Engineering artificial systems with natural intelligence

Thesis by
Guruprasad Raghavan

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2023
Defended January 17th 2023

Guruprasad Raghavan
ORCID: 0000-0002-1970-9963

Acknowledgements

As my life at Caltech as a PhD student is reaching its final play after five *long* years, I'm grateful to have the opportunity to formally recollect and remember all the people who've walked into my life since, the phenomenal brain-storming sessions I've been a part of, the healthy arguments and disagreements along the way, the conferences I've had the fortune to present in, and finally the refreshing retreats I've attended at picturesque venues organized by the Chen Institute and BBE department at Caltech.

I believe Acknowledgements are redundant as no amount of words could ever express your love and gratitude towards those who have played a significant role in shaping you to what you are today. Words being a poor substitute to what I feel, it will be my sincere attempt to thank and state my gratitude here.

First and foremost, I would like to thank my PhD advisor and mentor, **Dr. Matt Thomson**. Although I never intended to pursue my PhD thesis with Matt (when I joined Caltech), a single exciting seminar given by Matt, on the computational modeling of the development of neural stem cells, that I attended during my first year at Caltech, altered my scientific trajectory for the best. Matt has been an extremely supportive mentor as he always provided me with the right amount of nudge when I needed it and the independence and time to explore and master a wide array of mathematical concepts. The facet that I've tried to imbibe from Matt over the years

is his ethereal excitement for anything pertaining to ”science”, a feature that kept me going during my PhD, despite the trenches I traversed during the pandemic years. Adding to that, I am deeply indebted to Matt for all the phenomenal discussions we've had about topics ranging from the pros/cons of being an entrepreneur (post PhD) to deciphering the real purpose of our lives. I'm also excited to continue our working relationship as co-founders of Yurts AI, a startup that dreams to make AI ubiquitous.

Next, comes the list of people without whom this PhD and thesis would not have seen the light of the day. My thesis committee members, **Dr. Erik Winfree**, **Dr. Ueli Rutishausser**, **Dr. Carlos Lois**. I'm very grateful to all of them for having taken out copious amounts of time from their busy schedule to brainstorm with me, help me design a road map on a yearly basis and kept checking in with me to ensure I'm making constant progress. Special thanks to Erik for embodying in me - the “Caltech spirit” while pursuing my PhD, enabling me to boldly pursue ideas that initially seemed outside my wheelhouse and also identified resources provided by Caltech to take my research a notch higher. Of the Thomson lab members, my special thanks to **Jerry** for giving me excellent feedback (especially for calling a spade - a - spade), **Ben Hoscheit** for our wide-ranging discussions and also for our runs together, **David** and **Tatyana**, for our long neurotecher discussions that stimulated an entire research project on “growing functional brains on a dish”. In the broader Caltech community, my sincere thanks to **Varun Wadia**, for not just being a great collaborator by picking up our Chen Innovator grant project when I was swamped with my thesis project, but also for being a warm friend and being a sounding board for all my frustrations when my motivation was an all-time low.

Apart from direct technical addition to my thesis, there are many people who have helped me successfully complete my PhD and have played a giant role in picking me

up when I was down. A large share of the credit goes to my Krishna-conscious (KC) circle of friends, **Raju**, **Akshay**, **Kiran**, **Kunal** & **Yamini**, **Pavan**, **Hari Bhakta**, **Sacidulal Prabhu**, **Vraja Kumuda**, **Vinay**, **Sindhu** for inspiring me with the way they lead their lives, for training me in the eastern philosophical ideals of life, and guiding me on the practical aspects of applying the philosophy to my life; For instance, by ensuring I wake up by 4:30/5AM, practice vocal meditation for a minimum of 30 min in the morning and by inspiring me to lead a life devoted to serving all living entities. Eternally grateful for their association as it has helped me own my mornings, and gradually embrace my higher self.

In addition to my KC circle, my sincere gratitude to my wing-mates from IIT Madras (**Atul**, **Shara**, **Nikhil**, **Suraj**, **Shantanu**, **Abhishek**, **Aritra**) who have been an integral part of my life over the last 12 years and my dear friends from my time at Carnegie Mellon University, **Sahil Rastogi**, **Lekha Mohan** for teaching me how to dance through life (not just metaphorically).

Last, but not the least, I would like to thank my family, my parents, my sister (**Shruthi**) and **Anusha**, my better half, the latest addition to my family! I have an amazing family, unique in many ways, and the stereotype of a perfect family in many others. They have been my pillar of support through-out these years and a chat with them has not just helped me wade through depression but has got me rejuvenated and excited to keep aiming higher. My special gratitude to Anusha for showing up eagerly to every single practice talk I've given over the last 2-3 years, for her incredible patience to hear the same talk a million times with the same enthusiasm, and for giving me both, emotional support and technical feedback before my job-talks, my defense and also while completing this thesis.

As one gets a PhD from Caltech only once, I would also like to take this opportunity to dedicate my PhD research and this thesis to the very people that have kept me motivated since the very beginning and are the 'behind-the-scenes' heroes in my journey: my family, the Vaishnavas, the exalted Acharyas (**Sri Madhvacharya**, **Sri Raghavendra Swamy**, **Srila Prabhupada**), and ultimately **Sri Krishna**.

Abstract

Although Deep neural networks achieve human-like performance on a variety of perceptual and decision-making tasks, they perform poorly when confronted with changing tasks or goals, and broadly fail to match the flexibility and robustness of human intelligence. Additionally, artificial neural networks rely heavily on human-designed, hand-programmed architectures for their remarkable performance. In this thesis, I work towards achieving two goals: (i) development of a set of mathematical frameworks inspired by facets of natural intelligence, to endow artificial networks with flexibility and robustness, two key traits of natural intelligence; and (ii) inspired by the development of the biological vision system, I propose an algorithm that can 'grow' a functional, layered neural network from a single initial cell, with the aim of enabling autonomous development of artificial networks akin to living neural networks.

For the first goal of endowing networks with flexibility and robustness, I propose a mathematical framework to enable continuous training of neural networks on a range of objectives by constructing path connected sets of networks, resulting in the discovery of a series of networks with equivalent functional performance on a given machine learning task. In this framework, I view the weight space of a neural network as a curved Riemannian manifold and move a network along a functionally invariant path in weight space while searching for networks that satisfy secondary objectives. A path-sampling algorithm trains computer vision and natural language processing networks with millions of weight parameters to learn a series of classification tasks

without performance loss while accommodating secondary objectives including network sparsification, incremental task learning, and increased adversarial robustness. Broadly, for achieving this goal, I conceptualize a neural network as a mathematical object that can be iteratively transformed into distinct configurations by the path-sampling algorithm to define a sub-manifold of networks that can be harnessed to achieve user goals.

For the second goal of 'growing' artificial neural networks in a manner similar to living neural networks, I develop an approach inspired by the mechanisms employed by the early visual system to wire the retina to the lateral geniculate nucleus (LGN), days before animals open their eyes. I find that the key ingredients for robust self-organization are (a) an emergent spontaneous spatiotemporal activity wave in the first layer and (b) a local learning rule in the second layer that 'learns' the underlying activity pattern in the first layer. As the bio-inspired developmental rule is adaptable to a wide-range of input-layer geometries and robust to malfunctioning units in the first layer, it can be used to successfully grow and self-organize pooling architectures of different pool-sizes and shapes. The algorithm provides a primitive procedure for constructing layered neural networks through growth and self-organization. Finally, I also demonstrate that networks grown from a single unit perform as well as hand-crafted networks on a wide variety of static (MNIST recognition) and dynamic (gesture-recognition) tasks. Broadly, the work in the second section of this thesis shows that biologically inspired developmental algorithms can be applied to autonomously grow functional 'brains' in-silico.

Published Content and Contributions

Raghavan, Guruprasad and Matt W. Thomson (2022). "Engineering flexible machine learning systems by traversing functionally invariant paths in weight space." In: In review at Nature Machine Intelligence. url: https://arxiv.org/abs/arXiv:2205.00334.

G.R proposed the paper, designed the algorithms, provided simulations and co-wrote the paper with M.W.T.

Bhamidipati, Pranav, Guruprasad Raghavan, and Matt W. Thomson (2021). "Traversing Geodesics to Grow Biological Structures." In: NeurIPS 2021 AI for Science Workshop.

G.R provided the simulations and algorithm, and co-wrote the paper with P.B

Raghavan, Guruprasad and Matthew W. Thomson (Dec. 2021). Resilience determination and damage recovery in neural networks. US Patent App. 17/349,743. url: https://patentimages.storage.googleapis.com/31/8f/0c/70d2c7ff3e6528/US20210397964A1.pdf

G.R provided the simulations and algorithm for the patent.

Raghavan, Guruprasad, Matthew W. Thomson, and Cong Lin (July 2021). Self organization of neuromorphic machine learning architectures. US Patent App. 17/127,762. url: https://patents.google.com/patent/US20210224633A1/en
G.R provided the simulations and algorithm for the patent.

Raghavan, Guruprasad, Jiayi Li, and Matt W. Thomson (2020). "Geometric algorithms for predicting resilience and recovering damage in neural networks." In: NeurIPS 2020 workshop on Deep Learning through Information Geometry.
G.R proposed the paper, designed the algorithms, provided simulations and co-wrote the paper with J.L and M.W.T.

Raghavan, Guruprasad, Cong Lin, and Matt W. Thomson (2020). "Self-organization of multi-layer spiking neural networks." In: arXiv preprint. url: https://arxiv.org/abs/2006.06902.
G.R provided simulations, designed algorithms and co-wrote paper with C.L and M.W.T

Raghavan, Guruprasad and Matt W. Thomson (2020). "Sparsifying networks by traversing Geodesics." In: NeurIPS 2020 workshop on Differential Geometry meets Deep Learning.
G.R proposed the paper, designed the algorithms, provided simulations and co-wrote the paper with M.W.T.

Talukder, Sabera$*$, Guruprasad Raghavan$*$, and Yisong Yue (2020). "Architecture Agnostic Neural Networks." In: NeurIPS 2020 workshop on BabyMind [Spotlight talk ($*$Co-authors)]. url: https://arxiv.org/abs/2011.02712
G.R provided simulations, designed algorithms and co-wrote paper with S.T and Y.Y

Raghavan, Guruprasad and Matt W. Thomson (2019). “Neural networks grown and self-organized by noise.” In: Advances in Neural Information Processing Systems 32. https://proceedings.neurips.cc/paper_files/paper/2019/file/1e6e0a04d20f50967c64dac2d639a577-Paper.pdf.

G.R proposed the paper, designed the algorithms, provided simulations and co-wrote the paper with M.W.T.

Contents

List of Figures

Chapter 1

Introduction

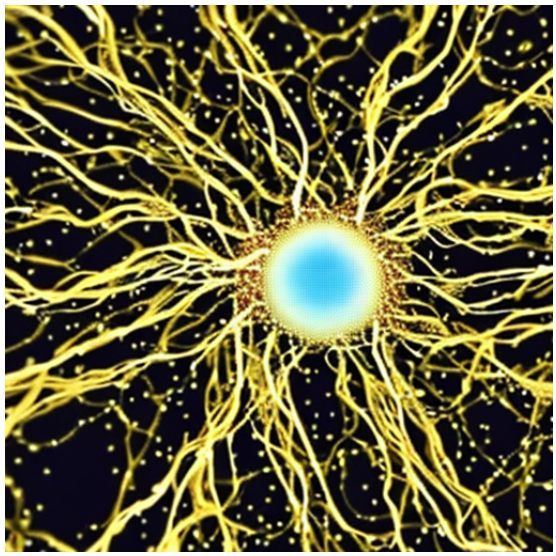


Figure 1.1: **Neuron-art: generated by text-to-image AI.**

"The intelligence attributed to machines almost always relies on rendering human skill invisible."

Section 1.1

Thesis statement

One of the biggest unresolved questions plaguing humanity to date, is the question of the nature of the self, the nature of life, the reality (or illusion) of the world around us, and ultimately our purpose (if there exists one). The questions concerning the nature of the self or life are not 'new' or 'modern' by any stretch of imagination. Prior to the availability of high-end experimental methods or technological tools, the primary mode of addressing (or answering) such questions was through philosophy and logic. However, in this thesis, I attempt to address a facet of this question by engaging the latest technological tools and developments in the fields of Artificial intelligence and Machine learning.

To address the specific question concerning the nature of life, mankind has a long history of attempting to recreate key symptoms of life (like movement, self-organization, homeostasis, self-replication, natural intelligence) using the mechanics of their contemporary technology. For instance, the earliest models were works of art, that comprised of paintings and statuettes with articulated arms and legs to capture the dynamic form of living beings. We also know that in ancient theatres (as described in the epics of the Ramayana [Orenstein, 2006]), puppets were built and controlled by human operators to mimic living beings and their social interactions. On the front of mimicking the workings of nature, the earliest mechanical device built by the Egyptians, called the Clepsydra [Hamilton, 1746] (Egyptian water clock), was used for tracking the position of the sun by controlling the water flow from an orifice, enabling an accurate measurement of time. In the same vein, I believe that endowing "natural intelligence," one of the key symptoms of life to artificial systems is one among many strategies to understand the workings of the self, and ultimately would assist in our understanding of the nature of life. Intelligence can be observed across

most species of life, and it is not limited to humans. Different animals have evolved different cognitive abilities and adaptations that allow them to survive and thrive in their specific environments. For example, some animals, such as primates and dolphins, have highly developed social intelligence and are able to communicate and cooperate with others in sophisticated ways. Other animals, such as birds and some insects, have developed advanced problem-solving and navigation abilities. From the mud-wasps building their well-measured nests [Smith, 1978], the beavers that build their dams [Żurowski, 1992], the song-birds that robustly sing their song [Wang et al., 2022], to the humans that can flexibly adapt to different environments, build technological tools, and finally strive to build AI systems that can 'outsmart' them, we can identify features of natural intelligence across different species of life.

Although the field of Artificial Intelligence (AI) has made rapid progress towards simulating natural intelligence in the last few years by outperforming humans on a few tasks (like image recognition, number arithmetic, game playing), they still fail to replicate many key principles of natural intelligence (like robustness, flexibility, continual learning, to name a few).

My thesis has been geared towards decoding and deciphering some of the key principles of natural intelligence with the goal of building artificial systems possessing these principles. The key approach that I have adopted for addressing this question during my PhD is to compare biological systems endowed with natural intelligence to today's modern Artificial Intelligence systems (primarily focused on systems built using artificial neural networks). Having identified gaps between the two by recognizing principles of intelligence present in biological systems and lacking in today's artificial systems, I have developed algorithms and procedures to endow artificial systems with those attributes of intelligence.

The advantages of developing artificial intelligence systems by closely examining biological systems is two fold. Firstly, it catalyzes the next-generation of AI sys-

tems moving one step closer to natural intelligence and secondly, it provides theories regarding biological mechanisms and illuminates deep rooted principles at play in natural systems.

Section 1.2

Background

I will review some of the previous work on this problem concerning the identification of principles of intelligence and their successful integration into artificial systems. I will begin by describing some of the very early ideas in this field as they are radically different from our modern approach, and then continue to the description of modern frameworks.

1.2.1. Ancients and AI

The residents of the Vedic civilization (going back more than 5000 years, circa 3000 BC) were one of the first to ponder the nature of life, and also, to my best knowledge, were the first to present a literary record of a competition between artificial birds and living birds. The statement has been recorded in the ancient text called the Srimad Bhagavatam, written around 3000 BC in Sanskrit by Saint Vyasadeva.

हंसपारावतव्रातैस्तत्र तत्र निकूजितम्। कृत्रिमान् मन्यमानै: स्वानधिरुह्याधिरुह्य च॥

Here, the author describes a King named "Kardama Muni" who lived in a large palace that housed a multitude of live swans and pigeons, as well as *artificial swans and pigeons so lifelike* that the real swans rose above them again and again, thinking them live birds like themselves [Bhaktivedanta, 1976]. Although being the earliest recorded evidence of the human society's interest in building biologically-inspired artificial systems, the text does not describe a mechanistic explanation of how this may be built, or how it was built.

Fast forward 3000 years to circa 250 BC, we find that the ancient Greek engineers were building human-like robots (e.g. automatic servant of Philon) [Valavanis, Vachtsevanos, and Antsaklis, 2007] to serve a jug of water, built using a complex mechanism of springs, weights and air pressure. Additionally, the 2000-year old Antikythera mechanism, considered to be a precursor of the computer, was developed by the Greeks to forecast astronomical events using gears and dials.

1.2.2. Modern 'birth' of AI

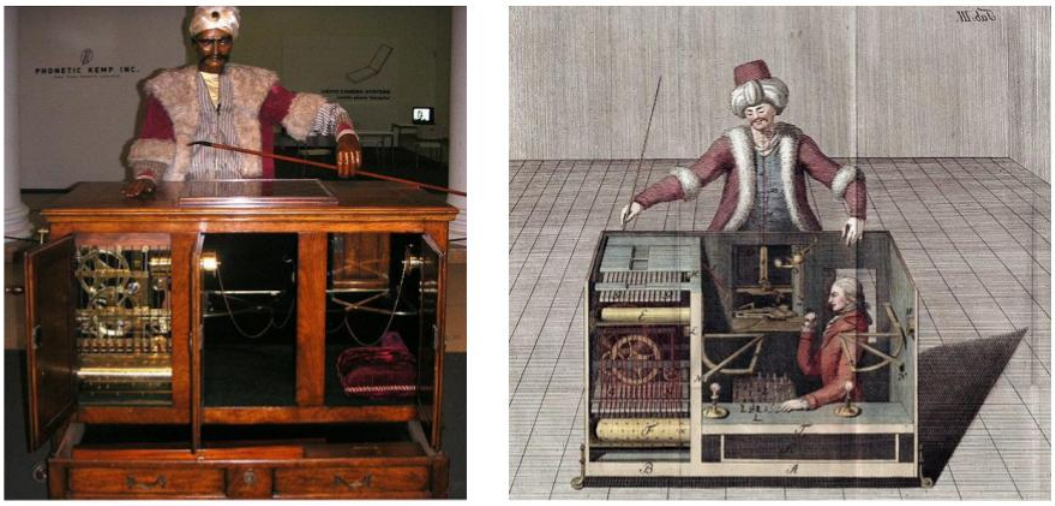


Figure 1.2: **The "Turk" and its influence on modern AI** (a) Wolfgang von Kempelen's chess-playing "Turk." (b) A human chess master concealed in the "Turk" automaton machine. Images retrieved from `https://en.wikipedia.org/wiki/Mechanical_Turk` and `https://www.uh.edu/engines/epi2765.htm`

In the year 1770, Hungarian author and inventor Wolfgang von Kempelen presented his invention, 'The Turk' [Stephens, 2022] at the court of the Austrian empress Maria Theresa. Kempelen's invention, 'The Turk,' was a life-sized automaton carved out of maple wood, dressed in Ottoman robes, sitting behind a wooden cabinet with a chessboard on top (Fig. 1.2a). He claimed that his machine can defeat any member of the court in the game of chess. When one of the empress' advisors took up the

challenge, he opened the doors of the cabinet and showed the intricate network of levers and cogs, that functioned with a clockwork-like mechanism. He subsequently inserted a key into the machine, wound it up and brought the automaton to life, enabling it to lift its wooden arm to make the first move. Within 30 minutes, the Turk defeated its opponent and became an instant sensation. Thereafter Kempelen toured the Turk around Europe and it has been documented to have defeated many formidable minds of that time, including Benjamin Franklin and Frederick the Great.

After Kempelen's death, the Turk was taken on a tour to Britain, and was paired to play with Charles Babbage. Although Babbage, the famed British mathematician and inventor, lost twice to the Turk, he successfully figured out that the Turk was not an 'intelligent' automaton but concealed a human chess master who was able to observe the chess board and controlled the Turk's movements (Fig. 1.2b). Historians suggest that the encounter with the Turk was the seed that led to the design of the first mechanical computer, the "Difference engine." Since then, there were many theoretical and experimental attempts to design and engineer intelligent artificial systems. One of the first designs and thought experiments, like the Analytical Engine, was conceived to have numerical abilities to perform a wide range of mathematical and logical operations, including addition, subtraction, multiplication, division, and evaluation of functions.

However, the idea of using brain-inspired computation or "neuromorphic" architecture sprouted in the early 1940s and 1950s, when researchers began to study the structure and function of neurons in the brain.

McCulloch-Pitts (MCP) neuron model. The MCP model was one of the first models to derive detailed neurobiological inspiration from the connectivity structure of the neurons in the brain to construct computing machines. The MCP model [McCulloch and Pitts, 1990], also known as the threshold logic unit (TLU), is a type

of artificial neuron developed in 1943 by Warren McCulloch and Walter Pitts. It is a simple model of a biological neuron that is designed to mimic the way that biological neurons process and transmit information. The MCP model consists of a set of inputs and a single output, and it is able to produce an output based on whether the weighted sum of its inputs exceeds a certain threshold. The MCP model is considered to be the first model of an artificial neuron, paving the way for much complex artificial neural networks.

Fukushima's Neocognitron. In addition to deriving structural inspiration from the brain (like the MCP model), Kunihiko Fukushima designed one of the first artificial neural network inspired by the Hubel and Weisel model [Hubel and Wiesel, 1959; Hubel and Wiesel, 1963] of functional connectivity and receptive fields in the cat's striate cortex, namely the simple and complex cells, to recognize patterns in visual images [Fukushima, 1980].

Deep networks. The McCullock-Pitts model [McCulloch and Pitts, 1990] of a simple neuron and the Neocognitron [Fukushima, 1980] architecture of simple and complex cells in the visual cortex were groundbreaking at the time of their development and played a key role in prompting the development of deep convolutional networks (LeCun) [LeCun, Boser, et al., 1990; LeCun, Bottou, et al., 1998; LeCun, Haffner, et al., 1999] and similar neural network architectures. The early 90s witnessed the advent of deep neural networks and corresponding algorithms to train and optimize neural networks on a large set of cognitive tasks. The first artificial neural networks were developed for processing visual images and were trained by the (now ubiquitous) back-propagation technique [Rumelhart, Hinton, and Williams, 1986]. In addition to research investment in developing novel network architectures, training algorithms, there was a push towards creating benchmarking datasets. One of the most popular image datasets compiled were MNIST, CIFAR-10, Caltech-101, [LeCun, Boser, et al.,

1990; Fei-Fei, Fergus, and Perona, 2004] which contained images of hand-written digits (0 to 9), images of 10 classes of objects (like birds, ships, automobiles) and images of 101 classes of objects (like faces, leopards, ceiling fans to name a few), respectively.

Deep Networks coupled with hardware engineering. Although deep network architectures were powerful mathematical tools, they were very slow to train and were not able to generalize very well to out-of-distribution data[1]. In 2012, AlexNet [Krizhevsky, Sutskever, and Hinton, 2012] developed by Alex Krichevzky and Geoff Hinton, was one of the first implementations of training neural networks on multiple GPU's (Graphical processing units), which increased the speed of training exponentially, and enabled networks to be trained on large corpii of images (or data) endowing them with better generalization capabilities.

Since then, a large number of deep network architectures have been developed for processing images, text, and audio. Some of the prominent ones that are still in use today are: AlexNet [Krizhevsky, Sutskever, and Hinton, 2012], ResNet [He et al., 2016], VGG [Simonyan and Zisserman, 2014], DenseNet [Huang et al., n.d.], MobileNet [Howard et al., 2017], SqueezeNet [Iandola et al., 2016], Variational Autoencoders [Kingma and Welling, 2013], to name a few.

Large language models. Most deep network architectures built until early 2010 were small networks with upto a million parameters. The implementation of networks on multiple GPU's propelled the advent of larger networks, with 100's of millions of parameters (e.g. AlexNet, ResNet, early transformers like BERT [Devlin et al., 2018], GPT [Radford et al., 2018]).

Although deep networks developed between 2010 and 2015 were 2 orders of magnitude larger than the ones developed in the early 2000's, GPU hardware advances

[1]Generalization remains a big challenge today too, because the bar for generalization has significantly grown over the last 10-15 years.

and access to very large training datasets, propelled the development of much larger models, almost 3 orders of magnitude (with $\sim$ 10 billion parameters) larger than previous ones.

Most of these very large models (like GPT-3 [Brown et al., 2020], LaMDA [Thoppilan et al., 2022], PaLM [Chowdhery et al., 2022]) are developed for natural language tasks, like text generation, sentiment analysis, text summarization and have been trained on very large text corpii, such as the entire wikipedia, C4 (Colossal, cleaned version of Common Crawls' web crawl corpus) [Dodge et al., 2021] with a raw size of 500 TiB.

We observe from Fig. 1.3 that we are currently (circa 2022) very close to engineering an artificial system as large as the human brain, with only 3 orders of magnitude to go. The human brain has roughly 100 billion neurons, where each neuron makes 10,000 connections, resulting in a total of 1000 trillion synaptic connections in the brain, while the largest artificial neural network built to-date has a trillion parameters.

Section 1.3

Are modern AI systems intelligent?

The unprecedented developments in the AI/ML field over the last 5 years (during my PhD) make us wonder if we have succeeded (as a community) in building an artificial system that is very life-like, endowed with intelligence?

To address this question, we need to define a set of key principles of intelligence that a system must possess, before being recognized as an 'intelligent' system.

1.3.1. Key principles of intelligence

In this section, I lay out some of the key principles of intelligence that have been inspired by my observation of natural intelligence at play across various species of living beings, and principles that an 'intelligent' system must possess.

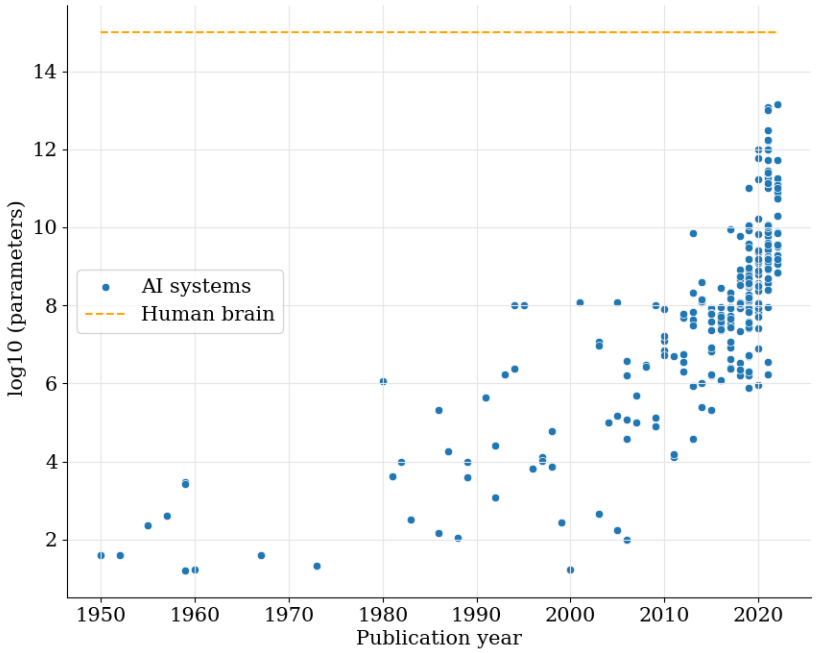


Figure 1.3: **Size of AI systems engineered over the last 60 years.**

- **Flexibility:** Living entities continuously gather new information by interacting with their environment and 'flexibly' update their internal hardware (biological networks) and internal model of the environment. They update their biological neural networks without abruptly forgetting all their past information learnt and acquired before [Neumann and Ammons, 1957; Cichon and Gan, 2015; Hayashi-Takagi et al., 2015; Yang, Pan, and Gan, 2009; Yang, Lai, et al., 2014]. On the other hand, current artificial neural networks developed in the AI/ML community struggle to incrementally update their networks without abruptly forgetting all their previous stored information, commonly referred to as catastrophic forgetting. Therefore, the ability to flexibly update their artificial networks as they interact with the environment is an important essential feature of intelligence.

- **Robustness**: Biological neural networks have evolved to maintain functional

performance despite significant circuit damage [Kitano, 2004; Félix and Barkoulas, 2015; Perez-Nieves et al., 2021]. Similarly, artificial intelligent systems should be resistant to adversarial damage, both incidental and intentional, the former being a random chance attack from the environment while the latter being a targeted sabotage.

- **Autonomous development:** As all living entities grow and self-organize their biological architecture from a single seed-cell, intelligent systems must not rely on humans for their hand-programmed design and development, but must be capable of autonomous development from a single "seed-cell."[2]

The principles described above are not necessarily exhaustive, but they provide a framework for understanding the key characteristics that systems should possess in order to be termed 'intelligent.'

Section 1.4

What's in this thesis?

The chapters that follow in this thesis will be a deep-dive into each of these principles of intelligence. In each chapter, I will compare and contrast biological brains (natural intelligence) with the latest artificial neural networks (ANN's) to identify flaws in the AI system. This will be followed by an in-depth treatment by reasoning mathematically and conceptually why current AI systems do not have a particular attribute. Finally, the chapter will end with a systematic algorithm or a procedure to engineer the intelligence attribute in the AI system (specifically focusing on artificial neural networks).

The chapters that follow will heavily borrow from published, in-review and unpublished content produced during my PhD. Specifically, Chapters 2 and 3 will borrow

[2] Although the entire development from the seed-cell can be made autonomous, the rules (or meta-rules) governing development from the seed-cell continues to be designed by the human.

heavily from the following papers:

- **Chapter 2: Engineering flexible and robust machine learning systems** borrowed from Raghavan and Thomson, 2022 and Raghavan, Li, and Thomson, 2020.
- **Chapter 3: Autonomous self-organization of neural networks from noise** borrowed from Raghavan and Thomson, 2019 and Raghavan, Lin, and Thomson, 2020.

References

Bhaktivedanta, Abhay C. (1976). "Chapter 20." In: *Srimad Bhagavatam*. Los Angeles, CA, USA: Bhaktivedanta Book Trust (BBT).

Brown, Tom et al. (2020). "Language models are few-shot learners." In: *Advances in Neural Information Processing Systems* 33, pp. 1877–1901.

Chowdhery, Aakanksha et al. (2022). "Palm: Scaling language modeling with pathways." In: *arXiv preprint arXiv:2204.02311*.

Cichon, Joseph and Wen-Biao Gan (2015). "Branch-specific dendritic Ca2+ spikes cause persistent synaptic plasticity." In: *Nature* 520.7546, pp. 180–185.

Devlin, Jacob et al. (2018). "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805*.

Dodge, Jesse et al. (2021). "Documenting large webtext corpora: A case study on the colossal clean crawled corpus." In: *arXiv preprint arXiv:2104.08758*.

Fei-Fei, Li, Rob Fergus, and Pietro Perona (2004). "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories." In: *2004 Conference on Computer Vision and Pattern Recognition Workshop*. IEEE, pp. 178–178.

Félix, Marie-Anne and Michalis Barkoulas (2015). "Pervasive robustness in biological systems." In: *Nature Reviews Genetics* 16.8, pp. 483–496.

Fukushima, Kunihiko (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." In: *Biological Cybernetics* 36.4, pp. 193–202.

Hamilton, Charles (1746). "XIV. A description of a clepsydra or water-clock." In: *Philosophical Transactions of the Royal Society of London* 44.479, pp. 171–174.

Hayashi-Takagi, Akiko et al. (2015). "Labelling and optical erasure of synaptic memory traces in the motor cortex." In: *Nature* 525.7569, pp. 333–338.

He, Kaiming et al. (2016). "Deep residual learning for image recognition." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Howard, Andrew G et al. (2017). "Mobilenets: Efficient convolutional neural networks for mobile vision applications." In: *arXiv preprint arXiv:1704.04861*.

Huang, G. et al. (n.d.). "Densely Connected Convolutional Networks. arXiv. 2016 doi: 10.48550." In: *arXiv preprint arXiv.1608.06993* 1608 ().

Hubel and Wiesel (1959). "Receptive fields of single neurones in the cat's striate cortex." In: *The Journal of Physiology* 148.3, p. 574.

— (1963). "Shape and arrangement of columns in cat's striate cortex." In: *The Journal of Physiology* 165.3, pp. 559–568.

Iandola, Forrest N. et al. (2016). "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." In: *arXiv preprint arXiv:1602.07360*.

