# Chapter 8. Appendix B: Population decoding: principles and methods

## 8.1 Motivation and principles

A principal goal of systems neuroscience is to understand what information is present in spike trains and in which form ("neural coding"). Neurons coding for a particular variable or stimulus feature are typically identified by comparing the mean firing rate between two conditions, repeated over many trials. While this identifies neurons that differ in firing rate on average, it tells us little about how a downstream region (that receives this signal) might use this information. In reality, the brain can not average over several trials. The relevant unit of information is thus what can be decoded from a single trial and not an average. Of course a downstream region receives the simultaneous outputs of many neurons rather then just one. Thus, some form of averaging between neurons can take place for a single trial. This does not necessarily improve the information content, however. Spiking of pairs of neighboring neurons can be highly correlated, and averaging correlated signals can either increase or decrease information content based on the exact nature of the correlations (Abbott and Dayan, 1999; Mazurek and Shadlen, 2002; Seung and Sompolinsky, 1993; Sompolinsky et al., 2001). External noise that influences the spiking of several neighbouring neurons results in positive correlations, as does common input or recurrent connectivity. This can lead to a decrease in information due to correlations (Sompolinsky et al., 2001), and imposes a fundamental constraint on the amount of information that can be represented by a population of neurons. Averaging a large number of

units would thus not necessarily improve information content. This effect has been demonstrated experimentally by recording from pairs of neurons in area MT of non-human primates (Zohary et al., 1994). Surprisingly, the spikes emitted by a single motion-selective MT unit allow perceptual discrimination that is as good as the psychophysical sensitivity of the animal. Thus, the sensitivity of a single neuron is indistinguishable from the sensitivity of the entire animal. The spiking of pairs of neurons was weakly positively correlated. In this particular instance this correlation resulted in no increase in information content if more than 50–100 units were averaged (Zohary et al., 1994). In this case, averaging could only improve the signal-to-noise ratio by 2–3 times (even if large numbers of units were considered).

How is information represented in single neurons as well as populations thereof? Comparing the output of neurons between different conditions requires the use of features of spike trains. The simplest example of a feature is counting the number of spikes emitted per units of time ("rate code"). However, many other features could possibly contain information as well, such as the time of occurrence of a spike ("temporal code"). Examples of more complex features are the number of spikes that are less than 5 ms apart (a burst), the correlations between different units, or the phase relationship of a spike to a particular frequency of the local field potential. Cortical as well as hippocampal neurons fire highly irregularly (Fenton and Muller, 1998; Shadlen and Newsome, 1998; Softky and Koch, 1993). The variability of spike times generally increases linearly with the mean firing rate (Poisson). Thus, increasing the firing rate alone does not reduce response variability. This inherent uncertainty imposes constraints on possible useful features, as any feature that relies on highly accurate timing is unlikely to be useful in general (with some exceptions). There are many proposals on what the fundamental unit of neural coding

is (Abeles, 1991; MacKay and McCulloch, 1952; Rieke, 1997). While there is little agreement, it is acknowledged that the simplest of all codes, spike counts, is surprisingly good. In some cases, incorporating time in addition to rate improves decoding performance. There are only a few demonstrated examples where a rate code alone does not allow reading out of a substantial amount of the information available (Butts et al., 2007; Johansson and Birznieks, 2004; Laurent, 2002; MacLeod et al., 1998; Montemurro et al., 2007; Stopfer et al., 1997). This statement only refers to readout of information and does not imply that precise spike timing is not important for other processes, such as plasticity (discussed elsewhere in this thesis). However, since there are exponentially more possible combinations the more neuronal features are used, a rate code can transmit much less information per unit of time compared to a more complex code. Population decoding is a powerful technique that can address questions of coding and the feature used (see below).

