On the interpretation of higher-order principal components

Allan Mancoo

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École normale supérieure

On the interpretation of higher-order principal components

Soutenue par
Allan MANCOO
Le 3 mars 2021

École doctorale n°158
Cerveau, Cognition, Comportement

Spécialité
Neurosciences

Composition du jury :

Juan GALLEGO Imperial College London	*Rapporteur*
Henning SPREKELER Technical University of Berlin	*Rapporteur / Président du Jury*
Alex CAYCO-GAJIC École normale supérieure	*Examinatrice*
Jakob MACKE Technical University of Munich	*Examinateur*
Shihab SHAMMA École normale supérieure	*Examinateur*
Sophie DENÈVE École normale supérieure	*Directrice de thèse*
Christian MACHENS Champalimaud Foundation	*Co-directeur de thèse*

ENS | PSL

Acknowledgements

I would like to thank my supervisor, Christian Machens, for his continuous scientific guidance and unwavering support throughout my PhD. I benefited greatly from his fascinating pedagogical skills, clarity of thoughts and ability to always keep the big picture of the scientific pursuit in sight. His mentorship will definitely guide me along my future endeavours. I would also like to thank my second supervisor, Sophie Denève, for her scientific scrutiny about my work and continuous support throughout.

This work was highly influenced by several scientific collaborations. I am grateful to Roozbeh Kiani and Gouki Okazawa for sharing their data willingly and getting me onboard an exciting scientific project. Their expertise as experimenters gave me new perspectives on my own work. I am also thankful to Sander Keemink who influenced much of the work on spiking neural networks. His geometric thinking and meticulousness were invaluable.

I extend my thanks to the members of my thesis committee: Byron Yu and Alfonso Renart. Their critical questioning and numerous pieces of advice were highly appreciated. I would also like to thank Severin, Sander and especially Bill for proof-reading chapters of this thesis.

Throughout this journey, I was lucky enough to be surrounded by some highly insightful and helpful people. My thanks goes to the lab members in Lisbon from whom I learnt a great deal and for the countless hours spent in inspiring discussions. Special thanks to Severin, Bill, Oihane for their continuous support and encouragement, Nuno for being the go-to person for my math questions and his always positive attitude and Michael for whenever I had to deal with a Linux issue. I am also thankful to people in the lab in Paris for always welcoming and including me whenever I visited.

Undoubtedly, this journey would not have been possible without the love and support of my family and friends. I am indebted to Alexandra for her outstanding

support, especially over the last year under some strenuous personal circumstances. I thank my parents for being a steadfast source of inspiration, my sister for always being there and my extended family and numerous friends for always believing in me.

Finally, I am grateful for the financial support during the first three years of my PhD provided by a 'Bourse d'études du Gouvernement français' from the Ministry for Europe and Foreign Affairs/ Embassy of France in Mauritius.

To the fond memories of Aja

Summary

Large-scale recordings of neural activity are nowadays widely carried out in many experimental labs, leading to the important question of how to extract the essential structures in population data. One common way of doing so is through the use of dimensionality reduction methods. However, interpretation of the results of these tools can be fraught with difficulties. Most commonly, linear methods such as principal component analysis (PCA) are used due to their minimal assumptions and ease of implementation. While they work well in finding linear projections of the data that explain most variance, they also usually display a tail of components, among which several resemble higher-order functions of some other components. We refer to these as 'higher-order' components (HOCs). While these HOCs suggest that the true underlying neural manifold is non-linear, it is still unclear how they emerge and how they should be interpreted.

In this thesis, we argue that these HOCs largely arise due to a well-known non-linearity — individual neuronal activity is non-negative, which yields a curvature to the neural manifold, but the resulting HOCs are otherwise functionally irrelevant. We lead our investigation with the crucial assumptions that readouts of population activity should be linear and lower-dimensional, and that overall firing rates should be limited for reasons of energy efficiency. We show, in simulations, that when neural activities are generated under these assumptions, then PCA sometimes extracts the true underlying signals, but often displays a tail of HOCs to compensate for the curvature of the manifold. We explain these findings geometrically and show that they resemble the HOCs often observed in real data. To remedy this issue, we propose a set of dimensionality reduction methods that incorporate the non-negativity constraints in a meaningful way. We validate our methods against ground truth data, but also show, in an example experimental dataset, that incorporating this simple non-linearity affords a more condensed description of the data than PCA.