Kingma, Diederik P. and Max Welling (2013). "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114*.

Kitano, Hiroaki (2004). "Biological robustness." In: *Nature Reviews Genetics* 5.11, pp. 826–837.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Advances in Neural Information Processing Systems* 25.

LeCun, Yann, Bernhard E. Boser, et al. (1990). "Handwritten digit recognition with a back-propagation network." In: *Advances in Neural Information Processing Systems*, pp. 396–404.

LeCun, Yann, Léon Bottou, et al. (1998). "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Yann, Patrick Haffner, et al. (1999). "Object recognition with gradient-based learning." In: *Shape, Contour and Grouping in Computer Vision*. Springer, pp. 319–345.

McCulloch, Warren S. and Walter Pitts (1990). "A logical calculus of the ideas immanent in nervous activity." In: *Bulletin of Mathematical Biology* 52.1, pp. 99–115.

Neumann, Eva and Robert B. Ammons (1957). "Acquisition and long-term retention of a simple serial perceptual-motor skill." In: *Journal of Experimental Psychology* 53.3, p. 159.

Orenstein, Claudia (2006). "Puppetry: A world history." In: *Theatre Journal* 58.2, pp. 375–376.

Perez-Nieves, Nicolas et al. (2021). "Neural heterogeneity promotes robust learning." In: *Nature Communications* 12.1, pp. 1–9.

Radford, Alec et al. (2018). "Improving language understanding by generative pre-training." In: *OpenAI blog*.

Raghavan, Guruprasad, Jiayi Li, and Matt W. Thomson (2020). "Geometric algorithms for predicting resilience and recovering damage in neural networks." In: *NeurIPS 2020 workshop on Deep Learning through Information Geometry*.

Raghavan, Guruprasad, Cong Lin, and Matt W. Thomson (2020). "Self-organization of multi-layer spiking neural networks." In: *arXiv preprint arXiv:2006.06902*.

Raghavan, Guruprasad and Matt W. Thomson (2019). "Neural networks grown and self-organized by noise." In: *Advances in Neural Information Processing Systems* 32. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/1e6e0a04d20f50967c64dac2d639a577-Paper.pdf.

— (2022). "Engineering flexible machine learning systems by traversing functionally invariant paths in weight space." In: *In review at Nature Machine Intelligence arXiv:2205.00334*.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning representations by back-propagating errors." In: *Nature* 323.6088, pp. 533–536.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556*.

Smith, Andrew P. (1978). "An investigation of the mechanisms underlying nest construction in the mud wasp Paralastor sp.(Hymenoptera: Eumenidae)." In: *Animal Behaviour* 26, pp. 232–240.

Stephens, Elizabeth (2022). "The mechanical Turk: A short history of 'artificial artificial intelligence'." In: *Cultural Studies*, pp. 1–23.

Thoppilan, Romal et al. (2022). "Lamda: Language models for dialog applications." In: *arXiv preprint arXiv:2201.08239*.

Valavanis, K., G. Vachtsevanos, and PJ Antsaklis (2007). "Technology and autonomous mechanisms in the mediterranean: From ancient Greece to Byzantium." In: *2007 European Control Conference (ECC)*. IEEE, pp. 263–270.

Wang, Bo et al. (2022). "Unsupervised Restoration of a Complex Learned Behavior After Large-Scale Neuronal Perturbation." In: *bioRxiv, https://www.biorxiv.org/content/10.1101/2022.09.09.507372v1*.

Yang, Guang, Cora Sau Wan Lai, et al. (2014). "Sleep promotes branch-specific formation of dendritic spines after learning." In: *Science* 344.6188, pp. 1173–1178.

Yang, Guang, Feng Pan, and Wen-Biao Gan (2009). "Stably maintained dendritic spines are associated with lifelong memories." In: *Nature* 462.7275, pp. 920–924.

Żurowski, Wirgiliusz (1992). "Building activity of beavers." In: *Acta Theriologica* 37.4, pp. 403–411.

Chapter 2

Engineering flexible AI systems by traversing functionally invariant paths (FIP)

Figure 2.1: **Neural networks weight space manifold art: generated by text-to-image AI.**

This chapter focuses on the "flexibility" and "robustness" attribute of intelligent systems. As stated in section 1.3.1, I define 'flexibility' as the systems' ability to retain previous knowledge and experiences while gathering and incorporating new information obtained by continuously interacting with the environment, and 'robustness' as the ability to maintain functional performance when subjected either to an intentional adversarial environment or a chance attack from the environment.

Section 2.1

ANN's suffer from catastrophic forgetting

Artificial neural networks out-perform humans on tasks ranging from image recognition and game playing to protein structure prediction [Silver et al., 2017; Jumper et al., 2021; Krizhevsky, Sutskever, and Hinton, 2012]. However, artificial neural networks fail to replicate the flexibility and robustness of human intelligence [Sünderhauf et al., 2018; Smale, 1998; Lee and Mumford, 2003]. Humans can learn new tasks and accommodate novel goals with minimal instruction and without loss of performance on existing tasks. Unlike humans, deep neural networks suffer catastrophic forgetting (CF) or significant performance decay, when trained to perform additional tasks or integrate new information [McCloskey and Cohen, 1989; Ratcliff, 1990]. For example, a network trained to recognize images of hand-written digits [Simard et al., 1998] will 'forget' the digit recognition task when trained to recognize additional objects like letters or faces. In addition to well-known flexibility limits, deep neural networks have other pathologies, like vulnerability to targeted corruption of input data or adversarial fragility where small, imperceptible changes in the input data can cause complete failure of network performance.

2.1.1. A single optimal weight configuration susceptible to CF

In artificial neural networks, network function is encoded in the mathematical weights that determine the strength of connections between neural units (Fig. 2.1, Fig. 2.2). Deep learning procedures train multi-layered neural networks to solve problems by adjusting the weights of a network based on an objective function that encodes the performance of a network on a specific task. Standard learning methods, like back-propagation and gradient descent [Rumelhart, Hinton, and Williams, 1986], adjust network weights to define a single, optimal weight configuration to maximize performance on a task specific objective function using training data. Training the network on new tasks through the traditional paradigm of stepping along the gradient of the task-specific objective function adjusts the networks' weights, inevitably resulting in the loss of information from previous tasks.

The weight adjustment problem underlies other challenges in modern machine learning. For example, it is advantageous to prune or sparsify a network to minimize the number of non-zero weights and thus reduce network memory and power consumption. Just like multi-task learning, network sparsification requires the adjustment of network weights while maintaining function, and sparsification procedures often proceed through heuristic weight pruning strategies. For adversarial robustness, a central goal is to identify ensembles of networks that perform a task with distinct weight configurations and thus avoid vulnerabilities associated with a single weight configuration.

2.1.2. Drifting in biological neural networks could enable flexibility

Unlike contemporary artificial neural nets, neural networks in the human brain perform multiple functions and can flexibly switch between different functional configurations based on context, goals or memory [Minxha et al., 2020]. Neural networks in the brain are hypothesized to overcome the limitations of a single, optimal weight

configuration and perform flexible tasks by continuously 'drifting' their neural firing states and neural weight configurations, effectively generating large ensembles of degenerate networks [Mau, Hasselmo, and Cai, 2020; Stringer et al., 2019]. Fluctuations might enable flexibility in biological systems by allowing neural networks to explore a series of network configurations while responding to sensory input.

Section 2.2

Mathematical framework

Broadly inspired by the 'drifting' observed in biological networks, we develop a geometric framework and algorithm to construct path connected sets of neural networks that solve a given machine learning task. Conceptually, we consider path-connected sets of neural networks, rather than single-networks (isolated points in weight space) to be the central objects of study and application. By building sets of networks rather than single networks, we search within a sub-manifold of weight space for networks that solve a given machine learning problem while accommodating a broad range of secondary goals. We view a neural networks' weight space as a Riemannian manifold equipped with a distance metric that represents task performance. Our core algorithm identifies functionally invariant paths in weight space that maintain network performance while 'searching-out' for other networks that satisfy additional objectives like sparsification or mitigating catastrophic interference. We demonstrate that the path sampling algorithm achieves high performance on a series of meta-tasks: sequential task learning, network sparsification, and adversarial robustness on a range of vision and language networks with millions of parameters obtaining performance similar to domain specific approaches.

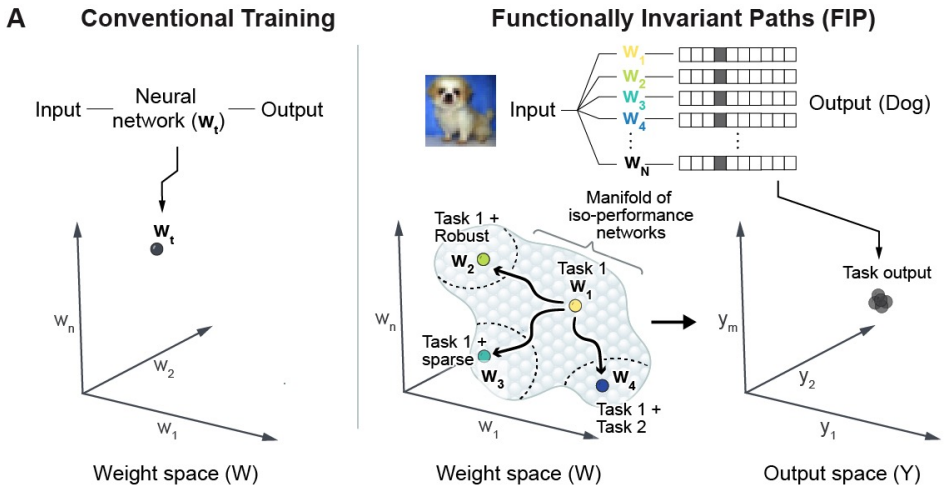


Figure 2.2: **Geometric Framework for constructing functionally invariant paths (FIP) in weight space.** (Left) Conventional training on a task finds a single trained network ($\mathbf{w_t}$) solution. (Right) The FIP strategy discovers a submanifold of iso-performance networks ($\mathbf{w_1}$, $\mathbf{w_2}$, ...,$\mathbf{w_N}$) for a task of interest, enabling the efficient search for networks endowed with adversarial robustness ($\mathbf{w_2}$), sparse networks with high task performance ($\mathbf{w_3}$) and for learning multiple tasks without forgetting ($\mathbf{w_4}$).

Construction of functionally invariant paths in weight space

We develop a mathematical framework that allows us to define and explore path-connected sets of neural networks that have divergent weight values but similar output on training data. We view the weight-space of a neural network as a Riemannian manifold equipped with a local distance metric [Amari, 2016; Benn and Tucker, 1987]. Using differential geometry, we construct paths through weight space that maintain the functional performance of a neural network while adjusting network weights to flow along a secondary goal. The secondary goal can be general, so that the framework can be applied to train networks on new classification tasks, to sparsify networks, and also to mitigate adversarial fragility.

The defining feature of a Riemannian manifold is the existence of a local distance metric. We construct a distance metric in weight space that defines the distance between two nearby networks to be their difference in output. We consider a neural network to be a smooth function, $f(\mathbf{x}; \mathbf{w})$, that maps an input vector, $\mathbf{x} \in \mathbb{R}^k$, to

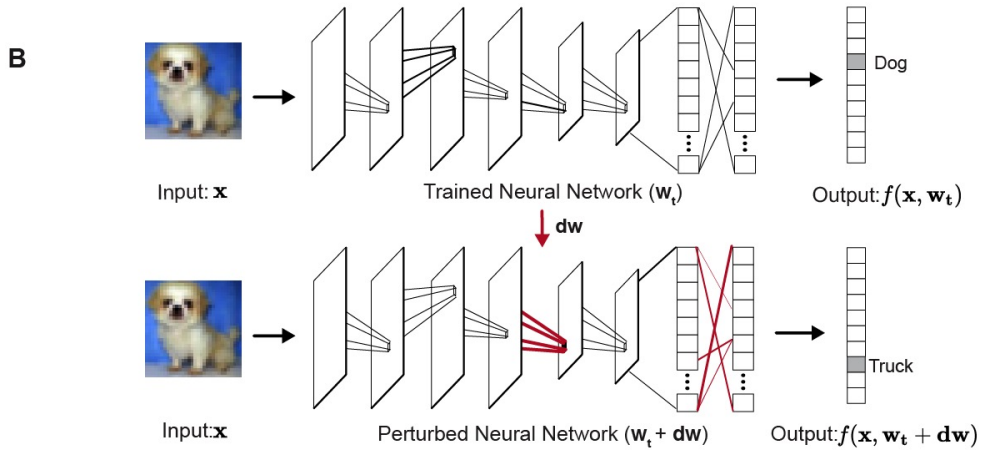


Figure 2.3: **Geometric Framework for constructing functionally invariant paths (FIP) in weight space.** (Top) A trained convolutional neural network with weight configuration ($\mathbf{w_t}$), represented by lines connecting different layers of the network, accepts an input image, $\mathbf{x}$, and produces a 10-element output vector, $f(\mathbf{x}, \mathbf{w_t})$. (Below) Perturbation of network weights by $\mathbf{dw}$ results in a new network with weight configuration $\mathbf{w_t} + \mathbf{dw}$ with an altered output vector, $f(\mathbf{x}, \mathbf{w_t} + \mathbf{dw})$, for the same input, $\mathbf{x}$.

an output vector, $f(\mathbf{x}; \mathbf{w}) = \mathbf{y} \in \mathbb{R}^{\mathrm{m}}$, where the map is parameterized by a vector of weights, $\mathbf{w} \in \mathbb{R}^{\mathrm{n}}$, that are typically set in training to solve a specific task. We refer to $W = \mathbb{R}^n$ as the *weight space* of the network, and we refer to $\mathcal{Y} = \mathbb{R}^m$ as the *output space* as shown in Fig. 2.3B [Mache, Szabados, and Bruin, 2006]. For pedagogical purposes, we will consider the action of f on a single input, $\mathbf{x}$. In the supplement we show that our results extend naturally to an arbitrary number of inputs $\mathbf{x_i}$.

We initially ask how the output, $f(\mathbf{x}; \mathbf{w})$, of a given neural network changes for small changes in network weights (Fig. 2.3). Given a neural network with weights $\mathbf{w_t}$, a fixed input $\mathbf{x}$, we can compute the output of the perturbed network, $\mathbf{w_t} + \mathbf{dw}$ for an infinitesimal weight perturbation, $\mathbf{dw}$ as

$$f(\mathbf{x}, \mathbf{w_t} + \mathbf{dw}) \approx f(\mathbf{x}, \mathbf{w_t}) + \mathbf{J}_{\mathbf{w_t}}\ \mathbf{dw}, \tag{2.1}$$

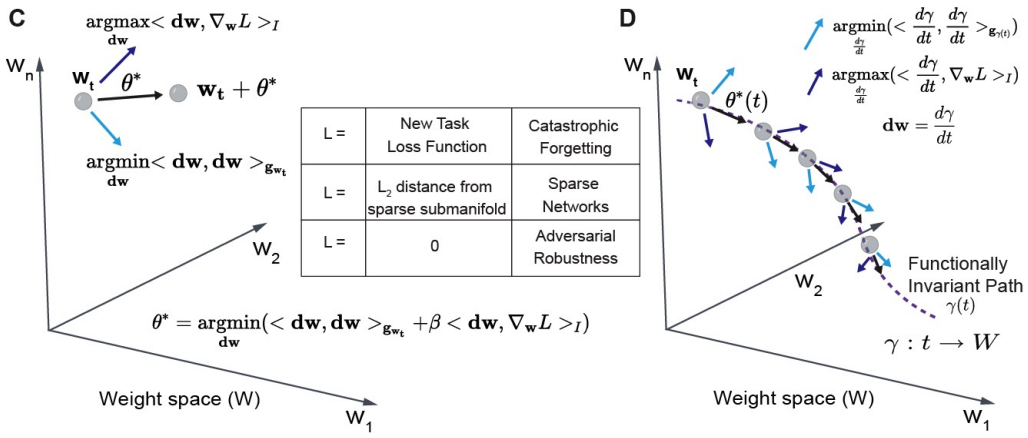


Figure 2.4: **Geometric Framework for constructing functionally invariant paths (FIP) in weight space.** The FIP algorithm identifies weight perturbations, θ^* that minimize distance moved in output space while maximizing alignment with gradient of a secondary objective function ($\nabla_{\mathbf{w}} L$). The light-blue arrow indicates ϵ-norm weight perturbation that minimizes distance moved in output space, dark-blue arrow is ϵ-norm weight perturbation that maximizes alignment with gradient of objective function, $L(\mathbf{x}, \mathbf{w})$. The secondary objective function $L(\mathbf{x}, \mathbf{w})$ is varied to solve distinct machine learning challenges. (D) The path algorithm defines functionally invariant paths, $\gamma(t)$, through iterative identification of ϵ-norm perturbations ($\theta^*(t)$) in the weight space.

where $\mathbf{J_{w_t}}$ is the Jacobian of $f(\mathbf{x}, \mathbf{w_t})$ for a fixed $\mathbf{x}$, $J_{ij} = \frac{\partial f_i}{\partial w_j}$, evaluated at $\mathbf{w_t}$.

Thus, the total change in network output for a given weight perturbation $\mathbf{dw}$ is

$$|f(\mathbf{x}, \mathbf{w_t} + \mathbf{dw}) - f(\mathbf{x}, \mathbf{w_t})|^2 = \mathbf{dw}^T \left(\mathbf{J_{w_t}}(\mathbf{x})^T \mathbf{J_{w_t}}(\mathbf{x})\right) \mathbf{dw} \tag{2.2}$$

$$|\langle \mathbf{dw}, \mathbf{dw} \rangle_{\mathbf{g_{w_t}}}|^2 = \mathbf{dw}^T \mathbf{g_{w_t}}(\mathbf{x}) \mathbf{dw}$$

where $\mathbf{g_{w_t}}(\mathbf{x}) = \mathbf{J_{w_t}}(\mathbf{x})^T \mathbf{J_{w_t}}(\mathbf{x})$ is the metric tensor evaluated at the point $\mathbf{w_t} \in W$ for a single data point, $\mathbf{x}$. The metric tensor is an $n \times n$ symmetric matrix that allows us to compute the change in network output for a perturbation along any direction in weight space as $\langle \mathbf{dw}, \mathbf{dw} \rangle_{\mathbf{g_{w_t}(x)}}$. The metric also allows us to compute the infinitesimal change in network output while moving along a path $\gamma(\mathbf{t})$ in weight space as the tangent vector $\psi(t) = \frac{\mathrm{d}\gamma(t)}{\mathrm{d}t}$.

At every point in weight space, the metric allows us to discover directions **dw** of movement that have large or small impact on the output of a network. As we move along a path, $\gamma(t) \subset W$, in weight space, we sample a series of neural networks over time, t. Using the metric we can define a notion of 'output velocity,' $\mathbf{v} = \frac{\mathrm{d}f(\mathbf{x},\gamma(t))}{\mathrm{dt}}$, that quantifies the distance a network moves in output space for each local movement along the weight space path $\gamma(t)$. We seek to identify 'Functionally invariant paths (FIPs)' in weight space along which the output velocity is minimized for a fixed magnitude change in weight. To do so, we solve the following optimization problem

$$\begin{aligned} \psi^*(t) = \underset{\frac{d\gamma}{dt}}{\operatorname{argmin}} & \langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_{\mathbf{g}_{\gamma(\mathbf{t})}} \\ \text{with } & \langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_I = \epsilon \end{aligned} \tag{2.3}$$

where we attempt to find a direction to perturb the network, such that it is ϵ units away in the weight space (in the euclidean sense) ($\langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_I = \epsilon$) while minimizing the distance moved in the networks' output space, given by $\langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_{\mathbf{g}_{\gamma(\mathbf{t})}}$. Here, I is an identity matrix, with the inner product $\langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_I$ capturing the euclidean distance in the weight space [Weisstein, 2014]. The optimization problem is a quadratic program at each point in weight space. The metric $\mathbf{g}$ is a matrix that takes on a specific value at each point in weight space, and we aim to identify vectors $\psi^*(t) = \frac{d\gamma(t)}{dt}$, that minimize the change in functional output of the network.

We will often amend the optimization problem with a second objective function $L(\mathbf{x}, \mathbf{w})$. We can enumerate paths that minimize the functional velocity in the output space while moving along the gradient of the second objective ($\nabla_{\mathbf{w}} L$). We define a path-finding algorithm that captures the trade off between these two terms

$$\theta^*(t) = \underset{\frac{d\gamma}{dt}}{\operatorname{argmin}}(\langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_{\mathbf{g}_{\gamma(t)}} + \beta \langle \frac{d\gamma}{dt}, \nabla_{\mathbf{w}} L \rangle_I)$$
$$\text{with } \langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_I = \epsilon \tag{2.4}$$

where now the first term, $\langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_{\mathbf{g}_{\gamma(t)}}$, identifies functionally invariant directions while the second term, $\langle \frac{d\gamma}{dt}, \nabla_{\mathbf{w}} L \rangle_I$, biases the direction of motion along the gradient of a second objective and β weighs the relative contribution of the two terms. When $L = 0$, the algorithm merely constructs paths in weight space that are approximately isofunctional ($\theta^*(t) = \psi^*(t)$), i.e. the path is generated by steps in the weight space comprising of networks with different weight configurations while preserving the input-output map. $L(\mathbf{x}, \mathbf{w})$ can also represent the loss function of a second task, for example a second input classification problem. In this case, we identify vectors that simultaneously maintain performance on an existing task (via term 1) while also improving performance on a second task by moving along the negative gradient of the second task loss function, $\nabla_{\mathbf{w}} L$. We think of constructing FIPs with different objective functions ($L(\mathbf{x}, \mathbf{w})$) similar to applying different "operations" to neural networks that identify sub-manifolds in the weight space of the network that accomplish distinct tasks of interest.

To approximate the solution to Eq-2.4, in large neural networks, we developed a numerical strategy that samples points in an ϵ ball around a given weight configuration, and then performs gradient descent to identify vectors $\theta^*(t)$. In the appendix, we extend the metric formulation to cases where we consider a set of N training data points, $\mathbf{X}$, and view $\mathbf{g}$ as the average of metrics derived from individual training examples. $\mathbf{g_w} = \mathbf{g_w}(\mathbf{X}) = \Sigma_{i=1}^{N} \mathbf{g_w}(\mathbf{x_i})/N$. The metric, $\mathbf{g}$, provides a local measure of *output distance* on the Riemannian manifold $(W, \mathbf{g_w})$. At each point in weight space, the metric defines the length, $\langle \mathbf{dw}, \mathbf{dw} \rangle_{\mathbf{g_w}}$, of a local perturbation by its impact on the

functional output of the network (Fig. 2.2).

Metric tensor eigenspectra reveals Sloppy/stiff directions

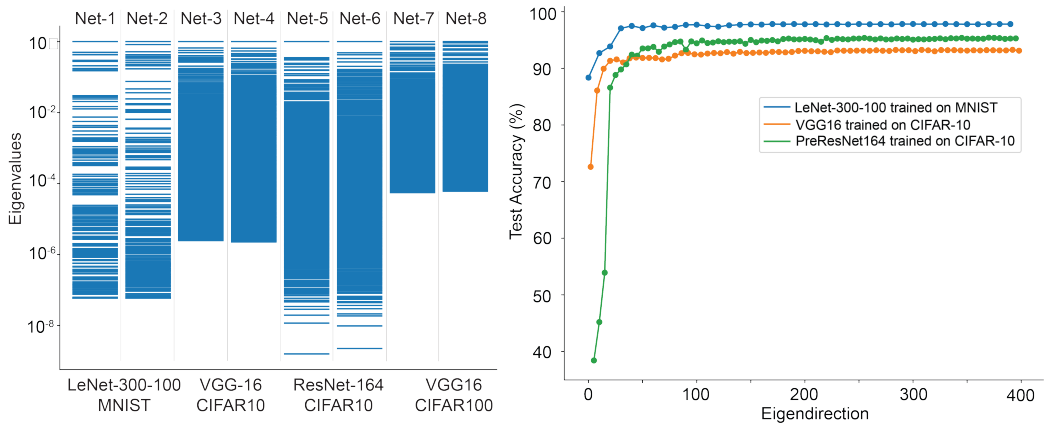


Figure 2.5: **Normalized Eigenvalues of the metric tensor (g_{w_t}) for various trained neural network architectures.** (Left) Eigenvalues are denoted by blue lines. The eigenspectra of two network instances from 4 different network architectures are presented here. The network architectures and the training dataset are mentioned along the x-axis. (Right) Test accuracy of networks perturbed along first 400 eigendirections.

We use our geometric framework to construct a metric tensor for a range of neural network architectures. The metric tensor is a device that provides information of how infinitesimal movements in the weight space changes the networks' functional mapping between a set of inputs (images or text) to outputs.

The eigenspectra of the metric tensor provides us with a glimpse of the local geometry of the functional manifold near a trained neural network. The eigendirections with high eigenvalues (stiff directions) correspond to network weight perturbations that majorly impact the functional performance (or input – output mapping) of the trained network. While eigendirections with low eigenvalues (sloppy directions) correspond to weight perturbations that cause almost no change to the functional performance. In Fig. 2.5(left), we find that ∼95% of the eigendirections of the metric tensor evaluated for a trained neural network are sloppy. We also observe that this finding is consistent across multiple network architectures (multilayer perceptron and

CNNs). For smaller networks, like LeNet-300-100 trained on MNIST, we construct the metric tensor over all the weights in the network (266,610 weights), while the larger networks (like, VGG16, ResNet-164) have been constructed using a subsampled set of weights (wherein 250,000 weights are sampled randomly across all layers of the network).

In Fig. 2.5(right), we find that networks perturbed along the stiff directions (first 10 eigendirections) incur a significant reduction in their test accuracy (from 98% to 88% in Lenet trained on MNIST, from 93% to 72% in VGG16 trained on CIFAR10). On the other hand, perturbing networks along their sloppy eigendirections (eigendirections: 20 and above) shows no profound drop in their test accuracy, remaining constant at 98%, 93% on Lenet and VGG16, respectively.

Section 2.3

Functionally invariant paths (FIP) alleviate catastrophic forgetting

We first apply the FIP framework to perform continual learning tasks without catastrophic forgetting on a series of image classification tasks with neural networks of increasing size. In catastrophic forgetting problems, we aim to modulate the weights of an existing neural network to achieve high performance on additional image classification tasks without loss of performance on previously learned tasks. A series of algorithms including the elastic weight consolidation (EWC) [Kirkpatrick et al., 2017], Gradient Episodic memory (GEM) [Lopez-Paz and Ranzato, 2017], Optimal Relevance Mapping (ORM)[Kaushik et al., 2021] have been developed to address catastrophic forgetting.

To circumvent catastrophic forgetting while learning sequential tasks, we train a neural network on a base task and modulate the weights in the network to accommodate additional tasks by solving the optimization problem in Eq-2.4, setting $L(\mathbf{x}, \mathbf{w})$

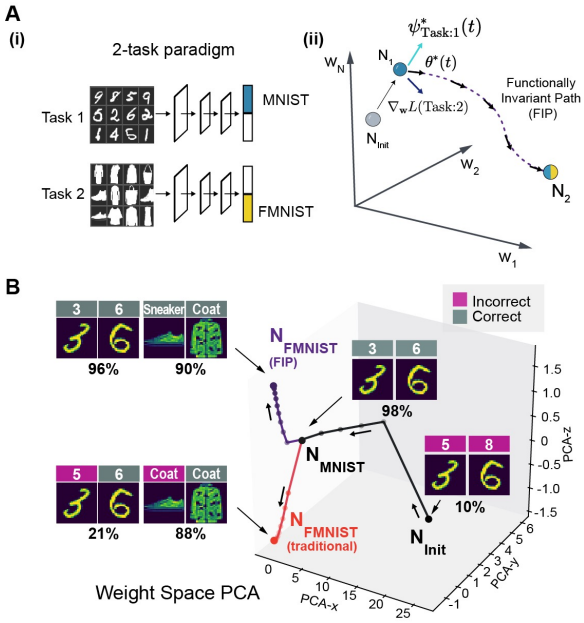


Figure 2.6: **Networks learn sequential tasks without catastrophic forgetting by traversing FIPs.** (A)(i) Training neural networks on a 2 task paradigm, with Task-1 being 10-digit recognition from MNIST and Task-2 being 10-item recognition from Fashion-MNIST. (ii) Schematic to construct FIPs in weight space to train networks on two tasks sequentially. (B) 3D lineplot where dots are weight configurations of 5-layered convolutional neural networks (CNNs) in PCA space. Training on two tasks sequentially via conventional approach takes the black followed by red path to reach N-FMNIST(traditional), while the path-finding algorithm takes the black followed by purple path to reach N-FMNIST(FIP). Images of digits-3,6 are from MNIST and sneaker, coat images are from Fashion-MNIST. Text labels above the image are networks' predictions and numbers below are the networks' test accuracy on MNIST and Fashion-MNIST.

as the classification loss function specified by the additional task while $\langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_{\mathbf{g}_{\text{Task1}}}$ measures distance moved in the networks' output using the metric from the initial task (in Eq-2.4 A(ii)). To accommodate the additional tasks, we append output nodes to the base network and solve the optimization problem for a fixed value of β by simultaneously minimizing the distance moved in the networks' output space (light-blue arrow in Fig. 2.6A) corresponding to the first task while maximizing alignment with the gradient of $L(\mathbf{x}, \mathbf{w})$ encoding the classification loss from the second task. In this manner, we construct a Functionally invariant path (FIP) (purple dotted line in Fig. 2.6A) in weight space generating a network that performs both Task 1 and Task 2.

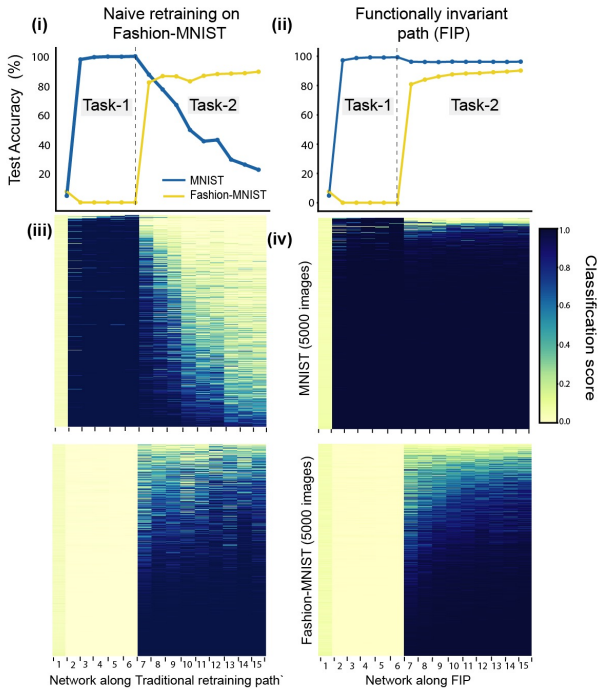


Figure 2.7: **Sequential learning of two image recognition tasks.** Test accuracy of networks learning two tasks sequentially by naive retraining on Fashion-MNIST (i) and traversing FIP (ii). Heatmaps capture classification score on 10k test images (5k images from each task) for networks obtained through FIP (iii) and traditional retraining strategy (iv).

To demonstrate performance of the FIP, we applied the framework to perform a canonical two-task sequential learning paradigm where a convolutional neural network (CNN) trained to classify handwritten digits from MNIST (Task 1) is modulated to also recognize ten classes of fashion apparel from the Fashion-MNIST dataset (FMNIST) (Task 2) (Fig. 2.6A(i)). For this sequential learning task, we use a CNN with 5 layers: 2 convolutional layers, with 32 and 64 convolutional filters each, and 3 fully connected layers - with 600, 120 and 20 nodes each. The network has a total of 1.4 million weights [LeCun, Haffner, et al., 1999; Krizhevsky, Sutskever, and Hinton, 2012]. The MNIST data set is a canonical data set representing $60,000$ examples of human hand written digits [LeCun, Bottou, et al., 1998] and the Fashion-MNIST data contains $60,000$ images of fashion items[Xiao, Rasul, and Vollgraf, 2017].

