliably train the network.

In this section, we construct intrinsic gradient networks with output functions reminiscent of belief propagation on loopy factor graphs, which we therefore call *belief-propagating intrinsic gradient networks*. Like belief propagation on loopy factor graphs, belief-propagating intrinsic gradient networks do not compute exact beliefs according to the probability calculus; rather, they generalize belief propagation on acyclic factor graphs by extending to a diverse set of connection topologies the ability to calculate the exact gradient using the intrinsic signals. In particular, belief-propagating intrinsic gradient networks can support highly recurrent (strongly connected) topologies, and so are more biologically plausible in this regard than belief propagation on acyclic factor graphs.

While these belief-propagating intrinsic gradient networks are a direct application of equation 2.17, the process of building a concrete network from our abstract formulas is unintuitive. To clarify this procedure, we demonstrate the step-by-step construction of a small but nontrivial belief-propagating intrinsic gradient network. This example network is itself recurrent, even though it corresponds to an acyclic factor graph. The same methodology can be used to build intrinsic gradient networks of arbitrary size and topology, including cyclic topologies.

We first construct two extremely small belief-propagating intrinsic gradient networks, depicted in figures 3.1 and 3.2, corresponding to part of a degree-two factor node and part of a degree-three variable node in a factor graph, respectively. Given assumptions (i) and (ii) in section 2.3.2, the intrinsic gradient equation (2.9) is linear in the output functions $\vec{F}$, so groups of output functions that

independently satisfy the homogeneous part of the intrinsic gradient equation (2.12) can be added together to build larger networks. This addition corresponds to the sum over k in equation 2.17. Our two initial belief-propagating intrinsic gradient networks are atomic in the sense that they cannot be broken down into the sum of smaller intrinsic gradient networks. Taking advantage of the linearity of the intrinsic gradient equation, we proceed to additively combine many instances of these atomic belief-propagating intrinsic gradient networks into a network corresponding to a small factor graph consisting of two variable nodes connected to each other by a factor node, as depicted in figure 3.3. Finally, we show how inputs can be introduced into the network, corresponding to a solution to the inhomogeneous part of the intrinsic gradient equation and the associated error function.

In order to build our atomic nodes, we first construct simple polynomial solutions to the intrinsic gradient equation (2.9) by carefully selecting a linear training function $\mathbf{T}$, the indices $\psi(k)$, and the differentiable functions $h_j^k(\vec{x})$ which parameterize equation 2.17. We choose $\mathbf{T}$ to be a pairwise permutation matrix, so $\mathbf{T} = \mathbf{T}^{-1} = \mathbf{T}^\top$ and $\mathbf{D} = \mathbf{I}$. Given this restriction, equation 2.18 becomes

$$\vec{F}(\vec{x}) = \mathbf{T} \cdot \frac{1}{2} \cdot \nabla \left(c + \sum_k \left[w_k \cdot x_{\psi(k)}^2 \cdot \prod_{j \neq \psi(k)} h_j^k \left(\frac{x_j^2}{x_{\psi(k)}^2} \right) \right] \right) . \tag{3.8}$$

We then choose the set of functions $h_j^k(x)$ such that for each k, $h_j^k(x) = x^{\frac{1}{n_k}}$ for $n_k - 1$ of the indices j with $j \neq \psi(k)$, and $h_j^k(x) = 1$ otherwise. We will use $n_k = 2$ in section 3.2.1 and $n_k = 3$ in section 3.2.2, resulting in nodes of degree two and three, respectively. Plugging these choices into equation 3.8, we obtain

$$\vec{F}(\vec{x}) = \mathbf{T} \cdot \frac{1}{2} \cdot \nabla \left(c + \sum_k \left[w_k \cdot \prod_{j \in J_k} x_j^{\frac{2}{n_k}} \right] \right) , \tag{3.9}$$

where J_k is a set of n_k distinct indices of $\vec{x}$, and we take advantage of the ability to compose $h_j^k(x)$ and its argument, even when this implies the inversion of a squaring operation.

3.2.1 Atomic degree-two factor node

We construct a parameterized degree-two node using the units x_a, x_b, x_α, and x_β, depicted in figure 3.1, by restricting k to the set $\{1\}$, and choosing $J_1 = \{a, b\}$ and $w_1 = 2 \cdot w$ in equation 3.9. This corresponds to a scalar function $g(\vec{x})$ in equations 2.16 and 2.17 defined by

$$g(\vec{x}) = 2 \cdot w \cdot x_a \cdot x_b \, . \tag{3.10}$$

Consider a training function $\vec{T}(\vec{x}) = \mathbf{T} \cdot \vec{x}$, where $\mathbf{T}$ is a pairwise permutation matrix that exchanges the pairs of units $x_a \leftrightarrow x_\alpha$ and $x_b \leftrightarrow x_\beta$. For instance, if $\vec{x} = \begin{bmatrix} x_a & x_b & x_\alpha & x_\beta \end{bmatrix}^\top$,

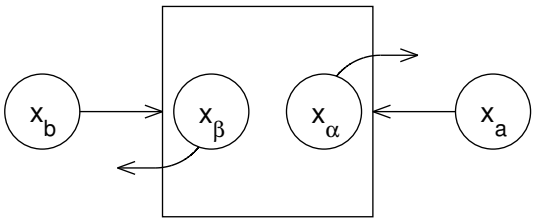


Figure 3.1: An atomic factor node for a belief-propagating intrinsic gradient network. The output functions of the units are specified by equation 3.12. Units x_α and x_β compute outputs analogous to the belief propagation messages out of a degree-two factor node connected to two unary variable nodes. The arrows from units x_a and x_b terminate on the square containing x_α and x_β to indicate that x_a and x_b project to both units in the square.

then

$$\vec{T}(\vec{x}) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_a \\ x_b \\ x_\alpha \\ x_\beta \end{bmatrix} = \begin{bmatrix} x_\alpha \\ x_\beta \\ x_a \\ x_b \end{bmatrix} . \tag{3.11}$$

Plugging $J_1 = \{a, b\}$ (with $k \in \{1\}$) and equation 3.11 into equation 3.9 (or equivalently plugging equations 3.10 and 3.11 into equation 2.17), we obtain

$$\vec{F}(\vec{x}) = \begin{bmatrix} F_a(\vec{x}) \\ F_b(\vec{x}) \\ F_\alpha(\vec{x}) \\ F_\beta(\vec{x}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ w \cdot x_b \\ w \cdot x_a \end{bmatrix} . \tag{3.12}$$

Given these choices, the sum over k disappears because k only assumes a single value. While it may initially seem odd that $F_a(\vec{x})$ and $F_b(\vec{x})$ are set equal to zero, we will see that these zeros constitute placeholders where this atomic belief-propagating intrinsic gradient network can be connected to other belief-propagating intrinsic gradient network nodes, or to inputs. Zero is the identity element for addition, so when the $F_a(\vec{x})$ or $F_b(\vec{x})$ of this atomic network is added to the overlapping output function of another network, the final value of $F_a(\vec{x})$ or $F_b(\vec{x})$ will be equal to that of the second network.

It is easy to confirm that these output functions satisfy the intrinsic gradient equation (2.9) with $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, $\vec{T}(\vec{x})$ defined by equation 3.11, and $E(\vec{x}) = 0$. Substituting these choices into

the intrinsic gradient equation, we find

$$\begin{aligned}
\vec{T}(\vec{x}) &= \vec{S}(\vec{x}, \vec{F}(\vec{x})) + \nabla E(\vec{x}) + \left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x}) \\
\vec{T}(\vec{F}(\vec{x})) &= \left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x}) \\
\begin{bmatrix} F_\alpha(\vec{x}) \\ F_\beta(\vec{x}) \\ F_a(\vec{x}) \\ F_b(\vec{x}) \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 & w \\ 0 & 0 & w & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_\alpha \\ x_\beta \\ x_a \\ x_b \end{bmatrix},
\end{aligned} \tag{3.13}$$

where the second line results from applying the definitions of $\vec{S}(\vec{a}, \vec{b})$ and $E(\vec{x})$, and the third line follows from substituting in the definitions of $\vec{T}(\vec{x})$ and $\vec{F}(\vec{x})$. It is easy to see that equation 3.13 is satisfied by the $\vec{F}(\vec{x})$ of equation 3.12, although it is important to note that the order of the components of $\vec{F}(\vec{x})$ is different. A similar derivation shows that these output functions still satisfy the intrinsic gradient equation when there are many additional units x_i, so long as $F_i(\vec{x}) = 0$ and $T_i(\vec{x})$ is independent of x_a, x_b, x_α, and x_β for all such units.

3.2.2 Atomic degree-three variable node

Following an analogous procedure, we can construct an unparameterized degree-three node, depicted in figure 3.2, by choosing $J_1 = \{a, b, c\}$ and $w_1 = 3$ in equation 3.9, with $k \in \{1\}$. This corresponds to a scalar function $g(\vec{x})$ in equations 2.16 and 2.17 defined by

$$g(\vec{x}) = 3 \cdot x_a^{2/3} \cdot x_b^{2/3} \cdot x_c^{2/3}. \tag{3.14}$$

Consider a training function $\vec{T}(\vec{x}) = \mathbf{T} \cdot \vec{x}$, where $\mathbf{T}$ is a pairwise permutation matrix that exchanges the pairs of units $x_a \leftrightarrow x_\alpha$, $x_b \leftrightarrow x_\beta$, and $x_c \leftrightarrow x_\gamma$. If $\vec{x} = \begin{bmatrix} x_a & x_b & x_c & x_\alpha & x_\beta & x_\gamma \end{bmatrix}^{\top}$, then

$$\vec{T}(\vec{x}) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_a \\ x_b \\ x_c \\ x_\alpha \\ x_\beta \\ x_\gamma \end{bmatrix} = \begin{bmatrix} x_\alpha \\ x_\beta \\ x_\gamma \\ x_a \\ x_b \\ x_c \end{bmatrix}. \tag{3.15}$$

Plugging $J_1 = \{a, b, c\}$ and equation 3.15 into equation 3.9 (or equivalently plugging equations 3.14

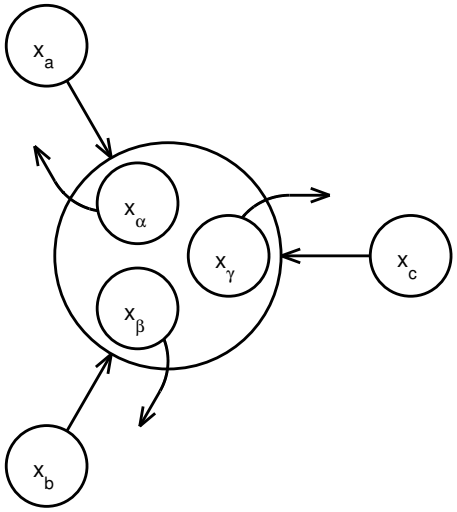


Figure 3.2: An atomic variable node for a belief-propagating intrinsic gradient network. The output functions of the units are specified by equation 3.16. Units x_α, x_β, and x_γ compute outputs analogous to the belief propagation messages associated with a unary degree-three variable node. The arrows from units x_a, x_b, and x_c terminate on the circle containing x_α, x_β, and x_γ to indicate that x_a, x_b, and x_c project to all units in the circle.

and 3.15 into equation 2.17), we obtain

$$\vec{F}(\vec{x}) = \begin{bmatrix} F_a(\vec{x}) \\ F_b(\vec{x}) \\ F_c(\vec{x}) \\ F_\alpha(x) \\ F_\beta(\vec{x}) \\ F_\gamma(\vec{x}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{x_b^{2/3} \cdot x_c^{2/3}}{x_a^{1/3}} \\ \frac{x_a^{2/3} \cdot x_c^{2/3}}{x_b^{1/3}} \\ \frac{x_a^{2/3} \cdot x_b^{2/3}}{x_c^{1/3}} \end{bmatrix} . \tag{3.16}$$

Once again, it is easy to confirm that these output functions satisfy the intrinsic gradient equation (2.9) with $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, $\vec{T}(\vec{x})$ defined by equation 3.15, and $E(\vec{x}) = 0$. Substituting

these choices into the intrinsic gradient equation, we find

$$\begin{aligned}
\vec{T}(\vec{x}) &= \vec{S}(\vec{x}, \vec{F}(\vec{x})) + \nabla E(\vec{x}) + \left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x}) \\
\vec{T}(\vec{F}(\vec{x})) &= \left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x}) \\
\begin{bmatrix} F_\alpha(\vec{x}) \\ F_\beta(\vec{x}) \\ F_\gamma(\vec{x}) \\ F_a(\vec{x}) \\ F_b(\vec{x}) \\ F_c(\vec{x}) \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 & -\frac{1}{3} \cdot \frac{x_b^{2/3} \cdot x_c^{2/3}}{x_a^{4/3}} & \frac{2}{3} \cdot \frac{x_c^{2/3}}{x_a^{1/3} \cdot x_b^{1/3}} & \frac{2}{3} \cdot \frac{x_b^{2/3}}{x_a^{1/3} \cdot x_c^{1/3}} \\ 0 & 0 & 0 & \frac{2}{3} \cdot \frac{x_c^{2/3}}{x_a^{1/3} \cdot x_b^{1/3}} & -\frac{1}{3} \cdot \frac{x_a^{2/3} \cdot x_c^{2/3}}{x_b^{4/3}} & \frac{2}{3} \cdot \frac{x_a^{2/3}}{x_b^{1/3} \cdot x_c^{1/3}} \\ 0 & 0 & 0 & \frac{2}{3} \cdot \frac{x_b^{2/3}}{x_a^{1/3} \cdot x_c^{1/3}} & \frac{2}{3} \cdot \frac{x_a^{2/3}}{x_b^{1/3} \cdot x_c^{1/3}} & -\frac{1}{3} \cdot \frac{x_a^{2/3} \cdot x_b^{2/3}}{x_c^{4/3}} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_\alpha \\ x_\beta \\ x_\gamma \\ x_a \\ x_b \\ x_c \end{bmatrix},
\end{aligned} \tag{3.17}$$

where the second line results from applying the definitions of $\vec{S}(\vec{a}, \vec{b})$ and $E(\vec{x})$, and the third line follows from substituting in the definitions of $\vec{T}(\vec{x})$ and $\vec{F}(\vec{x})$. It is easy to see that equation 3.17 is satisfied by the $\vec{F}(\vec{x})$ of equation 3.16, although it is important to note that the order of the components of $\vec{F}(\vec{x})$ is different. As in the case of the degree-two node of section 3.2.1, these output functions still satisfy the intrinsic gradient equation when there are many additional units x_i, so long as $F_i(\vec{x}) = 0$ and $T_i(\vec{x})$ is independent of x_a, x_b, x_c, x_α, x_β, and x_γ for all such units x_i.

3.2.3 Assembling a larger network

Since the intrinsic gradient equation (2.9) is linear in $\vec{F}(\vec{x})$ when assumptions (i) and (ii) in section 2.3.2 are satisfied, the output functions corresponding to many such atomic belief-propagating intrinsic gradient networks can be added together to form larger belief-propagating intrinsic gradient networks. Such a summation of distinct networks is equivalent to the summation over k in equation 3.9 (and in equation 2.17, from which equation 3.9 was derived). To form the larger network depicted in figure 3.3, we add together four instances of the atomic degree-two factor node constructed in section 3.2.1, and four instances of the atomic degree-three variable node constructed in section 3.2.2. The four instances of the atomic degree-two factor node are defined on overlapping units, and together compose the equivalent of a single factor node in the corresponding factor graph. Each atomic factor node contributes a single parameter to the collective factor node. Complementarily, each group of two instances of the atomic degree-three variable node gives rise to the equivalent of a single binary variable node.

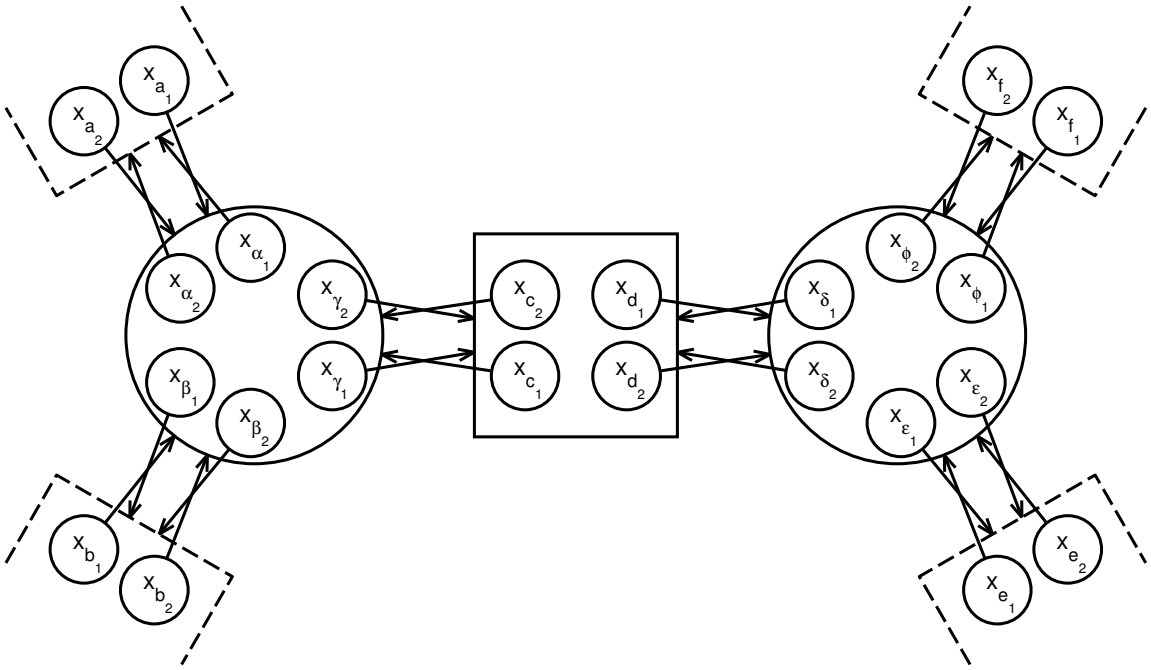


Figure 3.3: A belief-propagating intrinsic gradient network. The output functions of the units are specified by equation 3.19. Units x_{α_i}, x_{β_i}, and x_{γ_i} ($i \in \{1, 2\}$) compute outputs analogous to the belief propagation messages out of a degree-three binary variable node; units x_{δ_i}, x_{ϵ_i}, and x_{ϕ_i} ($i \in \{1, 2\}$) compute outputs analogous to the belief propagation messages out of a second degree-three binary variable node; units x_{c_i} and x_{d_i} ($i \in \{1, 2\}$) compute outputs analogous to the belief propagation messages out of a degree-two factor node connecting two binary variable nodes. Dashed half-boxes enclose units that have output functions implicitly set equal to zero in equation 3.19, but which could be inputs or part of additional factor nodes if the intrinsic gradient network were suitably extended. An arrow from a unit terminating on a solid square or circle enclosing a group of units indicates that the unit at the source of the arrow potentially projects to all units in the enclosed group.

Phrased directly in terms of equation 3.9, we choose

$$
\begin{aligned}
J_1 &= \{a_1, b_1, c_1\} \\
J_2 &= \{a_2, b_2, c_2\} \\
J_3 &= \{\gamma_1, \delta_1\} \\
J_4 &= \{\gamma_1, \delta_2\} \\
J_5 &= \{\gamma_2, \delta_1\} \\
J_6 &= \{\gamma_2, \delta_2\} \\
J_7 &= \{d_1, e_1, f_1\} \\
J_8 &= \{d_2, e_2, f_2\} \ .
\end{aligned}
$$

This corresponds to a scalar function $g(\vec{x})$ in equations 2.16 and 2.17 defined by the sum over the following terms:

$$
\begin{aligned}
(i) \quad & \begin{cases} 3 \cdot x_{a_1}^{2/3} \cdot x_{b_1}^{2/3} \cdot x_{c_1}^{2/3} \\ 3 \cdot x_{a_2}^{2/3} \cdot x_{b_2}^{2/3} \cdot x_{c_2}^{2/3} \end{cases} \\
(ii) \quad & \begin{cases} 2 \cdot w_1 \cdot x_{\gamma_1} \cdot x_{\delta_1} \\ 2 \cdot w_2 \cdot x_{\gamma_1} \cdot x_{\delta_2} \\ 2 \cdot w_3 \cdot x_{\gamma_2} \cdot x_{\delta_1} \\ 2 \cdot w_4 \cdot x_{\gamma_2} \cdot x_{\delta_2} \end{cases} \\
(iii) \quad & \begin{cases} 3 \cdot x_{d_1}^{2/3} \cdot x_{e_1}^{2/3} \cdot x_{f_1}^{2/3} \\ 3 \cdot x_{d_2}^{2/3} \cdot x_{e_2}^{2/3} \cdot x_{f_2}^{2/3} \ . \end{cases}
\end{aligned}
\tag{3.18}
$$

The terms in groups (i) and (iii) of equation 3.18 correspond to four atomic degree-three variable nodes. They do not have any trainable parameters, corresponding to unparameterized functions $h_j^k(x)$ in equation 2.16. The terms in group (ii) of equation 3.18 correspond to four atomic degree-two factor nodes. These eight atomic nodes, each of which has one output in each direction, will combine to yield binary factor and variable nodes with two outputs in each direction (i.e., messages of length two).

Each group of terms (and corresponding group of atomic nodes) marked by the curly braces in equation 3.18 is defined over a disjoint set of units. As a result, when the gradient operator is applied to the sum of the terms in equation 3.18, each curly-brace-delimited group of terms gives rise to non-zero contributions in a disjoint subset of the entries of the gradient vector. Given a training function that maps between subsets of the groups in a pairwise manner, this segregation of the units into disjoint sets can be maintained in the mapping to the output functions on the left-hand side of equation 2.17. In the case of equation 3.18, the arguments of each curly-brace-delimited group

of terms will constitute the input to a single composite variable or factor node in the network of figure 3.3, and the disjoint set of output functions to which they are mapped by the training function will correspond to the outputs from that composite node.

More specifically, consider a training function $\vec{T}(\vec{x}) = \mathbf{T} \cdot \vec{x}$, where $\mathbf{T}$ is a pairwise permutation matrix that exchanges the pairs of units $x_{a_i} \leftrightarrow x_{\alpha_i}$, $x_{b_i} \leftrightarrow x_{\beta_i}$, $x_{c_i} \leftrightarrow x_{\gamma_i}$, $x_{d_i} \leftrightarrow x_{\delta_i}$, $x_{e_i} \leftrightarrow x_{\epsilon_i}$, and $x_{f_i} \leftrightarrow x_{\phi_i}$ for $i \in \{1, 2\}$. This $\mathbf{T}$ never exchanges units that are arguments of the same curly-brace-delimited group of terms in equation 3.18. When the arguments of the curly-brace-delimited groups of terms are disjoint and $\mathbf{T}$ is a pairwise permutation with this property, the set of units that $\mathbf{T}$ exchanges with the arguments of such a group of terms of $g(\vec{x})$ can be thought of as a *node*. For example, $\mathbf{T}$ maps $x_{c_i} \leftrightarrow x_{\gamma_i}$ and $x_{d_i} \leftrightarrow x_{\delta_i}$, so group (ii) induces a node consisting of the units x_{c_1}, x_{c_2}, x_{d_1}, and x_{d_2}. The inputs to this node are the original arguments of group (ii): x_{γ_i} and x_{δ_i}. A node takes inputs from adjacent nodes and generates outputs which are then passed back to these adjacent nodes, as in figure 3.3.

With $g(\vec{x})$ defined by equation 3.18 and using the dynamics of equation 2.1, we will see that the outputs from each node are analogous to the messages from a factor node or variable node in a factor graph, hence the choice of nomenclature. However, we will use the term *node* to refer to such disjoint subsets of units whenever $\mathbf{T}$ is a pairwise permutation matrix that exchanges the between but not within the subsets, even if the dynamics are unlike those of belief propagation. In appendix A.2, we develop a closely related definition of modularity, and show that intrinsic gradient networks with this modular structure must have linear training functions and use the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, given a few reasonable assumptions. Most of the examples of nodes we present in this thesis are also modules, as defined in appendix A.2.

This training function $\mathbf{T}$, in conjunction with equations 2.17 and 3.18, induces the output functions

$$
\begin{aligned}
(i) \quad & \begin{cases} F_{\alpha_i}(\vec{x}) = (x_{a_i})^{-1/3} \cdot (x_{b_i})^{2/3} \cdot (x_{c_i})^{2/3} \\ F_{\beta_i}(\vec{x}) = (x_{a_i})^{2/3} \cdot (x_{b_i})^{-1/3} \cdot (x_{c_i})^{2/3} \\ F_{\gamma_i}(\vec{x}) = (x_{a_i})^{2/3} \cdot (x_{b_i})^{2/3} \cdot (x_{c_i})^{-1/3} \end{cases} \\
(ii) \quad & \begin{cases} F_{c_1}(\vec{x}) = w_1 \cdot x_{\delta_1} + w_2 \cdot x_{\delta_2} \\ F_{c_2}(\vec{x}) = w_3 \cdot x_{\delta_1} + w_4 \cdot x_{\delta_2} \\ F_{d_1}(\vec{x}) = w_1 \cdot x_{\gamma_1} + w_3 \cdot x_{\gamma_2} \\ F_{d_2}(\vec{x}) = w_2 \cdot x_{\gamma_1} + w_4 \cdot x_{\gamma_2} \end{cases} \\
(iii) \quad & \begin{cases} F_{\delta_i}(\vec{x}) = (x_{d_i})^{-1/3} \cdot (x_{e_i})^{2/3} \cdot (x_{f_i})^{2/3} \\ F_{\epsilon_i}(\vec{x}) = (x_{d_i})^{2/3} \cdot (x_{e_i})^{-1/3} \cdot (x_{f_i})^{2/3} \\ F_{\phi_i}(\vec{x}) = (x_{d_i})^{2/3} \cdot (x_{e_i})^{2/3} \cdot (x_{f_i})^{-1/3} \end{cases}
\end{aligned}
\tag{3.19}
$$

for $i \in \{1, 2\}$, as depicted in figure 3.3. Using the full discrete update $\vec{x}(t+1) = \vec{F}(\vec{x}(t))$ of

equation 2.1, the equations of group (ii) of equation 3.19 exactly compute the belief propagation messages through a pairwise factor between two binary variables. We shall thus refer to a set of units with output functions like those of group (ii) as a *factor node*. The large square in figure 3.3 represents the factor node generated by group (ii). In contrast, the equations of groups (i) and (iii) of equation 3.19 compute something analogous to the belief propagation messages out of two binary variable nodes, but raised to the power 2/3 and normalized by the reciprocal incoming messages. We thus refer to a set of units with output functions like those of group (i) or (iii) as a *variable node*. The two large circles in figure 3.3 represent the variable nodes generated by groups (i) and (iii). Each belief propagation message component is associated with a distinct element of $\vec{x}$, so a binary variable connected to three factors in a factor graph corresponds to six units x_i and their associated output functions $F_i(\vec{x})$.

In later sections, we will extend our usage of the term *variable node* to refer to any node consisting of a group of units with unparameterized output functions that project in more than two directions, even if the output functions are unlike those of groups (i) and (iii). Variable nodes are analogous to the visible and hidden layers in a traditional feedforward neural network, and we will sometimes use *visible layer* and *hidden layer* to refer to variable nodes that do and do not have input parameters, respectively. Units that project from a visible layer carry input messages; units that project to a visible layer carry output messages.

As can be seen from equations 2.17 and 3.9, the training function $\mathbf{T}$ determines the connectivity of the resulting network, specifying to which output function $F_i(\vec{x})$ the message formed by $\frac{\partial}{\partial x_i} g(\vec{x})$ projects. Complementarily, viewed in terms of the inputs rather than outputs, $\mathbf{T}$ specifies which units influence output function $F_i(\vec{x})$. Using the dynamics $\vec{x}(t+1) = \vec{F}(\vec{x}(t))$ of equation 2.1, the output function $F_i(\vec{x})$ specifies the output from unit x_i, so $\mathbf{T}$ directly controls how the gradient messages are routed through the units. For example, in equations 3.18 and 3.19 and figure 3.3, $\mathbf{T}$ exchanges $x_{c_i} \leftrightarrow x_{\gamma_i}$ and $x_{d_i} \leftrightarrow x_{\delta_i}$. Block (ii) of $g(\vec{x})$ is defined in terms of x_{γ_i} and x_{δ_i}, so units x_{γ_i} and x_{δ_i} are the inputs to the factor node corresponding to block (ii). However, the action of $\mathbf{T}$ implies that block (ii) actually determines the dynamics of units x_{c_i} and x_{d_i}, which take input from units x_{γ_i} and x_{δ_i} in the adjacent variable nodes, and project back to their associated variable nodes.

Additional nodes can be added to the network by including additional terms in $g(\vec{x})$. In particular, additional terms of the form of groups (i) and (iii) add an additional degree-three variable node to the network; additional terms of the form of group (ii) add an additional pairwise factor node. As described in section 2.4, input signals can also be added to these dynamics by choosing a

non-zero error function $E(\vec{x}, \vec{w})$. For instance the error function[2]

$$E(\vec{x}) = \sum_{i \in \{1,2\}} c_{a_i} \cdot x_{\alpha_i} + c_{b_i} \cdot x_{\beta_i} + c_{e_i} \cdot x_{\epsilon_i} + c_{f_i} \cdot x_{\phi_i}$$

induces the inputs

$$F_{a_i}(\vec{x}) = c_{a_i}$$
$$F_{b_i}(\vec{x}) = c_{b_i}$$
$$F_{e_i}(\vec{x}) = c_{e_i}$$
$$F_{f_i}(\vec{x}) = c_{f_i}$$

for $i \in \{1, 2\}$.

Higher-degree (or lower-degree) variable and factor nodes (i.e., with more or fewer connection directions) can be constructed by increasing (or decreasing) the size of n_k in equation 3.9. Both variable and factor nodes of degree two have output functions identical to those in belief propagation, but differ from belief propagation for higher-degree nodes. Variable and factor nodes with more values per variable (rather than the binary variables of equations 3.18 and 3.19 and figure 3.3) can be constructed by increasing the number of terms in the corresponding blocks of $g(\vec{x})$. In all cases, the number of units x_i must be increased accordingly. Unlike traditional belief propagation, the topology of the connections in these intrinsic gradient networks is unrestricted, so networks with the form of equations 3.18 and 3.19 can be expanded to arbitrary sizes and topologies, including highly recurrent configurations with many loops.

3.3 Hierarchical logistic intrinsic gradient networks

Equation 2.17 is also consistent with output functions more reminiscent of a hierarchical sigmoidal artificial neural network. In a traditional one-hidden-layer neural network, information flows in a single feedforward direction, from the input units, to the hidden units, and then to the output units. While intrinsic gradient networks are generally recurrent rather than feedforward, they can still have a hierarchical connection topology similar to that in a feedforward neural network, with each layer of units connecting only to one superordinate and one subordinate layer. Intrinsic gradient networks can also support output functions based upon sigmoids as in traditional neural networks, rather than the belief-propagation-like output functions discussed in section 3.2.

In particular, we can construct hierarchical intrinsic gradient networks using degree-three variable

[2]We use the term *error function* in accordance with our previous nomenclature, but the intrinsic gradient equation (2.9) merely ensures that the gradient of the error function can be calculated, without implying that the error function should be either minimized or maximized.

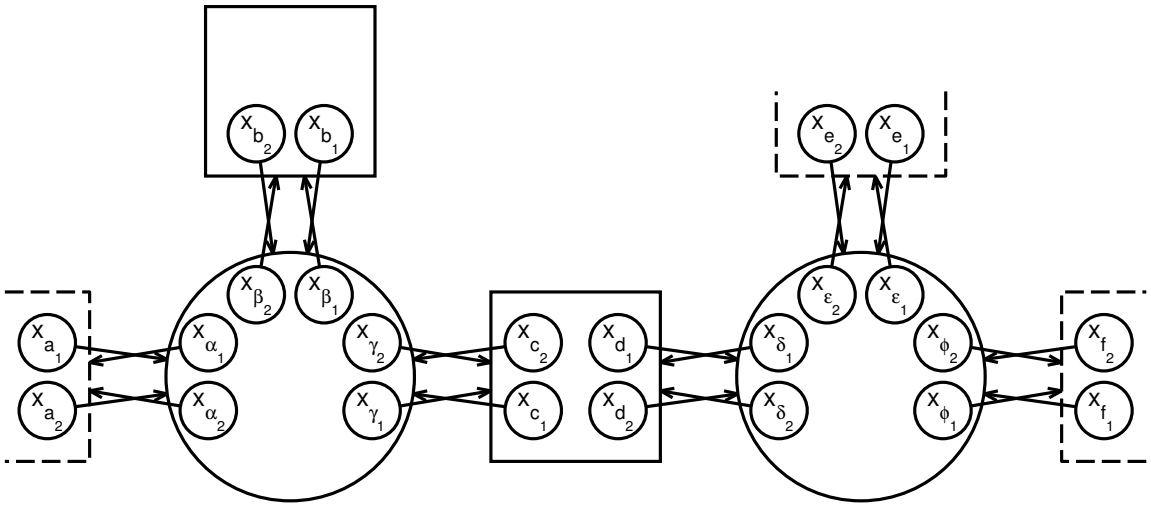


Figure 3.4: Part of a hierarchical logistic intrinsic gradient network. The hierarchy runs from left to right. The output functions of the units are specified by equation 3.21. Units x_{α_i}, x_{γ_i}, x_{δ_i}, and x_{ϕ_i} ($i \in \{1, 2, \cdots, n\}$) have output functions analogous to the sigmoidal transfer function in a traditional neural network. Units x_{c_i} and x_{d_i} ($i \in \{1, 2, \cdots, n\}$) compute the weighted sums on which the sigmoidal transfer function operates in a traditional neural network. Units x_{β_i}, x_{b_i}, and x_{ϵ_i} ($i \in \{1, 2, \cdots, n\}$) implement recurrent connections within a single layer. Dashed half-boxes enclose units that have output functions implicitly set equal to zero in equation 3.21, but which could be inputs or part of additional factor nodes if the intrinsic gradient network were suitably extended. An arrow from a unit terminating on a solid square or circle enclosing a group of units indicates that the unit at the source of the arrow potentially projects to all units in the enclosed group.

nodes, with one variable node giving rise to each layer. As described in section 3.2, degree-three variable nodes have projections in three directions, whereas each layer in a hierarchical network must project in only two directions. To construct a hierarchical network with one degree-three variable node per layer, a factor node (similar to group (*ii*) in equations 3.18 and 3.19), can be used to connect a variable node (such as groups (*i*) and (*iii*) in equations 3.18 and 3.19) to itself, as shown in the top-left of figure 3.4. In this manner, factor nodes can be used to cap all but two connections from each variable node.[3] Additional pairwise factor nodes can then be used to form a linear chain of variable nodes and factor nodes, as shown along the bottom of figure 3.4. Rather than use many small parallel pathways between small variable nodes, the number of units in each factor and variable node can be expanded to correspond to a full layer in a traditional hierarchical neural network, as in figure 3.5. Moreover, the functions $h_j^k(x)$ in equation 2.16 can be logistic, like in a traditional artificial neural network, rather than polynomial. We call the resulting network a *hierarchical logistic intrinsic gradient network*.

[3]More than one connection must be capped to construct a hierarchical network with one variable node per layer if we use variable nodes with degree greater than three. However, it is not necessary to cap all but two connections to construct a hierarchical network in general. For instance, a hierarchical network with a slightly different topology could be constructed by allowing each layer of the hierarchy to have multiple, interconnected variable nodes.

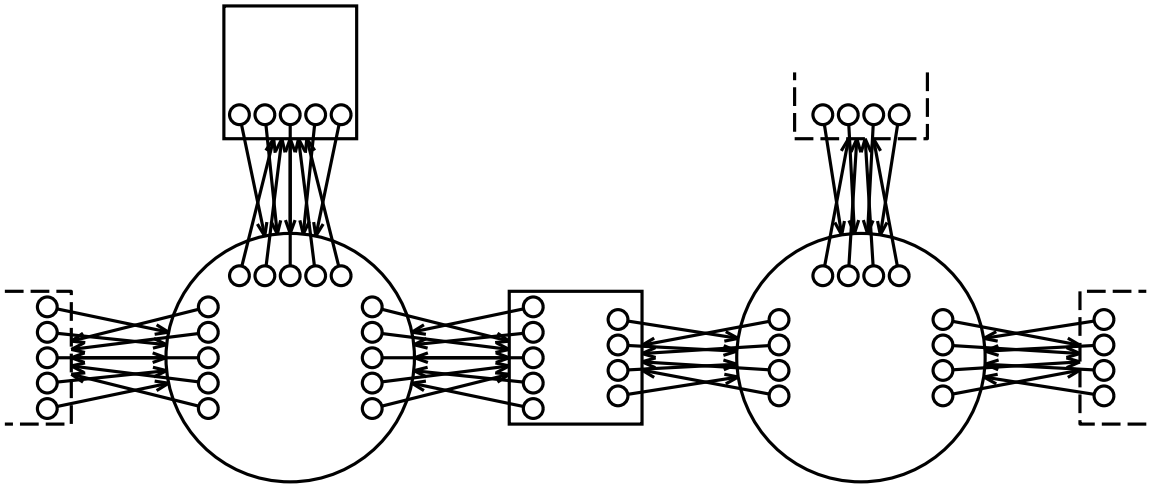


Figure 3.5: A denser hierarchical intrinsic gradient network. The hierarchy runs from left to right. The output functions of the units are specified by equation 3.21, using a labelling following the same pattern as that in figure 3.4. While this figure shows five units projecting in each direction to and from the first layer, and four units projecting in each direction to and from the second layer, each layer may be of any size. Dashed half-boxes enclose units that have output functions implicitly set equal to zero in equation 3.21, but which could be inputs or part of additional factor nodes if the intrinsic gradient network were suitably extended. An arrow from a unit terminating on a solid square or circle enclosing a group of units indicates that the unit at the source of the arrow potentially projects to all units in the enclosed group.

Consider the case where $g(\vec{x})$ is the sum of the terms

$$\begin{aligned}
&2 \cdot \sum_i \sigma\left(\frac{x_{a_i}}{x_{b_i}}\right) \cdot x_{b_i}^2 \cdot \sigma\left(\frac{x_{c_i}}{x_{b_i}}\right) \\
&2 \cdot \sum_{i,j \mid i \neq j} w_{\beta_{ij}} \cdot x_{\beta_i} \cdot x_{\beta_j} \\
&2 \cdot \sum_{i,j \mid i \neq j} w_{\gamma_{ij}} \cdot x_{\gamma_i} \cdot x_{\delta_j} \\
&2 \cdot \sum_i \sigma\left(\frac{x_{d_i}}{x_{e_i}}\right) \cdot x_{e_i}^2 \cdot \sigma\left(\frac{x_{f_i}}{x_{e_i}}\right),
\end{aligned} \tag{3.20}$$

where $i, j \in \{1, 2, \cdots, n\}$, n is the size of a layer in the analogous hierarchical neural network, and $\sigma(x)$ is the logistic function $\sigma(x) = (1 + e^{-x})^{-1}$. Equation 3.20 is consistent with equation 2.17, with a pairwise permutation training function $\vec{T}$ (so $D_i = 1$ for all i), $\psi(1) = b_1$, $h^1_{a_1}(x) = \sigma(x^{1/2})$, $h^1_{c_1}(x) = \sigma(x^{1/2})$, $\cdots$, $\psi(n+1) = \beta_1$, $h^{n+1}_{\beta_2}(x) = w_{\beta_{12}} \cdot x^{1/2}$, and so on. The subscripted indices in equation 3.20 allow us to easily generalize to hierarchical networks with an arbitrary number of units in each layer, as suggested by figure 3.5. Moreover, the set of terms in equation 3.20 can be extended to yield deeper networks by adding an additional set of terms corresponding to the second through the last lines of equation 3.20 for each additional layer. Inputs can be added to the network

by choosing an appropriate error function $E(\vec{x})$, as described in section 2.4.

If $\mathbf{T}$ exchanges the pairs of units $x_{a_i} \leftrightarrow x_{\alpha_i}$, $x_{b_i} \leftrightarrow x_{\beta_i}$, $x_{c_i} \leftrightarrow x_{\gamma_i}$, $x_{d_i} \leftrightarrow x_{\delta_i}$, $x_{e_i} \leftrightarrow x_{\epsilon_i}$, and $x_{f_i} \leftrightarrow x_{\phi_i}$ for all i, then

$$
\begin{array}{ll}
(i) & \left\{ \begin{array}{l}
F_{\alpha_i}(\vec{x}) = \sigma\left(\dfrac{x_{a_i}}{x_{b_i}}\right) \cdot \left[1 - \sigma\left(\dfrac{x_{a_i}}{x_{b_i}}\right)\right] \cdot x_{b_i} \cdot \sigma\left(\dfrac{x_{c_i}}{x_{b_i}}\right) \\
F_{\beta_i}(\vec{x}) = \sigma\left(\dfrac{x_{a_i}}{x_{b_i}}\right) \cdot \sigma\left(\dfrac{x_{c_i}}{x_{b_i}}\right) \cdot \\
\qquad\qquad \cdot \left(2 \cdot x_{b_i} - x_{a_i} \cdot \left[1 - \sigma\left(\dfrac{x_{a_i}}{x_{b_i}}\right)\right] - x_{c_i} \cdot \left[1 - \sigma\left(\dfrac{x_{c_i}}{x_{b_i}}\right)\right]\right) \\
F_{\gamma_i}(\vec{x}) = \sigma\left(\dfrac{x_{a_i}}{x_{b_i}}\right) \cdot x_{b_i} \cdot \sigma\left(\dfrac{x_{c_i}}{x_{b_i}}\right) \cdot \left[1 - \sigma\left(\dfrac{x_{c_i}}{x_{b_i}}\right)\right]
\end{array} \right. \\
(ii) & \left\{ F_{b_i}(\vec{x}) = \sum_j w_{\beta_{ij}} \cdot x_{\beta_j} \right. \\
(iii) & \left\{ \begin{array}{l}
F_{c_i}(\vec{x}) = \sum_j w_{\gamma_{ij}} \cdot x_{\delta_j} \\
F_{d_i}(\vec{x}) = \sum_j w_{\gamma_{ji}} \cdot x_{\gamma_j}
\end{array} \right. \\
(iv) & \left\{ \begin{array}{l}
F_{\delta_i}(\vec{x}) = \sigma\left(\dfrac{x_{d_i}}{x_{e_i}}\right) \cdot \left[1 - \sigma\left(\dfrac{x_{d_i}}{x_{e_i}}\right)\right] \cdot x_{e_i} \cdot \sigma\left(\dfrac{x_{f_i}}{x_{e_i}}\right) \\
F_{\epsilon_i}(\vec{x}) = \sigma\left(\dfrac{x_{d_i}}{x_{e_i}}\right) \cdot \sigma\left(\dfrac{x_{f_i}}{x_{e_i}}\right) \cdot \\
\qquad\qquad \cdot \left(2 \cdot x_{e_i} - x_{d_i} \cdot \left[1 - \sigma\left(\dfrac{x_{d_i}}{x_{e_i}}\right)\right] - x_{f_i} \cdot \left[1 - \sigma\left(\dfrac{x_{f_i}}{x_{e_i}}\right)\right]\right) \\
F_{\phi_i}(\vec{x}) = \sigma\left(\dfrac{x_{d_i}}{x_{e_i}}\right) \cdot x_{e_i} \cdot \sigma\left(\dfrac{x_{f_i}}{x_{e_i}}\right) \cdot \left[1 - \sigma\left(\dfrac{x_{f_i}}{x_{e_i}}\right)\right]
\end{array} \right.
\end{array}
\tag{3.21}
$$

for $i \in \{1, 2, \cdots, n\}$, as depicted in figure 3.4. The units x_{β_i} and x_{b_i} carry the recurrent projection from the first layer to itself, the units x_{γ_i} and x_{d_i} carry the projection from the first layer up to the second layer, and the units x_{δ_i} and x_{c_i} carry the projection from the second layer back to the first layer. In a traditional hierarchical artificial neural network, only the intrinsic gradient network units with Greek-letter subscripts, like x_{β_i}, x_{γ_i}, and x_{δ_i}, would be directly represented by neural network units; the weighted sums performed by the intrinsic gradient network units with Latin-letter subscripts, like x_{b_i}, x_{c_i}, and x_{d_i}, would be performed within Greek-subscripted neural network units. Nevertheless, these weighted sums between and within layers are identical to those found in traditional artificial neural networks.

Instead of applying a direct sigmoidal transfer function to these weighted sums, output functions $F_{\gamma_i}(\vec{x})$ and $F_{\delta_i}(\vec{x})$ first normalize the bottom-up and top-down weighted sums by the recurrent weighted sum, and scale their output by the recurrent signal and a simple function of the reciprocal signal. Similarly, $F_{\beta_i}(\vec{x})$ is a simple function of sigmoids of the normalized inputs. The Greek-subscripted units have no constant offset, in contrast to traditional neural networks, but the divisive normalization by the recurrent signal can serve a similar function of centering the input within

the appropriate portion of the dynamic range of the sigmoid. An actual constant offset to the dynamics of an intrinsic gradient network would correspond to a linear error function, as discussed in section 2.4.

We thus see that simple intrinsic gradient networks can be constructed with dynamics similar to those of traditional systems like belief propagation on loopy factor graphs or hierarchical artificial neural networks. However, due to their carefully designed dynamics, intrinsic gradient networks can be trained directly and exactly once they reach a fixed point, without the need for a separate backpropagation stage or approximations of uncertain fidelity. More complicated intrinsic gradient networks may yield even more interesting dynamics.

3.3.1 An intuitive description of the derived computational paradigm

It has been repeatedly observed that feature extraction followed by pooling of similar features is an effective strategy for object recognition (Fukushima, 1980; Lecun et al., 1998; Serre et al., 2005; Jarrett, et al., 2009; Coates et al., 2010). In the traditional framework, feature extraction generally consists of a linear filter followed by a static sigmoidal transfer function, while pooling can consist of a sum, max, or other monotonically increasing function over a fixed set of extracted features. The linear filters are generally trained to minimize some error function, although they may be partially fixed *a priori*, as in the case of the first layer of HMAX (Serre et al., 2005), and can yield surprisingly good results even when set randomly (Jarrett, et al., 2009). The pooling stage is generally assumed to be fixed.

Intrinsic gradient networks of the form described in this section can perform operations qualitatively similar to both feature extraction and pooling using distinct, hard-wired mechanisms within each layer. Consider the output from unit x_{ϕ_i} at the fixed point:

$$\begin{aligned} x_{\phi_i} = F_{\phi_i}(\vec{x}) &= \sigma\left(\frac{x_{d_i}}{x_{e_i}}\right) \cdot x_{e_i} \cdot \sigma\left(\frac{x_{f_i}}{x_{e_i}}\right) \cdot \left[1 - \sigma\left(\frac{x_{f_i}}{x_{e_i}}\right)\right] \\ &= \sigma\left(\frac{\sum_j w_{\gamma_{ji}} \cdot x_{\gamma_j}}{x_{e_i}}\right) \cdot x_{e_i} \cdot \sigma\left(\frac{x_{f_i}}{x_{e_i}}\right) \cdot \left[1 - \sigma\left(\frac{x_{f_i}}{x_{e_i}}\right)\right] \\ &= \sigma\left(\frac{\sum_j w_{\gamma_{ji}} \cdot x_{\gamma_j}}{x_{e_i}}\right) \cdot x_{e_i} \cdot \nu_i \,, \end{aligned} \tag{3.22}$$

as follows from equation 3.21, defining $\nu_i = \sigma\left(\frac{x_{f_i}}{x_{e_i}}\right) \cdot \left[1 - \sigma\left(\frac{x_{f_i}}{x_{e_i}}\right)\right]$. Ignoring the effect of x_{e_i} and ν_i, unit x_{ϕ_i} performs feature extraction consisting of a sigmoid of a linear transformation of the input, $\vec{x}_\gamma$. This operation arises from the combined effect of the factor node corresponding to group (iii) and the variable node corresponding to group (iv) in equation 3.21.

Of course, x_{e_i} and ν_i also affect x_{ϕ_i} at the fixed point. Indeed, the x_{e_i} can induce behavior similar to max pooling. After extending the intrinsic gradient network of equation 3.21 in the obvious way

to include $F_{e_j}(\vec{x}) = \sum_i w_{\epsilon_{ji}} \cdot x_{\epsilon_i}$, the unit x_{e_j} is a function of x_{ϵ_i}. Equation 3.21 implies that

$$F_{\epsilon_i}(\vec{x}) = F_{\delta_i}(\vec{x}) \cdot \left(\frac{1}{1 - \sigma\left(\frac{x_{d_i}}{x_{e_i}}\right)} - \frac{x_{d_i}}{x_{e_i}} \right) + F_{\phi_i}(\vec{x}) \cdot \left(\frac{1}{1 - \sigma\left(\frac{x_{f_i}}{x_{e_i}}\right)} - \frac{x_{f_i}}{x_{e_i}} \right),$$

so at a fixed point x_{ϵ_i} is a weighted sum of x_{δ_i} and x_{ϕ_i}, the feedback and feedforward message associated with unit x_{ϵ_i}. At a fixed point, x_{e_i} is the weighted sum of the x_{ϵ_j}, so x_{e_i} converges to something like a scaled, weighted average of the feedforward and feedback messages out of the node. Plugging this result into equation 3.22, x_{ϕ_i} is normalized and scaled relative to a group of similar units.

An appropriate set of non-negative recurrent parameters $w_{\epsilon_{ji}}$ can cause groups of units within each layer to mutually suppress each other. So long as the normalizing effect of x_{e_i} dominates over the scaling effect of x_{e_i} in equation 3.22, the largest feedforward and feedback messages out of the node will more effectively suppress the other units than vice versa, thereby reducing the suppression of the largest units in a feedback loop. At the fixed point of this feedback process, the feedforward unit with the largest output should be considerably more active than the others, constituting an approximate max operation consistent with the pooling stage in traditional hierarchical object recognition systems.

Since something like feature extraction and pooling are performed by distinct portions of equation 3.21, they will inevitably be interleaved, giving rise to something like a convolutional network (Lecun et al., 1998). Both the extracted features and the groups over which pooling is performed are trainable with the gradient computed at the fixed points in an intrinsic gradient network. Unlike traditional convolutional networks, the intrinsic gradient network described in this section is highly recurrent, potentially facilitating powerful computations, as described in section 4.3.2. This computational power follows not from a wholly *ad hoc* choice of network dynamics, but rather from the biologically-inspired requirement that the gradient be easily calculable from the fixed points.

3.4 Practical implementations of highly recurrent intrinsic gradient networks

Up until this point, we have taken an analytic approach to intrinsic gradient networks. To this end, we have successfully constructed a large class of novel, highly recurrent networks for which the gradient of the error function can be calculated from the intrinsic signals at the fixed points. This gradient can be used to train the parameters of the network to minimize the error function. Indeed, a variety of powerful minimization algorithms, including stochastic gradient descent, quasi-Newton methods, and conjugate gradients, require only the signals directly available within an intrinsic

gradient network (Bishop, 1995). We have also shown that both belief propagation on acyclic factor graphs and recurrent backpropagation are instances of intrinsic gradient networks, so it is clear that intrinsic gradient networks can be at least as computationally powerful and easily trainable as these familiar algorithms. Unfortunately, neither belief propagation on acyclic factor graphs nor recurrent backpropagation are highly recurrent, so it is difficult to reconcile these networks with the recurrence of the cortex (Felleman & Van Essen, 1991; Douglas & Martin, 2004).

Our novel, highly recurrent intrinsic gradient networks, such as those constructed in sections 3.2 and 3.3, do not suffer from this inconsistency, and in this sense constitute more plausible models of cortical computation. As we will discuss further in section 4.3, such highly recurrent intrinsic gradient networks can have a connection topology consistent with that observed in the cortex. Moreover, we will find that gradient descent along the error function of such networks consists of a pseudo-Hebbian update, consistent with the adjustment of synaptic strength in the cortex (Malenka & Bear, 2004).

However, we have not yet established that our novel, highly recurrent intrinsic gradient networks can be successfully trained to perform interesting computations. Such a failure of training could take three forms: our novel, highly recurrent intrinsic gradient networks could prove unable to find a fixed point of the output functions, and thus fail to calculate the gradient; the gradient could be insufficient to efficiently descend the convoluted error function landscape; or the local minima to which the training algorithm converges could have insufficiently low error.

We partially address the concern that a fixed point will not be found in appendix A.5, where we construct a class of highly recurrent intrinsic gradient networks with dynamics that provably converge to a fixed point of the output functions. Moreover, we demonstrate in appendix A.1 that approximate fixed points are sufficient for training. Specifically, an approximate gradient can be calculated using $\vec{x}$ such that $\vec{F}(\vec{x}) \approx \vec{x}$; it does not matter if this $\vec{x}$ is far from any $\vec{x}^*$ where $\vec{F}(\vec{x}^*) = \vec{x}^*$. Nevertheless, we might still be worried that typical intrinsic gradient networks with simple, intuitive dynamics, like those of equation 2.1, enter a limit cycle or chaotic attractor and never approach anything like a fixed point of the output functions. In such cases, the training function $\vec{T}(\vec{x})$ would not necessarily compute the gradient of the intrinsic gradient network accurately, and a gradient-based learning algorithm using $\vec{T}(\vec{x})$ might fail to minimize the error function.

Even if a fixed point can be found and the gradient can be calculated, the error function landscape could be too convoluted to succumb to gradient-based minimization techniques. Recurrent networks can be highly nonlinear, and small changes to the parameters can have large, contextually-dependent effects. The error function landscape could thus contain deep, winding canyons, such that any local, low-order approximation is inaccurate beyond a small region surrounding the current point in parameter space. Since gradient-based training algorithms are dependent upon such local, low-order approximations, they are extremely inefficient in such circumstances.

Finally, we might worry that the local minima found by our intrinsic gradient networks do not actually solve our intended problem. For instance, if the network outputs oscillate rapidly but shallowly as a function of the parameters, it is easy to imagine that the error function landscape might contain many small, shallow local minima corresponding to each cycle of the oscillations. Since the outputs of the network and thus the value of the error function vary little within each cycle, such local minima would be little better than a random point in parameter space, and the vast majority of the local minima would not reduce the error function below an acceptable threshold.

Even the global minimum of the error function may not be low enough, as often occurs with perceptrons. Whereas feedforward artificial neural networks with a single hidden layer are universal function approximators (Bishop, 1995), feedforward neural networks without hidden units (i.e., perceptrons) can only perform linear discriminations (Minsky & Papert, 1969). A simple perceptron can be trained simply, effectively, and in a biologically plausible manner, but it cannot compute XOR of two inputs, regardless of the choice of parameters. In the absence of a proof that all intrinsic gradient networks are universal function approximators, we might thus be concerned that our novel, highly recurrent intrinsic gradient networks suffer from a similar limitation and cannot capture the required input-output mappings.