However, it is possible that the neural manifold exhibits some additional non-linearity besides the non-negativity constraints. We hypothesize lastly in this thesis,

that this non-linearity may emerge according to the computations done by the network. However, a clear understanding of computations done by biologically realistic neural networks is still missing. We complement this thesis by investigating the computations done by spiking neural networks (SNNs). We show that a broad class of SNNs can be understood through a framework inspired from convex optimization theory, with the network parameters intimately linked to the parameters of the underlying convex optimization problems. Thanks to this perspective, we show that input-driven SNNs fundamentally compute convex input-output functions and these computations can be understood (and learnt) geometrically. Finally, we show in simulations, that the resulting networks can display several biological features such as asynchronous and irregular spike trains, robustness to perturbations, among others.

Contents

Abbreviations

A1 primary auditory cortex

ABL Absolute Binaural Level

CCA Canonical Correlation Analysis

dPCA demixed Principal Component Analysis

DV decision variable

HOC higher-order component

ILD Inter-aural Level Difference

LIF Leaky Integrate-and-Fire

LIP lateral intraparietal

LN linear-nonlinear

PCA Principal Component Analysis

PFC prefrontal cortex

PSTH peri-stimulus histogram

QP quadratic program

ReLu rectified-linear unit

RF response field

SNN Spiking Neural Network

Chapter 1

General Introduction

Ever since the painstaking task by Santiago Ramón y Cajal in meticulously describing the fine organisation of nerve cells in neural tissues (Cajal, 1906), the individual neuron has been considered as the central unit of anatomy of the nervous system. Mostly due to technological limitations, the approach to study the nervous system has then been to measure the activity of one cell at a time, e.g. via single-cell electrophysiology (Hubel et al., 1957). This nonetheless, led to the important discovery that neurons would often respond to specific features of the outside world, which defined their receptive fields (Barlow, 1953; Hubel and Wiesel, 1959). Since then, a standard approach to understand brain functions has been to map such receptive fields across various brain areas across several species (e.g. (Mountcastle, 1957; Georgopoulos et al., 1982; Desimone et al., 1984)).

However, with more recent theoretical and experimental discoveries (Yuste, 2015; Eichenbaum, 2018), it is becoming questionable whether studying single neurons alone remains a promising route going forward. Instead, increasingly prevalent is the idea that neurons in a population collectively form a functional unit (Saxena and Cunningham, 2019). Thus, measuring the joint activity of the neurons in the population would be necessary to unravel emergent properties of the network — i.e. properties that arise from neuronal interactions in the network, but cannot be obtained from any individual neuron alone — to understand brain function. Accordingly, there has been a fervent endeavour, largely empowered by major advances in recording techniques in recent years (Stevenson and Kording, 2011; Ahrens et al., 2012; Stringer et al., 2019a), to record simultaneously as many neurons as possible. The hope is that these recordings would give a representative view of the population, and thus would allow us to advance our understanding of the brain.

This long-sought data however, presents a fundamental challenge in itself. As the

data has as many dimensions as the number of recorded neurons, and each neuron has its own unique activity traced over time, the data is both high-dimensional and complex. Consequently, questions of how to make sense of it or how to even start exploring and visualizing it immediately appear. A common approach to address these questions has been through statistical methods, namely dimensionality reduction methods, that aim to find a smaller set of variables that summarize the joint activity of the neurons (Cunningham and Yu, 2014; Pang et al., 2016). The hope is that these variables would then capture the essential structures in the data, and yield succinct and interpretable descriptions of what the brain might be representing and computing.

While these methods have generally led to key insights in neural population data (Cunningham and Yu, 2014; Keemink and Machens, 2019), interpretations of the results are not always straightforward. Usually, these methods predict many more structures in the data than can immediately be understood and interpreted. Hence, it is an open question what these other structures mean. In this thesis, we largely focus on this issue of how to correctly interpret the structures in population data resulting from dimensionality reduction, in particular linear methods. In the rest of this introductory chapter, we will review the motivation for dimensionality reduction and then, will make more concrete this issue of interpretations.

1.1 Analysing high-dimensional neural data

With major advances in recording techniques in recent years, we can nowadays record simultaneously from hundreds to thousands of neurons across a range of species. As noted in (Stevenson and Kording, 2011), the number of neurons that we can record simultaneously has been increasing exponentially with time (see Fig. 1.1 for a timeline) and today, up to 10^4 neurons in the visual cortex of mice (Stringer et al., 2019a) or even, the whole brain of larval zebrafish (Ahrens et al., 2012)) can be recorded. As a result, the data we are collecting are becoming increasingly high-dimensional as each recorded neuron contributes a dimension.