We initially trained the CNN to perform MNIST digit classification with $\sim 98\%$

accuracy using gradient descent based weight optimization (Fig. 2.6B, Fig. 2.7). If we simply re-train the MNIST classification network to classify Fashion-MNIST garments through gradient descent on FMNIST dataset, the network rapidly loses accuracy on the MNIST task as accuracy increases on FMNIST (Fig. 2.7). In contrast to gradient descent retraining, our FIP algorithm discovers a curved path of networks that simultaneously retain performance on the first task (MNIST), maintaining 98% to 96% test accuracy while reaching 89% performance on the second task (Fashion-MNIST) (Fig. 2.7(ii)). The networks along the curved FIP path retain their ability to recognize images of digits '3' and '6,' while also classifying images of fashion-apparel 'Sneaker' and 'Coat' from Fashion-MNIST (Fig. 2.6B).

Network's weight changes along FIP during sequential task learning

So far, we have shown that traversing an FIP in weight space enables a 5-layered CNN (2 convolutional layers and 3 fully connected layers) to continually learn two sequential tasks (MNIST and Fashion-MNIST).

As the FIP discovers networks that simultaneously retain the previous task information while learning the new task, we are interested in understanding how the networks' weights change as we traverse the FIP. Most continual learning methods that use regularization methods [Kirkpatrick et al., 2017] force the networks' weights to be similar to the network trained on task-1 while learning a new task (task-2).

However, we observe that traversing the FIP for continual learning applications results in large changes to network weights (Fig. 2.8). For instance, we find that weights across all layers (L1 to L5), barring the batchnorm layers, experience an average weight update of 100% from their original weights, while the weights in the batchnorm layers are updated by 25%. Additionally, we observe that the variance of the weight update decreases from layer-1 to layer-3 and increases from layer 3 to layer 5, suggesting that most weights in the middle layers (layer 2 to layer 4) of the CNN

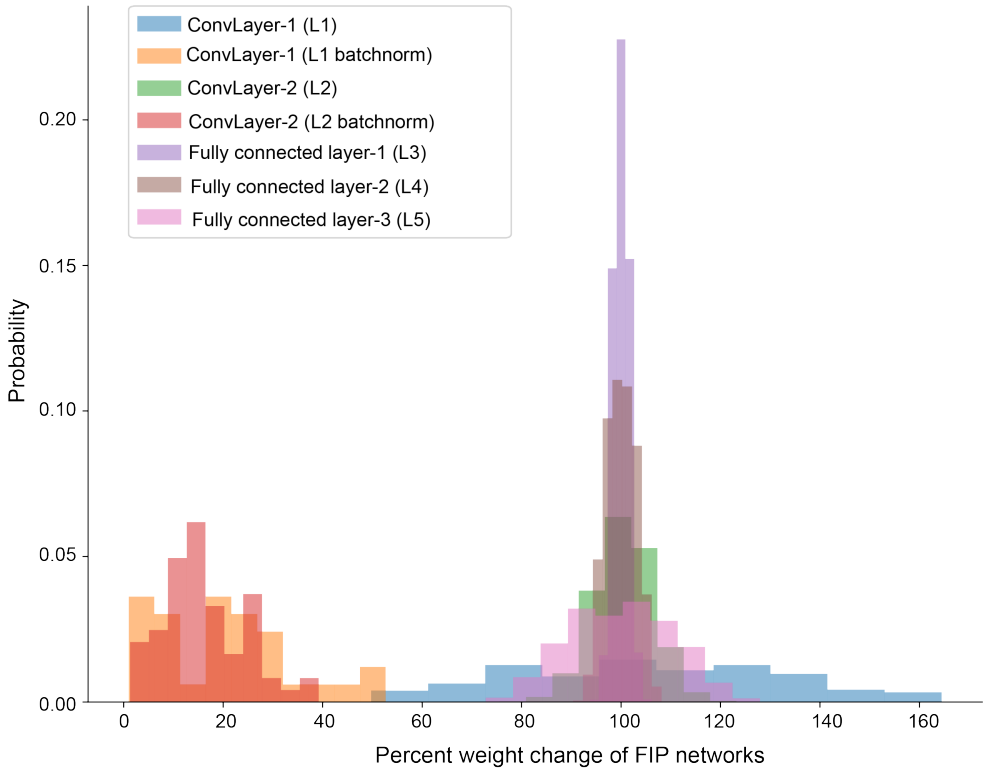


Figure 2.8: **Weight changes to networks along FIP during sequential learning.** Density histogram captures percentage weight change incurred by weights across all layers of a 5 layered CNN, while traversing an FIP constructed for sequential learning of MNIST and Fashion-MNIST.

encode features of the new task, while the weights in the first and last layer retain information from the previous task.

2.3.1. FIPs alleviate CF in a 5-task paradigm

To demonstrate sequential task learning, we applied our framework to a Multi-layer perceptron (MLP) with 2 hidden layers of 400 nodes each and 10 output classes, by subjecting them to 5 sequential tasks derived from the MNIST dataset [Van de Ven and Tolias, 2019]. As shown in Fig. 2.9A(i), each task is generated from different subsets of image-classes from the MNIST dataset. Task-1 comprises of MNIST digits 0's and 1's, Task-2 comprises of MNIST digits 2's and 3's and so on.

To circumvent catastrophic forgetting in the 5 sequential task paradigm, we solve the optimization problem in Eq-4 by setting L as the loss function specified by the new

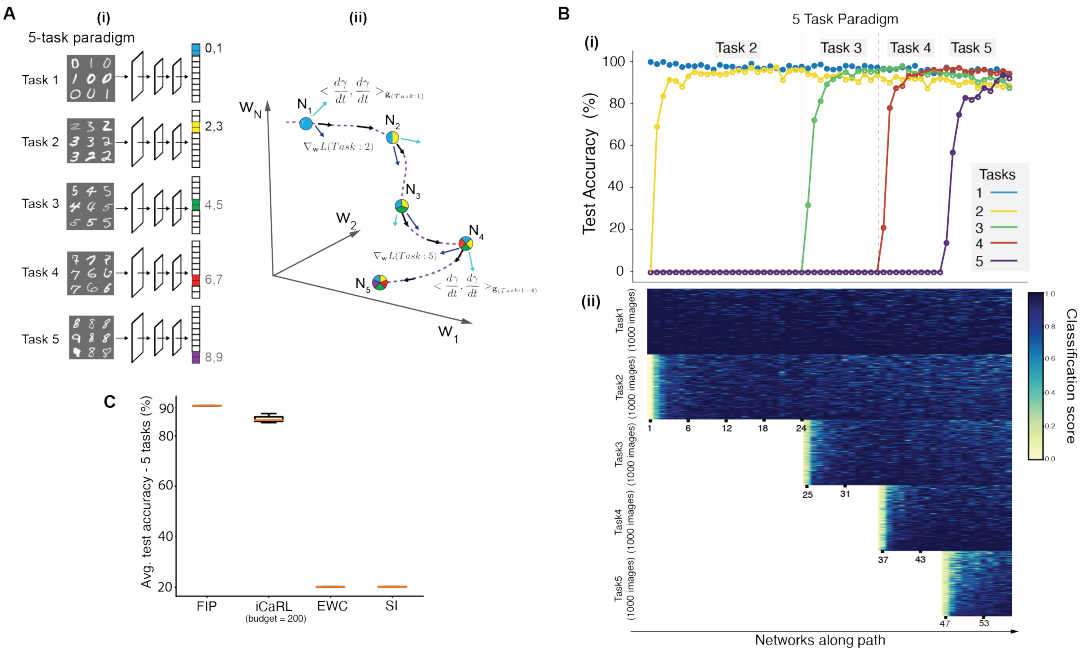


Figure 2.9: **FIP alleviates CF in 5-sequential task paradigm for splitMNIST**(A) (i) Neural network with 10 output classes are trained on 5 task paradigm, with every task containing a different subset of MNIST digits. (ii) Schematic to construct FIPs in weights space in order to train neural networks on 5 sequential tasks. (B) (i) Test accuracy of networks while traversing FIPs to learn 5 sequential tasks. The dashed lines indicate that the networks encounter a new task. (ii) Heatmap displays classification score for networks along FIP on 5k images, with 1k images sampled from every task. (C) FIP surpasses state-of-art methods in mitigating catastrophic forgetting in 5-task paradigm.

task (Task-i) represented by dark-blue arrows in Fig-2D(ii), while $< \frac{d\gamma}{dt}, \frac{d\gamma}{dt} >_{\mathbf{g}_{Task-1:i-1}}$ is chosen to be the distance moved in output space for a small number of inputs sampled from all previous tasks (from Task-1 to Task:i-1) represented by light-blue arrows in Fig. 2.9A(ii).

We train the MLP on the first task (recognizing images of digits 0s and 1s) by gradient descent and get a network that performs at an accuracy of 98% on the first task. In Fig. 2.9A(ii), the first point on the x-axis corresponds to the network trained on the first task alone, and the blue line records test accuracy on the first task.

Having trained the MLP on the first task, we train the network on subsequent tasks (e.g. recognizing images of digits 2s and 3s, recognizing digits 4s and 5s) by

constructing FIPs in the weights space and obtain networks that simultaneously retain performance on all previous tasks while learning a new task. In Fig. 2.9B(i), the region corresponding to Task-2 (first 25 points along the x-axis) captures the test accuracy of networks on the first two tasks (blue, yellow respectively) while traversing the FIP in weights space beginning from N_1 (MLP trained on Task-1). The networks along the path retain their performance on Task-1 (blue line) at 98% while increasing their accuracy on Task-2 (yellow line) to 97%. The region corresponding to Task-3 (subsequent 11 points, 26 to 36, on the x-axis) comprise of networks that retain their performance on both Task-1 and Task-2 (at 97% and 95% respectively) while increasing their accuracy on Task-3 (green line) to 97%. Finally, on introducing Task-5, corresponding to points 47-58 on the x-axis in region labeled Task-5, we uncover networks that perform at 96.74%, 88.64 %, 89.33 %, 94.36 % and 91.23 % on Tasks 1 to 5 respectively, with an average performance of 92.01% on the MNIST dataset having shown 2 classes at a time.

The networks discovered along the FIP mitigate catastrophic forgetting while being trained on multiple tasks in sequence as they retain their classification score on previous tasks while increasing their score on new tasks. In Fig. 2.9B(ii), the heatmap captures the classification score by feeding 1000 images from each task to the network (or 5000 images from 5 tasks). The first 24 networks (1-24) retain their classification score on Task-1, while increasing their classification score on Task-2. The subsequent 11 networks (25-36) retain their classification score on both Task-1 and Task-2 while increasing its score on Task-3. The last segment of the path (corresponding to network-47 to network-58) retain their score on Tasks 1 through 4, while increasing their classification score on Task-5. Having presented all 5 tasks to the network, our path-finding framework discovers networks that perform at $92.1 \pm 0.06\%$ accuracy on the 5 tasks, while the conventional method performs at $18 \pm 2\%$ [Van de Ven and Tolias, 2019].

Our FIP approach performs better than current state-of-art methods to alleviate catastrophic forgetting while learning sequential tasks, like Elastic Weight Consolidation (EWC) on the 5-task paradigm (FIP: 92.1±0.06%, EWC:20±0.08%) (Fig. 2.9C).

Section 2.4

FIPs enable iterative continuous learning

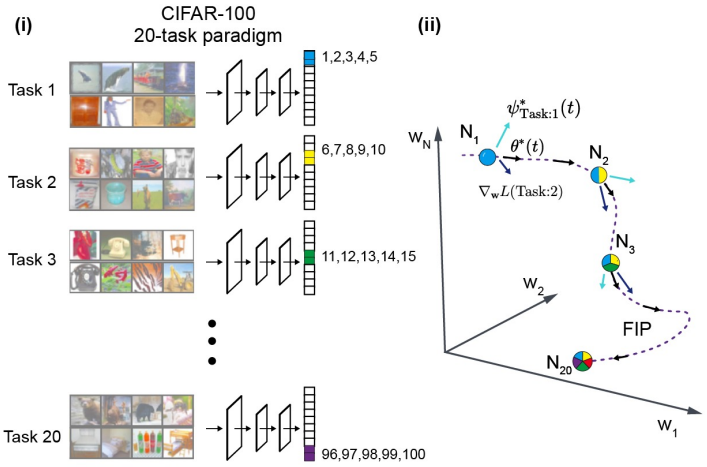


Figure 2.10: **Sequential training of networks on 20 image recognition tasks.** (i) Neural network with 100 output classes are trained on 20 task paradigm, with every task containing 5 non-overlapping classes of natural images sampled from CIFAR100 dataset. (ii) Schematic to construct FIPs in weight space to train neural networks on 20 sequential tasks.

Due to the generality of the framework, the FIP approach naturally scales to learn a longer series of tasks without catastrophic forgetting through iterative application of Eq-2.4. We applied the FIP to learn 20 sequential image recognition tasks using subsets of the CIFAR100 image database 2.10(i) [Ven, Siegelmann, and Tolias, 2020]. At each round, we modulate a large CNN, ResNet18, to recognize an additional five object categories in CIFAR100 (2.10(i)).

We first train ResNet18 network to obtain 78% accuracy on Task-1 by gradient descent using the Xavier initialization [Glorot and Bengio, 2010] protocol. To learn

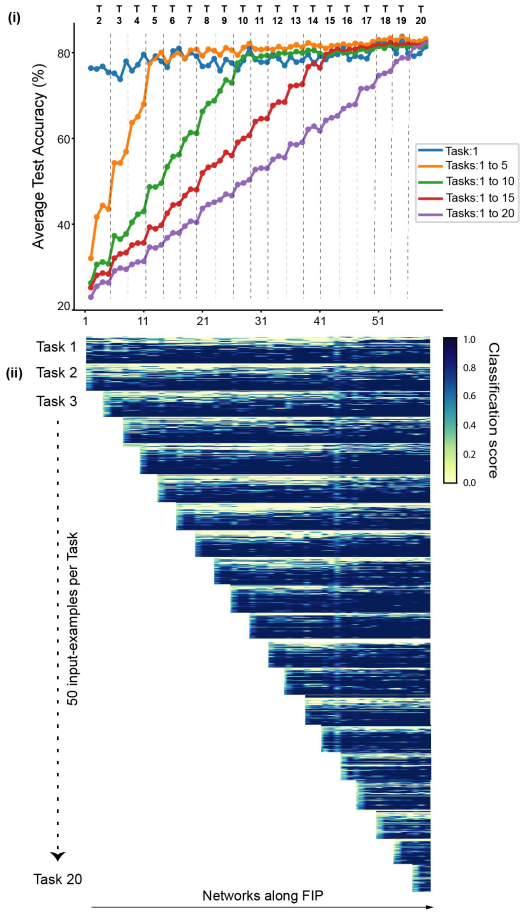


Figure 2.11: **FIP traversal of networks while being trained on 20 image recognition tasks.** (i) Average test accuracy of networks along FIP while learning 20 sequential tasks. The networks to the right of a dashed line encounter a new task (T-i), referring to the i'th task. (ii) Heatmap displays classification score for networks along FIP on 1k test images, with 50 images sampled from every task.

additional tasks without forgetting, we solve the optimization problem in Eq-2.4 by setting $L(\mathbf{x}, \mathbf{w})$ as the loss function specified by the incoming new task (Task-i), while $\langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_{\mathbf{g}_{\text{Task}-1:i-1}}$ is set to be the distance moved in output space for a small number of inputs sampled from all the previous tasks (from Task:1 to Task:i-1). In this way, we iteratively construct FIPs in the weight space and obtain networks that simultaneously retain performance on all previous tasks while learning a new task (Fig. 2.10).

The FIP strategy achieves high accuracy on incremental tasks broadly across the

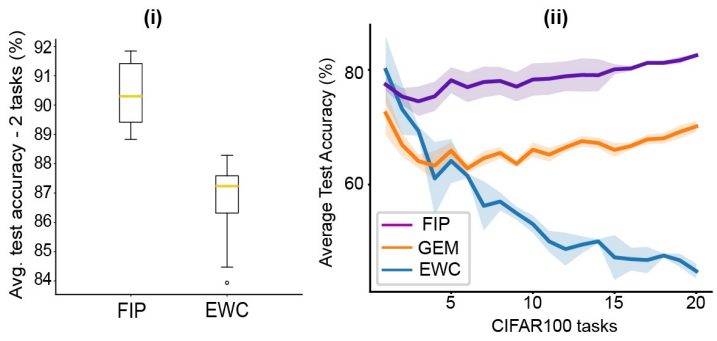


Figure 2.12: **FIP traversal alleviates catastrophic forgetting.** FIP surpasses state-of-art methods in mitigating catastrophic forgetting in 2-task paradigm (i) and 20-task CIFAR100 paradigm (ii). Error bars indicate standard deviation over 5 trials.

held-out testing set (Fig. 2.11) . Having presented all 20 tasks to the network, the FIP algorithm discovers networks that perform at 82.54±0.17% accuracy while naive re-training performs at 26.86±1.05%. The FIP approach outperforms other methods that have been introduced to mitigate catastrophic forgetting, specifically Elastic weight consolidation on the 2 task paradigm (Fig. 2.12(i)) (FIP: 91±1.1 %, EWC: 87±1.6%) and 20-task paradigm (FIP: 82.54±0.17%, EWC:44.9±0.01%) (Fig. 2.12(ii)) and Gradient episodic memory (GEM) with a memory budget of 500 memories from each task previously encountered [Kirkpatrick et al., 2017; Lopez-Paz and Ranzato, 2017]. While the FIP algorithm has conceptual similarities with EWC, the mathematical generality of the FIP approach allows the approach to scale to perform multiple iterative incremental learning tasks and to explicitly construct functionally invariant paths that span long distances in the weight space. EWC tends to find networks in the vicinity of a previously trained network by computing a local Fisher Information metric.

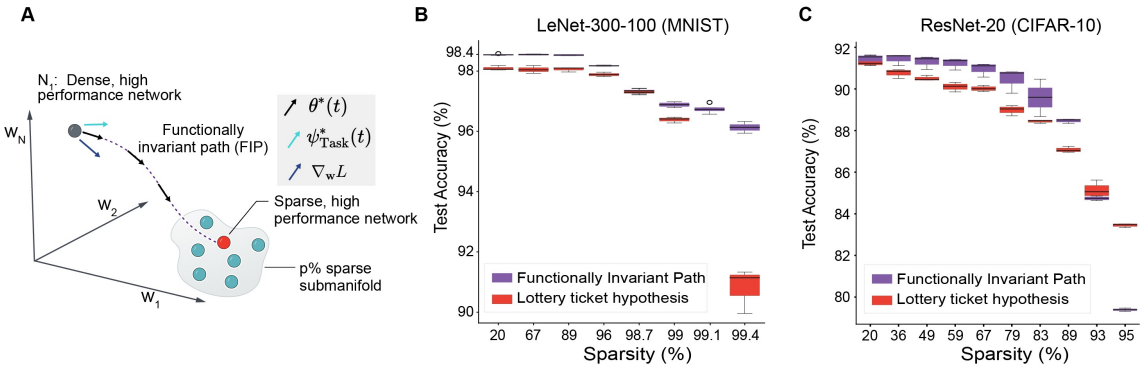


Figure 2.13: **Sparse networks discovered by traversing FIPs in the weight space.** (A) Schematic to construct FIP from N_1 to p% sparse submanifold. (B, C) Performance of sparse networks discovered by FIPs (purple) and Lottery ticket hypothesis (red) across a wide range of sparsities on MNIST (B) and CIFAR-10 (C).

Section 2.5

Network sparsification via path-connected network sets

The critical aspects of the FIP framework is that the framework generalizes beyond sequential task training to address a broad range of machine learning meta-problems by considering a more general set of secondary objective functions. In particular, we next apply the FIP framework to perform sparsification, reducing the number of non-zero weights, of neural networks, which is important for reducing the memory and computational footprint of a network [Blalock et al., 2020]. To sparsify neural networks, we solve Eq-2.4, the core FIP optimization problem, with a secondary loss function $L(w_t, w, p)$ that measures the euclidean distance between a network and it's p-sparse projection obtained by setting p% of the networks' weights to zero.

Using the framework, we discovered a series of sparsified LeNet-300-100 networks with sparsities ranging from 20% to 99.4% that exhibit a high performance on MNIST digit classification [LeCun, Bottou, et al., 1998]. LeNet-300-100 [LeCun, Haffner, et al., 1999] is a multilayer perceptron with two hidden layers consisting of 300 and

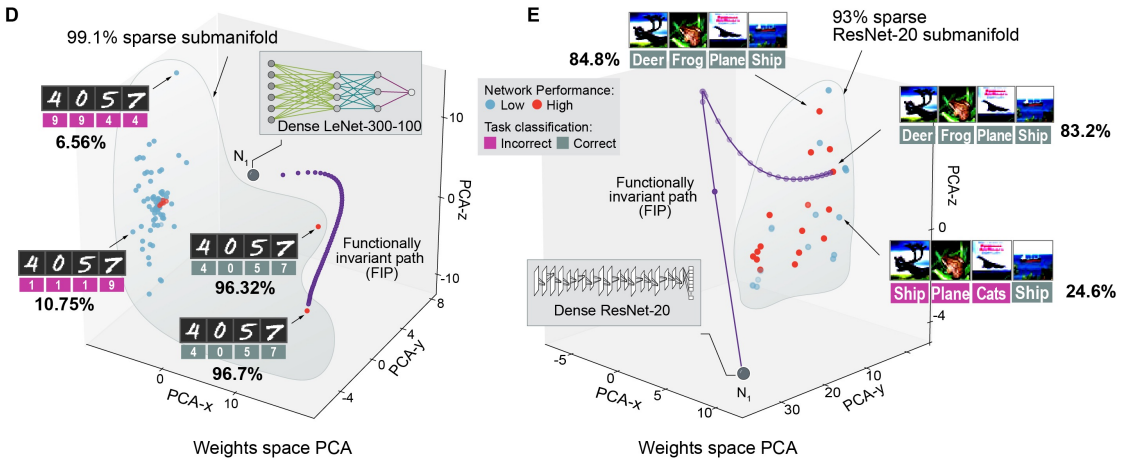


Figure 2.14: **Visualizing sparse networks discovered by FIPs in the weight space.**(D) Scatterplot where the dots are weight configurations of LeNet-300-100 networks in PCA space. The FIP (purple line) beginning from N_1 (grey dot) discovers high-performance LeNet's in the 99.1% sparse submanifold (red dots). Blue dots are random sparse networks in the 99.1% sparse submanifold. Digits-4,0,5,7 are from MNIST, text-labels below the image are network predictions and the number below is the networks' test accuracy on MNIST. (E) Scatterplot where the dots are weight configurations of ResNet-20 networks in PCA space. The FIP beginning at N_1 (grey dot) discovers high-performance ResNet-20 networks in the 93% sparse submanifold (red dots). Blue dots are random sparse networks in 93% sparse submanifold. Deer, frog, plane, ship images are from CIFAR-10, text-labels below the image are network predictions and the number adjacent is the networks' test accuracy on CIFAR10.

100 nodes each, and a total of 484000 non-zero weights. Although most networks randomly sampled from the 99.1% sparse submanifold in the weight space perform poorly on the MNIST task (with test accuracies ranging from 6 to 10%), the FIP algorithm finds a curved path in the weight space from densely connected LeNet (with test accuracy of 98% on MNIST) to networks in the 99.1% sparse submanifold that perform at test accuracies between 96.3% to 96.8% on the MNIST classification task (Fig. 2.14D). The FIP-discovered networks have diverse inter-layer connectivity structures (Fig. 2.15F) placing non-zero weights (black vertical bars) in distinct locations. While the FIP solutions vary in their local architectures, there are also patterns across networks. Specifically, across the space of sparse solutions 99.2±0.2% and 98.4±0.3 % of the weights between layers 1-2 and 2-3, respectively, are zeroed ,

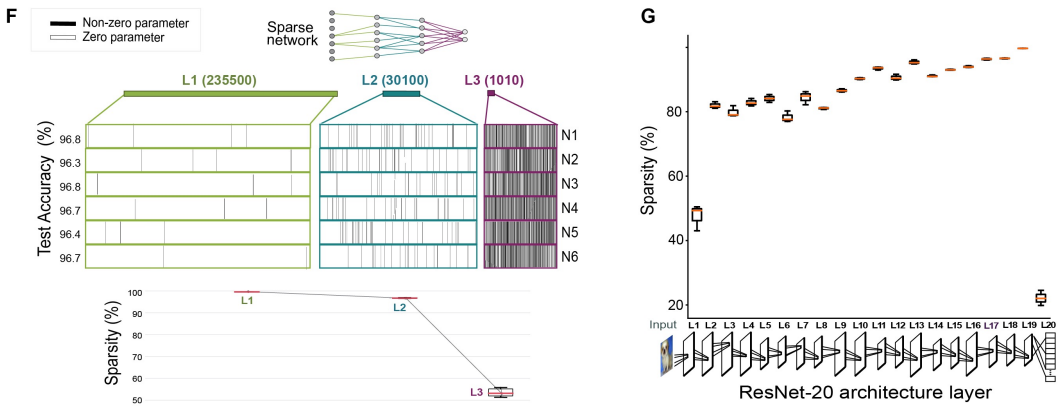


Figure 2.15: **Sparse network architectures discovered by traversing FIPs in the weight space.** (F) Sparse LeNet connectivity visualized by plotting vertical lines in color-coded rows to represent non-zero weights. (Below) Boxplot shows sparsity across LeNet's layers. (G) Boxplot shows the sparsity across ResNet-20's layers (over n=6 sparsified ResNet's). The cartoon below depicts the ResNet-20 architecture.

while only 52±4% of the weights between layers 3-4 are zeroed out. The differential sparsification across different layers indicates that connections between the first few layers may contain more redundancy than the later layers.

The sparse networks discovered by traversing FIPs have a higher task-performance than the ones discovered through prune-retrain cycles employed within frameworks like the lottery ticket hypothesis (LTH) [Frankle and Carbin, 2018](Fig. 2.13B). Across a wide range of sparsities (from 20% to 99.4%), FIP discovers networks that are comparable in test accuracy to those obtained by LTH. The FIP also discovers extremely sparse networks with high-performance. The FIP method finds a 99.4% sparse network performing at an accuracy of 96±0.6%, while the LTH strategy finds a 99.4% sparse network performing at 91±3% on the MNIST dataset.

The FIP algorithm scales to discover sparse counterparts of CNN networks with multiple layers and skip connections. We applied the FIP framework to sparsify the ResNet-20 architecture [He et al., 2016] with twenty convolutional layers, trained to recognize images of automobiles, animals and man-made structures from the CIFAR-10 dataset. Although most networks sampled from the 93% sparse submanifold of

ResNet20 networks perform at accuracies between 18 to 30% on CIFAR10, an FIP constructed from a dense, trained ResNet20 network to the 93% sparse submanifold identifies high performance sparse ResNet20 networks with accuracies between 82 to 84.7% on CIFAR10.

Like the LeNet example, the inter-layer connectivity of sparse ResNet-20 networks are distinct locally, but have globally conserved patterns. For instance, the weights between layers 2 to 19 have an average inter-layer sparsity of 85% while having a maximum sparsity of 99.2%, present between layers 18-19 (penultimate layer), across all sparsified ResNet20 networks (in Fig. 2.15G) we find that while the weights between layer 1-2 (first 2 layers) and the layers 19-20 (last 2 layers) are least sparsified (41% and 24% sparse respectively). The differential sparsification across different layers points to the fact that the redundancy in ResNet-20 architectures is encoded primarily between layers 3 and 18. The FIPs discover high performance sparse ResNet20 networks on a wide range of sparse submanifolds (from 20% to 95%) that are at par with the prune-retrain techniques like the lottery ticket hypothesis (LTH) [Frankle and Carbin, 2018].

Section 2.6

Path-connected sets of networks confer robustness against adversarial attack

The path connected sets of networks generated by the FIP can also be applied to perform inference and increase the robustness of inference tasks to data perturbation. Although deep networks have achieved remarkable performance on image-recognition tasks, human-imperceptible additive perturbations, known as adversarial attacks, can be applied to an input image and induce catastrophic errors in deep neural networks (Fig. 2.17). The FIP algorithm provides an efficient strategy to increase network robustness and mitigate adversarial failure by generating path-connected sets of net-

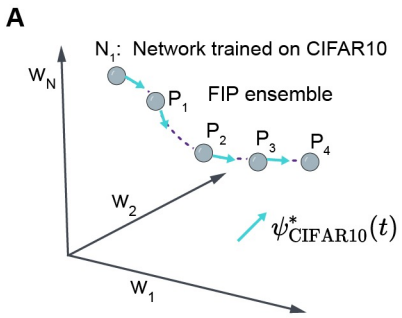


Figure 2.16: **FIPs in weight space generate ensembles of networks that confer adversarial robustness** (A) Schematic to generate FIP ensemble (P_1,...,P_4) by sampling networks along FIP (purple dotted line) beginning at network-N_1. FIP is constructed by identifying a series of weight perturbations that minimize the distance moved in networks' output space.

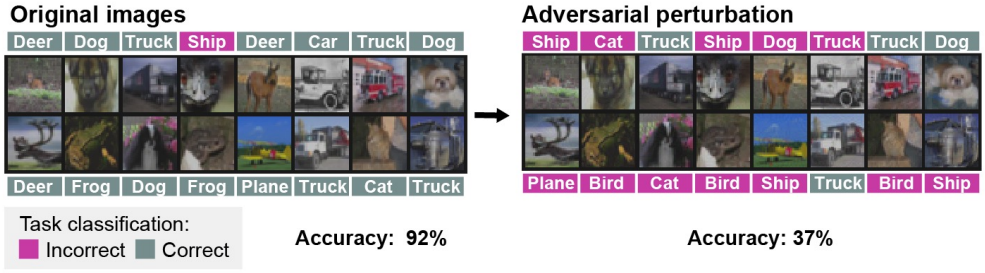


Figure 2.17: **Adversarial images.** Original CIFAR10 images (left) and Adversarial CIFAR-10 images (right) are shown. The text-labels (left, right) above the images are predictions made by a network trained on CIFAR-10. Trained networks' accuracy on the original and adversarial images are shown below.

works with diverse weights. We, then, apply the path connected network sets to perform robust image classification by averaging their output.

To demonstrate that the FIP algorithm can mitigate adversarial attacks, we trained a 16 layered CNN, VGG16 with 130 million parameters to classify CIFAR10 images with 92% test accuracy. We, then, generated adversarial test images using the projected gradient descent attack strategy. On adversarial test images, the performance of VGG16 dropped to 37% (Fig. 2.18C, Fig. 2.19D). To mitigate the adversarial performance loss, we applied the FIP algorithm to generate an ensemble of functionally invariant networks by setting $L = 0$ in the optimization problem in

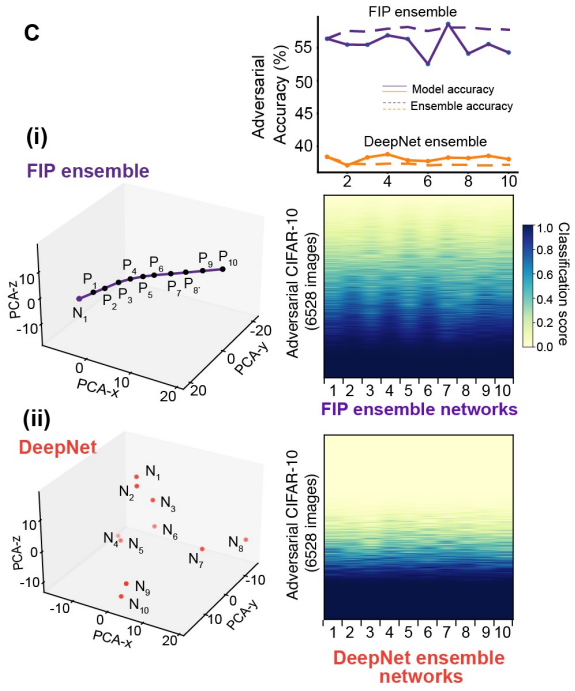


Figure 2.18: **Adversarial robustness conferred by FIP-discovered ensembles of networks** (C) (Top) Line-plot (solid) shows the individual network performance on adversarial inputs, (dashed) shows the joint ensemble accuracy on adversarial inputs, for FIP ensemble (purple) and DeepNet ensemble (orange). (i,ii-Left) FIP ensemble in purple (P_1,P_2,...,P_{10}) and DeepNet ensemble in orange (N_1,N_2,...,N_{10}) are visualized on weight space PCA. (i,ii-Right) Heatmaps depict classification score of networks in FIP ensemble and DeepNet ensemble on 6528 adversarial CIFAR-10 examples.