These issues are certain to arise in some cases. For instance, if the functions $h_j^k(x)$ of equation 2.17 are chosen to have large, high-frequency oscillations, then the dynamics $\vec{x}(t+1) = \vec{F}(\vec{x}(t))$ of equation 2.1 are unlikely to converge. On the other hand, the simple homogeneous solutions discussed in section 2.3.5, such as $\vec{F}(\vec{x}) = 0$, converge quickly using the dynamics of equation 2.1. However, they are clearly not powerful enough to compute nontrivial input-output mappings.

These potential failures of effective training are difficult to characterize analytically. Finding Lyapunov functions to prove that a set of dynamics converges to a fixed point is something of a black art, and determining whether an arbitrary recurrent network converges is a manifestation of the non-computable halting problem, since Turing machines can be formulated as recurrent networks. The speed of learning is often sensitive to hyperparameters like the initialization of the parameters, and so is not well defined by the network architecture alone (LeCun et al., 1998). In general, the computational power of intrinsic gradient networks can only be evaluated analytically given a more explicit characterization of the fixed points than we currently possess.

3.4.1 A proof-of-concept implementation

It is thus sensible to address the possible failure of effective training by investigating some practical examples. In this section, we describe two hierarchical intrinsic gradient networks similar to those of section 3.3, and show that they can accurately calculate the gradient and learn nontrivial input-output mappings via stochastic gradient descent. We assess their performance on a generalization of the XOR problem and on handwritten digit recognition. These examples establish that, while

we cannot rule out any of the three failure mechanisms described above on a theoretical basis, highly recurrent intrinsic gradient networks need not fall prey to any of them. Moreover, we use two different sets of output functions (one polynomial like in section 3.2, and one sigmoidal as in section 3.3) to demonstrate that the efficacy of intrinsic gradient networks is not dependent on any particular choice of $h_j^k(x)$ in equation 2.16.

The examples constructed in this section are intended solely as a proof-of-concept. The performance of our example intrinsic gradient networks on handwritten digit recognition is not state-of-the-art, but this should not be interpreted as a reflection on the overall capabilities of intrinsic gradient networks. Intrinsic gradient networks must converge to a fixed point, and so require many more computations to produce an output than a feedforward neural network of comparable size. To minimize the computational burden of our simulations, we consider only extremely small intrinsic gradient networks. However, a large feature space, consisting of many hidden units, is essential for good performance on visual recognition tasks using traditional feedforward networks (Coates et al., 2010). Typical feedforward artificial neural networks used to perform handwritten digit recognition use hundreds if not thousands of units in the hidden layer (Simard et al., 2003; Ciresan et al., 2010), whereas we restrict ourselves to only thirty-six units in each direction per hidden layer. It is plausible to hypothesize that performance would improve significantly were we to use larger intrinsic gradient networks.

The relative inefficiency of intrinsic gradient networks is exacerbated by the serial nature of modern CPU architectures, which can only update (on the order of) one unit at a time. In the cortex, in contrast, each unit could be implemented by a disjoint set of neurons operating in parallel, so all units could be updated simultaneously, rendering the time required to find a fixed point much less dependent on the size of the network. Future work will need to investigate the scaling properties of intrinsic gradient networks, perhaps using GPUs to mimic the parallel dynamics of the cortex.

Classical machine learning problems, like classification and regression, do not perfectly align with the structure of intrinsic gradient networks. As discussed in section 2.4, intrinsic gradient networks induce a particular relationship between the inputs and the error function. Every output on which the error function depends makes a complementary contribution to the input. In contrast, supervised learning paradigms like classification and regression define the error function in terms of a set of outputs for which there are no complementary inputs. We wish to demonstrate that intrinsic gradient networks can be used to solve such problems, while recognizing that supervised learning paradigms are not the only way to formalize cortical learning. To solve classification problems with intrinsic gradient networks, we choose a particular mapping of inputs and outputs onto the network, along with an attendant set of approximations to the gradient of the supervised error function. While this approach is sensible, there are many other ways to perform classification with intrinsic gradient networks, some of which are likely to yield superior performance.

The examples discussed in this section also constitute a mere proof-of-concept, in that we focus our attention on a single connection topology and only two sets of functions $h_j^k(x)$. These hyperparameters bear a superficial resemblance to those that have proven effective in feedforward neural networks, but there is no real reason to believe that the resulting intrinsic gradient networks are optimal, or even typical of the performance of the class of intrinsic gradient networks constructed in section 2.3. The space of highly recurrent intrinsic gradient networks captured by equation 2.17 is even broader than the space of feedforward artificial neural networks. Just as the hyperparameters of feedforward neural networks must be carefully selected to obtain good performance (LeCun et al., 1998), it is likely that the performance of intrinsic gradient networks is similarly dependent on the hyperparameters. Our experiments in this section constitute a first stab in the darkness of this enormous parameter space. Additional theory and practical experience will be necessary to distinguish the most powerful intrinsic gradient networks from poor and average intrinsic gradient networks. Such explorations are beyond the scope of this thesis.

Since we are interested in evaluating the computational power of highly recurrent intrinsic gradient networks in general rather than solving a particular machine learning task, we do not perform any preprocessing on the inputs. In contrast, standard systems for performing handwritten digit recognition often make use of explicit brightness and contrast normalization, if not more sophisticated whitening of the data (Coates et al., 2010). Systems like HMAX even preprocess the inputs using an untrained linear-nonlinear filter bank (Serre et al., 2005). Our performance would likely improve were we to make use of these standard but powerful augmentations.

Despite these caveats, the examples developed in this section demonstrate that our novel, highly recurrent intrinsic gradient networks can behave sensibly. Moreover, they serve as a basis for future explorations.

3.4.2 Splitting the error function into a wake component and a sleep component

In section 2.4.2, we constructed solutions to the full inhomogeneous intrinsic gradient equation (2.9) with the error function chosen to be the familiar sum of squares error, $E_{SS}(\vec{x}) = \frac{1}{2} \cdot \sum_i (x_i - c_i)^2$. In additional to being computationally parsimonious, the sum of squares error corresponds to the negative log likelihood of the input if the network is understood to induce independent Gaussian probability distributions centered on each output (Bishop, 1995). The sum of squares error (and the complementary negative sum of squares objective function) is thus a well-motivated choice for the error function. As discussed in section 2.4, we prefer to maximize the negative sum of squares error rather than minimize the sum of squares error, since the negative sum of squares error induces input signals that are of the same sign as the optimal output signals.

However, a direct implementation of the sum of squares error has some undesirable properties in the highly recurrent intrinsic gradient network investigated in this section. Unlike a linear error function or a Kullback-Leibler divergence with unnormalized outputs, the minimum of the sum of squares error (and the maximum of the negative sum of squares error) is achievable with finite signals; specifically, when $x_i = c_i$ for all outputs x_i. At these points, the gradient is necessarily equal to zero if it is defined. In an intrinsic gradient network, the gradient is a simple function of the fixed point, so a zero gradient has strong implications for the optimal fixed points. Specifically, as per equation 2.7, $\frac{dE(\vec{x})}{dw'} = \sum_i T_i(\vec{x}) \cdot \left(\frac{\partial F_i(\vec{x},\vec{w})}{\partial w'} \right)$ at the fixed point. If the training function $\vec{T}(\vec{x}) = \mathbf{T} \cdot \vec{x}$ is a pairwise permutation (as in the networks of sections 3.2 and 3.3) and each output function $F_i(\vec{x})$ is independently parameterized, then $x_i = 0$ for all x_i that the pairwise permutation $\vec{T}(\vec{x})$ maps to x_j such that $F_j(\vec{x})$ is parameterized and $\frac{\partial F_j}{\partial \vec{w}} \neq 0$ at the fixed point. In the examples of sections 3.2 and 3.3, this would imply that all inputs in one direction to each factor node are set to zero, and consequently the outputs of the factor nodes in these directions are also zero. Such fixed points are not desirable minima for the error function, suggesting that we should choose a different error function.

An appropriate modification to the error function is suggested by an additional undesirable property of the sum of squares error. When used in conjunction with simple, intuitive dynamics like $\vec{x}(t+1) = \vec{F}(\vec{x}(t))$ of equation 2.1 and a pairwise permutation training function $\vec{T}(\vec{x}) = \mathbf{T} \cdot \vec{x}$, the log component of the input induced by the negative sum of squares error can constitute strong positive feedback. As output x_i increases, the complementary input x_j increases according to $x_i \cdot \log(|x_i|)$. Not only can this positive feedback destabilize the fixed point given a fixed set of parameters, but since the gradient is a simple function of the fixed point, positive feedback can also manifest in stochastic gradient ascent on the negative sum of squares error function.

We can avoid positive feedback due to the log component, retain the intuitive input parameterization induced by the negative sum of squares error, and simultaneously keep activity from going to zero at the minima of the error function by taking advantage of the linearity of the gradient and splitting the negative sum of squares error into two parts. Specifically, instead of $E_{NSS}(\vec{x}) = -\frac{1}{2} \cdot \sum_i (x_i - c_i)^2$, consider the two component error functions

$$\begin{aligned} E_{wake} &= \sum_i c_i \cdot x_i \text{ and} \\ E_{sleep} &= \frac{1}{2} \cdot \sum_i x_i^2 \,. \end{aligned} \tag{3.23}$$

The sum in the definition of E_{sleep} only runs over a subset of the units x_i. Since $E_{NSS}(\vec{x}) = E_{wake}(\vec{x}) - E_{sleep}(\vec{x}) - \frac{1}{2} \cdot \sum_i c_i^2$, we see that $\frac{dE_{NSS}(\vec{x})}{d\vec{w}} = \frac{dE_{wake}(\vec{x})}{d\vec{w}} - \frac{dE_{sleep}(\vec{x})}{d\vec{w}}$, where $\frac{dE}{d\vec{w}}$ is the vector of total derivatives of the error function with respect to the parameters.

3.4.3 Approximating the test network with a wake network and sleep network

Reflecting this decomposition, we construct two intrinsic gradient networks, the wake network and the sleep network, with identical internal parameters but with input parameters selected so that the networks calculate the gradients of $E_{wake}(\vec{x})$ and $E_{sleep}(\vec{x})$, respectively. Given the assumptions of section 2.3.2, the input to the wake network is $\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot \vec{G}(\vec{x})$, where $G_i(\vec{x}) = c_i$; the input to the sleep network is $\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot \vec{G}(\vec{x})$, where $G_i(\vec{x}) = -x_i \cdot \log(|x_i|)$, as discussed in section 2.4. The minima of the difference between these two error functions, evaluated at the fixed points of the respective networks, occur when $\frac{dE_{wake}(\vec{x})}{d\vec{w}} = \frac{dE_{sleep}(\vec{x})}{d\vec{w}}$. This equality constraint does not imply that either component gradient $\frac{dE_{wake}(\vec{x})}{d\vec{w}}$ or $\frac{dE_{sleep}(\vec{x})}{d\vec{w}}$ must be small, so the activity in the wake and sleep networks need not go to zero at these minima. Moreover, $-x_i \cdot \log(|x_i|)$ now provides negative feedback for large x_i in the sleep network, and is thus less subject to instability.

However, the wake and sleep networks that calculate the gradients of E_{wake} and E_{sleep} have different inputs, and so calculate their gradients at different fixed points. As a result, the difference between the gradients of E_{wake} and E_{sleep} calculated by the corresponding intrinsic gradient networks is at best an approximation of the gradient of the negative sum of squares error, which is defined on a single network at a single fixed point. Nevertheless, if we focus on the wake network, then from an intuitive perspective we simply want to maximize E_{wake} while minimizing the magnitude of the wake network outputs. The sleep network will serve as a reasonable heuristic stand-in for the wake network so long as it calculates the gradient of the squared output magnitude at similar fixed points to those found by the wake network, even if the fixed points are not identical.

More rigorously, we want to maximize the negative sum of squares error on a test network, which has inputs that reflect the information available from the environment. These inputs are potentially similar but not precisely identical to the inputs of the wake and the sleep network. To train the test network, we thus want the gradient of the negative sum of squares error, or equivalently the difference between the gradients of E_{wake} and E_{sleep}, on this test network. Reassuringly, we can show that the gradients of E_{wake} and E_{sleep} on the test network are approximately equal to the gradients of these error functions on their respective networks if the wake and sleep networks find fixed points similar to those of the test network, and the higher derivatives of the output functions $\vec{F}$ are small.

To actually compute the gradient of the wake or sleep error function on the test network, we could linearize the test network at its fixed point and perform recurrent backpropagation, using the chosen error function to provide the input solely to the backpropagation calculation (Almeida, 1987; Pineda, 1987). As we see from appendix A.1, such a linearization and backpropagation is exactly the operation performed by the forward direction of any intrinsic gradient network at a fixed point.

However, intrinsic gradient networks tie the fixed point at which the linearization is performed to the error function that determines the input to the recurrent backpropagation. The wake and sleep networks use the desired error functions, but linearize at a different point. The gradient calculated by the wake or sleep network for their respective error functions should thus be a good approximation of recurrent backpropagation in the test network if the linearization is at a sufficiently similar point, despite the difference in input, and if the gradient changes slowly with respect to this linearization point.

Specifically, from appendix A.1, we see that

$$\left(\frac{dE}{d\vec{w}}\right)^{\top} = \left(\nabla^{\top} E\right) \cdot \left(\mathbf{I} - \mathbf{J}_x\right)^{-1} \cdot \mathbf{J}_w \,,$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{J}_x$ is the Jacobian of $\vec{F}(\vec{x}, \vec{w})$ with respect to $\vec{x}$, and $\mathbf{J}_w$ is the Jacobian of $\vec{F}(\vec{x}, \vec{w})$ with respect to $\vec{w}$. Our two-stage approximation of the negative sum of squares error gradient using the wake and sleep networks thus differs from the true gradient on the test network only in the use of $\mathbf{J}_x$ and $\mathbf{J}_w$ evaluated at the wake or sleep network's fixed point, rather than the test network fixed point. As a result, the wake and sleep networks can be used to compute a good approximation of the gradient of the test network with the wake and sleep error functions if the fixed points are similar and $\vec{F}(\vec{x})$ has small higher derivatives, since then both Jacobians are similar for the wake, sleep, and test networks. Moreover, $\mathbf{J}_w$ is generally very easy to calculate directly, so it can be evaluated in the training network itself, rather than the wake and sleep networks, to yield an even more accurate approximation.

3.4.4 Matching fixed points between the wake and sleep networks

The approximation described above suffices when the wake and sleep networks have nearby fixed points, but the sleep network will often have fixed points that are unlike any single wake network fixed point. The wake network naturally provides the desired outputs as input to both the bottom-most and top-most variable node, whereas the sleep network does not provide the desired output to either. As a result, fixed points in the sleep network corresponding to a spurious combination of bottom-up and top-down inputs are common in intrinsic gradient networks like those of section 3.3; if the factor node between two hidden variable nodes has approximately uniform parameters, the two hidden variable nodes are functionally disconnected. For example, if the network is trained with images of handwritten digits as the desired output at the bottom of the network and a 1-of-n representation of the digit classification as the desired output at the top of the network, then the sleep network can easily develop fixed points consisting of a well-formed digit image and a non-corresponding but also well-formed 1-of-n digit classification. These spurious combinations are prevented in the wake network by the data-driven bottom-up and top-down input.

The computational inconvenience of unmatched wake and sleep networks reflects a more fundamental conceptual problem. If we are to use a hierarchical intrinsic gradient network like that in section 3.3 to perform classification, regression, or other typical machine learning tasks using the standard feedforward arrangement of inputs and outputs, then we must provide data-driven input only to the bottom-most variable node. The desired output should then correspond to the units complementary to the top-most variable node, which constitutes the top-level output. The lack of complementarity between the connections of the inputs and the outputs gives rise to a corresponding inconsistency in the network structure. The wake component of the sum of squares error function (E_{wake}) naturally provides the desired outputs as input to both the bottom-most and top-most variable node, whereas the sleep component (E_{sleep}) does not provide the desired output to either. We want to shape the fixed point of a test network given only the bottom-up input, which is a hybrid of the wake and sleep networks induced by E_{wake} and E_{sleep}.

This discrepancy between the inputs to the wake, sleep, and test networks, and the associated break from the conditions required for the accurate approximation of the gradient of the test network, would not have a deleterious effect if minimizing E_{sleep} naturally reduced the probability of converging to a fixed point, while maximizing E_{wake} naturally increased the probability of converging to a fixed point. Such training dynamics would ensure that wake, sleep, and test networks all have similar fixed points, since sleep network fixed points unmatched to a wake network fixed point would be eliminated, while wake network fixed points unmatched to a sleep network fixed point would generate a corresponding fixed point. Assuming that the network state is initialized randomly, this property is present if minimizing E_{sleep} for a fixed point shrinks the basin of attraction of that fixed point, while maximizing E_{wake} increases the basin of attraction for the associated fixed point. While we cannot rigorously prove the existence of this phenomenon, we can make a reasonable plausibility argument. We will focus on the change in the basin of attraction due to gradient descent on E_{sleep}. The complementary result for E_{wake} follows from a similar argument.

Consider a network with the dynamics of equation 2.1, satisfying the intrinsic gradient equation (2.9) with the slack function $\vec{S}(\vec{a},\vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$ (i.e., equation 2.10).[4] If we consider the dynamics of the network state transformed by the training function, $\vec{x}^T(t) = \vec{T}(\vec{x}(t))$, we obtain

$$\vec{x}^T(t+1) = \vec{c} + \mathbf{J}_x^\top \cdot \vec{x}^T(t)\,, \tag{3.24}$$

where $\vec{c} = \nabla E$, and $\mathbf{J}_x^\top = \nabla \vec{F}^\top$ is the transpose of the Jacobian of $\vec{F}$ with respect to $\vec{x}$, both evaluated at $\vec{x}(t)$. If $\mathbf{J}_x^\top$ has a full set of linear independent eigenvectors, then the fixed points of

[4] It is necessary to select a particular set of dynamics, since the basins of attraction are only defined relative to the dynamics.

these dynamics satisfy

$$\begin{aligned}
\vec{x}^T &= (\mathbf{I} - \mathbf{J}_x^\top)^{-1} \cdot \vec{c} \\
&= \left(\mathbf{Q} \cdot \mathbf{I} \cdot \mathbf{Q}^{-1} - \mathbf{Q} \cdot \mathbf{\Lambda} \cdot \mathbf{Q}^{-1}\right)^{-1} \cdot \vec{c} \\
&= \left[\mathbf{Q} \cdot (\mathbf{I} - \mathbf{\Lambda}) \cdot \mathbf{Q}^{-1}\right]^{-1} \cdot \vec{c} \\
&= \mathbf{Q} \cdot (\mathbf{I} - \mathbf{\Lambda})^{-1} \cdot \mathbf{Q}^{-1} \cdot \vec{c} \\
&= \sum_i p_i \cdot \frac{1}{1 - \Lambda_{ii}} \cdot \vec{q}_i \,,
\end{aligned}$$

where $\mathbf{Q}$ is a matrix with the unit-length eigenvectors of $\mathbf{J}_x^\top$ along its columns, $\mathbf{\Lambda}$ is the diagonal matrix of the corresponding eigenvalues, $\vec{q}_i$ is the ith column of $\mathbf{Q}$, and p_i is the ith element of $\mathbf{Q}^{-1} \cdot \vec{c}$. The eigenvalues satisfy $|\Lambda_{ii}| < 1$ at the stable fixed points of the dynamics of equation 3.24. Given these bounds, larger eigenvalues generally yield larger fixed points, so reducing the squared magnitude of $\vec{x}^T = \vec{T}(\vec{x})$ at the fixed points should tend to reduce the eigenvalues. The squared magnitude of $\vec{T}(\vec{x})$ is equal to that of $\vec{x}$ when $\vec{T}(\vec{x})$ is a pairwise permutation, and E_{sleep} is proportional to the squared magnitude of a subset of the elements of $\vec{x}$, so minimizing E_{sleep} should also tend to reduce the eigenvalues of $\mathbf{J}_x^\top$ at the associated fixed point.

At the same time, the eigenvectors of $\mathbf{J}_x^\top$ with the largest eigenvalues dominate the dynamics of equation 3.24; $\vec{x}^T$ moves towards the various eigenvectors with speed exponentially related to their eigenvalues. Of course, $\mathbf{J}_x^\top$ and $\vec{c}$ change as a function of $\vec{x}^T$. Nevertheless, to the extent that the eigenvectors and eigenvalues of $\mathbf{J}_x^\top$ change smoothly as a function of $\vec{x}^T$, if there are multiple fixed points near $\vec{x}^T$, the dynamics will be attracted by each fixed point to the degree that its principal eigenvectors dominate $\mathbf{J}_x^\top$ at the current point. If the eigenvalues change smoothly, fixed points with larger magnitudes, and thus larger eigenvalues, will dominate the dynamics over a larger range, and so have a larger basin of attraction. Minimizing E_{sleep} at a fixed point should reduce the eigenvalues of $\mathbf{J}_x^\top$, so minimizing E_{sleep} at a fixed point should tend to shrink the fixed point's basin of attraction. A similar argument shows that E_{wake} tends to grow with the eigenvalues of $\mathbf{J}_x^\top$, so maximizing E_{wake} at a fixed point should tend to increase the fixed point's basin of attraction.

The nomenclature "wake" and "sleep" is not accidental. This two stage learning procedure is analogous to that used in Boltzmann machines and their kin (Ackley et al., 1985; Hinton et al., 1995), which also break the parameter update into a wake stage and a sleep stage. Furthermore, the wake error function and network dynamics depend upon active sensory stimuli, as experienced by an animal while awake, whereas the sleep error function and dynamics depend only upon the intrinsic properties of the network, much like dreaming. If we choose output functions that are always or almost always positive and increasing function of their parameters, then we can see from equation 2.7 that ascent of the wake network's gradient increases parameter values, and descent of

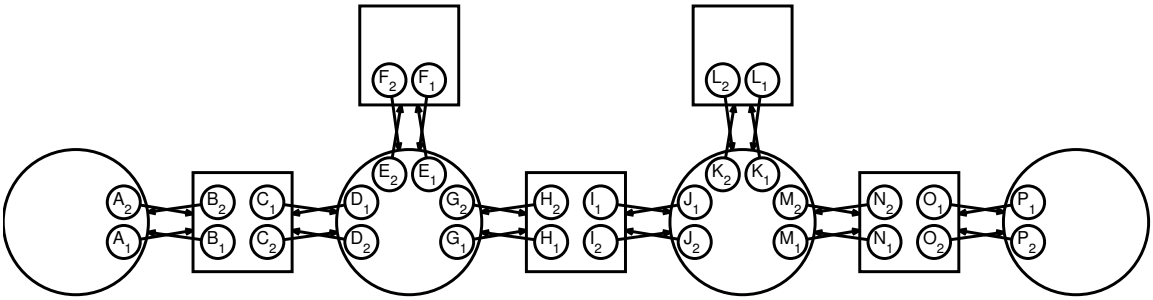


Figure 3.6: Network topology used in the examples. The bottom of the hierarchy is on the left; the top of the hierarchy is on the right. Only two units are shown projecting in each direction due to size constraints in the figure, but the examples use larger layers. An arrow from a unit terminating on a solid square or circle enclosing a group of units indicates that the unit at the source of the arrow potentially projects to all units in the enclosed group.

the sleep network's gradient decreases parameter values. This is consistent with the observation that cortical synaptic strengths tend to increase over the course of a waking interval, and decrease over the course of an interval spent asleep (Vyazovskiy et al., 2008).

3.4.5 Implementation structure

In the examples discussed below, we use a connection topology of the form depicted in figure 3.6, which extends that of section 3.3. We refer to the units A_i in figure 3.6 as the bottom-up inputs, which have complementary bottom-level outputs B_i; we refer to the units P_i as the top-down inputs, which have complementary top-level outputs O_i. The first and the last variable nodes thus correspond to the visible layers of the network, whereas the central two variable nodes constitute hidden layers. Figure 3.6 depicts only two units in each direction of each layer for the sake of size and simplicity, but in the examples we use thirty-six units in each direction of each hidden layer, and visible layers of sizes determined by the data set.

We train these intrinsic gradient networks to classify images. For each image, the constituent pixels of the image are associated with the bottom-up inputs and their complementary outputs in the network, and the one-of-n encoding of the image classification is associated with the top-down inputs and their complementary outputs. We wish to maximize the sum, over all images, of the negative sum of squares error between these inputs and their complementary outputs in a test network, in which the image is presented at the bottom-up inputs, and the classification is read out from the top-level outputs.

We use stochastic gradient descent to train the networks; on each iteration, we subtract a vector to the parameters proportional to the gradient of the sum of squares error induced by the current input. As a result, we need to calculate (an approximation of) the gradient of the negative sum of squares error for each image separately. The details of the implementation are discussed in

Figure 3.7: Colormap used in the following figures. Black always corresponds to the value 0. Positive values are represented by grays to the right; negative values are represented by reds and yellows to the left. The colormap is scaled independently for each plot to maximize the dynamic range.

appendix A.9, but are summarized here. As in section 3.4.2, we break the gradient calculation into distinct wake and sleep stages, and wish to maximize the wake error function and minimize the sleep error function. To this end, we accumulate a pair of matched wake and sleep gradients associated with a single element of the training data set before executing each parameter update. An approximation to the fixed point of both the wake network and the sleep network is found using 500 iterations of dynamics like those of equation 2.3. We calculate an approximation to ∇E_{wake} and ∇E_{sleep} from these approximate fixed points using equation 2.7. Rather than allow the magnitude of the parameter updates to vary arbitrarily, we normalize the difference between the matched wake and sleep gradients by the low-pass filtered Euclidean norm of the difference between paired wake and sleep stage gradients. We then multiply this normalized gradient by a fixed constant and add it to the parameter vector. This ensures that the average magnitude of the overall parameter update is roughly constant over time. Future work should consider choosing the step size based upon an approximation of the Hessian (LeCun et al., 1998).

After training, we evaluate the classification performance using a test network consisting of a hybrid of the wake and sleep networks. The bottom-up inputs of the test network are constant, as in the wake network, but the top-down inputs are linear functions of the complementary output, as in the sleep network. The unit activities of the test network are initialized by a heuristic propagation of information from the bottom-up inputs up the hierarchy, and then allowed to converge to a fixed point using 500 iterations of dynamics like those of equation 2.3.

3.4.6 The stripes data set: A generalization of XOR

To demonstrate the ability of intrinsic gradient networks to learn simple but nonlinear transformations, we train belief-propagating and sigmoidal intrinsic gradient networks with twelve 4x4 bottom-up input images consisting of parallel stripes with four different orientations and three offsets. Each bottom-up input image is associated with a top-down one-of-n encoding of the stripe orientation. To balance the relative contributions of the bottom-up and top-down patterns to the error function, we effectively use four copies of each one-of-n variable, although the parameters of the atomic factor nodes connecting to each of these copies are restricted to be identical. The twelve bottom-up input images are depicted in the columns labeled “opt out bot” in figures 3.12 and 3.17. White squares

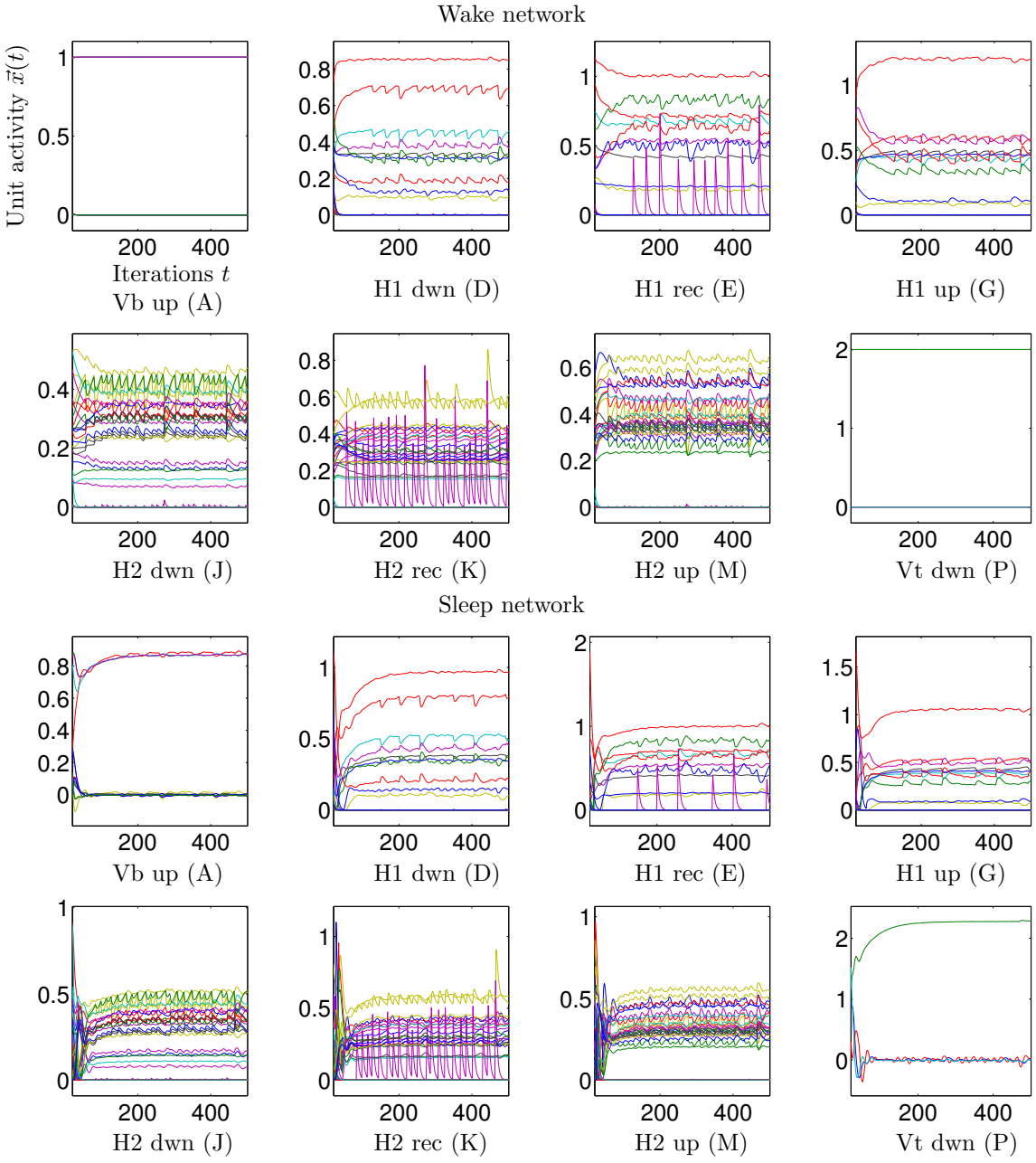


Figure 3.8: Evolution of unit activities $\vec{x}(t)$ using a belief-propagating intrinsic gradient network ($h(x) = x^{1/3}$) and parameters trained with 200,000 iterations of stochastic gradient descent on the stripes data set, alternating between the wake and sleep error functions. Activities are shown for a single representative run of the wake network and the sleep network. The x axis denotes the number of iterations of the dynamics; the y axis indicates the value of each unit. Units projecting in the same direction from a common variable node are plotted together. Factor node units are not shown, since they are just a linear transformation of the variable node units. The first twenty iterations of the activity evolution are not shown since the initial transients, resulting from the random initialization of the units, can dwarf the final activities. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6.

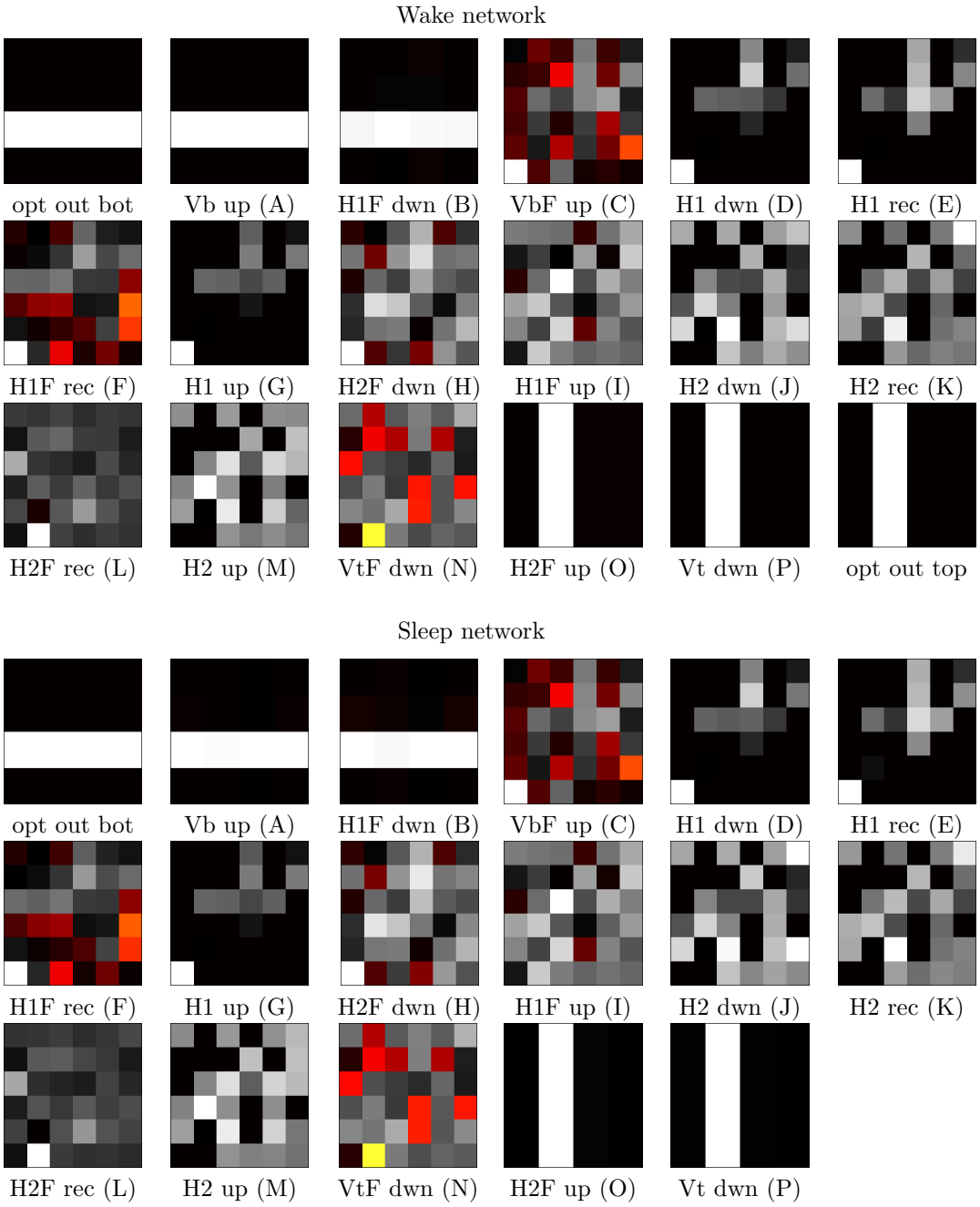


Figure 3.9: Final unit activities $\vec{x}$ after 500 iterations of the network dynamics using a belief-propagating intrinsic gradient network ($h(x) = x^{1/3}$) and parameters trained with 200,000 iterations of stochastic gradient descent on the stripes data set, alternating between the wake and sleep error functions. Each pixel corresponds to one unit in the indicated layer. Activities are shown for a single representative run of the wake network and the sleep network. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range.

correspond to inputs with value 1; black squares correspond to inputs with value 0. We refer to this as the *stripes* data set.

It can be seen that no linear classifier can produce the correct output for any of the one-of-n output units in the stripes data set, as in the classic XOR problem (Minsky & Papert, 1969). The bottom-up input images are designed so that the sum of all images with the same orientation is the identical across orientations. If $\vec{a}$, $\vec{b}$, and $\vec{c}$ are the three input images with one orientation (e.g., vertical), a linear classifier can only correctly identify these three images if there exists some weight vector $\vec{w}$ and scalar threshold t such that

$$\vec{w}^\top \cdot \vec{a} > t$$
$$\vec{w}^\top \cdot \vec{b} > t$$
$$\vec{w}^\top \cdot \vec{c} > t\,.$$

However, if $\vec{d}$, $\vec{e}$, and $\vec{f}$ are the three input images with some other orientation (e.g., horizontal), then $\vec{a}+\vec{b}+\vec{c} = \vec{d}+\vec{e}+\vec{f}$, and without loss of generality $\vec{w}^\top \cdot \vec{d} > t$, so the linear classifier for the first orientation incorrectly recognizes a pattern with the second orientation.

We train networks with the structure of figure 3.6 and thirty-six units in each direction (down, recurrent, and up) per hidden layer to perform this nonlinear classification task. We use both belief-propagating and sigmoidal intrinsic gradient networks, with $h_j^k(x)$ described in appendix A.9, to show that equation 2.17 is indeed compatible with diverse functions $h_j^k(x)$. All results are produced using the parameters learned after 200,000 iterations of stochastic gradient descent, alternating between the wake and the sleep error functions, with the learning rate $2 \cdot 10^{-2}$. Belief-propagating intrinsic gradient networks have no self-connections in the recurrent factor nodes; sigmoidal intrinsic gradient networks allow self-connections in the recurrent factor nodes.

Figures 3.8 and 3.9 show the evolution of unit activities and the final state after 500 iterations of the network dynamics of a belief-propagating intrinsic gradient network trained on the stripes data set. In these and all other figures, *opt out bot* and *opt out top* refer to the optimal outputs at the bottom and top of the hierarchy, respectively, which determine the error function and thus the input messages. *Vb* and *Vt* refer, respectively, to the "visible" inputs at the bottom of the hierarchy, corresponding to the stripe image, and the top of the hierarchy, corresponding to the stripe orientation. *H1* and *H2* refer respectively to the first and second layers of hidden variable nodes. *F* indicates that the messages have been transformed by the adjacent factor node. *Up*, *dwn*, and *rec* refer to signals propagating up the hierarchy, down the hierarchy, and recurrently from a layer to itself, respectively. It can be seen that both wake and sleep networks evolve smoothly on the whole and find a state where $\vec{F}(\vec{x}) \approx \vec{x}$, but a true fixed point is not found. After 500 iterations of unit evolution using parameters trained by 200,000 stochastic gradient descent updates, the states

of the wake and sleep networks are generally indistinguishable, implying that stochastic gradient descent has converged and that a local maximum of the negative sum of squares error has been found.

The evolution of the wake, sleep, and negative sum of squares error functions over the course of training are shown in figure 3.10, confirming that stochastic gradient ascent on the difference between the gradients calculated by the wake and sleep networks effectively maximizes the negative sum of squares error. For computational efficiency, we show the evolution of the negative sum of squares error at the fixed points of the wake network, but direct evaluation of the test network (figure 3.12) shows that it is also successfully trained. The sleep error function initially increases rather than decreases as a result of training, since it is in conflict with the wake error function; parameter changes that increase $\sum_i c_i \cdot x_i$ also tend to increase $\frac{1}{2} \cdot \sum_i x_i^2$ when $c_i, x_i \geq 0$ for all i. The effect of the sleep error function is apparent in the fact that the wake error function plateaus rather than increasing without bound.

Figure 3.11 shows the trained parameters of the factor node connecting the bottom-up inputs to the first hidden layer, corresponding to units B_i and C_i in figure 3.6. These matrices roughly correspond to the receptive field of the units of the first hidden layer. Many appear to consist of linear combinations of the bottom-up input images. Figure 3.12 shows performance of the test network, which only receives bottom-up input from the environment, after training. All input images are correctly classified.

Figures 3.13, 3.14, 3.15, 3.16, and 3.17 depict the same analyses for a sigmoidal intrinsic gradient network. On the whole, the behavior of the sigmoidal intrinsic gradient network after training is similar to that of the belief-propagating intrinsic gradient network, but the unit activities evolve more smoothly, and the negative sum of squares error increases faster and more consistently over the course of training.

3.4.7 Empirical evaluation of the gradient

We can also use the stripes data set to directly test the theoretical properties of intrinsic gradient networks. By construction, it is possible to calculate the gradient in an intrinsic gradient network, such as the network used in section 3.4.6, using the unit activities at the fixed point (as specified by equation 2.7). Moreover, in a small network, we can empirically estimate the gradient by perturbing each parameter in turn and dividing the resulting change in the error function by the size of the parameter perturbation. If our derivation is sound, then these two computations of the gradient should yield the same value. In figure 3.18, we plot the empirically estimated components of the gradient versus the components calculated by equation 2.7 for a sigmoidal intrinsic gradient network ($h(x) = x/(1+x)$) with the connection topology depicted in figure 3.6 and thirty-six units in each direction of each hidden layer, just as in section 3.4.6. The network has been trained by 200,000

stochastic gradient descent updates, and exhibits perfect classification performance, as demonstrated in figure 3.17. The top-level variable is effectively duplicated four times to balance the contributions of the bottom-up and top-down inputs to the error function.

The match between equation 2.7 and the empirically estimated gradient is nearly perfect, regardless of whether the error function used is E_{wake} or E_{sleep}, so long as the error function used to empirically estimate the gradient also induces the inputs to the network in accordance with the intrinsic gradient equation (2.9). We can also see that linear inputs are consistent with an error function equal to the Euclidean norm of the complementary outputs, so long as the inputs are held constant when perturbing the parameters, as implied by the discussion in appendix A.9. Finally, we can see that the fixed points of the wake and sleep networks together produce a reasonable approximation of the empirical gradient on the test network. In all cases, the functions E_{wake} and E_{sleep} of equation 3.23 are mixed, as described in appendix A.9.

3.4.8 The MNIST data set: Handwritten digit recognition

Figures 3.19, 3.20, 3.21, 3.22, and 3.23 repeat the analyses of section 3.4.6 for a belief-propagating intrinsic gradient network trained on the threes, fours, and fives from the MNIST data set of 60,000 centered, scaled, 28 x 28 grayscale handwritten digits. Although there are 784 bottom-layer input variables corresponding to the pixel representation of each handwritten, we only duplicate the top-layer input variable carrying the one-of-n representation of the digit's ID sixty-four times. Figures 3.24, 3.25, 3.26, 3.27, and 3.28 repeat these analyses for a sigmoidal intrinsic gradient network. As in the belief-propagating intrinsic gradient network, the top-layer input variable carrying the one-of-n representation of the digit's ID is duplicated sixty-four times. Many of the receptive fields in figure 3.27 have a structure similar to the Gabors commonly observed in V1. As can be seen from figure 3.28, when a digit is misclassified, the test network often improperly interprets the pixel input, reconstructing the bottom-up input using features representative of the predicted classification, rather than the true classification.

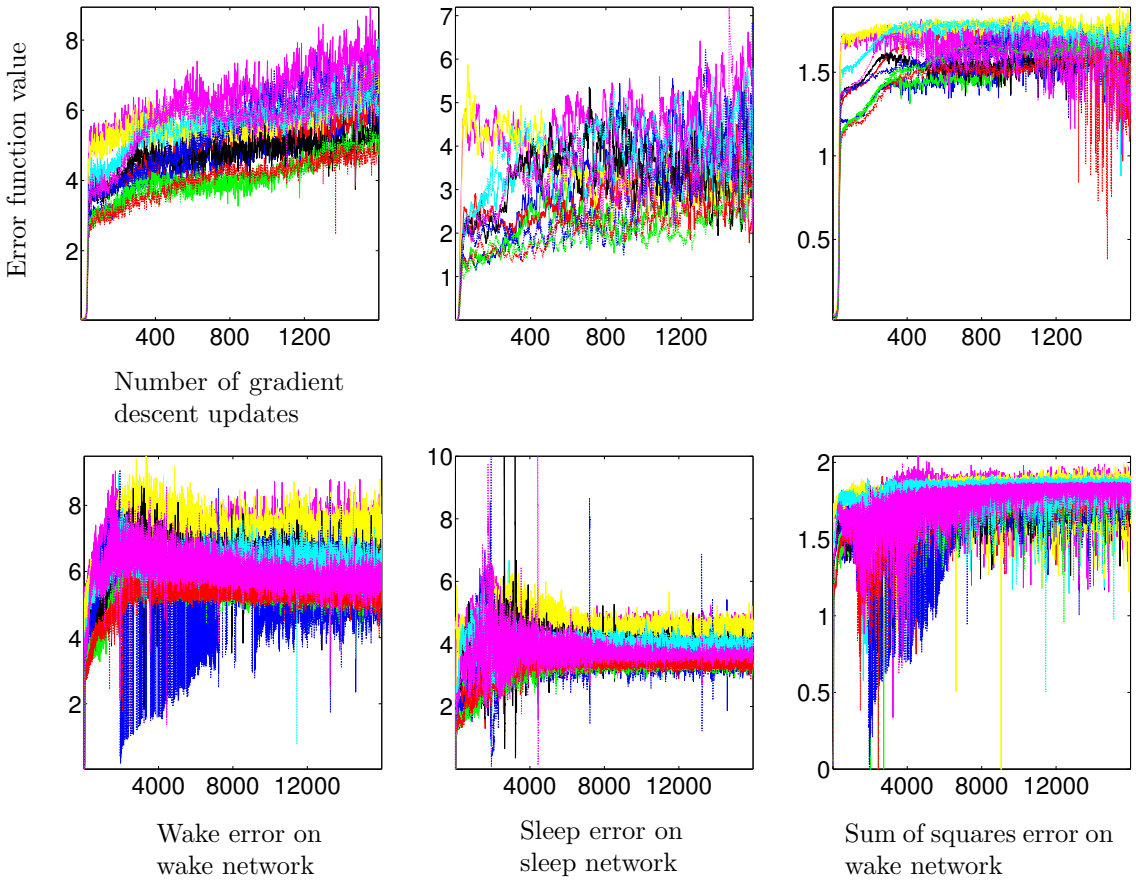


Figure 3.10: Evolution of the error functions due to stochastic gradient descent on the stripes data set starting from randomly initialized parameters using a belief-propagating intrinsic gradient network ($h(x) = x^{1/3}$). The error function associated with each element of the data set is plotted on a separate line. The x axis indexes the number of times the parameters have been updated in response to each element of the data set. The sleep network plots are smoothed with a boxcar filter of size 20 so instances when the network fails to converge to the optimal fixed point do not obscure the overall trend. Large fluctuations of the sum of squares error on the wake network arise because the network occasionally converges to a non-dominant fixed point. The two rows of figures display the evolution of the error function at two different scales, so both the initial and the asymptotic dynamics can be clearly seen.

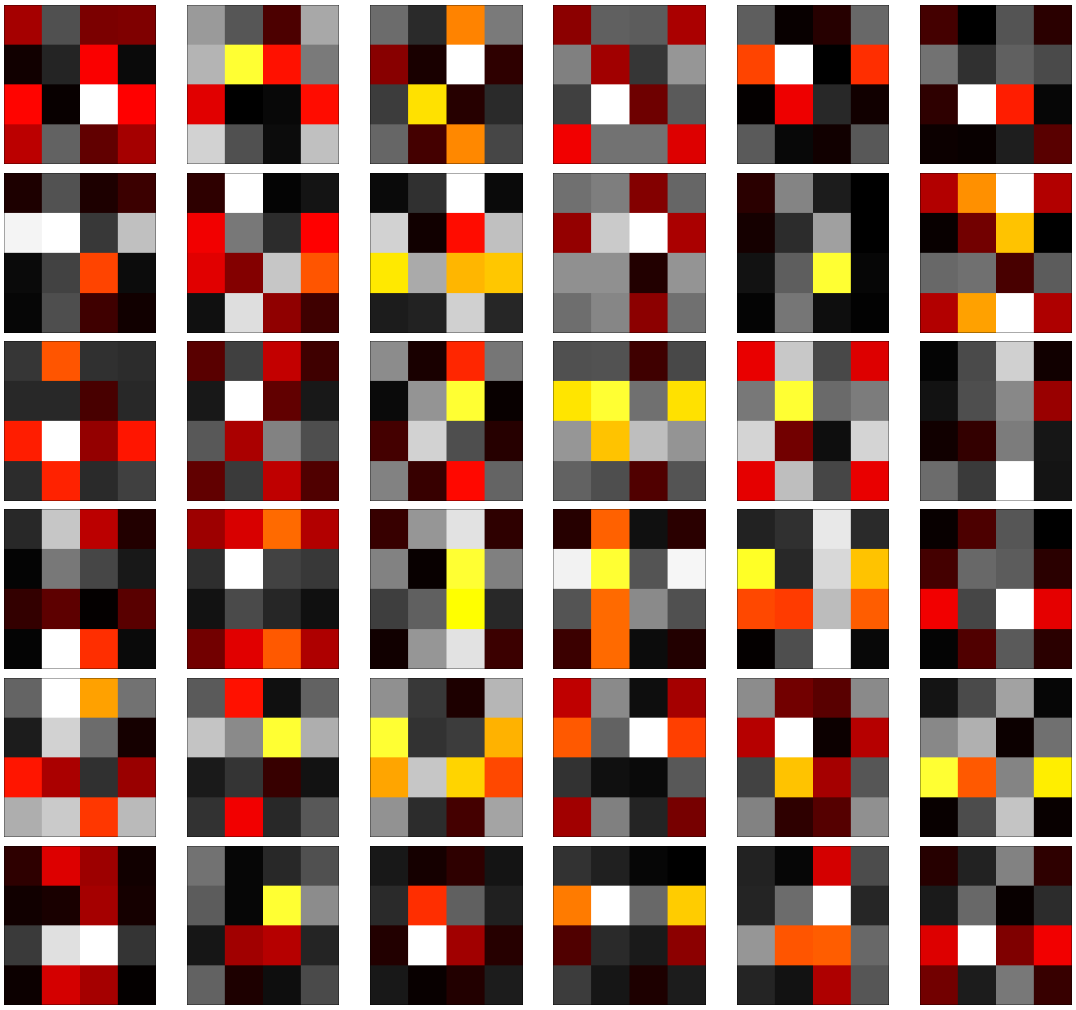


Figure 3.11: Factor matrices connecting the visible units to each hidden unit using a belief-propagating intrinsic gradient network ($h(x) = x^{1/3}$) and parameters trained with 200,000 iterations of stochastic gradient descent on the stripes data set, alternating between the wake and sleep error functions. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range.

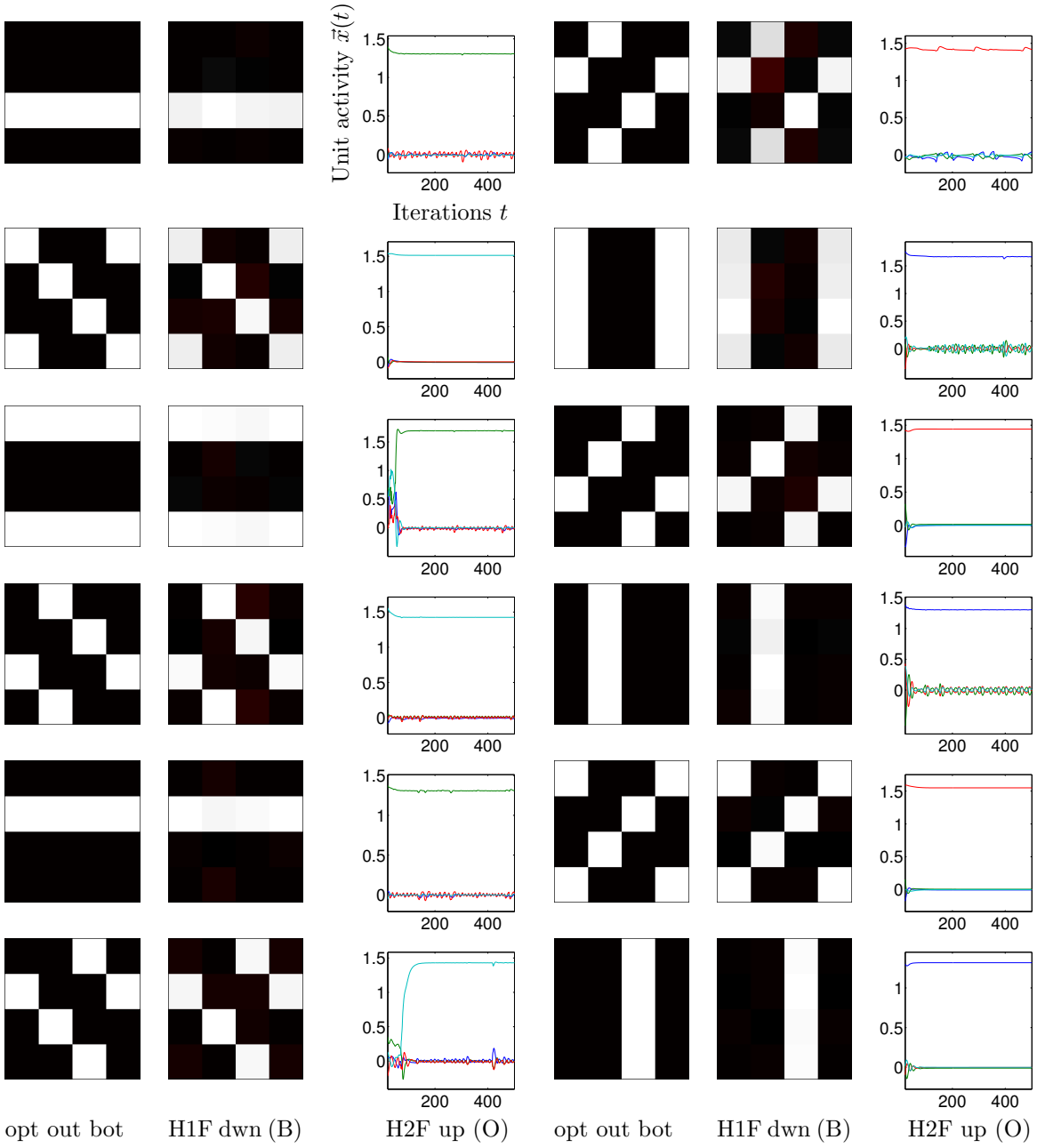


Figure 3.12: Optimal bottom-layer output, actual final bottom-layer output, and evolution of actual top-layer output over 500 iterations of the network dynamics for each element of the stripes data set using a belief-propagating intrinsic gradient network ($h(x) = x^{1/3}$) and parameters trained with 200,000 iterations of stochastic gradient descent on the stripes data set, alternating between the wake and sleep error functions. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range. In the plots of the evolution of the top-layer output, green corresponds to lines with angle 0°, red to lines with angle 45°, blue to lines with angle 90°, and cyan to lines with angle 135°.