An alternative method for quantifying the information present in the output of a population of neurons is decoding. This method avoids averaging entirely (both across trials and across neurons). Mathematically, a decoder is a function $y = f(x)$ that takes as input information about all available neurons (such as firing rate or spike times) and gives as output a prediction of what the input represents. For example, x could be the response of a sensory neuron in response to a stimulus y. The sensory neuron transforms the stimulus y into a neural response $x = g(y)$. The task of the decoder is to reverse this and reconstruct, from only observing the neuronal response $x$, the input $y$. Thus in this example $f = g^{-1}$. Traditional neurophysiology focuses on establishing the transformation of input to neuronal output, i.e., $x = g(y)$. However, from the

perspective of the brain, estimating ("decoding") the external world from the neuronal responses is the relevant task (Bialek et al., 1991). $f(x)$ can be determined automatically by a machine learning algorithm ("classifier") from a subset of the data. Alternatively, f(x) can also be pre-defined by a model with a few parameters estimated from the data. The performance of such a decoder, on a separate test set, is an estimate of how much information about the state of the neural system can be extracted, on a trial-by-trial basis, by the decoding technique used. Note the importance of an entirely independent test set. One mistake, for example, is to pre-select neurons (or time bins for LFP) using a statistical test and then estimate decoding performance using leave-one-out cross-validation. Such a test set is not independent: the samples were already used to pre-select the input. Thus the decoder will, by definition, return above-chance performance (as the inputs passed a statistical test). While this can still be meaningful (for example to estimate how difficult it is to decode on a single trial or compare different conditions), the fact that the information is present as such is not a new finding. If the particular classifier used has a structure that could conceivably be implemented by a network of realistic neurons (such as a linear weighted sum, as used in this thesis), one can reasonably assume that a real neural network could be reading out this information in a similar fashion. If, on the other hand, the information can only be extracted by a mechanism that can hardly be implemented biologically (such as requiring sub-millisecond resolution), it is less plausible that this information could be read out by another brain area.

A decoding approach is particularly useful to quantify how much information is present about the state of the system in a population of neurons. Plotting decoding performance as a function of variables such as number of units, time relative to stimulus onset, time resolution

(binning), or the neuronal code used allows quantification of such effects. Also, decoding can determine what state the neural population represented during a behavioral error (such as forgetting a picture or making the wrong perceptual decision). This allows one to clearly discern whether an area follows the decision, the motor output, or the sensory input. Another question that can be investigated using decoding is the latency of the response: By what point in time does a population of neurons distinguish best between two particular stimuli? Decoding can also be used to distinguish between different kind of neuronal codes: Does a code that distinguishes exact spike timing contain more information then a rate code? The answer to this question will depend on the exact experiment and experimental model but can easily be investigated with decoding.

There is a trade-off between readout difficulty and the robustness and richness of the code. The easiest code to read out would be if every neuron represents a particular concept exclusively ("grandmother neuron"). Thus decoding is trivial — counting the number of spikes emitted by this neuron is sufficient. The number of represented concepts is limited by the number of neurons. This representation is, however, not robust. If this single neuron dies, the representation is lost. On the other hand a fully distributed code can represent vastly more concepts ( $2^N$ ), but such a representation is very difficult to read out—a decoder needs access to all neurons simultaneously. The decoding approach to neuronal spike train analysis is useful to specify the difficulty of readout, because a given decoder quantifies how much information can be read out with the given complexity of the decoder.

There are a large variety of techniques for constructing population decoders. These include simple linear-sum type decoders such as the perceptron and go all the way to highly non-linear support vector machines (SVMs). In the following section the decoding

techniques used in this thesis are summarized. In our experience the performance of highly complex decoders is often only marginally better than simple linear decoders (in the context of the analysis performed here). Thus, in the interest of understanding neural coding rather then machine learning, it is (in our opinion) often advisable to use a very simple linear decoder. If it turns out that higher-order interactions between terms are important, it is still possible to include these in a linear decoder by introducing additional variables that represent the higher-order terms.

Apart from being a valuable approach for data analysis, single-trial decoding of neural activity also has practical applications. It has been demonstrated that small numbers of neurons in an appropriate area of the brain provide enough information to allow a human or monkey to remote-control a robotic device (or a computer) by thought only (Andersen et al., 2004; Carmena et al., 2003; Hochberg et al., 2006; Rizzuto et al., 2005; Serruya et al., 2002). This is a direct demonstration of the value of decoding approaches.