Eq-2.4 and setting $\langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_{\mathbf{g}_{\text{CIFAR10}}}$ to be the distance moved in the networks' output space for CIFAR-10 images.

Having constructed the FIP in the weight space, we introduce a selection-criteria to sample diverse networks along the FIP to construct the FIP ensemble. As we want the FIP ensemble to be robust to adversarial input perturbation, we generate random perturbations in the image space (within an ϵ-l_∞ ball) and compute the distance moved in the networks' output space for a small perturbation in the image-space. We record the distance moved in the networks' output space (across all networks in the constructed FIP) and plot a distribution of the distance moved in the output space for a small perturbation in the image-input space. In Fig. 2.20A, we find that some

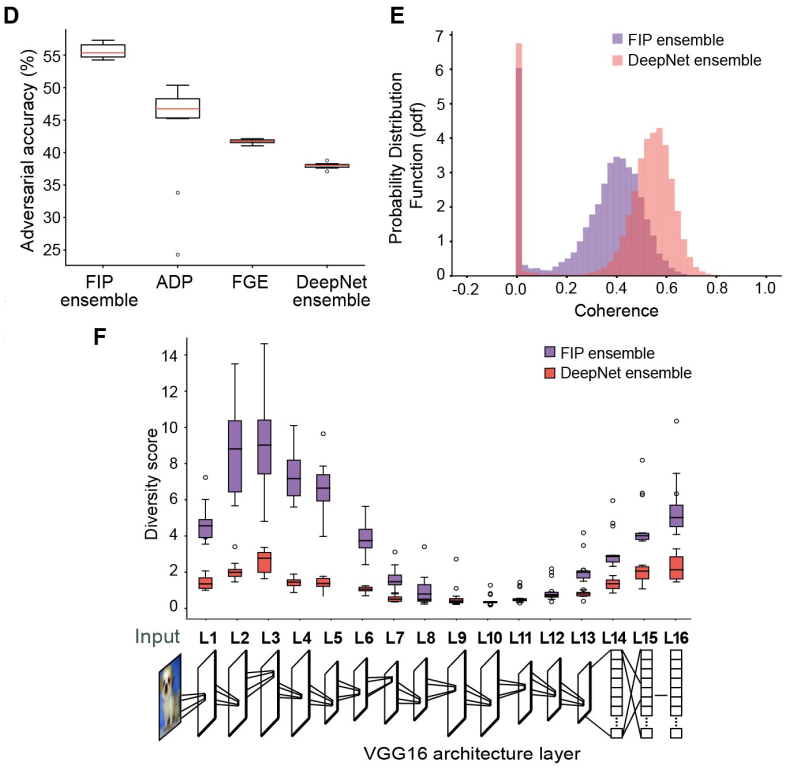


Figure 2.19: **Comparing adversarial robustness across different benchmarks.** (D) Boxplot compares adversarial accuracy (over 10k adversarial examples) across different ensembling techniques (n=3 trials). (E) Histogram of coherence values for FIP (purple) and DeepNet ensemble (orange). (F) Boxplot shows the ensemble diversity score across VGG16 layers over n=1000 CIFAR10 image inputs. The cartoon below depicts the VGG16 network architecture.

networks along the FIP exhibit smaller perturbation in the output space and have a narrower distribution across 10k perturbed training inputs, while others exhibit larger perturbation in the output space. We choose networks that exhibit a smaller perturbation in the output space for constructing the FIP ensemble. We use the FIP ensemble to classify images by summing the 'softmaxed' outputs of the ensemble.

Using an ensemble of ten networks sampled along an FIP, we achieve accuracy of 55.61 $\pm$ 1.1 % surpassing the performance of the DeepNet ensemble performance (composed of 10 independently trained deep networks) by 20.62% (Fig. 2.18C). The FIP ensemble's adversarial performance also surpasses other state of the art ensemble approaches including Adaptive Diversity promoting (ADP, 43.84 $\pm$ 7.8%) ensemble

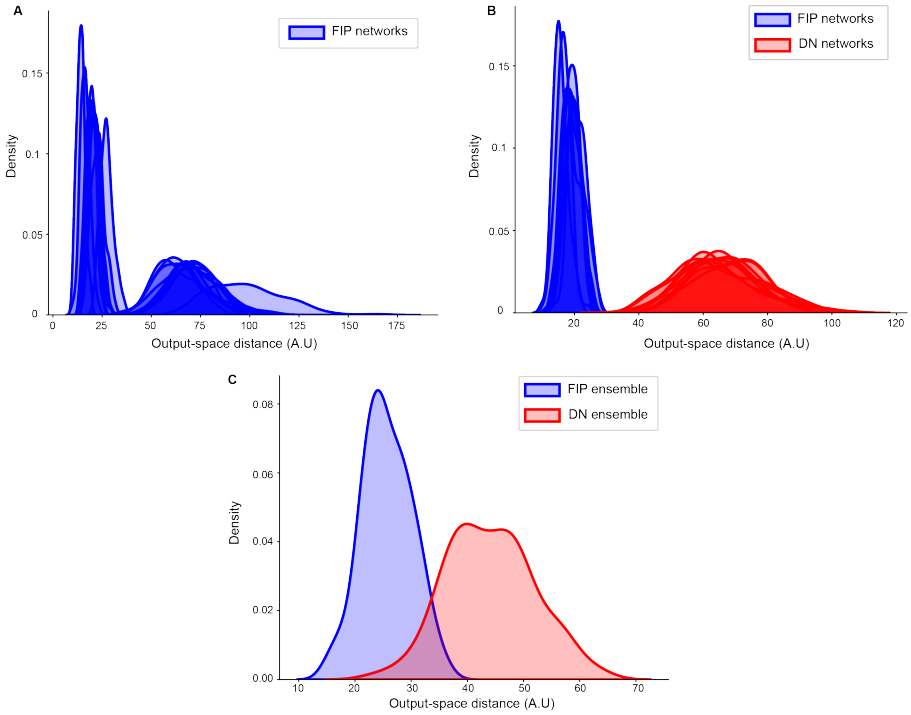


Figure 2.20: **FIP ensemble construction** (A) Distribution of distances moved in networks' output space over 10k image-inputs perturbed within ϵ-l_∞ ball, for individual networks along the FIP. (B) Comparing distribution of distances moved in networks' output space over 10k image-inputs perturbed within ϵ-l_∞ ball, for individual networks in the FIP ensemble (blue) and networks in the DN ensemble (red). (C) Distribution of distances moved in networks' output space over 10k image-inputs perturbed within ϵ-l_∞ ball, for the entire FIP ensemble (blue) and the entire DN ensemble (red).

and the Fast Geometric Ensembling (FGE, 41.7 ± 0.34) method. The two factors contributing to the FIP ensemble's robustness are (i) high intra-ensemble weight diversity, calculated by the representation diversity score (Supplementary) and (ii) low coherence (Supplementary) with a trained surrogate network (used to generate adversarial images) (Fig. 2.19E,F). FIP Networks have a higher representation diversity score in their early processing layers, from Layer 1 to layer 6, when compared to the DeepNet ensemble, indicating that individual networks in the FIP ensemble extract different sets of local features from the adversarial image, preventing networks to rely on similar spurious correlations for image classification.

Generating a large diverse set of functionally similar networks enables us to

identify networks that exhibit smaller perturbation in the output space for an ϵ-perturbation in the input space, when compared to the networks in the DeepNet ensemble (composed of 10 independently trained deep networks), as seen in Fig. 2.20B,C. The blue histogram corresponds to the output distance distribution of networks in the FIP, while the red histogram corresponds to the output distance distributions of networks in the DeepNet ensemble.

Section 2.7

FIP endows flexibility to very large systems (100s of millions of parameters)

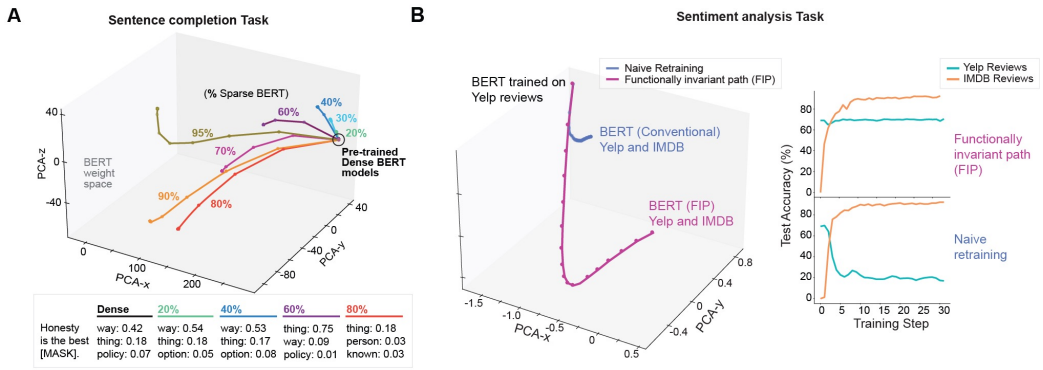


Figure 2.21: **FIPs in large transformer based language model (BERT) weight space.** (A, B(left)) 3D lineplot where dots are weight configurations of 12-layered BERT models with 12 attention heads per layer in PCA space. (A) FIP from pre-trained dense BERT (black circle) discovers high-performance sparse counterparts (labeled p% sparse BERT, where p $\in [20, 30, ...95]$). Table shows "Fill in the Mask" training sample and sparse BERT predictions of the masked word alongside the confidence score. (B, left) Yelp-BERT retrained on IMDB using FIP (purple) and naive retraining (blue). (B, right) BERT performance on Yelp and IMDB.

Conceptually, the most important aspect of the FIP framework is that it unifies a series of machine learning meta-tasks (continual learning, sparsification) into a single mathematical framework, ultimately endowing flexibility to the system. Mathematically, when we solve Equation 2.4 with a given secondary loss function, we move an

existing network $\mathbf{w(0)}$ along a path in weight space generating a new network $\mathbf{w(t)}$ where the parameter t increments the length of a path. Each additional loss function generates a new transformation of a base network, for example, generating a network adjusted to accommodate an additional data analysis problem or a secondary objective like sparsification. Network transformation maps can be iterated and applied to generate a family of neural networks optimized for distinct sub-tasks. The resulting path connected set of neural networks can then be queried for networks that achieve specific user goals or solve additional machine learning problems.

The problem of customization is particularly important for transformer networks that have recently emerged as state of the art architectures in natural language processing [Vaswani et al., 2017; Brown et al., 2020; Liu et al., 2019; Wolf et al., 2020]. We applied the iterative FIP framework to generate a large number of natural language processing networks customized for different sub-tasks and sparsification goals. Transformer networks incorporate layers of attention heads that provide contextual weight for positions in a sentence. Transformer networks are often trained on a generic language processing task like sentence completion where the network must infer missing or masked words in a sentence. Models are, then, fine tuned for specific problems including sentiment analysis, question answering, text generation, and general language understanding [Vaswani et al., 2017]. Transformer networks are large containing hundreds of millions of weights, and so model customization can be computationally intensive.

We applied the FIP framework to perform a series of model customization tasks through iterative application of FIP transformations on distinct goals. The BERT network has a total of twelve layers, or transformer blocks, with 12 self-attention heads in each layer and a total of 110 million parameters [Devlin et al., 2018]. Using the FIP framework, we applied two operations (Continual Learning (CL), Compression (Co)) sequentially to BERT models trained on a range of language tasks, by constructing

FIPs in the BERT weight space using different objective functions ($L(\mathbf{x}, \mathbf{w})$) while solving the optimization problem in Eq-2.4. We, first, demonstrated that the FIP framework allowed us to generate versions of the base BERT model that are customized for single-sub-tasks. For example, we sparsified BERT, generating networks with $20 - 80\%$ weights set to zero (Fig. 2.21A). Similarly, we applied the FIP to adjust the base BERT model, trained on sentence completion, to perform sentiment analysis on YELP and IMDB movie reviews.

FIP alleviates CF in large language models (BERT)

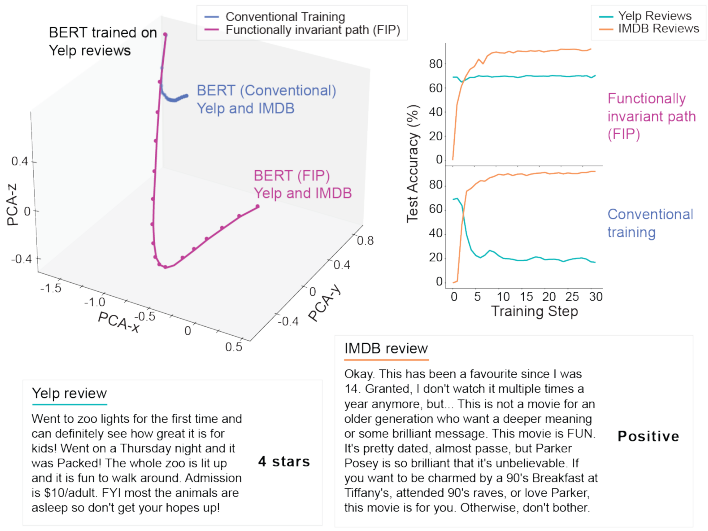


Figure 2.22: **FIPs in large transformer based language model (BERT) weight space.** (Left) 3D lineplot where dots are weight configurations of 12-layered BERT models with 12 attention heads per layer in PCA space. Training BERT (previously trained on Yelp) on IMDB reviews via conventional training (blue) and via FIP (purple). (Right) 2D lineplots show BERT performance on Yelp-reviews (blue) and IMDB-reviews (orange) for FIP (above) and conventional training (below). Example test sample from Yelp and IMDB dataset, with the prediction by BERT along FIP.

Training BERT to detect customer opinions of a product based on text reviews left on websites like Yelp or IMDB, results in catastrophic forgetting, especially while sequentially training on multiple user-review datasets (say, yelp-reviews followed by IMDB). The line plots in Fig 2.22 (right) demonstrate that conventional training

of BERT on IMDB reviews increases its performance on sentiment classification on IMDB (from 0% to 92%, orange) while abruptly forgetting sentiment analysis on Yelp-reviews (dropping from an accuracy of 69.9% to 17%, blue) within 30 training steps. We circumvent CF in BERT by solving the optimization problem in Eq-2.4 by setting $L(\mathbf{x}, \mathbf{w})$ as the loss function specified by the new task (IMDB sentiment-analysis) and $\langle \frac{d\gamma}{dt}, \frac{d\gamma}{dt} \rangle_{\mathbf{g}_{\text{Task}-1}}$ as the distance moved in output space for a small fraction of inputs (0.3%) sampled from the previous task (Yelp review-rating). The line plots in Fig 2.22A (right) demonstrate that traversing the FIP maintains BERT performance on Yelp-reviews (at 70%, blue) while increasing its accuracy on IMDB review classification (from 0% to 92%, orange). The FIP in BERT weight space (Fig. 2.22A (left), purple) is much longer than the route taken by conventional training (Fig. 2.22 (left), blue), enabling global exploration of the BERT weight-landscape to identify networks that simultaneously maintain performance on Yelp-reviews while learning the IMDB sentiment classification.

2.7.1. FIP used to apply 'task' arithmetic in sequence

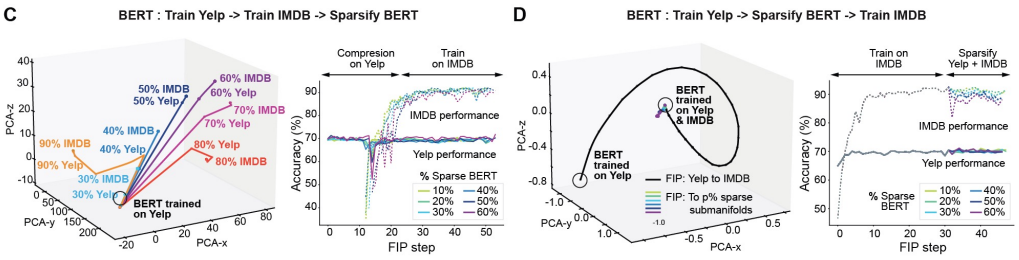


Figure 2.23: **Visualizing FIPs in BERT weight space** (C,D (left)) 3D lineplot where dots are weight configurations of 12-layered BERT models with 12 attention heads per layer in PCA space. (C, left) FIP initially identify high performance sparse BERTs (for sparsities ranging from 10% to 80%) followed by re-training on IMDB. (C, right) BERT accuracy on Yelp (solid) and IMDB (dashed) dataset along the FIP. (D, left) FIP initially retrains BERT on new task (IMDB) and then discovers a range of sparse BERTs. (D, right) BERT accuracy on Yelp (solid) and IMDB (dashed) dataset along the FIP.

Next, we demonstrated that we could apply the operations CL and Co in sequence.

Beginning with the BERT base model, we applied the FIP model to generate networks that could perform YELP and IMDB sentiment analysis, and then compressed the resulting networks to generate 6 distinct networks with sparsity ranging from $10-60\%$ where all sparsified networks maintained performance on the sentiment analysis tasks (Fig. 2.23C). We, then, performed the same operations but changed the order of operations. We, first, sparsified the base BERT model to the same six sparsifications and, then, applied the CL operation to sub-train the resulting models on IMDB sentiment analysis. The models generated through application of CL-Co(w) and Co-CL(w) achieved similar functional performance in terms of sparsification and task performance but had distinct weight configurations.

2.7.2. FIPs in BERT retains natural language understanding

In the main-text (Fig. 2.23), we demonstrated that we can compress a pre-trained BERT to varying levels of sparsity on a sentence completion task (masked language modeling task) by traversing FIPs in the BERT weight space. The line plot in Fig. 2.24C shows that sparser counterparts discovered by the FIP maintain their perplexity score (methods) as a function of increasing sparsity, indicating that BERTs with 80% of their weights (parameters) set to zero retain their ability to parse sentences and find appropriate words for the fill-in-the-blanks task (Fig. 2.21A (table)). Additionally, we find that the FIP length in weight space from the pre-trained BERT to its sparser counterparts increases with the degree of sparsification. In Fig. 2.21A, we find that the FIP length (methods) to the 80% sparse BERT is 96.95% longer than the FIP length to the 20% sparse BERT.

FIP traversal discovers sparse BERTs that retain their ability on general language understanding tasks (GLUE) Wang et al., 2018. Originating from dense BERTs fine-tuned on various GLUE tasks, the FIPs discover sparse BERTs that perform within 15% accuracy of their dense counterparts as shown in barplot of Fig. 2.24D.

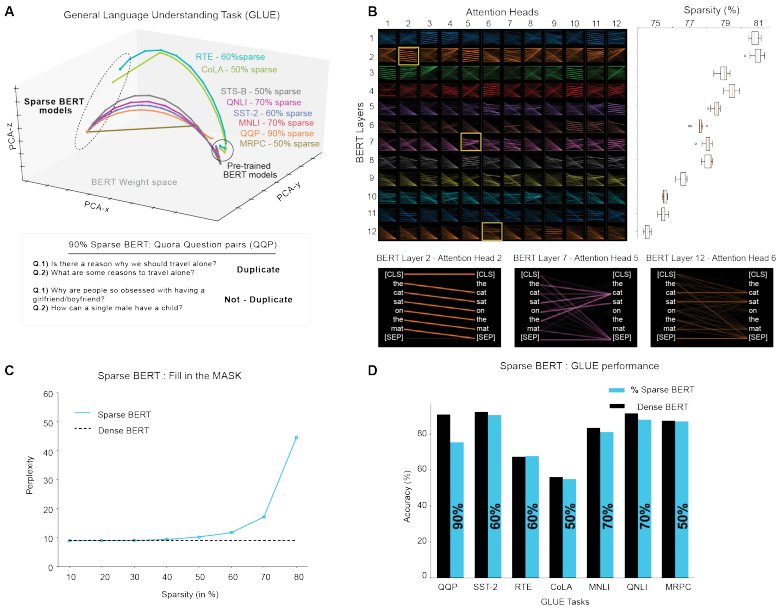


Figure 2.24: **FIPs in large transformer based language model (BERT) weight space.** (A) FIP from BERT's fine-tuned on GLUE task to their p% sparse counterpart. Dots in 3d lineplot are weight configurations of 12-layered BERT models with 12 attention heads per layer in PCA space. Legend abbreviations are the different GLUE tasks and BERT sparsity, elaborated in the text. Table shows test example from QQP task and the inference from a 90% sparse BERT. (B) Birds eyeview of attention across all layers and attention heads for a specific input on an 80% sparse BERT. Boxplot (right) shows sparsity distribution of attention heads in each layer across all the BERT layers. Visualizing attention patterns of a single attention head in a layer for a specific input "the cat sat on the mat." (C) Perplexity score for sparse BERT (solid blue) and dense BERT (dashed blue). (D) GLUE performance of dense BERT (black) and p% sparse BERT (blue).

The QQP (Quora question pairs) task (Fig. 2.24A) that involves detecting if two questions are paraphrased versions of each other can be performed by the 90% sparse BERT, suggesting that QQP-like tasks have lower task-complexity than RTE-like tasks (recognizing textual entailment).

In Fig. 2.24B, we find that the attention heads in each layer are equally uniformly sparsified (see variance of the boxplot in Fig. 2.24B (right)) suggesting that all attention heads within a layer are equally relevant for the considered language task. In Fig. 2.24B, we find that attention heads across different layers in the 80% sparsified BERT has subtly different attention mechanisms, as every word in the sentence at-

tends to different segments of the input sentence incorporating different contextual information. For instance, each word in the 2nd attention head of layer 2 attends to its next word in the sentence, all the words in the 5th attention head of layer 7 attend to the 'subject' ("cat") of the input sentence, while the nouns ("cat," "mat") in the 6th attention head of layer 12 are attending to themselves, while the prepositions ("on," "the") are indicating the presence of an end-of-statement.

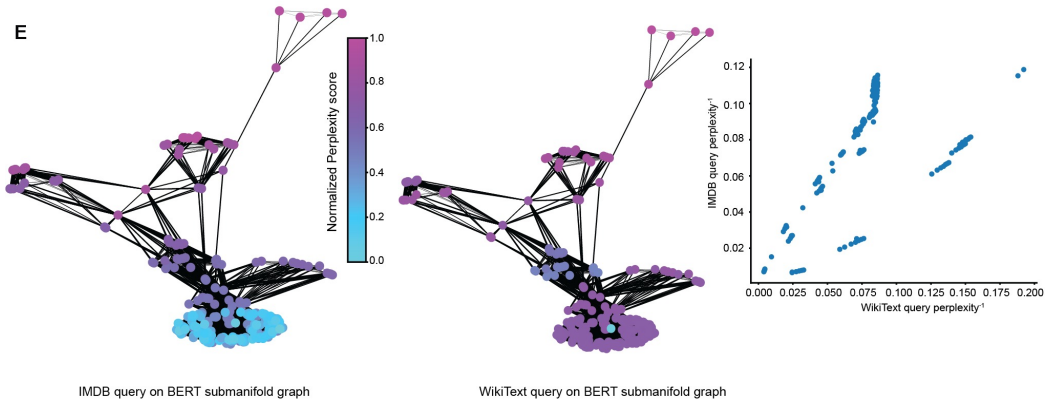


Figure 2.25: **Graph-connected networks in BERT weight space** (E) Graph connected set of 300 BERT models trained on sentence completion on wikipedia and yelp datasets, colored by perplexity scores evaluated using two new query datasets, wikitext and imdb. Nodes correspond to individual BERT models and edges correspond to euclidean distance between BERT models in weight space. (E, rightmost) Scatter plot of inverse perplexity scores for two queries—IMDB and WikiText datasets.

In total, we used the FIP framework to generate networks using a set of different natural language tasks and compression tasks yielding a path-connected set of 300 BERT models. The 300 networks define a sub-manifold of weight space that contains models customized for distinct sub-tasks (Fig. 2.25). The sub-manifold, then, provides a computational resource for solving new problems. We can query the resulting FIP sub-manifold with unseen data by using perplexity (Supplementary) as a measure of a networks intrinsic ability to separate an unseen data set. Using perplexity, we queried the FIP sub-manifold of BERT networks with IMDB data and Wikitext data. We found that the distinct language data sets achieve minimal perplexity on different

classes of networks. Wikitext obtains optimal performance on CoCL networks while IMDB achieves optimal performance on CLCo networks. These results demonstrate that the FIP framework can generate diverse sets of neural networks by transforming networks using distinct meta-tasks. The sub-manifolds of weight space can, then, be queried inexpensively to identify networks pre-optimized for new machine learning problems.

Section 2.8

Conclusion

We have introduced a mathematical theory and algorithm for training path connected sets of neural networks to solve machine learning problems. We demonstrate that path connected sets of networks can be applied to diversify the functional behavior of a network, enabling the network to accommodate additional tasks, to prune weights, or generate diverse ensembles of networks for preventing failure to adversarial attack. More broadly, our mathematical framework provides a useful conceptual view of neural network training. We view a network as a mathematical object that can be transformed through iterative application of distinct meta operations. Meta-operations move the network along paths within the weight space of a neural network. Thus, we identify path connected sub-manifolds of weight space that are specialized for different goals. These sub-manifolds can be enumerated using the FIP algorithm and then queried as a computational resource and applied to solve new problems (Fig. 2.25E).

Fundamentally, our work exploits a parameter degeneracy that is intrinsic to large mathematical models. Previous work has demonstrated that large statistical models often contain significant parameter degeneracy [Machta et al., 2013] leading to 'flat' objective functions [Hochreiter and Schmidhuber, 1997; Hochreiter and Schmidhuber, 1994; Tsuzuku, Sato, and Sugiyama, 2020]. Recent work in physics has demonstrated

that physical models with large numbers of parameters often contain parameter degeneracy such that model parameters can be set to any value within a sub-manifold of parameter space without loss of accuracy in predicting experimental data [Machta et al., 2013]. Mathematically, in the supporting materials we show that the neural networks that we analyze have mathematical signatures of parameter degeneracy after training, through spectral analysis of the metric tensor. Modern deep neural networks contain large numbers of parameters that are fit based on training data, and so are related mathematical objects to physical models with large numbers of parameters set by experimental data , and in fact, exact mappings between statistical mechanics models and neural networks exist [Mehta and Schwab, 2014]. Spectral analysis demonstrates that the weight space contains significant sub-spaces where movement of parameters causes insignificant change in network behavior. Our FIP algorithm explores these degenerate sub-spaces or sub-manifolds of parameter space. Implicitly we show that exploration of the degenerate sub-space can find regions of flexibility where parameters can accommodate a second task (a second image classification task) or goal like sparsification. We apply basic methods from differential geometry to identify and traverse these degenerate sub-spaces. In the Supporting Materials we show that additional concepts from differential geometry including the covariant derivative along a weight space path can be applied to refine paths by minimizing not only the velocity along a weight space path but also acceleration.

Broadly, our results shift attention from the study of single networks to the path-connected sets of neural networks. Biological systems have been hypothesized to explore a range of effective network configurations due to both fluctuation induced drift and modulation of a given network by other sub-systems within the brain. By shifting attention from networks as single configurations or points in weight space to exploring sub-manifolds of the weight space, the FIP framework may help illuminate a potential principle of flexible intelligence and motivate the development of mathe-

matical methods for studying the local and global geometry of functionally invariant solution sets to machine learning problems [Smale, 1998; Mumford and Desolneux, 2010].

Section 2.9

Appendix

In the main-text, we have introduced a geometric framework to solve three core challenges in modern machine learning, namely: (i) Alleviating Catastrophic forgetting, (ii) Network sparsification and (iii) Increasing robustness against adversarial attacks. We will describe the datasets, parameters/hyperparameters used for the algorithms and the pseudocode for each of the core challenges addressed in the main-text.

Catastrophic Forgetting

Datasets and preprocessing:. The models were tested on two paradigms:

- **2-sequential task paradigm**, where the model is exposed two tasks, sampled from the MNIST and Fashion-MNIST dataset sequentially. The MNIST training dataset contains 60,000 gray-scale images of 10 classes of hand-written digits (0–9), and Fashion MNIST training dataset contains 60,000 gray-scale images of 10 classes of fashion items (e.g., purse, pants, etc.). The test set contains 10,000 additional images from each dataset. Together, the two datasets contain 20 classes. The 10 digits in MNIST are labelled 0–9, and the 10 classes in Fashion MNIST are labelled 10–19 in our experiments. Images and labels corresponding to the first 10 classes (MNIST) are fed to the network as Task-1, followed by the images and labels from the next 10 classes (Fashion-MNIST) as Task-2.

- **SplitCIFAR100: 20 sequential task paradigm**, where the model is exposed to 20 tasks, sampled from the CIFAR100 dataset. The CIFAR100 dataset

contains 50,000 RGB images for 100 classes of real-life objects in the training set, and 10,000 images in the testing set. Each task requires the network to identify images from 5 non-overlapping CIFAR100 classes.

Network architecture:. All state-of-art methods for alleviating CF (presented in the main-text) in the 2-task and 20-task paradigm used the same network architecture, as described below.

- We use a 5-layered CNN with a total of 20 output classes (10 from MNIST and 10 from Fashion-MNIST) for the 2 task paradigm. The first 2 layers are convolutional layers, with 32 and 64 conv-filters with 3x3 kernel size, respectively. The last three layers are fully connected layers with 600, 120 and 20 nodes in Layers 3, 4, and 5, respectively. Both the convolutional layers have a 2D batchnorm as well as a 2x2 MaxPool layer. All the layers (except layer 5) has a ReLU non-linearity. The 5 layer CNN has a total of 1.4 million trainable parameters.

- We use a Reduced ResNet18 with a total of 100 output classes for the 20 task paradigm from SplitCIFAR100. The Reduced ResNet18 has three times lesser feature maps in each of the layers (as compared to ResNet18 He et al., 2016), same as the architecture introduced in Lopez-Paz and Ranzato, 2017. Reduced ResNet18 has a total of 1.1 million trainable parameters.

Pseudo-code: FIP construction for CF problems.

Code specifications. All the code was written in the PyTorch framework, and the automatic-differentiation package was extensively used for constructing computational graphs and computing gradients for updating network parameters. The code for constructing FIP's for the 2-task and 20-task paradigm were run on Caltech's

Algorithm 1 FIP construction for CF problems

Require: λ, η: step-size hyperparameters, N_T: Number of sequential tasks

procedure FIP-CF(λ, η, N_T)
    random initialize $\mathbf{w_0}$
    $B_i \leftarrow \{\}\ \forall$ i = 1,2, ..., N_T ▷ Buffer with n_{mem} memories from previous tasks
    for $i \leftarrow 1$ *to* N_T **do**
        end
        $\mathbf{w_i} \leftarrow \mathbf{w_{i-1}}$
    ($\mathbf{x}$, $\mathbf{t}$) $\leftarrow$ Task-i ▷ minibatch of images (x) and target labels (t) from task-i
    $B_i \leftarrow B_i \cup \mathbf{x}$ ▷ update buffer
    CEloss $\leftarrow$ crossEntropy($f(\mathbf{x}, \mathbf{w_i}), \mathbf{t}$) ▷ Classification loss for new task
    Yloss $\leftarrow$ 0 **for** $j \leftarrow 1$ *to i-1* **do**
        end
        Yloss += Ydist($f(\mathbf{x}, \mathbf{w_i}), f(B_j, \mathbf{w_{i-1}})$) ▷ Distance moved in output space (Y)
    $S \leftarrow$ CEloss + λ*Yloss ▷ Construct FIP with direction from loss gradient
    $\mathbf{w_i} \leftarrow \mathbf{w_i} - \eta \nabla_{\mathbf{w_i}} S$
    $\mathbf{w_i}$
end procedure

High-Performance computing cluster—using a single GPU for a total time of 1 hour and 10 hours, respectively (for the 2-task, 20-task paradigm).

Parameters used. The parameters used for current state-of-art methods across different models and datasets have been selected after grid-search to maximize accuracy.