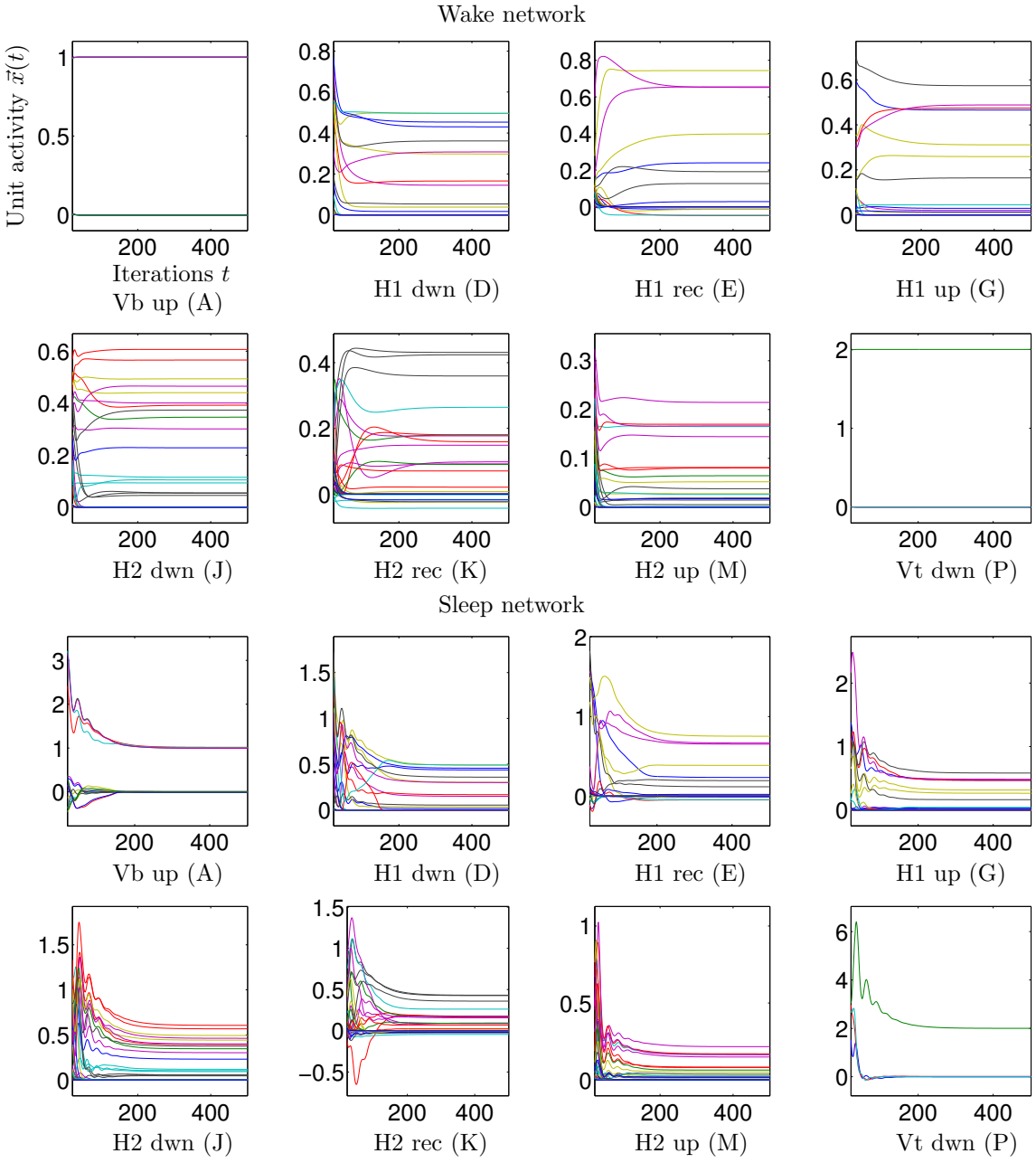


Figure 3.13: Evolution of unit activities $\vec{x}(t)$ using a sigmoid intrinsic gradient network ($h(x) = x/(1+x)$) and parameters trained with 200,000 iterations of stochastic gradient descent on the stripes data set, alternating between the wake and sleep error functions. Activities are shown for a single representative run of the wake network and the sleep network. The x axis denotes the number of iterations of the dynamics; the y axis indicates the value of each unit. Units projecting in the same direction from a common variable node are plotted together. Factor node units are not shown, since they are just a linear transformation of the variable node units. The first twenty iterations of the activity evolution are not shown since the initial transients, resulting from the random initialization of the units, can dwarf the final activities. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6.

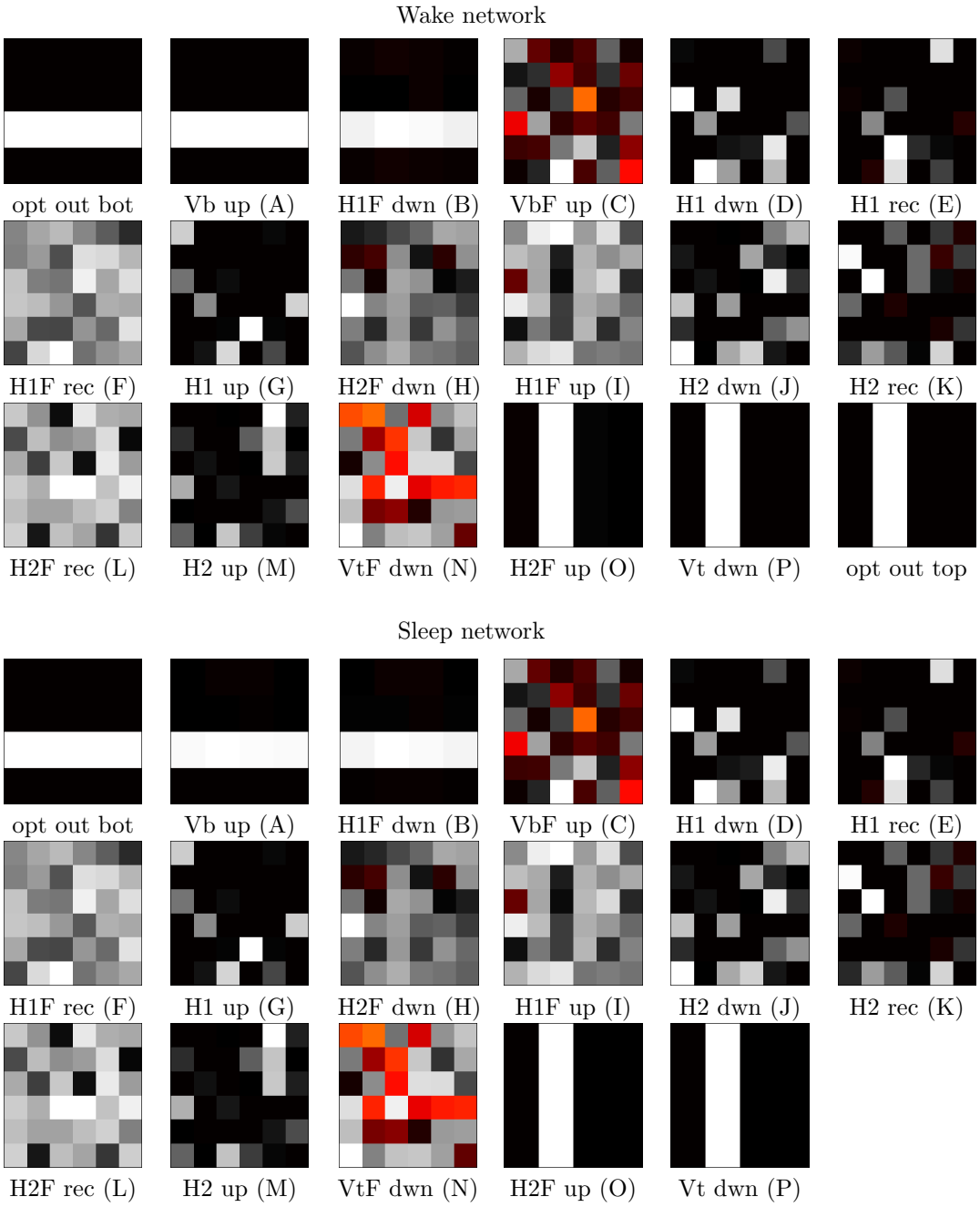


Figure 3.14: Final unit activities $\vec{x}$ after 500 iterations of the network dynamics using a sigmoid intrinsic gradient network ($h(x) = x/(1+x)$) and parameters trained with 200,000 iterations of stochastic gradient descent on the stripes data set, alternating between the wake and sleep error functions. Each pixel corresponds to one unit in the indicated layer. Activities are shown for a single representative run of the wake network and the sleep network. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range.

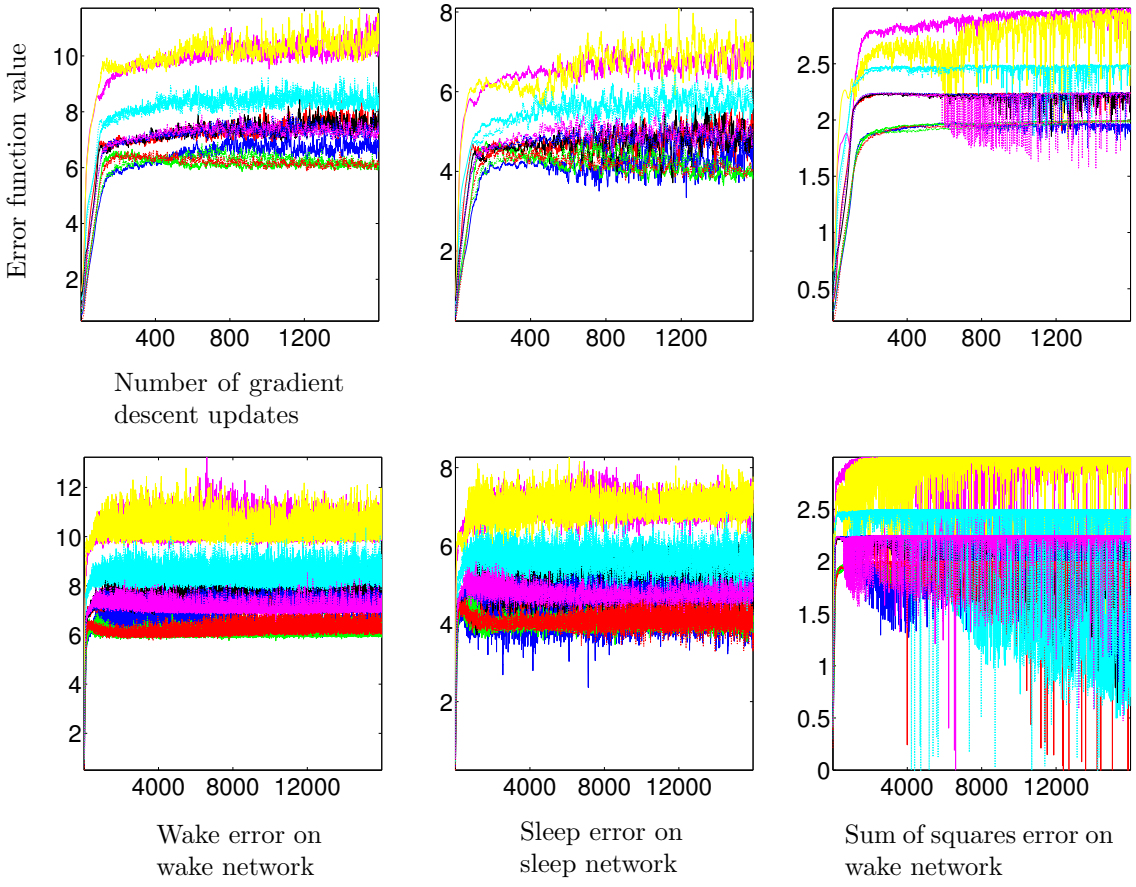


Figure 3.15: Evolution of the error functions due to stochastic gradient descent on the stripes data set starting from randomly initialized parameters using a sigmoid intrinsic gradient network ($h(x) = x/(1+x)$). The error function associated with each element of the data set is plotted on a separate line. The x axis indexes the number of times the parameters have been updated in response to each element of the data set. The sleep network plots are smoothed with a boxcar filter of size 20 so instances when the network fails to converge to the optimal fixed point do not obscure the overall trend. Large fluctuations of the sum of squares error on the wake network arise because the network occasionally converges to a non-dominant fixed point. The two rows of figures display the evolution of the error function at two different scales, so both the initial and the asymptotic dynamics can be clearly seen.

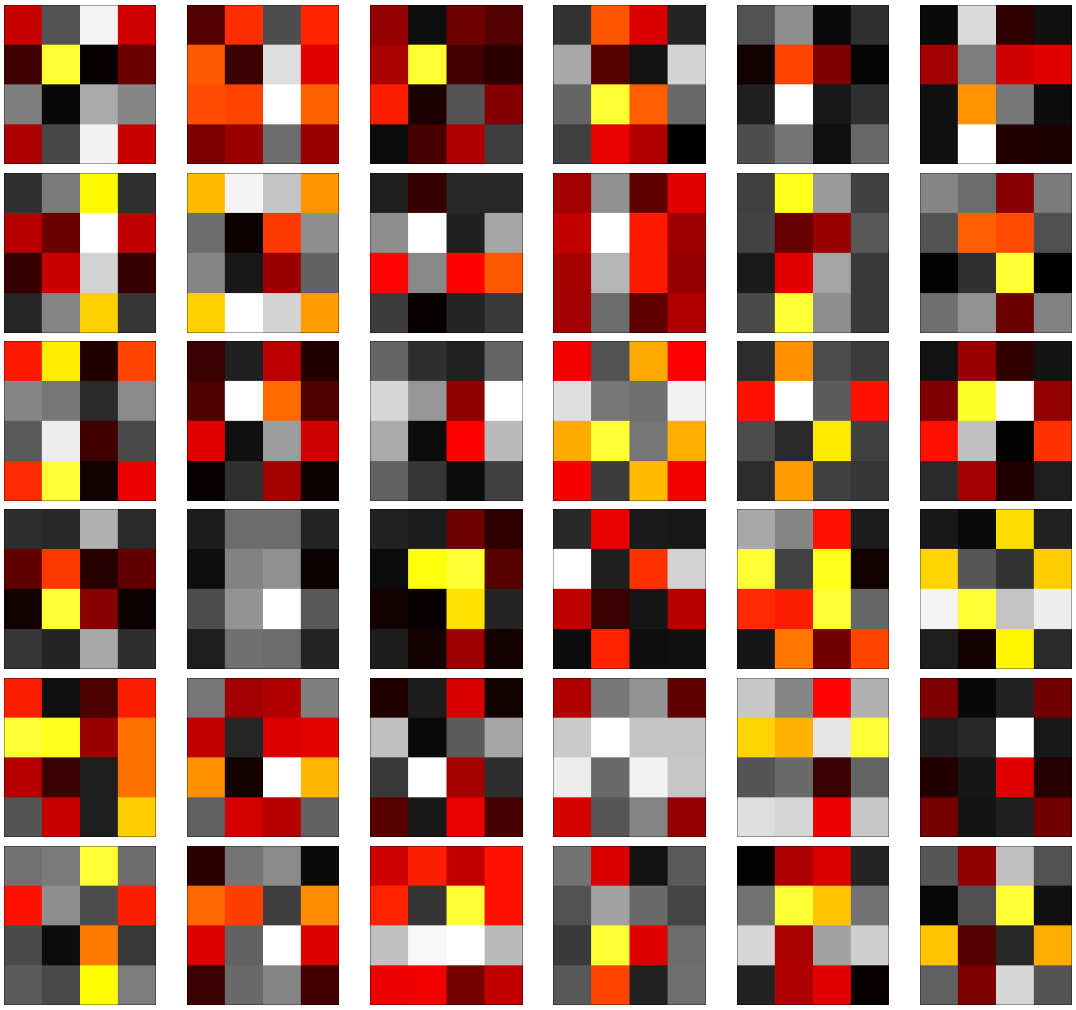


Figure 3.16: Factor matrices connecting the visible units to each hidden unit using a sigmoid intrinsic gradient network ($h(x) = x/(1 + x)$) and parameters trained with 200,000 iterations of stochastic gradient descent on the stripes data set, alternating between the wake and sleep error functions. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range.

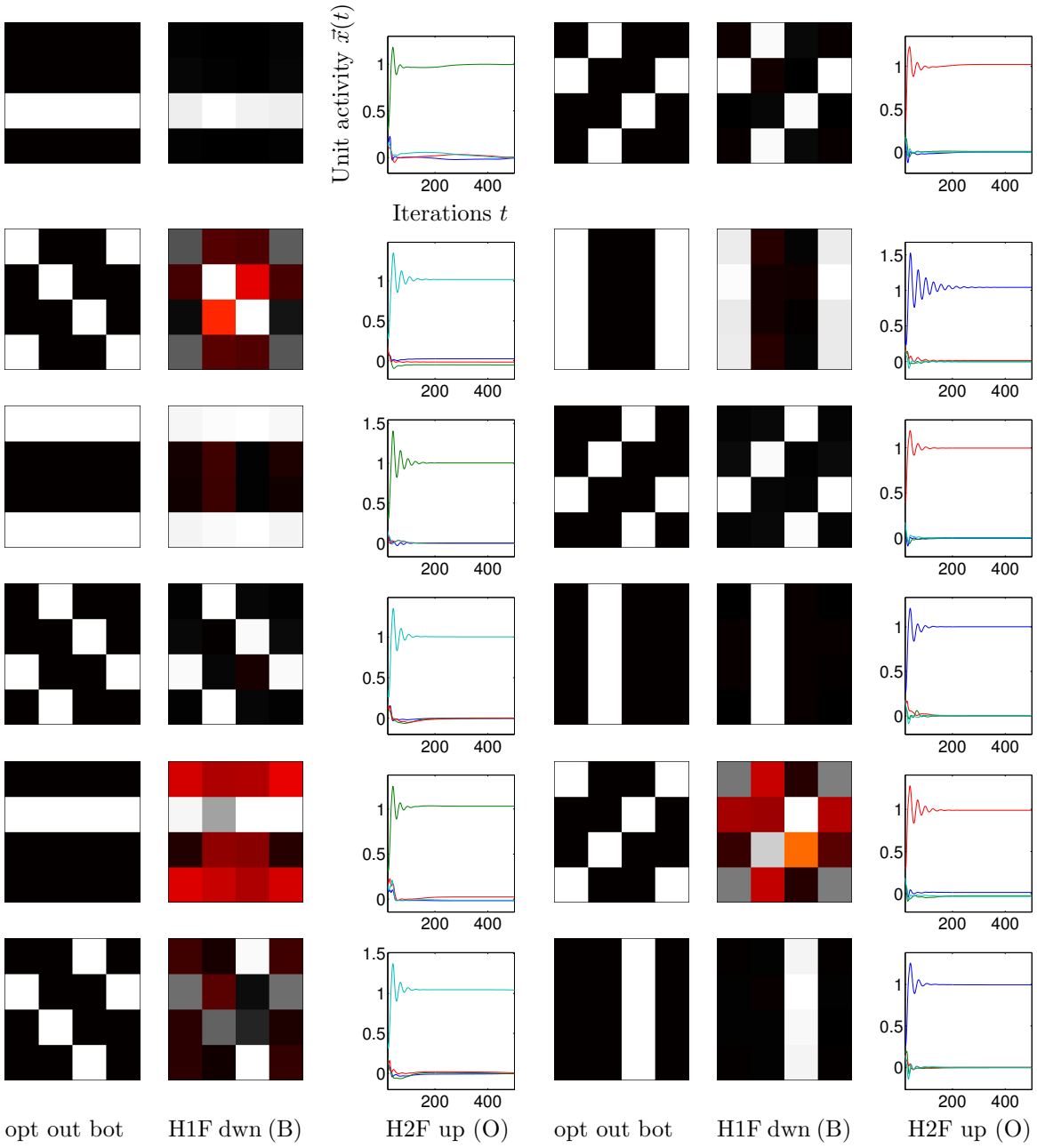
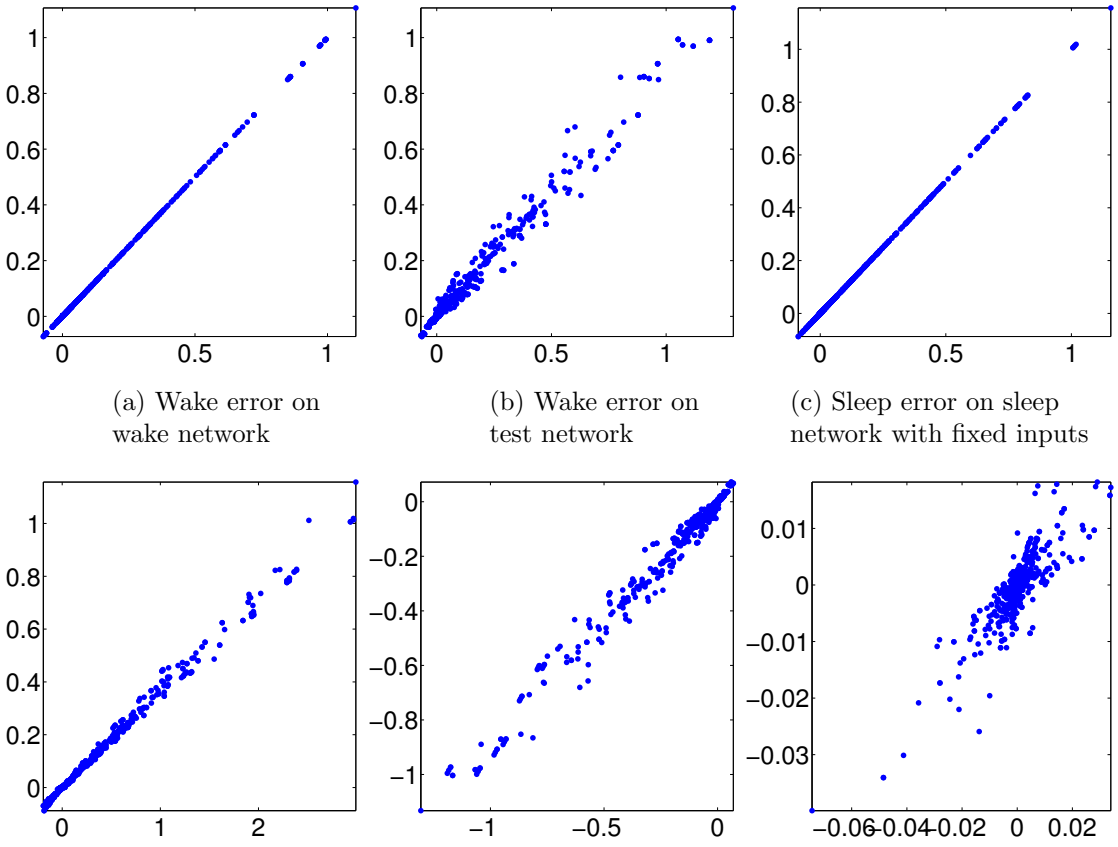


Figure 3.17: Optimal bottom-layer output, actual final bottom-layer output, and evolution of actual top-layer output over 500 iterations of the network dynamics for each element of the stripes data set using a sigmoid intrinsic gradient network ($h(x) = x/(1+x)$) and parameters trained with 200,000 iterations of stochastic gradient descent on the stripes data set, alternating between the wake and sleep error functions. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range. In the plots of the evolution of the top-layer output, green corresponds to lines with angle 0°, red to lines with angle 45°, blue to lines with angle 90°, and cyan to lines with angle 135°. Some of the trials have converged to suboptimal, non-dominant fixed points, corresponding to the large downward fluctuations in the sum of squares error plot of figure 3.15.



(a) Wake error on wake network

(b) Wake error on test network

(c) Sleep error on sleep network with fixed inputs

(d) Sleep error on sleep network with variable inputs

(e) Sleep error on test network

(f) Sum of squares error on test network

Figure 3.18: The empirical derivatives of the error function with respect to the parameters (on the x axis), estimated as the change in the error function induced by a small perturbation of a parameter divided by the size of the perturbation, versus the corresponding derivatives calculated according to equation 2.7 (on the y axis), using a belief-propagating intrinsic gradient network ($h(x) = x/(1+x)$) with the structure of figure 3.6 and thirty-six units in each direction (down, recurrent, and up) per hidden layer. (a) Empirical derivatives of E_{wake} on the wake network versus the derivatives calculated by the wake network. (b) Empirical derivatives of E_{wake} on the test network versus the derivatives calculated by the wake network. (c) Empirical derivatives of E_{sleep} on the sleep network with fixed inputs versus the derivatives calculated by the sleep network. (d) Empirical derivatives of E_{sleep} on the sleep network with freely varying inputs versus the derivatives calculated by the sleep network. (e) Empirical derivatives of E_{sleep} on the test network versus the derivatives calculated by the sleep network. (f) Empirical derivatives of E_{NSS} on the test network versus the difference of the derivatives calculated by the wake and sleep networks.

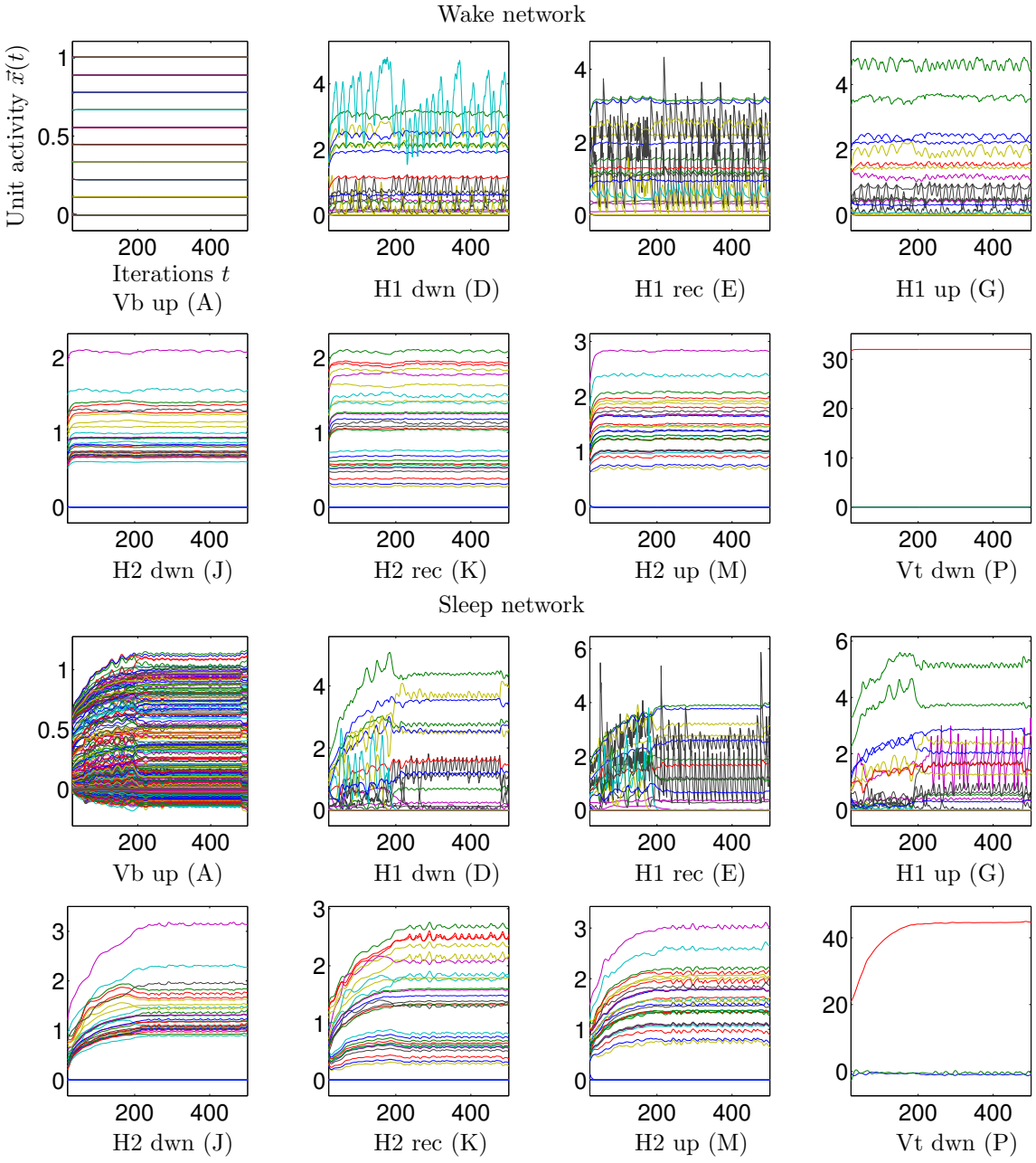


Figure 3.19: Evolution of unit activities $\vec{x}(t)$ using a belief-propagating intrinsic gradient network ($h(x) = x^{1/3}$) and parameters trained with 500,000 iterations of stochastic gradient descent on the MNIST data set, alternating between the wake and sleep error functions. Activities are shown for a single representative run of the wake network and the sleep network. The x axis denotes the number of iterations of the dynamics; the y axis indicates the value of each unit. Units projecting in the same direction from a common variable node are plotted together. Factor node units are not shown, since they are just a linear transformation of the variable node units. The first twenty iterations of the activity evolution are not shown since the initial transients, resulting from the random initialization of the units, can dwarf the final activities. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6.

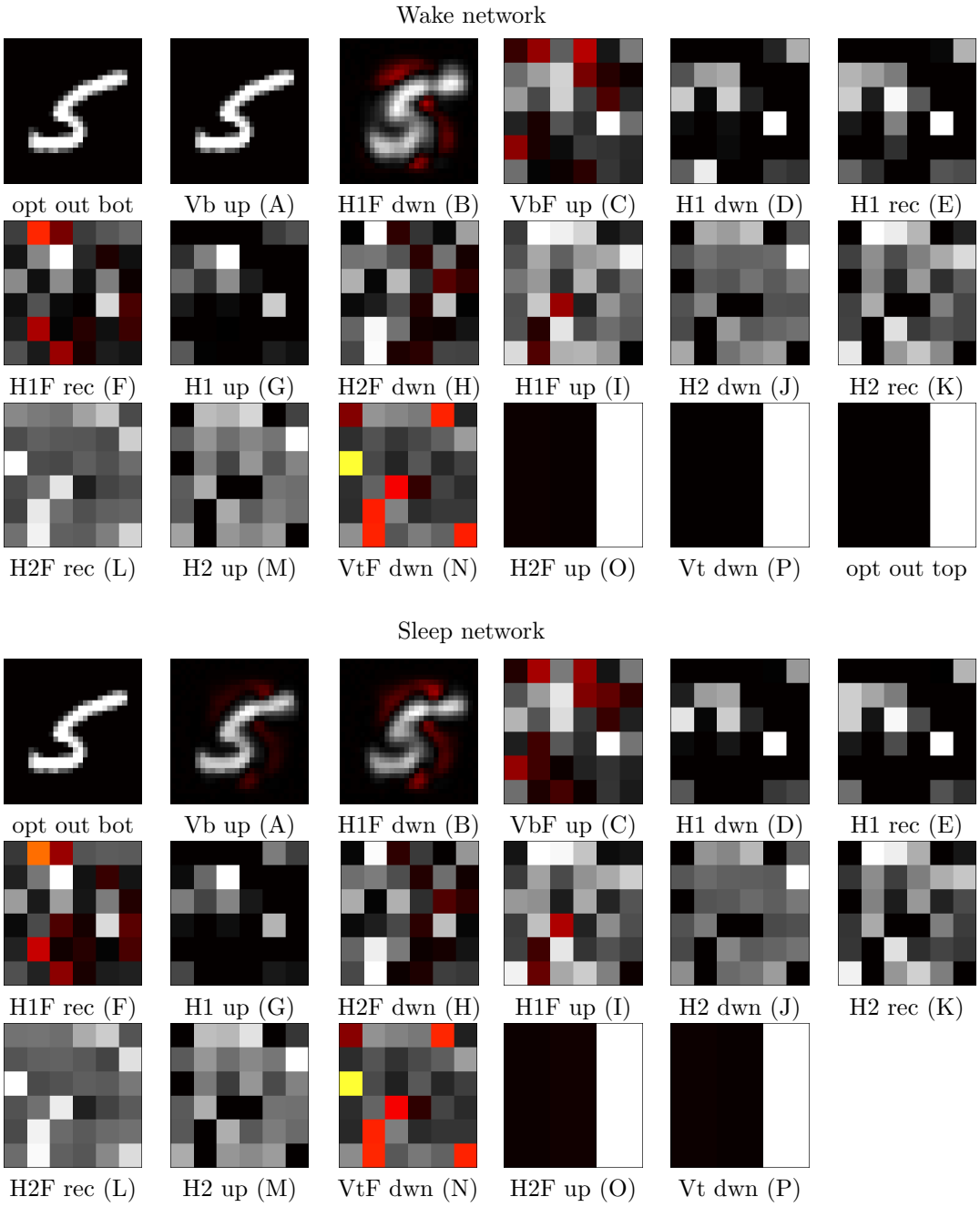


Figure 3.20: Final unit activities $\vec{x}$ after 500 iterations of the network dynamics using a belief-propagating intrinsic gradient network ($h(x) = x^{1/3}$) and parameters trained with 500,000 iterations of stochastic gradient descent on the MNIST data set, alternating between the wake and sleep error functions. Each pixel corresponds to one unit in the indicated layer. Activities are shown for a single representative run of the wake network and the sleep network. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range.

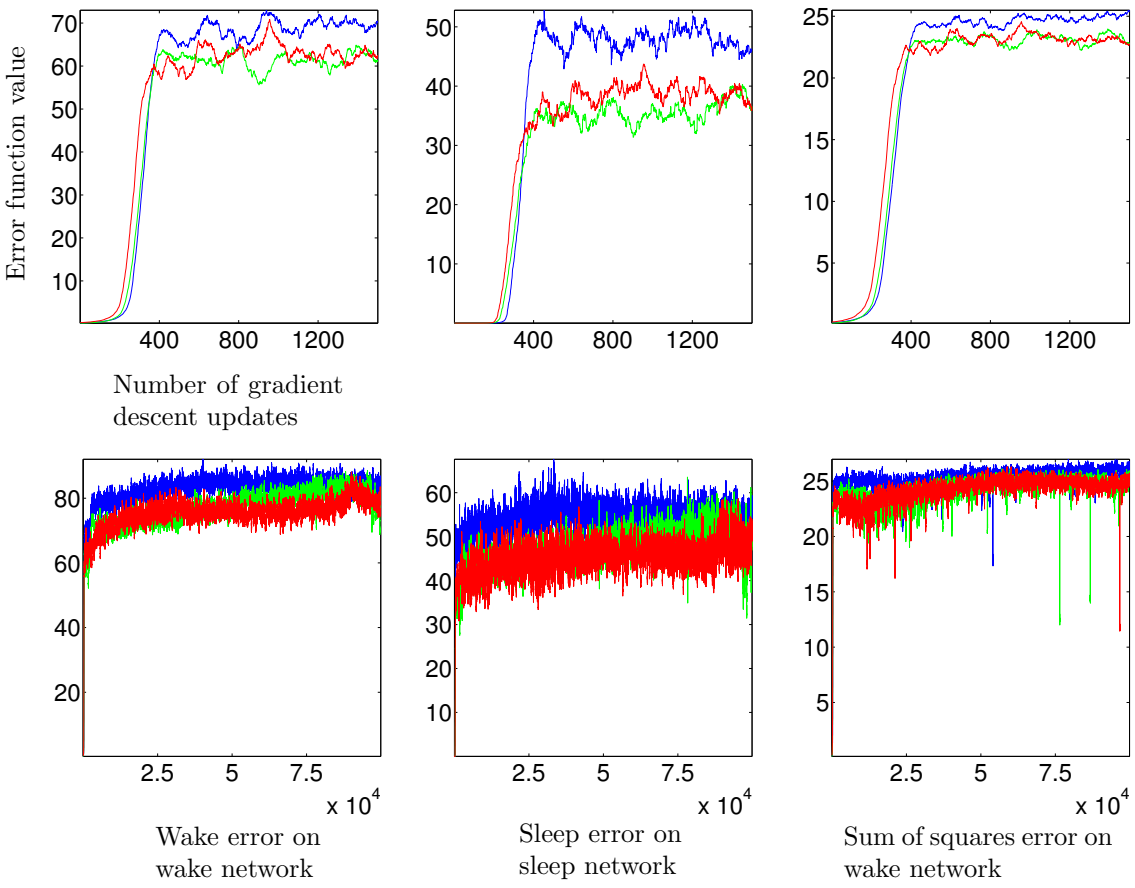


Figure 3.21: Evolution of the error functions due to stochastic gradient descent on the MNIST data set starting from randomly initialized parameters using a belief-propagating intrinsic gradient network ($h(x) = x^{1/3}$). Each separate line plots error functions associated with the elements of a single digit class (i.e., 3, 4, and 5). The x axis indexes the number of times the parameters have been updated in response to each digit class, using both wake and sleep error functions. The y axis indicates the value of the designated error function for the element of the data set (of the appropriate digit class) currently being used for training. Since each index corresponds to a different instance of the appropriate digit class, the value of the error function will jitter even in the absence of training, and the plots are smoothed with a boxcar filter of size 100. The two rows of figures display the evolution of the error function at two different scales, so both the initial and the asymptotic dynamics can be clearly seen.

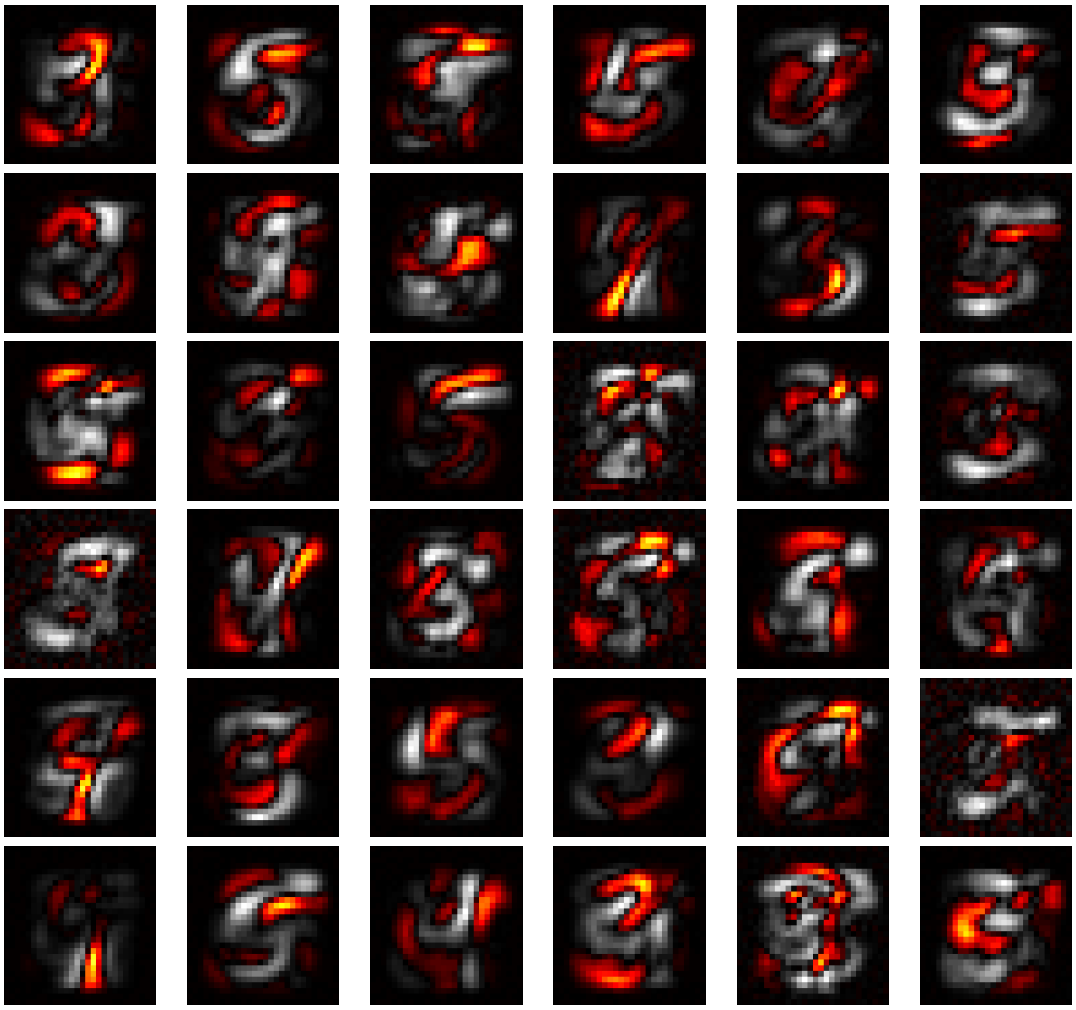


Figure 3.22: Factor matrices connecting the visible units to each hidden unit using a belief-propagating intrinsic gradient network ($h(x) = x^{1/3}$) and parameters trained with 500,000 iterations of stochastic gradient descent on the MNIST data set, alternating between the wake and sleep error functions. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range.

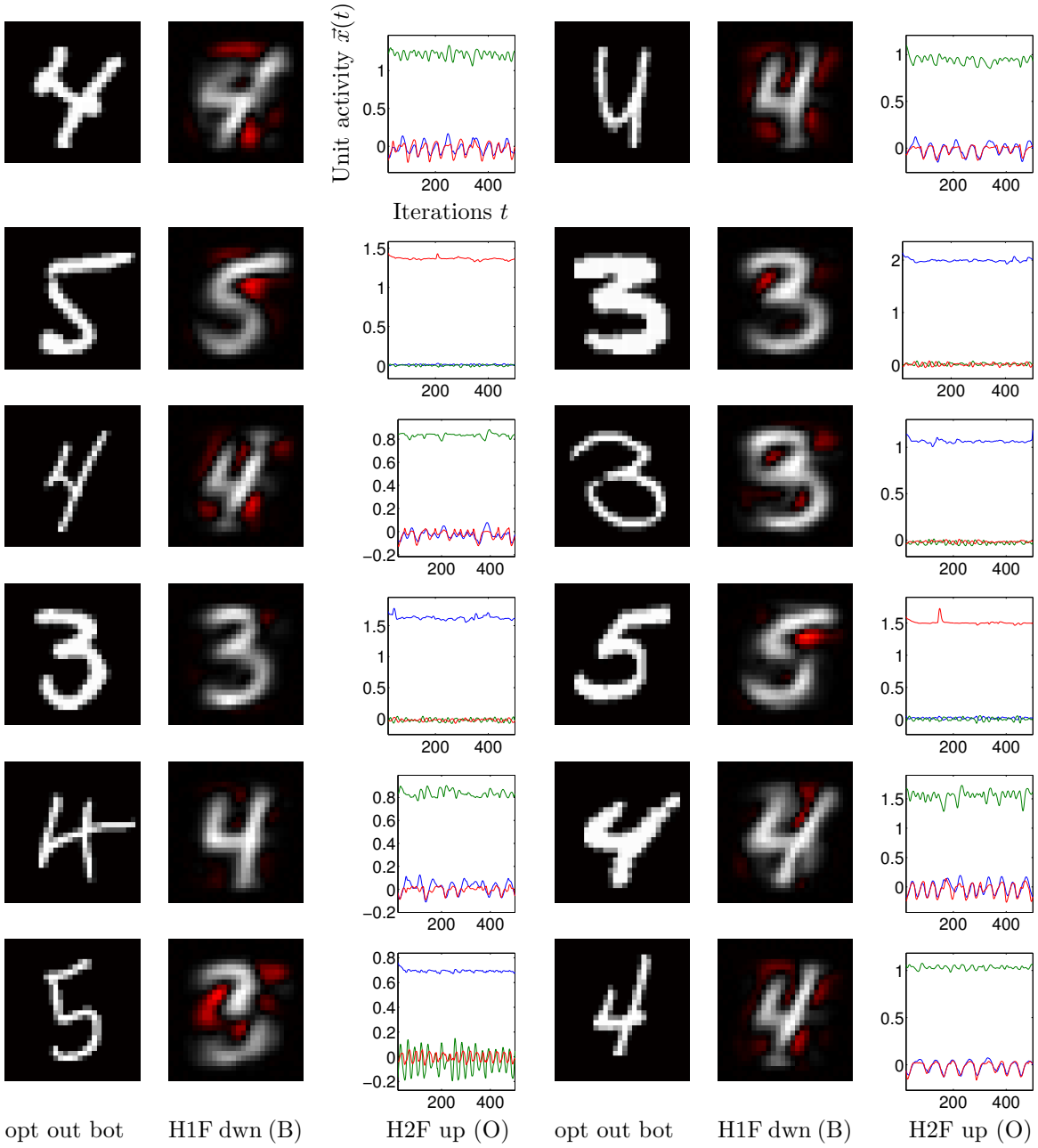


Figure 3.23: Optimal bottom-layer output, actual final bottom-layer output, and evolution of actual top-layer output over 500 iterations of the network dynamics for each element of the MNIST data set using a belief-propagating intrinsic gradient network ($h(x) = x^{1/3}$) and parameters trained with 500,000 iterations of stochastic gradient descent on the MNIST data set, alternating between the wake and sleep error functions. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range. In the plots of the evolution of the top-layer output, blue corresponds to the digit 3, green to the digit 4, and red to the digit 5.

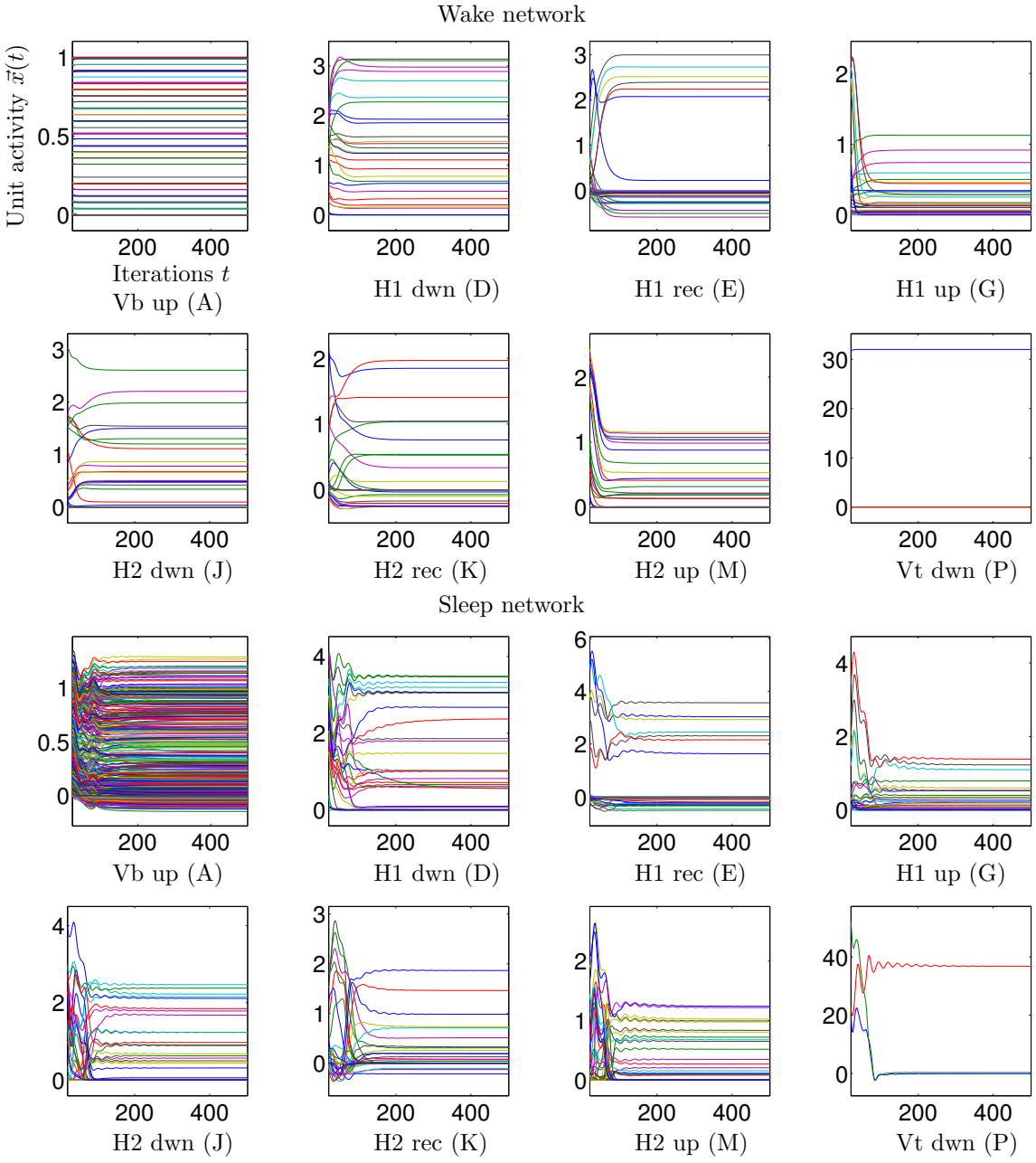


Figure 3.24: Evolution of unit activities $\vec{x}(t)$ using a sigmoid intrinsic gradient network ($h(x) = x/(1+x)$) and parameters trained with 500,000 iterations of stochastic gradient descent on the MNIST data set, alternating between the wake and sleep error functions. Activities are shown for a single representative run of the wake network and the sleep network. The x axis denotes the number of iterations of the dynamics; the y axis indicates the value of each unit. Units projecting in the same direction from a common variable node are plotted together. Factor node units are not shown, since they are just a linear transformation of the variable node units. The first twenty iterations of the activity evolution are not shown since the initial transients, resulting from the random initialization of the units, can dwarf the final activities. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6.

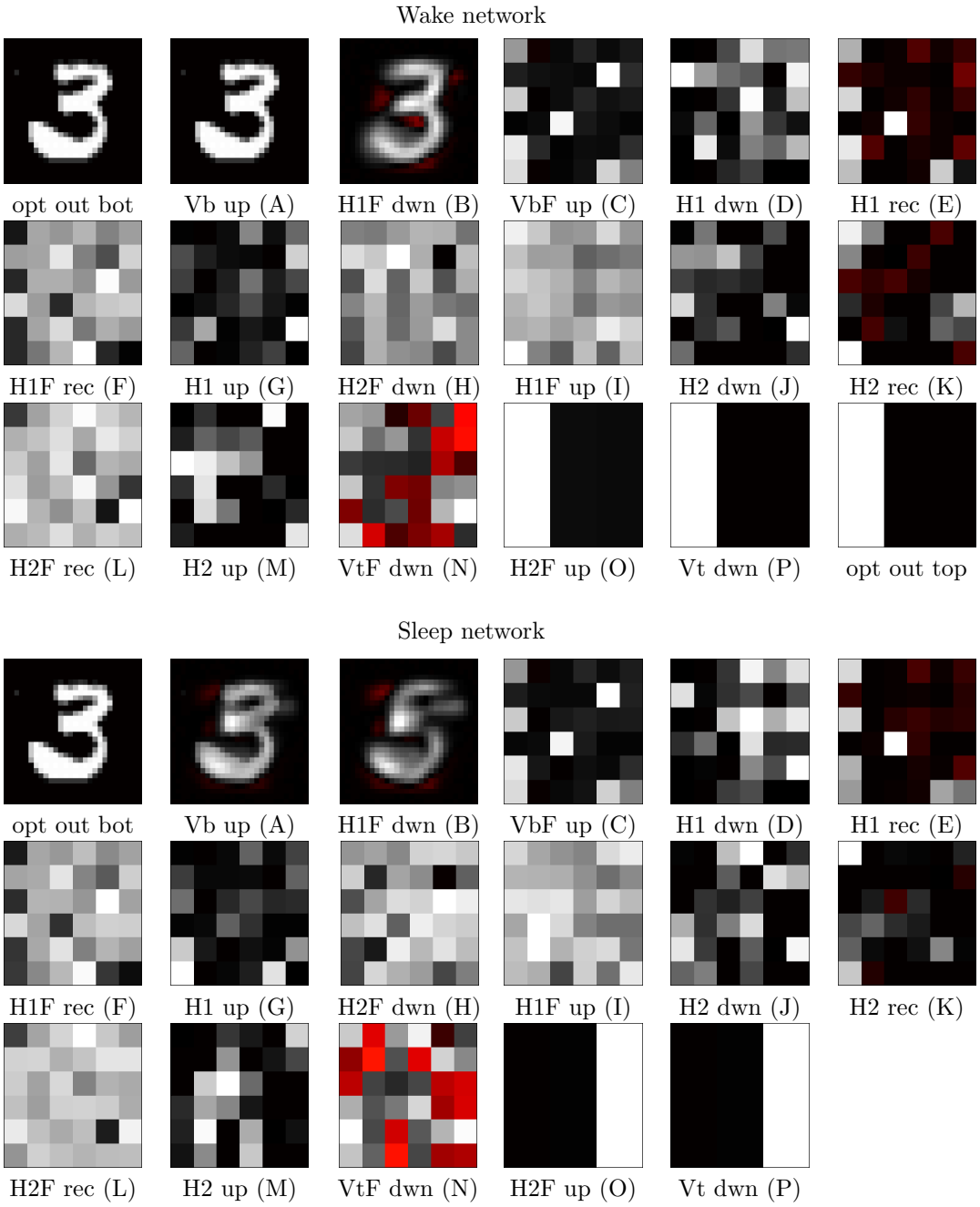


Figure 3.25: Final unit activities $\vec{x}$ after 500 iterations of the network dynamics using a sigmoid intrinsic gradient network ($h(x) = x/(1+x)$) and parameters trained with 500,000 iterations of stochastic gradient descent on the MNIST data set, alternating between the wake and sleep error functions. Each pixel corresponds to one unit in the indicated layer. Activities are shown for a single representative run of the wake network and the sleep network. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range.

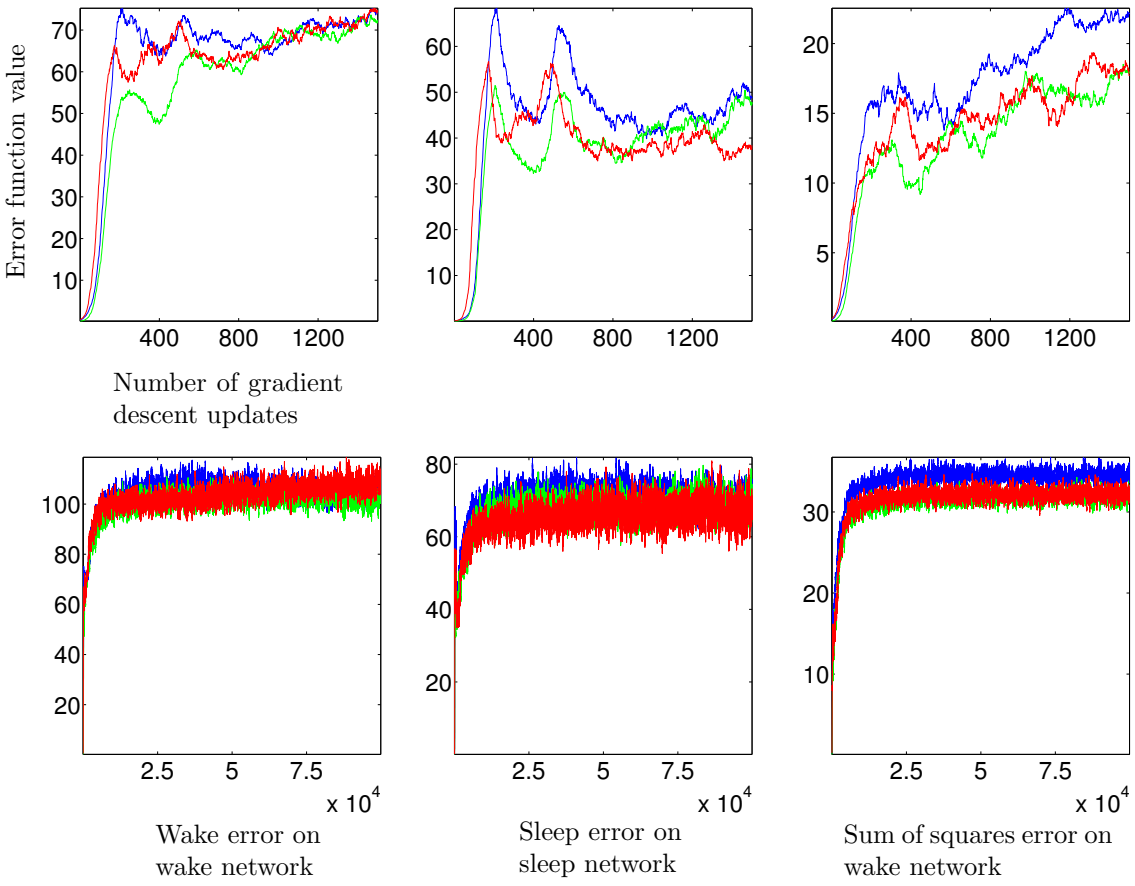


Figure 3.26: Evolution of the error functions due to stochastic gradient descent on the MNIST data set starting from randomly initialized parameters using a sigmoid intrinsic gradient network ($h(x) = x/(1+x)$). Each separate line plots error functions associated with the elements of a single digit class (i.e., 3, 4, and 5). The x axis indexes the number of times the parameters have been updated in response to each digit class, using both wake and sleep error functions. The y axis indicates the value of the designated error function for the element of the data set (of the appropriate digit class) currently being used for training. Since each index corresponds to a different instance of the appropriate digit class, the value of the error function will jitter even in the absence of training, and the plots are smoothed with a boxcar filter of size 100. The two rows of figures display the evolution of the error function at two different scales, so both the initial and the asymptotic dynamics can be clearly seen.