At the same time, as soon as hundreds of neurons are taken into consideration, the joint activity of these neurons becomes highly complex. This issue is even more severe in higher-order areas of the brain, e.g. the prefrontal cortex (PFC): despite the simplicity of tasks that need to be solved, it is usually observed that the neuronal responses display a dazzling degree of heterogeneity (Brody et al., 2003; Machens et al., 2010; Mante et al., 2013; Rigotti et al., 2013). As an example, we illustrate

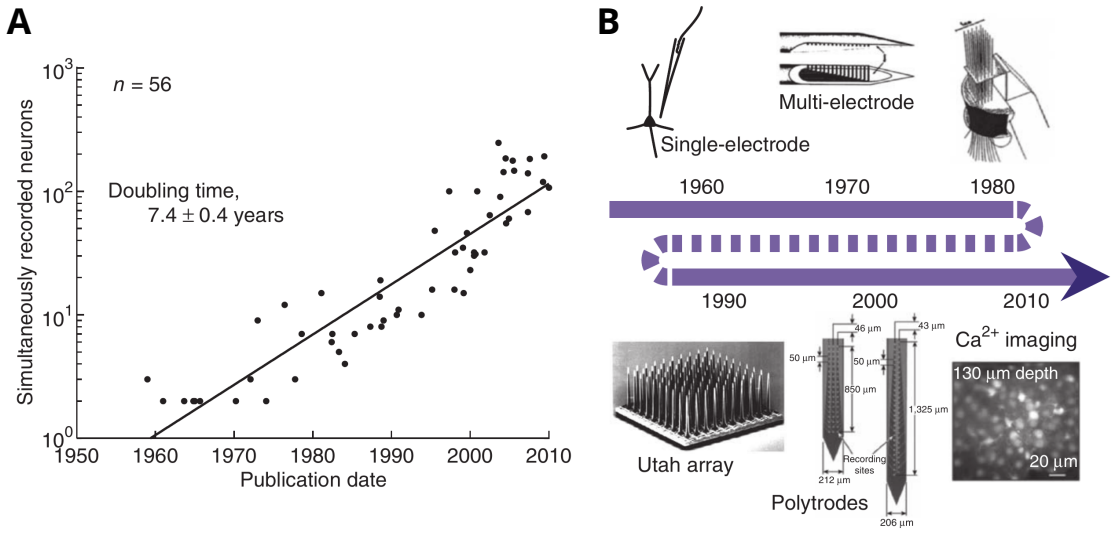

Figure 1.1: Exponential growth in number of neurons that can be recorded. Figure from (Stevenson and Kording, 2011) (A) Number of neurons recorded across several studies as a function of time. This number has been increasing exponentially, doubling approximately every seven years. (B) Timeline of recording techniques from single-electrode to in-vivo optical imaging that allowed this increase in (A).

in Fig. 1.2, the heterogeneity of neuronal activity in the PFC of monkeys during a working memory task (Romo et al., 1999; Brody et al., 2003). To solve the task, the animals needed to keep information about a first stimulus (F1) in memory for some time (a delay period) and compare it to a second stimulus (F2) that arrived after the delay period. Interestingly, during the task, the recorded neurons displayed a vast diversity of responses, with various degrees of tuning to the stimulus, to elapsed time and combinations of both (Brody et al., 2003; Machens et al., 2010)(Fig. 1.2C). This immediately raises the question of how to make sense of this data, and thereby reach conclusions about brain functions.

1.1.1 Neuron-by-neuron analysis to understand population data

When the experiment involves a behavioural task, one common approach has been been to relate individual neuronal responses to the task variables, e.g. task parameters or stimulus, which hopefully would then give clues about what the population is doing. This can be done, e.g. by regressing the neuronal responses onto the task variables (e.g. (Brody et al., 2003) for the PFC dataset we mentioned earlier) and thereby, infer the tuning of the neurons to these variables according to the regression coefficients. In turn, to understand the representation of these task variables at the population level, the distribution of these regression coefficients across all the

recorded neurons can then be computed to give a probabilistic description of the population.

However, this neuron-by-neuron analysis may fail to give a coherent and interpretable picture of the joint neural responses at the population level (Wohrer et al., 2013). This is because this analysis is strongly biased towards the model used to explain individual neuronal activity; the regression model, for instance, may fail to capture the fine details of neuronal responses and thus, when the resulting regression coefficients are brought together, they will miss out on possibly important structures in the population. For example, as we mentioned earlier, neuronal responses are typically highly heterogeneous in higher-order brain areas, with simultaneous tuning to various task parameters (Mante et al., 2013; Kobak et al., 2016; Rigotti et al., 2013; Raposo et al., 2014), and it is unlikely that the neuronal model would capture all these intricacies in the data. Hence, this two-step approach of analysing population data will most likely lose some information.

1.1.2 Analysis of neural populations via dimensionality reduction

To address the above issue, one can instead directly (in one step) analyse the ensemble of recorded neurons, instead of first looking at single neurons and then extrapolating to the population. A common approach to do so is through the use of dimensionality reduction methods (Cunningham and Yu, 2014; Pang et al., 2016).