- **Functionally invariant path (FIP) for 2-task paradigm**: $\eta = 0.01$, optimizer used: Adam, weight decay = 2e-4, $\lambda = 1$, n-memories from previous task = 500/60000 (=0.8% previous dataset).
- **Elastic weight consolidation (EWC) for 2-task paradigm**: Optimizer used = Adam, EWC regularization coefficient (λ)=5000, learning-rate = 0.001, batch-size = 128, number of data samples from previous task to construct Fisher metric = 500.

- **Functionally invariant paths (FIP) for 20-task paradigm**: $\eta = 0.01$, optimizer used: Adam, weight decay = 2e-4, $\lambda = 1$, n-memories from previous task = 250/2500 (=10% previous tasks).

- **Gradient episodic memory (GEM) for 20-task paradigm**: n-memories from previous task = 250, learning-rate = 0.01, number of epochs (per task) = 20, memory-strength = 0.5, batch-size = 128.

- **Elastic Weight consolidation (EWC) for 20-task paradigm**: Optimizer used = Adam, EWC++ alpha = 0.9, EWC regularization coefficient (λ)=5000, learning-rate = 0.001, Fisher metric update after 50 training iterations, batch-size = 128.

Implementation of other CF methods:. We implemented the Elastic Weight consolidation (EWC) method by adapting code from the repository: `https://github.com/moskomule/ewc.pytorch`. The Gradient episodic memory (GEM) method was applied by adapting code from: `https://github.com/facebookresearch/GradientEpisodicMemory`.

Network sparsification

Datasets and preprocessing:. The models were sparsified on two well-known image datasets:

- **MNIST**: The MNIST training dataset contains 60,000 gray-scale images of 10 classes of hand-written digits (0–9). The test set contains 10,000 additional images from the 10 digit classes.

- **CIFAR-10**: The CIFAR10 training dataset contains 50,000 RGB images of 10 classes of natural images (like trucks, horses, birds, ships, to name a few). The test set contains 10,000 additional images from each of the 10 classes.

Network architecture:. The networks used for demonstrating the strategy of constructing FIP in weight space for compression are:

- We used Multilayer perceptron (LeNet-300-100), which has 3 fully connected layers for the MNIST task. The first layer (input) has 784 nodes, hidden layers 2, 3 have 300 and 100 nodes respectively. The last layer (output) has 10 nodes (corresponding to 10 digit classes in the dataset). LeNet-300-100 has a total of 484000 trainable parameters (all non-zero, post training on MNIST). All the layers (except output layer) has ReLU non-linearity.

- We used ResNet20 with a total of 10 output classes for training and compression on the CIFAR-10 dataset. The ResNet-20 network has 20 convolutional layers with skip connections, with a total of 0.27 million trainable parameters.

Algorithm 2 FIP construction for network sparsification

Require: λ, η: step-size hyperparameters
Require: p: Final desired network sparsity (in %)
Require: $\mathbf{w_t}$: Network trained on MNIST or CIFAR-10 dataset
procedure FIP-Sparse(λ, η, p, $\mathbf{w_t}$)
  $\mathbf{w} \leftarrow \mathbf{w_t}$**while** *(*$||\mathbf{w}||_0/||\mathbf{w_t}||_0$*) NOT (*$1 - p/100$*)* **do**
  end
  Until $\mathbf{w}$ not in p% sparse submanifold
  $\mathbf{w_p} \leftarrow$ project($\mathbf{w}$, p) ▷ Set p% of smallest weights to zero
  $L(\mathbf{w}) \leftarrow ||\mathbf{w} - \mathbf{w_p}||_2$
  $\mathbf{x} \leftarrow$ Dataset (MNIST or CIFAR) ▷ Sample minibatch of images from dataset
  OPloss $\leftarrow$ odist($f(\mathbf{x}, \mathbf{w}), f(\mathbf{x}, \mathbf{w_t})$) ▷ Distance moved in output space
  $S \leftarrow$ OPloss + λ * $L(\mathbf{w})$
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} S$ ▷ Constructing FIP towards sparse submanifold
  $\mathbf{w}$
end procedure

Pseudo-code: FIP construction for network sparsification.

Code specifications. All the code was written in the PyTorch framework, and the automatic-differentiation package was extensively used for constructing compu-

tational graphs and computing gradients for updating network parameters. The code for constructing FIP's to the p% sparse submanifolds were run on Caltech's High-Performance computing cluster - using a single GPU for a total time ranging between 2-6 hours for final network sparsity's below 80%, and between 24-30 hours for identifying high performance networks in submanifolds with larger than 80% sparsity.

Parameters used:.

- **FIP for network sparsification:** $\lambda = 1$, $\eta = 0.01$, optimizer used: Adam ($\beta = (0.9, 0.999)$), final (desired) network sparsities for LeNet-300-100 on MNIST: $p = [20\%, 67\%, 89\%, 96\%, 98.7\%, 99\%, 99.1\%, 99.4\%]$, final (desired) network sparsities for ResNet-20 on CIFAR-10: $p = [20\%, 36\%, 49\%, 59\%, 67\%, 79\%, 83\%, 89\%, 93\%, 95\%]$.

- **Lottery ticket hypothesis:** *(For LeNet-MNIST)*:= batch-size = 128, model-init = kaiming-normal, batchnorm-init = uniform, pruning-strategy => sparse-global, pruning-fraction = 0.2, pruning-layers-to-ignore = fc.weight, optimizer-name = sgd, learning rate = 0.1, training-steps = 40 epochs. *(For ResNet20-CIFAR10)*: batch-size = 128, model-init = kaiming-normal, batchnorm-init = uniform, pruning-strategy = sparse-global, pruning-fraction = 0.2, optimizer-name = sgd, learning rate = 0.1, training-steps = 160 epochs, momentum = 0.9, gamma = 0.1, weight-decay => 0.0001.

Implementation of other sparsification methods. We implemented the Lottery ticket hypothesis for sparsifying both, LeNet-300-100 trained on MNIST and ResNet20 trained on CIFAR-10. To do so, we adapted code from the repository: `https://github.com/facebookresearch/open_lth`.

Adversarial robustness

Datasets and preprocessing. The models were trained on CIFAR-10 dataset and the adversarial examples were generated on the same using the projected gradient descent (PGD) method.

- **CIFAR-10**: The CIFAR10 training dataset contains 50,000 RGB images of 10 classes of natural images (like trucks, horses, birds, ships to name a few). The test set contains 10,000 additional images from each of the 10 classes.

Network architecture. For the adversarial robustness section, we used the VGG-16 network Simonyan and Zisserman, 2014, which has 16 layers, and a total of 138 million trainable parameters.

Generating an adversarial attack. We used the projected gradient descent (PGD) method to generate CIFAR-10 data samples that are imperceptibly similar to their original images for humans, but cause significant performance loss to deep networks. The PGD attack computes the best direction (in image space) to perturb the image such that it maximizes the trained networks' loss on the image while constraining the L_{inf} norm of the perturbation.

The procedure for generating adversarial inputs are detailed below:

- Randomly initialize a VGG16 network and train it on CIFAR-10 (trained network = $\mathbf{w_t}$)
- Take a single image-input ($\mathbf{x}$) from the CIFAR-10 dataset and pass it through the trained network, and calculate the gradient of the classification loss (cross-entropy (C) with respect to the input (grad = $\nabla_{\mathbf{x}} C(\mathbf{w_t}, \mathbf{x}, \mathbf{y})$).
- Construct an adversarial input ($\mathbf{x}'$) by taking multiple steps (S) in the image-input space, wherein the adversary is within within an ϵ-l_∞ bound. $\mathbf{x}^{t+1} = \prod_{x+S}$

$(\mathbf{x}^t + \alpha\ \mathrm{sgn}(\nabla_{\mathbf{x}} C(\mathbf{w_t}, \mathbf{x}, \mathbf{y})))$. Here, we take as many steps (S) as required until the adversarial input ($\mathbf{x}^{t+1}$) exits the ϵ-l_∞ bound. We choose ϵ=0.3 and α=2/255 for generating CIFAR10 adversarial examples against VGG16 networks.

Representation diversity score. We compute the representation diversity score for both ensembles (FIP, DeepNet) by evaluating the standard deviation of the L_2 norm of the network's activation across all networks in the ensemble along each layer for a set of image-inputs.

Coherence between two models. We compute the overlap of the Adversarial subspace between networks in the FIP ensemble and the trained surrogate network by evaluating the cosine distance between the gradients of the loss function of the FIP networks and the trained surrogate network with respect to an input (x).

Say, the gradient of loss function with respect to input (x) for the two models are: $J_x(\theta_0, x, y)$ and $J_x(\theta_1, x, y)$. The cosine distance between the gradients is evaluated as:

$$CS(\bigtriangledown_x J_0, \bigtriangledown_x J_1)) = \frac{< \bigtriangledown_x J_0, \bigtriangledown_x J_1 >}{|\bigtriangledown_x J_0||\bigtriangledown_x J_1|}. \tag{2.5}$$

The cosine distance between the gradients provides a quantitative measure for how likely an adversarial input that affects the surrogate network would attack the model sampled along an FIP.

To evaluate the coherence across all sampled models in the FIP and a trained surrogate network, we measure the maximum cosine similarity between all pairs of gradient vectors in the set.

$$\max_{a \in 1,..,N} CS(\bigtriangledown_x J_a, \bigtriangledown_x J_s)) \tag{2.6}$$

Here, J_a refers to the gradient of 'N' networks sampled along the FIP for a single input (x), while J_s refers to the gradient of the trained surrogate network for the input (x).

Algorithm 3 FIP for adversarially robust ensembles

Require: η : step-size, $\mathbf{w_t}$: Network trained on CIFAR-10 dataset, ϵ: l_∞ of adversary perturbation

Require: δ : permissible change in output distance, **max-iter:** number of steps in the FIP

procedure FIP-ENSEMBLE(η, $\mathbf{w_t}$, δ, ϵ)
  $\mathbf{w} \leftarrow \mathbf{w_t}$
  $ii \leftarrow 0$ ▷ setting counter = 0
  $F \leftarrow \{\}$ ▷ List of networks in the FIP ensemble **while** $ii \leq$ *max-iter* **do**
  end
  $(\mathbf{x}, \mathbf{y}) \leftarrow$ Dataset (CIFAR10) ▷ Sample minibatch of images from dataset
  S $\leftarrow$ odist$(f(\mathbf{x}, \mathbf{w}), f(\mathbf{x}, \mathbf{w_t}))$ ▷ Output space distance for varying network's weights
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} S$ ▷ Construct undirected FIP
  $\mathbf{x}' \leftarrow \mathbf{x} + \epsilon \text{sgn}(\nabla_{\mathbf{x}} C(\mathbf{w}, \mathbf{x}, \mathbf{y}))$
  $H \leftarrow$ odist$(f(\mathbf{x}, \mathbf{w}), f(\mathbf{x}', \mathbf{w}))$ ▷ Output space distance for perturbed input **if** $H \leq \delta$ **then**
  end
  $F \leftarrow F \cup \mathbf{w}$
  $ii \leftarrow ii + 1$
  F ▷ Returning FIP ensemble with adversarial robustness
end procedure

Pseudo-code: FIP for adversarial robust ensembles.

Code specifications. All the code was written in the PyTorch framework, and the automatic-differentiation package was extensively used for constructing computational graphs and computing gradients for updating network parameters. The code for constructing undirected FIPs in the weight space, followed by sampling a small sub-set of networks along the FIP was run on Caltech's High-Performance computing cluster - using a single GPU for a total time ranging between 2-6 hours.

Parameters used:. To generate ensembles of deep-networks, we selected parameters after a grid-search to maximize robustness against adversarial failure.

- **FIP ensemble**: η=0.01, ϵ=0.3, Mini-batch size = 100, δ=35 (Inputs to the FIP construction/ensemble pseudo-code detailed above).
- **Adaptive diversity promoting (ADP) ensemble**: alpha = 2, beta = 0.5, (alpha, beta are parameters maximizing diversity of ensemble) optimizer used = SGD, momentum = 0.9, learning rate = 0.05, weight-decay = 2e-4, batch-size = 128, num-networks-per-ensemble = 3, 5, 10 (three different ensembles).
- **Fast Geometric ensembling (FGE)**: model = VGG16, epochs = 40, weight-decay = 3e-4, learning-rate-1 = 0.5e-2 learning-rate-2 = 1e-2, cycle = 2.

Implementation of other ensemble generation methods for adversarial robustness:. We generated ensembles of deep networks (VGG16) using three state-of-art methods. The first method, "DeepNet (DN) ensemble" are constructed by training multiple independently initialized VGG16 networks. The second method 'Adaptive Diversity promoting (ADP)' is obtained by adapting the code in this repository: `https://github.com/P2333/Adaptive-Diversity-Promoting`. The third method 'Fast geometric ensembling' is obtained by adapting the code taken from this repository: `https://github.com/timgaripov/dnn-mode-connectivity`.

FIP for multiple 'operations' on language model — BERT

We scale our FIP geometric framework to perform multiple operations (like, continual learning, compression) on BERT language models that are very large and are capable of parsing large amounts of text scraped from different internet sources (like, wikipedia, yelp reviews, imdb reviews to name a few).

Datasets and preprocessing. We performed two operations, (i) network compression and (ii) continual learning on BERT, fine-tuned on different language datasets, downloaded from the HuggingFace (HF) website.

- **Wikipedia:** The Wikipedia - English datasets are downloaded from this link: `https://huggingface.co/datasets/wikipedia`. We used the Wikipedia dataset on the masked learning model (MLM) task.
- **Yelp reviews:** The Yelp-reviews datasets is obtained from hugging-face, downloaded via this link: `https://huggingface.co/datasets/yelp_review_full`.
- **IMDB reviews:** The IMDB-reviews datasets is obtained from hugging-face, downloaded via this link: `https://huggingface.co/datasets/imdb`.
- **GLUE dataset:** The GLUE dataset is obtained from hugging-face, downloaded from this link: `https://huggingface.co/datasets/glue`. The GLUE tasks performed in this paper are: (i) Quora Question Pairs (QQP), (ii)Stanford sentiment (SST-2), (iii) Corpus of linguistic acceptability (CoLA), (iv) Recognizing textual entailment (RTE), (v) Multi-genre natural language inference (MNLI), (vi) Microsoft research paraphrase corpus (MRPC), (vii) Question-answering natural language inference (QNLI).

Network architecture. BERT is a popular transformer model with 12 layers (transformer blocks), each with a hidden size of 768, 12 self-attention heads in each layer with a total of 110 million parameters Devlin et al., 2018. BERT has been pre-trained on 45GB of wikipedia, using the masked language model (MLM) task, and next sentence prediction (NSP).

Sentence completion (Masking) tasks: For the masking tasks (wherein 15% of the words in the input sentence are masked (or blanked)), the BERT network has

an MLM head appended to the network. The MLM head produces a 3-dim tensor as the output, wherein the dimensions correspond to (i) number of sentences in a single batch (batch-size), (ii) number of blanked out words in a sentence, and (iii) number of tokens in the BERT vocabulary: 30,000 tokens.

Sentence classification tasks: For the sentence classification task, a sentence classifier head was appended to the BERT architecture. Here, the classifier head produces a 2-dim output tensor, wherein the dimensions correspond to (i) batch-size and (ii) number of unique classes in the classification problem.

Code specifications. All the code was written in the PyTorch framework, and the automatic-differentiation package was extensively used for constructing computational graphs and computing gradients for updating network parameters. The code for constructing FIPs in the BERT weight space for continual learning on Yelp and IMDB sentiment analysis, and for BERT sparsification were run on Caltech's High-Performance computing cluster - using two GPU's for a total time of 2 hours, 6 to 30 hours (for the continual learning, sparsification operations respectively).

Parameters used:.

- **Functionally invariant path (FIP) for continual learning**: η = 2e-5, λ=1, n-memories from previous task = 2000/650000 = (0.8% previous dataset), Optimizer used: AdamW.
- **Functionally invariant path (FIP) for BERT sparsification**: η = 2e-5, λ=1, Optimizer used: AdamW, Final (desired) network sparsities for the GLUE task: Task (p% sparse): RTE (60% sparse), CoLA (50% sparse), STS-B (50% sparse), QNLI (70% sparse), SST-2 (60% sparse), MNLI (70% sparse), QQP (90% sparse), MRPC (50% sparse), Final (desired) network sparsities for

Wikipedia sentence completion: [10%, 20%, . . . , 90%].

FIP length. The FIP length is evaluated by sampling a large number of networks along the FIP, and summing the euclidean distance between all consecutive pairs of networks. Say, the weights of networks sampled along the FIP are denoted by: $\mathbf{w_1}, \mathbf{w_2}, \mathbf{w_3}, ..., \mathbf{w_n}$

$$\text{FIP-length} = \Sigma_{i=2}^{n} ||\mathbf{w_i} - \mathbf{w_{i-1}}||_2.$$

Perplexity score: language models. Perplexity is an evaluation metric used to measure how "surprised" a language model is when it encounters a new task. That is, a higher perplexity implies more surprise, suggesting that the language model does not have much insight into how language works. Mathematically, we define perplexity as the exponential of the cross entropy loss on the evaluation dataset.

$$\text{PPL}(\theta) = \exp[\Sigma_{i=1}^{n_e} l(\theta, x_i)]$$

where, θ is the parameters of the BERT model, x_1, x_2, .., x_{n_e} are the n_e inputs from the evaluation dataset. $l(\theta, x_i)$ evaluates the cross-entropy loss of a BERT model parameterized by θ for a single input x_i.

Section 2.10

Mathematical musings

In the main-text, we apply mathematical tools from differential geometry to study the response of neural networks to weight perturbation. Our fundamental construction is that we consider weight space, W, to be a smooth manifold endowed with a Riemannian metric, $\mathbf{g}$, so that the pair $(W, \mathbf{g})$ is a Riemannian manifold. We construct the metric through a pullback procedure where we pullback a distance metric

on the output space Y onto the weight space W. Following this construction, our analysis of weight change follows naturally by using standard tools from differential geometry including the tangent space, the covariant derivative and the geodesic to analyze changes in network performance given changes in network weights.

An important aspect of our construction is that we proceed by considering the weight space itself to be the manifold, and pull-back a *distance metric* onto W. The construction allows us to isolate mathematical complexity concerning the definition of the neural network within the construction of the metric itself. Following the construction of the metric, we can conveniently analyze weight perturbation by simply applying our non-Euclidean metric tensor to calculate distances within W where W is homeomorphic to standard Euclidean space. In what follows we discuss the construction of the Riemannian manifold and how the mathematical properties of $\mathbf{g}$ as a positive (semi)-definite bilinear form arise.

A Riemannian manifold simply consists of a smooth topological manifold endowed with a Riemannian metric [Loring W Tu, 2017]. A smooth topological manifold, M, is a locally Euclidean space [Loring W. Tu, 2011]. By locally Euclidean, we mean that around every point, $p \in M$ there is a function, ϕ, that maps a neighborhood of M, U where $p \in U \subset M$, to $\mathbb{R}^n$ ($\phi : U \to \mathbb{R}^n$) so that the collection $\{(U_\alpha, \phi_\alpha)\}$, known as an atlas, covers M . In the general case, we may need many different open sets U_α to cover M. The case of weight space is quite convenient in that a single map, the identity map, gives us an atlas for W.

Our weight space W is trivially homeomorphic (and diffeomorphic) to $\mathbb{R}^n$ by the identity map, and so therefore W is a smooth manifold. It is the simplicity of the manifold that that gives our method much of its practical power.

Now, we introduce a metric onto W that endows the manifold with a notion of distance that encapsulates the function properties of the underlying neural network. Intuitively, we can think of W as becoming a curved space due to the influence of

the functional properties of the neural network on the local structure of space. Our approach has analogies with physical models where the path of a particle through an ambient space can be influenced by a metric which is the manifestation of a physical force like gravity [Thorne, Misner, and Wheeler, 2000]. We view neural networks as dynamically moving along a smooth manifold whose notion of distance is functional.

Specifically, we consider a neural network to be a smooth, $\mathbb{C}^\infty$function $f(\mathbf{x}, \mathbf{w})$, that maps an input vector, $\mathbf{x} \in \mathbb{R}^{\mathrm{k}}$, to an output vector, $f(\mathbf{x}, \mathbf{w}) = \mathbf{y} \in \mathbb{R}^{\mathrm{m}}$. The function, $f(\mathbf{x}, \mathbf{w})$, is parameterized by a vector of weights, $\mathbf{w} \in \mathbb{R}^{\mathrm{n}}$, that are typically set in training to solve a specific task. In general, several popular neural network functions like the rectified linear unit (ReLU) are not actually $\mathbb{C}^\infty$(do not have continuous derivatives of all orders). For example, the ReLU function $h(x) = \max(x, 0)$ has a discontinuity at $h'(0)$. However, the function is commonly approximated by the soft-plus function $h(x) = \log(1 + \exp(x))$ which is $\mathbb{C}^\infty$, and so there is not a fundamental problem.

The training data itself has an interesting and more subtle impact on our metric. To construct a metric on W, we first consider the map generated by the network f given a fixed data point $\mathbf{x}$

$$f(\mathbf{x}, \mathbf{w}) : W \to \mathbb{R}^m. \tag{2.7}$$

We call this map the functional map. A specific example of such a map is that $\mathbf{x}$ could be a specific vector of image data from MNIST, and f maps this data to a $m = 10$ dimensional space that scores the image for each of the 10 possible handwritten digits. Globally, we note that f in general will not be one-to-one or onto.

Locally, we ask how the output of f changes for an infinitesimal weight change

$$f(\mathbf{x}, \mathbf{w} + \mathbf{dw}) = f(\mathbf{w}) + df \tag{2.8}$$

$$= f(\mathbf{w}) + \frac{\partial f_i}{\partial \mathbf{w_j}} \mathbf{dw_j} \tag{2.9}$$

where we take $\mathbf{dw_i}$ to be standard basis vectors in W, and $\mathbf{J} = \frac{\partial f_i}{\partial \mathbf{w_i}}$ is the Jacobian matrix of partial derivatives. In general $\mathbf{J}$ will be an $n \times m$ matrix and, therefore, $\text{rank}(J) \leq \min(n, m)$, so that the rank of J is obviously determined by the number of weights and the number of output functions. A key difference between our framework and classical settings in which differential geometry is applied is that, for us, $n \neq m$. In fact, it will be a very special case that achieves an equality of weights and output function.

To construct our metric, we use mean squared error to measure the distance between network outputs generated by the unperturbed ($\mathbf{w}$) and perturbed networks ($\mathbf{w} + \mathbf{dw}$) as

$$d(\mathbf{w}, \mathbf{w} + \mathbf{dw}) = |f(\mathbf{x}, \mathbf{w}) - f(\mathbf{x}, \mathbf{w} + \mathbf{dw})|^2 = \mathbf{dw}^T \ (\mathbf{J}^T \ \mathbf{J}) \ \mathbf{dw} \tag{2.10}$$

$$= \mathbf{dw}^T \ \mathbf{g_w} \ \mathbf{dw} \tag{2.11}$$

where we have used the local notion of distance to derive a metric, $\mathbf{g}$, that converts local weight perturbations into a distance. It is important to notice that $\mathbf{J}$ and $\mathbf{g}$ are fields that vary across W. We can evaluate the metric at a position location in space or as we move along a path through weight space.

Formally, we think of our metric as providing an inner product at every location in weight space. For general manifolds, the mathematical construction is to consider a tangent plane or tangent space at each point $p \in W$, and to imagine a plane that locally approximates a curved manifold at each point. In this case, the metric tensor provides a local inner product and hence a local notion of distance.

Therefore, we define an inner product on the tangent space at any point $p \in W$ as

$$\langle u, v \rangle_p = u^T \mathbf{g}_p v \tag{2.12}$$

where we take $\mathbf{u}, \mathbf{v}$ to be vectors in the tangent space, and we use $\mathbf{g}_p$ to indicate our metric evaluated at the point p. Formally, tangent vectors are typically constructed as local differential operators, but can be viewed intuitively as small arrows at p [Loring W Tu, 2017; Thorne, Misner, and Wheeler, 2000].

A Riemannian metric is an inner product that satisfies a set of conditions. The inner product must be symmetric, bilinear and positive definite. The positive definite condition can be relaxed through construction of a pseudo-metric. The inner product provides the familiar notions of distance that exist in classical Euclidean spaces.

In general, the notion of a metric is separate from its representation as a matrix, but there is a natural map between inner products and matrices that we can exploit. We can see that our metric satisfies symmetry and linearity through our definition of the metric as a product of the Jacobian matrix and its transpose. Linearity is a natural consequence of standard matrix operations. In the case of symmetry, $\langle u, v \rangle_p = u^T J^T J v = (Jv)^T (u^T J^T)^T = v^T J^T J u = \langle v, u \rangle_p$. Therefore, our metric is, in general, both symmetric and linear in its arguments.

However, the positive definiteness of our metric is determined by the rank of the Jacobian matrix, $\mathbf{J}$. In the typical case $n > m$, and the rank of the Jacobian matrix will be limited by m. Our metric, $\mathbf{g}$ when viewed as a local bilinear form or as an $n \times n$ matrix will not be full rank and will be a pseudo-metric. We can analyze the a metric by considering its representation as a matrix, and, thus, apply tools from linear algebra. In general a matrix $A \in \mathbb{R}^{n \times n}$ is positive definite if $x^T A x > 0 \; \forall x \in \mathbb{R}^n, \mathbf{x} \neq 0$ or equivalently eigenvalues of $A \; \lambda_i > 0 \; \forall i$. Alternately, a positive semi-definite matrix A has $x^T A x \geq 0 \; \forall x \in \mathbb{R}^n$ and $A \; \lambda_i \geq 0 \; \forall i$.

Since $\mathbf{g}$ is the product $\mathbf{J}^T\mathbf{J}$, $\mathbf{g}$ has $\lambda_i \geq 0$ as can be seen simply by considering the singular value decomposition of $\mathbf{J}$. However, the matrix rank of $\mathbf{g}$ at a point on our manifold is similarly bounded by the rank of J, and $\text{rank}(\mathbf{g}) = \text{rank}(\mathbf{J})$. Therefore, $\mathbf{g}$ will have k eigenvalues that are identically zero $\lambda_i = 0$ where $k = n - \text{rank}(J) < n - m$, so that, in general, a metric constructed based on a single training example is not positive definite but positive semi-definite. Our key results can be applied to both Riemannian manifolds as well as pseudo-Riemannian manifolds.

We can increase the rank of the metric by extending our construction to multiple data points. We can consider a set of data examples, X, so that $\mathbf{x_i} \in X$. For a single example, our neural network function f generates an output $f(\mathbf{x}, \mathbf{w}) \in \mathbb{R}^m$. We call the output space for a single example, $\mathcal{Y}_{x_i}$, and we consider the direct sum of the functional spaces generated from a set of training examples

$$\mathcal{Y} = \bigoplus_{x_i \in X} Y_{x_i}. \tag{2.13}$$

Each Y_{x_i} is homeomorphic to $\mathbb{R}^m$ and $\mathcal{Y}$ is homeomorphic to a direct sum of p copies of $\mathbb{R}^m$. In this case, $\dim(\mathcal{Y}) = m \times p$ where $p = |X|$ is the number of data points used in the construction and $\dim(Y_{x_i}) = m$.

The construction generalizes our notion of functional distance, so that now functional distance involves a sum over all $\mathbf{x_i} \in X$ as

$$d_{\mathcal{Y}}(f(\mathbf{X}; \mathbf{w}), f(\mathbf{X}; \mathbf{w'})) = \sum_{i=1}^{p} \sum_{j=1}^{m} |f_j(\mathbf{x_i}, \mathbf{w}) - f_j(\mathbf{x_i}, \mathbf{w'})|^2 \tag{2.14}$$

where the sum is performed over a set of input vectors $\mathbf{x_i} \in \mathbf{X}$ and over all components j of the output.

The form of the metric tensor also has a natural generalization to the case of multiple input data points, and simply becomes a sum

$$\mathbf{g} = \sum_{i=1}^{p} \mathbf{g}_{\mathbf{x_i}} \tag{2.15}$$

where each $\mathbf{g}_{\mathbf{x_i}}$ is the metric tensor generate from a single data point $\mathbf{x_i}$. Intuitively, our metric is, therefore, just a sum of the metrics generated for each input data point. Even if each metric is a rank-1 matrix, the sum of a set of such rank-1 matrices has increased rank.

Practically, the result is important in applications because the rank of $\mathbf{g}$ is influenced by both inherent properties of the neural network at a point in weight space and the number of training examples. When $n > m \times p$, the Jacobian matrix is not full rank, and so the rank of the metric is data limited. When $m \times p > n$, the Jacobian matrix can still contain degenerate directions due to the geometry of the function f. In many practical cases, it is the curvature of f that we want to examine, and so we want the option of saturating the rank of of metric. Numerically, we show an example in Fig. 2.26 where we evaluate the rank of the metric tensor for network MLP-1 trained on MNIST.

Figure 2.26: **Rank of metric tensor increases with the size of the training data set.** The rank of the metric tensor, $\mathbf{g}$, for MLP-1 with 10 hidden nodes is plotted using variable sizes of the MNIST training dataset. The rank of the metric tensor increases with data set size.

Output space distance metrics:

Many different functional metrics (and pseudo-metrics) are applied in machine learning to analyze the distance between network outputs $f(X, \mathbf{w})$. Specifically, we have found it useful to use both, the Bhattacharyya Distance [Bhattacharyya, 1943] and Euclidean distance to quantify the distance between the function outputs (after the softmax operation).

The Bhattacharyya Distance measures the similarity between two probability distributions [Bhattacharyya, 1943]. To get probability vectors as network outputs, we construct the output space with softmax'ed neural network outputs.

The Bhattacharyya Distance between two softmax'ed network output vectors $(h_1 = f(\mathbf{x}, \mathbf{w_1}), h_2 = f(\mathbf{x}, \mathbf{w_2}))$ is given by:

$$D_B(h_1, h_2) = -ln(dot(\sqrt{h(\mathbf{x}, \mathbf{w_1})}, \sqrt{h(\mathbf{x}, \mathbf{w_2})}))$$
$$D_B(h_1, h_2) = -ln(\sqrt{h(\mathbf{x}, \mathbf{w_1})}^T \sqrt{h(\mathbf{x}, \mathbf{w_2})}).$$

Here, $f(\mathbf{x}, \mathbf{w_1})$ and $f(\mathbf{x}, \mathbf{w_2})$ are neural network outputs passed through a softmax function. For the MNIST, CIFAR-10 dataset, they are 10 element vectors, while for CIFAR-100, they are 100 element vectors. dot(h_1,h_2) is the dot product of the two probability vectors.

Local analysis, tangent space and tangent vectors

A central insight in differential geometry is that the structure of a manifold can be analyzed by considering the tangent space at each point on the manifold and as well as the properties of the Riemannian metric when restricted to that tangent space. Intuitively, the tangent space is a local linear approximation of a manifold at a single point. The Riemannian metric yields an inner product that allows us to calculate the length of tangent vectors within the tangent plane or space. By calculating inner

productions within weight space W we are able to determine the functional response of a network to a local weight perturbation.

The tangent space of W, $T_w(W)$ at a point, $\mathbf{w}$, can be constructed by considering a set of local tangent vectors at a point. Tangent spaces carry algebraic structure of vector spaces. Tangent vectors can be intuitively viewed as tangent arrows anchored at a specific point on the manifold, but are formally defined as a set of local differential operators. For our weight space W, the set of local operators $\frac{d}{dw^i}$, a basis for the tangent space. The differential operators provide a basis:

$$B = \{e_1 = \partial_1, \ldots, e_D = \partial_D\}$$

with

$$\partial_i := \frac{\partial}{\partial w_i}.$$

We can think of these local differential operators as local perturbation operators which carry information about how infinitesimal weight changes impact the functional performance of a network. Formally, the **Riemannian metric**, $\mathbf{g}$, then defines an inner, $\langle e_i, e_j \rangle_{\mathbf{g_w}}$ product within the tangent plane

$$g_w : T_w W \times T_w W \to \mathbb{R} \tag{2.16}$$

$$\langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{g_w}} = \sum_{ij} g_{ij} \; u_i \; v_j. \tag{2.17}$$

We calculate the inner product of a tangent vector $\langle e_i, e_i \rangle_{\mathbf{g_w}}$ which quantifies the total change in functional performance across all training examples given a perturbation. In our formulation of local movement in weight space, we analyze the response of a neural network to weight perturbation by calculating the squared length of tangent vectors which represent local functional perturbations using the metric tensor.