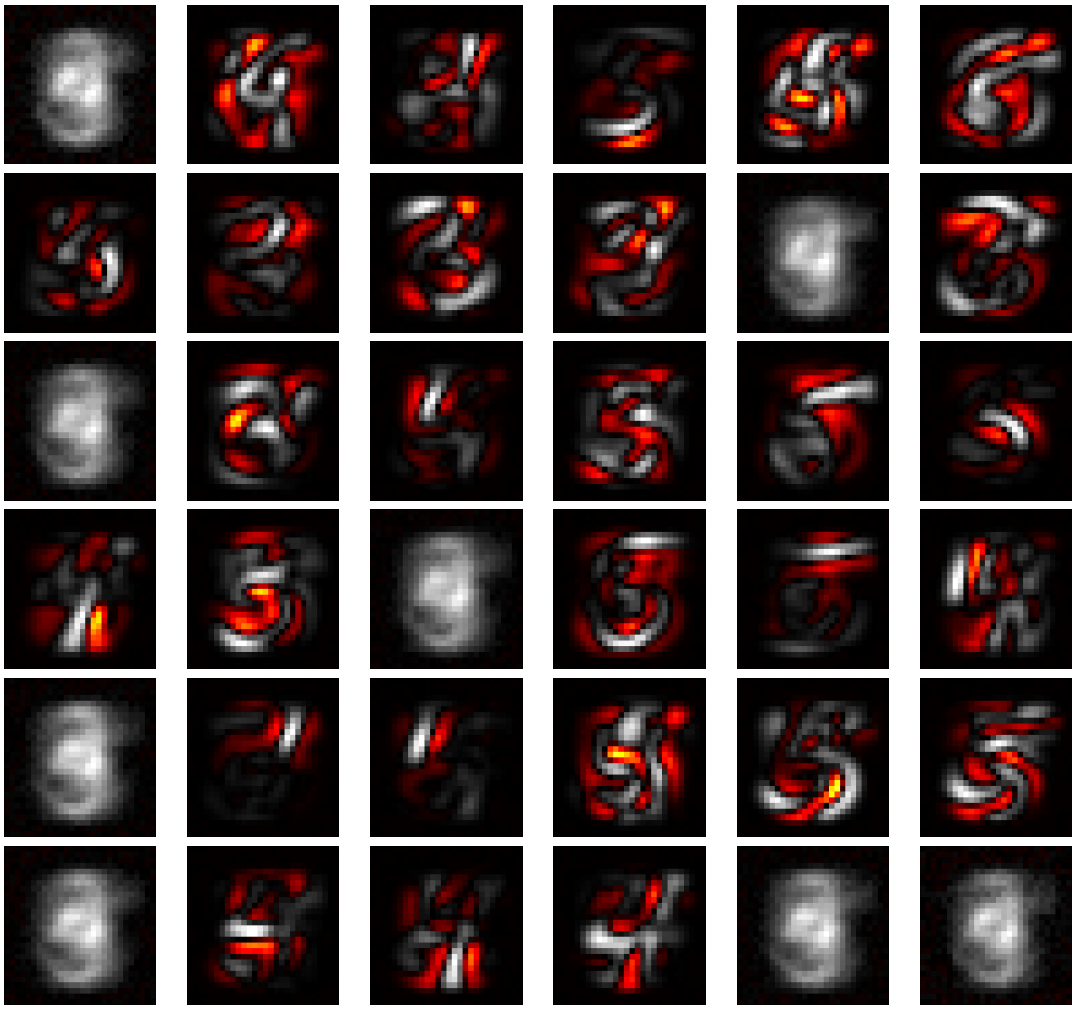


Figure 3.27: Factor matrices connecting the visible units to each hidden unit using a sigmoid intrinsic gradient network ($h(x) = x/(1 + x)$) and parameters trained with 500,000 iterations of stochastic gradient descent on the MNIST data set, alternating between the wake and sleep error functions. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range.

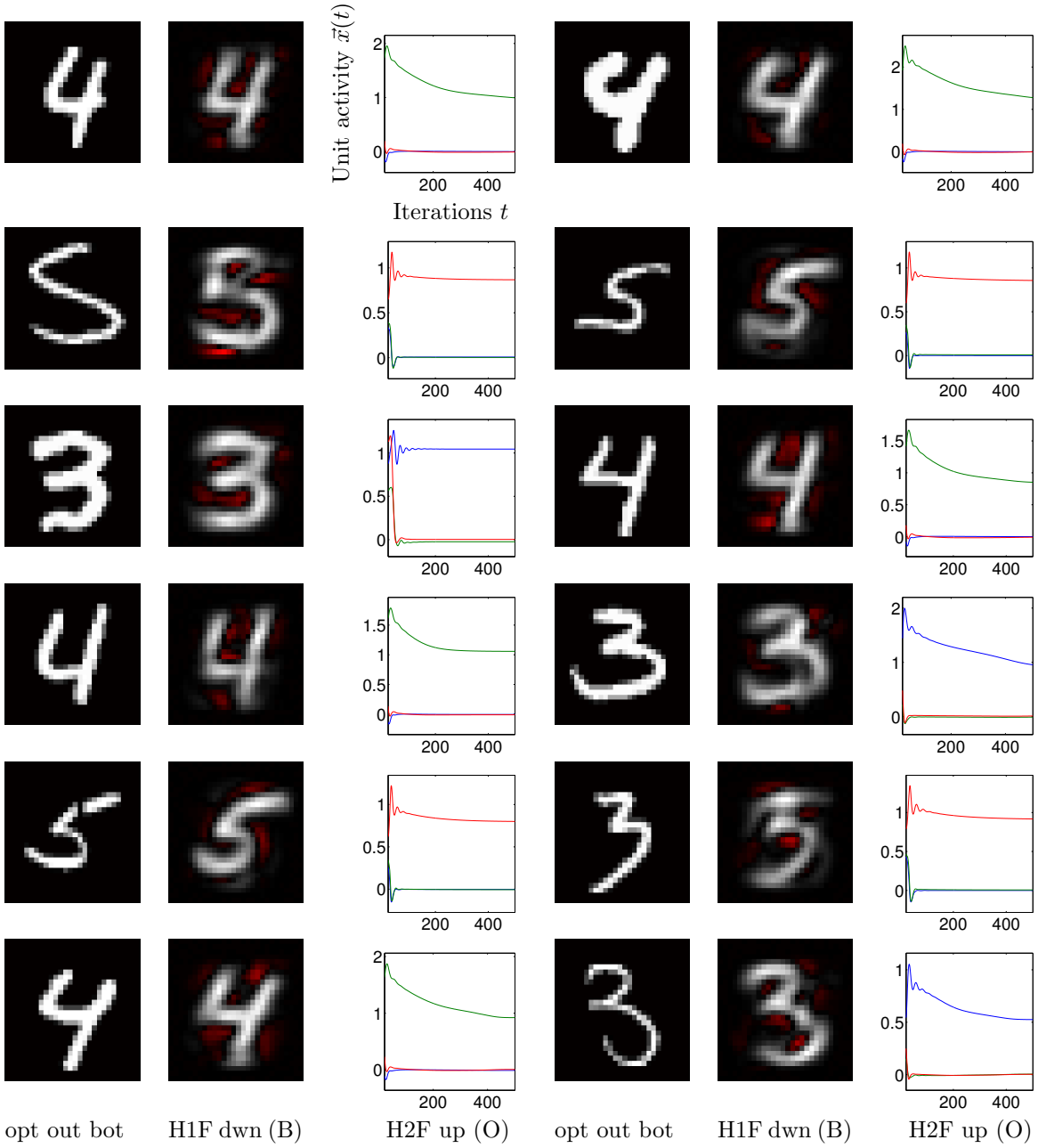


Figure 3.28: Optimal bottom-layer output, actual final bottom-layer output, and evolution of actual top-layer output over 500 iterations of the network dynamics for each element of the MNIST data set using a sigmoid intrinsic gradient network ($h(x) = x/(1 + x)$) and parameters trained with 500,000 iterations of stochastic gradient descent on the MNIST data set, alternating between the wake and sleep error functions. The abbreviations used to identify the various groups of units depicted in each subplot are described in the main text; the letters in parentheses correspond to those in figure 3.6. The colormap, depicted in figure 3.7, is scaled independently for each plot to maximize the dynamic range. In the plots of the evolution of the top-layer output, blue corresponds to the digit 3, green to the digit 4, and red to the digit 5.

Chapter 4

Biological interpretation

Intrinsic gradient networks are an attempt to construct a theory of cortical computation by considering biological constraints on learning and the topology of the cortical network, in addition to the more conventional experimental observations of stimulus-response properties and network dynamics. The hallmark of a successful theory is its ability to make correct predictions beyond the observations on which it was based. We have already shown in chapter 3 that intrinsic gradient networks make plausible computational predictions. In particular, we have demonstrated that intrinsic gradient networks can learn nontrivial input-output mappings, with receptive fields broadly consistent with experimental observations, whereas we only assumed that intrinsic gradient networks are able to compute the gradient of an error function from their intrinsic network activity. In this chapter, we will consider the implications of intrinsic gradient networks for more biological properties.

We begin in section 4.1 by examining the assumptions underlying intrinsic gradient networks, and show that they provide a unifying account of a diverse set of seemingly unrelated biological phenomena. In section 4.2, we describe some additional biological predictions which follow from our solution to the intrinsic gradient equation (2.9) in section 2.3.4. Finally, in section 4.3 we propose a preliminary mapping from a class of intrinsic gradient networks onto the network of cortical neurons, and show that this mapping satisfies four additional biologically motivated constraints.

4.1 The biological plausibility of the assumptions underlying intrinsic gradient networks

In the preceding chapters, we have described a specific structure for intrinsic gradient networks, and implied that this structure has some relation to the cortex. While we have rigorously derived many of the details of intrinsic gradient networks, the entire structure rests on the underlying definition of computation and learning. The biological plausibility of intrinsic gradient networks is therefore dependent on the consistency of these definitions with biological evidence. In the absence

of a constructive specification of cortical computation and learning, which is our ultimate goal, computation and learning are most easily defined implicitly via a characterization of the structure and properties of a system performing the computation. The characterization of computation and learning underlying intrinsic gradient networks specifies the nature of the inputs and outputs, the architecture of the signals within the system, and the distinguishing features of effective learning in terms of these inputs, outputs, and internal signals. In this section, we examine the biological plausibility of this characterization of computation and learning.

Intrinsic gradient networks are based on the assumption that the cortex can be effectively modeled as a discrete network of real-valued units, as opposed to a continuous field of activity. This assumption is motivated by the cellular organization of the brain, and subsumes traditional artificial neural networks. Our intuition is generally guided by the further assumption that each unit in an intrinsic gradient network corresponds to a single neuron, which communicates via its firing rate. In this case, the output of each neuron (and thus the value of each unit) can be characterized by a non-negative real number. However, units in an intrinsic gradient network could also correspond to small populations of neurons such as cortical columns (Mountcastle, 1997; Lansner, 2009), or parts of neurons like single dendritic branches (Poirazi et al., 2003). Similarly, while the value of a unit might parsimoniously correspond to the firing rate of a neuron (or cortical column, or dendritic branch), it could just as easily correspond to the correlation structure within a population of neurons, or even the phase of spiking of a single neuron relative to a global synchronizing signal like the theta LFP rhythm. The requirement that units be real-valued scalars can also be relaxed without altering our results; complex or vector-valued units are equivalent to a population of real scalar-valued units.

The set of units, and the underlying neurons, that are active at an output state can be understood as a Hebbian cell assembly (Hebb, 1949; Harris, 2005), a recurrently connected group of neurons that represent a computational state through their stable coactivation. Such a cell assembly does not consist of a collection of nodes, but rather a subset of units within each node that are activated by a given input. A unit and its associated neuron(s) may be strongly active in many different configurations, driven by different inputs, just like in Hebb's original conception. However, whereas traditional Hebbian learning merely reinforces whatever attractors currently exist, gradient descent in an intrinsic gradient network is guaranteed to modify the attractors to minimize the error function.

In general, we do not assume any particular information encoding scheme or corresponding biophysical mechanisms; rather, we derive constraints on more abstract computational dynamics. These abstract dynamics can then be implemented by a variety of different physical mechanisms. Nevertheless, our assumptions about cortical computation and learning do have implications for experimentally observable features of the brain, and we endeavor to show how the assumptions underlying intrinsic gradient networks can be reconciled with the properties of the brain.

4.1.1 Continuous learning and production of outputs

Models of neural computation should reflect the fact that the brain is able to learn using only its intrinsic signals, and is too densely recurrent to contain training signals that do not affect the outputs (Felleman & Van Essen, 1991; Douglas & Martin, 2004; Malenka & Bear, 2004). This requirement that learning be based upon the intrinsic signals implies that learning can only be performed at a subset of the possible network states. Any model of the cortical neural network defines a mapping from the inputs and parameters to a resulting network state (or set of states), with the output defined as a subset of this resulting network state. Effective learning is then characterized by an error function, which defines the desirability of the output for each input. As the error function varies, so does the optimal output for each input, and thus the optimal parameters which produce these changing outputs via the mapping. Consequently, any effective learning update, such as a gradient-descent step, must vary depending upon the choice of the error function, even if the inputs and thus the network state as a function of the parameters remain constant. If the learning update is correctly calculated for one choice of the error function by a fixed function at a particular network state, it is necessarily calculated incorrectly at that state for other choices of the error function.

The states at which a fixed function of the network state correctly calculates the learning update thus vary with the error function. If a learning update must be selected by a fixed function of the network state, then only a subset of the possible network states can be trained for any given error function.[1] At states where an acceptable learning update is not calculated by the fixed function, the output cannot be efficiently manipulated to minimize the error function, and as a result may be extremely bad. Such unregulated network states should not be used to generate output, lest maladaptive actions result. The subset of network states at which the learning update can be calculated thus constitute the set of acceptable output states, where training helps to ensure that the output is desirable.

Such a potentially sparse mapping from inputs and parameters to acceptable network states and outputs suggests that a network may not associate each input with an acceptable output state. In apparent contrast, the brain seems to produce learned motor outputs continuously for all input trajectories. Traditional models of neural computation usually characterize the dynamics of the network state rather than a subset of output states, implicitly suggesting, if not directly implying, that all network states are actively controlled by learning. Nevertheless, since a learning update is only calculable by a fixed function at the acceptable output states, we instead define intrinsic gradient networks in terms of this subset of acceptable output states. The network may still pass through complicated trajectories to reach an acceptable output state, but learning only directly optimizes the acceptable output states themselves.

[1] The set of fixed points of the output functions $\left\{\vec{x}^* | \vec{F}(\vec{x}^*) = \vec{x}^*\right\}$ is an example of such a subset of the possible network states.

We formalize the assumption that the cortex can be modeled as a discrete network of real-valued units, for which outputs and learning are only produced at a restricted set of acceptable output states, by defining the acceptable output states to be the fixed points of a set of functions $\vec{F}$. That is, the acceptable network output states consist of $\vec{x}^*$ such that $\vec{F}(\vec{x}^*) = \vec{x}^*$. As discussed in section 2.1, this fixed point condition can be reformulated as $\vec{Z}(\vec{x}) = \vec{0}$, where $\vec{0}$ is the vector for which each element is 0, by choosing $\vec{Z}(\vec{x}) = \vec{F}(\vec{x}) - \vec{x}$. There are no *a priori* restrictions on the set of acceptable output states $\left\{\vec{x}^* | \vec{F}(\vec{x}^*) = \vec{x}^*\right\} \subset \mathbb{R}^n$.

While the fixed points of $\vec{F}$ may be relatively easy to recognize, they are generally hard to find. If $\vec{F}$ can be evaluated in polynomial time, the problem of finding acceptable output states is equivalent to the complexity class NP (nondeterministic polynomial time), many of the elements of which are believed to require exponential time to solve. The set of output functions $\vec{F}$ thus does not imply an obvious set of fast network dynamics. In the previous sections, we side-stepped this complication by assuming that the acceptable output states are recognized within a potentially independent set of network dynamics, and used to generate outputs when they are produced. For this reason, we referred to acceptable states simply as output states.

It will often be possible for the network dynamics to continuously remain within the subset of acceptable output states, in which case the entire trajectory consists of trained output states and output is produced continuously. Indeed, trajectories arise naturally, but implicitly, from a set of acceptable output states if there is exactly one acceptable output state for each configuration of the inputs, and the network outputs at time t affect the network inputs at time $t + \Delta t$. Alternatively, the network dynamics may be used to produce outputs continuously, even if they are not always at a fixed point of the output functions and thus subject to training. The resulting untrained outputs are likely to be reasonable if the error function is smooth and the dynamics always remain close to a trained output state. For instance, the fixed points may correspond to saddle points rather than stable attractors of the dynamics, in which case the network falls into a stable heteroclinic channel, passing smoothly from the region around one trained saddle node to the region around another trained saddle node along the unstable separatrices (Rabinovich et al., 2008a,b).

Like motor outputs, perceptual learning occurs continuously in the brain, even without attention, awareness, or explicit reward (Herzog & Fahle, 1997; Watanebe et al., 2001). Consistent with this observation but unlike traditional reinforcement learning or variations thereof specifically adapted for neural networks (Williams, 1992; Sutton & Barto, 1998; Seung, 2003), intrinsic gradient networks do not require a reward signal, which is generally defined only sparsely over time. While a reward signal could modulate the learning step performed using the gradient, for instance scaling the step size of gradient descent, a reward signal is not necessary for most gradient-based training algorithms (Bishop, 2006). As a result, intrinsic gradient networks can learn continuously, so long as they are at an output state.

4.1.2 Biological plausibility of the fixed point

Although the requirement that output states be recognizable when generated by an independent set of network dynamics does not place any direct constraints on the network dynamics themselves, it is difficult to imagine a biologically plausible neural implementation in which the neural dynamics are unconnected to the mechanism that identifies the acceptable output states. If the dynamics were independent of the output functions, then the network could only stumble upon an output state by chance, and would produce outputs very slowly. Moreover, to detect that $\vec{F}(\vec{x}) = \vec{x}$ given dynamics independent of $\vec{F}$, the brain would presumably need an auxiliary network of neurons evaluating the activity of the main network and signaling the occurrence of output states. While loops through the thalamus, basal ganglia, and other subcortical structures have the requisite connection topology, such a signal triggering motor output and learning would be experimentally conspicuous by virtue of its consistent timing, and is not observed in the brain.

These concerns are addressed if the network dynamics themselves converge to a fixed point of the output functions $\vec{F}$. In this case, output production need only be contingent on the stabilization of cortical activity. Neural dynamics could be specifically chosen to converge quickly, and the cessation of neural fluctuations could plausibly be detected locally, without an auxiliary observer network. We thus explore biological evidence supporting the assumption that cortical activity converges to a fixed point, which specifies the motor output and from which the gradient of an error function is calculated.

A variety of experimental evidence supports the more general assumption that the cortex produces output at only a subset of its possible states. In monkeys that have been trained to execute an instructed reach movement after a "go" signal, microstimulation of dorsal premotor cortex shortly after the go signal delays the motion (Churchland & Shenoy, 2007). Likewise, microstimulation shortly before the go signal, but after the reach target is indicated, eliminates the reduction in response time resulting from motor preparation during the post-instruction delay period. Importantly, in both cases, microstimulation does not significantly alter the trajectory of the motion. Similar results have been obtained with transcranial magnetic stimulation in humans (Day et al., 1989). Thus, the cortex appears to be able to detect and correct an artificially-induced perturbation of its state, implying that it has some mechanism for identifying valid output states, and can withhold motor output until such a valid state is reached. The assumption that outputs are produced only at the fixed points of a set of output functions merely formalizes the set of valid output states, without imposing any additional restrictions.

The assumption that output functions are defined in terms of firing rates, so that outputs are only produced when the neuronal firing rates are at a fixed point of the output functions, is equivalent to Churchland's optimal-subspace hypothesis (Churchland et al., 2006). The optimal-subspace hypothesis holds that motor outputs can only be produced after firing rates converge to some sub-

space of the space of possible firing rates across the population of neurons. This instantiation of the fixed point hypothesis has experimentally observable consequences for the evolution of the Fano factor over time. The Fano factor, a measure of variability of the firing rate over repeated trials, is equal to one for the Poisson processes often assumed to underlie single neuron spiking dynamics, so long as the firing rate is consistent between trials (Koch, 1999). To the extent that a Poisson process is a good model of neuronal firing statistics, a Fano factor greater than one is due to between-trial variability in the underlying firing rate, and a decrease in the Fano factor implies an increase in the between-trial consistency of the underlying firing rate.

In monkeys trained to make an instructed reach after a delay, the Fano factor of the firing rate measured in dorsal premotor cortex decreases in response to both the target and the go signal, and is positively correlated with the response time after the go signal (Churchland et al., 2006). Moreover, the Fano factor decreases in response to sensory stimulus in a wide variety of cortical areas in monkeys and cats (Churchland et al., 2010). The implied increase in between-trial consistency of the underlying firing rate in response to sensory stimuli and before motor output suggests that the firing rate is being restricted to a consistent, small subspace, such as the subspace defined by the fixed points of the output functions. Indeed, population firing rate vectors in rat auditory cortex are restricted to a small subspace of the possible configurations, consistent with output states belonging to a small subspace (Luczak et al., 2009).

The fixed points of the output functions $\vec{F}$ are especially easy to identify if, in addition to the output functions being defined in terms of the neuronal firing rates, the fixed points of the network dynamics match those of the output functions. In this case, the network finishes its computation and produces an output when the firing rates converge to a stable configuration. In rat auditory cortex, population firing rates in response to pure tones converge to a fixed point in about 300 milliseconds (Bartho et al., 2009). Similarly, in the visual cortical pathway, transient onset responses give way to a relatively stable sustained response by about 300 milliseconds after stimulus onset (Hegde, 2008). This 300 millisecond interval also matches the interval between saccades (Henderson & Hollingworth, 1998), the duration of the attentional blink (Raymond et al., 1992; Duncan et al., 1994), the interval at which priming has the greatest effect (Zago et al., 2005), and the period over which briefly flashed visual stimuli induce increases in firing rates (Rolls & Tovee, 1994), suggesting that the interval required for cortical sensory activity to converge to a fixed point matches that within which processing of a visual stimulus is largely completed.

The biological plausibility of computations based upon fixed points of neural firing rates is further supported by experiments in both rat sensory cortex and monkey motor cortex which have revealed that, when not driven by a continuously changing salient sensory input, cortical firing rates move through a sequence of quasi-stable states (Radons et al., 1994; Abeles et al., 1995; Seidemann et al., 1996; Jones et al., 2007; Kemere et al., 2008). Specifically, analysis of neural activity using

hidden Markov models has shown that ensembles of six to one hundred neurons remain in states characterized by relatively constant firing rates for hundreds of milliseconds, interspersed with state transitions lasting no more than tens of milliseconds. These quasi-stable states appear to be fixed points of rate-coded neural dynamics, which are subject to perturbations causing transitions between nearby fixed points. The observed transitions between the quasi-stable states would arise naturally if the fixed points were saddle points rather than stable attractors, in which case the network falls into a stable heteroclinic channel (Rabinovich et al., 2008a,b). If the cortex spends most of its time in a fixed point of the output functions corresponding to a quasi-stable state, then the gradient of an intrinsic gradient network can be calculated from the intrinsic activity throughout these quasi-stable intervals.

Such fixed points are unlikely to be merely the result of a feedforward network responding to stable stimuli. Physiological and anatomical evidence indicate that the feedback connections within the cortical network are fast (Girard et al., 2001), functionally relevant (Ringach et al., 1997; Lamme & Roefsema, 2000; Angelucci et al., 2002), and are activated quickly enough to contribute to convergence to a fixed point. The entire visual processing hierarchy responds almost simultaneously, with dendritic activation in IT lagging V1 by only 23 ms (Schroeder et al., 1998; Foxe & Simpson, 2002) and stimulus-driven spiking in V4 following that in V1 by only 38 ms (Schmolesky et al., 1998). The dorsal visual stream responds even more rapidly. Feedback connections thus influence feedforward projections even relatively early in the response to a stimulus.

All of these experimental manifestations of the fixed point are necessarily indirect, since the fixed points only appear to be stable for at most a few hundred milliseconds. In 300 ms, a neuron firing at 20 Hz only spikes six times. It is not possible to detect convergence to a fixed point given so few spikes of a single neuron over a single trial if the neuron is governed by a Poisson process, since the interspike intervals will not be consistent. Even in a recording from a large population of neurons, firing-rate fixed points lasting only a few times the average interspike interval will not be obvious without careful statistical analysis operating over multiple trials, unless groups of neurons with matched firing rates can be identified *a priori*. It is thus unsurprising that fixed points are not a standard electrophysiological observation, even if cortical firing rates are actually converging to fixed points lasting hundreds of milliseconds.

So long as the network state evolves quickly compared to the external inputs, the network of units may always be near a fixed point (and thus an output state), even when the inputs are not constant. The time scale of neural activity is about 10 ms at the scale of a single neuron and 100 ms at the scale of a small network of neurons (Newell, 1990). Transitions between quasi-stable states are executed in tens of milliseconds or less (Seidemann et al., 1996). In contrast, the time scale of interactions with the environment is on the order of 1–10 seconds (Newell, 1990). If neural activity is generally convergent for each input, as is suggested by the decrease of the Fano factor (Churchland

et al., 2006, 2010) and the observed quasi-stable states (Radons et al., 1994; Abeles et al., 1995; Seidemann et al., 1996; Jones et al., 2007; Kemere et al., 2008), and the fixed point changes smoothly or infrequently with respect to the input, such fast neural dynamics might be able to track the fixed point associated with the input. In this case, most learning and output would occur at or near a fixed point.

Nevertheless, the cortical neural network will probably take some time to converge to a fixed point in response to a surprising or novel stimulus, and the fastest behavioral responses may be produced before convergence is achieved. An initial selective cortical response to a stimulus can develop in about 150 ms (Thorpe, et al, 1996; Hung et al., 2005), even when the stimulus is subject to fast masking that impinges upon recurrent computation (VanRullen & Koch, 2002). However, recognition accuracy does not improve with repeated exposure to visual stimuli that are presented only briefly relative to their perceptual difficulty, and then followed by a masking stimulus (Ahissar & Hochstein, 1997; Rubin et al., 1997). When the same stimuli are also presented for more extended durations, or easy stimuli are also presented, accuracy does improve even for briefly presented stimuli.

These experimental results are parsimoniously explained if the fast cortical response present 150 ms after a stimulus constitutes the initial part of a trajectory towards a fixed-point attractor, and nonconvergent cortical responses to brief or difficult stimuli cannot support learning in an underlying intrinsic gradient network. Neural dynamics should converge more reliably in response to the longer-duration and easier stimuli, facilitating initial learning and eventually driving dynamics to converge even for the brief and difficult stimuli. This interpretation is consistent with the observation that neural activity later in the response to a stimulus provides more detailed information than the early activity, which only discriminates between broad stimulus categories (Sugase et al., 1999).

Furthermore, to avoid the failure of training when the neural state is not allowed to develop over time, we might expect that, once convergence begins, the cortex will generally wait for it to complete before responding to other stimuli. This prediction is borne out in the attentional blink phenomenon: full processing of one stimulus blocks processing of subsequent stimuli for about 300 ms (Raymond et al., 1992; Duncan et al., 1994). While perceptual learning occurs in response to unattended stimuli presented along with an attended target (Seitz & Watanabe, 2003), such perceptual learning does not occur if the unattended stimulus and target are presented during the attentional blink (Seitz et al., 2005). Presentation of rewarding stimuli, which would be expected to initiate convergence to a fixed point, can also trigger learning of simultaneously presented but unattended stimuli (Seitz et al., 2009).

There are also a number of strong computational motivations for basing outputs and training upon the fixed points. Experience with recurrent artificial neural networks suggests that it is not difficult to construct recurrent networks that reliably converge to a fixed point (Cohen & Grossberg, 1983; Atiya, 1988; Hirsch, 1989; Matsuoka, 1992). In particular, learning can induce a non-convergent

network to become convergent (Ottaway et al., 1988). Training based upon the converged network state is standard practice in feedforward networks, such as artificial neural networks and acyclic factor graphs with belief propagation, for which the fixed point of network activity corresponds to a single full pass through the network (Bishop, 1995; Kschischang et al., 2001). In Turing machines, if the halting states are understood to leave the tape and the read-write head unchanged, then the halting states are in fact the fixed points of the Turing machine dynamics. A fixed point of a Turing machine thus corresponds to its output after completing a computation, and Turing machines are traditionally programmed so that their converged network states have particular desired properties as a function of their input. The fixed point of more conventional recurrent neural networks can analogously be understood as the output of the network after computation halts. Training the fixed points thus corresponds to training the final output of the network, as opposed to the network's working state in the middle of the computation.

4.1.3 Biological plausibility of gradient descent

Intrinsic gradient networks are based on the assumption that the gradient of an error function, rather than some other training signal, can be calculated at the output states. Since the gradient could be represented and used within the brain in a variety of different ways, it is difficult to derive experimental predictions directly from the assumption that the gradient in particular is calculable. However, experimental evidence and theoretical considerations are largely consistent with the stronger assumption that the cortex calculates the gradient of the error function associated with the current input, and then uses it to perform stochastic gradient descent; that is, in response to each input, the cortex increments each parameter in proportion to the gradient of the associated error function (Bishop, 1995, 2006).

In section 2.1.2, we assumed that the full error function, which defines the desirability of the network in terms of its inputs and outputs, is in fact the average of many component error functions $E(\vec{x}, \vec{w})$. Each component error function corresponds to one set of inputs to the network and the associated desired outputs. This decomposition of the full error function into a sum of components, each dependent on only a single input-output pair, underlies most modern machine learning techniques (Bishop, 2006). In section 3.4, each component error function corresponded to a distinct image and its classification. In a biological context, a single component error function is induced by the sensory stimuli that the brain experiences over a brief interval of time, along with the concomitant motor responses. Each iteration of stochastic gradient descent consists of shifting the parameters by a small amount along the gradient of one such component error function, corresponding to the current input.

Stochastic gradient descent satisfies the desideratum that parameter updates should cumulatively minimize the full error function, but each individual parameter update should only depend on the

current component error function. The gradient is a linear operator, so the average of the gradients of the component error functions is equal to the gradient of the full error function. As a result, the additive conjunction of many small steps along the component gradients is approximately equal to one step along the full gradient. So long as the scaling of the component steps decreases appropriately, stochastic gradient descent is guaranteed to locally minimize a reasonably well-behaved full error function (Bishop, 1995). While stochastic gradient descent is not the only training algorithm that exhibits this concurrence with the segmentation of experience into independent episodes, it is a member of the relatively small class of acceptable algorithms that minimize the full error function via independent operations on the component error functions.

Computational complexity and memory considerations support the use of stochastic gradient descent in the cortex. Even when the full error function is simply a sum of its component error functions, most algorithms for minimizing the full error function require access to all component error functions simultaneously (Bishop, 2006). If the component error functions are only experienced one at time, as in the brain, the components must then be saved or integrated over time to approximate the full error function. A neural implementation of a learning algorithm like stochastic gradient descent can be considerably simpler and require much less memory than most other learning algorithms, since the parameter updates can be performed online in response to individual component error functions, yet nonetheless minimize the full error function (Bottou & Bosquet, 2008).

Online learning like that performed by stochastic gradient descent is also consistent with the experimental observation that learning in the brain only improves performance on a single task (corresponding to a single component error function) at a time, although the cumulative effect of learning is to improve performance on all tasks (reviewed in Tsodyks & Gilbert, 2004). Consecutive training on related tasks generally produces interference rather than reinforcing prior learning (Brashers-Krug et al., 1996; Shadmehr & Brashers-Krug, 1997), and performance generally does not improve over the waking period following a training epoch (reviewed in Stickgold, 2005). It thus appears that learning in the brain only depends on the currently experienced component error function, rather than a buffer or average of recent component error functions.

The learning algorithm used by the brain is also constrained by its behavior on larger temporal scales. While the sensory environment experienced by the brain is often stable in many respects, forces like the slow succession of the seasons and long-distance travel can substantially change the environment. These influences can also return the sensory environment to its original state after an extended perturbation. In order for animals to behave efficiently despite environmental variations, new learning must not disrupt the stability of established responses over time (Karni & Sagi, 1993). This problem is known as the stability-plasticity dilemma (Carpenter & Grossberg, 1988). The knowledge of flowers and swimming accrued during the summer must not destroy the knowledge of snow and ice skating gained during the winter. Similarly, sensory map plasticity is considerably

smaller in adults (who have already accumulated considerable experience) than in juveniles (who are less experienced and have less to lose), suggesting that parameter change is reduced in magnitude (Karmarkar & Dan, 2006). These considerations suggest that parameter changes in response to any single sensory stimulus should be relatively small. The optimal small parameter update corresponds to a step along the gradient. When performed repeatedly in response to different inputs, such a succession of small steps along the gradient is equivalent to stochastic gradient descent.

Finally, in a variety of models, activity learned using gradient descent is consistent with observed neural activity. For instance, when gradient descent is used to train a one-hidden-layer sigmoidal neural network to produce a body-centered representation of visual stimuli from the eye position and eye-centered visual signals experimentally observed in the posterior parietal cortex, it induces activity in the hidden layer similar to that observed in area 7a (Zipser & Andersen, 1988). Likewise, gradient descent induces a sparse network to exhibit activity similar to that in primary visual cortex (V1) when used to train the network to reconstruct its inputs (Olshausen & Field, 1996).

We can thus see that the assumptions underlying intrinsic gradient networks are consistent with experimental observations of the brain. In particular, cortical computation appears to be mediated by the convergence of neuronal firing rates to a fixed point, and the brain must learn via an algorithm like stochastic gradient descent if it is to improve its overall performance through operations based on its current sensory stimulus, while simultaneously preserving prior learning. To the extent that our simple assumptions[2] coherently explain a wide variety of disparate experimental evidence, they constitute a nontrivial theory of neural function in their own right. In section 4.2, we will see that the mathematical implications of these assumptions are consistent with additional experimental evidence, further reinforcing the biological plausibility of intrinsic gradient networks.

4.2 Biological predictions of intrinsic gradient networks

Intrinsic gradient networks satisfying the assumptions of section 2.3.2 encompass a large class of dynamics, as is evident from the fact that each function $h_j^k(x)$ in equation 2.17, indexed by j and k, can be any differentiable scalar function. By carefully choosing the functions $h_j^k(x)$, intrinsic gradient networks can be constructed that exhibit a variety of biological properties. In particular, intrinsic gradient networks that are highly recurrent, trainable in a pseudo-Hebbian manner based upon locally available signals, and composed of independently parameterized units with a minimal *a priori* connection topology will be explored in section 4.3.

Nevertheless, intrinsic gradient networks also have a variety of essential properties that follow from the assumption that the gradient is computable at the fixed points of the output functions,

[2]I.e., that the brain computes and learns by converging to a quasi-stable fixed point of its firing rates, from which it is able to compute the gradient of the component error function defined by the current stimulus and perform stochastic gradient descent

sometimes augmented with the assumptions of section 2.3.2. These basic properties can be used to generate predictions regarding the electrophysiological and anatomical properties that the cortex would exhibit, were it to implement an intrinsic gradient network. In this section, we enumerate some of these predictions, noting in particular where the properties of intrinsic gradient networks have already been experimentally confirmed in the cortex.

4.2.1 Multiplicative combination of bottom-up and top-down signals

As implied by equation 2.17 in section 2.3, in the modular intrinsic gradient networks we've found, each output function consists of a sum of products. We reproduce equation 2.17 here for convenience:

$$\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \nabla \left(c + \sum_k \left[x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot \prod_{j \neq \psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}} \right) \right] \right).$$

Each such product term in an output function is induced by a single term in equation 2.17 of the form $x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot \prod_{j \neq \psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}} \right)$. As a result, each of the factors contributing to one of these product terms is a function of at most two other units in the intrinsic gradient network, one of which is unique to the current factor (x_j in equation 2.17), and the other of which normalizes all factors (except the first) of the product term ($x_{\psi(k)}$ in equation 2.17). Excluding the effect of the common normalization, each output function is thus a sum of products of independent functions of its inputs.

In particular, consider an element of the gradient for which the partial derivative operation leaves only a single non-zero term in the sum over k in equation 2.17, as occurs when there is only a single index $k = k'$ for which $h_j^k(x)$ is non-constant for a chosen index $j = j'$. In this case,

$$\begin{aligned}
&\frac{\partial}{\partial x_{j'}} \left(c + \sum_k \left[x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot \prod_{j \neq \psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}} \right) \right] \right) \\
&= \frac{\partial}{\partial x_{j'}} \left[x_{\psi(k')}^{\frac{D_{\psi(k')}+1}{D_{\psi(k')}}} \cdot \prod_{j \neq \psi(k)} h_j^{k'} \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k')}^{\frac{D_{\psi(k')}+1}{D_{\psi(k')}}}} \right) \right] \\
&= x_{\psi(k')}^{\frac{D_{\psi(k')}+1}{D_{\psi(k')}}} \cdot \left(\left[\frac{d}{dx} h_{j'}^{k'}(x) \right] \cdot \frac{D_{j'}+1}{D_{j'}} \cdot \frac{x_{j'}^{\frac{1}{D_{j'}}}}{x_{\psi(k')}^{\frac{D_{\psi(k')}+1}{D_{\psi(k')}}}} \right) \cdot \prod_{j \neq \psi(k), j'} h_j^{k'} \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k')}^{\frac{D_{\psi(k')}+1}{D_{\psi(k')}}}} \right),
\end{aligned}$$

where in the last line, $\frac{d}{dx} h_{j'}^{k'}(x)$ is evaluated at $x_{j'}^{\frac{D_{j'}+1}{D_{j'}}} / x_{\psi(k')}^{\frac{D_{\psi(k')}+1}{D_{\psi(k')}}}$. If $\mathbf{T}$ is a pairwise permutation matrix, then setting aside the common normalizing input, the corresponding output function is the

product of independent functions of the inputs to the associated unit. For instance, in the hierarchical intrinsic gradient network described in section 3.3, each feedforward and feedback output function is the product of a function of its bottom-up inputs, and a function of its top-down inputs, as is evident in equation 3.21.

If the cortex implements such an intrinsic gradient network with unit activities represented by neuronal firing rates, then equation 2.17 predicts that neural activity after computation is completed should be the product of functions, each dependent on a single cortical area that projects to the current area. These functions can each be characterized by a receptive field. To the extent that the various cortical areas that project to the current cortical area represent disjoint features of the sensory stimulus, such as different stimulus modalities, these receptive fields will be similarly disjoint. The firing rate of neurons within the current cortical area will then result from the product of these receptive fields on distinct sets of stimulus features.

To understand the implications of this prediction of multiplicative interactions, we must first characterize the relationship between receptive fields in the successive areas of a sensory processing hierarchy. In the brain, information can be profitably conceptualized as flowing unidirectionally, despite the presence of recurrent connections which ensure that every signal propagates in all directions. For instance, visual information can be understood as propagating progressively from the retina to the LGN (lateral geniculate nucleus), V1, V2, V4, and IT (inferior temporal cortex) (Lamme & Roefsema, 2000; Serre et al., 2005), even though all of the connections in the ventral visual system are bidirectional (Felleman & Van Essen, 1991). Similarly, attentional and motor information is generally assumed to flow in the opposite direction through each area in a sensory processing hierarchy, from the more abstract and motor-related frontal areas to the more sensory areas in the occipital, temporal, and parietal cortices (Treue, 2001). This view of information transformation and processing is supported by electrophysiological evidence, which shows that as one moves up a sensory processing hierarchy, the representation of the sensory stimulus (as characterized by the receptive field) becomes progressively more abstract, and attentional modulation increases in magnitude (Maunsell & Newsome, 1987; Treue, 2001). As a result, relative to any single area, the downstream areas contain sensory information, and the upstream areas contain attentional and motor information.

Given this progressive transition from sensory to attentional and motor representations throughout a sensory hierarchy, intrinsic gradient networks predict that receptive fields defined in terms of the feedforward sensory signals should be multiplicatively modulated by the feedback attentional and motor signals. This prediction is confirmed by experimental evidence, which suggests that receptive fields throughout both the dorsal and the ventral visual processing hierarchy, defined primarily in terms of visual stimuli, are multiplicatively modulated by both spatial and feature-based attention (McAdams & Maunsell, 1999; Treue & Martinez-Trujillo, 1999; Williford & Maunsell, 2006). The

attention-based gain factor is independent of the stimulus (Martinez-Trujillo & Treue, 2004), as suggested by intrinsic gradient networks.[3]

The multiplicative combination of receptive fields is also observed in the cortex between different stimulus and output modalities. Stimulus or motor-driven firing rates are multiplicatively modulated by gaze direction in the posterior parietal cortex (Andersen et al., 1985; Brotchie et al., 1995), dorsal premotor cortex (Boussaoud et al., 1998), and primary visual cortex (Trotter & Celebrini, 1999). Parietal reach region neurons responsive to the destination of a reach in eye-centered coordinates are multiplicatively modulated by initial hand position (Buneo et al., 2002). Such multiplicatively combined receptive fields are traditionally called gain fields, and can be used to perform coordinate transformations using only a linear combination of the gain field outputs (Salinas & Thier, 2000; Salinas & Abbott, 2001). While it is possible to construct intrinsic gradient networks that do not combine different input sources in a simple, multiplicative manner, it is encouraging that these prominent electrophysiological phenomena are so naturally captured by intrinsic gradient networks.

The consistency of intrinsic gradient networks with gain fields also casts light on the computational capabilities of intrinsic gradient networks. Gain fields facilitate efficient coordinate transformations (Salinas & Abbott, 2001). While intrinsic gradient networks are derived based upon the computational requirement that the gradient be computable from the fixed point of the output functions, they are not subject to any direct restrictions on the computational power of the network. For instance, if we choose $h_j^k(x) = 0$ for all j, k, and x in equation 2.17, as in section 2.3.5, the resulting system is an intrinsic gradient network, but is clearly not capable of nontrivial computation. In contrast, if we choose a hierarchical topology with sigmoidal $h(x)$ like that described in section 3.3, then the variable nodes are approximately a gain field. As a result, the linear operations performed by the factor nodes can carry out a coordinate transformation on the variable nodes.

At the same time, intrinsic gradient networks constitute a biologically plausible manifestation of gain fields, explaining how learning might be implemented in gain fields with many layers. Although the coordinate transformations performed by gain fields can be learned by Hebbian mechanisms when all units can be directly driven by sensory inputs while training, it has thus far been unclear how to train hierarchical networks of gain fields with hidden layers interposed between the input layers (Salinas & Abbott, 2001). Such a hierarchical network corresponds to the hierarchical arrangement of cortical areas (Felleman & Van Essen, 1991), and must be addressed if gain fields are to model the many successive areas of each cortical hierarchy in which gain field activity has been experimentally observed. Intrinsic gradient networks of the sort described in section 4.3 constitute just such a hierarchical network of gain fields, in which the hidden layers can still be trained via Hebbian

[3]Intrinsic gradient networks predict that the attention-based gain factor should be independent of the stimulus to the extent that top-down and bottom-up signals do not influence each other in the network. If, for instance, top-down attentional signals significantly alter the bottom-up sensory signals, the response at any given layer of a hierarchical intrinsic gradient network need not be a simple product of attentional and sensory receptive fields.

learning.

4.2.2 Reconstruction of sensory inputs in primary sensory cortices

As we have seen in sections 2.1.4 and 2.4, the inputs to an intrinsic gradient network are a manifestation of its error function and vice versa. Error functions traditionally used in machine learning, such as the sum of squares error, are minimized when a subset of the output signals match an *a priori* ideal output. When the error function is defined in terms of an ideal output, the inputs of a corresponding intrinsic gradient network are generally a function of these ideal outputs. For instance, using a linear error function, $\vec{c}^{\top} \cdot \vec{x}$, consisting of the inner product of the actual output $\vec{x}$ with an ideal output $\vec{c}$, the inputs to a compatible intrinsic gradient network with a pairwise permutation training function are equal to the ideal values $\vec{c}$ of the output units; with the negative sum of squares error function $-\frac{1}{2} \cdot \sum_i (x_i - c_i)^2$, the inputs are equal to the ideal values plus a regularizing term: $c_i + x_i \cdot \log(x_i)$.

Consequently, when the inputs to an intrinsic gradient network (using this sort of error function and a pairwise permutation training function) are determined by an external environment, the particular error function whose gradient is calculated by an intrinsic gradient network has an ideal output value determined by the input. The minimization of the error function thus induces the network to act like an autoencoder, which reconstructs its inputs on its outputs. If the cortex implements such an intrinsic gradient network, the resulting autoencoder should have experimentally identifiable consequences.

The cortex receives most of its sensory input from the thalamus, and projects back to the thalamus from pyramidal cells in layer 6 (Binzegger et al., 2004; Douglas & Martin, 2004). If the cortex implements an intrinsic gradient network with an error function like the linear or negative sum of squares error and a pairwise permutation training function, activity in primary sensory cortices, especially in layer 6 neurons that project back to the thalamus, should subserve the reconstruction of the thalamic inputs. Such reconstructive responses would be indistinguishable from the receptive field-based activity commonly observed in primary sensory cortices, so long as the transformation from thalamic (or sensory) input to cortical activity is invertible by the projection from cortex to thalamus. For instance, if simple cells in layer 6 of the primary visual cortex are an approximately linear, invertible function of the sensory stimulus, as is often assumed in the experimental literature (Hubel & Wiesel, 1962), then the sensory stimulus can be reconstructed from layer 6 activity via a complementary (inverse) linear function implemented by the axons and synapses projecting from layer 6 of the primary visual cortex to the LGN.

On the other hand, if the instantaneous signal from the environment is momentarily either noisy or incomplete, then the reconstruction of the underlying sensory input will be most accurate when based upon the learned structure of the environment, as in the case of filling-in at the blind

spot (Komatsu, 2006). The primary visual cortex generally receives stimuli over the entire visual field. However, when only one eye is open, no visual information is received within the blind spot of the open eye, corresponding to the region of the retina where the optic nerve leaves the eye. Nevertheless, we are generally not conscious of the blind spot's existence unless a small stimulus is carefully positioned so that it disappears in the blind spot. Even with only one eye open, the blind spot is perceptually "filled in" with the texture of the area surrounding it. Extensive binocular experience has trained the cortex to infer the content of the blind spot from the activity in rest of the visual field.

While the perceptual experience of filling-in could potentially be mediated solely by activity in higher areas of the visual hierarchy, intrinsic gradient networks predict that the reconstruction of the stimulus in the blind spot should also project from the primary visual cortex to the thalamus. So long as an underlying intrinsic gradient network does not learn to treat monocular input as a distinct stimulus paradigm, the autoencoding error function is only minimized when the blind spot is reconstructed by layer 6 of the primary visual cortex when no actual information is available regarding the contents of the blind spot. In this case, the stimulus in the blind spot must be inferred from the surrounding visual field.

Single-cell electrophysiology in the region of the primary visual cortex corresponding to the blind spot is consistent with this prediction. For many neurons with receptive fields falling at least partially within the blind spot, the response to a bar across the blind spot is stronger than the sum of the separate responses to the components of the bar on either side of the blind spot (Fiorani, et al., 1992; Matsumoto & Komatsu, 2005). That is, these neurons respond as if the portion of the receptive field within the blind spot were stimulated by the occluded part of the bar, thereby inferring the contents of the blind spot from the surrounding stimuli. Similar responses are observed in the primary visual cortex when a mask is used to block stimulus in a region of the primary visual cortex which normally receives input from both eyes (Fiorani, et al., 1992), or when a surface is used to completely cover the blind spot (Komatsu et al., 2000), suggesting that stimulus reconstruction in the primary visual cortex is a very general phenomenon. Moreover, this reconstruction phenomenon is observed primarily in granular and infragranular layers, the latter of which projects back to the thalamus; stimulus reconstruction is rarely observed in the supragranular layers of the primary visual cortex, which project to higher regions of the visual processing hierarchy. Further neural evidence for filling-in in primary sensory cortices has been reviewed by Komatsu (2006).

Thalamic afferent activity also differs from the underlying visual scene in certain optical illusions. In particular, when one object occludes another, they meet along a line across which the texture changes. Although the luminance may momentarily be the same on either side of the line in some small regions, this consistency is generally incidental and temporary, and an efficient reconstruction of the visual scene should infer the existence of a line along the most likely occlusion border. Indeed,

an illusory contour is perceptible in stimuli like the Kanizsa figure, which is efficiently interpreted in terms of occlusion and in which the inferred border of the occluding object is visible as a line even when there is no luminance difference across the line (Kanizsa, 1979). As predicted by intrinsic gradient networks, these reconstructed illusory contours are also represented in the activity of the primary visual cortex (Grosof et al., 1993; Sheth et al., 1996; Lee & Nguyen, 2001). Occlusion can also occur in auditory stimulus, and a corresponding reconstruction is apparent in auditory cortex activity, as predicted by intrinsic gradient networks (Petkov et al., 2007).

4.2.3 Complementary, reciprocal connections from V1 to LGN

Given the assumptions of section 2.3.2 and a pairwise permutation matrix $\mathbf{T}$, intrinsic gradient network units generally come in reciprocally connected pairs. As can be seen from equation 2.17, any summand of $g(\vec{x})$ that contains a function of unit x_j will induce a term in the jth component of the gradient, $\frac{\partial}{\partial x_j}$. This term is itself a function of x_j if the corresponding summand is not linear in x_j.[4] Equation 2.17 then implies that this component of the gradient (which is a function of x_j) defines the output function $F_i(\vec{x})$, and thus unit x_i, where the pairwise permutation matrix $\mathbf{T}$ maps x_i to x_j and x_j to x_i. Complementarily, $F_j(\vec{x})$ itself is defined by the partial derivative with respect to x_i, and thus generally depends on x_i. As a result, amongst the units for which x_j is an input, there is generally a unit x_i which projects back to x_j.

The manifestation of this fine-scale reciprocity at a larger scale is discussed in sections 3.2 and 3.3. As an example of this reciprocity, if two variable nodes a and b, as defined in section 3.2, are directly connected, the units of node a that project to node b are arguments of the units of node b that project to node a, and vice versa. This reciprocal connectivity is even more general when viewed in terms of atomic nodes, as in sections 3.2 and 3.3, which consist of all units derived from a single summand of $g(x)$. All connections between atomic nodes are reciprocal given the assumptions of section 2.3.2 and a pairwise permutation matrix $\mathbf{T}$, as can be seen in the examples of chapter 3, regardless of the choices of $h_j^k(x)$.

This reciprocal connectivity is consistent with the connection topology of the cortex, in which connections between cortical areas are almost universally reciprocal (Felleman & Van Essen, 1991). One of the most anatomically and functionally distinct, and best studied, example of such reciprocal connections is that between the primary visual cortex (V1) and the LGN. The primary visual cortex receives inputs from three distinct neural populations in the LGN: magnocellular cells, which are responsive to low contrasts and high temporal frequencies and have significant surround suppres-

[4] As an example of the more general consequences of this property, $\frac{\partial g(\vec{x})}{\partial x_j}$ will not be a function of x_j if for all k, $\psi(k) \neq j$ and $h_j^k \left(\begin{array}{c} \frac{D_j+1}{D_j} \\ x_j \\ \frac{D_{\psi(k)}+1}{D_{\psi(k)}} \\ x_{\psi(k)} \end{array} \right)$ is either not a function of x_j or equal to $x_j \cdot f(x_{\psi(k)})$ for some function f. Perhaps perversely, the factor nodes of sections 3.2 and 3.3 have this unusual form.

sion; parvocellular cells, which are responsive to higher contrasts and lower temporal frequencies, and have less surround suppression; and koniocellular cells, which are intermediate between magnocellular and parvocellular cells in terms of contrast, spatial and temporal frequencies, and which are responsive to the full range of colors (Derrington & Lennie, 1984). As predicted by intrinsic gradient networks satisfying the assumptions of section 2.3.2 and with pairwise permutation training functions, there is also a prominent projection from the primary visual cortex back to the LGN, arising from layer 6. This projection is mediated by three distinct populations of corticogeniculate cells, with response properties matching these three inputs (Grieve & Sillito, 1995; Briggs & Usrey, 2009). These three populations appear to be at least partially physically segregated into a band in upper layer 6 projecting to the parvocellular layers of the LGN, and a band in lower layer 6 projecting to the magnocellular layers of the LGN. Consistent with the input requirements of reciprocal output functions in intrinsic gradient networks, there is a prominent thalamic projection to layer 6 aligned with these bands of the corticogeniculate neurons, in addition to the LGN projection to layer 4.

In an intrinsic gradient network of the form of equation 2.17 with a pairwise permutation training function, each output function is induced by a single element of the gradient, all the summands of which depend upon a single reciprocal input. Since LGN neurons have center-surround receptive fields, if the projection from LGN to V1 is only capable of performing a linear transformation, the reciprocal input signals to the units mediating the corticogeniculate messages will have response properties similar to simple receptive fields (Hubel & Wiesel, 1962). Correspondingly, simple cells (with receptive fields that are linear in the visual input) compose both a majority of layer 6 neurons in V1 in general, and a majority of the corticogeniculate neurons in particular (Swadlow & Weyand, 1987; Grieve & Sillito, 1995; Briggs & Usrey, 2009). All of the corticogeniculate cells projecting to the parvocellular layers of the LGN have simple receptive fields (Briggs & Usrey, 2009), and Swadlow & Weyand (1987) found that all corticogeniculate cells were simple, whereas simple cells are rarely found outside of these two thalamorecipient layers (Martinez et al., 2005).