## 8.2 Definitions

The matrix $Z$ contains the data samples. Rows correspond to training samples (n) and columns to variables (p, such as spike counts). Thus, $Z$ is of dimensionality $n \times p$. $Z$ contains a column of 1s to account for a linear offset relative to the mean. Each sample has one numerical label (for example -1 or 1, in the binary case), which is stored in the vector $y$ of dimensionality $n \times 1$. The vector of weights $w$ is of dimensionality $n \times 1$.

### 8.3 Multiple linear regression

With multiple linear regression (Eq B1), the weights w are determined by multiplying the inverse of data samples $Z$ with the training labels y (Johnson and Wichern, 2002).

$$w = \left[Z'Z\right]^{-1} Z'y \qquad \text{(Eq B1)}$$

### 8.4 Regularized least-square regression (RLSC)

Multiple linear regression can not determine the weight vector unambiguously if the sample matrix is ill conditioned or can not be inverted for other reasons (Eq B1). Even if the matrix can be inverted, the resulting matrix can be numerically unstable. In practice, such problems often arise due to larger number of variables then training samples (i.e., 100 neurons and 50 trials). Another common source is linear dependency between variables, which leads to rank-deficiency.

To circumvent this problem, additional constraints (on the weights) such as smoothness or small numerical values need to be enforced. One way to achieve this is to add a constant term (regularizer). Here, we used regularized least squares (RLSC) to achieve this. In RLSC (Evgeniou et al., 2000; Hung et al., 2005; Rifkin et al., 2003), an additional term is added to the data samples (Eq B2). Here, $I$ is the identity matrix and $\lambda$ is a scalar parameter (the regularizer).

$$w = \left[Z'Z + \lambda I\right]^{-1} Z'y \qquad \text{(Eq B2)}$$

The value of the regularizer is arbitrary. The bigger it is, the more constraints are placed on the solution (the less the solution is determined by the data samples). A small value of the regularizer,

on the other hand, makes the solution close to the multiple linear regression solution. Importantly, however, even a small value of the regularizer punishes unrealistically large weights and also guarantees full rank of the data matrix. Regularization becomes particularly important when there are a large number of input variables relative to the number of training samples.

The value of the regularizer is an important determinant of the performance of the classifier. Thus, great care must be taken to choose it appropriately. In our experience, the exact value of the regularizer is not very important (not sensitive). It is enough to get it approximately right (i.e., 10 or 100). If the only aim is to make multiple linear regression numerically stable it is sufficient to add a very small regularizer, such as $\lambda = 0.01$ (this value was used in this thesis, unless stated otherwise). Increasing the value of $\lambda$ will lead to an increased training error but a decreased testing error (at least initially). It is an indicator that regularization is necessary (due to overfitting) if this pattern is observed. Thus it is possible to find a good regularizer value by plotting the testing and training error as a function of $\lambda$. The test set used to optimize the value of $\lambda$ has to be different than the test set used to determine classifier performance. This is only true if $\lambda$ is optimized in this way. If, on the other hand, $\lambda$ is constant, no separate test set is required.

One problem with regularization is bias. For illustration, assume a binary classification problem where 80% of the samples are of one class (with label 1). A classifier trained (using RLSC) with a large $\lambda$ on this dataset will predict "1" for every test sample. Due to the inherent bias in the prior distribution, the classifier will achieve 80% correct performance entirely due to chance. Thus, while this is a binary classification problem, the chance level for this regularizered classifier is 80% rather then 50%. If the bias in the test set is different than in the training set, the

chance performance is unclear. One is often presented with this situation in the context of the analysis of populations of neurons, such as training on behaviorally correct trials and then decoding the error trials (which typically have different bias). To circumvent this problem, the number of samples in each class needs to be balanced artificially before training the classifier (by removing samples from the bigger class). The excluded samples can nevertheless be used by training the same classifier multiple times based on a randomly sampled subset of the training data. If excluding samples is not an option, the chance level on the test set needs to be established using a bootstrap procedure. It can not be assumed to be 50%.