Affine Connections and Covariant Differentiation

Given a manifold, the metric $\mathbf{g}$ provides a local definition of tangent vector length at a single point on the manifold. To compare tangent vectors at different points $\mathbf{w_1}$ and $\mathbf{w_2}$ or along a path $\gamma(t)$ we require a notion of differentiation. The covariant derivative provides exactly this notion allowing us to express changes in the length of tangent vectors given displacement in weight space. Specifically, the covariant derivative $\nabla_v u$ is a linear map, $\nabla : V \times V \to V$ that computes the local change in orientation of a tangent vector $v \in V$ as the vector moves along a second tangent vector $w \in V$. Informally, the covariant derivative quantifies how translation of a vector transforms (rotations, contractions) the tangent vector as measured by the basis provided by other tangent vectors.

As a specific case, given a path $\gamma(t) \in W$, a tangent or velocity vector can be defined at a point $t = t'$ as $\frac{\partial \gamma(t)}{\partial t} = \gamma'(t)$. The metric, at a point $\mathbf{w}$, measures the length of this tangent vector $\sqrt{\langle \gamma'(t), \gamma'(t) \rangle_{\mathbf{g(w)}}}$. In our context, the velocity vectors measure the infinitesimal change in performance incurred by a network as it moves along $\gamma(t)$ from $\gamma(t') \to \gamma(t') + dt\gamma'(t)$.

To better describe the geometric objects on a manifold, we develop the concept of differentiation on a manifold that is in independent of local charts, that is, we would like to develop a derivative operator whose components transform like tensor. In order to define a derivative operator, we need to be able to compare vectors and tensors based at different points on the manifold. The machinery we use is called "affine connection".

In our mathematical framework, the tangent vector $\dot{\gamma}(t) = \frac{\partial \gamma(t)}{\partial t}$ measures the rate of change in functional performance (or network output) along a path $\gamma(t) \in W$ where t is a time-like quantity that parameterizes distinct points along the path $\gamma(t)$.

The length of the tangent vector $\sqrt{\langle \dot{\gamma}, \dot{\gamma} \rangle_g}$ measures the absolute scalar change in functional performance over time. Our first order FIP framework focuses on the

construction of paths $\gamma(t)$ that move between points in weight space $\gamma(0) = w_t$ and $\gamma(1) = w_0$ while minimizing the velocity (strictly) the speed of change in a network's output as the network is moved along the path. In some cases, it is useful to consider both velocity of a network at a specific point, but also the change in velocity as a function of time alone a path. For example, we might want to construct paths that achieve a constant velocity along a path where the change in network output per fixed step dt along the path $\gamma(t)$.

For such paths we have $\sqrt{\langle\dot{\gamma},\dot{\gamma}\rangle_g} = C$ and so that $\langle\gamma,\dot{\gamma}\rangle_g = 0$ and the acceleration of the network must be in a direction orthogonal to the velocity vector along the path γ [Loring W Tu, 2017].

Constant velocity paths are known as geodesic paths, and can be constructed as follows [Loring W Tu, 2017]. Consider a path in weight space as $\gamma(t) = (\gamma_1(t), \gamma_2(t) \dots \gamma_n(t))$ and define the velocity vector of the network along the path as $V = \dot{\gamma(t)} = \sum_i \frac{d\gamma_i}{dt}\hat{x_i}$. Now we can derive an expression for $\frac{dV}{dT}$ as

$$\frac{dT}{dt} = \sum_i \gamma_i(t)\hat{x_i} + \gamma.$$

Consider a point P and a neighboring point Q on the weight manifold, where Q is at a parameter distance Δt from P along curve γ. Let $v(t)$ and $v(t+\Delta t)$ be members of the vector field at P and Q. We then define a new vector field v_0 which equals $v(t)$ at Q and is parallel-transported along γ. The covariant derivative of $v(t)$ at P can be expressed as

$$\nabla_{\gamma(t)} v(t) = \lim_{\Delta t \to 0} \frac{v_0(t) - v(t)}{\Delta t}.$$

Acceleration can also be calculated conveniently from the definition of the Riemannian metric as an inner product [Loring W Tu, 2017] by considering a path, $\gamma(t) \in W$, $t \in [0, 1]$ and velocity vectors $\frac{d\gamma}{dt}|_{t=t'} = v(t)$ calculated at different points in time.

The break-down speed is defined along the path as the inner product of the tangent vector:

$$s = \langle v, v \rangle = \sum_{i=1,j=1}^{n} v_i \; v_j \; g_{ij}.$$

We note that speed is typically defined as $\sqrt{\langle v, v \rangle}$, but we define speed as above due to our squared loss function which differs from the traditional euclidean distance that provides the convention for speed.

In our definition, acceleration is

$$\begin{aligned} \frac{ds}{dt} &= \frac{dg_{ij}}{dt} + 2v_i \frac{dv_j}{dt} \\ &= \frac{dg_{ij}}{dx_k} v + 2v_i \frac{dv_j}{dt}. \end{aligned}$$

Acceleration of a tangent vector can be calculated using a finite difference of the neural networks' softmaxed output along the path (FIP).

References

Amari, Shun-ichi (2016). *Information geometry and its applications*. Vol. 194. Springer.

Benn, Ian and Robin Tucker (1987). *An introduction to spinors and geometry with applications in physics*. published by: Adam Hilger Ltd.

Bhattacharyya, Anil (1943). "On a measure of divergence between two statistical populations defined by their probability distributions." In: *Bulletin of Calcutta Mathematical Society* 35, pp. 99–109.

Blalock, Davis et al. (2020). "What is the state of neural network pruning?" In: *arXiv preprint arXiv:2003.03033*.

Brown, Tom et al. (2020). "Language models are few-shot learners." In: *Advances in Neural Information Processing Systems* 33, pp. 1877–1901.

Devlin, Jacob et al. (2018). "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805*.

Frankle, Jonathan and Michael Carbin (2018). "The lottery ticket hypothesis: Finding sparse, trainable neural networks." In: *arXiv preprint arXiv:1803.03635*.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, pp. 249–256.

He, Kaiming et al. (2016). "Deep residual learning for image recognition." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

Hochreiter, Sepp and Jürgen Schmidhuber (1994). "Simplifying neural nets by discovering flat minima." In: *Advances in Neural Information Processing Systems* 7.

— (1997). "Flat minima." In: *Neural Computation* 9.1, pp. 1–42.

Jumper, John et al. (2021). "Highly accurate protein structure prediction with AlphaFold." In: *Nature* 596.7873, pp. 583–589.

Kaushik, Prakhar et al. (2021). "Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping." In: *arXiv preprint arXiv:2102.11343*.

Kirkpatrick, James et al. (2017). "Overcoming catastrophic forgetting in neural networks." In: *Proceedings of the National Academy of Sciences* 114.13, pp. 3521–3526.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Advances in Neural Information Processing Systems* 25.

LeCun, Yann, Léon Bottou, et al. (1998). "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Yann, Patrick Haffner, et al. (1999). "Object recognition with gradient-based learning." In: *Shape, Contour and Grouping in Computer Vision*. Springer, pp. 319–345.

Lee, Tai Sing and David Mumford (2003). "Hierarchical Bayesian inference in the visual cortex." In: *JOSA A* 20.7, pp. 1434–1448.

Liu, Yinhan et al. (2019). "Roberta: A robustly optimized bert pretraining approach." In: *arXiv preprint arXiv:1907.11692*.

Lopez-Paz, David and Marc'Aurelio Ranzato (2017). "Gradient episodic memory for continual learning." In: *Advances in Neural Information Processing Systems* 30.

Mache, Detlef H., József Szabados, and Marcel G. de Bruin (2006). *Trends and Applications in Constructive Approximation*. Vol. 151. Springer Science & Business Media.

Machta, Benjamin B. et al. (2013). "Parameter space compression underlies emergent theories and predictive models." In: *Science* 342.6158, pp. 604–607.

Mau, William, Michael E. Hasselmo, and Denise J. Cai (2020). "The brain in motion: How ensemble fluidity drives memory-updating and flexibility." In: *Elife* 9, e63550.

McCloskey, Michael and Neal J. Cohen (1989). "Catastrophic interference in connectionist networks: The sequential learning problem." In: *Psychology of Learning and Motivation*. Vol. 24. Elsevier, pp. 109–165.

Mehta, Pankaj and David J. Schwab (2014). "An exact mapping between the variational renormalization group and deep learning." In: *arXiv preprint arXiv:1410.3831*.

Minxha, Juri et al. (2020). "Flexible recruitment of memory-based choice representations by the human medial frontal cortex." In: *Science* 368.6498, eaba3313.

Mumford, David and Agnès Desolneux (2010). *Pattern Theory: The Stochastic Analysis of Real-World Signals*. CRC Press.

Ratcliff, Roger (1990). "Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions." In: *Psychological Review* 97.2, p. 285.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning representations by back-propagating errors." In: *Nature* 323.6088, pp. 533–536.

Silver, David et al. (2017). "Mastering the game of go without human knowledge." In: *nature* 550.7676, pp. 354–359.

Simard, Patrice Y. et al. (1998). "Transformation invariance in pattern recognition—tangent distance and tangent propagation." In: *Neural Networks: Tricks of the Trade*. Springer, pp. 239–274.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556*.

Smale, Steve (1998). "Mathematical problems for the next century." In: *The mathematical Intelligencer* 20.2, pp. 7–15.

Stringer, Carsen et al. (2019). "Spontaneous behaviors drive multidimensional, brainwide activity." In: *Science* 364.6437, eaav7893.

Sünderhauf, Niko et al. (2018). "The limits and potentials of deep learning for robotics." In: *The International Journal of Robotics Research* 37.4-5, pp. 405–420.

Thorne, Kip S, Charles W Misner, and John Archibald Wheeler (2000). *Gravitation*. publisher: Freeman.

Tsuzuku, Yusuke, Issei Sato, and Masashi Sugiyama (2020). "Normalized flat minima: Exploring scale invariant definition of flat minima for neural networks using pac-bayesian analysis." In: *International Conference on Machine Learning*. PMLR, pp. 9636–9647.

Tu, Loring W (2017). *Differential geometry: Connections, Curvature, and characteristic classes*. Vol. 275. Springer.

— (2011). *An introduction to manifolds. Second*. Springer, New York.

Van de Ven, Gido M. and Andreas S. Tolias (2019). "Three scenarios for continual learning." In: *arXiv preprint arXiv:1904.07734*.

Vaswani, Ashish et al. (2017). "Attention is all you need." In: *Advances in Neural Information Processing Systems* 30.

Ven, Gido M. van de, Hava T. Siegelmann, and Andreas S. Tolias (2020). "Brain-inspired replay for continual learning with artificial neural networks." In: *Nature Communications* 11.1, pp. 1–14.

Wang, Alex et al. (2018). "GLUE: A multi-task benchmark and analysis platform for natural language understanding." In: *arXiv preprint arXiv:1804.07461*.

Weisstein, Eric W (2014). "Metric Tensor." In: *https://mathworld.wolfram.com/*.

Wolf, Thomas et al. (2020). "Transformers: State-of-the-art natural language processing." In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45.

Xiao, Han, Kashif Rasul, and Roland Vollgraf (2017). "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms." In: *arXiv preprint arXiv:1708.07747*.

Chapter 3

Growing and self-organizing neural networks from noise

Figure 3.1: **Spiking neural networks art: generated by text-to-image AI.**

Having explored two key attributes of intelligence, namely flexibility and robustness in the last chapter, in this chapter I will explore another important feature of intelligence, namely the ability of intelligence systems to grow and self-organize their intelligence-endowing machinery (aka their neural networks). Although artificial neural networks are very powerful and can be trained to perform a wide range of tasks, they heavily rely on human-designed, hand-programmed architectures for their remarkable performance. However on the other hand, we know that living neural networks emerge through a process of growth and self-organization, beginning with a single cell and ultimately resulting in a brain, an organized and functional computational device. In this chapter, my goal is to ask and address the question: "Can we develop artificial computational devices that can grow and self-organize without human intervention?" To engineer such an independently growing and self-organizing system, we propose a biologically inspired developmental algorithm that can 'grow' a functional, layered neural network from a single initial cell. The algorithm organizes inter-layer connections to construct a convolutional pooling layer, a key constituent of convolutional neural networks (CNN's). Our approach is inspired by the mechanisms employed by the early visual system to wire the retina to the lateral geniculate nucleus (LGN), days before animals open their eyes. We find that the key ingredients for robust self-organization are an emergent spontaneous spatiotemporal activity wave in the first layer and a local learning rule in the second layer that 'learns' the underlying activity pattern in the first layer. In addition to autonomous self-organization, we discovered that the algorithm is adaptable to a wide-range of input-layer geometries, robust to malfunctioning units in the first layer, and so can be used to successfully grow and self-organize pooling architectures of different pool-sizes and shapes. The algorithm provides a primitive procedure for constructing layered neural networks through growth and self-organization. Broadly, this chapter demonstrates that biologically inspired developmental algorithms can be applied to autonomously grow

functional 'brains' in-silico.

Section 3.1

Neural inspiration for growing and self-organizing artificial neural networks

Living neural networks in the brain perform an array of computational and information processing tasks including sensory input processing [Glickfeld and Olsen, 2017; Peirce, 2015], storing and retrieving memory [Tan, Wills, and Cacucci, 2017; Denny, Lebois, and Ramirez, 2017], decision making [Hanks and Summerfield, 2017; Padoa-Schioppa and Conen, 2017], and more globally, generate the general phenomena of "intelligence." In addition to their information processing feats, brains are unique because they are computational devices that actually self-organize their intelligence. In fact brains ultimately grow from single cells during development. Engineering has yet to construct artificial computational systems that can self-organize their intelligence. In this paper, inspired by neural development, we ask how artificial computational devices might build themselves without human intervention.

Deep neural networks are one of the most powerful paradigms in Artificial Intelligence. Deep neural networks have demonstrated human-like performance in tasks ranging from image and speech recognition to game-playing [Koch, Zemel, and Salakhutdinov, 2015; Song and Cai, 2015; Silver et al., 2017]. Although the layered architecture plays an important role in the success [Saxe et al., 2011] of deep neural networks, the widely accepted state of art is to use a hand-programmed network architecture [Krizhevsky, Sutskever, and Hinton, 2012] or to tune multiple architectural parameters, both requiring significant engineering investment. Convolutional neural networks, a specific class of DNNs, employ a hand programmed architecture that mimics the pooling topology of neural networks in the human visual system.

In this chapter, we develop strategies for *growing a neural network* autonomously

from a single computational "cell" followed by *self-organization* of its architecture by implementing a wiring algorithm inspired by the development of the mammalian visual system. The visual circuity, specifically the wiring of the retina to the lateral geniculate nucleus (LGN) is stereotypic across organisms, as the architecture always enforces pooling (retinal ganglion cells (RGC's) pool their inputs to LGN cells) and retinotopy. The pooling architecture (Fig. 3.2a) is robustly established early in development through the emergence of spontaneous activity waves (Fig. 3.2b) that tile the light insensitive retina [Meister et al., 1991]. As the synaptic connectivity between the different layers in the visual system get tuned in an activity-dependent manner, the emergent activity waves serve as a signal to alter inter-layer connectivity much before the onset of vision.

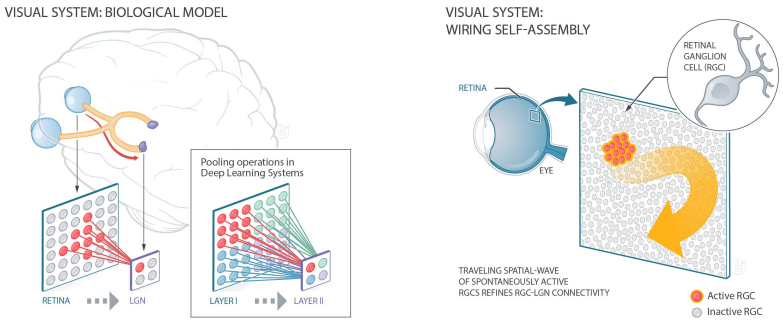


Figure 3.2: **Wiring of the visual circuitry** (A) Spatial pooling observed in wiring from the retina to LGN and in CNN's. (B) Synchronous Spontaneous bursts (retinal waves) in the light-insensitive retina serve as a signal for wiring retina to the brain.

The main contribution of this chapter is that we propose a developmental algorithm inspired by visual system development to *grow and self-organize a retinotopic pooling architecture*, similar to modern convolutional neural networks (CNNs). Once a pooling architecture emerges, any non-linear function can be implemented by units in the second layer to morph it into functioning as a convolution or a max/average pooling. We show that our algorithm is adaptable to a wide-range of input-layer geometries, robust to malfunctioning units in the first layer and can grow pooling architectures of different shapes and sizes, making it capable of countering the key

challenges accompanying growth. We also demonstrate that 'grown' networks are functionally similar to that of hand-programmed pooling networks, on conventional image classification tasks. As CNN's represent a model class of deep networks, we believe the developmental strategy can be broadly implemented for the self-organization of intelligent systems.

Section 3.2

Related work

Computational models for self-organizing neural networks dates back many years, with the first demonstration being Fukushima's neocognitron [Fukushima, 1988; Fukushima and Wake, 1991], a hierarchical multi-layered neural network capable of visual pattern recognition through learning. Although weights connecting different layers were modified in an unsupervised fashion, the network architecture was hard-coded, inspired by Hubel and Wiesel's [Hubel and Wiesel, 1963] description of simple and complex cells in the visual cortex. Fukushima's neocognitron inspired modern-day convolutional neural networks (CNN) [LeCun et al., 1990]. Although CNN's performed well on image-based tasks, they had a fixed, hand-designed architecture whose weights were altered by back-propagation. The use of a fixed, hand-designed architecture for a neural network changed with the advent of neural architecture search [Elsken, Metzen, and Hutter, 2018b], as neural architectures became malleable to tuning by neuro-evolution strategies [Stanley and Miikkulainen, 2002; Real, Moore, et al., 2017; Real, Aggarwal, et al., 2018], reinforcement learning [Zoph and Le, 2016 and multi-objective searches [Elsken, Metzen, and Hutter, 2018a; Zhou and Diamos, 2018]. Neuro-evolution strategies have been successful in training networks that perform significantly much better on CIFAR-10, CIFAR-100 and Image-Net datasets. As the objective function being maximized is the predictive performance on a single dataset, the evolved networks may not generalize well to multiple datasets. On the contrary,

biological neural networks in the brain grow architecture that can generalize very well to innumerable datasets. Neuroscientists have been very interested in how the architecture in the visual cortex emerges during brain development. Meister et al [Meister et al., 1991] suggested that spontaneous and spatially organized synchronized bursts prevalent in the developing retina guide the self-organization of cortical receptive fields. In this light, mathematical models of the retina and its emergent retinal waves were built [Godfrey and N. V. Swindale, 2007], and analytical solutions were obtained regarding the self-organization of wiring between the retina and the LGN [Haussler, 1983; Willshaw and Von Der Malsburg, 1976; Eglen and Gjorgjieva, 2009; N. Swindale, 1996; N. Swindale, 1980]. Computational models have been essential for understanding how self-organization functions in the brain, but have not been generalized to growing complex architectures that can compute. One of the most successful attempts at growing a 3D model of neural tissue from simple precursor units was demonstrated by [Zubler and Douglas, 2009] that defined a set of minimal rules that could result in the growth of morphologically diverse neurons. Although the networks were grown from single units, they weren't functional as they were not equipped to perform any task. To bridge this gap, in this chapter we attempt to grow and self-organize functional neural networks from a single precursor unit.

Section 3.3

Bio-inspired developmental algorithm

In our procedure, the pooling architecture emerges through two processes, growth of a layered neural network followed by self-organization of its inter-layer connections to form defined 'pools' or receptive fields. As the protocol for growing a network is relatively straightforward, our emphasis in the next few sections is on the self-organization process, following which we will combine the growth of a layered neural network with its self-organization in the penultimate section of this chapter.

We first abstract the natural development strategy as a mathematical model around a set of input sensor nodes in the first layer (similar to retinal ganglion cells) and processing units in the second layer (similar to cells in the LGN).

Self-organization comprises of two major elements: (1) A **spatiotemporal wave generator** in the first layer driven by noisy interactions between input-sensor nodes and (2) A **local learning rule** implemented by units in the second layer to learn the "underlying" pattern of activity generated in the first layer. The two elements are inspired by mechanisms deployed by the early visual system. The retina generates spontaneous activity waves that tile the light-insensitive retina; the activity waves serve as input signals to wire the retina to higher visual areas in the brain [Wong, 1999; Antón-Bolaños et al., 2019].

3.3.1. Spontaneous spatio-temporal wave generator

The first layer of the network can serve as a noise-driven spatiotemporal wave generator when (i) its constituent sensor-nodes are modeled via an appropriate dynamical system and (ii) when these nodes are connected in a suitable topology. In this paper, we model each sensor node using the classic Izikhevich neuron model[1] [Izhikevich, 2003] (dynamical system model), while the input layer topology is that of local-excitation and global-inhibition, a motif that is ubiquitous across various biological systems [Kutscher, Devreotes, and Iglesias, 2004; Xiong et al., 2010]. A minimal dynamical systems model coupled with the local-excitation and global-inhibition motif has been analytically examined in the supplemental materials to demonstrate that these key ingredients are *sufficient* to serve as a spatio-temporal wave generator. The **Izhikevich model** captures the activity of every sensor node ($v_i(t)$) through time, the noisy behavior of individual nodes (through $\eta_i(t)$) and accounts for interactions between nodes defined by a synaptic adjacency matrix ($S_{i,j}$). The Izhikevich model

[1] In section 3.6, we show that the self-organization is possible in a LIF (Linear Integrate and Fire) neuron model as well.

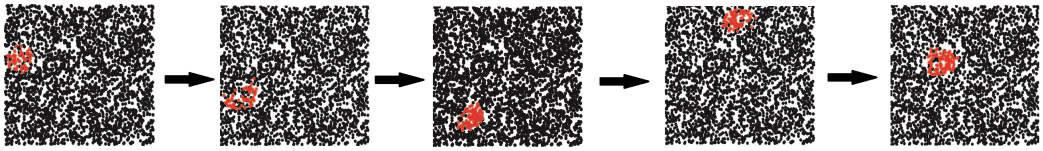


Figure 3.3: **Emergent spatiotemporal waves tile the first layer.** The red-nodes indicate active-nodes (firing), black nodes refer to silent nodes and the arrows denote the direction of time.

equations are elaborated in Box-1. The **input layer topology** (local excitation, global inhibition) is defined by the synaptic adjacency matrix ($S_{i,j}$). Every node in the first layer makes excitatory connections with nodes within a defined local excitation radius. $S_{i,j} = 5$, when distance between nodes i and j are within the defined excitation radius of 2 units; $d_{ij} \leq 2$. Each node has decaying inhibitory connections with other nodes present above a defined global inhibition radius ($S_{i,j}$ = -2 exp(-d_{ij}/10), when distance between nodes i and j are above a defined inhibition radius of 4 units; $d_{ij} \geq 4$) (see supporting information).

On implementing a model of the resulting dynamical system, we observe the emergence of spontaneous spatiotemporal waves that tile the first layer for specific parameter regimes (see Figure 3.3 and videos in supplemental materials).

Dynamical model for input-sensor nodes in the lower layer (layer-I):

$$\frac{\mathrm{d}v_i}{\mathrm{d}t} = 0.04v_i^2 + 5v_i + 140 - u_i + \sum_{j=1}^{N} S_{i,j}\mathcal{H}(v_j - 30) + \eta_i(t) \tag{3.1}$$

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = a_i(b_i v_i - u_i) \tag{3.2}$$

with the auxiliary after-spike reset:

$$v_i(t) > 30, \text{then} : \begin{cases} v_i(t + \Delta t) = c_i \\ u_i(t + \Delta t) = u_i(t) + d_i \end{cases}$$

where: (1) v_i is the activity of sensor node i; (2) u_i captures the recovery of sensor node i; (3) $S_{i,j}$ is the connection weight between sensor-nodes i and j; (4) N is the number of sensor-nodes in layer-I; (5) Parameters a_i and b_i are set to 0.02 and 0.2 respectively, while c_i and d_i are sampled from the distributions $\mathcal{U}(-65, -50)$ and $\mathcal{U}(2, 8)$ respectively. Once set for every node, they remain constant during the process of self-organization. The initial values for $v_i(0)$ and $u_i(0)$ are set to -65 and -13 respectively for all nodes. These values are taken from Izhikevich's neuron model [Izhikevich, 2003]; (6) $\eta_i(t)$ models the noisy behavior of every node i in the system, where $< \eta_i(t)\eta_j(t') > = \sigma^2\ \delta_{\mathrm{i,j}}\delta(\mathrm{t} - \mathrm{t}')$. Here, $\delta_{\mathrm{i,j}}$, $\delta(\mathrm{t} - \mathrm{t}')$ are Kronecker-delta and Dirac-delta functions respectively, and $\sigma^2 = 9$; (7) $\mathcal{H}$ is the unit step function:

$$\mathcal{H}(v_i - 30) = \begin{cases} 1, & v_i \geq 30 \\ 0, & v_i < 30. \end{cases}$$

3.3.2. Local learning rule

Having constructed a spontaneous spatiotemporal wave generator in layer-I, we implement a local learning rule in layer-II that can learn the activity wave pattern in the first layer and modify its inter-layer connections to generate a pooling architecture. Many neuron inspired learning rules can learn a sparse code from a set of input examples [Olshausen and Field, 1996]. Here, we model processing units as rectified linear units (ReLU) and implement a modified Hebbian rule for tuning the inter-layer weights to achieve the same. Individual ReLU units compete with one another in a winner take all fashion.

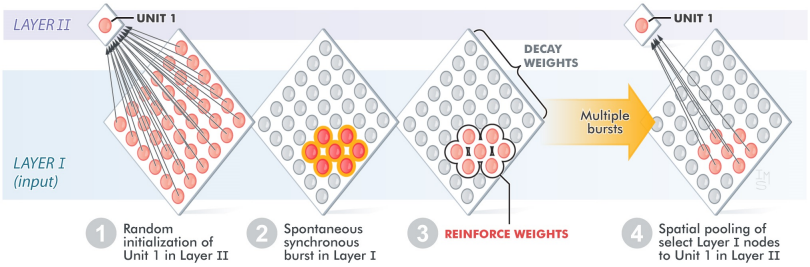


Figure 3.4: **Spiking neural network learning rule.**

Initially, every processing unit in the second layer is connected to all input-sensor nodes in the first layer. As the emergent activity wave tiles the first layer, at most a single processing unit in the second layer is activated due to the winner-take-all competition. The weights connecting the activated unit in the second layer to the input-sensor nodes in the first layer are updated by the modified Hebbian rule (Box-2). Weights connecting active input-sensor nodes and activated processing units are reinforced while weights connecting inactive input-sensor nodes and activated processing units decay (cells that fire together, wire together). Inter-layer weights are updated continuously throughout the self-organization process, ultimately resulting in the pooling architecture (See Fig. 3.4 and supplemental materials).

Modifying inter-layer weights

$$w_{i,j}(t+1) = \begin{cases} w_{i,j}(t) + \eta_{learn}\mathcal{H}(v_i(t) - 30)y_j(t+1) & y_j(t+1) > 0 \\ w_{i,j}(t) & \text{otherwise} \end{cases}$$

where: (1) $w_{i,j}(t)$ is the weight of connection between sensor-node i and processing unit j at time 't' (inter-layer connection); (2) η_{learn} is the learning rate; (3) $\mathcal{H}(v_i(t) - 30)$ is the activity of sensor node i at time 't'; and (4) $y_j(t)$ is the activation of processing unit j at time 't.'

Once all the weights $w_{i,j}(t+1)$ have been evaluated for a processing unit j, they are mean-normalized to prevent a weight blow-up. Mean normalization ensures that the mean strength of weights for processing unit j remains constant during the self-organization process.

Having coupled the spontaneous spatiotemporal wave generator and the local learning rule, we observe that an initially fully connected two-layer network (Fig. 3.5a) becomes a pooling architecture, wherein input-sensor nodes that are in close proximity to each other in the first layer have a very high probability of connecting to the same processing unit in the second layer (Fig. 3.5b & 3.5c). More than 95% of the sensor-nodes in layer-I connect to processing units in layer-II (higher layer) through well-defined pools, ensuring that spatial patches of nodes connected to units in layer-II tile the input layer (Fig. 3.5d). Tiling the input layer ensures that most sensor nodes have an established means of sending information to higher layers after the self-organization of the pooling layer.

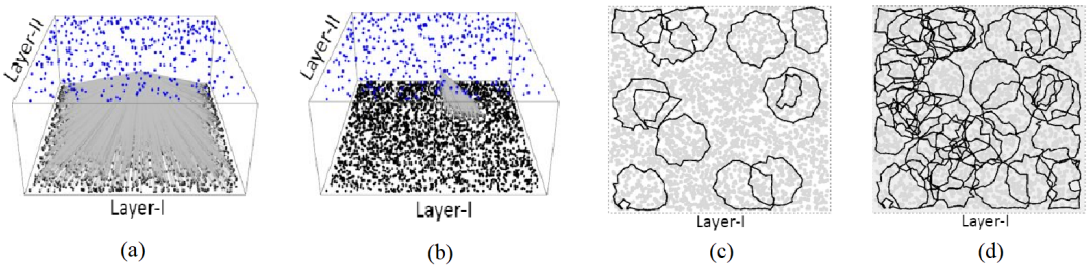


Figure 3.5: **Self-organization of Pooling layers.** (A) The initial configuration, wherein all nodes in the lower layer are connected to every unit in the higher layer. (B) After the self-organization process, a pooling architecture emerges, wherein every unit in layer-II is connected to a spatial patch of nodes in layer-I. (A,B) Here, connections from nodes in layer-I to a single unit in layer-II (higher layer) are shown. (C) Each contour represents a spatial patch of nodes in layer-I connected to a single unit in layer-II. (D) More than 95% of the nodes in layer-I are connected to units in the layer-II through well-defined pools, as the spatial patches tile layer-I completely.

Section 3.4

Minimal model for observing emergent spatiotemporal waves

In this section, we provide an analytical solution for the emergence of a spatiotemporal wave through noisy interactions between constituent nodes in the same layer. As stated in the previous section, the key ingredients for having a layer of nodes function as a spatiotemporal wave generator are:

- Each sensor-node should be modeled as a dynamical systems model,
- Sensor-nodes should be connected in a suitable topology (for e.g. local excitation ($r_e < 2$ and global inhibition ($r_i > 4$).

On modeling all nodes in the system using a simple set of ordinary differential equations (ODE's), we highlight the conditions required for observing a stationary bump in a network of spiking sensor-nodes and to observe instability of the stationary bump resulting in a traveling wave.

3.4.1. Sensor-nodes arranged in a line

We choose a configuration where N sensor-nodes are randomly arranged in a line (as shown in Fig. 3.6).

Figure 3.6: Sensor nodes arranged in a line.

The activity of N sensor nodes, arranged in a line as in Fig. 3.6, are modeled using a minimal ODE model as described below:

$$\tau_d \frac{\mathrm{d}x(u_i,t)}{\mathrm{d}t} = -x(u_i,t) + \sum_{u_j \in \mathcal{U}} S(u_i,u_j)\mathcal{F}(x(u_j,t)) \quad \forall i \in 1,...,N. \tag{3.3}$$

Here, u_i represents the position of nodes on a line; $x(u_i,t)$ defines the activity of sensor node positioned at u_i at time t; S_{u_i,u_j} is the strength of connection between nodes positioned at u_i and u_j; τ_d controls the rate of decay of activity; $\mathcal{U}$ is the set of all sensor nodes in the system (u_1,u_2,...,u_N) for N sensor nodes; and $\mathcal{F}$ is the non-linear function required to convert activity of nodes to spiking activity. Here, $\mathcal{F}$ is the heaviside function with a step transition at 0.