In contrast, if the training function $\vec{T}$ is linear but not a pairwise permutation, then the output functions projecting from V1 to the LGN would be expected to correspond to complex cells, since they sum over many nonlinear transformations of sums of LGN inputs. Even if $\vec{T}$ is a pairwise permutation matrix, however, the feedforward messages out of V1 share a common feedback message from higher in the visual processing hierarchy but have many different inputs from LGN. As a result, pyramidal neurons in layer 2/3, which project up the cortical hierarchy, are predicted to have complex receptive fields in general, as is observed in the cortex.

While it might seem that strong predictions can be made about neural dynamics based upon the shape of $h(x)$, so long as each output function is the sum of many terms, each of these terms will have a distinct normalization $x_{\psi(k)}$ even if they share a common input (as discussed above). The resulting sum over differently normalized inputs will blur the shape of the function $h(x)$, making strong

predictions difficult. Nevertheless, it is notable that if $h(x)$ is sigmoid, corresponding to the sigmoid relationship generally observed between sensory input and firing rate (Albrecht & Hamilton, 1982), then $h'(x)$ is non-monotonic. Careful observations in the visual system have consistently revealed that the response to increasing stimulus strength can also be non-monotonic, thus resembling both sigmoidal $h(x)$ and peaked $h'(x)$ in turn (Ledgeway et al., 2005; Peng & Van Essen, 2005; Peirce, 2007). Non-monotonic intensity response functions are common in the auditory cortex (Barbour & Wang, 2003).

4.3 A biologically plausible class of intrinsic gradient networks

In section 1, we observed that the cortex is highly recurrent. In particular, there seems to be a directed path from each neuron, cortical column, and area to every other neuron, cortical column, and area, respectively. Intrinsic gradient networks like belief propagation on acyclic factor graphs, discussed in section 3.1, need not have any recurrent loops, although the intrinsic gradient networks constructed in sections 3.2 and 3.3 clearly demonstrate that intrinsic gradient networks can be highly recurrent.

More generally, the cortex has a number of essential characteristics that are not directly implied by the definition of intrinsic gradient networks, but which are consistent with a large subset of intrinsic gradient networks. In this section, we describe four constraints on intrinsic gradient networks inspired by the properties of the cortex. We then identify a class of intrinsic gradient networks, of which the two examples of sections 3.2 and 3.3 are instances, which satisfy these constraints. Finally, we describe a simple, plausible mapping between these intrinsic gradient networks and the network of neurons in the cortex.

4.3.1 Constraint 1: Intrinsic gradient network units are independently parameterized

Neurons are necessarily controlled by independent parameters, since the synapses and other parameter-implementing structures of each neuron are physically distinct. Moreover, aside from gap junctions and retrograde transmission of unconventional neurotransmitters like nitric oxide, each cortical neuron can only produce output mediated by a sequence of action potentials propagated down its axon. The parameters of one neuron influence other neurons primarily through the single signal transmitted through its axonal tree. Neurons partition the parameters and associate them with a unique output, and thus seem to constitute the basic computational units of the brain. It is therefore

natural to identify each neuron with a single unit in an intrinsic gradient network.[5]

We restrict our attention to intrinsic gradient networks in which the dynamics of the units $\vec{x}$, characterized by $\vec{F}(\vec{x}, \vec{w})$, are independently parameterized. In the intrinsic gradient networks defined by equation 2.18, however, single parameters appear repeatedly in multiple output functions. For an intrinsic gradient network of this form to model the brain, either multiple instantiations of each parameter would need to be trained to be equal, or a single representation of each parameter would need to be trained to project to the various units which produce outputs based upon it.

4.3.2 Constraint 2: The network of units is highly recurrent

Anatomical, electrophysiological, and computational evidence suggests that the cortex is highly recurrent. For every long-range axonal projection in the cortex, there appears to be a complementary projection in the opposite direction, both between cortical areas (Felleman & Van Essen, 1991) and between cortical columns in a single area (Angelucci et al., 2002). These reciprocal projections form a network of interconnected tight loops, since the anatomy of the cortical microcircuit does not keep feedback signals segregated from feedforward signals at the scale of a single cortical column (Douglas & Martin, 2004). It is well known that feedforward projections tend to terminate in layer 4, whereas feedback projections tend to terminate in layers 2/3, 5, and 6. However, local projections interconnect these layers; there is a strong loop from layer 4 to 2/3 to 5 to 6 and back to 4 (Binzegger et al., 2004). It is difficult to imagine how feedforward and feedback signals could avoid influencing each other in this architecture.

Moreover, electrophysiological evidence indicates that feedback signals in the brain directly affect feedforward signals and vice versa. Top-down attention and expectations can alter activity throughout the cortical hierarchy (reviewed in Gilbert & Sigman, 2007). Neurons near the bottom of the processing hierarchy in supposedly unimodal sensory areas can exhibit responses to stimuli from other modalities (reviewed in Ghazanfar & Schroeder, 2006). Anatomical and functional evidence even suggest that the non-classical components of receptive fields in the primary visual cortex are driven primarily by feedback from higher areas (Angelucci et al., 2002). These manifestations of feedback signals in the brain are incompatible with the backpropagation signals required to calculate gradients in traditional artificial neural networks (Parker, 1985; Grossberg, 1987; Crick, 1989; Stork, 1989; Zipser & Rumelhart, 1993). The recurrence observed in the brain is also incompatible with belief propagation on acyclic factor graphs, which assumes that the network has no cycles (Kschischang et al., 2001).

Although deterministic recurrent neural networks are not commonly applied to static problems in the machine learning literature (the few examples using recurrent backpropagation include Qian

[5] As we discussed in section 4.1, an intrinsic gradient network unit may be composed of many neurons or correspond to only part of a neuron, to the extent that either groups of neurons or parts of neurons have independent parameters associated with a single output.

& Sejnowski, 1989; Kamimura, 1991; Behrens et al., 1991; Jones, 1992; Pineda, 1995; O'Reilly, 2001), recurrent neural networks are potentially very computationally powerful, and thus an appropriate architecture for the cortex. Consider the problem of determining whether a visual image contains a closed loop, such as a circle. In human vision, closed loops exhibit pop-out in a field of line segments, and even a small break or discontinuity in the loop considerably reduces this effect (Kovács & Julesz, 1993; Pettet et al., 1998). However, a closed loop cannot be identified on the basis of its parts in any simple way. Any part of a closed circle, short of the entire thing, could just as easily be part of an unclosed line segment. A figure which contains a closed loop may have many other unclosed line segments, so merely finding the ends of lines does not indicate the absence of a closed loop. It is thus difficult to construct efficient feedforward networks for detecting closed loops, particularly if the number of hierarchical layers is limited (Bengio & LeCun, 2007). Indeed, the abstraction of this task to the longest circuit problem, in which the task is to determine if an undirected graph has a simple cycle containing some minimum number of nodes, is NP-complete. Nevertheless, even small recurrent networks are able to find closed loops quickly and naturally (e.g., Pettet et al., 1998).

In light of the anatomical, functional, and computational evidence for recurrence in the cortex, we restrict our attention to intrinsic gradient networks in which the units $\vec{x}$ are strongly connected. That is, we require that every pair of units be connected by a directed path (of some length) in both directions.

4.3.3 Constraint 3: The connection topology is not specified in detail *a priori*

The human genome is not large enough to specify exact connections between individual neurons, and there is little detailed order apparent in axonal and dendritic arbors (Binzegger et al., 2005, 2007). While the active synapses of the brain develop a highly structured connection topology, as discussed in section 4.3.4, the neural substrate on which this network forms is much less ordered. Potential synapses, consisting of an axon and dendrite that pass within the distance bridgeable by a dendritic spine, exist with high probability between all cortical excitatory neurons in a column with radius of 100 to 300 μm (Kalisman et al., 2005; Stepanyants et al., 2008). Only about 25% of these potential synapses are realized at any given time (Stepanyants et al., 2002), and most of the physically present synapses are weak (Song et al., 2005). Potential synapses mediated by longer-distance projections are similarly promiscuous, although not as dense. Even in the adult brain, new dendritic spines and axonal boutons regularly extend and form new synapses, and later break synaptic contact and retract (reviewed in Holtmaat & Svoboda, 2009). A synapse between any two neurons with overlapping axonal and dendritic projection regions could plausibly have existed at some point in the past, and could very well exist again at some point in the future, although the

creation or elimination of a synapse occurs over a time scale longer than that required to alter the strength of an existing synapse.

It is thus biologically plausible to assume that all adjacent axons and dendrites, and by extension all nearby cortical excitatory neurons, have a parameterized connection, even if no synapse is currently present or active. Similarly, a parameterized connection exists between a substantial fraction of the pairs of excitatory neurons in more distantly connected columns. The current absence of a synapse effectively constitutes a synapse with no strength and slow dynamics. In order for an intrinsic gradient network to capture this characteristic, the set of parameterized connections to a unit/neuron should include connections from almost all units/neurons that project to its region (as defined by a detailed mapping between the intrinsic gradient network units and the neurons, such as the one we construct below). We therefore restrict our attention to intrinsic gradient networks for which a detailed *a priori* connection topology is not necessary for efficient learning. In particular, we focus on intrinsic gradient networks in which, if two nodes (as defined in section 3.2.3) are connected, their units are effectively connected all-to-all.

4.3.4 Constraint 4: Learning is based upon local signals within small processing assemblies

Since the brain is a physical system, parameters such as the strength of a synapse can only be learned based upon the physical signals that actually reach the physical manifestation of the parameter. In the cortex, neither molecular nor electrical postsynaptic signals seem to be targeted to individual synapses, so the individual synapses of a neuron can only be differentially trained based upon their presynaptic inputs (Waters et al., 2005). Other signals propagate to, and thus likely pertain to, large groups of synapses, such as all the synapses on a dendritic branch. We therefore restrict our attention to intrinsic gradient networks in which the training of the parameters of each unit x_i only depends upon signals directly available to unit x_i. We assume that arguments of $F_i(\vec{x})$ are directly available to x_i, since biologically plausible dynamics for x_i are based upon $F_i(\vec{x})$, as in equations 2.1 and 2.2.

The connections between cortical neurons seem to form clusters, which may correspond to local processing assemblies. There are intraconnected but not interconnected sub-groups of pyramidal neurons and fast-spiking inhibitory neurons spanning layer 2/3 and layer 4 (Yoshimura et al., 2005; Yoshimura & Callaway, 2005), and clustering rules also seem to govern the connections between layer 2/3 and layer 5 (Kampa et al., 2006). Local projections from layer 5 corticothalamic pyramidal neurons are stereotyped based upon anatomy, physiology, and position (Kozloski et al., 2001). Connections between pairs and triads of layer 5 pyramidal neurons tend to be reciprocal, and synapses within such clusters are correlated in strength (Song et al., 2005). A few strong connections seem

to dominate dynamics in a sea of weak connections (Song et al., 2005). Clusters bound by strong connections are also apparent in larger groups of layer 5 pyramidal neurons (Perin et al., 2011). Finally, neurons that form from a single radial glial cell during development are strongly preferentially interconnected (Yu et al., 2009). These experiments suggest that cortical neurons form local, interconnected clusters, which effectively share information. We thus consider intrinsic gradient networks in which the units are further organized into disjoint *processing assemblies*, inspired by cortical columns (Mountcastle, 1997), within which units are interconnected and all outputs are available for training purposes.

Synaptic weight changes in cortical neurons often depend on local dendritic activity as much or more than backpropagating action potentials (reviewed in Kampa et al., 2007), suggesting that local signals within an interconnected cluster of neurons could influence training without necessarily disrupting information transmission between the clusters. Correspondingly, we allow each component of the training function $T_i(\vec{x})$ to depend on units that project to or are members of, and whose signals are thus available to, the processing assembly of which unit x_i is a member. However, we do not require that the arguments of $T_i(\vec{x})$ necessarily be arguments of $F_i(\vec{x})$.

4.3.5 Satisfying constraint 1: Intrinsic gradient networks with independently parameterized units

We can construct a large class of intrinsic gradient networks that satisfy the first constraint by combining a generalization of the factor nodes of section 3.2.1 with a generalization of the variable nodes of section 3.2.2. This set of intrinsic gradient networks will include the examples of sections 3.2 and 3.3, so both figure 3.3 and figure 3.4 can be used as a visual reference for the following discussion. As in sections 3.2 and 3.3, we assume that $\vec{S}(\vec{a},\vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, $\vec{T}(\vec{x},\vec{w}) = \mathbf{T}\cdot\vec{x}$, $\mathbf{T}$ is a pairwise permutation matrix, and use the conservative vector field formulation of section 2.3.4, with $g(\vec{x})$ defined as in equation 2.16. The terms of $g(\vec{x})$ are defined to be of two types, which we call factor-defining terms and variable-defining terms, due to the similarity they bear to the factor nodes and variable nodes constructed in sections 3.2.1 and 3.2.2. We assume dynamics defined by equations 2.1 and 2.18.

We require that the *factor-defining terms* $g_k(\vec{x})$ have a multiplicative parameter and satisfy

$$x_i \cdot \frac{\partial g_k(\vec{x})}{\partial x_i} = x_j \cdot \frac{\partial g_k(\vec{x})}{\partial x_j} \tag{4.1}$$

for all i and j such that $\frac{\partial g_k(\vec{x})}{\partial x_i} \neq 0$ and $\frac{\partial g_k(\vec{x})}{\partial x_j} \neq 0$, while simultaneously satisfying equation 2.16. For instance, they may be of the form $w_{ab}\cdot x_a \cdot x_b$, corresponding to atomic degree-two factor nodes as defined in section 3.2.1; of the form $w_{abc}\cdot x_a^{2/3}\cdot x_b^{2/3}\cdot x_c^{2/3}$, like parameterized versions of the atomic degree-three variable variable nodes of section 3.2.2; or of the form $w_{abcd}\cdot x_a^{1/2}\cdot x_b^{1/2}\cdot x_c^{1/2}\cdot x_d^{1/2}$,

which corresponds to the choice $\psi(k) = a$ and $h_b^k(x) = h_c^k(x) = h_d^k(x) = x^{1/4}$ (and $h_j^k(x) = 1$ for other indices j) in equation 2.18. A finite set of factor-defining terms may share the same parameter.

We only require that the *variable-defining terms* be unparameterized. Given our previous assumptions about the slack function and training function, the kth variable-defining term has the form

$$g_k(\vec{x}) = x_{\psi(k)}^2 \cdot \prod_{j \neq \psi(k)} h_j^k \left(\frac{x_j^2}{x_{\psi(k)}^2} \right) ,$$

where $h_j^k(x)$ is a differentiable function for each index j. This generalizes the atomic degree-three variable nodes in sections 3.2.2 and 3.3 by allowing different functions $h_j^k(x)$ and nodes with more (or fewer) than three connections.

In section 4.3.1, we asserted that each unit should be independently parameterized. The variable-defining terms have no parameters, so their contributions to the units trivially satisfy this constraint. The factor-defining terms are more troublesome. Each factor-defining term contributes a single parameter to many distinct units. The equality between these parameters implied by equation 2.18 is in apparent contradiction to the independence assumed in section 4.3.1.

Fortunately, equality within these parameter groups is established and maintained naturally so long as learning is performed via gradient descent on the error function with weight decay, and the groups of associated parameters follow the same derivative. Gradient descent with weight decay induces exponential decay of the initial conditions. Specifically, if $\frac{dw}{dt} = -\alpha \cdot w + \beta \cdot \frac{dE}{dw}$, then

$$w(t) = w(0) \cdot e^{-\alpha \cdot t} + \beta \cdot e^{-\alpha \cdot t} \cdot \int_0^t e^{\alpha \cdot \tau} \cdot \frac{dE(\tau)}{dw} \cdot d\tau \,,$$

ensuring that two parameters quickly converge to a common value if they follow the same gradient. The shared parameters required by equation 2.18 can thus be implemented by independent, physically distinct parameters so long as the derivatives of the error function with respect to these groups of parameters are equal.

We must therefore evaluate whether the derivatives of the error function with respect to each parameter induced by a single factor-defining term are equal. Substituting equation 2.17 and the chosen values of the training function, ψ_k, and h_j^k into equation 2.7, the derivative of the error

function with respect to some parameter w' of a factor term is

$$\begin{aligned}
\frac{dE(\vec{x}^*)}{dw'} &= (\mathbf{T}\cdot\vec{x}^*)^\top \cdot \boxplus_i \left[\left.\frac{\partial F_i(\vec{x})}{\partial w'}\right|_{\vec{x}^*} \right] \\
&= (\mathbf{T}\cdot\vec{x})^\top \cdot \frac{\partial}{\partial w'} \left[\mathbf{T}^{-1}\cdot(\mathbf{I}+\mathbf{D})^{-1}\cdot\nabla\left(\sum_k g_k(\vec{x})\right)\right] \\
&= \sum_k (\mathbf{T}\cdot\vec{x})^\top \cdot \mathbf{T}^{-1}\cdot(\mathbf{I}+\mathbf{D})^{-1}\cdot\nabla\left(\frac{\partial g_k(\vec{x})}{\partial w'}\right) \\
&= \sum_k \vec{x}^\top\cdot\mathbf{D}^\top\cdot\frac{1}{2}\cdot\boxplus_i\left[\frac{\partial^2 g_k(\vec{x})}{\partial x_i \partial w'}\right] \\
&= \sum_k \frac{1}{2}\cdot\vec{x}^\top\cdot\boxplus_i\left[\frac{\partial^2 g_k(\vec{x})}{\partial x_i \partial w'}\right] \qquad (4.2)
\end{aligned}$$

where $\boxplus_i$ denotes a column vector over the indicated variable, the entries of which are given by the expression in square brackets, and $\mathbf{D} = \mathbf{I}$ since we assume that $\mathbf{T}$ is a pairwise permutation. In all lines after the first, the units are assumed to be evaluated at a fixed point x^*. The assumption that units are independently parameterized is equivalent to setting all but one element of the vector over i in the last two lines equal to zero. The derivatives with respect to the now-independent parameters induced by each single factor-defining term g_m are equal by virtue of equation 4.1, and because $\frac{\partial g_k(\vec{x})}{\partial w'} = 0$ for the other factor-defining terms and the unparameterized variable-defining terms. To follow the gradient of the error function given linked parameters, these derivatives should be scaled as if the parameters were not independent, which will not affect this equality since all corresponding derivatives will be scaled by the same amount.

4.3.6 Satisfying constraints 2 and 3: Highly recurrent intrinsic gradient networks with a minimal *a priori* connection topology

There are many ways to construct strongly connected intrinsic gradient networks with minimal *a priori* connection topologies. We propose a particular framework, consistent with both our solution to the first constraint in section 4.3.5 and our eventual solution to the fourth constraint in section 4.3.7. We assume that the network consists of factor nodes and variable nodes, as in section 4.3.5, that generalize those discussed in sections 3.2 and 3.3. This dichotomy will eventually allow us to minimize the specificity of the associated connection topology. To ensure that these two classes of nodes are well-defined, we assume that the units $\vec{x}$ can thus be partitioned into two groups, *variable units* and *factor units*, such that each summand of $g(\vec{x})$ only contains units from one of the two groups, and $\mathbf{T}$ maps between the two groups. Factor-defining terms are defined exclusively over variable units; variable-defining terms are defined exclusively over factor units. Given these assumptions, equations 2.1 and 2.18 imply that the dynamics of each factor unit are determined entirely by factor-defining terms, which contain variable units. Similarly, the dynamics of each variable unit are determined

entirely by variable-defining terms, which contain factor units.

In addition, we assume that variable-defining terms induce a partition on the set of all factor units. Each factor unit x_j is associated with a single variable-defining term k, such that either $\psi(k) = j$ or $h_j^k(x) \neq 1$; otherwise, $h_j^k(x) = 1$. Each variable-defining term gives rise to many variable units: one for $\psi(k)$ and one for each j such that $h_j^k(x) \neq 1$. However, since the variable-defining terms are defined over disjoint sets of factor units, each variable unit receives a contribution from only one variable-defining term. We continue to refer to the set of variable units whose output functions are determined by a single variable-defining term as an atomic variable node.

Finally, we assume that the variable-defining terms (and their associated atomic variable nodes) can be collected into composite variable nodes, such that factor nodes connect pairs of these composite variable nodes in an all-to-all manner. That is, if a factor-defining term connects an atomic variable node of one composite variable node to an atomic variable node in some other composite variable node, then every pair of atomic variable nodes in these two composite variable nodes is connected by some factor-defining term. The *a priori* connection topology need only specify the pairs of composite variable nodes to be connected, rather than the detailed connectivity between atomic variable nodes or individual units. This is particularly intuitive if the composite variable nodes are spatially delimited, potentially corresponding to something like a cortical column.

The nature of factor nodes ensures that the all-to-all connections between the composite variable nodes are reciprocal. Moreover, inputs to an atomic variable node generally affect all of its outputs. As a result, these networks are strongly connected, and the constraint of section 4.3.2 is satisfied so long as the network does not consist of multiple discrete components. The chosen slack function $\vec{S}$ and training function $\vec{T}$ imply that the intrinsic gradient equation (2.9) is linear in the output functions $\vec{F}$, so sets of output functions can be composed additively to build networks of variable and factor nodes with arbitrary size and topology, as described in section 3.2.3.

4.3.7 Satisfying constraint 4: A mapping onto the network of cortical neurons facilitating pseudo-Hebbian synaptic learning

The locality of learning can only be evaluated in the context of a physical implementation that defines which signals are physically available to each unit. We therefore propose a mapping between the class of intrinsic gradient networks described above and the network of cortical neurons. Specifically, we assume that each atomic variable node (i.e., all of the variable units induced by a single variable-defining term) and all of the factor units that feed into it are implemented by a small group of interconnected neurons, constituting (part of) a processing assembly. The outputs of all the constituent units of a processing assembly are available for training purposes throughout the processing assembly. Such groups of neurons might correspond to cortical columns, and could be

formed by genetically-specified developmental processes independent of the intrinsic gradient network dynamics. These groups of neurons do not explicitly share parameters, although as shown in section 4.3.5, equality between specific pairs of parameters can be established and maintained by biologically plausible training procedures.

The dynamics of such a processing assembly are biologically plausible. We assume that the factor-defining terms are of the form $w_{ij} \cdot x_i \cdot x_j$, so with the dynamics of equation 2.1, each factor unit simply computes the weighted sums of a set of variable units. Specifically, $x_k = \sum_j w_{ij} \cdot x_j$, where $\mathbf{T}$ maps x_i to x_k and vice versa, x_k is a factor unit, and the x_j are all variable units. If the values of the variable units are represented by neuronal firing rates, the factor units could naturally be manifested in the overall membrane potential of dendritic branches. When synapses of varying strength receive presynaptic spikes at fixed rates, low-pass filtering of the synaptic input and passive addition of the membrane potential within the dendritic tree intrinsically perform a weighted sum of the input rates (Koch, 1999). The variable units could then be implemented by a nonlinear transformation of the dendritic membrane potentials into axonal firing rate outputs via active, voltage-dependent mechanisms. Other biophysical implementations of these dynamics are also possible.

Given this mapping, the signals required to compute the derivative of the error function with respect to a parameter are physically available at the synapses that implement the parameter. That is, a group of neurons implementing both the variable units induced by a single variable-defining term and the factor units that drive them, constituting a processing assembly, can indeed compute $\frac{dE(x^*)}{dw_{ij}} = x_i \cdot x_j$ for each parameter w_{ij} of the processing assembly. Moreover, the corresponding gradient ascent update is pseudo-Hebbian.

The parameter w_{ij} corresponds to many synapses, but it only parameterizes the input to two atomic variable nodes. These two projections are symmetric, so we consider only one, without loss of generality. A synapse with parameter w_{ij} (induced by factor-defining term $w_{ij} \cdot x_i \cdot x_j$) performs part of the computation of some factor unit x_k, with $F_k(\vec{x}) = w_{ij} \cdot x_i + \sum_{i'} w_{i'j} \cdot x_{i'}$, such that $\mathbf{T}$ maps j to k and vice versa. Since the variable-defining terms contain disjoint sets of factor units, the factor unit x_k projects to the single atomic variable node induced by the variable-defining term containing x_k. However, this single variable-defining term induces many variable units, which are implemented by the somata of a group of interconnected neurons. The single factor unit x_k with parameter w_{ij} thus corresponds to not just one synapse, but a set of synapses (with weights trained to be equal, as in section 4.3.5, since they follow the same gradient) in the dendritic trees of each of these neurons.

The value of x_i is clearly available to all of these synapses, since it is an input to $F_k(\vec{x})$. Moreover, we previously defined $\mathbf{T}$ to map between x_j and x_k, so the dynamics of x_j are determined by the partial derivative of $g(x)$ with respect to x_k. The single variable-defining term containing x_k thus corresponds to a group of nodes including x_j which are part of a single processing assembly. We have

assumed that the output of each element of a processing assembly is available to all other members for training purposes, so x_j is available postsynaptically at all synapses with weight w_{ij}. The unit x_i is the presynaptic signal to the synapses with parameter w_{ij}. The unit x_j is postsynaptic, in that it is an output of the processing assembly. Equation 4.2 thus implies that $\frac{dE}{dw_{ij}}$ is the product of the presynaptic signal x_i and the postsynaptically available signal x_j, so gradient descent is pseudo-Hebbian.

These somewhat convoluted relationships are easily visualized in figure 3.3. The left and right halves of this figure, split through the central composite factor node, constitute two separate processing assemblies. Factor units x_{c_1} and x_{c_2} of the left-most processing unit receive presynaptic input from units x_{δ_1} and x_{δ_2} of the right-most processing unit, and project to units x_{α_i}, x_{β_i}, and x_{γ_i}. The synaptic projection from each of x_{δ_1} and x_{δ_2} to each of x_{α_i}, x_{β_i}, and x_{γ_i} is mediated by a distinct synapse. The derivative of the error function with respect to each of these synaptic weights is equal to the product of the presynaptic input, either x_{δ_1} or x_{δ_2}, and a postsynaptic signal, either x_{γ_1} or x_{γ_2}, local to the encompassing processing assembly. Figure 3.2 also constitutes a processing assembly, albeit an "atomic" one.

The number of different local outputs from variable units required to train the synapses of any one neuron is equal to the number of components in the associated variable-defining term. For instance, each variable-defining term might have three components, corresponding to bottom-up input, reciprocal connections within a cortical area, and top-down input. Each of these different outputs might terminate on a different part of the dendritic tree in an anatomically defined manner, so that the only synapse-specific input required for training is the natural presynaptic input. Alternatively, the segregation of different input sources may be effected by the layered structure of the cortex.

Chapter 5

Discussion

Cortical computation is a difficult phenomenon to investigate experimentally. Experimental techniques can only probe a miniscule fraction of the cortex at one time, and the requisite preparations, such as slicing and anesthesia, can substantially distort the operation of the cortex. It is thus unsurprising that many experimental studies reach seemingly contradictory conclusions about the detailed properties of the brain. While much remains to be learned about the cortex, a number of characteristic properties have been corroborated by substantial experimental evidence, and should be taken into account in any biologically plausible model of cortical computation. Considerable evidence supports the conclusion that the cortex has a highly recurrent connection topology (Felleman & Van Essen, 1991; Douglas & Martin, 2004). A directed path consisting of some number of successive projections connects each cortical neuron to every other cortical neuron, so each neuron can potentially influence the activity of every other neuron in the cortex. Moreover, while the brain is a singularly effective in learning from its environment, many experiments show that it must carry out this learning using only a simple, local function of the intrinsic signals within its network of neurons (Malenka & Bear, 2004).

In this thesis, we use these reliable cortical properties to derive a model of cortical computation. Intrinsic gradient networks are the novel class of deterministic networks for which the gradient of an error function, defined in terms of the network's activity after convergence, can be calculated simply from the intrinsic network activity. Once such a gradient has been computed, the parameters of the network can easily be trained via gradient descent on the error function. Since a large subset of intrinsic gradient networks are highly recurrent and locally trainable, they constitute a substantial step towards reconciling the power of gradient-based training in neural networks with the observed properties of the cortex.

We derive a characteristic equation for intrinsic gradient networks, the intrinsic gradient equation (2.9), and find a large set of solutions. The intrinsic gradient equation (2.9) serves a role similar to that of the Navier-Stokes equation in fluid dynamics, in that it is a system of nonlinear partial differential equations that characterizes a class of relevant phenomena. Extending this metaphor,

trying to find intrinsic gradient networks that can use the gradient to learn effectively in practice, or that have other biological properties, is like trying to design an efficient airplane wing based upon the Navier-Stokes equation. Both problems are of considerable interest, but of exceptional difficulty because of the analytic intractability of nonlinear partial differential equations. To address this obstacle, we focus primarily on intrinsic gradient networks with an intuitively modular structure. These modular intrinsic gradient networks are characterized by the slack function, $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, and the class of training functions, $\vec{T}(\vec{x}) = \mathbf{T} \cdot \vec{x}$, with $\left(\mathbf{T}^{-1}\right)^{\top} \cdot \mathbf{T}$ diagonal. Within this subset of modular intrinsic gradient networks, the intrinsic gradient equation is linear in the output functions $\vec{F}$. This linearity both allows us to find a large class of analytic solutions to the intrinsic gradient equation, and ensures that the output functions $\vec{F}$ of these solutions can be combined to yield new, larger solutions, including highly recurrent networks.

Since the solution we find for the intrinsic gradient equation (2.9) is a closed-form expression, it is easy to plug values into its parameters and construct particular intrinsic gradient networks, whereas it is difficult to find any solutions for the intrinsic gradient equation directly. In addition to showing that recurrent backpropagation and belief propagation on acyclic factor graphs constitute intrinsic gradient networks, we construct two novel examples of intrinsic gradient networks with dynamics similar to hierarchical neural networks and loopy belief propagation. In contrast to recurrent backpropagation or belief propagation on acyclic factor graphs, both of which exhibit extremely limited recurrence, these novel instances of intrinsic gradient networks are highly recurrent, while retaining the ability to calculate the gradient using a simple, local function of the converged network activity. We further demonstrate that these novel intrinsic gradient networks perform sensibly on real-world problems like handwritten digit recognition.

Finally, we address the detailed relationship between intrinsic gradient networks and the network of cortical neurons. Intrinsic gradient networks use the necessity of simple, effective learning within the highly recurrent cortical network to derive constraints on the computational architecture itself. These constraints are at the algorithmic level, and so do not offer immediately testable predictions about biophysical dynamics, but they do constitute novel and specific targets for new biophysical models. We also derive some predictions about the network-level properties of the cortex, and show that these are consistent with electrophysiological and psychophysical evidence in the brain. To further establish the biological plausibility of intrinsic gradient networks, we suggest a simple mapping from a class of intrinsic gradient networks onto the cortex, within which neurons are independently parameterized, connections are highly recurrent and require minimal *a priori* specification, and synaptic learning is pseudo-Hebbian.

Many authors have suggested frameworks within which the biophysical dynamics of the cortex could be used to perform useful computation (reviewed in Koch, 1999). On the other hand, diverse architectures have been developed to solve tasks like those faced by the brain (Bishop, 2006). How-

ever, the construction of a biologically plausible computational architecture that efficiently learns to perform difficult perceptual and motor planning tasks has remained an open problem, in large part because the gradient-based learning methods that have proven so effective in a machine learning context have been difficult to reconcile with the rampant recurrence of the cortex. Intrinsic gradient networks bridge this gap, identifying a large class of highly recurrent networks in which the gradient is a simple, local function of the converged network activity.

Appendix A

Detailed derivations

A.1 Full derivation of the intrinsic gradient equation

The gradient of an error function defined in terms of a fixed point of a network can be calculated using recurrent backpropagation, even when the fixed point itself is difficult or impossible to evaluate analytically. Pineda (1987) and Almeida (1987) derived recurrent backpropagation specifically for sigmoidal neural networks, for which the output of each unit is a function of the weighted sum of its inputs (reviewed in Pearlmutter, 1995). Their method can be generalized to arbitrary units (Pineda, 1995). Given the definitions of section 2.1, the dynamics have a fixed point at those configurations $\vec{x}^*$ at which

$$\vec{x}^*(\vec{w}) = \vec{F}(\vec{x}^*(\vec{w}), \vec{w}). \tag{A.1}$$

Since $\vec{F}(\vec{x}, \vec{w})$ is dependent on the parameters $\vec{w}$, the fixed points $\vec{x}^*$ are also a function of the parameters, although their dependence on the parameters is generally nontrivial.

From equation A.1, we can derive the gradient of a fixed point with respect to the network parameters. For any parameter w', equation A.1 together with the chain rule implies that

$$\frac{dx_i^*(\vec{w})}{dw'} = \frac{dF_i(\vec{x}^*(\vec{w}), \vec{w})}{dw'} = \left(\sum_j \left. \frac{\partial F_i(\vec{x}, \vec{w})}{\partial x_j} \right|_{\vec{x}^*} \cdot \frac{dx_j^*(\vec{w})}{dw'} \right) + \left(\left. \frac{\partial F_i(\vec{x}, \vec{w})}{\partial w'} \right|_{\vec{x}^*} \right). \tag{A.2}$$

Equation A.2 effectively defines $\frac{dx_i^*(\vec{w})}{dw'}$ as the fixed point of a recurrent update, with the right-hand side of equation A.2 determining the new value of the left-hand side; this recurrence corresponds to the real-time recurrent learning algorithm (Williams & Zipser, 1989; Atiya & Parlos, 2000). To collect the coefficients of the $\frac{dx_i^*(\vec{w})}{dw'}$ terms, we define the matrix

$$\mathbf{L} = \boxplus_{ij} \left[\delta_{ij} - \left. \frac{\partial F_i(\vec{x}, \vec{w})}{\partial x_j} \right|_{\vec{x}^*} \right],$$

where $\boxplus_{ij}$ indicates the matrix for which the element in row i and column j is given by the expression

in square brackets, and δ_{ij} is the Kronecker delta. Moving the first addend of the right-hand side of equation A.2 to the left and switching to matrix notation, equation A.2 becomes

$$\mathbf{L} \cdot \boxplus_j \left[\frac{dx_j^*(\vec{w})}{dw'} \right] = \boxplus_i \left[\frac{\partial F_i(\vec{x}, \vec{w})}{\partial w'} \bigg|_{\vec{x}^*} \right] , \tag{A.3}$$

where $\boxplus_i$ indicates a column vector over the indicated variable, the entries of which are given by the expression in square brackets.

The matrix $\mathbf{L}$ is invertible if and only if none of its eigenvalues are equal to zero. $\mathbf{L}$ is proportional to the linearization of the right-hand side of equation 2.2, so if a fixed point of those dynamics is stable, then $\mathbf{L}$ has real parts of all eigenvalues less than or equal to zero. However, using the dynamics of equation 2.2, a fixed point may be neutrally stable or attracting even if the eigenvalue with the largest real part is equal to zero, in which case $\mathbf{L}$ is not invertible. The stability of the dynamics of equation 2.2 thus does not necessarily imply that $\mathbf{L}$ is invertible, although it is strongly suggestive. Moreover, entirely different dynamics might be used so long as the fixed points match those of $\vec{F}(\vec{x})$. Regardless, we will assume for the moment that $\mathbf{L}$ is invertible at the fixed points to motivate the following derivation. At the end of this section, we will demonstrate that the conclusion of the derivation remains correct even when $\mathbf{L}$ is singular.

Left-multiplying both sides of equation A.3 by the inverse of $\mathbf{L}$, we can solve for the derivatives of the fixed point:

$$\boxplus_j \left[\frac{dx_j^*(\vec{w})}{dw'} \right] = \mathbf{L}^{-1} \cdot \boxplus_i \left[\frac{\partial F_i(\vec{x}, \vec{w})}{\partial w'} \bigg|_{\vec{x}^*} \right] .$$

Thus, $\mathbf{L}^{-1}$ can be understood as the matrix which tells us how the fixed point will react to a small change in w', in terms of how just a single iteration of $\vec{F}$ would react. For any error function $E(\vec{x}, \vec{w})$ and internal parameter w' such that $E(\vec{x}, \vec{w})$ does not depend directly on w', when evaluated at the fixed point, the chain rule in conjunction with the equations above yields

$$\frac{dE(\vec{x}^*(\vec{w}))}{dw'} = \sum_j \frac{\partial E(\vec{x})}{\partial x_j} \bigg|_{\vec{x}^*} \cdot \frac{d\vec{x}_j^*(\vec{w})}{dw'} = \boxplus_j \left[\frac{\partial E(\vec{x})}{\partial x_j} \right]^\top \cdot \mathbf{L}^{-1} \cdot \boxplus_i \left[\frac{\partial F_i(\vec{x}, \vec{w})}{\partial w'} \right] . \tag{A.4}$$

On the right-hand side, and in the following through the end of this section, all partial derivatives are evaluated at the fixed point.

The difficult computation required in equation A.4 is the inversion of $\mathbf{L}$. The asymptotically fastest known matrix inversion algorithms have time complexity $O(n^{2.376})$ for an $n \times n$ matrix (Coppersmith & Winograd, 1990). The number of rows and columns in $\mathbf{L}$ is equal to number of units x_i, and may be as large as 10^{10} in the human cortex if each unit is implemented by a single neuron (Koch, 1999), so the explicit inversion of $\mathbf{L}$ is likely to be extremely time-consuming. In addition to the computational difficulty of inverting $\mathbf{L}$, we wish to calculate the gradient of the error function $\frac{dE}{d\vec{w}}$ using only the intrinsic signals $\vec{x}$ in the network. However, there may be many more

parameters w_i than units in the network, so the gradient $\frac{dE}{d\vec{w}}$ may have many more components than there are signals to calculate it.

We thus split equation A.4 into two parts: one which effectively inverts $\mathbf{L}$ using only a small number of signals, and a second which uses this computation to calculate $\frac{dE}{d\vec{w}}$ locally. Specifically, we define a vector $\vec{T}(\vec{x}^*, \vec{w})$ to encapsulate part of the right-hand side:

$$\vec{T}^\top(\vec{x}^*, \vec{w}) = \coprod_j \left[\frac{\partial E(\vec{x})}{\partial x_j} \right]^\top \cdot \mathbf{L}^{-1}, \tag{A.5}$$

where $\vec{T}^\top(\vec{x}^*, \vec{w})$ and $\coprod_j \left[\frac{\partial E(x)}{\partial x_j} \right]^\top$ are row vectors, and equation A.5 only defines $\vec{T}$ at the fixed points $\vec{x}^*$ of $\vec{F}$. As we shall see below, this $\vec{T}$ can be calculated at the fixed points of $\vec{F}$ via a method roughly equivalent to backpropagation-through-time (Rumelhart et al., 1986). Using the definition of $\vec{T}$ from equation A.5, equation A.4 becomes

$$\frac{dE(\vec{x}^*(w))}{dw'} = \vec{T}^\top(\vec{x}^*, \vec{w}) \cdot \coprod_i \left[\left. \frac{\partial F_i(\vec{x}, \vec{w})}{\partial w'} \right|_{\vec{x}^*} \right]. \tag{A.6}$$

Since $\vec{T}(\vec{x}^*, \vec{w})$ performs the difficult part of the calculation of the gradient which is used to train the parameters, we refer to it as the *training function*. As with $\vec{F}(\vec{x})$, we sometimes write $\vec{T}(\vec{x})$ and omit explicit mention of the dependence on the parameters to avoid excessively cluttered notation.

The calculation of the training function $\vec{T}$ specified by equation A.5 requires the inversion of $\mathbf{L}$, which is an extremely complicated and non-local computation for most matrices $\mathbf{L}$. In contrast, we want $\vec{T}$ to be computable in a simple way with a small set of local signals. $\mathbf{L}$ is defined in terms of $\vec{F}$, so we might hope that by choosing $\vec{F}$ carefully, we would be able to make the calculation of $\vec{T}$ simple and local. More specifically, we would like to be able to solve for $\vec{F}$ given $\vec{T}$, so that we can specify a simple form for $\vec{T}$ and then infer what $\vec{F}$ are compatible with that $\vec{T}$. As the first step towards this goal, we write equation A.5 in terms of $\vec{F}$; we right-multiply by $\mathbf{L}$, apply the definition of $\mathbf{L}$, consider just the jth element on each side, and rearrange to find:

$$\begin{aligned}
\vec{T}^\top(\vec{x}^*, \vec{w}) \cdot \mathbf{L} &= \coprod_j \left[\frac{\partial E(\vec{x})}{\partial x_j} \right]^\top \\
\vec{T}^\top(\vec{x}^*, \vec{w}) \cdot \coprod_{i,j} \left[\delta_{ij} - \frac{\partial F_i(\vec{x}, \vec{w})}{\partial x_j} \right] &= \coprod_j \left[\frac{\partial E(\vec{x})}{\partial x_j} \right]^\top \\
T_j(\vec{x}^*, \vec{w}) - \sum_i \frac{\partial F_i(\vec{x}, \vec{w})}{\partial x_j} \cdot T_i(\vec{x}^*, \vec{w}) &= \frac{\partial E(\vec{x})}{\partial x_j} \\
T_j(\vec{x}^*, \vec{w}) &= \frac{\partial E(\vec{x})}{\partial x_j} + \sum_i \frac{\partial F_i(\vec{x})}{\partial x_j} \cdot T_i(\vec{x}^*, \vec{w}).
\end{aligned} \tag{A.7}$$

Equation A.7 is superficially very similar to equation A.2, with T_i playing the role of $\frac{dx_j^*(\vec{w})}{dw'}$, but the index of $\vec{T}$ on the right-hand side of equation A.7 matches the numerator of the partial derivative,

whereas the index of $\frac{dx_j^*(\vec{w})}{dw'}$ on the right-hand side of equation A.2 matches the denominator of the partial derivative. We obtain equation 2.8 by writing equation A.7 in matrix notation.

As a result of our prescient decomposition of equation A.4 into two parts, the elements of $\vec{T}$ can be put into bijection with those of $\vec{x}$, whereas there are potentially many more parameters than output functions. Other decompositions are also possible. For instance, we might attempt to compute $\boxplus_j\left[\frac{d\vec{x}_j^*(\vec{w})}{dw'}\right] = \mathbf{L}^{-1} \cdot \boxplus_i\left[\frac{\partial F_i(\vec{x})}{\partial w'}\right]$ directly. However, $\boxplus_j\left[\frac{d\vec{x}_j^*(\vec{w})}{dw'}\right]$ is different for each w', so it is difficult to use $\vec{x}$ to calculate these terms for all w'. Real-time recurrent learning, which uses this decomposition, must generally be run for each parameter separately (Williams & Zipser, 1989; Atiya & Parlos, 2000).

Even if $\mathbf{L}$ is not invertible, equation A.7 is equivalent to backpropagation-through-time on the error function $E(\vec{x})$, subject to the condition $\vec{F}(\vec{x}^*) = \vec{x}^*$ (Rumelhart et al., 1986). That is, if equation A.7 is viewed as a dynamic update, with the right-hand side at time t determining the value of the left-hand side at time $t+1$, it manifests an unrolling of the chain rule applied to the gradient of the error function. Since x^* is a fixed point, equations A.6 and A.7 compute the gradient of the error function, regardless of the invertibility of $\mathbf{L}$.

A.1.1 Necessity condition

We generally assume a particular form for the training function $\vec{T}$, and derive compatible output functions $\vec{F}$. This approach will only be successful if simple functions $\vec{T}(\vec{x}, \vec{w})$ exist which can be used to calculate the gradient of an error function at the fixed point of the dynamics. Fortunately, simple and efficiently computable $\vec{T}$ satisfying equation A.5 must exist so long as the gradient of the error function is simply and efficiently computable, and $\vec{F}$ can be divided into independently parameterized groups. We can rewrite equation A.4 as a single matrix equation over all parameters:

$$\begin{aligned}\left(\boxplus_k\left[\frac{dE(\vec{x}^*(\vec{w}))}{dw_k}\right]\right)^\top &= \left(\boxplus_k\left[\sum_j \frac{\partial E(\vec{x})}{\partial x_j} \cdot \frac{d\vec{x}_j^*(\vec{w})}{dw_k}\right]\right)^\top \\ &= \boxplus_j\left[\frac{\partial E(\vec{x})}{\partial x_j}\right]^\top \cdot \mathbf{L}^{-1} \cdot \boxplus_{ik}\left[\frac{\partial F_i(\vec{x}, \vec{w})}{\partial w_k}\right].\end{aligned}$$

Applying the definition of $\vec{T}$ from equation A.5, we then find

$$\begin{aligned}\left(\boxplus_k\left[\frac{dE(\vec{x}^*(w))}{dw_k}\right]\right)^\top &= \vec{T}^\top(\vec{x}^*, \vec{w}) \cdot \boxplus_{ik}\left[\frac{\partial F_i(\vec{x}, \vec{w})}{\partial w_k}\right] \\ \left(\boxplus_k\left[\frac{dE(\vec{x}^*(w))}{dw_k}\right]\right)^\top \cdot \left(\boxplus_{ik}\left[\frac{\partial F_i(\vec{x}, \vec{w})}{\partial w_k}\right]\right)^+ &= \vec{T}^\top(\vec{x}^*, \vec{w}) \cdot \boxplus_{ik}\left[\frac{\partial F_i(\vec{x}, \vec{w})}{\partial w_k}\right] \cdot \left(\boxplus_{ik}\left[\frac{\partial F_i(\vec{x}, \vec{w})}{\partial w_k}\right]\right)^+ \\ &= \vec{T}^\top(\vec{x}^*, \vec{w})\end{aligned}$$

where $\mathbf{N}^+$ is the Moore-Penrose pseudoinverse of matrix $\mathbf{N}$, and the last equation follows so long as the rows of $\boxplus_{ik}\left[\frac{\partial F_i(\vec{x},\vec{w})}{\partial w_k}\right]$ are linearly independent (or, equivalently, the columns span the full n-dimensional Euclidean space, where n is the number of units), which should generally be the case assuming that there are more parameters w_k than output functions F_i. Moreover, if $\vec{F}$ can be partitioned into groups defined in terms of disjoint groups of parameters, then the units and parameters can be ordered such that $\boxplus_{ik}\left[\frac{\partial F_i(\vec{x},\vec{w})}{\partial w_k}\right]$ is block diagonal, as is its pseudoinverse. Each element of the pseudoinverse only depends upon members of the same block, so if the gradient of the error function is simply and efficiently computable at the fixed points of the dynamics, then $\vec{T}$ is the weighted sum of a small number of these simple, efficient computations. That is, a $\vec{T}$ satisfying equations A.6 and A.7 can necessarily be calculated by a simple, efficient function if $\frac{dE}{d\vec{w}}$ can be.

A.1.2 Alternative derivation based on Lagrange multipliers

We can also derive the intrinsic gradient equation (2.9) using an approach analogous to that of LeCun (1988). Consider the Hamiltonian

$$\mathcal{H}(\vec{x},\vec{w},\vec{B}) = E(\vec{x}) + \vec{B}\cdot(\vec{F}(\vec{x},\vec{w}) - \vec{x})\,.$$

At the fixed point, $\vec{F}(\vec{x},\vec{w}) - \vec{x} = 0$, so $\mathcal{H}(\vec{x},\vec{w},\vec{B}) = E(\vec{x})$ regardless of the particular values of $\vec{w}$ and $\vec{B}$. We restrict our attention to such fixed points, which we previously identified as the outputs of the network. That is, we assume that $\vec{x}$ is a function of $\vec{w}$, such that $\vec{F}(\vec{x},\vec{w}) = \vec{x}$. We refrain from explicitly writing $\vec{x}(\vec{w})$ to avoid overly cluttered notation. At such $\vec{x}$, $\frac{dE}{d\vec{w}} = \frac{d\mathcal{H}}{d\vec{w}}$, and we can calculate the gradient using the Hamiltonian.

We've gone through the trouble of introducing the Hamiltonian since, by properly choosing the Lagrange multipliers $\vec{B}$, we can ensure that $\frac{d\mathcal{H}}{d\vec{w}}$ and thus $\frac{dE}{d\vec{w}}$ are analytically simple. By the chain rule,

$$\begin{aligned}\frac{d\mathcal{H}(\vec{x},\vec{w},\vec{B})}{d\vec{w}} &= \frac{\partial\mathcal{H}}{\partial\vec{x}}\cdot\frac{d\vec{x}}{d\vec{w}} + \frac{\partial\mathcal{H}}{\partial\vec{w}}\cdot\frac{d\vec{w}}{d\vec{w}} + \frac{\partial\mathcal{H}}{\partial\vec{B}}\cdot\frac{d\vec{B}}{d\vec{w}} \\ &= \frac{\partial\mathcal{H}}{\partial\vec{x}}\cdot\frac{d\vec{x}}{d\vec{w}} + \frac{\partial\mathcal{H}}{\partial\vec{w}}\,,\end{aligned}$$

where the second line follows because $\frac{d\vec{w}}{d\vec{w}} = \mathbf{I}$ and because $\frac{\partial\mathcal{H}}{\partial\vec{B}} = \vec{F}(\vec{x},\vec{w}) - \vec{x} = 0$ (since $\vec{x}$ is always chosen to be a fixed point of $\vec{F}(\vec{x},\vec{w})$ given $\vec{w}$). We will choose the Lagrange multipliers $\vec{B}$ so that $\frac{\partial\mathcal{H}}{\partial\vec{x}} = 0$ when $\vec{F}(\vec{x}) = \vec{x}$, in which case

$$\frac{dE}{d\vec{w}} = \frac{d\mathcal{H}(\vec{x},\vec{w},\vec{B})}{d\vec{w}} = \frac{\partial\mathcal{H}}{\partial\vec{w}} = \vec{B}\cdot\frac{\partial\vec{F}}{\partial\vec{w}}$$

so long as we restrict the gradient to the internal parameters w' for which $\frac{\partial E}{\partial w'} = 0$. The compu-

tationally nontrivial component of the gradient of the error function is thus encapsulated in the Lagrange multipliers $\vec{B}$. The requirement that $\frac{\partial \mathcal{H}}{\partial \vec{x}} = 0$ when $\vec{F}(\vec{x}) = \vec{x}$ implies that

$$0 = \frac{\partial E(\vec{x})}{\partial x_i} + \left(\sum_j B_j \cdot \frac{\partial F_j}{\partial x_i} \right) - B_i \tag{A.8}$$

for each i, at $\vec{x}$ such that $\vec{F}(\vec{x}) = \vec{x}$. Rearranging and coalescing the instances of equation A.8 using matrix notation, we find

$$\vec{B} = \frac{\partial E(\vec{x})}{\partial \vec{x}} + \left(\nabla \vec{F}^{\top} \right) \cdot \vec{B} \tag{A.9}$$

at the fixed points of $\vec{F}$. If we further require that $\vec{T}(\vec{x}^*, \vec{w}) = \vec{B}$, then equation A.9 is equivalent to equations A.7 and 2.8.

A.1.3 Intrinsic gradient networks calculate approximate gradients at approximate fixed points

In practical circumstances, such as in a neural implementation, it may not be possible to identify the exact fixed points of $\vec{F}$. For instance, using fixed point iteration or an algorithm that locally minimizes that magnitude of $\vec{F}(\vec{x}) - \vec{x}$, it may only be possible to find $\vec{x}$ such that $\vec{F}(\vec{x}, \vec{w}) \approx \vec{x}$. Such $\vec{x}$ may in fact be far from a true fixed point x^*, for which $\vec{F}(x^*, \vec{w}) = \vec{x}^*$. Fortunately, we can extend the derivation of appendix A.1.2 to show that, when the intrinsic gradient equation (2.9) is satisfied with an appropriate slack function, the training function $\vec{T}$ approximately calculates the gradient when the network state $\vec{x}$ approximates a fixed point, even if it is not near a true fixed point.

Consider the Hamiltonian

$$\mathcal{H}(\vec{x}, \vec{w}, \vec{B}) = E(\vec{x}) + \vec{B} \cdot \left(\vec{F}(\vec{x}, \vec{w}) - \vec{x} + \vec{N}(\vec{w}) \right)$$

where $\vec{x}$ is a function of $\vec{w}$ such that $\vec{F}(\vec{x}(\vec{w}), \vec{w}) \approx \vec{x}(\vec{w})$. Defining $\vec{x}$ as a function of $\vec{w}$ simply formalizes the fact that the network consistently finds an approximate fixed point as the parameters are changed. That is, it identifies the trajectory of the approximate fixed points as the parameters evolve. The particular function $\vec{x}(\vec{w})$ may vary depending upon the details of the network dynamics, and may even be a function of other external parameters. We generally refrain from writing $\vec{x}(\vec{w})$ to avoid overly cluttered notation. The function $\vec{N}(\vec{w})$ offsets the error in the calculation of the fixed point; it is defined so that $\vec{F}(\vec{x}, \vec{w}) - \vec{x} + \vec{N}(\vec{w}) = 0$ when $\vec{x}$ is chosen according to its function of $\vec{w}$. Since $\vec{F}(\vec{x}, \vec{w}) \approx \vec{x}$, $\vec{N}(\vec{w}) \approx 0$. $\vec{N}$ is naturally a function of $\vec{w}$ since $\vec{x}$ is a function of $\vec{w}$.

We wish to calculate $\frac{dE}{d\vec{w}}$ under these circumstances where the network consistently comes close to satisfying the fixed point condition. Since $\vec{N}(\vec{w})$ is constructed so that $\vec{F}(\vec{x}, \vec{w}) - \vec{x} + \vec{N}(\vec{w}) = 0$ at

the $\vec{x}$ of interest, $\mathcal{H}(\vec{x},\vec{w},\vec{B}) = E(\vec{x})$ and $\frac{dE}{d\vec{w}} = \frac{d\mathcal{H}}{d\vec{w}}$ at the $\vec{x}$ of interest, regardless of the particular values of $\vec{w}$ and $\vec{B}$. By the chain rule

$$\begin{aligned}\frac{d\mathcal{H}(\vec{x},\vec{w},\vec{B})}{d\vec{w}} &= \frac{\partial\mathcal{H}}{\partial\vec{x}}\cdot\frac{d\vec{x}}{d\vec{w}} + \frac{\partial\mathcal{H}}{\partial\vec{w}}\cdot\frac{d\vec{w}}{d\vec{w}} + \frac{\partial\mathcal{H}}{\partial\vec{B}}\cdot\frac{d\vec{B}}{d\vec{w}} \\ &= \frac{\partial\mathcal{H}}{\partial\vec{x}}\cdot\frac{d\vec{x}}{d\vec{w}} + \frac{\partial\mathcal{H}}{\partial\vec{w}}\,,\end{aligned}$$

where the second line follows because $\frac{d\vec{w}}{d\vec{w}} = \mathbf{I}$ and because $\frac{\partial\mathcal{H}}{\partial\vec{B}} = \vec{F}(\vec{x},\vec{w}) - \vec{x} + \vec{N}(\vec{w}) = 0$ (since $\vec{N}(\vec{w})$ is always chosen to offset the error of the fixed point). Proceeding as in appendix A.1.2, we find

$$\begin{aligned}\frac{\partial\mathcal{H}}{\partial x_i} &= \frac{\partial E(\vec{x})}{\partial x_i} + \left(\sum_j B_j\cdot\frac{\partial F_j(\vec{x},\vec{w})}{\partial x_i}\right) - B_i + \left(\sum_j B_j\cdot\frac{\partial N_j(\vec{w})}{\partial x_i}\right) \\ &= \frac{\partial E(\vec{x})}{\partial x_i} + \left(\sum_j B_j\cdot\frac{\partial F_j}{\partial x_i}\right) - B_i\,,\end{aligned}$$

where the second line follows because $\vec{N}(\vec{w})$ is only a function of $\vec{w}$, rather than $\vec{x}$.

The intrinsic gradient equation requires that

$$\vec{T}(\vec{x}) = \vec{S}(\vec{x},\vec{F}(\vec{x})) + \nabla E(\vec{x}) + \left(\nabla\vec{F}^{\top}(\vec{x})\right)\cdot\vec{T}(\vec{x})\,.$$

Thus, if we choose $\vec{B} = \vec{T}(\vec{x})$ (as we are free to do, since $\vec{B}$ is an arbitrary parameter which we introduced for our convenience) we find

$$\frac{\partial\mathcal{H}}{\partial x_i} = -\vec{S}(\vec{x},\vec{F}(\vec{x}))\,.$$

By definition, $\vec{S}(\vec{a},\vec{b}) = 0$ if $\vec{a} = \vec{b}$, and using a slack function like $\vec{S}(\vec{a},\vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, $\vec{S}(\vec{a},\vec{b}) \approx 0$ when $\vec{a} \approx \vec{b}$, so long as $\vec{T}$ has small derivatives. Thus, assuming that $\vec{F}(\vec{x}) \approx \vec{x}$, we find that $\frac{\partial\mathcal{H}}{\partial x_i} \approx 0$.

As a result,

$$\begin{aligned}\frac{dE}{d\vec{w}} = \frac{d\mathcal{H}(\vec{x},\vec{w},\vec{B})}{d\vec{w}} &= \frac{\partial\mathcal{H}}{\partial\vec{x}}\cdot\frac{d\vec{x}}{d\vec{w}} + \frac{\partial\mathcal{H}}{\partial\vec{w}} \\ &= \frac{\partial\mathcal{H}}{\partial\vec{x}}\cdot\frac{d\vec{x}}{d\vec{w}} + \vec{B}\cdot\frac{\partial\vec{F}(\vec{x},\vec{w})}{\partial\vec{w}} + \vec{B}\cdot\frac{\partial\vec{N}}{\partial\vec{w}} \\ &= -\vec{S}(\vec{x},\vec{F}(\vec{x}))\cdot\frac{d\vec{x}}{d\vec{w}} + \vec{T}(\vec{x})\cdot\frac{\partial\vec{F}}{\partial\vec{w}} + \vec{T}(\vec{x})\cdot\frac{\partial\vec{N}}{\partial\vec{w}} \\ &\approx \vec{T}(\vec{x})\cdot\frac{\partial\vec{F}}{\partial\vec{w}} \qquad\qquad (A.10)\end{aligned}$$

when $\vec{F}(\vec{x}) \approx \vec{x}$, $\frac{d\vec{x}}{d\vec{w}}$ and $\vec{T}(\vec{x})$ are bounded, and $\frac{\partial\vec{N}}{\partial\vec{w}} \approx 0$. Since $\vec{N}(\vec{w}) = \vec{x}(\vec{w}) - \vec{F}(\vec{x}(\vec{w}))$, the

restriction that $\frac{\partial \vec{N}}{\partial \vec{w}} \approx 0$ basically implies that the error of the approximate fixed point must change gradually as the parameters are varied. It constitutes a restriction on the type of approximate fixed points at which the training function $\vec{T}(\vec{x})$ is a good approximation of the gradient in an intrinsic gradient network. Intuitively, if $\vec{x} - \vec{F}(\vec{x})$ varies quickly as $\vec{w}$ is changed, then the restriction that $\vec{F}(\vec{x}, \vec{w}) \approx \vec{x}$ does not capture most of the local variation of $\vec{x}$ as a function of $\vec{w}$. It is then not surprising that an algorithm based upon the gradient of the points where $\vec{F}(\vec{x}, \vec{w}) = \vec{x}$ does not yield a good approximation to the gradient of $\vec{x}(\vec{w})$.

A.1.4 Approximate output functions and training functions

We can also evaluate the effect of incorrect $\vec{T}$ or $\vec{F}$ using the Hamiltonian formulation. Rather than or in addition to assuming that $\vec{F}(\vec{x}) \approx x$ along the trajectory $\vec{x}(\vec{w})$, assume that the intrinsic gradient equation (2.9) is only approximately satisfied. That is,

$$\vec{T}(\vec{F}(\vec{x})) = \vec{S}'(\vec{x}, \vec{F}(\vec{x})) + \nabla E(\vec{x}) + \left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x})$$

where $\vec{S}'(\vec{x}, \vec{F}(\vec{x})) \approx 0$ when $\vec{x} = \vec{F}(\vec{x})$. The derivation of equation A.10 still applies, with $\vec{N}(\vec{w}) = 0$ if the exact fixed points can be found. The error in the calculation of $\frac{dE}{d\vec{w}}$ using the training function, $-\vec{S}(\vec{x}, \vec{F}(\vec{x})) \cdot \frac{d\vec{x}}{d\vec{w}}$, is a function of quantities that can be measured based upon the actual operation of the network: specifically, the degree to which equation A.7 is not satisfied at the fixed point and the rate of change of the fixed point in response to a small change in the parameters.

A.2 Modular intrinsic gradient networks

The fixed point equation $\vec{F}(\vec{x}^*) = \vec{x}^*$ defines a subspace of $\mathbb{R}^n$ over which the intrinsic trainability constraint

$$\vec{T}(\vec{x}^*) = \nabla E(\vec{x})\big|_{x^*} + \left(\nabla \vec{F}^{\top}(\vec{x})\Big|_{x^*}\right) \cdot \vec{T}(\vec{x}^*) \tag{A.11}$$

must hold (copied for convenience from equation 2.8). However, the subspace where $\vec{F}(\vec{x}^*) = \vec{x}^*$ is difficult to identify in general. In sections 2.1 and 2.2, we avoid the necessity of identifying the subspace where $\vec{F}(\vec{x}^*) = \vec{x}^*$ by instead explicitly specifying the deviation of equation A.11 from equality as a function of $\vec{x}$ and $\vec{F}(\vec{x})$, using the slack function $\vec{S}(\vec{x}, \vec{F}(\vec{x}))$. Since we require that $\vec{S}(\vec{x}, \vec{F}(\vec{x})) = 0$ when $\vec{F}(\vec{x}) = \vec{x}$, the constraint of equation A.11 holds for x^* such that $\vec{F}(\vec{x}^*) = \vec{x}^*$, but need not hold for other $\vec{x} \in \mathbb{R}^n$. Unfortunately, we have little direct intuition for how the slack function should be chosen for $\vec{x}$ such that $\vec{F}(\vec{x}) \neq \vec{x}$.

Naively, we might think that we can avoid choosing a slack function by extending the intrinsic trainability constraint of equation A.11 to all values of $\vec{x} \in \mathbb{R}^n$. If the intrinsic trainability constraint holds for all $\vec{x} \in \mathbb{R}^n$, it certainly holds on the subspace where $\vec{F}(\vec{x}^*) = \vec{x}^*$. However, such an extension

is equivalent to choosing the slack function $\vec{S}(\vec{a}, \vec{b}) = 0$ in the intrinsic gradient equation (2.9), which we consider in appendix A.3.2.

On the other hand, we must also be careful to avoid enforcing equation A.11 over too small a subspace of $\vec{x}$. The output functions $\vec{F}$ are parameterized by $\vec{w}$, which includes the inputs to the network in the form of the input parameters, as well as the internal parameters which must be trained by the learning algorithm. We want to be able to calculate the gradient of the error function for all possible values of the inputs and the internal parameters. We thus need equation A.11 to hold on the subspace of $\vec{x}$ for which $\vec{F}(\vec{x}, \vec{w}) = \vec{x}$ for any possible value of $\vec{w}$. Otherwise, either training the internal parameters or changing the inputs could disrupt our ability to continue training.

The prospect of considering a large set of parameters $\vec{w}$ might initially appear to complicate the problem. However, further consideration of equation A.11 will show that the resulting restrictions on $\vec{F}(\vec{x})$ can have a modular structure, with independent subsets of equation A.11 defined on disjoint sets of units. Given a few simple restrictions on $\vec{T}$ and the parameterization of $\vec{F}(\vec{x}, \vec{w})$, the extension to a large set of $\vec{w}$ will in fact simplify the structure of the space of $\vec{x}$ where $\vec{F}(\vec{x}, \vec{w}) = \vec{x}$ for some $\vec{w}$, and thus where equation A.11 must hold. The resulting modular structure of equation A.11 will allow us to break this large, unmanageable equation into a set of smaller, simpler equations, which can be solved separately.

A.2.1 Definition of modules

We define a *module* to be a group of units for which the restriction imposed on the constituent parameters by equation A.11 is independent of parameters or units outside of the module. Parameters of units within a module may be linked, but changing the parameters of a unit in one module does not affect the allowable parameters of the units in any other module. We further require that the inputs to a module be disjoint from both the outputs of the module (i.e., the units in the module), and the inputs to all other modules. If unit x_i is in a module, then for all j such that unit x_j is also in that module, $F_j(\vec{x})$ does not depend on x_i. Each unit projects to at most one module. Finally, we assume that the connections of $\vec{T}(\vec{x})$ are a subset of the bidirectional connections of $\vec{F}(\vec{x})$. That is, we assume that $T_i(\vec{x})$ can only depend on x_j such that $F_i(\vec{x})$ is a function of x_j and $F_j(\vec{x})$ is a function of x_i. Intuitively, this implies that a unit can only calculate the gradient for some other unit if the two units communicate with each other.

In a traditional feedforward neural network (ignoring any backpropagation signals), a single unit is basically a module, since the inputs of each such unit are disjoint from its outputs, and the parameters of each unit can be chosen independently of every other unit (although the inputs to each unit are not disjoint). Modules in intrinsic gradient networks, such as the composite nodes of section 3.2, are necessarily larger, since they must facilitate the computation of the gradient in addition to the feedforward computation. For instance, when the backpropagation signals are

included with a traditional neural network to form an intrinsic gradient network, as in section 3.1.2, all the linear transformations between a pair of adjacent layers (including both feedforward and backpropagation signals) together constitute a module.

Signal fan-out between modules is performed explicitly by multiple units in a module, rather than implicitly by allowing a single unit to influence output functions of multiple modules. In section 3.2, composite factor nodes, and both atomic and composite variable nodes, can be understood as modules. In contrast, the atomic factor nodes of section 3.2 are not modules, since a single variable node unit projects to multiple atomic factor node units.

Since the units in a module are disjoint from their inputs, the outputs of a module at a fixed point can be calculated quickly and analytically given the inputs to the module. Moreover, since the parameters within a module are only constrained relative to each other, it is possible to choose the parameters of a module independently of the parameters of all other modules. We can generally generate all possible inputs to any given module by varying the independent parameters of other modules (in particular, the input parameters), so each module must satisfy equation A.11 for all possible inputs. In section A.2.3, we will show that this implies that modular intrinsic gradient network have the slack function $\vec{S}(\vec{a},\vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$. First, though, we must characterize modular intrinsic gradient networks.

In general, equation A.11 defines all units in terms of each other, so that changing the parameters of one unit can force a complementary change in the parameters of all other units. Our definition of modules as being independently parameterized with disjoint inputs implies a corresponding independence in equation A.11. Specifically, equation A.11 must break into disjoint components corresponding to the modules. This is difficult to see directly from equation A.11; however, since we are only interested in equation A.11 at $\vec{x}^*$ for which $\vec{F}(\vec{x}^*) = \vec{x}^*$, we are free to transform the point at which equation A.11 is applied according to $\vec{x}^* \rightarrow \vec{F}(\vec{x}^*)$. This leaves all elements of the desired subspace unchanged. Moreover, this transformation can be applied to only a subset of the instances of the term $\vec{x}^*$ in equation A.11 without altering the constraint in the subspace within which $\vec{F}(\vec{x}^*) = \vec{x}^*$. Applying one such substitution on the left-hand side of equation A.11, we obtain

$$\vec{T}(\vec{F}(\vec{x}^*)) = \nabla E(\vec{x})\big|_{x^*} + \left(\nabla \vec{F}^{\top}(\vec{x})\Big|_{x^*} \right) \cdot \vec{T}(\vec{x}^*)\,, \tag{A.12}$$

which would be equivalent to the intrinsic gradient equation (2.9) with our favored slack function $\vec{S}(\vec{a},\vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, were we to enforce equation A.12 for all $\vec{x} \in \mathbb{R}^n$, rather than just those $\vec{x}^*$ for which $\vec{F}(\vec{x}^*) = \vec{x}^*$. Equation A.12 will prove to be a natural characterization of modular intrinsic gradient networks. The independence of the modules will be manifested in the independence of groups of rows of equation A.12. In section A.2.3, we will use this independence to extend equation A.12 to all $\vec{x}$ within each such group, proving that the slack function must be $\vec{S}(\vec{a},\vec{b}) =$

$\vec{T}(\vec{a}) - \vec{T}(\vec{b})$.

A.2.2 The restrictions of connection topology on modularity

The ability to break equation A.12 into disjoint components can be analyzed by considering the arguments on which each row is defined. The rows can be partitioned into groups with disjoint domains only if they implement independent constraints over units with disjoint arguments; that is, if the partitions correspond to distinct modules. We abstract the identity of the arguments from the rest of the computation by considering vectors of functions $\vec{\mathcal{F}}$ and $\vec{\mathcal{T}}$ that capture the mapping of inputs to outputs performed by $\vec{F}$ and $\vec{T}$. The function $\mathcal{F}_i$ maps from a vector of sets to a set, with output equal to the union of the input elements with indices matching the arguments of $F_i(\vec{x})$; $\mathcal{F}_i(\vec{\mathcal{X}}) = \cup_{j \in \arg(F_i)} \mathcal{X}_j$. For instance, if $F_i(\vec{x}) = \frac{x_a^{2/3} \cdot x_b^{2/3}}{x_c^{1/3}}$, then the indices of the arguments of $F_i(\vec{x})$ are a, b, and c, so $\mathcal{F}_i(\vec{\mathcal{X}}) = \mathcal{X}_a \cup \mathcal{X}_b \cup \mathcal{X}_c$. The function $\mathcal{F}_i^{-1}$ is not a true inverse, but rather captures the mapping of inputs to outputs performed by $\vec{F}^{-1}$. It maps from a vector of sets to a set, with output equal to the union of the input elements with indices matching functions $F_j(\vec{x})$ with x_i as an argument; $\mathcal{F}_i^{-1}(\vec{\mathcal{X}}) = \cup_{j | x_i \in \arg(F_j)} \mathcal{X}_j$, and it is defined in this manner even if $\vec{F}(\vec{x})$ is not actually invertible. Similarly, $\mathcal{T}_i$ is a function from a vector of sets to a sets, with output equal to the union of the input elements matching the arguments of $T_i(\vec{x})$. We further define $\vec{\mathcal{X}}^x$ to be a vector of sets, with $\mathcal{X}_i^x = \{x_i\}$.

We can use these set-theoretic tools to construct an abstraction of equation A.12 where each entry of the resulting vector equation contains the arguments of the corresponding entry in equation A.12. A module will then corresponds to a group of rows in this vector equation, such that the union of the contents of the rows is disjoint from the union of the contents of the rows of all other modules. Modules identified in this manner have independent parameterizations and disjoint domains, consistent with our definition. We will further show that the indices of the rows constituting a module correspond to its elements (outputs), so the union of the contents of the rows of a module is also disjoint from the indices of the rows in the group, since the outputs of a module are disjoint from the inputs. For simplicity, we assume that the internal structure of the network is independent of the inputs; that is, we assume that the solutions to the intrinsic gradient equation (2.9) can be split into a homogeneous part with $\nabla E(\vec{x}) = 0$ that is independent of inhomogeneous part which accounts for the chosen error function. Consequently, we only address the homogeneous part of the equation below, and set $\nabla E(\vec{x}) = 0$.

The arguments of each row of the right-hand side of equation A.11 are a subset (not necessarily proper) of

$$\vec{\mathcal{F}}^{-1}(\vec{\mathcal{F}}(\vec{\mathcal{X}}^x)) \cup \vec{\mathcal{F}}^{-1}(\vec{\mathcal{T}}(\vec{\mathcal{X}}^x)) \,. \tag{A.13}$$

As mentioned above, the function $\vec{\mathcal{F}}^{-1}(\mathcal{X})$ is not a true inverse; it maps from the outputs to the

inputs of $\vec{F}$, rather than from the inputs to the outputs. The first component of expression A.13, $\vec{\mathcal{F}}^{-1}(\vec{\mathcal{F}}(\vec{\mathcal{X}}^x))$, reflects the fact that the non-zero terms of the ith row of $\nabla\vec{F}^\top(\vec{x})$ (and thus the arguments of row i of equation A.11) include all units that are inputs to an $F_j(\vec{x})$ that has x_i as an input. The second term of expression A.13, $\vec{\mathcal{F}}^{-1}(\vec{\mathcal{T}}(\vec{\mathcal{X}}^x))$, reflects the fact that the arguments of row i of equation A.11 also include all arguments of $T_j(\vec{x})$ for j such that x_i is an input to $F_j(\vec{x})$, corresponding to filtering $\vec{T}(\vec{x})$ through the non-zero elements of the ith row of $\nabla\vec{F}^\top(\vec{x})$. Our assumption about $\vec{T}$ implies that $\vec{\mathcal{T}}(\vec{\mathcal{X}}^x) \subset \vec{\mathcal{F}}(\vec{\mathcal{X}}^x) \cap \vec{\mathcal{F}}^{-1}(\vec{\mathcal{X}}^x)$, so the second term of expression A.13 is a subset of the first, and expression A.13 is equal to $\vec{\mathcal{F}}^{-1}(\vec{\mathcal{F}}(\vec{\mathcal{X}}^x))$. Modules have disjoint inputs, so all of the units that have x_i as an input, $\mathcal{F}_i^{-1}(\vec{\mathcal{X}}^x)$, must be in a single module. Since row i of expression A.13 is equal to $\vec{\mathcal{F}}^{-1}(\vec{\mathcal{F}}(\vec{\mathcal{X}}^x))$, the arguments of row i on the right-hand side of equation A.11 only contain inputs to the module for which x_i is an input.

The arguments of each row of the left-hand side of equation A.11 are clearly $\vec{\mathcal{T}}(\vec{\mathcal{X}}^x)$. Since we assume that $\vec{\mathcal{T}}(\vec{\mathcal{X}}^x) \subset \vec{\mathcal{F}}(\vec{\mathcal{X}}^x) \cap \vec{\mathcal{F}}^{-1}(\vec{\mathcal{X}}^x)$ and that modules have disjoint inputs, the left-hand side of equation A.11 is only dependent upon the outputs of (i.e., units that are elements of) the module for which x_i is an input. We've assumed that modules function at any fixed point, regardless of their input. Consequently, if we transform the left-hand side of equation A.11 using $\vec{F}(\vec{x})$ so the module outputs match their inputs, each row of equation A.11 must be satisfied for all (module inputs) $\vec{x}$. That is, modular intrinsic gradient networks must satisfy equation A.12 for all $\vec{x}$.

Considering equation A.12 directly, our assumption that the training function $\vec{T}$ is a subset of the bidirectional connections of the output functions $\vec{F}$ implies that a module only contributes to the rows of equation A.12 corresponding to its inputs (arguments), that these rows of equation A.12 are determined only by that single module, and that the only units present in the final equations of these rows are the inputs (arguments) of the module defining the rows. On the left-hand side, $\vec{T}$ maps the output functions F_i of a module to the rows corresponding to its input units x_j. Because each unit only projects to a single module, no other modules map to these rows. On the right-hand side, F_i only contributes to rows of the transposed Jacobian corresponding to its input units x_j, and rows of the transposed Jacobian corresponding to the inputs of a module only have non-zero entries in columns corresponding to the elements of the module. The training function $\vec{T}$ maps from the inputs (arguments) of a module to its elements, so when we take the inner product of the transposed Jacobian and $\vec{T}(\vec{x}^*)$, only the inputs (arguments) of a module in the right-most $\vec{x}^*$ align with the non-zero columns within a row of the transposed Jacobian induced by that module.

A.2.3 Modular intrinsic gradient networks have the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$

Given the existence of the modules with independent inputs and disjoint outputs identified above, the space $\vec{F}(\vec{x}) = \vec{x}$ takes a very simple form. Restricting our attention to a single module (as is sufficient for equation A.12 because the modules correspond to independent sets of rows), $\vec{F}(\vec{x}) = \vec{x}$ holds exactly when the outputs of the module are set by $\vec{F}(\vec{x})$ for every valid set of inputs. If we make the rather modest assumption that all inputs are possible for a module (at some fixed point of the network) given each configuration of the parameters directly governing the module, then equation A.12 over $\vec{F}(\vec{x})$ must hold for all configurations $\vec{x}$ of the inputs to the module (and for each configuration of the parameters directly governing the input-output mapping of the module). The assumption that all inputs to a module are possible with some parameterization of the other modules follows directly if we require that a module satisfy the intrinsic trainability constraint (equation A.11) for multiple network topologies (i.e., multiple arrangements of modules) as well as multiple parameter settings for those network topologies. If all inputs to a module are controlled directly by input parameters (e.g., with a linear error function, as in section 2.4), then any possible input configuration to the module can be made to satisfy $\vec{F}(\vec{x}) = \vec{x}$ simply by choosing the input parameters appropriately.

Since all dependencies between the output functions of a single module are contained within the module, and the fixed point condition can reasonably be extended to all inputs to the module, we can solve for the output functions of each module independently without choosing a slack function. The slack function was only necessary to deal with the cases where $\vec{F}(\vec{x}) \neq \vec{x}$. All of the elements x_i of $\vec{x}$ that influence the rows of equation A.12 corresponding to the inputs to a module, are themselves inputs to the module, which can generally be set arbitrarily, independent of the outputs. The fixed point is thus satisfied for each input configuration by appropriately choosing the values of an independent set of output units x_j so that they match the values of the associated output functions F_j. As a result, equation A.12 can be extended to

$$\vec{T}(\vec{F}(\vec{x})) = \nabla E(\vec{x})|_x + \left(\nabla \vec{F}^\top(\vec{x}) \Big|_x \right) \cdot \vec{T}(\vec{x}) \tag{A.14}$$

for all $\vec{x}$, which is identical to equation 2.13 and thus corresponds to our favored slack function, $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$.

Interestingly, other slack functions seem to be able to yield different sets of output functions for which the homogeneous solutions have a similar structure, such as the example described in appendix A.3.3. In that case, the full value of each unit is split between the homogeneous and the inhomogeneous solution. Nevertheless, the inhomogeneous solutions seem unlikely to cancel out the homogeneous solutions to recover the result of equation A.14. It seems plausible that most inputs to

each module do not correspond to a fixed point, in which case the arguments of this section do not apply. That is, the assumption that all inputs to a module are possible fixed points seems to force the choice of the slack function. If a different slack function is imposed, this implicitly determines a sparser set of possible inputs at the fixed points.

Viewed in another way, the output functions consistent with different slack functions may not break into independent modules, as we have assumed in this section. For instance, as discussed in appendix A.8, belief propagation does not correspond to the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$. While the factor and variable nodes of belief propagation might initially seem to constitute independent modules, the connections between the nodes are severely restricted: no loops are permitted within the topology. The factor and variable nodes of belief propagation are thus not independent modules, since the modules cannot be combined arbitrarily.

A.2.4 Modular intrinsic gradient networks with linear parameterizations have linear training functions

Consider the case where each module is fully linearly parameterized, so that all outputs are possible by adjusting the inputs and parameters, and the effects of the parameters combine linearly. In particular, if $\vec{F}(\vec{x}, \vec{w})$ and $\vec{F}(\vec{x}, \vec{w}')$ are both acceptable output functions for the module (i.e., solutions to equation A.11), then $\alpha \cdot \vec{F}(\vec{x}, \vec{w})$ and $\vec{F}(\vec{x}, \vec{w}) + \vec{F}(\vec{x}, \vec{w}')$ must also be acceptable output functions for the module (for all real α), although they need not be produced by the parameters $\alpha \cdot \vec{w}$ or $\vec{w} + \vec{w}'$, respectively. For instance, we could use $\vec{F}(\vec{x}, \vec{w}) = \vec{N}^{\top}(\vec{w}) \cdot \vec{F}'(\vec{x})$, where $\vec{N}(\vec{w})$ is an invertible function of $\vec{w}$. No basis output $\vec{F}(\vec{x})$ is rendered inadmissible by the linearity requirement, and in this sense the linear parameterization does not directly constrain $\vec{F}(\vec{x}, \vec{w})$ as a function of the units $\vec{x}$, but given some $\vec{F}(\vec{x}, \vec{w})$, other $\vec{F}(\vec{x}, \vec{w}')$ can be inferred.

We will show that if an intrinsic gradient network is modular and fully linearly parameterized, then the training function $\vec{T}$ must be linear. Consider two solutions to the homogeneous part of equation A.12 with different parameters, $F(\vec{x}, \vec{w})$ and $F(\vec{x}', \vec{w}')$, restricted to the rows of the resulting vector equation corresponding to the inputs to the selected module, such that $\vec{x}$ and $\vec{x}'$ are both fixed points of their respective output functions but identical on the inputs to a given module. By assumption, we can focus on just the homogeneous part of the equation now and deal with the inhomogeneous solution later.[1] When we add the two corresponding instances of equation A.12 (restricted to the rows of the vector equation corresponding to the inputs to the selected module), we obtain

$$\vec{T}(F(\vec{x}, \vec{w})) + \vec{T}(F(\vec{x}, \vec{w}')) = \left[\nabla \left(F(\vec{x}, \vec{w}) + F(\vec{x}, \vec{w}')\right)^{\top}\right] \cdot \vec{T}(\vec{x}) \,, \tag{A.15}$$

[1] Since we will find that $\vec{T}$ is linear, an inhomogeneous solution can be additively composed with any linear combination of homogeneous solutions.

since the inputs $\vec{x}$ are the same for both instances. If $F(\vec{x},\vec{w}) + F(\vec{x},\vec{w}')$ is also a solution (for some choice of the parameters) when restricted to these rows, then

$$\vec{T}\left(F(\vec{x},\vec{w}) + F(\vec{x},\vec{w}')\right) = \left[\nabla\left(F(\vec{x},\vec{w}) + F(\vec{x},\vec{w}')\right)^{\top}\right] \cdot \vec{T}(\vec{x}) \tag{A.16}$$

results from equation A.12, and

$$\vec{T}\left(F(\vec{x},\vec{w}) + F(\vec{x},\vec{w}')\right) = \vec{T}\left(F(\vec{x},\vec{w})\right) + \vec{T}\left(F(\vec{x},\vec{w}')\right)$$

because the right-hand sides of equations A.15 and A.16 are identical. By virtue of the assumption that the outputs of a module can take on any value, $\vec{T}$ must be linear for the selected rows. We can repeat the process for all modules, to prove that $\vec{T}$ is linear for all rows. Interestingly, the linearity of $\vec{T}$ seems to be induced by the presence of the transposed Jacobian on the right-hand side of the intrinsic gradient equation (2.9), and thus the requirement that the network calculate its own gradient. Without the transposed Jacobian, there would be no reason for the parameterization of the output functions to have any implications for the training function.

From a biological perspective, simple training functions $\vec{T}(\vec{x})$ of this form are of particular interest, since they imply that almost all of the work of computing the gradient is performed by the network itself as it finds a fixed point of the output functions $\vec{F}(\vec{x})$. The machinery required to train the parameters based upon the network state, in contrast, is extremely simple, consistent with the synaptic update mechanisms observed in the cortex (Malenka & Bear, 2004), as discussed in section 4.3.

A.2.5 Alternative definitions of the output state

The same constraints as equation A.11 and A.12 result if the output is calculated when $\vec{\mathscr{F}}(\vec{x}) = c \cdot \vec{x}$ for any $c \in \mathbb{R}$, $c \neq 0$. A different font is used to indicate that while $\vec{\mathscr{F}}$ is also a vector of output functions, it differs from the functions $\vec{F}$ discussed in the rest of this thesis in that the subspace within which the output and thus the gradient is calculated is defined by a different equation. An equivalent derivation to that in appendix A.1 yields the constraint

$$c \cdot \vec{T}(\vec{x}^*) = \nabla E(\vec{x})\big|_{x^*} + \left(\nabla \vec{\mathscr{F}}^{\top}(\vec{x})\Big|_{x^*}\right) \cdot \vec{T}(\vec{x}^*) \tag{A.17}$$

for x^* such that $\vec{\mathscr{F}}(\vec{x}^*) = c \cdot \vec{x}^*$. Transforming the instance of $\vec{x}^*$ on the left-hand side of equation A.17, just as in equation A.12, we obtain

$$c \cdot \vec{T}\left(\frac{1}{c} \cdot \vec{\mathscr{F}}(\vec{x}^*)\right) = \nabla E(\vec{x})\big|_{x^*} + \left(\nabla \vec{\mathscr{F}}^{\top}(\vec{x})\Big|_{x^*}\right) \cdot \vec{T}(\vec{x}^*)\,,$$

which is identical to equation A.12 so long as $\vec{T}(\vec{x})$ is linear in $\vec{x}$. Alternatively, we can introduce a slack function $\vec{S}(c \cdot \vec{x}, \vec{\mathscr{F}}(\vec{x}))$. By choosing the standard slack function $\vec{S}(c \cdot \vec{x}, \vec{\mathscr{F}}(\vec{x})) = \vec{T}(c \cdot \vec{x}) - \vec{T}(\vec{\mathscr{F}}(\vec{x}))$, we also obtain the original intrinsic gradient equation, so long as $\vec{T}(\vec{x})$ is linear.

Given a vector of output functions $\vec{\mathscr{F}}$ which satisfy equation A.17 when $\vec{\mathscr{F}}(\vec{x}) = c \cdot \vec{x}$, we can construct a homologous vector of output functions $\vec{F}$ which satisfy equation A.11 when $\vec{F}(x) = \vec{x}$ by defining $\vec{F}(\vec{x}) = \vec{\mathscr{F}}(\vec{x}) + (1-c) \cdot \vec{x}$. The fixed points at which $\vec{F}(\vec{x}) = x$ and an output is produced are not those where $\vec{\mathscr{F}}(\vec{x}) = \vec{x}$, but rather where $\vec{\mathscr{F}}(\vec{x}) = c \cdot \vec{x}$. Examination of the homogeneous part of equation A.12 reveals that, so long as $\vec{T}$ is linear, $c \cdot \vec{x}$ is a solution for $\vec{F}(\vec{x})$ (since $\nabla(c \cdot \vec{x})^\top = c \cdot \mathbf{I}$). This accounts for the difference between the solutions derived using $\vec{\mathscr{F}}(\vec{x}) = c \cdot \vec{x}$ with different values of c. If $\vec{T}$ is linear, any solution for $\vec{\mathscr{F}}(\vec{x})$ is also a solution for $\vec{F}(\vec{x})$ and vice versa, since they only differ by a multiple of $\vec{x}$, which can freely be added to any solution. Interestingly, if $\mathbf{D} = \mathbf{I}$, then it can further be seen by examination of equation 2.13 that $c \cdot \vec{x}$ is a solution for $\vec{G}(\vec{x})$ for all $c \in \mathbb{R}$.

A.3 A menagerie of slack functions

In this appendix, we consider three possible slack functions $\vec{S}$ in more detail. These three slack functions yield different sets of output functions $\vec{F}$. Unfortunately, it does not seem as if output functions corresponding to different slack functions can be combined into a single intrinsic gradient network. If the output functions are divisible into disjoint modules, each module could in principle have a different slack function. However, as we saw in appendix A.2, such modular intrinsic gradient networks necessarily have the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$.

A.3.1 Slack function example 1: $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$

In section 2.2, we examined the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, and combined it with the intrinsic gradient equation (2.9) to derive equation 2.10. We can show that if equation 2.10 holds, then the dynamics $\vec{x}(t+1) = \vec{F}(\vec{x}(t))$ of equation 2.1 are qualitatively similar to backpropagation-through-time (Rumelhart et al., 1986). Backpropagation-through-time computes the gradient of an error function defined over all time on a discrete-time non-equilibrium recurrent neural network by spatially unrolling the network activity over time into a hierarchical feedforward network, and then performing backpropagation on the unrolled feedforward network. Specifically, for a discrete-time recurrent network with the dynamics $\vec{x}(t+1) = \vec{F}(\vec{x}(t))$, backpropagation-through-time computes

$$\frac{dE(\vec{x})}{dx_i(t)} = \frac{\partial E(\vec{x})}{\partial x_i(t)} + \sum_j \frac{\partial F_j(\vec{x}(t)))}{\partial x_i(t)} \cdot \frac{dE(\vec{x})}{dx_j(t+1)}$$

for all x_i and t. Equation 2.10 corresponds to backpropagation-through-time through the mapping $\vec{T}(\vec{F}(\vec{x})) \sim \frac{dE}{d\vec{x}(t)}$ and $\vec{T}(\vec{x}) \sim \frac{dE}{d\vec{x}(t+1)}$, with ∇E and $\nabla \vec{F}^\top$ evaluated at $\vec{x}(t)$. Equations 2.1 and 2.10

thus implement an approximate backpropagation-through-time in which the backpropagation step is actually performed forwards in time, with the partial derivative taken at the current value of $\vec{x}$, rather than the $\vec{x}$ corresponding to the appropriate point of the forward propagation. This approximation is exact when the network is at a fixed point, since in that case the evolution of the network forwards in time is equivalent to the evolution of the network backwards in time.

Alternatively, equation 2.10 with the dynamics of equation 2.1 can be understood as an iterative procedure for satisfying equation 2.8. Each iteration of $\vec{x}(t+1) = \vec{F}(\vec{x}(t))$ sets each element of $\vec{x}(t+1)$ such that equation 2.8 is satisfied, assuming that no element of $\vec{x}$ on the right-hand side actually changes as a result of the update. Of course, if the dynamics have not yet reached a fixed point, then $\vec{x}$ does change on the right-hand side, so equation 2.8 is not actually satisfied as a result of the update, and the process must be repeated until a fixed point is reached.

As an example of the restrictions implicit in equation 2.10, consider the case where every summand of each function $T_i(\vec{x})$ has total degree q in $\vec{x}$, and every summand of each function $F_i(\vec{x})$ has total degree r in $\vec{x}$, where the *total degree* is the sum of the exponents of each factor in a summand. For example, in the expression $a^3 \cdot b^{\frac{3}{2}} \cdot c^{-2} + c^{\frac{5}{2}} + a^4 \cdot b^{-\frac{3}{2}}$, all summands have total degree $\frac{5}{2}$, whereas in the expression $a^3 \cdot b^{\frac{3}{2}} \cdot c^{-4} + c^{\frac{7}{3}} + a^3 \cdot b^{-\frac{3}{2}}$, all summands have different total degree. With these restrictions on the total degree of $\vec{T}(\vec{x})$ and $\vec{F}(\vec{x})$, both $\vec{T}(\vec{F}(\vec{x}))$ and $\left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x})$ yield vectors for which all terms have the same total degree. Specifically, all terms of $\vec{T}(\vec{F}(\vec{x}))$ have total degree $r \cdot q$, while all terms of $\left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x})$ have total degree $r - 1 + q$. If all the terms of $\nabla E(\vec{x})$ also have homogeneous total degree, then the total degree of $\vec{T}(\vec{F}(\vec{x}))$, $\nabla E(\vec{x})$, and $\left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x})$ must be equal, from which we can conclude

$$\begin{aligned} r \cdot q &= r - 1 + q \\ (q-1) \cdot (r-1) &= 0\,, \end{aligned}$$

so $q = 1$ or $r = 1$. That is, if $\vec{T}$, $\vec{F}$, and E have homogeneous total degree, then either $\vec{T}$ has a total degree of one or $\vec{F}$ has a total degree of one. Since linear functions have a total degree of one, this argument serves as further motivation to consider linear $\vec{T}$, as we do in section 2.3.

A.3.2 Slack function example 2: $\vec{S}(\vec{a}, \vec{b}) = \vec{0}$

It is also possible to construct nontrivial intrinsic gradient networks using the simplest possible slack function, $\vec{S}(\vec{a}, \vec{b}) = \vec{0}$, where $\vec{0}$ is a vector in which each element has the value 0. This slack function has the desirable property of ensuring that equation 2.8 holds everywhere, rather than just at the fixed points. Given this slack function, the intrinsic gradient equation (2.9) becomes

$$\vec{T}(\vec{x}) = \nabla E(\vec{x}) + \left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x})\,, \tag{A.18}$$

where we explicitly note the normally implicit dependence of the error function E and training function $\vec{T}$ on the parameters. Equation A.18 is a system of linear partial differential equations in $\vec{F}$, so a full solution for $\vec{F}$ is the sum of a particular solution to the inhomogeneous equation

$$\left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x}) = \vec{T}(\vec{x}) - \nabla E(\vec{x}) \,, \tag{A.19}$$

which is merely a rewriting of equation A.18, and a linear combination of solutions to the homogeneous equation

$$\left(\nabla \vec{F}^{\top}(\vec{x})\right) \cdot \vec{T}(\vec{x}) = \vec{0} \,, \tag{A.20}$$

which only contains the terms of equation A.18 that are dependent on $\vec{F}$. As in the case of equation 2.11 and 2.12, if the solutions to these two equations are independently parameterized, the solution to the inhomogeneous equation generates the input parameters, whereas the solutions to the homogenous equation generate only internal parameters. The input parameters are combined with the internal parameters to construct a full solution.

If $\vec{T}(\vec{x}) = \mathbf{T} \cdot \vec{x}$ is linear and invertible, with $\left(\mathbf{T}^{-1}\right)^{\top} \cdot \mathbf{T} = \mathbf{D}$ where $\mathbf{D}$ is a diagonal matrix, we can construct solutions to equation A.20 in a manner similar to that used in section 2.3.4. A derivation analogous to that in section 2.3.4 and appendix A.6 shows that

$$\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot \mathbf{D}^{-1} \cdot \nabla \left(c + \sum_k x_{\psi(k)} \cdot \prod_{j \neq \psi(k)} h_j^k \left(\frac{x_j}{x_{\psi(k)}} \right) \right) \tag{A.21}$$

yields solutions to the homogeneous equation A.20, where c is a constant, $\psi : \mathbb{Z}^+ \to \mathbb{Z}^+$ is a function from the positive integers to the indices of the units in the network, and each $h_j^k(x)$ indexed by j and k is a differentiable scalar function. The inhomogeneous equation A.19 is then solved by the sum of $\vec{F}^{T-part}(\vec{x}) = \vec{x}$ and an output function $\vec{F}^{E-part}(\vec{x})$ that accounts for the error function. For instance, if $E(\vec{x}) = \vec{c} \cdot \vec{x}$, then $\vec{F}^{E-part}(\vec{x}) = \left(\mathbf{T}^{-1}\right)^{\top} \cdot \vec{G}(\vec{x})$ where $G_i(\vec{x}) = -c_i \cdot \log(x_i)$ works; if $E(\vec{x}) = \frac{1}{2} \cdot \sum_i x_i^2$, then we can use $\vec{F}^{E-part}(\vec{x}) = \left(\mathbf{T}^{-1}\right)^{\top} \cdot \vec{G}(\vec{x})$ where $G_i(\vec{x}) = -x_i$. These intrinsic gradient networks are not modular, as defined in appendix A.2, since the solution to the inhomogeneous equation A.19 implies that x_i is an argument of $F_i(\vec{x})$, so the inputs and outputs of each module are not disjoint.

A.3.3 Slack function example 3: $\vec{S}(\vec{a}, \vec{b}) = \tau \cdot (\vec{T}(\vec{a}) - \vec{T}(\vec{b}))$

In section A.3.1, we observed that equation 2.10 with the dynamics of equation 2.1 can be understood as an iterative procedure for satisfying equation 2.8. Generalizing this full update, it is also reasonable to consider a partial update that moves each entry of $\vec{T}(\vec{F}(\vec{x}))$ a fraction of the distance between the current value of $\vec{T}(\vec{x})$ and the value required according to equation 2.8. If each partial

update is allowed to be of a different size τ_i, this corresponds to $\vec{S}(\vec{a},\vec{b}) = \boldsymbol{\tau}\cdot(\vec{T}(\vec{a}) - \vec{T}(\vec{b}))$, where $\boldsymbol{\tau}$ is a diagonal matrix for which we denote the ith diagonal entry by τ_i. The intrinsic gradient equation (2.9) then becomes

$$\boldsymbol{\tau}\cdot\vec{T}(\vec{F}(\vec{x})) = (\boldsymbol{\tau}-\mathbf{I})\cdot\vec{T}(\vec{x}) + \nabla E(\vec{x}) + \left(\nabla\vec{F}^{\top}(\vec{x})\right)\cdot\vec{T}(\vec{x})\,. \tag{A.22}$$

Equation A.22 reduces to equation 2.10 if $\boldsymbol{\tau} = \mathbf{I}$. If $\vec{T}(\vec{x})$ is linear, with $\vec{T}(\vec{x}) = \mathbf{T}\cdot\vec{x}$, then the homogeneous equation corresponding to equation A.22 is

$$\boldsymbol{\tau}\cdot\mathbf{T}\cdot\vec{F}(\vec{x}) = \left(\nabla\vec{F}^{\top}(\vec{x})\right)\cdot\mathbf{T}\cdot\vec{x}\,. \tag{A.23}$$

Assuming a linear, invertible $\vec{T}(\vec{x}) = \mathbf{T}\cdot\vec{x}$ with $\left(\mathbf{T}^{-1}\right)^{\top}\cdot\mathbf{T} = \mathbf{D}$ where $\mathbf{D}$ is a diagonal matrix, an equivalent derivation to that in section 2.3.4 and appendix A.6 yields the solution

$$\vec{F}(\vec{x}) = \mathbf{T}^{-1}\cdot(\mathbf{D}+\boldsymbol{\tau})^{-1}\cdot\nabla\left(c + \sum_k\left[x_{\psi(k)}^{\frac{D_{\psi(k)}+\tau_{\psi(k)}}{D_{\psi(k)}}}\cdot\prod_{j\neq\psi(k)}h_j^k\left(\frac{x_j^{\frac{D_j+\tau_i}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+\tau_{\psi(k)}}{D_{\psi(k)}}}}\right)\right]\right) \tag{A.24}$$

to equation A.23. This reduces to equation A.21 when $\boldsymbol{\tau} = \mathbf{0}$, and to equation 2.17 when $\boldsymbol{\tau} = \mathbf{I}$. As in section A.3.2, the inhomogeneous equation A.22 is solved by the sum of $\vec{F}^{T-part}(\vec{x}) = \vec{x}$ and an output function $\vec{F}^{E-part}(\vec{x})$ that accounts for the error function. For instance, if $\mathbf{T}$ is symmetric and $E(\vec{x}) = \vec{c}\cdot\vec{x}$, then $\vec{F}^{E-part}(\vec{x}) = \left(\mathbf{T}^{-1}\right)^{\top}\cdot\vec{G}(\vec{x})$ where $G_i(\vec{x}) = \frac{c_i}{\tau_i}$ works; if $\mathbf{T}$ is symmetric and $E(\vec{x}) = \frac{1}{2}\cdot\sum_i x_i^2$, then we can use $\vec{F}^{E-part}(\vec{x}) = \left(\mathbf{T}^{-1}\right)^{\top}\cdot\vec{G}(\vec{x})$ where $G_i(\vec{x}) = \frac{x_i}{\tau_i - 1}$. As when $\boldsymbol{\tau} = \mathbf{0}$ in section A.3.2, these intrinsic gradient networks do not satisfy the definition of modularity presented in appendix A.2, since the solution to the inhomogeneous equation A.22 implies that x_i is an argument of $F_i(\vec{x})$, so the inputs and outputs of each module are not disjoint.

A.4 Construction of a homologous intrinsic gradient network with linear training function

Given an intrinsic gradient network with an arbitrary training function, we can construct a larger, homologous intrinsic gradient network with a linear training function for which the output functions of the original intrinsic gradient network are a subset of those of the new network. Specifically, we replace each unit of the original network x_i with a new unit $\mathrm{x}_{i_{ff}}$, and each element of the original

training function $T_i(\vec{x})$ with a new, unparameterized unit $\mathrm{x}_{i_{fb}}$ of $\vec{x}$. We then choose

$$\begin{aligned}
\mathrm{x}_{i_{ff}} &= x_i \\
\mathrm{F}_{i_{ff}}(\vec{\mathrm{x}}) &= F_i(\vec{x}) \\
\mathrm{F}_{i_{fb}}(\vec{\mathrm{x}}) &= T_i(\vec{x}) \\
\mathrm{T}_{i_{ff}}(\vec{\mathrm{x}}) &= \mathrm{x}_{i_{fb}} \\
\mathrm{T}_{i_{fb}}(\vec{\mathrm{x}}) &= 0\,.
\end{aligned} \tag{A.25}$$

We use unitalicized symbols to denote the units and functions of the homologous network, but the distinction between the original and the homologous network is generally clear from the context and the subscripts. In the homologous network, $\vec{\mathrm{T}}(\vec{\mathrm{x}})$ is linear in $\vec{\mathrm{x}}$, although it is not invertible.

We shall now show that if the original network satisfied the intrinsic gradient equation (2.9), then this homologous construction also satisfies the intrinsic gradient equation, although with a different slack function. Since x_i is equivalent to $\mathrm{x}_{i_{ff}}$ we sometimes use functions of the original units x_i in the context of the homologous network, with the understanding that they refer to the corresponding homologous units $\mathrm{x}_{i_{ff}}$. Starting from the intrinsic gradient equation, we find:

$$\begin{aligned}
T_i(\vec{x}) &= S_i(\vec{x}, \vec{F}(\vec{x})) + \frac{dE(\vec{x})}{dx_i} + \sum_j \frac{\partial F_j(\vec{x})}{\partial x_i} \cdot T_j(\vec{x}) \\
\mathrm{F}_{i_{fb}}(\vec{\mathrm{x}}) &= S_i(\vec{x}, \vec{F}(\vec{x})) + \frac{dE(\vec{x})}{dx_i} + \sum_j \frac{\partial \mathrm{F}_{j_{ff}}(\vec{\mathrm{x}})}{\partial \mathrm{x}_{i_{ff}}} \cdot \mathrm{F}_{j_{fb}}(\vec{\mathrm{x}}) \\
\mathrm{x}_{i_{fb}} &= \left(\mathrm{x}_{i_{fb}} - \mathrm{F}_{i_{fb}}(\vec{\mathrm{x}})\right) - \left(\sum_j \frac{\partial \mathrm{F}_{j_{ff}}(\vec{\mathrm{x}})}{\partial \mathrm{x}_{j_{ff}}} \cdot \left(\mathrm{x}_{j_{fb}} - \mathrm{F}_{j_{fb}}(\vec{\mathrm{x}})\right)\right) + S_i(\vec{x}, \vec{F}(\vec{x})) \\
&\quad + \frac{dE(\vec{x})}{d\mathrm{x}_{i_{ff}}} + \sum_j \frac{\partial \mathrm{F}_{j_{ff}}(\vec{\mathrm{x}})}{\partial \mathrm{x}_{i_{ff}}} \cdot \mathrm{x}_{j_{fb}} \\
\mathrm{T}_{i_{ff}}(\vec{\mathrm{x}}) &= \left(\mathrm{x}_{i_{fb}} - \mathrm{F}_{i_{fb}}(\vec{\mathrm{x}})\right) - \left(\sum_j \frac{\partial \mathrm{F}_{j_{ff}}(\vec{\mathrm{x}})}{\partial \mathrm{x}_{j_{ff}}} \cdot \left(\mathrm{x}_{j_{fb}} - \mathrm{F}_{j_{fb}}(\vec{\mathrm{x}})\right)\right) + S_i(\vec{x}, \vec{F}(\vec{x})) \\
&\quad + \frac{dE(\vec{x})}{d\mathrm{x}_{i_{ff}}} + \sum_j \frac{\partial \mathrm{F}_{j_{ff}}(\vec{\mathrm{x}})}{\partial \mathrm{x}_{i_{ff}}} \cdot \mathrm{T}_{j_{ff}}(\vec{\mathrm{x}}) \\
&= \mathrm{S}_{i_{ff}}(\vec{\mathrm{x}}, \vec{\mathrm{F}}(\vec{\mathrm{x}})) + \frac{dE(\vec{\mathrm{x}})}{d\mathrm{x}_{i_{ff}}} + \sum_j \left(\frac{\partial \mathrm{F}_{j_{ff}}(\vec{\mathrm{x}})}{\partial \mathrm{x}_{i_{ff}}} \cdot \mathrm{T}_{j_{ff}}(\vec{\mathrm{x}}) + \frac{\partial \mathrm{F}_{j_{fb}}(\vec{\mathrm{x}})}{\partial \mathrm{x}_{i_{ff}}} \cdot \mathrm{T}_{j_{fb}}(\vec{\mathrm{x}})\right)\,,
\end{aligned} \tag{A.26}$$

where

$$\mathrm{S}_{i_{ff}}(\vec{\mathrm{x}}, \vec{\mathrm{F}}(\vec{\mathrm{x}})) = \left(\mathrm{x}_{i_{fb}} - \mathrm{F}_{i_{fb}}(\vec{\mathrm{x}})\right) - \left(\sum_j \frac{\partial \mathrm{F}_{j_{ff}}(\vec{\mathrm{x}})}{\partial \mathrm{x}_{j_{ff}}} \cdot \left(\mathrm{x}_{j_{fb}} - \mathrm{F}_{j_{fb}}(\vec{\mathrm{x}})\right)\right) + S_i(\vec{x}, \vec{F}(\vec{x})).$$

The first line restates the intrinsic gradient equation for the original network; the second line results

from applying equation A.25, the third line follows by adding and subtracting quantities that equal zero at a fixed point of the output functions, the fourth line is constructed by again applying equation A.25, and the fifth line collects appropriate terms into a new slack function. Moreover, it is easy to see that

$$\mathrm{T}_{i_{fb}}(\vec{\mathrm{x}}) = \frac{dE(\vec{\mathrm{x}})}{d\mathrm{x}_{i_{fb}}} + \sum_j \left(\frac{\partial \mathrm{F}_{j_{ff}}(\vec{\mathrm{x}})}{\partial \mathrm{x}_{i_{fb}}} \cdot \mathrm{T}_{j_{ff}}(\vec{\mathrm{x}}) + \frac{\partial \mathrm{F}_{j_{fb}}(\vec{\mathrm{x}})}{\partial \mathrm{x}_{i_{fb}}} \cdot \mathrm{T}_{j_{fb}}(\vec{\mathrm{x}}) \right) \tag{A.27}$$

since all terms are equal to zero, either directly by equation A.25, or since $E(\vec{\mathrm{x}})$ and $\mathrm{F}_{j_{ff}}(\vec{\mathrm{x}})$ are independent of $\mathrm{x}_{i_{fb}}$ by construction. Equation A.27 matches the form of the intrinsic gradient equation with $\mathrm{S}_{i_{fb}}(\vec{\mathrm{x}}, \vec{\mathrm{F}}(\vec{\mathrm{x}})) = 0$. Equations A.26 and A.27 together thus demonstrate that our constructed network, with dynamics on the feedforward units equivalent to that of the original network but with a linear training function, satisfies the intrinsic gradient equation with the given slack function.

A.5 Provably convergent dynamics

We can construct provably convergent dynamics for any homogeneous intrinsic gradient network satisfying assumptions (i) through (iii) of section 2.3.2 as well as

$$(iv) \quad \mathbf{T} = \mathbf{T}^\top$$
$$(v) \quad g(\vec{x}) - \vec{x}^\top \cdot \mathbf{T} \cdot \vec{x} \quad \text{is bounded above or below.}$$

In particular, these convergent dynamics apply to any solution to equation 2.17 satisfying assumptions (iv) and (v). Assumption (iv) implies assumption (iii), since $\left(\mathbf{T}^{-1}\right)^\top \cdot \mathbf{T} = \mathbf{I}$ if $\mathbf{T}$ is symmetric. If in addition $\mathbf{T}$ is positive-definite and $g(\vec{x})$ is bounded above, we will find that the dynamics of equation 2.2 are an instance of this class of provably convergent dynamics.

Our construction depends upon a scalar function $V(x)$, which serves as a Lyapunov function. Consider the scalar function

$$V(\vec{x}) = g(\vec{x}) - \frac{1}{2} \cdot \vec{x}^\top \cdot (\mathbf{D} + \mathbf{I}) \cdot \mathbf{T} \cdot \vec{x}\,, \tag{A.28}$$

for which

$$\nabla V(\vec{x}) = \nabla g(\vec{x}) - (\mathbf{D} + \mathbf{I}) \cdot \mathbf{T} \cdot \vec{x}$$

since $\mathbf{T}$ is symmetric by assumption (iv), so $\mathbf{D} = \left(\mathbf{T}^{-1}\right)^\top \cdot \mathbf{T} = \mathbf{I}$ and $\mathbf{D} + \mathbf{I} = 2 \cdot \mathbf{I}$, which commutes with other matrices. $V(\vec{x})$ is also a function of the parameters $\vec{w}$ since $g(\vec{x})$ is a function of $\vec{w}$, but we

omit mention of $\vec{w}$ to avoid cluttered notation. At the stationary points of $V(\vec{x})$, where $\nabla V(\vec{x}) = 0$,

$$\mathbf{T}^{-1} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \nabla V(\vec{x}) = 0$$

as well, since $\mathbf{T}$ is invertible by assumption and $\mathbf{D} + \mathbf{I} = 2 \cdot \mathbf{I}$ is clearly invertible. We can substitute equation A.28, the definition of $V(\vec{x})$, into this expression to obtain

$$\begin{aligned} 0 &= \mathbf{T}^{-1} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \nabla g(\vec{x}) - \mathbf{T}^{-1} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot (\mathbf{D} + \mathbf{I}) \cdot \mathbf{T} \cdot \vec{x} \\ &= \mathbf{T}^{-1} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \nabla g(\vec{x}) - \vec{x} \\ \vec{x} &= \mathbf{T}^{-1} \cdot (\mathbf{D} + \mathbf{I})^{-1} \cdot \nabla g(\vec{x}) = \vec{F}(\vec{x}) \,, \end{aligned} \tag{A.29}$$

where the last equality follows from the the definitions of $\vec{G}(\vec{x})$ and $g(x)$ in sections 2.3.3 and 2.3.4. As a result, any dynamics that converge to a stationary point of $V(\vec{x})$ also converge to a fixed point of $\vec{F}(\vec{x})$. Since every intrinsic gradient network that satisfies assumptions (i) through (iii) in section 2.3.2 conforms to the conservative vector field formulation, if such an intrinsic gradient network has a Lyapunov function $V(x)$, its output functions can be constructed from $V(x)$ via equations A.28 and A.29. Correspondingly, given the output functions $\vec{F}$ of an intrinsic gradient equation satisfying assumptions (i) through (iii) of section 2.3.2, its Lyapunov function (if it has one) must be defined by equation A.28.