Each sensor-node has the same topology of connections, ie fixed strength of positive connections between nodes within a radius r_e, no connections from a radius r_e to r_i, and decaying inhibition above a radius r_i. This is depicted in Fig. 3.7

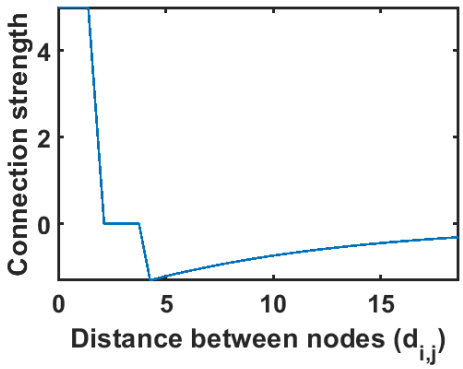


Figure 3.7: Strength of connections between sensor-nodes.

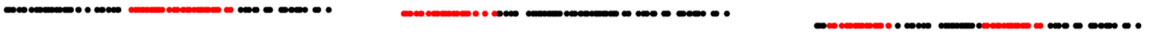
Fixed point analysis. We determine the stable activity states of nodes placed in a line by a fixed point analysis, similar to what Amari developed in [Amari, 1977] for the case when there are infinite nodes.

$$x(u_i) = \sum_{u_j \in \mathcal{U}} S(u_i, u_j)\mathcal{F}(x(u_j)) \quad \forall i \in 1, ..., N. \tag{3.4}$$

On solving this system of non-linear equations simultaneously, we get a fixed point, i.e. a vector $x^* \in \mathcal{R}^N$, corresponding to the activity of N sensor nodes positioned at (u_1,u_2,...,u_N). To assess their spiking from the activity of sensor-nodes, we have

$$s_i = \mathcal{F}(x(u_i)) \quad \forall i \in 1, ..., N \tag{3.5}$$

As the weight matrix (S_{u_i,u_j}) used incorporates the local excitation ($r_e < 2$) and global inhibition ($r_i > 4$) (Fig. 3.7), we get solutions with a single bump of activity (Fig. 3.8a, 3.8b), two bumps of activity (Fig. 3.8c) or a state when all nodes are active.



(a) (b) (c)

Figure 3.8: **Fixed points:** Multiple fixed points are obtained by solving N non-linear equations simultaneously. Some of the solutions obtained are: (A) a single bump at the center, (B) a single bump at one of the edges and (C) two bumps of activity.

Stability of fixed points. To assess the stability of these fixed points, we evaluate the eigenvalues of the Jacobian for this system of differential equations. As there are N differential equations, the Jacobian ($\mathbb{J}$) is an NxN matrix.

$$\begin{aligned}
\frac{\mathrm{d}x(u_i,t)}{\mathrm{d}t} &= \frac{-x(u_i,t)}{\tau_d} + \sum_{u_j \in \mathcal{U}} \frac{S(u_i,u_j)\mathcal{F}(x(u_j))}{\tau_d} \\
\frac{\mathrm{d}x(u_i,t)}{\mathrm{d}t} &= f_i(u_1,u_2,...,u_N) \\
f_i(u_1,u_2,...,u_N) &= \frac{-x(u_i)}{\tau_d} + \sum_{u_j \in \mathcal{U}} \frac{S(u_i,u_j)\mathcal{F}(x(u_j))}{\tau_d} \\
\mathbb{J}(i,j) &= \frac{\partial f_i(u_1,u_2,...,u_N)}{\partial x(u_j)}.
\end{aligned} \tag{3.6}$$

On evaluating the Jacobian ($\mathbb{J}$) at the fixed points obtained (x^*), we get:

$$\begin{aligned}
\mathbb{J}(i,i) &= \frac{\partial f_i}{\partial x(u_i)} \\
\mathbb{J}(i,i) &= \frac{-1}{\tau_d} \\
\mathbb{J}(i,j) &= S(u_i,u_j)\mathcal{F}^{'}(x(u_j))\frac{\partial x(u_j)}{x(u_j)} \\
\mathbb{J}(i,j) &= S(u_i,u_j)\delta(x(u_j)) \\
\mathbb{J}(i,j) &= 0 \ \ \forall x(u_j) \neq 0.
\end{aligned} \tag{3.7}$$

Here, $\mathcal{F}$ is the Heaviside function and its derivative is the dirac-delta(δ); where, $\delta(x) = 0$, for $x \neq 0$ and $\delta(x) = \infty$ for $x = 0$.

For a fixed point, where $x^*(u_k) \neq 0$, $\forall k \in 1,...,N$, the Jacobian is a diagonal matrix with $\frac{-1}{\tau_d}$ in its diagonals. This implies that the eigenvalues of the Jacobian are $\frac{-1}{\tau_d}$ ($\tau_d > 0$), which assures that the fixed point $x* \in \mathcal{R}^N$ is a stable fixed point[2].

Destabilizing the fixed point. The addition of gaussian noise (whose amplitude is appropriately scaled) to the ODE's described earlier, we can effectively destabilize the fixed point, resulting in a traveling wave. The equations with the addition of a noise term are:

[2]If all the eigenvalues are negative, the fixed point is stable, however even if a single eigenvalue is positive, the fixed point is unstable.

$$\tau_d \frac{\mathrm{d}x(u_i,t)}{\mathrm{d}t} = -x(u_i,t) + \sum_{u_j \in \mathcal{U}} S(u_i,u_j)\mathcal{F}(x(u_j,t)) + \eta_i(t) \quad \forall i \in 1,...,N. \tag{3.8}$$

Here, $\eta_i(t)$ models the noisy behavior of every node i in the system, where $< \eta_i(t)\eta_j(t') > = \sigma^2\ \delta_{\mathrm{i,j}}\delta(\mathrm{t}-\mathrm{t}')$. Here, $\delta_{\mathrm{i,j}}$, $\delta(\mathrm{t}-\mathrm{t}')$ are Kronecker-delta and Dirac-delta functions, respectively, and σ^2 captures the magnitude of noise added to the system.

The network of sensor nodes is robust to a small amplitude of noise ($\sigma^2 \in (0,4)$), while a larger amplitude of noise ($\sigma^2>5$) can destabilize the bump, forcing the system to transition to another bump in its local vicinity. Continuous addition of high amplitudes of noise forces the bump to move around in the form of traveling waves. The behavior is consistent with the linear stability analysis because noise can push the dynamical system beyond the envelop of stability for a given fixed point solution.

3.4.2. Sensor nodes arranged in a 2-dimensional square

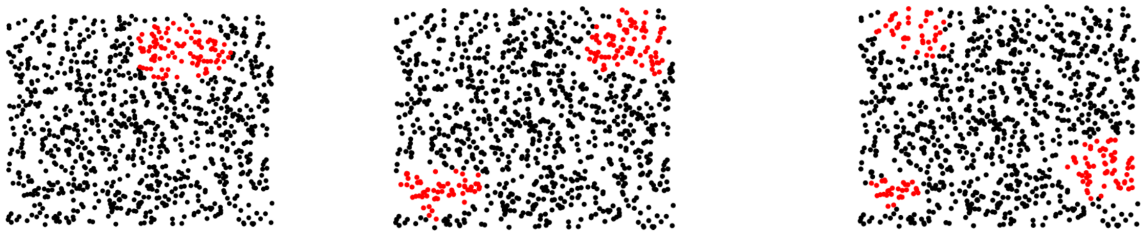
In this section, we arrange N sensor nodes arbitrarily on a 2-dimensional square as shown in Fig. 3.9, with the same local structure (local excitation and global inhibition).

The activity of these sensor nodes are modeled using the minimal ODE model described earlier (in equation-3.3).

Figure 3.9: **Sensor nodes placed arbitrarily on a square plane.**

We obtain the fixed points ($\mathrm{x}^* \in \mathcal{R}^N$), by solving N simultaneous non-linear equations using BBsolve [Varadhan, Gilbert, et al., 2009] package in R-language. We notice that the fixed point solutions have a variable number of activity bumps in the

2D plane as shown in Fig. 3.10a, 3.10b & 3.10c.



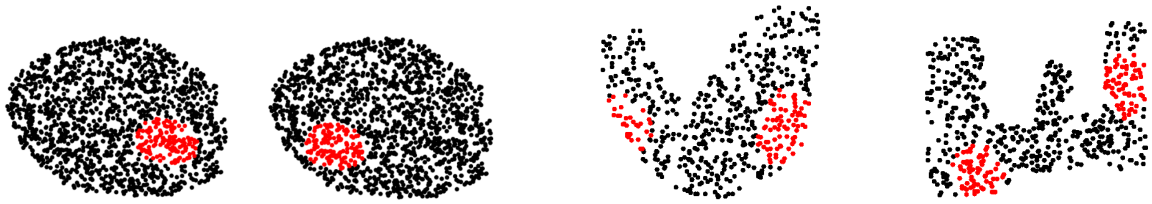
(a) (b) (c)

Figure 3.10: **Stable Fixed points:** Multiple fixed points are obtained by solving N non-linear equations simultaneously. Some of the solutions obtained are: (A) a single bump, (B) two bumps and (C) three bumps of activity.

3.4.3. Sensor nodes arranged in a 2-dimensional sheet with arbitrary geometry

In this section, we arrange sensor nodes on a 2D sheet in any arbitrary geometry as shown in Fig. 3.11. Although the macroscopic geometry of the sheet changes, the local structure of sensor nodes in conserved (i.e. local excitation and global inhibition).

The fixed points are evaluated by simultaneously solving the non-linear system of equations. We notice that the bumps are stable fixed points even when sensor nodes are placed on a 2-dim sheet of arbitrary geometry.



(a) (b) (c) (d)

Figure 3.11: **Stable Fixed points:** Multiple fixed points are obtained by solving N non-linear equations simultaneously. Some of the solutions obtained are: (A,B) a single bump for a circular geometry (C,D) two bumps of activity for arbitrary geometry.

Section 3.5

Modular SNN Tool-kit: Dynamical systems framework

In the previous two sections, I have laid down the minimum components (namely, spontaneous spatiotemporal waves and local learning rules) for self-organizing a two layered neural network. In order to build a scalable multi-layer SNN, we propose a dynamical systems framework for the self-organization algorithm. It utilizes the following key concepts of (i) emergent spatio-temporal waves of firing neurons, (ii) dynamic learning rules for updating inter-layer weights and (iii) non-linear activation and input/output competition rules between layers to build a modular spiking sub-structure. The modular spiking sub-structure can be stacked to form multi-layered SNNs with an arbitrary number of layers that self-organize into a wide variety of connectivity architectures. The following sections describe the tool-kit required to build a single module that can be seamlessly stacked to self-organize multi-layer SNN architectures. We describe our framework by discussing the SNN model that generates waves and the learning/competition rules that achieve inter-layer connectivity.

3.5.1. Governing equations of "neuronal waves"

The essential building block for SNNs is a spiking neuron model that describes the state of every single neuron over time, often represented by a dynamical system. Here, we choose a modified version of the popular Leaky-Integrate-and-Fire (LIF)[3] model with an additional adjacency matrix term and input term (from preceding layers), coupled with a dynamical threshold equation. The vectorized governing equations for each layer l reads

[3]As LIF models are widely used on neuromorphic chips, we shifted our scale-up procedure from Izhikevich to the LIF neuron model.

$$\begin{aligned}\frac{d}{dt}\boldsymbol{v} &= -\frac{1}{\tau_v}\boldsymbol{v} + \mathbf{S}\mathcal{H}(\boldsymbol{v}-\boldsymbol{\theta}) + \mathbf{S}^x\boldsymbol{x} \\ \frac{d}{dt}\boldsymbol{\theta} &= -\frac{1}{\tau_\theta}(v^{th}-\boldsymbol{\theta}) \odot (1-\mathcal{H}(\boldsymbol{v}-\boldsymbol{\theta})) + \theta^{+}\mathcal{H}(\boldsymbol{v}-\boldsymbol{\theta})\end{aligned} \tag{3.9}$$

where v is the voltage, θ is the variable firing threshold, $\boldsymbol{x}$ is the input signal to this layer, $\mathcal{H}$ is the (element-wise) heavy-side function and $\odot$ denotes the Hadamard product. $\mathbf{S}$ is the intra-layer adjacency matrix and $\mathbf{S}^x$ is the spike input matrix. All vectors and matrices are elements of $\mathbb{R}^{n_l}$ and $\mathbb{R}^{n_l \times n_l}$, respectively, where n_l is the number of neurons in layer l.

A neuron i fires a spike when its voltage v_i exceeds its threshold θ_i. After firing, the neuron's voltage is reset to v^{reset}. The dynamic threshold equation for θ is governed by a homoeostasis mechanism to ensure that no neuron can spike excessively. Concretely, it increases θ by a rate θ^+ whenever a neuron is spiking, until θ exceeds v and the neuron fires no more. It then decays θ exponentially to a default threshold v^{th}. All additional hyper-parameters are summarized in the appendix.

$\mathbf{S} \in \mathbb{R}^{n_l \times n_l}$ encodes the spatial-connectivity of neurons within the layer (that can have arbitrary geometry [Raghavan and Thomson, 2019]) and is biologically inspired [Kutscher, Devreotes, and Iglesias, 2004; Xiong et al., 2010]. Authors in [Laing and Chow, 2001] have since used the intra-layer connectivity to prove the existence of spatio-temporal wave states in both 1D and 2D geometries of connected spiking neurons. In our multi-layer SNN, $\mathbf{S}\mathcal{H}(\boldsymbol{v}-\boldsymbol{\theta})$ serves as a back-coupling term, crucial for the development of coherent wave dynamics in subsequent layers. The optional spike-input matrix $\mathbf{S}^x \in \mathbb{R}^{n_l \times n_l}$ can be used to further control the input received from preceding layers. We encode the geometry of the layer and an isotropic kernel with a tunable excitation and inhibition radius and amplitude factors into $\mathbf{S}$. The kernel leads to positive intra-layer neuronal connectivity inside the excitation radius r^i and decaying negative connections outside the inhibition radius r^o. Concretely, the adjacency matrix with kernel is given by

$$S_{i,j} = \begin{cases} a^i D_{i,j}, & D_{i,j} < r^i \\ -a^o e^{(-D_{i,j}/10)}, & D_{i,j} > r^o \end{cases} \tag{3.10}$$

where $D_{i,j} \in \mathbb{R}^{n_l \times n_l}$ is the matrix of spatial distances between each neuron and a^i/a^o are the excitation and inhibition amplitude factors. One can now vary the kernel radii and other hyper-parameters to control the emergent wave properties and obtain an array of wave phenomena with interesting shapes and dynamics. A few exemplary wave regimes are depicted in Fig. 3.17(B).

3.5.2. Learning rules

Having constructed a spontaneous spatio-temporal wave generator across multiple layers in the previous section, we implement a local STDP learning rule to update inter-layer connectivity based on the patterns of the emergent waves, in order to self-organize SNNs into a wide variety of architectures. STDP potentiates connections between neurons that spike within a short interval to each other and provides lower updates for those neurons that have distant spike-times. As a simple STDP rule, we use the Hebbian rule to only link the synchronous pre- and post-synaptic firings of neurons for the dynamic update of weights between the two connected layers. We note that there are many types of sophisticated STDP rules such as additive STDP or triplet STDP [Markram, Gerstner, and Sjöström, 2011; Bichler et al., 2011], however, we use a rather simple rule to only emphasize the effectiveness of our contribution. The learning rule can be integrated into our dynamical system as the dynamical matrix equation:

$$\frac{d}{dt}\mathbf{W}^{(l_1)} = \eta(\boldsymbol{y}^{(l_1)} \otimes \boldsymbol{y}^{(l_2)}) \tag{3.11}$$

where η is the learning rate, $\boldsymbol{y}^{(l_1)} \in \mathbb{R}^{n_{l_1}}$ and $\boldsymbol{y}^{(l_2)} \in \mathbb{R}^{n_{l_2}}$ denote the spiking output signals of the two layers that $\mathbf{W}^{(l_1)} \in \mathbb{R}^{n_{l_1} \times n_{l_2}}$ connects and $\otimes$ is the outer product

of the two vectors. The specific variables coupled in Eq. 3.11 can be customized to achieve various desired connectivity architectures.

3.5.3. Competition rules

In addition to the learning rules, we can also introduce various "competition rules" on the layer inputs and outputs to further localize connections with different strengths, to form pooling architectures. For instance, by coupling the spiking outputs in Eq. 3.11 with $\boldsymbol{y}^{(l_2)}$ filtered by a "winner-take-all" competition rule, one can enforce the formation of pools from l_1 to the maximum spiking neuron in l_2. An input spike signal $\boldsymbol{x}$ can similarly be filtered. The winner-take-all competition rule for a vector $\boldsymbol{x}$ reads:

$$f^{\mathcal{C}}(\boldsymbol{x}) : \begin{cases} x_i = 0, & \forall x_i < \max(x) \\ x_i = \max(x), & \text{otherwise.} \end{cases} \tag{3.12}$$

The competition rule f^C works on each neuron i within a layer l. From Eq. 3.12, many variations like "k-best-performers" and other competition rules can be derived and applied to achieve pools of different shapes and weightings throughout the layers.

3.5.4. Multi-layer SNN learning algorithm

With the three building blocks (Eq. 3.9, Eq. 3.11, Eq. 3.12) established, the algorithmic flow of an input signal $\boldsymbol{x}^{(1)}$ of a layer ($l_1 = 1$) to the input $\boldsymbol{x}^{(2)}$ of the next layer ($l_2 = 2$) is elaborated in algorithm-4. In algorithm-4, LIF$(\cdot)$ stands shorthand for a time-integration pass through Eq. 3.9 and $\mathcal{H}^{(1)}_{v,\theta}$ is the respective spike vector. Furthermore, $f^{\mathcal{C}^{(1)}_y}$ and $f^{\mathcal{C}^{(2)}_x}$ are the (optional) competition rules for the output of l_1 and input to l_2 respectively and $g(\cdot)$ denotes the activation function of the layer, which is a rectified linear unit (ReLU) in our case. As one can see, the entire algorithm is model-able as a large dynamical system —coupling the wave dynamics equations of

Algorithm 4 Multi-layer SNN Dynamical System

Input : Signal $\boldsymbol{x}^{(1)}(t)$ as input to input layer $l = 1$.
Output: Weights $\mathbf{W}^{(l)}(t)$ & spiking outputs $\boldsymbol{y}^{(l)}(t)$ for all layers $l \geq 1$.

for $t = 1...N_t$ *in* Δt *time-steps* **do**
  for $l = 1...N_l$ *in layers* **do**
    $\mathcal{H}_{v,\theta}^{(l)}$ $\text{LIF}^{(l)}(\boldsymbol{x}^{(l)}, \Delta t)$ *integrate input with LIF by* Δt
    $\boldsymbol{y}^{(l)}$ $f^{\mathcal{C}_y^{(l)}}(\mathcal{H}_{v,\theta}^{(l)})$ *apply output competition rule to spikes*
    if $l \geq 2$ **then**
      $\mathbf{W}^{(l-1)}$ $\text{LR}^{(l-1)}(\boldsymbol{y}^{(l-1)}, \boldsymbol{y}^{(l)}, \Delta t)$ integrate learning rule of preceding weights
    end
    $\boldsymbol{z}^{(l+1)}$ $\mathbf{W}^{(l)}\boldsymbol{y}^{(l)}$ multiply local weights to output signal
    $\boldsymbol{a}^{(l+1)}$ $g(\boldsymbol{z}^{(l+1)})$ apply activation function
    $\boldsymbol{x}^{(l+1)}$ $f^{\mathcal{C}_x^{(l+1)}}(\boldsymbol{a}^{(l+1)})$ apply input competition rule to obtain signal for next layer
  end
end

individual layers with the weight dynamics equations given by STDP learning rules between the layers. We integrate all equations in time at the same time-level by using a Runge-Kutta-4 time-stepping scheme for numerical integration.

Section 3.6

Self-organizing multi-layer spiking neural networks

The modular tool-kit introduced in the previous section enables the efficient, autonomous self-organization of large multi-layer SNNs. The key ingredients required for self-organization are (i) traveling waves that emerge simultaneously across multiple layers and (ii) a dynamic learning rule that tunes the connectivity between any two layers based on the properties of the waves tiling the layers. We demonstrate the entire self-organization process in Fig. 3.12 (moving from left to right). The two major components of the self-organization process are elaborated in the following

subsections.

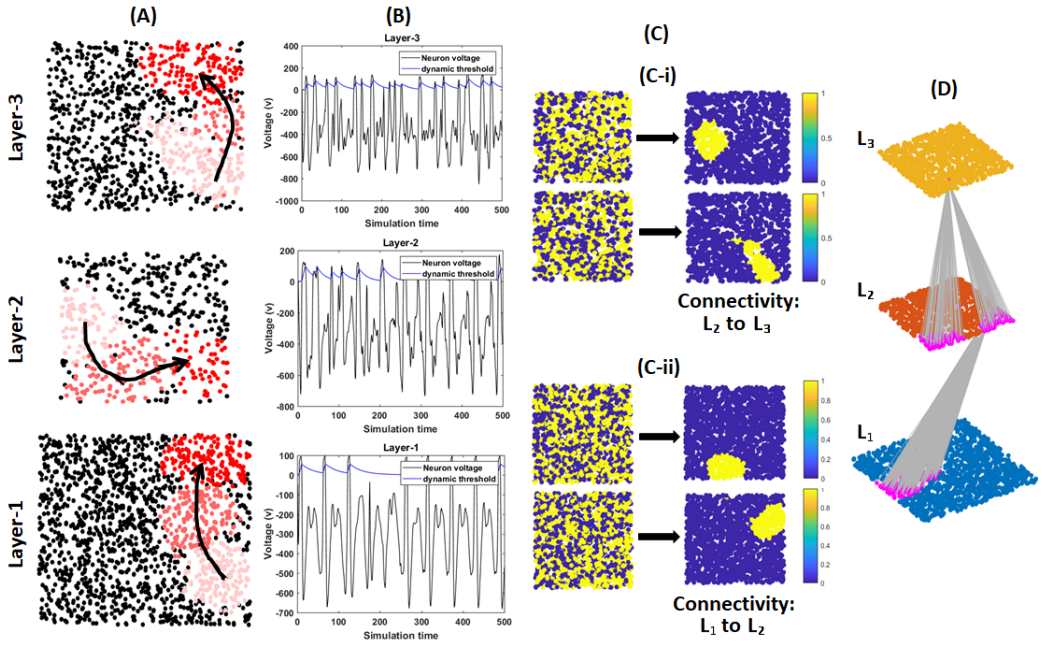


Figure 3.12: **Self-organizing multi-layer spiking neural networks** (A) Emergent spatio-temporal waves in L_1 trigger neuronal waves in higher layers (L_2, L_3). Black nodes indicate the neuron positions within a layer and shades of red depict firing nodes. The lighter red represents nodes that fired at an earlier time-point. Lighter red to dark red captures the motion of the waves on each layer. (B) Tracking the voltage v of a single neuron in each layer over time. The neuron 'fires' when the v crosses its dynamic threshold (blue line). (C) Self-organization process transforms a randomly wired inter-layer connectivity (left of the arrow) to a pooling architecture (right), wherein units in higher layers (L_2, L_3) are connected to a spatial patch of nodes in its preceding layer. Each subplot displays the connectivity of a single unit in a higher layer to all units in the preceding layer. Yellow/blue represent regions with/without presence of connections. Connectivity of 4 units each in L_3 and L_2 are depicted in C-i and C-ii, respectively. (D) 3D rendering of the final self-organized architecture.

3.6.1. Emergent activity waves in multiple layers

Stochastic communication between spiking neurons in layer-1 arranged in a local-excitation, global inhibition connectivity leads to the emergence of spontaneous traveling activity waves within the layer. The waves in layer-1 trigger waves in layer-2 that subsequently initiates waves in layer-3. The traveling waves across the 3 layers are depicted in Fig. 3.12A. We observe that the algorithm enables the motion of

waves in higher layers without the need for a constant stimulation from the lower layers. In other words, the wave activity in higher layers, once triggered, can 'stay alive' even if there is no spiking activity in the lower layers. Another key property of the traveling waves in the higher layers is that they have their own autonomy/'curiosity' to explore different regions within the layer. The level of 'curiosity' is dependent on the input from the preceeding layer and the strength of intra-layer connectivity. This forces the wave to not arbitrarily stray away from the source of the input-signal.

We also point out that waves in any layer are observed primarily due to the spiking dynamics of individual neurons. In Fig. 3.12B, we show the voltage trace of one neuron within each layer along with its spiking threshold. A neuron fires only when its voltage surpasses the spiking threshold, and the spiking frequency within each layer governs the dynamics of the activity wave.

3.6.2. Local learning rules leads to self-organization

The activity waves generated in each layer serve as a signal to modify their inter-layer weights. Along with the 'signal,' we need local learning rules to update inter-layer connections. Here, we use Hebbian-based STDP rules (described in section 3.5.2) coupled with competition rules (described in section 3.5.3) to update inter-layer weights. In Fig. 3.12, we depict the simultaneous activity-wave driven self-organization across multiple layers. The connectivity between the layers go from a random configuration to pooling structures between the layers, guided by the dynamics of the activity wave. A final self-organized multi-layer spiking network is rendered in 3D in Fig. 3.12D.

Section 3.7

Features of the developmental algorithm

In this section, we show that spatiotemporal waves can emerge and travel over layers with arbitrary geometries and even in the presence of defective sensor-nodes. As the

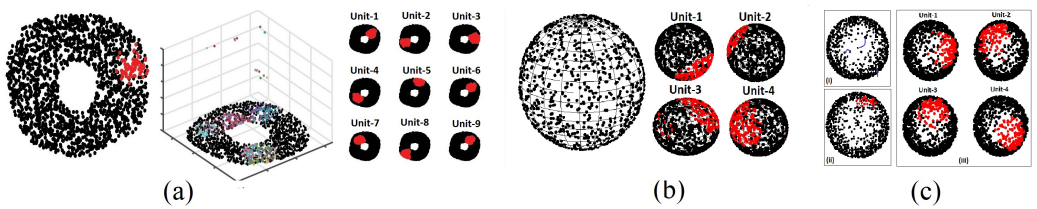
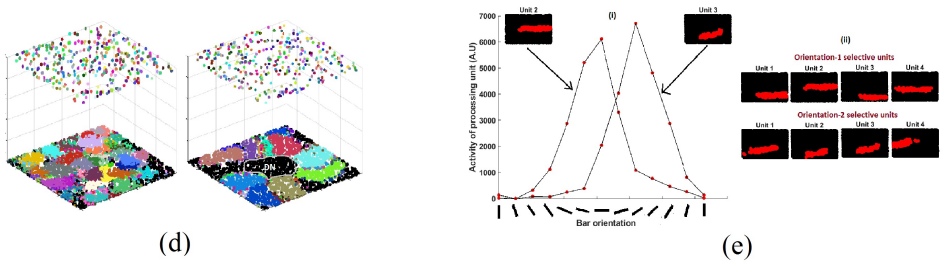


Figure 3.13: **Self-organization of pooling layers for arbitrary input-layer geometry.** (A) The left most image is a snapshot of the traveling wave as it traverses layer-I; Layer-I has sensor-nodes arranged in an annulus geometry; red nodes refer to firing nodes. On coupling the spatiotemporal wave in layer-I to a learning rule in layer-II, a pooling architecture emerges. The central image refers to the 3d visualization of the pooling architecture, while each subplot in the right-most image depicts the spatial patch of nodes in layer-I connected to a single processing unit in layer-II. (B) Self-organizing pooling layers on a sphere. (B-ii) Upstream units connect to spatial patches of nodes on the sphere. (C) Self-organizing networks on Poincare disks with a hyperbolic distribution of input sensor nodes (C-ii) Snapshot of a traveling bump. (C-iii) Receptive fields of units in layer-II.

local structure of sensor-node connectivity (local excitation and global inhibition) in the input layer in conserved over a broad range of macroscale geometries (Fig. 3.13a-c), we observe traveling activity waves on input layers with arbitrary geometries and in input-layers that have defects or holes. The coupling of the traveling activity wave in layer-I and a learning rule in layer-II results in the emergence of pooling architecture.

Furthermore, we demonstrate that the size and shape of the emergent spatiotemporal wave can be tuned by altering the topology of sensor-nodes in the layer. Coupling the emergent wave in layer-I with a learning rule in layer-II leads to localized receptive fields that tile the input layer.

Together, the wave and the learning rule endow the developmental algorithm with useful properties: (i) **Flexibility**: Spatial patches of sensor-nodes connected to units in layer-II can be established over arbitrary input-layer geometries. In Fig. 3.13a, we show that an emergent spatiotemporal wave on a torus-shaped input layer coupled with the local learning rule (section-3.3.2) in layer-II, results in a pooling architecture. We also show that the developmental algorithm can self-organize net-



(d) (e)

Figure 3.14: **Self-organization of pooling layers for arbitrary input-layer geometry.** (D) The figure on the left depicts a self-organized pooling layer when all input nodes are functioning. Once these inter-layer connections are established, a small subset of nodes are damaged to assess if the pooling architecture can robustly re-form. The set of nodes within the grey boundary, titled 'DN', are defective nodes. The figure on the right corresponds to pooling layers that have adapted to the defects in the input layer, hence not receiving any input from the defective nodes.(E) (E-i) Tuning curve shows that units in layer-2 have a preferred orientation. (E-ii) Oriented receptive fields of units in layer-II.

works on arbitrary curved surfaces (Fig. 3.13b). Flexibility to form pooling layers on arbitrary input-layer geometries is useful for processing data acquired from unconventional sensors, like charge-coupled devices that mimic the retina [Sandini et al., 1993]. The ability to self-organize pooling layers on curved surfaces makes it extremely useful for spherical image analysis. Spherical images acquired by omnidirectional cameras [Scaramuzza, 2014] placed on drones are becoming increasingly ubiquitous, and their analysis necessitates neural networks that can tile 3-dimensional surfaces. (ii) **Robustness**: Spatial patches of sensor-nodes connected to units in layer-II can be established in the presence of defective sensor nodes in layer-I. As shown in Fig. 3.14d, we initially self-organize a pooling architecture for a fully functioning set of sensor-nodes in the input-layer. To test robustness, we ablate a few sensor-nodes in the input-layer (captioned 'DN'). Following this perturbation, we observe that the pooling architecture re-emerges, wherein spatial-pools of sensor-nodes, barring the damaged ones, re-form and connect to units in layer-II. (iii) **Reconfigurable**: The size and shape of spatial pools generated can be modulated by tuning the structure of the emergent traveling wave (Fig. 3.15f & 3.15g). In Fig. 3.15h, we show that the size of spatial-pools can be altered in a controlled manner by modifying the topology

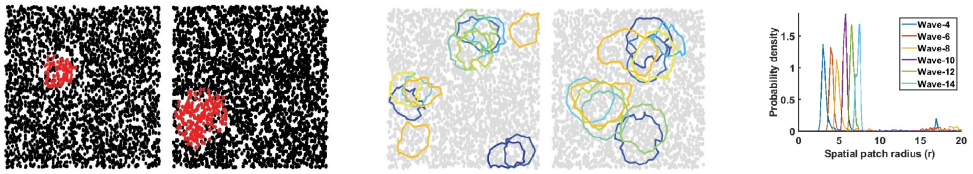
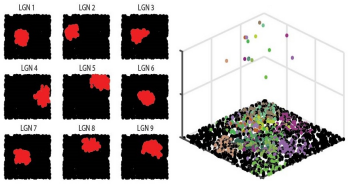


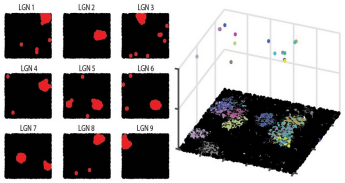
Figure 3.15: **Pooling layers are reconfigurable.** (F) By altering layer-I topology (excitation/inhibition radii), we can tune the size of the emergent spatial wave. The size of the wave is 6 A.U (left) and 10 A.U (right). (G) Altering the size of the emergent spatial wave tunes the emergent pooling architecture. The size of the pools obtained are 4 A.U (left), obtained from a wave-size of 6 A.U and a pool-size of 7 A.U (right), obtained from a wave-size of 10 A.U. (H) A large set of spatial-pools are generated for every size-configuration of the emergent wave. The distribution of spatial-pool sizes over all pools generated by a specific wave-size are captured by a kernel-smoothed histogram. Wave-4 in the legend corresponds to a histogram of pool-sizes generated by an emergent wave of size 4 A.U (blue line). We observe that spatial patches that emerge for every configuration of the wave have a tightly regulated size.

of layer-I nodes. Wave-x in the legend corresponds to an emergent wave generated in layer-I when every node in layer-I makes excitatory connections to other nodes in its 2 unit radius and inhibitory connections to every node above x unit radius. This topological change alters the properties of the emergent wave, subsequently changing the resultant spatial-pool size. The histograms corresponding to these legends capture the distribution of spatial-pool sizes over all pools generated by a given wave-x. The histogram also highlights that the size of emergent spatial-pools are tightly regulated for every wave-configuration. (iv) **Scaling to large layers**: We also show that the self-organization of pooling architectures can be scaled to large input layers. Large layers are defined based on the number of sensor nodes in the layer. We observe that enforcing a spatial bias on the initial set of connections from units in layer-II to the nodes in the input layer, enables us to speed up the process of self-organization. Our simulations show that the self-organization of pooling layers can be scaled up to large layers (upto 50000 nodes) without being very expensive, as an increase in number of sensor-nodes results in multiple simultaneous waves tiling the input layer, effectively forming pooling architectures in parallel. (v) **Diverse multi-layer architectures**:

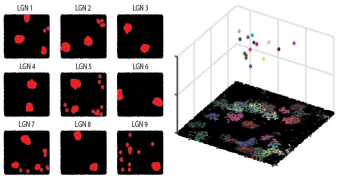
Finally, we demonstrate that designing the modular tool-kit in a dynamical systems framework allows us to tune the emergent wave dynamics on each layer, ultimately resulting in different self-organized architectures. The wave dynamics in each layer can be tuned by varying (i) excitation/inhibition connectivity (r^i, r^o) between neurons within every layer and (ii) by altering the time-constants and other hyper-parameters governing the spiking dynamics of neurons in each layer. In Fig. 3.17B, we portray a broad range of wave dynamics achievable on the layers of the network.