Gradient ascent or descent on $V(\vec{x})$ is an instance of dynamics that converge to a stationary point of $V(\vec{x})$, so long as $V(\vec{x})$ is bounded above or below, respectively (i.e., assumption (v), since $\mathbf{D} = \mathbf{I}$ by assumption (iv)). It is easy to show that $V(\vec{x})$ is bounded above if $g(x)$ is the sum of the term $\vec{x}^{\top} \cdot (\mathbf{D} + \mathbf{I}) \cdot \mathbf{T} \cdot \vec{x}$ and a set of non-positive terms, such as $-(\vec{x} - \mathbf{W} \cdot \vec{x})^2$ for some matrix $\mathbf{W}$. Such a sum of terms satisfies equation 2.16, and so induces output functions of an intrinsic gradient network through equation 2.17. If $\mathbf{W}$ is a block matrix with all diagonals equal to zero except the diagonal directly above the main diagonal, the resulting intrinsic gradient network consists of layers of units which are trained to reconstruct each other using dictionaries defined by $\mathbf{W}$.

Alternatively, $V(\vec{x})$ is bounded above if $\mathbf{T}$ is positive-semidefinite and $g(\vec{x})$ is bounded above. These additional criteria are easy to satisfy within the class of intrinsic gradient networks discussed in this thesis. The pairwise permutation matrices often used for $\mathbf{T}$ can be made positive-semidefinite by adding a diagonal component proportional to $\mathbf{I}$. Functions $g(\vec{x})$ of the form of equation 2.16 are bounded above (by 0) if all functions $h_j^k(x)$ are bounded, either non-negative or non-positive, and for each k, the set $\left\{h_j^k(x) \mid j \neq \psi(k)\right\}$ has an odd number of non-positive elements. Bounded non-negative functions include sigmoids like the logistic function used in section 3.3. When these conditions hold, each summand of $g(\vec{x})$ in equation 2.16 is non-positive, since $x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} = x_{\psi(k)}^2 \geq 0$ given the assumption that $\mathbf{T}$ is symmetric, and this term is multiplied by a finite set of bounded

terms, an odd number of which are non-positive.

Moreover, if $\mathbf{T}$ is positive-definite, then the update direction of the dynamics of equation 2.2 always have a positive dot product with $\nabla V(\vec{x})$:

$$\begin{aligned}
[\nabla V(\vec{x})]^\top \cdot \left[\vec{F}(\vec{x}) - \vec{x}\right] &= \left[\nabla g(\vec{x}) - (\mathbf{D}+\mathbf{I}) \cdot \mathbf{T} \cdot \vec{x}\right]^\top \cdot \left[\mathbf{T}^{-1} \cdot (\mathbf{D}+\mathbf{I})^{-1} \cdot \nabla g(\vec{x}) - \vec{x}\right] \\
&= \left[\mathbf{T}^{-1} \cdot (\mathbf{D}+\mathbf{I})^{-1} \cdot \nabla g(\vec{x}) - \vec{x}\right]^\top \cdot \left[\mathbf{T} \cdot (\mathbf{D}+\mathbf{I})\right] \cdot \\
&\qquad \cdot \left[\mathbf{T}^{-1} \cdot (\mathbf{D}+\mathbf{I})^{-1} \cdot \nabla g(\vec{x}) - \vec{x}\right] \\
&= 2 \cdot \vec{y} \cdot \mathbf{T} \cdot \vec{y} \\
&> 0\,,
\end{aligned}$$

where $\vec{y} = \mathbf{T}^{-1} \cdot (\mathbf{D}+\mathbf{I})^{-1} \cdot \nabla g(\vec{x}) - \vec{x} = \vec{F}(\vec{x}) - \vec{x}$, the second line follows because $\mathbf{T}$ is symmetric, the third line follows because $\mathbf{D}+\mathbf{I} = 2 \cdot \mathbf{I}$, and the fourth line follows because $\mathbf{T}$ is positive-definite by assumption. We can thus see that $V(\vec{x})$ is a Lyapunov function for equation 2.2, and these dynamics are guaranteed to converge.

Thus far in this section, we have constructed Lyapunov functions for homogeneous intrinsic gradient networks for which $\nabla E = 0$. Since the intrinsic gradient equation (2.9) is linear in $\vec{F}(\vec{x})$ given assumptions (i) and (ii) of section 2.3.2, full intrinsic gradient networks are the sum of a homogeneous and an inhomogeneous part, as discussed in section 2.3.1. Fortunately, when the error function consists of a sum of terms, each of which is only a function of a single unit x_i (as in section 2.4.4), the Lyapunov function can easily be modified to accommodate the entire network, rather than just the homogeneous part, by adding in an inhomogeneous component. Specifically, the term

$$V_E(\vec{x}) = 2 \cdot \sum_i \int G_i(\vec{x}) \cdot dx_i$$

should be added to $V(\vec{x})$ as defined by equation A.28, where $G_i(\vec{x}) = -x_i \cdot \int \left(x_i^{-2} \cdot \frac{\partial E(\vec{x})}{\partial x_i} \cdot dx_i\right)$ as in equation 2.19. The term $V_E(\vec{x})$ is defined so that

$$\begin{aligned}
\mathbf{T}^{-1} \cdot (\mathbf{D}+\mathbf{I})^{-1} \cdot \nabla V_E(\vec{x}) &= \mathbf{T}^{-1} \cdot \vec{G}(\vec{x}) \\
&= \vec{F}_E(\vec{x})\,,
\end{aligned}$$

where $F_E(\vec{x})$ is a solution to the inhomogeneous part of the intrinsic gradient equation. The first line follows from the fundamental theorem of calculus, since our assumptions imply that $\mathbf{D}+\mathbf{I} = 2 \cdot \mathbf{I}$, and because each $G_i(\vec{x})$ is only a function of the single variable x_i. The second line follows from the definition of $\vec{G}(\vec{x})$ in section 2.3.3 and the derivation in section 2.4.4. An analogous derivation to that in equation A.29 shows that the stationary points of $V(\vec{x}) + V_E(\vec{x})$ are identical to the fixed points of $\vec{F}(\vec{x}) + \vec{F}_E(\vec{x})$, so $V(\vec{x}) + V_E(\vec{x})$ is a Lyapunov function for the full inhomogeneous intrinsic

gradient network if it is bounded above or below.

For instance, if $E(\vec{x}) = \vec{c}^{\top} \cdot \vec{x}$ for some constant vector $\vec{c}$, then the term $\sum_i 2 \cdot x_i \cdot c_i$ should be added to $V(\vec{x})$ if we choose all constants of integration to be zero. If we use the same linear error function, but choose the constant of integration in the definition of $\vec{G}(\vec{x})$ to be 1, and the constant of integration in the definition of $V_E(\vec{x})$ to be $-\frac{c_i^2}{2}$, then the term $-\sum_i (c_i - x_i)^2$ should be added to $V(\vec{x})$. In this case, the fixed points of the intrinsic gradient network minimize the sum of squares error between $\vec{c}$ and $\vec{x}$, whereas following the gradient of the error function minimizes $\vec{c}^{\top} \cdot \vec{x}$ at those fixed points. On the other hand, if we use the negative sum of squares error $E(\vec{x}) = \frac{1}{2} \cdot \sum_i (x_i - c_i)^2$ directly, then the term $\sum_i 2 \cdot x_i \cdot c_i + \frac{1}{2} \cdot x_i^2 \cdot (2 \cdot \log(|x_i|) - 1)$ should be added to $V(\vec{x})$.

Even if $V(\vec{x})$ is not bounded, it can be used to induce a probabilistic interpretation for the associated intrinsic gradient network. Specifically, we can define

$$P(\vec{x}) = \frac{1}{Z} \cdot m(V(\vec{x}))\,, \tag{A.30}$$

where $m(x)$ is a non-negative monotonically increasing function such as $(1 + e^{-x})^{-1}$, and $Z = \int m(V(\vec{x}))d\vec{x}$.[2] The output states that correspond to (local) maxima of $V(\vec{x})$ are then (local) maximum *a posteriori* configurations of this probability distribution. In particular, when we use the linear error function $E(\vec{x}) = \vec{c}^{\top} \cdot \vec{x}$ and choose the constants of integration appropriately, maximizing the *a posteriori* probability of a configuration also minimizes the Euclidean distance between $\vec{c}$ and $\vec{x}$.

If we use the negative sum of squares error $E(\vec{x}) = -\frac{1}{2} \cdot \sum_i (x_i - c_i)^2$, then while the dynamics of such an intrinsic gradient network find locally maximal *a posteriori* network states $\vec{x}$ (given fixed parameters $\vec{w}$) according to the distribution of equation A.30, gradient descent on the error function E maximizes the log likelihood of these locally maximal *a posteriori* points according to a Gaussian distribution centered at $\vec{c}$: $P(\vec{x}) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot e^{-\sum_i (x_i - c_i)^2}$. Training via gradient descent on the error function E assumes that the dynamics remain at a fixed point, and thus at a locally maximal *a posteriori* configuration of equation A.30. Unlike algorithms based on expectation maximization, training does not ignore the effect that modifying the parameters has on the units themselves. At the same time, though, it does not maximize the probability assigned to the locally maximal *a posteriori* configuration using equation A.30; it minimizes $E(\vec{x})$ at a maximum of $V(\vec{x})$, rather than maximizing $V(\vec{x})$ itself.

[2] $P(x)$ only constitutes a probability distribution if the integral defining Z converges.

A.6 Conservative vector field solution

Plugging equation 2.16 into equation 2.15,

$$
\begin{aligned}
g(\vec{x}) &= c + \left[\nabla^\top \left(c + \sum_k \left[x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot \prod_{j\neq\psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}}\right)\right]\right)\right] \cdot (\mathbf{D}+\mathbf{I})^{-1} \cdot \mathbf{D} \cdot \vec{x} \\
&= c + \sum_k \left[\left(\nabla^\top \left[x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot \prod_{j\neq\psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}}\right)\right]\right) \cdot (\mathbf{D}+\mathbf{I})^{-1} \cdot \mathbf{D} \cdot \vec{x}\right] \\
&= c + \sum_k \left[\sum_i \left(\frac{\partial}{\partial x_i}\left[x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot \prod_{j\neq\psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}}\right)\right] \cdot \frac{D_i}{D_i+1} \cdot x_i\right)\right] \\
&= c + \sum_k \left[\left(\frac{D_{\psi(k)}+1}{D_{\psi(k)}} \cdot x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}-1} \cdot \prod_{j\neq\psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}}\right)\right.\right. \\
&\qquad - \sum_{i\neq\psi(k)} \left[x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot h_i^{k\prime}\left(\frac{x_i^{\frac{D_i+1}{D_i}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}}\right) \cdot \frac{D_{\psi(k)}+1}{D_{\psi(k)}} \cdot \frac{x_i^{\frac{D_i+1}{D_i}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}+1}} \cdot\right. \\
&\qquad \left.\left. \cdot \prod_{j\neq i,\psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}}\right)\right]\right) \cdot \frac{D_{\psi(k)}}{D_{\psi(k)}+1} \cdot x_{\psi(k)} \\
&\qquad + \sum_{i\neq\psi(k)} \left(\left[x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot h_i^{k\prime}\left(\frac{x_i^{\frac{D_i+1}{D_i}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}}\right) \cdot \frac{D_i+1}{D_i} \cdot \frac{x_i^{\frac{D_i+1}{D_i}-1}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}} \cdot\right.\right. \\
&\qquad \left.\left. \cdot \prod_{j\neq i,\psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}}\right)\right] \cdot \frac{D_i}{D_i+1} \cdot x_i\right)\right] \\
&= c + \sum_k x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}} \cdot \prod_{j\neq\psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}}\right)
\end{aligned}
$$

which matches equation 2.16, as desired. The first line of the fourth equation follows from taking the partial derivative of $x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}$ with respect to $x_{\psi(k)}$. The second line of the fourth equation follows from taking the partial derivative of $\prod_{j\neq\psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}}\right)$ with respect to $x_{\psi(k)}$. The third line

of the fourth equation follows from taking the partial derivative of $\prod_{j \neq \psi(k)} h_j^k \left(\frac{x_j^{\frac{D_j+1}{D_j}}}{x_{\psi(k)}^{\frac{D_{\psi(k)}+1}{D_{\psi(k)}}}} \right)$ with respect to all x_i other than $x_{\psi(k)}$. The second and the third line cancel out exactly, leaving only the desired terms from the first line.

A.7 Generalized polynomial assumption

Instead of the conservative vector field formulation of section 2.3.4, assume that $F_i(\vec{x}) = \sum_j w_{ij}^F \cdot \prod_k x_k^{v_{jk}}$, where $w_{ij}^F, v_{jk} \in \mathbb{R}$ for all i, j, and k. In the term w_{ij}^F, F is part of the name of the parameter, rather than an exponent. There can be an infinite number of terms in the sum, so long as w_{ij}^F is bounded for all i and j, and for each k, v_{jk} is enumerable over j in a monotonic manner. Intuitively, we wish to consider output functions F_i that can be written as a polynomial series on the elements of $\vec{x}$, generalized to allow non-integral exponents. This class of functions, which we refer to as *generalized polynomials*, includes all functions expressible by a Maclaurin series. We continue to assume that $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, and that $\vec{T}(\vec{x})$ is a linear, invertible function of $\vec{x}$, independent of $\vec{w}$, with $\left(\mathbf{T}^{-1}\right)^\top \cdot \mathbf{T} = \mathbf{D}$, so equation 2.13 still applies. Using the definition $\vec{G}(\vec{x}) = \mathbf{T} \cdot \vec{F}(\vec{x})$, the generalized polynomial assumption implies

$$G_i(\vec{x}) = \sum_j w_{ij}^G \cdot \prod_k x_k^{v_{jk}}, \tag{A.31}$$

where $w_{ij}^G = \sum_k T_{ik} \cdot w_{kj}^F$ is bounded if w_{ij}^F is bounded. As before, G is part of the name of the parameter w_{ij}^G, rather than an exponent.

It can be seen that, when $\vec{G}$ and ∇E are generalized polynomials, equation 2.13 implies that for each entry of the vectors, the coefficient of each summand from the left-hand side of equation 2.13 is equal to the coefficient of the corresponding summand on the right-hand side. This is true even if non-integral exponents are allowed. If v_{jk} is monotonically enumerable over j, then the exponents of the right-hand side of equation 2.13 are also monotonically enumerable. This follows since each summand of $G_i(\vec{x})$ gives rise to no more than one summand for each entry of $\left(\nabla \vec{G}^\top(\vec{x})\right) \cdot \mathbf{D} \cdot \vec{x}$, and the exponents of these corresponding summands differ by a fixed constant. Consider an enumeration of the summands comprising one entry of the left-hand side of equation 2.13, such that the minima of the exponents of the summands are monotonically increasing, and a similar enumeration of the corresponding entry of the right-hand side of equation 2.13. The first elements of these two enumerations, as well as the coefficients of the corresponding terms, must be equal; these terms have the smallest exponent and thus dominate the behavior of the two generalized polynomials for small x_i. These matching terms can then be subtracted from both sides, and the process repeated. Since the exponents and thus the terms are enumerable, any given term will

be subject to this matching procedure after only a finite number of iterations. A complementary argument applies when the enumeration of v_{jk} over j is monotonically decreasing, but in this case the coefficients of the term with the largest exponent must match, since it dominates the behavior for large x_i.

Given this relationship between the transformed output functions $G_i(\vec{x})$, it is profitable to rewrite equation A.31 in the form

$$G_i(\vec{x}) = \sum_n w_i^n \cdot \prod_j x_j^{v_j^n - \delta_{ij}}, \tag{A.32}$$

where n indexes both w_i^n and v_j^n, rather than serving as an exponent. We refer to the set of monomials with index n associated with different transformed output functions G_i as a *monomial family*. We sometimes write $\vec{v}^n$ for the vector of exponents associated with monomial family n. As before, δ is the Kronecker delta.

The coefficients w_i^n are trainable parameters as before, but indexed by both transformed output function i and monomial family n. The exponents v_j^n are untrained constants, also indexed by both transformed output function j and monomial family n. They further parameterize the transformed output functions beyond the original $\vec{w}$. As we shall explore in detail, equation 2.13 restricts the coefficient w_i^n of transformed output function G_i and monomial family n in terms of other coefficients w_j^n in the same monomial family n, but w_i^n is independent of other monomial families. The trained parameters $\vec{w}$ and the untrained $\vec{v}$ are also linked, as shall be detailed below.

The exponents v_j^n in equation A.32 are the same regardless of which transformed output function G_i is under consideration. That is, there is single set of interrelated monomial forms held in common between all of the transformed output functions. Nevertheless, the class of functions described by equation A.32 captures the full set of functions analytic around zero, since any polynomial can be formed by setting $w_i^n = 0$ for unwanted monomials. It also captures some functions, like fractional exponents of x, that are not analytic around zero.

When the form of equation A.32 is substituted into equation 2.13, the following linear equation on the parameters results for all output functions i and monomial families n:

$$w_i^n = \left(\frac{\partial E}{\partial x_i}\right)_{(n)} + \sum_j (v_i^n - \delta_{ij}) \cdot D_j \cdot w_j^n \tag{A.33}$$

where $\left(\frac{\partial E}{\partial x_i}\right)_{(n)}$ is the coefficient of the term of $\frac{\partial E}{\partial x_i}$ with the form of monomial family n for output function i; that is, the term in which unit x_j has exponent $v_j^n - \delta_{ij}$ for all j. The set of linear equations specified by equation A.33 for the coefficients of a single monomial family can be written

in matrix form as

$$\begin{aligned} \vec{w}^n &= (\nabla E)_{(n)} + \mathbf{M} \cdot \mathbf{D} \cdot \vec{w}^n \\ (\mathbf{I} - \mathbf{M} \cdot \mathbf{D}) \cdot \vec{w}^n &= (\nabla E)_{(n)} \end{aligned} \tag{A.34}$$

where

$$\mathbf{M} = \begin{bmatrix} (v_i^n - 1) & v_i^n & \cdots & v_i^n \\ v_j^n & \left(v_j^n - 1\right) & \cdots & v_j^n \\ \vdots & \vdots & \ddots & \vdots \\ v_z^n & v_z^n & \cdots & (v_z^n - 1) \end{bmatrix} = \begin{bmatrix} v_i^n \\ v_j^n \\ \vdots \\ v_z^n \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} - \mathbf{I},$$

and $\vec{w}^n$ is the column vector with elements w_i^n for all $x_i \in \vec{x}$. Clearly, if $\mathbf{I} - \mathbf{M} \cdot \mathbf{D}$ is invertible, the vector $\vec{w}^n$ is only consistent with equation A.33 if

$$\vec{w}^n = (\mathbf{I} - \mathbf{M} \cdot \mathbf{D})^{-1} \cdot (\nabla E)_{(n)} . \tag{A.35}$$

That is, the parameters are completely determined by the error function, so all parameters are input parameters and no learning is possible.

The matrix $\mathbf{I} - \mathbf{M} \cdot \mathbf{D}$ is singular if and only if there exists some vector $\vec{u} \neq 0$ such that $(\mathbf{I} - \mathbf{M} \cdot \mathbf{D}) \cdot \vec{u} = 0$. However,

$$(\mathbf{I} - \mathbf{M} \cdot \mathbf{D}) \cdot \vec{u} = (\mathbf{I} + \mathbf{D}) \cdot \vec{u} - \vec{v}^n \cdot \left(\sum_i D_i \cdot u_i \right), \tag{A.36}$$

so if $\mathbf{I} - \mathbf{M} \cdot \mathbf{D}$ is singular, then $(\mathbf{I} + \mathbf{D}) \cdot \vec{u}$ must be parallel to $\vec{v}^n$. That is, $\vec{u} = \alpha \cdot (\mathbf{I} + \mathbf{D})^{-1} \cdot \vec{v}^n$ for some scalar α, which when substituted back into equation A.36 implies that there exists a $\vec{u}$ such that $(\mathbf{I} - \mathbf{M} \cdot \mathbf{D}) \cdot \vec{u} = 0$ if and only if

$$\sum_{i | D_i \neq -1} \frac{D_i}{1 + D_i} \cdot v_i^n = 1, \tag{A.37}$$

and $v_i = 0$ for all i such that $D_i = -1$. The matrix $\mathbf{I} + \mathbf{D}$ is diagonal, so its inverse is easy to construct when it exists. The choice of $\mathbf{D}$ thus restricts all v_i^n. For instance, equation A.37 implies that v_i^n cannot be unbounded and strictly positive if $\mathbf{D}$ is strictly positive.

If equation A.37 is not satisfied, $\mathbf{I} - \mathbf{M} \cdot \mathbf{D}$ is invertible, and the coefficients for monomial family n are fixed by the error function; the coefficients are all input parameters and cannot be trained, by gradient descent or otherwise. In particular, if $(\nabla E)_{(n)} = 0$, then $\vec{w}^n = 0$. In contrast, if $\mathbf{I} - \mathbf{M} \cdot \mathbf{D}$ is not invertible but equation A.34 is satisfiable, $\vec{w}^n$ may have a component of arbitrary magnitude

in the nullspace of $\mathbf{I} - \mathbf{M} \cdot \mathbf{D}$. Thus, $\vec{w}^n = \vec{w}^n_{input} + \vec{w}^n_{internal}$, where

$$\vec{w}^n_{input} = (\mathbf{I} - \mathbf{M} \cdot \mathbf{D})^+ \cdot (\nabla E)_{(n)} \tag{A.38}$$

and $\mathbf{N}^+$ is the Moore-Penrose pseudoinverse of matrix $\mathbf{N}$, and

$$(w^n_{internal})_i = \alpha_n \cdot (1 + D_i)^{-1} \cdot v^n_i \tag{A.39}$$

for some scalar α_n when $D_i \neq -1$, and $(w^n_{internal})_i = 0$ otherwise. The matrix $\mathbf{I} - \mathbf{M} \cdot \mathbf{D}$ never has more than one eigenvector with eigenvalue equal to zero, since only one vector $\vec{u}$ satisfies equation A.36, modulo a scaling factor.

Any set of output functions of the form of equation A.32, satisfying the restrictions of equations A.37 and A.39, is also a valid solution to equation 2.17. Consider a monomial family of equation A.32 with exponents v^n_i and coefficients w^n_i. Equations 2.17 and A.32 are of approximately the same form, and it can be seen that the exponents of all output functions in the conservative vector field formulation will match those of the generalized polynomial formulation if there is a bijection between the summands of $g(x)$ and the monomial families. A summand of $g(x)$ matching a chosen monomial family can be constructed within the framework of equation 2.17 by choosing $\psi(k) = i$ for one element of the monomial family, $h^k_j(x) = x^{\frac{D_j}{D_j+1} \cdot v^n_j}$ for the other elements of the monomial family, and $h^k_j(x) = 1$ otherwise. The exponent of the elements of the family other than i are clearly correct. Equation 2.17 implies that the exponent of element i is

$$v^n_i = \frac{D_i + 1}{D_i} \cdot \left(1 - \sum_{j \neq i} \frac{D_j}{D_j + 1} \cdot v^n_j\right)$$

which is exactly the value obtained by solving equation A.37 for v^n_i. Once the exponents match, it can immediately be seen that the coefficients generated by equation 2.17 are exactly those of equation A.39, with w_k playing the role of α.

A.8 Belief propagation on an acyclic factor graph is an intrinsic gradient network

We shall show that belief propagation on acyclic factor graphs satisfies the intrinsic gradient equation (2.9) with the negative log likelihood of a configuration of the observed variables as the error function (where by configuration we mean an assignment of a value to each of the indicated random variables). The requisite slack function will not be $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$, unlike section 2.2; the corresponding training function $\vec{T}$ will not be linear, unlike section 2.3; and the resulting intrinsic

gradient network will only be compatible with acyclic connection topologies.

We first review the definition and relevant properties of factor graphs and belief propagation in section A.8.1. We then demonstrate in section A.8.2 that the negative log likelihood can be defined in terms of the belief propagation messages on an acyclic factor graph, and its gradient can be calculated directly from these messages. The necessity condition of the intrinsic gradient equation, discussed in appendix A.1, thus implies that belief propagation on an acyclic factor graph constitutes an intrinsic gradient network, but the identity of the corresponding slack function $\vec{S}$ and training function $\vec{T}$ are not obvious. In section A.8.3 we construct a linear training function $\vec{T}$ and show that it satisfies the intrinsic gradient equation with output functions corresponding to the belief propagation messages and an error function related to the likelihood, rather than the log likelihood. However, the inputs in this network do not match those in belief propagation. Finally, in section A.8.4 we augment the training function with a nonlinear component, and show that it satisfies the intrinsic gradient equation with belief propagation dynamics and the negative log likelihood as the error function.

A.8.1 Definition of factor graphs and belief propagation

A factor graph defines a probability distribution over a vector of discrete random variables $\vec{V}$, with values $v_i \in \mathcal{V}_i$, in terms of a vector of functions $\vec{f}$ indexed by a, with $f_a : \vec{\mathcal{V}} \to \mathbb{R}^+$ (reviewed in Kschischang et al., 2001; Yedidia et al., 2005). The set of random variables $\vec{V}$ and the set of functions $\vec{f}$ are generally not in correspondence, and the sets of indices i and a are disjoint. Each function $f_a(\vec{v})$ only depends directly on a subset of the random variables V_i, which we call the *neighborhood of* f_a, or $N(a)$. We also denote the set of indices i such that v_i is an argument of $f_a(\vec{v})$ by $N(a)$; the correct interpretation will be clear from the context. We indicate a configuration over $N(a)$ by $\vec{v}_a \in \vec{\mathcal{V}}_a$; that is, $\vec{v}_a$ specifies the value v_i of random variable V_i for all $i \in N(a)$. Correspondingly, we refer to the set of functions $f_a(\vec{v})$ for which v_i is an argument as the *neighborhood of* V_i, or $N(i)$. Once again, we also use $N(i)$ to denote the set of indices a such that v_i is an argument of $f_a(\vec{v})$.

The probability of a configuration $\vec{v} \in \vec{\mathcal{V}}$ of the variables $\vec{V}$ is defined by

$$P(\vec{v}) = \frac{\prod_a f_a(\vec{v})}{\sum_{\vec{\nu} \in \vec{\mathcal{V}}} \prod_a f_a(\vec{\nu})} \,.$$

The functions $\vec{f}$ are factors of the numerator of the probability distribution, so the functions $\vec{f}$ are called *factors*, and these models are called *factor graphs*. This probability distribution can be represented by a graph with a circular *variable node* for each random variable V_i, a square *factor node* for each factor $f_a(\vec{v})$, and an undirected edge connecting each factor node to the variable nodes in its neighborhood and vice versa (Kschischang et al., 2001). A factor graph is *acyclic* if the associated graph has no loops, as in figure A.1.

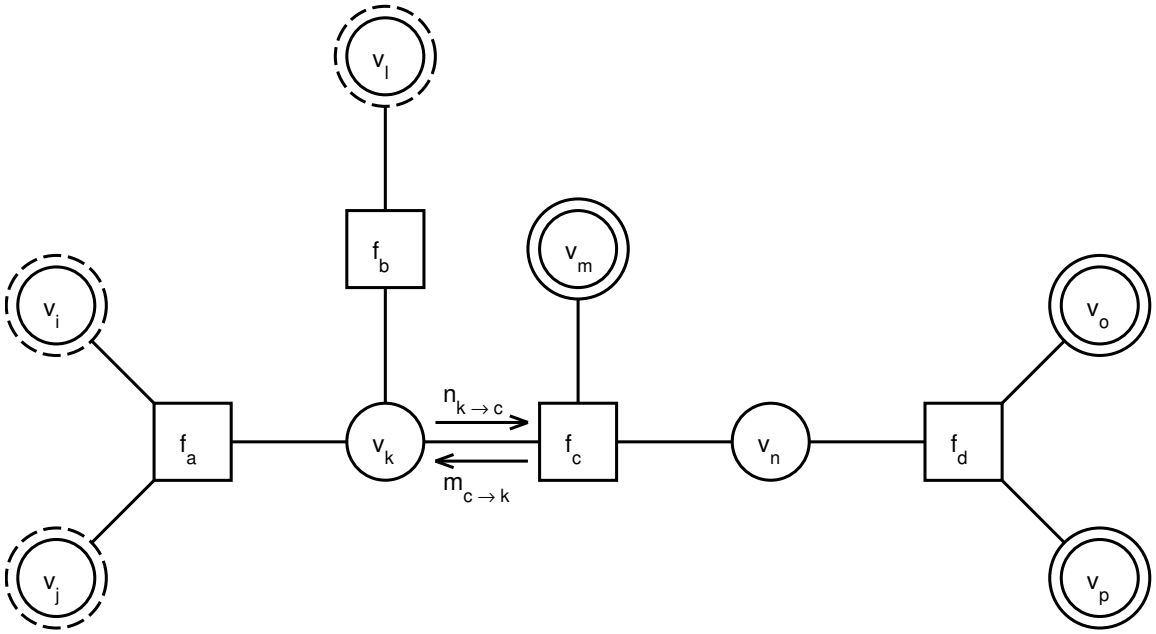


Figure A.1: An acyclic factor graph. Variables are denoted by circles; factors are denoted by squares. Observed variables are marked by a double circle, whereas hidden variables are indicated by a single circle. The observed variables behind the message $n_{k\to c}$ have a dashed double circle, whereas the observed variables behind the message $m_{c\to k}$ have a solid double circle.

Belief propagation is a set of message-passing dynamics capable of calculating the marginal probabilities of the random variables in an acyclic factor graph (Pearl, 1988; Aji & McEliece, 2000; Kschischang et al., 2001). Belief propagation has been discovered independently in multiple disciplines, and is also known as the sum-product algorithm and the forward-backward algorithm. In belief propagation, a distinct message $n_{i\to a}(v_i) \in \mathbb{R}^+$ is passed from variable node i to factor node a for each possible value $v_i \in \mathcal{V}_i$ of random variable V_i, and a separate message $m_{a\to i}(v_i) \in \mathbb{R}^+$ is passed from factor node a to variable node i for each value $v_i \in \mathcal{V}_i$ of random variable V_i, as depicted in figure A.1. Intuitively, the message $n_{i\to a}(v_i)$ communicates the degree to which variable node i believes that variable V_i has value v_i (excluding the evidence from factor node a), and message $m_{a\to i}(v_i)$ communicates the degree to which factor node a believes that variable V_i has value v_i (excluding the evidence from variable node i). These messages are updated according to the dynamics

$$\begin{array}{lcll} n_{i\to a}(v_i) & \Leftarrow & \prod_{b\in N(i)\setminus a} m_{b\to i}(v_i) & \text{when } a \in N(i) \\ m_{a\to i}(v_i) & \Leftarrow & \sum_{\vec{v}_a|v_i} f_a(\vec{v}_a) \cdot \prod_{j\in N(a)\setminus i} n_{j\to a}(v_j) & \text{when } i \in N(a)\,, \end{array} \tag{A.40}$$

where $N(i) \setminus a$ denotes the set of indices of all the factors that are neighbors of variable V_i except for factor f_a, $N(a) \setminus i$ denotes the set of indices of all variables that are neighbors of factor f_a except for V_i, $\sum_{\vec{v}_a|v_i}$ denotes a sum over all the configurations $\vec{v}_a$ of the variables that are arguments of f_a, such that V_i takes the value v_i, and the value of v_j in the second equation matches that in $\vec{v}_a$.

The random variables $\vec{V}$ can be divided into two subgroups: the observed variables $\vec{V}^{obs}$, which are intended to model an externally defined probability distribution, and the hidden variables $\vec{V}^{hid}$, which are internal to the factor graph and help to construct the distribution over the observed variables. We assume that all internal variable nodes correspond to hidden variables, and all leaf variable nodes correspond to observed variables; any acyclic factor graph can be converted into this form. Since each configuration $\vec{v}$ of all variables corresponds to a configuration $\vec{v}^{obs}$ of the observed variables and a configuration $\vec{v}^{hid}$ of the hidden variables, $P(\vec{v}) = P(\vec{v}^{obs}, \vec{v}^{hid})$ and $P(\vec{v}^{obs}) = \sum_{\vec{v}^{hid}} P(\vec{v}^{obs}, \vec{v}^{hid})$. Consider a probability distribution of interest (such as a finite data set, consisting of observations of the outside world, in which all elements are assigned uniform probability, and all other configurations are assigned zero probability) and a factor graph that models it, such that each element of the probability distribution of interest assigns a value to each observed variable of the factor graph. The sum of the negative log likelihoods assigned by the factor graph to each configuration of the observed variables, weighted by the desired probability of the configurations, is a commonly used error function for training such factor graphs (Yedidia et al., 2005). It is equivalent up to a constant to the Kullback-Leibler divergence between the desired probability distribution and that defined by the factor graph.

A.8.2 The negative log likelihood and its gradient can be calculated by belief propagation

It is not initially obvious that belief propagation on acyclic factor graphs has any relationship to intrinsic gradient networks. We shall show that the negative log likelihood constitutes an error function, defined in terms of the belief propagation messages, for which the gradient can be calculated from the converged belief propagation messages. The necessity condition for intrinsic gradient networks, developed in appendix A.1, then implies that belief propagation on acyclic factor graphs is an intrinsic gradient network.

The negative log likelihood of a configuration of the observed variables can be calculated for an acyclic factor graph directly from two sets of converged belief propagation messages: one with the messages from the observed variable nodes set based upon the observed values, and the other with the messages from the observed variable nodes set to be uniform and independent of the observed values (Ackley et al., 1985; Kschischang et al., 2001). A simple derivation, replicated below, demonstrates that the gradient of this negative log likelihood, $-\frac{d \log(P(\vec{v}^{obs}))}{d\vec{w}}$, can be calculated from the marginal probabilities of the groups of variables connected to each factor. In an acyclic factor graph, these marginal probabilities can also be calculated by performing belief propagation to convergence twice; once with the messages from the observed variable nodes set based upon the observed values, and once with the messages from the observed variable nodes set to be uniform and independent of

the observed values. We note that these two applications of belief propagation are analogous, respectively, to the wake and sleep stage of Boltzmann machines.

Since an error function and its gradient can be calculated from the intrinsic signals at the fixed point, the necessity condition discussed in appendix A.1 implies that belief propagation on an acyclic factor graph must be an instance of an intrinsic gradient network. In this subsection, we confirm that both the negative log likelihood and its gradient can be calculated from belief propagation messages in acyclic factor graphs. We then map belief propagation on acyclic factor graphs onto the intrinsic gradient formalism in the following two subsections.

The likelihood (and from it the negative log likelihood) of a configuration of the observed variables in an acyclic factor graph can be computed by

$$P(\vec{v}^{obs}) = \frac{\prod_{a \in N(i)} m^{obs}_{a \to i}(v_i^{obs})}{\sum_{\nu_i \in \mathcal{V}_i} \prod_{a \in N(i)} m^{all}_{a \to i}(\nu_i)} \tag{A.41}$$

for any i such that V_i is an observed variable, as can be seen from consideration of the distributive law (Aji & McEliece, 2000; Kschischang et al., 2001). The messages $\vec{n}^{obs}$ and $\vec{m}^{obs}$ result from performing belief propagation to convergence with the messages from each observed variable node i set such that $n^{obs}_{i \to a}(\nu_i) = 1$ if the value of $\nu_i \in \mathcal{V}_i$ matches the value in configuration $\vec{v}^{obs}$, and $n^{obs}_{i \to a}(\nu_i) = 0$ otherwise; the terms $\vec{n}^{all}$ and $\vec{m}^{all}$ result from performing belief propagation to convergence with all messages from the observed variable nodes set equal to 1. Qualitatively, equation A.41 corresponds to combining the beliefs in the observed variables from all sources, and normalizing by the potential degree of belief in all possible combinations of observed variables. We can eliminate the need to choose a particular observed variable by summing over all observed variables:

$$P(\vec{v}^{obs}) = \frac{\sum_{i | V_i \text{is observed}} \prod_{a \in N(i)} m^{obs}_{a \to i}(v_i^{obs})}{\sum_{i | V_i \text{is observed}} \sum_{\nu_i \in \mathcal{V}_i} \prod_{a \in N(i)} m^{all}_{a \to i}(\nu_i)} \,. \tag{A.42}$$

The first sum in the numerator and denominator is over all observed variables, whereas the second sum in the denominator is over all values of the chosen variable. The negative log likelihood can easily be calculated by simply taking the additive inverse of the logarithm of this calculation, thus demonstrating that the negative log likelihood of a configuration of the observed variables in an acyclic factor graph can be calculated from the belief propagation messages at a fixed point.

The gradient of the negative log likelihood can also be calculated from the belief propagation messages at a fixed point. Consider the most general parameterization of the factors, where each possible output of each function $f_a(\vec{v}_a)$ is independently parameterized. Specifically, let $f_{a'}(\vec{u}_{a'}) = w'$ for some factor $f_{a'}$ and configuration of its neighborhood $\vec{u}_{a'} \in \vec{\mathcal{V}}_{a'}$. This configuration can be divided up into $\vec{u}_{a'} = \left(\vec{u}^{obs}_{a'}, \vec{u}^{hid}_{a'}\right)$, where $\vec{u}^{obs}_{a'}$ is a configuration of the observed variables in $N(a')$, and $\vec{u}^{hid}_{a'}$ is a configuration of the hidden variables in $N(a')$. The derivative of the log likelihood of

a configuration $\vec{v}^{obs}$ of the observed variables with respect to w' is then

$$\begin{aligned}
\frac{d\log(P(\vec{v}^{obs}))}{dw'} &= \frac{d}{dw'}\log\left(\sum_{\vec{\nu}^{hid}\in\vec{\mathcal{V}}^{hid}} P(\vec{v}^{obs},\vec{\nu}^{hid})\right) \\
&= \frac{d}{dw'}\log\left(\frac{\sum_{\vec{\nu}^{hid}\in\vec{\mathcal{V}}^{hid}}\prod_a f_a(\vec{v}^{obs},\vec{\nu}^{hid})}{\sum_{\vec{\nu}\in\vec{\mathcal{V}}}\prod_a f_a(\vec{\nu})}\right) \\
&= \delta_{\vec{v}^{obs}_{a'}\vec{u}^{obs}_{a'}}\cdot\frac{\sum_{\vec{\nu}^{hid}\in\vec{\mathcal{V}}^{hid}}\delta_{\vec{\nu}^{hid}_{a'}\vec{u}^{hid}_{a'}}\cdot\prod_{a\neq a'} f_a(\vec{v}^{obs},\vec{\nu}^{hid})}{\sum_{\vec{\nu}^{hid}\in\vec{\mathcal{V}}^{hid}}\prod_a f_a(\vec{v}^{obs},\nu^{hid})} \\
&\quad - \frac{\sum_{\vec{\nu}\in\vec{\mathcal{V}}}\delta_{\vec{\nu}_{a'}\vec{u}_{a'}}\cdot\prod_{a\neq a'} f_a(\vec{\nu})}{\sum_{\vec{\nu}\in\vec{\mathcal{V}}}\prod_a f_a(\vec{\nu})} \\
&= \frac{1}{f_{a'}(\vec{u}_{a'})}\cdot\left(P(\vec{u}_{a'}|\vec{v}^{obs}) - P(\vec{u}_{a'})\right)\ , \qquad \text{(A.43)}
\end{aligned}$$

where $\delta_{\vec{\nu}^{hid}_{a'}\vec{u}^{hid}_{a'}} = 1$ if $\nu_i^{hid} = u_i^{hid}$ for all hidden variables $V_i \in N(a')$, and 0 otherwise; $\delta_{\vec{\nu}^{obs}_{a'}\vec{u}^{obs}_{a'}}$ is defined analogously; and $\delta_{\vec{\nu}_{a'}\vec{u}_{a'}} = 1$ if $\nu_i = u_i$ for all variables $V_i \in N(a')$, and 0 otherwise. We can easily calculate the gradient of the negative log likelihood with respect to the parameters using the converged belief propagation messages, since the required marginal probabilities can themselves be calculated from the converged belief propagation messages (Kschischang et al., 2001):

$$P(\vec{u}_{a'}|\vec{v}^{obs}) = \frac{f_{a'}(\vec{u}_{a'})\cdot\prod_{i\in N(a')} n^{obs}_{i\to a'}(u_i)}{\sum_{\vec{\nu}_{a'}\in\vec{\mathcal{V}}_{a'}} f_{a'}(\vec{\nu}_{a'})\cdot\prod_{i\in N(a')} n^{obs}_{i\to a'}(\nu_i)}\ . \qquad \text{(A.44)}$$

Qualitatively, equation A.44 corresponds to combining the beliefs in the selected variable values $\vec{u}_{a'}$ from all sources, assuming that the observed variables take on the specified values, and normalizing by the belief in all possible combinations of these variables. The same function of the converged belief propagation messages computes $P(\vec{u}_{a'})$ when using $n^{all}_{i\to a'}$ in place of $n^{obs}_{i\to a'}$.

An intrinsic gradient network is defined by the ability to calculate the gradient of an error function from the intrinsic signals. Since both the negative log likelihood and its gradient are calculable from the belief propagation messages after convergence, belief propagation on an acyclic factor graph constitutes an intrinsic gradient network, with the negative log likelihood serving as the error function.[3] However, it is not obvious that these dynamics satisfy the intrinsic gradient equation (2.9), even though the necessity condition of the intrinsic gradient equation, discussed in appendix A.1, implies that a slack function $\vec{S}(\vec{x})$ and a training function $\vec{T}(\vec{x})$ satisfying equation 2.7 must exist. We attempt to construct an appropriate training function in section A.8.3, and finally succeed in section A.8.4. We do not explicitly derive an analytical form for the slack function, but it follows trivially from the intrinsic gradient equation (2.9) after we construct the training function.

[3]Technically, an acyclic factor graph must be duplicated into a parallel wake and sleep network, each with independent belief propagation dynamics, in order to be an intrinsic gradient network.

A.8.3 An intrinsic gradient network with a linear training function which approximates belief propagation

Using equation A.42, the negative log likelihood can be split into two separate component error functions:

$$\begin{aligned}
-\log(P(\vec{v}^{obs})) &= -\log\left(\frac{\sum_{i|V_i\text{is observed}}\prod_{a\in N(i)} m^{obs}_{a\to i}(v^{obs}_i)}{\sum_{i|V_i\text{is observed}}\sum_{\nu_i\in\mathcal{V}_i}\prod_{a\in N(i)} m^{all}_{a\to i}(\nu_i)}\right) \\
&= -\log\left(\sum_{i|V_i\text{is observed}}\prod_{a\in N(i)} m^{obs}_{a\to i}(v^{obs}_i)\right) + \log\left(\sum_{i|V_i\text{is observed}}\sum_{\nu_i\in\mathcal{V}_i}\prod_{a\in N(i)} m^{all}_{a\to i}(\nu_i)\right) \\
&= E_{wake}(\vec{v}^{obs}) + E_{sleep}(\vec{v}^{obs})\,,
\end{aligned}$$

where

$$\begin{aligned}
E_{wake}(\vec{v}^{obs}) &= -\log\left(\sum_{i|V_i\text{is observed}}\prod_{a\in N(i)} m^{obs}_{a\to i}(v^{obs}_i)\right) \text{ and} \\
E_{sleep}(\vec{v}^{obs}) &= \log\left(\sum_{i|V_i\text{is observed}}\sum_{\nu_i\in\mathcal{V}_i}\prod_{a\in N(i)} m^{all}_{a\to i}(\nu_i)\right). \qquad \text{(A.45)}
\end{aligned}$$

Since we have restricted our attention to acyclic factor graphs in which the visible variables always correspond to leaf nodes, the products in equation A.45 always consist of a single term. We will show in section A.8.4 that belief propagation on an acyclic factor graph constitutes an intrinsic gradient network for these two error functions separately, if the messages from the observed variables are chosen in accordance with the definitions of $n^{obs}_{i\to a}$ and $n^{all}_{i\to a}$ in section A.8.2. Specifically, we will show that belief propagation satisfies the restriction

$$\vec{T}(\vec{F}(\vec{x}^*)) = \nabla E(\vec{x})\big|_{x^*} + \left(\nabla\vec{F}^{\top}(\vec{x})\Big|_{x^*}\right)\cdot\vec{T}(\vec{x}^*)\,, \qquad \text{(A.46)}$$

which is equivalent to equation 2.8 with the definition $\vec{F}(\vec{x}^*) = \vec{x}^*$ applied to the left-hand side. The intrinsic gradient equation (2.9) can be recovered from equation 2.8 by extending it to all $\vec{x}$ and defining

$$\vec{S}(\vec{x},\vec{F}(\vec{x})) = \vec{T}(\vec{x}) - \nabla E(\vec{x})\big|_{x} + \left(\nabla\vec{F}^{\top}(\vec{x})\Big|_{x}\right)\cdot\vec{T}(\vec{x})\,.$$

An intrinsic gradient network for the composite negative log likelihood can be constructed by placing the two component networks in parallel with shared parameters, since the total gradient for such a composite network is the sum of the gradients of the constituent networks.

To establish the consistency of intrinsic gradient networks with belief propagation on acyclic factor graphs, consider an intrinsic gradient network in which the units $\vec{x}$ correspond to the belief

propagation messages $n_{i\to a}(v_i)$ and $m_{a\to i}(v_i)$ for all $v_i \in \mathcal{V}_i$, and the output functions $\vec{F}(\vec{x})$ are given by the belief propagation dynamics (equation A.40). The resulting intrinsic gradient network has dynamics equivalent to belief propagation if we use the dynamics $\vec{x}(t+1) = \vec{F}(\vec{x}(t))$ of equation 2.1. The vectors $\vec{m}$ and $\vec{n}$ collect all messages of the form $m_{a\to i}(v_i)$ and $n_{i\to a}(v_i)$, respectively, so $\vec{x} = (\vec{m}, \vec{n})$. For the sake of clarity, we continue to use belief propagation notation to refer to the elements of $\vec{x}$. That is, instead of x_j, we write $m_{a\to i}(v_i)$ or $n_{i\to a}(v_i)$. We refer to the elements of $\vec{x} = (\vec{m}, \vec{n})$ interchangeably as units and messages, depending upon which context is more appropriate.

We say that message $n_{i\to a}(v_i)$ is *complementary* to message $m_{a\to i}(v_i)$ and vice versa. As can be seen from figure A.1, complementary messages pass in opposite directions between a pair of nodes. Moreover, we say an observed variable is *behind* a message in an acyclic factor graph if there exists a path through the factor graph from the observed variable to the source of the message that does not include the destination of the message; that is, if the observed variable is behind the directed edge corresponding to the message in the factor graph. In figure A.1, the observed variables with a dashed double circle are behind message $n_{k\to c}(v_k)$; the observed variables with a solid double circle are behind message $m_{c\to k}(v_k)$.

Our construction of intrinsic gradient networks for E_{wake} and E_{sleep} will require a nonlinear training function $\vec{T}$. Rather than directly tackle this complicated problem, we first construct intrinsic gradient networks for a related pair of error functions for which a linear $\vec{T}$ suffices. Specifically, we will construct an intrinsic gradient network of the form of equation 2.18. Such a network satisfies the intrinsic gradient equation (2.9) with the slack function $\vec{S}(\vec{a}, \vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$ by design. Based upon these simpler intrinsic gradient networks, we can then construct (in section A.8.4) the requisite nonlinear $\vec{T}$ for E_{wake} and E_{sleep}, such that equation A.46 is still satisfied.

If the training function $\vec{T}(\vec{x})$ is a simple pairwise permutation matrix, then $\mathbf{D} = \left(\mathbf{T}^{-1}\right)^{\top} \cdot \mathbf{T} = \mathbf{I}$, and equation 2.17 (and thus equation 2.18) is only consistent with output functions for which the sum of the degree of all variables (the total degree, as defined in section A.3.1) in each summand is equal to one. In contrast, as can be seen from equation A.40, the sum of the degree of all variables in each summand of a belief propagation message is one less than the degree of the associated node (the number of factor nodes a variable node is connected to, and vice versa), and generally much larger than one. To reconcile this property with equation 2.18, consider the case where the training function $T_i(\vec{x})$ corresponding to each message x_i is equal to the complementary message multiplied by the number of observed variables behind the complementary message. That is, consider a $\vec{T}(\vec{m}, \vec{n})$ such that

$$\begin{aligned} T_{n_{i\to a}(v_i)}(\vec{m}, \vec{n}) &= B_{a\to i} \cdot m_{a\to i}(v_i) \\ T_{m_{a\to i}(v_i)}(\vec{m}, \vec{n}) &= B_{i\to a} \cdot n_{i\to a}(v_i)\,, \end{aligned} \tag{A.47}$$

where $B_{a\to i}$ is the number of observed variables behind the message from factor node a to variable node i, and $B_{i\to a}$ is the number of observed variables behind the message from variable node i to factor node a. Such a $\vec{T}(\vec{x})$ is clearly linear, and corresponds to a scaled pairwise permutation matrix. Given this definition, it follows that $D_{n_{i\to a}(v_i)} = \frac{B_{i\to a}}{B_{a\to i}}$ and $D_{m_{a\to i}(v_i)} = \frac{B_{a\to i}}{B_{i\to a}}$. The definitions of $B_{a\to i}$ and $B_{i\to a}$ imply that

$$B_{a\to i} = \sum_{j\in N(a)\setminus i} B_{j\to a}\,, \tag{A.48}$$

so

$$D_{n_{i\to a}(v_i)} = \frac{B_{i\to a}}{\sum_{j\in N(a)\setminus i} B_{j\to a}} \text{ and} \tag{A.49}$$

$$\frac{D_{n_{i\to a}(v_i)}+1}{D_{n_{i\to a}(v_i)}} = \frac{\sum_{j\in N(a)} B_{j\to a}}{B_{i\to a}}\,. \tag{A.50}$$

Similar relations hold for $D_{m_{a\to i}(v_i)}$.

Given this choice of $\vec{T}(\vec{x})$, we can choose values of $\psi(k)$ and $h_j^k(x)$ in equation 2.18 so that the resulting $\vec{F}$, coupled with the dynamics of equation 2.1, is equivalent to belief propagation. Specifically, we construct a single summand of $g(\vec{x})$ in equations 2.16 and 2.18 for each value of each hidden variable V_i, and for each configuration $\vec{v}_a$ of the variables in the neighborhood of each factor f_a. For the summand corresponding to value $v_i \in \mathcal{V}_i$, the component units are $m_{a\to i}(v_i)$ for all $a \in N(i)$; for the summand corresponding to configuration $\vec{v}_a \in \vec{\mathcal{V}}_a$, the component units are $n_{i\to a}(v_i)$ for all $i \in N(a)$. For each such summand k of $g(\vec{x})$, we choose $\psi(k)$ to correspond to one of the component units. We set $h_j^k(x) = x^{\frac{D_j}{D_j+1}}$ for every other component unit $j \neq \psi(k)$ of summand k, and set $h_j^k(x) = 1$ for all non-components j. As a result, the exponent of a unit in summand k is zero for non-components and one for components other than $\psi(k)$. Moreover, using equations 2.16 and A.50, the exponent of component unit $\psi(k)$ is

$$\frac{D_{\psi(k)}+1}{D_{\psi(k)}} - \sum_{m\in M\setminus\psi(k)} \frac{D_{\psi(k)}+1}{D_{\psi(k)}}\cdot\frac{D_m}{D_m+1} = \frac{\sum_{m\in M} B_m}{B_{\psi(k)}} - \sum_{m\in M\setminus\psi(k)} \frac{\sum_{n\in M} B_n}{B_{\psi(k)}}\cdot\frac{B_m}{\sum_{n\in M} B_n} = 1\,,$$

where M is the set of messages with the same destination as message ψ_k, and B_m is the number of observed variables behind message m. As can be seen from equation A.40, the belief propagation message out of a variable node or factor node is always the product of all inputs except that complementary to the output message. The choices of $\psi(k)$ and $h_j^k(x)$ detailed above ensure that the gradient in equation 2.18 yields terms of the same form.

Moreover, for $m \in M$, where M is a set of messages with the same destination, equation A.49

implies that $(1 + D_m)^{-1} = \frac{\sum_{n \in M \setminus m} B_n}{\sum_{n \in M} B_n}$, and equations A.47 and A.48 imply that

$$\begin{aligned} T^{-1}_{m_{a \to i}(v_i)}(\vec{m}, \vec{n}) &= \frac{1}{B_{a \to i}} \cdot n_{i \to a}(v_i) \\ &= \frac{1}{\sum_{j \in N(a) \setminus i} B_{j \to a}} \cdot n_{i \to a}(v_i)\,, \end{aligned}$$

so equation 2.18 yields

$$\begin{aligned} F_{m_{a \to i}(v_i)} = \frac{1}{B_{a \to i}} &\cdot (1 + D_{n_{i \to a}(v_i)})^{-1} \cdot \\ &\cdot \frac{\partial}{\partial n_{i \to a}(v_i)} \left(\sum_i \sum_{v_i} w_{v_i} \cdot \prod_{a \in N(i)} m_{a \to i}(v_i) + \sum_a \sum_{\vec{v}_a} w_{\vec{v}_a} \cdot \prod_{j \in N(a)} n_{j \to a}(v_j) \right) \\ = \frac{1}{\sum_{j \in N(a) \setminus i} B_{j \to a}} &\cdot \frac{\sum_{j \in N(a) \setminus i} B_{j \to a}}{\sum_{j \in N(a)} B_{j \to a}} \cdot \frac{\partial}{\partial n_{i \to a}(v_i)} \left(\sum_{\vec{v}_a | v_i} w_{\vec{v}_a} \cdot \prod_{j \in N(a)} n_{j \to a}(v_j) \right) \\ = \frac{1}{\sum_{j \in N(a)} B_{j \to a}} &\cdot \sum_{\vec{v}_a | v_i} w_{\vec{v}_a} \cdot \prod_{j \in N(a) \setminus i} n_{j \to a}(v_j)\,. \end{aligned} \tag{A.51}$$

A complementary set of results holds for $F_{n_{i \to a}(v_i)}$. The sum $\sum_{j \in N(a)} B_{j \to a}$ is always equal to the total number of observed variables, so if $w_{\vec{v}_a} = f_a(\vec{v}_a)$ and $w_{v_i} = 1$, then equation A.51 with the dynamics of equation 2.1 is equivalent to belief propagation as characterized by equation A.40, scaled by the number of observed variables. Moreover, since the parameterization is arbitrary, we can simply scale all the parameters by the number of observed variables to obtain the exact belief propagation dynamics.