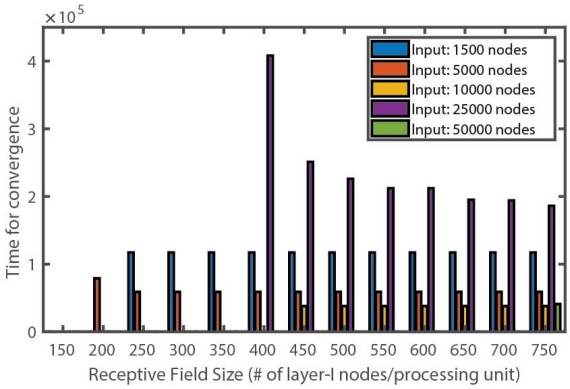
(a) Input layer: 1500 nodes



(b) Input layer: 5000 nodes



(c) Input layer: 10000 nodes



(d) Time complexity for self-organization of pooling layers

Figure 3.16: **Developmental algorithm scales efficiently to very large input layers:** (A) Layer-I has 1500 nodes and layer-II has 400 nodes. The emergent wave in layer-I results in a single traveling wave that tiles layer-I. (B) Layer-I has 5000 nodes and layer-II has 400 nodes. The emergent wave in layer-I results in a single traveling wave that tiles layer-I. (C) Layer-I has 10000 nodes and layer-II has 400 nodes. The emergent wave in layer-I results in a multiple traveling wave that tile layer-I simultaneously. This results in a single processing unit receiving pools from different regions. (D) The histogram captures the time taken for a pooling layer to form for variable number of input sensor nodes (1500, 5000, 10000, 25000 and 50000 nodes). With an increase in the number of sensor-nodes, the speed of self-organization increases as multiple waves tile the input layer simultaneously.

Along with varying wave dynamics, modifying the size and shape of waves across

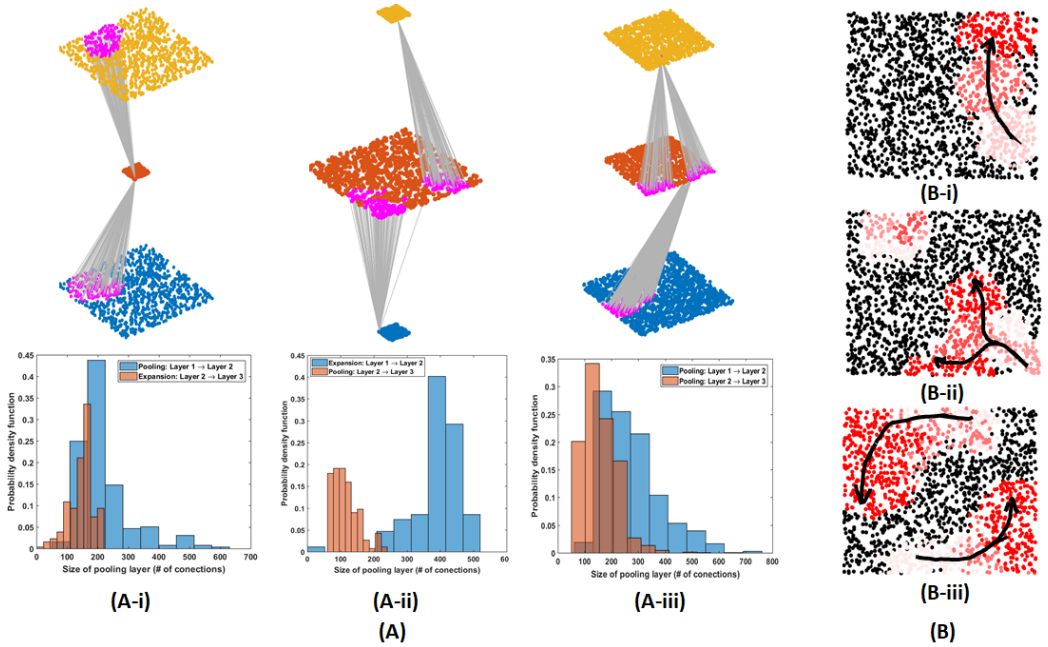


Figure 3.17: **Flexibility of the framework**. **(A)** *Self-organizing a variety of neural architectures*: (A-i) Pooling followed by expansion (autoencoder) (A-ii) Expansion followed by pooling (A-iii) Consecutive pooling structures. Histograms capture the sizes of emergent pooling and expansion structures in the self-organized network. **(B)** *Regimes of wave dynamics*: (B-i) Stable single wave (B-ii) Unstable splitting and merging waves (B-iii) Stable periodically rotating fluid-like wave.

different layers, and the number of nodes in each layer, we are able to self-organize a wide variety of multi-layer neural network architectures (Fig. 3.17). Here, we demonstrate efficient self-organization of three common neural architectures: (i) (Self-organized autoencoder) Pooling followed by expansion, (ii) Expansion followed by a pooling layer, (iii) Consecutive pooling operations (Self-organized retinotopic pooling structure). The histograms in Fig. 3.17 capture the size of the self-organized pooling and expansion structures between the layers. The size of a pooling structure from $L_1 \rightarrow L_2$ is the number of connections a single node in L_2 makes with nodes in L_1, while the size of the expansion structure from $L_2 \rightarrow L_3$ is the number of connections a single node in L_2 makes with nodes in L_3. As the pooling and expansion structures follow a sharp uni-modal distribution, we infer that our algorithm imposes a tight control over the size of the self-organized structures.

Section 3.8

Growth of a two-layered neural network from a single cell

As the developmental algorithm (introduced in section 3) is flexible to varying scaffold geometries and tolerant to malfunctioning nodes, it can be implemented for growing a system, enabling us to push AI in the direction towards being more 'life-like' by reducing human involvement in the design of complex functioning architectures. The growth paradigm implemented in this section has been inspired by mechanisms that regulate neocortical development [Rakic, 2000; Bystron, Blakemore, and Rakic, 2008].

The process of growing a layered neural network involves two major sub-processes. One, every 'node' can divide horizontally to produce daughter nodes that populates the same layer; two, every node can divide vertically to produce daughter processing units that migrate upwards to populate higher layers. Division is stochastic and is controlled by a set of random variables. Having defined the 3D scaffold, we seed a single unit (Fig. 3.18a). As horizontal and vertical division ensues to form the layered neural network, inter-layer connections are modified based on the emergent activity wave in layer-I and a learning rule (section-3.2) in layer-II, to form a pooling architecture. A detailed description of the growth rule-set coupled with a flow chart governing the growth of the network is appended to the supplemental materials.

Having intertwined the growth of the system and self-organization of inter-layer connections, we make the following observations: (1) spatiotemporal waves emerge in the first layer much before the entire layer is populated (Fig. 3.18b), (2) self-organization of inter-layer connections commences before the layered network is fully constructed (Fig. 3.18c) and (3) Over time, the system reaches a steady state as the number of 'cells' in the layered network remains constant and most processing units in the second layer connect to a pool of nodes in the first layer, resulting in the

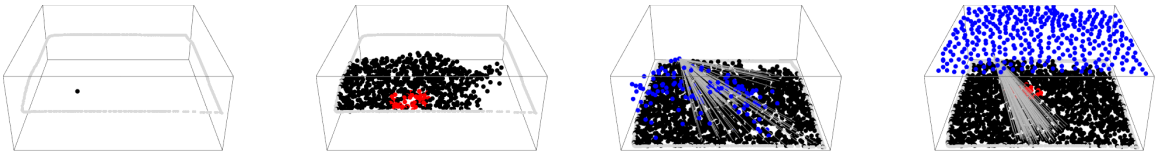


Figure 3.18: **Growing a layered neural network** (A) A single computational "cell" (black node) is seeded in a scaffold defined by the grey boundary. (B) Once this "cell" divides, daughter cells make local-excitatory and global-inhibitory connections. As the division process continues, noisy interactions between nodes results in emergent spatiotemporal waves (red-nodes). (C) Some nodes within layer-I divide to produce daughter cells that migrate upwards to form processing units (blue nodes). The connections between the two layers are captured by the lines that connect a single unit in a higher layer to nodes in the first layer (Only connections from a single unit are shown).(D) After a long duration, the system reaches a steady state, where two layers have been created with an emergent pooling architecture.

pooling architecture (Fig. 3.18d). Videos of networks growing on arbitrary scaffolds are added to the supplemental materials.

Section 3.9

Functionality of grown and self-organized networks

In the previous section, we demonstrated that we can successfully grow multi-layered pooling networks from a single unit. In this section, we show that these networks are functional.

3.9.1. Unsupervised feature extraction

For the task of unsupervised feature extraction, we feed a stream of images as input to the algorithm in real-time, with a frame rate of one image every 5 seconds, while time-integrating the multi-layered SNN (Fig. 3.19).

As a structured image-input is available, the parameter regime for the input layer (L_1) is chosen to ensure that noisy clusters of firing neurons shaped like the input image (here, MNIST digits) with spatio-temporal oscillations appear. Although there

are no activity waves in L_1, we demonstrate that waves **will** still emerge in the subsequent layers.

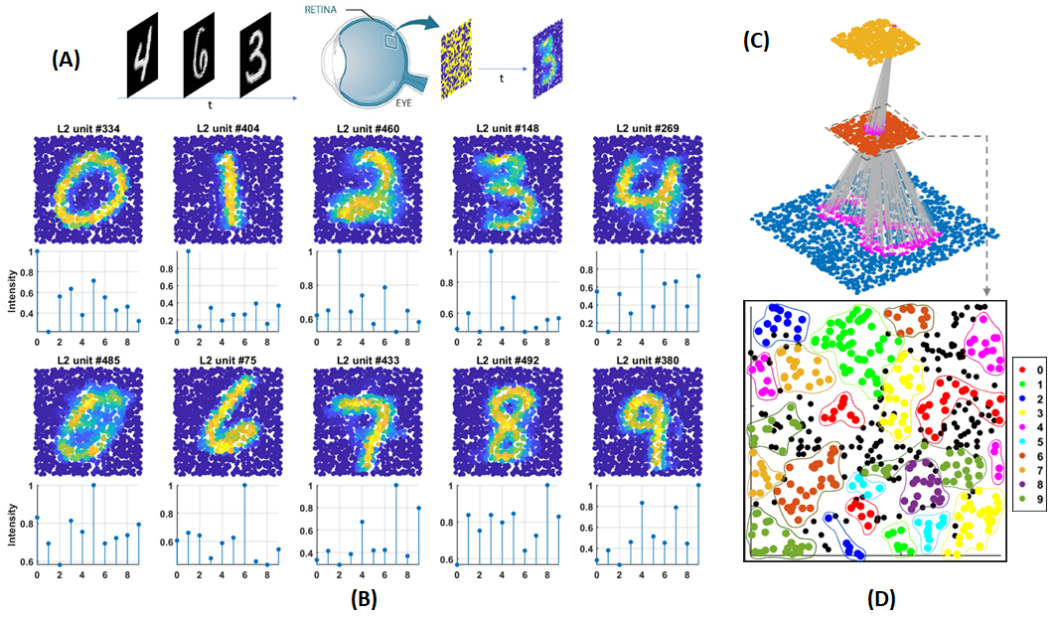


Figure 3.19: **Unsupervised learning of self-organized networks**. (A) Schematic of bio-inspired real-time learning: a 3-layered SNN learns on 2000 images, while being forward-integrated in time; it tests on circa 8000 images. (B) Unsupervised feature extraction forms pools that resemble MNIST digits: $\mathbf{W}^{(1)}$ weights of 10 exemplary L_2 neurons connecting to displayed L_1 neurons that form pools in shapes of digits. The respective tuning curves of each L_2 unit shows the (0-1-scaled) mean output spike intensities to input spikes of all kinds of digits in the test set − demonstrating the specialized L_2 unit spiking most intensely for one specific digit. (C) Exemplary connectivity pattern of the 3-layered network: pooling connection in shape of an '8'. (D) Coherent learning clusters in the L_2 that each, as a local group, specialize on learning/classifying a certain class of input digit.

The local learning rules coupled with competition rules enable many L_2 neurons to extract features from the input image (MNIST digits). Also, certain L_2 units **specialize** on a single class of MNIST digits. The specialization of L_2 units for a single class of MNIST digits is clearly observed by visualizing its self-organized connectivity to the input-layer and its tuning curves, both depicted in Fig. 3.19B. The tuning curve for an L_2 unit is generated by feeding 10 classes of MNIST digits to the network and recording its spiking intensity. For instance, in Fig. 3.19B, L_2 unit #404 has a connectivity to the input-layer that resembles MNIST digit '1' and

its tuning curve (plotted below) confirms that L_2 unit #404 maximally spikes when MNIST digits of class '1' are fed as input. Another interesting feature of our self-organization algorithm is that the neurons in L_2 that specialize for certain classes of MNIST digits, also **spatially cluster** within the layer. The spatial clustering of L_2 units for different MNIST classes are shown in Fig. 3.19D. The different node-colors correspond to neurons in L_2 that specialize to different MNIST classes. The spatial clustering of input-classes in L_2 is a direct consequence of the emergent spatio-temporal waves in L_2. Since the inter-layer connectivity is randomly initialized (mean: $\mu = 1$, std. dev. $\sigma = 0.5$) at $t = 0$, even if a learning rule enabled the learning and increased specialization of certain L_2 units, one would not observe the formation of any type of spatial clustering of input-classes, i.e., the distribution of specialized neurons would be arbitrary, if it was not for the wave. The spatio-temporal wave in L_2 enables the formation of spatially coherent connections that proceed to become specialized coherent learning structures within L_2.

3.9.2. Supervised learning (MNIST)

We demonstrate functionality of networks grown and self-organized from a single unit (Fig. 3.20c) by evaluating their train and test accuracy on a classification task. Here, we train networks to classify images of handwritten digits obtained from the MNIST dataset (Fig. 3.20e). To interpret the results, we compare it with the train/test accuracy of hand-crafted pooling networks and random networks. Hand-crafted pooling networks have a user-defined pool size for all units in layer-II (Fig. 3.20b), while random networks have units in layer-II that connect to a random set of nodes in layer-I without any spatial bias (Fig. 3.20c), effectively not forming a pooling layer.

To test functionality of these networks, we couple the two-layered network with a linear classifier that is trained to classify hand-written digits from MNIST on the basis of the representation provided by these three architectures (hand-crafted, self-

organized and random networks). We observe that self-organized networks classify with a 90% test accuracy, are statistically similar to hand-crafted pooling networks (90.5%, p-value = 0.1591) and are statistically better than random networks (88%, p-value = 5.6 x 10^{-5}) (Fig. 3.20a). Performance is consistent over multiple self-organized networks. These results demonstrate that self-organized neural networks are functional and can be adapted to perform conventional machine-learning tasks, with the additional advantage of being autonomously grown from a single unit.

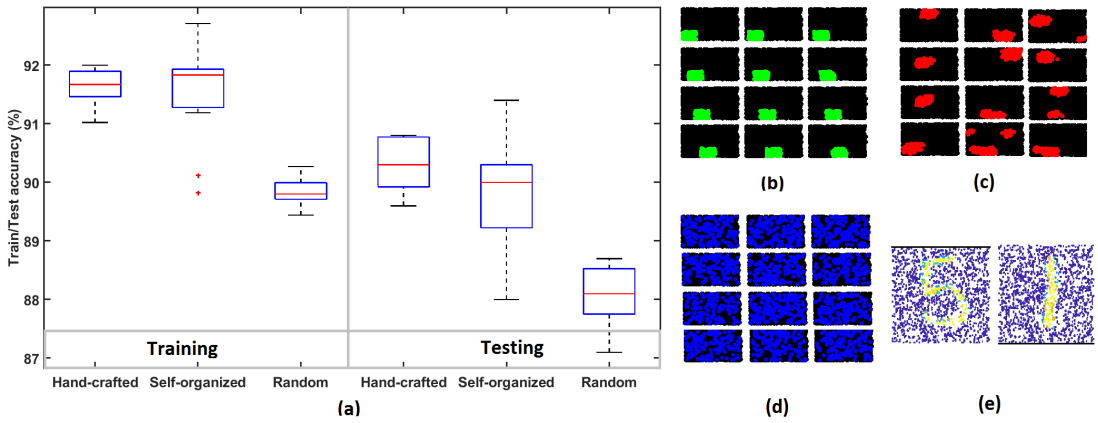


Figure 3.20: **Networks grown from a single unit are functional.** Three kinds of networks are trained and tested on images obtained from the MNIST database. We use 10000 training samples and 1000 testing samples. The 3 kinds of networks are: (i) Hand-crafted, (ii) Self-organized networks and (iii) random networks. The training procedure is run over n=11 networks to ensure that the developmental algorithm always produces functional networks. (A) The box-plot captures the training and testing accuracy of these 3 networks. We notice that the testing accuracy of self-organized networks is comparable to that of to that of hand-crafted networks (p-value = 0.1591>0.05) and are much better than random networks (p-value = 5.6 x 10^{-5}). (B, C, D) Each unit in the second layer is connected to a set of nodes in the lower layer. The set it is connected to are defined by the green, red or blue nodes in the subplots shown. (B) Hand-crafted (C) Self-organized and (D) Random-basis.(E) Two MNIST images as seen in the first layer.

3.9.3. Supervised learning (Gesture recognition)

In addition to static image recognition tasks, spiking neural networks can be used for dynamic event learning, owing to their intrinsic time-dependent evolution of their neural states (unlike most ReLU based artificial neural networks). In order to test

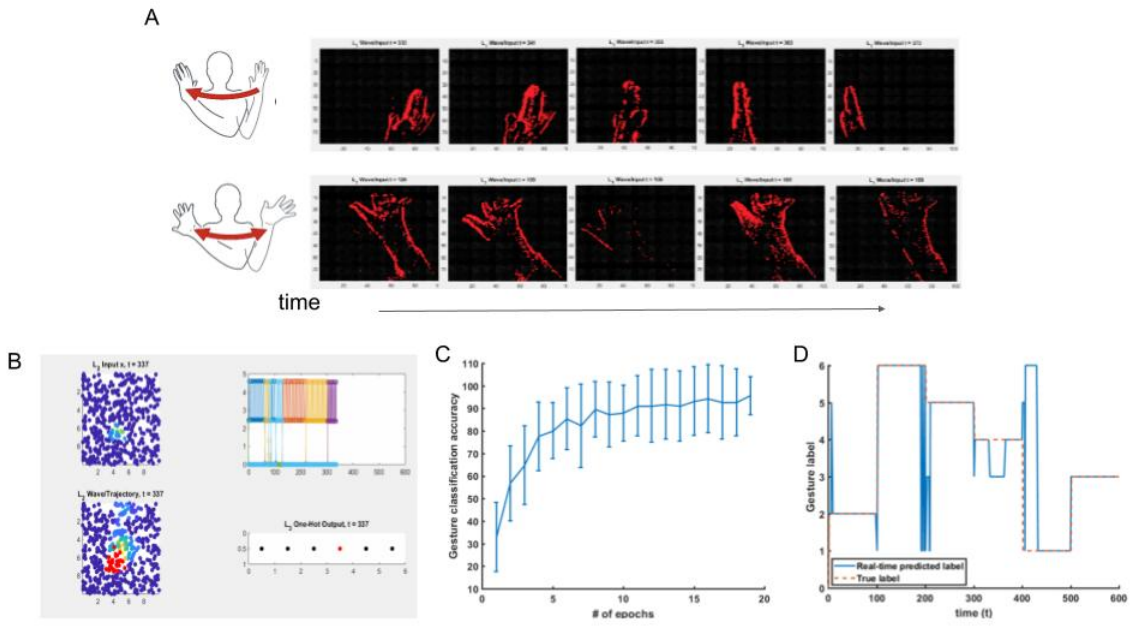


Figure 3.21: **Real time gesture capture with self-organized neural networks.** (A) We trained our networks on data from a benchmark American sign language data set. Data consists of video rate images from an event based camera where example gestures are shown in A. (B) Images showing two layers of our self-organized neural network responding to gesture sequences. The sub-panel on the right of (B) shows the network classifying the images into distinct categories (C) Our networks achieve >90% accuracy after only 6 rounds of training. (D) Networks run on real-time images streams making dynamic inferences (blue curve) to identify the gesture being displayed (red curve indicates ground truth).

their prowess on dynamic tasks, we trained spiking networks on a gesture recognition task.

We find that self-organized spiking networks can learn the gesture recognition task within a small number of training examples (Fig. 3.21) learning the task within a few seconds or minutes of training. Here, our training dataset has a total of six different classes of (hand) gestures that have been captured using a DVS[4] camera (Fig. 3.21A).

[4]DVS (Dynamic vision sensors) are functionally inspired by the biological retina, i.e., they only capture and relay a difference in brightness and not the entire frame.

Section 3.10

Discussion

In this chapter, we address a pertinent question of how artificial computational machines could be built autonomously with limited human intervention. Currently, architectures of most artificial systems are obtained through heuristics and hours of painstaking parameter tweaking. Inspired by the development of the brain, we have implemented a developmental algorithm that enables the robust growth and self-organization of functional layered neural networks.

Implementation of the growth and self-organization framework brought many crucial questions concerning neural development to our attention. Neural development is classically defined and abstracted as occurring through discrete steps, one proceeding the other. However in reality, development is a continuous flow of events with multiple intertwined processes [Arlotta and Vanderhaeghen, 2017]. In our work on growing artificial systems, we observed the mixing of processes that control growth of nodes and self-organization of connections between nodes. The mixing of growth and connection processes got us interested in how timing can be controlled when processes occur in parallel.

The work also reinforces the significance of brain-inspired mechanisms for initializing functional architecture to achieve generalization for multiple tasks. A peculiar instance in the animal kingdom is the presence of precocial species, animals whose young are functional immediately after they are born (examples include domestic chickens, horses). One mechanism that enables functionality immediately after birth is spontaneous activity that assists in maturing neural circuits much before the animal receives any sensory input. Although we have shown how a layered architecture (mini-cortex) can emerge through spontaneous activity in this paper, our future work will focus on growing multiple components of the brain, namely a hippocampus and

a cerebellum, followed by wiring these regions in a manner useful for an organism's functioning. This paradigm of growing mini-brains in-silico will allow us to (i) explore how different components in a biological brain interact with one another and guide our design of neuroscience experiments and (ii) equip us with systems that can autonomously grow, function and interact with the environment in a more 'life-like' manner.

References

Amari, Shun-ichi (1977). "Dynamics of pattern formation in lateral-inhibition type neural fields." In: *Biological Cybernetics* 27.2, pp. 77–87.

Antón-Bolaños, Noelia et al. (2019). "Prenatal activity from thalamic neurons governs the emergence of functional cortical maps in mice." In: *Science*, eaav7617.

Arlotta, Paola and Pierre Vanderhaeghen (2017). "Editorial overview: Developmental neuroscience 2017." In: *Current opinion in neurobiology* 42, A1.

Bichler, Olivier et al. (2011). "Unsupervised features extraction from asynchronous silicon retina through spike-timing-dependent plasticity." In: *The 2011 International Joint Conference on Neural Networks*. IEEE, pp. 859–866.

Bystron, Irina, Colin Blakemore, and Pasko Rakic (2008). "Development of the human cerebral cortex: Boulder Committee revisited." In: *Nature Reviews Neuroscience* 9.2, p. 110.

Denny, Christine A., Evan Lebois, and Steve Ramirez (2017). "From engrams to pathologies of the brain." In: *Frontiers in Neural Circuits* 11, p. 23.

Eglen, Stephen J. and Julijana Gjorgjieva (2009). "Self-organization in the developing nervous system: Theoretical models." In: *HFSP Journal* 3.3, pp. 176–185.

Elsken, Thomas, Jan Hendrik Metzen, and Frank Hutter (2018a). "Efficient multi-objective neural architecture search via lamarckian evolution." In: *arXiv preprint arXiv:1804.09081*.

— (2018b). "Neural architecture search: A survey." In: *arXiv preprint arXiv:1808.05377*.

Fukushima, Kunihiko (1988). "Neocognitron: A hierarchical neural network capable of visual pattern recognition." In: *Neural Networks* 1.2, pp. 119–130.

Fukushima, Kunihiko and Nobuaki Wake (1991). "Handwritten alphanumeric character recognition by the neocognitron." In: *IEEE Transactions on Neural Networks* 2.3, pp. 355–365.

Glickfeld, Lindsey L. and Shawn R. Olsen (2017). "Higher-order areas of the mouse visual cortex." In: *Annual Review of Vision Science* 3, pp. 251–273.

Godfrey, Keith B. and Nicholas V. Swindale (2007). "Retinal wave behavior through activity-dependent refractory periods." In: *PLoS Computational Biology* 3.11, e245.

Hanks, Timothy D. and Christopher Summerfield (2017). "Perceptual decision making in rodents, monkeys, and humans." In: *Neuron* 93.1, pp. 15–31.

Haussler, A. (1983). "Development of retinotopic projections: An analytical treatment." In: *Journal of Theoretical Neurobiology* 2, pp. 47–73.

Hubel and Wiesel (1963). "Shape and arrangement of columns in cat's striate cortex." In: *The Journal of Physiology* 165.3, pp. 559–568.

Izhikevich, Eugene M (2003). "Simple model of spiking neurons." In: *IEEE Transactions on neural networks* 14.6, pp. 1569–1572.

Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov (2015). "Siamese neural networks for one-shot image recognition." In: *ICML Deep Learning Workshop*. Vol. 2.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Advances in Neural Information Processing Systems* 25.

Kutscher, Brett, Peter Devreotes, and Pablo A. Iglesias (2004). "Local excitation, global inhibition mechanism for gradient sensing: An interactive applet." In: *Science Signaling (STKE)* 2004.219, pl3–pl3.

Laing, Carlo R and Carson C Chow (2001). "Stationary bumps in networks of spiking neurons." In: *Neural computation* 13.7, pp. 1473–1494.

LeCun, Yann et al. (1990). "Handwritten digit recognition with a back-propagation network." In: *Advances in Neural Information Processing Systems*, pp. 396–404.

Markram, Henry, Wulfram Gerstner, and Per Jesper Sjöström (2011). "A history of spike-timing-dependent plasticity." In: *Frontiers in Synaptic Neuroscience* 3, p. 4.

Meister, Markus et al. (1991). "Synchronous bursts of action potentials in ganglion cells of the developing mammalian retina." In: *Science* 252.5008, pp. 939–943.

Olshausen, Bruno A. and David J. Field (1996). "Emergence of simple-cell receptive field properties by learning a sparse code for natural images." In: *Nature* 381.6583, p. 607.

Padoa-Schioppa, Camillo and Katherine E. Conen (2017). "Orbitofrontal cortex: A neural circuit for economic decisions." In: *Neuron* 96.4, pp. 736–754.

Peirce, Jonathan W. (2015). "Understanding mid-level representations in visual processing." In: *Journal of Vision* 15.7, pp. 5–5.

Raghavan, Guruprasad and Matt W. Thomson (2019). "Neural networks grown and self-organized by noise." In: *Advances in Neural Information Processing Systems* 32. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/1e6e0a04d20f50967c64dac2d639a577-Paper.pdf.

Rakic, Pasko (2000). "Radial unit hypothesis of neocortical expansion." In: *Novartis Foundation Symposium*. Wiley Online Library, pp. 30–52.

Real, Esteban, Alok Aggarwal, et al. (2018). "Regularized evolution for image classifier architecture search." In: *arXiv preprint arXiv:1802.01548*.

Real, Esteban, Sherry Moore, et al. (2017). "Large-scale evolution of image classifiers." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 2902–2911.

Sandini, Giulio et al. (1993). "Retina-like CCD sensor for active vision." In: *Robots and Biological Systems: Towards a New Bionics?* Springer, pp. 553–570.

Saxe, Andrew M et al. (2011). "On Random Weights and Unsupervised Feature Learning." In: *ICML*. Vol. 2. 3, p. 6.

Scaramuzza, Davide (2014). "Omnidirectional camera." In: *Computer Vision: A Reference Guide*, pp. 552–560.

Silver, David et al. (2017). "Mastering the game of go without human knowledge." In: *nature* 550.7676, pp. 354–359.

Song, William and Jim Cai (2015). "End-to-end deep neural network for automatic speech recognition." In: *Standford CS224D Reports*.

Stanley, Kenneth O. and Risto Miikkulainen (2002). "Evolving neural networks through augmenting topologies." In: *Evolutionary Computation* 10.2, pp. 99–127.

Swindale, N. (1980). "A model for the formation of ocular dominance stripes." In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 208.1171, pp. 243–264.

— (1996). "The development of topography in the visual cortex: A review of models." In: *Network: Computation in Neural Systems* 7.2, pp. 161–247.

Tan, Hui Min, Thomas Joseph Wills, and Francesca Cacucci (2017). "The development of spatial and memory circuits in the rat." In: *Wiley Interdisciplinary Reviews: Cognitive Science* 8.3, e1424.

Varadhan, Ravi, Paul Gilbert, et al. (2009). "BB: An R package for solving a large system of nonlinear equations and for optimizing a high-dimensional nonlinear objective function." In: *Journal of Statistical Software* 32.4, pp. 1–26.

Willshaw, David J. and Christoph Von Der Malsburg (1976). "How patterned neural connections can be set up by self-organization." In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 194.1117, pp. 431–445.

Wong, Rachel O. (1999). "Retinal waves and visual system development." In: *Annual Review of Neuroscience* 22.1, pp. 29–47.

Xiong, Yuan et al. (2010). "Cells navigate with a local-excitation, global-inhibition-biased excitable network." In: *Proceedings of the National Academy of Sciences* 107.40, pp. 17079–17086.

Zhou, Yanqi and Gregory Diamos (2018). "Neural architect: A multi-objective neural architecture search with performance prediction." In: *Proceedings of Machine Learning and Systems*.

Zoph, Barret and Quoc V. Le (2016). "Neural architecture search with reinforcement learning." In: *arXiv preprint arXiv:1611.01578*.

Zubler, Frederic and Rodney Douglas (2009). "A framework for modeling the growth and development of neurons and networks." In: *Frontiers in Computational Neuroscience* 3, p. 25.