Belief propagation only yields the gradient of the negative log likelihood via equations A.43 and A.44 when the input messages from the observed variables are set appropriately. Specifically, equation A.44 requires that the messages from the observed variables be $n^{obs}_{i \to a}(u_i) = \delta_{u_i v_i^{obs}}$ in the wake stage, and $n^{all}_{i \to a}(u_i) = 1$ in the sleep stage. As described in section 2.4, input dynamics do not arise as part of equation A.51's solution to the homogeneous part of equation 2.13; they come from the solution to the full inhomogeneous equation. However, the gradient of E_{wake} from equation A.45 yields terms of the form

$$\begin{aligned} \frac{\partial E_{wake}}{\partial m^{obs}_{a \to i}(u_i)} &= -\frac{\delta_{u_i v_i^{obs}} \cdot \prod_{b \in N(i) \setminus a} m^{obs}_{b \to i}(u_i)}{\sum_{j | V_j \text{is observed}} \prod_{a \in N(j)} m^{obs}_{a \to j}(v_j^{obs})} \\ &= -\frac{\delta_{u_i v_i^{obs}}}{\sum_{j | V_j \text{is observed}} \prod_{a \in N(j)} m^{obs}_{a \to j}(v_j^{obs})}\,, \end{aligned} \tag{A.52}$$

where V_i is an observed variable with value v_i^{obs} in the wake stage, since we restrict our attention to the case where the observed variables correspond to leaf nodes in an acyclic factor graph. When plugged into equation 2.10, which generalizes equation 2.13, this error function is not consistent

with the expected input dynamics of $n^{obs}_{i\to a}$. The error function E_{sleep} yields a similar gradient, which is correspondingly incompatible with the expected input dynamics of $n^{all}_{i\to a}$. The desired input dynamics only arise if we use an error function proportional to

$$E_{wake}(\vec{v}^{obs}) = -\sum_{i|V_i \text{is observed}} \prod_{a\in N(i)} m^{obs}_{a\to i}(v^{obs}_i) \text{ and}$$
$$E_{sleep}(\vec{v}^{obs}) = \sum_{i|V_i \text{is observed}} \sum_{\nu_i\in\mathcal{V}_i} \prod_{a\in N(i)} m^{all}_{a\to i}(\nu_i)\,. \tag{A.53}$$

It is thus clear that our initial choice of the training function $\vec{T}$ was not correct.

A.8.4 An intrinsic gradient network with a nonlinear training function which is identical to belief propagation

The desired input components of $\vec{F}(\vec{x})$ are constant, so when restricted to the inputs, $\nabla\vec{F}^{\top}(\vec{x}) = 0$, and the inhomogeneous part of equation 2.10 becomes $\vec{T}(\vec{F}(\vec{x})) = \nabla E$. As we have seen in equation A.52, if $\vec{T}(\vec{x})$ is linear, $\vec{T}(\vec{F}(\vec{x}))$ lacks the desired scaling by

$$\mathcal{N} = \left(\sum_{i|V_i \text{is observed}} \prod_{a\in N(i)} m^{obs}_{a\to i}(v^{obs}_i)\right)^{-1}.$$

However, if we modify $\vec{T}(\vec{x})$ by scaling the previously linear function by $\mathcal{N}$, it can be seen that the desired inputs are indeed a solution to the inhomogeneous part of equation 2.10. Moreover, since every element of $\vec{T}(\vec{x})$ is scaled by the same factor, the set of dynamics $\vec{F}(\vec{x})$ which satisfy the homogeneous part of the solution is unchanged at the fixed point $\vec{x}^* = \vec{F}(\vec{x}^*)$, as can be seen from equation 2.10. The resulting nonlinear training function is thus

$$T_{n_{i\to a}(v_i)}(\vec{m},\vec{n}) = \frac{B_{a\to i}}{\sum_{i|V_i \text{is observed}} \prod_{a\in N(i)} m^{obs}_{a\to i}(v^{obs}_i)} \cdot m_{a\to i}(v_i)$$
$$T_{m_{a\to i}(v_i)}(\vec{m},\vec{n}) = \frac{B_{i\to a}}{\sum_{i|V_i \text{is observed}} \prod_{a\in N(i)} m^{obs}_{a\to i}(v^{obs}_i)} \cdot n_{i\to a}(v_i)\,.$$

A complementary approach yields the training function for E_{sleep}.

Since $m^{obs}_{a\to i}(v^{obs}_i)$ generally changes with repeated applications of the belief propagation $\vec{F}(\vec{x})$ if $\vec{x}$ is not a fixed point, the scaling factor $\mathcal{N}$ need only be the same on the two sides of equation 2.10 at the fixed point. Thus, our chosen $\vec{F}$, E, $\vec{T}$ satisfy equation 2.8, but not equation 2.10, and the associated slack function is not the conventional $\vec{S}(\vec{a},\vec{b}) = \vec{T}(\vec{a}) - \vec{T}(\vec{b})$ of section 2.2. The correct slack function can be derived directly from the intrinsic gradient equation (2.9).

This construction provides an alternative interpretation for belief propagation on acyclic factor graphs. Rather than maximizing the likelihood in a probabilistic model, from which only the

marginal probabilities can ever be extracted, gradient descent on this intrinsic gradient network directly minimizes an explicit error function (equation A.45) defined on the outputs of the network. The messages in this network are deterministic, and need not be interpreted as representing probabilities. Such networks can be generalized by considering related error functions, training functions, and slack functions, which together yield alternative dynamics which still calculate the gradient of a chosen error function.

In contrast, belief propagation is traditionally extended to recurrent networks by holding both the local dynamics and the error function constant, resulting in loopy belief propagation. Whereas belief propagation on acyclic factor graphs exactly calculates the marginal probabilities, and thus the gradient of the log likelihood, this extension to loopy graphs is approximate and heuristic in nature (Murphy et al., 1999). As a result, loopy belief propagation does not exactly calculate either marginal probabilities or the gradient of the negative log likelihood. When used for training, such approximate inference techniques can yield very poor results (Kulesza & Pereira, 2008).

The compatibility of the belief propagation dynamics with equation 2.10 depends upon the ability to specify the number of observed variables behind a message, such that equation A.48 is satisfied. This is not possible if there are loops in the graph. Indeed, there does not appear to exist a training function $\vec{T}(\vec{x})$ that satisfies equation 2.8 with output functions $\vec{F}(\vec{x})$ defined by belief propagation on a loopy graph. It is thus sensible to consider generalizations of the belief propagation dynamics for which equation 2.8 can be satisfied on a wider range of topologies with simple choices of $\vec{T}(\vec{x})$, such as that given in equation 2.17. As an example, the internal messages of loopy belief propagation are consistent with equation A.24 if $\vec{T}(\vec{x})$ is a pairwise permutation matrix and all factor and variable nodes have degree $2 \cdot \tau$, but the requisite input messages satisfying the full inhomogeneous equation A.22 are very different than the constant input messages used in loopy belief propagation. In section 3.2, we construct a modular intrinsic gradient network with simple input dynamics and internal dynamics qualitatively similar to those of loopy belief propagation.

A.9 Implementation details

We draw the initial parameters for the factor nodes of the intrinsic gradient network depicted in figure 3.6, with thirty-six units in each direction of each hidden layer and visible layers of sizes determined by the data set, from normalized uniform distributions with offsets and ranges listed in table A.1. These distributions are normalized with respect to the network size by scaling both the offsets and ranges by the inverse of the size[4] of the largest layer to which the associated factor node is connected. This normalization ensures that the average value of the inputs to each variable node remains consistent despite variations in the layer sizes, given fixed inputs to the preceding factor

[4]By the size of a layer, we mean the number of units in each direction of the layer, or equivalently the number of atomic variable nodes composing the associated composite variable node.

	bottom-up (B, C)	top-down (N, O)	recurrent (F, L)	hierarchical (H, I)
polynomial offset	0.5	0.5	0.5	0.5
polynomial range	5	5	0	0.1
sigmoid offset	0.2	7	16	12
sigmoid range	16	64	3	16

Table A.1: Parameters governing the initial uniform distribution of the simulation parameters. Given offset o and range r, the probability mass function for parameter x is $\rho(x) = \frac{1}{r}$ if $0 \leq \frac{x-o}{r} \leq 1$ and 0 otherwise. The letters in parentheses correspond to those in figure 3.6.

node.

These offset and range hyperparameters are chosen with some care to ensure that both the wake and sleep networks initially converge to stable fixed points with non-zero unit activities, with the average activity of the units in the sleep network less than that in the wake network. The motivation for this initial balance between wake and sleep will be explained below. With some combinations of $h(x)$, error functions, and dynamics, we have had difficulty finding hyperparameters that induce the dynamics to converge to non-zero unit activities. This is consistent with the observed properties of the brain, which can enter epileptic oscillations or an unresponsive coma when the balance of modulatory neurotransmitters is disrupted (McCormick, 1992). As in our intrinsic gradient networks, stable neural dynamics thus appear to be dependent upon appropriate hyperparameters, rather than being an intrinsic and inevitable property of the brain's organization. We view the selection of effective hyperparameters as an engineering challenge, likely solved by evolutionary adaptation in the brain.

Unfortunately, as training progresses, the sleep network input $\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot \vec{G}(\vec{x})$, where $G_i(\vec{x}) = -x_i \cdot \log(|x_i|)$, tends to induce oscillations using dynamics like those of equation 2.1. Instead of the sleep network input directly induced by E_{sleep} of equation 3.23, we thus use inputs that induce a more stable approximation to the gradient of the sleep error function. In section 2.3.5, we observed that $\vec{F}(\vec{x}) = \mathbf{T} \cdot \vec{x}$ constitutes a solution to the homogeneous part of the intrinsic gradient equation (2.9) given the assumptions of section 2.3 and a pairwise permutation matrix $\mathbf{T}$. If we restrict the set of k for which $h_j^k(x) = 1$ to k such that $x_{\psi(k)}$ is an output unit, and choose $h_j^k(x) = 0$ otherwise, we similarly find that adding a scaled copy of the outputs to the complementary inputs (as defined by pairwise permutation matrix $\mathbf{T}$) of an intrinsic gradient network yields another intrinsic gradient network with an identical error function.

As a variation on the method discussed in section 2.4, this linear function of $\vec{x}$ in the input can also be understood as a constant input that is adjusted until the system reaches a fixed point. Before a fixed point is reached, we can imagine ourselves continuously altering the constant input to be $\vec{F}(\vec{x}) = \mathbf{T} \cdot \vec{\alpha}$, where $\alpha_i = x_i$ for output units x_i and $\alpha_i = 0$ for non-output units. Once we reach a fixed point, we know what the real constant input should be, and could re-run the network with the final value of $\vec{\alpha}$ held constant and find the same fixed point. At this fixed point, the carefully

selected constant input $\mathbf{T} \cdot \vec{\alpha}$, where $\alpha_i = x_i$ for output units x_i and $\alpha_i = 0$ otherwise, exactly balances out the sleep error function $E_{sleep}(x) = \frac{1}{2} \cdot \sum_i x_i^2$ in the intrinsic gradient equation (2.9). Thus, we can consider the error function for which we have calculated the gradient using this input to be the sleep stage error, so long as $\vec{\alpha}$ is held constant at its final value. Of course, any training step that alters the internal parameters will generally change the constant input required to balance $E_{sleep}(x) = \frac{1}{2} \cdot \sum_i x_i^2$, so the network for which we are minimizing the sleep error function changes as we train the network. However, this perturbation of the "constant" inputs will grow small as we approach a local maximum of the negative sum of squares error and the parameters converge. In the simulations of the sleep network in section 3.4, we thus use the input $\vec{F}(\vec{x}) = \mathbf{T} \cdot \vec{x}^{output}$, where $\vec{x}_i^{output} = x_i$ for output units x_i and $x_i^{output} = 0$ otherwise, in place of $\vec{F}(\vec{x}) = \mathbf{T}^{-1} \cdot \vec{G}(\vec{x})$, where $G_i(\vec{x}) = -x_i \cdot \log(|x_i|)$.

To keep the wake and sleep network fixed points near each other in accordance with section 3.4.2, we also mix the wake and sleep error functions. Specifically, we add a constant component, proportional to the bottom-up input originally dictated by E_{wake} of equation 3.23, to the input of both the wake network and the sleep network. The difference between the wake and sleep error functions (assuming they converge to the same fixed point) is not altered by this modification. We then scale down both the wake and the sleep error functions so that the final bottom-up wake input is unchanged by the overall transformation. This scaling does not alter the location of the maxima of the difference between the wake and sleep error functions, and simply scales the gradients by a constant factor.

Because of the resulting wake error function component included in the sleep network, it seems to be helpful to choose the initial parameters so the wake network is initially more active than the sleep network, as mentioned above. Otherwise, the minimization of the sleep error function, including the component from the original wake error function, dominates the learning process, whereas the wake error function should be maximized rather than minimized. If the wake network is not initially more active than the sleep network, the networks tend to learn fixed points with the sign of the bottom-level outputs flipped relative to the desired outputs.

In the pattern classification tasks we explore in sections 3.4.6 and 3.4.8, there are many bottom-up inputs, corresponding to the pixels of an image, but very few top-down inputs, corresponding to the possible categories of the images. To balance the importance of the pixel representation of the image and its classification in the error function, we scale up the components of the error function corresponding to the classification. This effectively makes many copies of the top-level units (P_i in figure 3.6). The intrinsic gradient equation (2.9) is linear in $\vec{F}$ given the assumptions of section 2.3, so scaling the error function scales the inputs by the same factor. We choose the scaling factors so that the effective number of top-level units is comparable to the number of bottom-level units (A_i in figure 3.6). When training, however, we scale down the gradient components associated with

units N_i and O_i to the same degree that the top-level inputs are scaled up. This ensures that the associated factor node is trained like one of a collection of many factor nodes which happen to have identical parameters. These implicitly replicated factor nodes are connected to the implicitly replicated top-level variable nodes, each of which makes a contribution to the error function with the same magnitude as the bottom-level variable node. If we didn't scale down the gradient components associated with units N_i and O_i, the associated factor node would be trained like a single factor node connected to a single top-level variable node of outsized importance, breaking symmetry with the bottom of the hierarchy. The final error functions are:

$$E_{wake} = \frac{1}{1 + \max_i \left(s_i^{wake-in-sleep} \right)} \cdot \sum_i s_i^{var-duplication} \cdot (1 + s_i^{wake-in-sleep}) \cdot c_i \cdot x_i \text{ and}$$
$$E_{sleep} = \frac{1}{1 + \max_i \left(s_i^{wake-in-sleep} \right)} \cdot \sum_i s_i^{var-duplication} \cdot \left(\frac{1}{2} \cdot x_i^2 + s_i^{wake-in-sleep} \cdot c_i \cdot x_i \right), \tag{A.54}$$

where scaling factor $s_i^{wake-in-sleep} = 1$ if x_i is a bottom-level output and zero otherwise, and scaling factor $s_i^{var-duplication}$ is comparable to the ratio of the number of top-down and bottom-up inputs if x_i is a top-level output,[5] and one otherwise.

If the parameters of the factor node connecting the first and second hidden layers are relatively uniform, the component of a fixed point restricted to the bottom-up input and first hidden layer is largely independent of the component of the fixed point restricted to the top-down input and second hidden layer. If we wish to train a test network to produce a classification or regression in the top-level outputs corresponding to the bottom-up inputs, then spurious fixed points due to incorrect pairings of bottom-up inputs and top-level outputs must be suppressed as described in section 3.4.4, even as fixed points corresponding to correct pairings are adjusted to exactly produce the desired outputs. The constant bottom-up input in the sleep stage (controlled by $s_i^{wake-in-sleep}$) tends to induce the attractor found in the first hidden layer to be similar in paired wake and sleep stages. We do not make a complementary modification to the top-down inputs, since that would induce the sleep stage fixed point in the second hidden layer to always match that in the wake stage, even when stable fixed points corresponding to incorrect combinations exist in the test network, where no top-down constant input can be provided.

In our implementation of belief-propagating intrinsic gradient network variable nodes, we use $h_j^k(x) = x^{1/3}$ if all inputs to the unit are greater than zero, and $h_j^k(x) = 0$ otherwise. This corre-

[5]The exact value of $s_i^{var-duplication}$ for the top-level outputs depends upon the particular application, and is specified in sections 3.4.6 and 3.4.8.

sponds, for instance, to the output function

$$F_\alpha(\vec{x}) = \begin{cases} \frac{x_b^{2/3} \cdot x_c^{2/3}}{x_a^{2/3}} & x_a, x_b, x_c \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

for variable node unit x_α in section 3.2, and ensures that the units always take on non-negative real values, like the firing rates of neurons. We can view this implementation of belief-propagating intrinsic gradient networks as mixing two solutions to the homogeneous intrinsic gradient equation for each atomic variable node. We use the solution described in section 3.2 when all inputs are greater than zero, and the solution $\vec{F}(\vec{x}) = 0$ when any input is less than or equal to zero. Because all combinations of such solutions satisfy the intrinsic gradient equation (2.9), the gradient of the associated network is always calculated correctly at a fixed point. At the boundaries between sets of output functions, the gradient of this hybrid network is not defined. However, the output functions and thus the gradient diverge for the original set of output functions when $x_i = 0$ anyway, so difficulties of this sort seem unavoidable with such symmetric polynomial solutions to the intrinsic gradient equation. Unsurprisingly, given that the output functions are not continuous when the denominator equals zero, we observe that dynamics like those of equation 2.1 sometimes do not converge to a fixed point using this intrinsic gradient network, even though the gradient is correctly calculated when a fixed point is found (for instance, see figure 3.8). Nevertheless, training is generally effective, as we demonstrate in sections 3.4.6 and 3.4.8.

In our implementation of hierarchical sigmoidal intrinsic gradient networks, we use $h_j^k(x) = \frac{x}{1+x}$ if all inputs to the unit are greater than zero and $h_j^k(x) = 0$ otherwise, instead of $h_j^k(x) = \sigma(x^{1/2})$. If $\mathbf{T}$ exchanges the pairs of units $x_{a_i} \leftrightarrow x_{\alpha_i}$, $x_{b_i} \leftrightarrow x_{\beta_i}$, and $x_{c_i} \leftrightarrow x_{\gamma_i}$ for all i (as depicted in

figures 3.2, 3.3, and 3.4) and $\psi(k) = b_i$, then

$$F_{\alpha_i}(\vec{x}) = 2 \cdot \frac{\frac{x_{a_i}}{x_{b_i}}}{\left(1 + \left(\frac{x_{a_i}}{x_{b_i}}\right)^2\right)^2} \cdot x_{b_i} \cdot \frac{\left(\frac{x_{c_i}}{x_{b_i}}\right)^2}{1 + \left(\frac{x_{c_i}}{x_{b_i}}\right)^2}$$

$$F_{\beta_i}(\vec{x}) = 2 \cdot \frac{\left(\frac{x_{a_i}}{x_{b_i}}\right)^2}{1 + \left(\frac{x_{a_i}}{x_{b_i}}\right)^2} \cdot x_{b_i} \cdot \frac{\left(\frac{x_{c_i}}{x_{b_i}}\right)^2}{1 + \left(\frac{x_{c_i}}{x_{b_i}}\right)^2}$$
$$- 2 \cdot x_{a_i} \cdot \frac{\frac{x_{a_i}}{x_{b_i}}}{\left(1 + \left(\frac{x_{a_i}}{x_{b_i}}\right)^2\right)^2} \cdot \frac{\left(\frac{x_{c_i}}{x_{b_i}}\right)^2}{1 + \left(\frac{x_{c_i}}{x_{b_i}}\right)^2} - \frac{\left(\frac{x_{a_i}}{x_{b_i}}\right)^2}{1 + \left(\frac{x_{a_i}}{x_{b_i}}\right)^2} \cdot 2 \cdot x_{c_i} \cdot \frac{\frac{x_{c_i}}{x_{b_i}}}{\left(1 + \left(\frac{x_{c_i}}{x_{b_i}}\right)^2\right)^2}$$

$$F_{\gamma_i}(\vec{x}) = \frac{\left(\frac{x_{a_i}}{x_{b_i}}\right)^2}{1 + \left(\frac{x_{a_i}}{x_{b_i}}\right)^2} \cdot x_{b_i} \cdot 2 \cdot \frac{\frac{x_{c_i}}{x_{b_i}}}{\left(1 + \left(\frac{x_{c_i}}{x_{b_i}}\right)^2\right)^2}$$

for $i \in \{1, 2, \cdots, n\}$ if $x_{a_i}, x_{b_i}, x_{c_i} > 0$. If x_{a_i}, x_{b_i}, or x_{c_i} is less than or equal to zero, then $F_{\alpha_i}(\vec{x}) = F_{\beta_i}(\vec{x}) = F_{\gamma_i}(\vec{x}) = 0$. This function is sigmoidal like the logistic function, but is less apt to produce negative values for the output functions when $x_{\psi(k)} < 0$; it is equivalent to a one-sided softsign function (Bergstra et al., 2009). Sigmoidal functions $h_j^k(x)$ are convenient, since they induce output functions that are continuous at $x_{\psi(k)} = 0$, and thus more likely to converge to a fixed point using dynamics like those of equation 2.1, as is evident in figure 3.13.

To find approximate fixed points, we initialize the units randomly from a uniform distribution between zero and one. We then perform 500 parallel updates using the dynamics

$$x_i(t+1) = 0.9 \cdot x_i(t) + 0.1 \cdot F_i(\vec{x}(t)) \tag{A.55}$$

for the recurrent projections from the variable nodes (units E_i and K_i in figure 3.6) and

$$x_i(t+1) = 0.8 \cdot x_i(t) + 0.2 \cdot F_i(\vec{x}(t)) \tag{A.56}$$

for all other units, corresponding to equation 2.3. Although this procedure does not guarantee convergence to a fixed point, its sufficiency is empirically demonstrated in sections 3.4.6 and 3.4.8.

Bibliography

Abeles, M., Bergman, H., Gat, I., Meilijson, I., Seidemann, E., Tishby, N., & Vaadia, E. (1995). Cortical activity flips among quasi-stationary states. *Proceedings of the National Academy of Sciences of the Unites States of America*, *92*, 8616–8620.

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science*, *9*, 147–169.

Ahissar, M., & Hochstein, S. (1997). Task difficulty and the specificity of perceptual learning. *Nature*, *387*, 401–406.

Aji, S. M., & McEliece, R. J. (2000). The generalized distributive law. *IEEE Transactions on Information Theory*, *46*(2), 325–343.

Albrecht, D. G., & Hamilton, D. B. (1982). Striate cortex of monkey and cat: Contrast response function. *Journal of Neurophysiology*, *48*(1), 217–237.

Almeida, L. B. (1987). A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Proceedings of the IEEE First International Conference on Neural Networks (ICNN 1987)* (pp. 609–618).

Andersen, R.A., Essick, G.K., & Siegel, R.M. (1985). Encoding of spatial location by posterior parietal neurons. *Science*, *230*, 450–458.

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought.* Mahwah, NJ: Erlbaum.

Angelucci, A., Levitt, J. B., Walton, E. J. S., Hupe, J. M., Bullier, J., & Lund, J. S. (2002). Circuits for local and global signal integration in primary visual cortex. *Journal of Neuroscience*, *22*(19), 8633–8646.

Atiya, A. F. (1988). Learning on a general network. In D. Z. Anderson (Ed.), *Neural Information Processing Systems* (pp. 22–30). New York: American Institute of Physics.

Atiya, A. F., & Parlos, A. G. (2000). New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, *11*(3), 697–709.

Barbour, D. L., & Wang, X. (2003). Contrast tuning in auditory cortex. *Science*, *299*, 173–175.

Bartho, P., Curto, C., Luczak, A., Marguet, S. L., & Harris, K. D. (2009). Population coding of tone stimuli in auditory cortex: Dynamic rate vector analysis. *European Journal of Neuroscience*, *30*, 1767–1778.

Barto, A. G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, *4*, 229–256.

Behrens, H., Gawronska, D., Hollatz, J., & Schürmann, B. (1991). Recurrent and feedforward backpropagation for time independent pattern recognition. In *Proceedings of the 1991 International Joint Conference on Neural Networks (IJCNN '91)* (Vol. 2, pp. 591–596).

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, & T. Hoffman (Eds.) *Advances in Neural Information Processing Systems (NIPS 19)* (pp. 153–160). Cambridge, MA: MIT Press.

Bengio, Y., & LeCun, Y. (2007). Scaling learning algorithms towards AI. In L. Bottou, O. Chapelle, D. DeCoste, & J. Weston (Eds). *Large-Scale Kernel Machines* (pp. 321–360). Cambridge, MA: MIT Press.

Bergstra, J., Desjardins, G., Lamblin, P., & Bengio, Y. (2009). *Quadratic polynomials learn better image features* (Technical Report 1337). Montreal, Canada: Universite de Montreal, Departement d'Informatique et de Recherche Operationnelle.

Binzegger, T. Douglas, R. J., & Martin, K. A. C. (2004). A quantitative map of the circuit of cat primary visual cortex. *Journal of Neuroscience*, *24*(39), 8441–8453.

Binzegger, T., Douglas, R. J., & Martin, K. A. C. (2005). Axons in cat visual cortex are topologically self-similar. *Cerebral Cortex*, *15*, 152–165.

Binzegger, T., Douglas, R. J., & Martin, K. A. C. (2007). Stereotypical bouton clustering of individual neurons in cat primary visual cortex. *Journal of Neuroscience*, *27*(45), 12242–12254.

Bishop, C. M. (1995). *Neural networks for pattern recognition.* Oxford: Oxford University Press.

Bishop, C. M. (2006). *Pattern recognition and machine learning.* New York: Springer.

Bottou, L., & Bousquet, O. (2008). The tradeoffs of large scale learning. In J. C. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.), *Advances in Neural Information Processing Systems 20 (NIPS 20)* (pp. 161–168). Cambridge, MA: MIT Press.

Boussaoud, D., Jouffrais, C., Bremmer, F. (1998) Eye position effects on the neuronal activity of dorsal premotor cortex in the macaque monkey. *Journal of Neurophysiology*, *80*, 1132–1150.

Brashers-Krug, T., Shadmehr, R., & Bizzi, E. (1996). Consolidation in human motor memory. *Nature*, *382*, 252–255.

Briggs, F., & Usrey, W. M. (2009). Parallel processing in the corticogeniculate pathway of the macaque monkey. *Neuron*, *62*, 135–146.

Brotchie, P.R., Andersen, R.A., Snyder, L.H., & Goodman, S.J. (1995). Head position signals used by parietal neurons to encode locations of visual stimuli. *Nature*, *385*, 232–235.

Buneo, C. A., Jarvis, M. R., Batista, A. P., & Andersen, R. A. (2002). Direct visuomotor transformations for reaching. *Nature*, *416*, 632–636.

Carpenter, G. A., & Grossberg, S. (1987). A massively parallel architecture for a self-organizing neural pattern recognition machine *Computer Vision, Graphics, and Image Processing*, *37*, 54–115.

Carpenter, G. A., & Grossberg, S. (1988). The ART of adaptive pattern recognition by a self-organizing neural network. *Computer*, *21*(3), 77–88.

Chialvo, D. R., & Bak, P. (1999). Learning from mistakes. *Neuroscience*, *90*(4), 1137–1148.

Churchland, M. M., Yu, B. M., Ryu, S. I., Santhanam, G, & Shenoy, K. V. (2006). Neural variability in premotor cortex provides a signature of motor preparation. *Journal of Neuroscience*, *26*(14), 3697–3712.

Churchland, M. M. & Shenoy, K. V. (2007). Delay of movement caused by disruption of cortical preparatory activity. *Journal of Neurophysiology*, *97*, 348–359.

Churchland, M. M., Yu, B. M., Cunningham, J. P., Sugrue, L. P., Cohen, M. R., Corrado, G. S., Newsome, W. T., Clark, A. M., Hosseini, P., Scott, B. B., Bradley, D. C., Smith, M. A., Kohn, A., Movshon, J. A., Armstrong, K. M., Moore, T., Chang, S. W., Snyder, L. H., Lisberger, S. G., Priebe, N. J., Finn, I. M., Ferster, D., Ryu, S. I., Santhanam, G., Sahani, M., Shenoy, K. V. (2010). Stimulus onset quenches neural variability: a widespread cortical phenomenon. *Nature Neuroscience*, *13*, 369–378.

Ciresan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2010). Deep, Big, Simple Neural Nets for Handwritten Digit Recognition. *Neural Computation*, *22*, 3207–3220.

Coates, A., Lee, H., & Ng, A. Y. (2010). An analysis of single-layer networks in unsupervised feature learning. In H. Lee, M. A. Ranzato, Y. Bengio, G. Hinton, Y. LeCun, & A. Ng (Organizers), *Deep Learning and Unsupervised Feature Learning.* Workshop conducted at the meeting of the Twenty-Fourth Annual Conference on Neural Information Processing Systems, Whistler, BC, Canada.

Cohen, M. A., & Grossberg, S. (1983). Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, *13*(5), 815–826.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, *42*, 393–405.

Coppersmith, D., & Winograd, S. (1990). Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, *9*(3), 251–280.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Leaming*, *20*, 273–297.

Crick, F. (1989). The recent excitement about neural networks. *Nature*, *337*, 129–132.

Crick, F., & Koch C. (1998). Constraints on cortical and thalamic projections: the no-strong-loops hypothesis. *Nature*, *391*, 245–250.

Day, B. L., Rothwell, J. C., Thompson, P. D., Maertens de Noordhout, A., Nakashima, K., Shannon, K., & Marsden, C. D. (1989) Delay in the execution of voluntary movement by electrical or magnetic brain stimulation in intact man: Evidence for the storage of motor programs in the brain. *Brain*, *112*, 649–663.

Deco, G., & Rolls, E. T. (2004). A neurodynamical cortical model of visual attention and invariant object recognition. *Vision Research*, *44*, 621–642.

Derrington, A. M., & Lennie, P. (1984). Spatial and temporal contrast sensitivities of neurones in lateral geniculate nucleus of macaque. *Journal of Physiology*, *357*, 219–240.

Douglas, R. J., & Martin, K. A. C. (2004). Neuronal circuits of the neocortex. *Annual Review of Neuroscience*, *27*, 419–451.

Durbin, R., & Rumelhart, D. E. (1989). Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, *1*, 133–142.

Duncan, J., Ward, R., & Shapiro, K. (1994) Direct measurement of attentional dwell time in human vision. *Nature*, *369*, 313–315.

Erhan, D., Manzagol, P-A., Bengio, Y., Bengio, S., & Vincent, P. (2009). The difficulty of training deep architectures and the effect of unsupervised pre-training. In D. van Dyk & M. Welling (Eds.) *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, (pp. 153–160). Retrieved from http://jmlr.csail.mit.edu/proceedings/papers/v5/erhan09a/erhan09a.pdf.

Erhan, D., Courville, A., Bengio, Y., & Vincent, P. (2010). Why does unsupervised pre-training help deep learning? In Y. W. Teh & M. Titterington (Eds.) *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, (pp. 201–208). Retrieved from http://jmlr.csail.mit.edu/proceedings/papers/v9/erhan10a/erhan10a.pdf.

Felleman, D. J., & Van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex*, *1*, 1–47.

Fiorani, M. F., Rosa, M. G. P., Gattass, R., & Rocha-Miranda, C. E. (1992). Dynamic surrounds of receptive fields in primate striate cortex: A physiological basis for perceptual completion? *Proceedings of the National Academy of Sciences of the Unites States of America*, *89*, 8547–8551.

Fischer, A., & Igel, C. (2010). Empirical analysis of the divergence of Gibbs sampling based learning algorithms for restricted Boltzmann machines. In K. Diamantaras, W. Duch, & L. Iliadis (Eds.) *20th International Conference on Artificial Neural Networks (ICANN 2010)* (Vol. 3, pp. 208–217). doi:10.1007/978-3-642-15825-4

Foxe, J. J., & Simpson, G. V. (2002). Flow of activation from V1 to frontal cortex in humans: A framework for defining "early" visual processing. *Experimental Brain Research*, *142*, 139–150.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*, 193–202.

Gilbert, C. D., & Sigman, M. (2007) Brain states: Top-down influences in sensory processing. *Neuron*, *54*, 677–696.

Girard, P., Hupe, M., & Bullier, J. (2001). Feedforward and feedback connections between areas V1 and V2 of the monkey have similar rapid conduction velocities. *Journal of Neurophysiology*, *85*, 1328–1331.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *6*(6), 721–741.

Ghazanfar, A. A., & Schroeder, C. E. (2006). Is neocortex essentially multisensory? *Trends in Cognitive Sciences*, *10*(6), 278–285.

Grieve, K.L., & Sillito, A.M. (1995). Differential properties of cells in the feline primary visual cortex providing the corticofugal feedback to the lateral geniculate nucleus and visual claustrum. *Journal of Neuroscience*, *15*(7), 4868–4874.

Grosof, D. H, Shapley, R. M., & Hawken, M. J. (1993). Macaque V1 neurons can signal 'illusory' contours. *Nature*, *365*, 550–552.

Grossberg, S. (1987). Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, *11*, 23–63.

Harris, K. D. (2005). Neural signatures of cell assembly organization. *Nature Reviews Neuroscience*, *6*, 399–407.

Harris, K. D. (2008). Stability of the fittest: Organizing learning through retroaxonal signals. *Trends in Neurosciences*, *31*(3), 130–136.

Hebb, D. O. (1949). *The organization of behavior.* New York: Wiley.

Hegde, J. (2008). Time course of visual perception: Coarse-to-fine processing and beyond. *Progress in Neurobiology*, *84*, 405–439.

Henderson, J.M., & Hollingworth, A. (1998). Eye movements during scene viewing: An overview. In G. Underwood (Ed.), *Eye Guidance in Reading and Scene Perception* (pp. 269–283). New York: Elsevier.

Herzog, M. H., & Fahle, M. (1997). The role of feedback in learning a vernier discrimination task. *Vision Research*, *37*(15), 2133–2141.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, *14*(8), 1711–1800.

Hinton, G. E., Dayan, P., Frey, B. J., & Neal, R. M. (1995). The "wake-sleep" algorithm for unsupervised neural networks. *Science*, *268*, 1158–1161.

Hinton G. E., & McClelland J. L. (1988). Learning representations by recirculation. In D. Z. Anderson (Ed.), *Neural Information Processing Systems* (pp. 358–366). New York: American Institute of Physics.

Hinton, G. E., Osindero, S., & Teh, Y-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–1554.

Hirsch, M. W. (1989). Convergent activation dynamics in continuous time networks. *Neural Networks*, *2*, 331-349.

Holtmaat, A., & Svoboda, K. (2009). Experience-dependent structural synaptic plasticity in the mammalian brain. *Nature Reviews Neuroscience*, *10*, 647–658.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the Unites States of America*, *79*, 2554–2558.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences of the Unites States of America*, *81*, 3088–3092.

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology*, *160*, 106–154.

Hung, C. P., Kreiman, G., Poggio, T., & DiCarlo, J. J. (2005). Fast readout of object identity from macaque inferior temporal cortex. *Science*, *310*, 863–866.

Izhikevich, E. M., & Edelman, G. M. (2008). Large-scale model of mammalian thalamocortical systems. *Proceedings of the National Academy of Sciences of the Unites States of America*, *105*(9), 3593–3598.

Jarrett, K., Kavukcuoglu, K., Ranzato, M. A., & LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Proceedings of the 12th International Conference on Computer Vision (ICCV 2009)* (pp. 2146–2153). Los Alamitos, CA: IEEE Computer Society.

Jones, M. J. (1992). *Using recurrent networks for dimensionality reduction.* (Master's thesis). Retrieved from http://hdl.handle.net/1721.1/7045.

Jones, L. M., Fontanini, A., Sadacca, B. F., Miller, P., & Katz, D. B. (2007). Natural stimuli evoke dynamic sequences of states in sensory cortical ensembles. *Proceedings of the National Academy of Sciences of the Unites States of America*, *104*, 18772–18777.

Kalisman, N., Silberberg, G., & Markram, H. (2005). The neocortical microcircuit as a *tabula rasa*. *Proceedings of the National Academy of Sciences of the Unites States of America*, *102*(3), 880–885.

Kamimura, R. (1991). Application of the recurrent neural network to the problem of language acquisition. In L. K. Barret (Ed.), *Proceedings of the Conference on Analysis of Neural Network Applications (ANNA '91)* (pp. 14–28). New York: ACM.

Kampa, B. M., Letzkus, J. J., & Stuart, G. J. (2006). Cortical feed-forward networks for binding different streams of sensory information. *Nature Neuroscience*, *9*(12), 1472–1473.

Kampa, B. M., Letzkus, J. J., & Stuart, G. J. (2007). Dendritic mechanisms controlling spike-timing-dependent synaptic plasticity. *TRENDS in Neurosciences*, *30*(9), 456–463.

Kanizsa, G. (1979). *Organization in vision: Essays on Gestalt perception.* New York: Praeger Publishers.

Karmarkar, U, R., & Dan, Y. (2006). Experience-dependent plasticity in adult visual cortex. *Neuron*, *52*, 577–585.

Karni, A., & Sagi, D. (1993). The time course of learning a visual skill. *Nature*, *365*, 250–252.

Kemere, C., Santhanam, G., Yu, B. M., Afshar, A., Ryu, S. I., Meng, T. H., & Shenoy, K. V. (2008). Detecting neural-state transitions using hidden Markov models for motor cortical prostheses. *Journal of Neurophysiology*, *100*, 2441–2452.

Koch, C. (1999). *Biophysics of computation: Information processing in single neurons.* New York: Oxford University Press.

Komatsu, H., Kinoshita, M., & Murakami, I. (2000). Neural responses in the retinotopic representation of the blind spot in the macaque V1 to stimuli for perceptual filling-in. *Journal of Neuroscience*, *20*(24), 9310–9319.

Komatsu, H. (2006). The neural mechanisms of perceptual filling-in. *Nature Reviews Neuroscience*, *7*, 220–231.

Körding, K. P., & König, P. (2001). Supervised and unsupervised learning with two sites of synaptic integration. *Journal of Computational Neuroscience*, *11*, 207–215.

Kovács, I., & Julesz, B. (1993). A closed curve is much more than an incomplete one: Effect of closure in figure-ground segmentation. *Proceedings of the National Academy of Sciences of the Unites States of America*, *90*, 7495–7497.

Kozloski, J., Hamzei-Sichani, F., & Yuste, R. (2001). Stereotyped position of local synaptic targets in neocortex. *Science*, *293*, 868–872.

Kulesza, A., & Pereira, F. (2008). Structured learning with approximate inference. In J. C. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.), *Advances in Neural Information Processing Systems 20 (NIPS 20)* (pp. 785–792). Cambridge, MA: MIT Press.

Kschischang, F. R., Frey, B. J., & Loeliger, H-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, *47*(2), 498–519.

Lamme, V. A. F., & Roelfsema, P. R. (2000). The distinct modes of vision offered by feedforward and recurrent processing. *Trends in Neurosciences*, *23*(11), 571–579.

Lansner, A. (2009). Associative memory models: From the cell-assembly theory to biophysically detailed cortex simulations. *Trends in Neurosciences*, *32*(3), 178–186.

Larochelle, H., Bengio, Y., Louradour, J., & Lamblin, P. (2009). Exploring strategies for training deep neural networks. *Journal of Machine Learning Research*, *10*, 1–40.

LeCun, Y. (1988). A theoretical framework for back-propagation. In D. Touretzky, G. Hinton, & T. Sejnowski (Eds.), *Proceedings of the 1988 Connectionist Models Summer School* (pp. 21–28). San Mateo, CA: Morgan Kaufmann.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeCun, Y., Bottou, L., Orr, G. B., & Muller, K.-R. (1998). Efficient backprop. In G. Orr, & K. Muller (Eds.), *Neural networks: Tricks of the trade*. Berlin: Springer.

Lecun, Y., & Cortes, C. (1998). *The MNIST database of handwritten digits.* Retrieved from http://yann.lecun.com/exdb/mnist/.

LeCun, Y., Huang, F. J., & Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004)* (pp. 97–104). Los Alamitos, CA: IEEE Computer Society.

Ledgeway, T., Zhan, C., Johnson, A. P., Song, Y., & Baker, C. L., Jr. (2005). The direction-selective contrast response of area 18 neurons is different for first- and second-order motion. *Visual Neuroscience*, *22*, 87–99.

Lee, T. S., & Mumford, D. (2003). Hierarchical Bayesian inference in the visual cortex. *Journal of the Optical Society of America A*, *20*(7), 1434–1448.

Lee, T. S., & Nguyen, M. (2001). Dynamics of subjective contour formation in the early visual cortex. *Proceedings of the National Academy of Sciences of the Unites States of America*, *98*(4), 1907–1911.

Long, P. M., & Servedio, R. A. (2010). Restricted Boltzmann machines are hard to approximately evaluate or simulate. In J. Fuürnkranz & T. Joachims (Eds.), *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)* (pp. 703–710). Retrieved from http://www.icml2010.org/papers/115.pdf.

Luczak, A., Bartho, P., & Harris, K. D. (2009). Spontaneous events outline the realm of possible sensory responses in neocortical populations. *Neuron*, *62*, 413–425.

Lundqvist, M., Rehn, M., Djurfeldt, M., & Lansner, A. (2006). Attractor dynamics in a modular network model of neocortex. *Network: Computation in Neural Systems*, *17*(3), 253–276.

Maass, W., Natschläger, T., & Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, *14*, 2531–2560.

Malenka, R. C., & Bear, M. F. (2004). LTP and LTD: An embarrassment of riches. *Neuron*, *44*, 5–21.

Markram, H. (2006). The blue brain project. *Nature Reviews Neuroscience*, *7*, 153–160.

Martinez, L. M., Wang, Q., Reid, R. C., Pillai, C., Alonso, J.-M., Sommer, F. T., & Hirsch, J. A. (2005). Receptive field structure varies with layer in the primary visual cortex. *Nature Neuroscience*, *8*(3), 372–379.

Martinez-Trujillo, J. C & Treue, S. (2004). Feature-based attention increases the selectivity of population responses in primate visual cortex. *Current Biology*, *14*, 744–751.

Marr, D. (1982). *Vision: A computational investigation into the human representation and processing of visual information.* San Francisco: W. H. Freeman.

Matsumoto, M., & Komatsu, H. (2005). Neural responses in the macaque V1 to bar stimuli with various lengths presented on the blind spot. *Journal of Neurophysiology 93*, 2374–2387.

Matsuoka, K. (1992). Stability conditions for nonlinear continuous neural networks with asymmetric connection weights. *Neural Networks*, *5*, 495–500.

Maunsell, J. H. R., & Newsome, W. T. (1987). Visual processing in monkey extrastriate cortex. *Annual Review of Neuroscience*, *10*, 363–401.

Mazzoni, P., Andersen, R. A., & Jordan, M. I. (1991). A more biologically plausible learning rule for neural networks. *Proceedings of the National Academy of Sciences of the Unites States of America*, *88*, 4433–4437.

McAdams, C. J., and Maunsell, J. H. R. (1999). Effects of attention on orientation-tuning functions of single neurons in macaque cortical area V4. *Journal of Neuroscience*, *19*(1), 431–441.

McCormick, D. A. (1992). Neurotransmitter actions in the thalamus and cerebral cortex. *Journal of Clinical Neurophysiology*, *9*(12), 212–223.

Minsky, M. L., & Papert, S. A. (1969). *Perceptrons.* Cambridge, MA: MIT Press.

Mountcastle, V. B. (1997). The columnar organization of the neocortex. *Brain*, *120*, 701–722.

Murphy, K. P., Weiss, Y., & Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In K. B. Laskey & H. Prade (Eds.) *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI '99)* (pp. 467–475). San Francisco, CA: Morgan Kaufmann.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Olshausen, B. A., & Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, *381*, 607–609.

O'Reilly, R. C. (1996). Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural Computation*, *8*, 895–938.

O'Reilly, R. C. (2001). Generalization in Interactive Networks: The benefits of inhibitory competition and Hebbian learning. *Neural Computation*, *13*, 1199–1241.

Ottaway, M. B., Simard, P. Y., & Ballard, D. H. (1988). Fixed point analysis for recurrent networks. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 1 (NIPS 1)* (pp. 149–159). San Mateo, CA: Morgan Kaufmann.

Parker, D. B. (1985). *Learning logic* (Technical report 47). Cambridge, MA: Massachusetts Institute of Technology, Center for Computational Research in Economics and Management Science.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference.* San Francisco, CA: Morgan Kaufmann.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation*, *1*, 263–269.

Pearlmutter, B. A. (1995). Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, *6*(5), 1212–1228.

Peirce, J. W. (2007). The potential importance of saturating and supersaturating contrast response functions in visual cortex. *Journal of Vision*, *7*(6), 1–10.

Peng, X., & Van Essen, D. C. (2005). Peaked encoding of relative luminance in macaque areas V1 and V2. *Journal of Neurophysiology*, *93*, 1620–1632.

Perin, R., Berger, T. K., & Markram, H. (2011). A synaptic organizing principle for cortical neuronal groups. *Proceedings of the National Academy of Sciences of the Unites States of America*, *108*(13), 5419–5424.

Petkov, C. I., O'Connor, K. N., & Sutter, M. L. (2007). Encoding of illusory continuity in primary auditory cortex. *Neuron*, *54*, 153–165.

Pettet, M. W., McKee, S. P., & Grzywacz, N. M. (1998). Constraints on long range interactions mediating contour detection. *Vision Research*, *38*(6), 865–879.

Pineda, F. J. (1987). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, *59*(19), 2229–2232.

Pineda, F. J. (1995). Recurrent backpropagation networks. In Y. Chauvin & D. E. Rumelhart (Eds.), *Backpropagation: Theory, Architectures, and Applications* (pp. 99–135). Hillsdale, NJ: Erlbaum.

Poirazi, P., Brannon, T., & Mel, B. W. (2003). Pyramidal neuron as two-layer neural network. *Neuron*, *37*, 989–999.

Qian, N., & Sejnowski, T. J. (1989). Learning to solve random-dot stereograms of dense and transparent surfaces with recurrent backpropagation. In D. Touretzky, G. Hinton, & T. Sejnowski (Eds.), *Proceedings of the 1988 Connectionist Models Summer School* (pp. 435–443). San Mateo, CA: Morgan Kaufmann.

Rabinovich, M., Huerta, R., & Laurent, G. (2008). Transient dynamics for neural processing. *Science*, *321*, 48–50.

Rabinovich, M. I., Huerta, R., Varona, P., & Afraimovich, V. S. (2008). Transient cognitive dynamics, metastability, and decision making. *PLoS Computational Biology*, *4*(5), e1000072.

Radons, G., Becker, J. D., Dülfer, B., Krüger, J. (1994). Analysis, classification, and coding of multielectrode spike trains with hidden Markov models. *Biological Cybernetics*, *71*, 359–373.

Raymond, J. E., Shapiro, K. L., & Arnell, K. M. (1992). Temporary suppression of visual processing in an RSVP task: An attentional blink? *Journal of Experimental Psychology*, *18*(3), 849–860.

Riesenhuber, M., & Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature Neuroscience*, *2*, 1019–1025.

Ringach, D. L., Hawken, M. J, & Shapley, R. (1997). Dynamics of orientation tuning in macaque primary visual cortex. *Nature*, *387*, 281–284.

Roelfsema, P. R., & van Ooyen, A. (2005). Attention-gated reinforcement learning of internal representations for classification. *Neural Computation*, *17*, 2176–2214.

Rolls, E. T., & Tovee, M. J. (1994). Processing speed in the cerebral cortex and the neurophysiology of visual masking. *Proceedings of the Royal Society of London B*, *257*, 9–15.

Rubin, N., Nakayama, K., & Shapley, R. (1997). Abrupt learning and retinal size specificity in illusory-contour perception. *Current Biology*, *7*(7), 461–467.

Rumelhart, D. E., McClelland, J. L., & the PDP Research Group (1986). *Parallel distributed processing: Explorations in the microstructure of cognition* (Volume 1: Foundations). Cambridge, MA: MIT Press.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Vol. 1. Foundations* (pp. 318–362). Cambridge, MA: MIT Press.

Salinas, E., & Thier, P. (2000). Gain modulation: A major computational principle of the central nervous system. *Neuron*, *27*, 15–21.

Salinas, E., & Abbott, L. F. (2001). Coordinate transformations in the visual system: How to generate gain fields and what to compute with them. In M. A. L. Nicolelis (Ed.), *Progress in Brain Research: Vol. 130. Advances in Neural Population Coding* (pp. 175–190). Amsterdam: Elsevier.

Schmolesky, M. T., Wang, Y., Hanes, D. P., Thompson, K. G., Leutgeb, S., Schall, J. D., & Leventhal, A. G. (1998). Signal timing across the macaque visual system. *Journal of Neurophysiology*, *79*, 3272–3278.

Schroeder, C. E., Mehta, A. D., & Givre, S. J. (1998). A spatiotemporal profile of visual system activation revealed by current source density analysis in the awake macaque. *Cerebral Cortex*, *8*, 575–592.

Seidemann, E., Meilijson, I., Abeles, M., Bergman, H., Vaadia, E. (1996). Simultaneously recorded single units in the frontal cortex go through sequences of discrete and stable states in monkeys performing a delayed localization task. *Journal of Neuroscience*, *16*(2), 752–768.

Seita, A. R., & Watanabe, T. (2003). Psychophysics: Is subliminal learning really passive? *Nature*, *422*, 36.

Seitz, A., Kim, D., & Watanabe, T. (2009). Rewards evoke learning of unconsciously processed visual stimuli in adult humans. *Neuron*, *61*, 700–707.

Seitz, A., Lefebvre, C., Watanabe, T., & Jolicoeur, P. (2005). Requirement for high-level processing in subliminal learning. *Current Biology*, *15*(18), R753–R755.

Serre, T., Kouh, M., Cadieu, C., Knoblich, U., Kreiman, G., & Poggio, T. (2005). *A theory of object recognition: Computations and circuits in the feedforward path of the ventral stream in primate visual cortex* (MIT-CSAIL-TR-2005-082). Cambridge, MA: Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory.

Seung, H. S. (2003). Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, *40*, 1063–1073.

Shadmehr, R., & Brashers-Krug, T. (1997). Functional stages in the formation of human long-term motor memory. *Journal of Neuroscience*, *17*(1), 409–419.

Sheth, B. R., Sharma, J., Rao, S. C., & Sur, M. (1996). Orientation maps of subjective contours in visual cortex. *Science*, *274*, 2110–2115.

Simard, P.Y., Steinkraus, D., & Platt, J. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR 2003)* (pp. 958–963). Los Alamitos, CA: IEEE Computer Society.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Vol. 1. Foundations* (pp. 194–281). Cambridge, MA: MIT Press.

Song, S., Sjöström, P. J., Reigl, M., Nelson, S., & Chklovskii, D. B. (2005). Highly nonrandom features of synaptic connectivity in local cortical circuits. *PLoS Biology*, *3*(3), 507–519.

Stepanyants, A., Hof, P. R., & Chklovskii, D. B. (2002). Geometry and structural plasticity of synaptic connectivity. *Neuron*, *34*, 275–288.

Stepanyants, A., Hirsch, J. A., Martinez, L. M., Kisvárdy, Z. F., Ferecsko, A. S., & Chklovskii, D. B. (2008). Local potential connectivity in cat primary visual cortex. *Cerebral Cortex*, *18*, 13–28.

Stickgold, R. (2005). Sleep-dependent memory consolidation. *Nature*, *437*, 1272–1278.

Stork, D. G. (1989). Is backpropagation biologically plausible? In *International Joint Conference on Neural Networks (IJCNN)* (Vol. 2, pp. 241–246). doi:10.1109/IJCNN.1989.118705

Sugase, Y., Yamane, S., Ueno, S., & Kawano, K. (1999). Global and fine information coded by single neurons in the temporal visual cortex. *Nature*, *400*, 869–873.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Swadlow, H. A., & Weyand, T. G. (1987). Corticogeniculate neurons, corticotectal neurons, and suspected interneurons in visual cortex of awake rabbits: Receptive-field properties, axonal properties, and effects of EEG arousal. *Journal of Neurophysiology*, *57*(4), 977–1001.

Thorpe, S. J., & Imbert, M. (1989). Biological constraints on connectionist modelling. In R. Pfeifer, F. Fogelman-Soulie, L. Steels, & Z. Schreter (Eds.), *Connectionism in Perspective* (pp. 63–92). New York: Elsevier.

Thorpe, S., Fize, D., & Marlot, C. (1996). Speed of processing in the human visual system. *Nature*, *381*, 520–522.

Treue, S., & Martinez-Trujillo, J. C. (1999). Feature-based attention influences motion processing gain in macaque visual cortex. *Nature*, *399*, 575–579.

Treue, S. (2001). Neural correlates of attention in primate visual cortex. *Trends in Neurosciences*, *24*(5), 295–300.

Trotter, Y., & Celebrini, S. (1999). Gaze direction controls response gain in primary visual cortex neurons. *Nature*, *398*, 239–242.

Tsodyks, M., & Gilbert, C. (2004). Neural networks and perceptual learning. *Nature*, *431*, 775–781.

VanRullen, R., & Koch, C. (2002). Visual selective behavior can be triggered by a feed-forward process. *Journal of Cognitive Neuroscience*, *15*(2), 209–217.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P-A. (2008). Extracting and composing robust features with denoising autoencoders. In A. McCallum and S. Roweis (Eds.), *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)* (pp. 1096–1103). Retrieved from http://icml2008.cs.helsinki.fi/papers/592.pdf

Vyazovskiy, V. V., Cirelli, C., Pfister-Genskow, M., Faraguna, U., & Tononi, G. (2008). Molecular and electrophysiological evidence for net synaptic potentiation in wake and depression in sleep. *Nature Neuroscience*, *11*(2), 200–208.

Watanabe, T., Nanez, J. E., & Sasaki, Y. (2001). Perceptual learning without perception. *Nature*, *413*, 844–848.

Waters, J., Schaefer, A., & Sakmann, B. (2005). Backpropagating action potentials in neurones: Measurement, mechanisms and potential functions. *Progress in Biophysics and Molecular Biology*, *87*, 145–170.

Werbos, P. J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* Ph.D. dissertation, Harvard University, Cambridge, MA.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*, 270–280.

Williford, T., & Maunsell, J. H. R. (2006). Effects of spatial attention on contrast response functions in macaque area V4. *Journal of Neurophysiology*, *96*, 40–54.

Xie, X., & Seung, H. S. (2003). Equivalence of backpropagation and contrastive Hebbian learning in a layered network. *Neural Computation*, *15*, 441–454.

Yedidia, J. S., Freeman, W. T., & Weiss, Y. (2005). Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, *51*(7), 2282–2312.

Yoshimura, Y., & Callaway, E. M. (2005). Fine-scale specificity of cortical networks depends upon inhibitory cell type and connectivity. *Nature Neuroscience*, *8*(11), 1552–1559.

Yoshimura, Y., Dantzker, J. L. M., & Callaway, E. M. (2005). Excitatory cortical neurons form fine-scale functional networks. *Nature*, *433*, 868–873.

Yu, Y-C., Bultje, R. S., Wang, X., & Shi, S-H. (2009). Specific synapses develop preferentially among sister excitatory neurons in the neocortex. *Nature*, *458*, 501–505.

Zago, L., Fenske, M. J., Aminoff, E., & Bar, M. (2005). The rise and fall of priming: How visual exposure shapes cortical representations of objects. *Cerebral Cortex*, *15*, 1655–1665.

Zipser, D., & Andersen, A. (1988). A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, *331*, 679–684.

Zipser, D., & Rumelhart, D.E. (1993). The neurobiological significance of the new learning models. In E. L. Schwartz (Ed.), *Computational neuroscience* (192–200). Cambridge, MA: MIT Press.