When recording many neurons in the population, say N neurons, the activity patterns reside in an N-dimensional space. However, it might be the case that several of these neurons covary according to smaller number of explanatory variables, say $M < N$ of these variables. Dimensionality reduction methods then aim to extract these M unobserved, or latent, variables from the population data, which will provide a summary of the data.

The M explanatory variables are usually extracted according to an objective specific to each method. For instance, the putative most commonly used method, namely Principal Component Analysis (PCA), aims to find latent variables, or principal components, that explain most variance in the data. The data variance not captured by these PCs are then discarded as noise. Importantly, in contrast to the previous approach (see section above), dimensionality reduction methods, irrespective of their individual objective, take all information across the ensemble of recorded neurons into account to find statistical features of interest in the data.

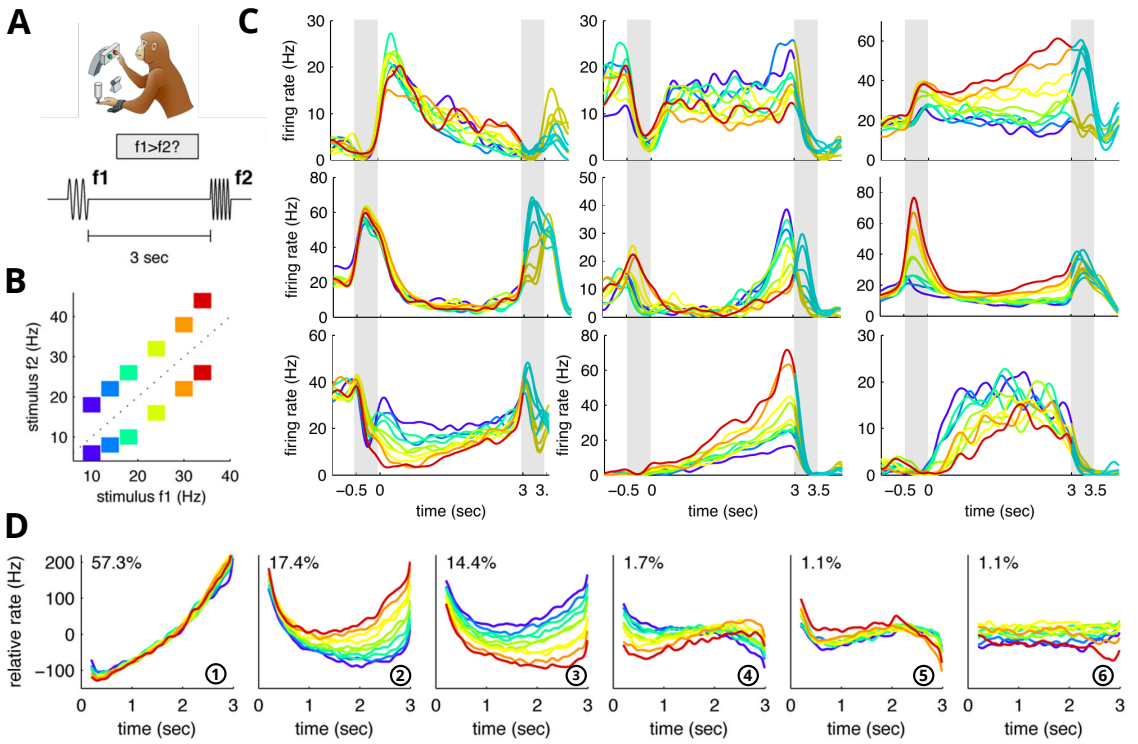

Figure 1.2: Highly heterogeneous neuronal activities in the prefrontal cortex (PFC) of monkeys while they solve a simple somatosensory working memory task (Romo and Salinas, 2003). Figure adapted from (Machens et al., 2010). (A) Cartoon of the task adapted from (Romo and Salinas, 2003). The monkey needed to distinguish whether a second vibrotactile stimulus, F2, was stronger than a first stimulus, F1, after a delay period, or not. Thereupon, the monkey needed to make a decision by pressing one of the buttons. (B) Color code according to the first stimulus. (C) Smoothed peri-stimulus histograms from nine example cells from PFC. These nine example cells illustrate the heterogeneity in neuronal responses despite the simplicity of the task. (D) First six principal components (PCs) that summarize the data from a monkey; numbers in circle show the ordering of the PCs according to variance explained (percentages on top).

Already for exploring and visualizing population data, dimensionality reduction can be a very useful tool. Indeed, instead of looking at the N-dimensional neural activity patterns which can be highly complex, it would suffice to simply look at the smaller M-dimensional latent variables to start building intuitions on what the data is representing. For instance, coming back to the PFC dataset (Fig. 1.2), upon applying PCA on the data during the delay period, it was found that around 95% of the variance of the data could be explained with $M = 6$ PCs (Machens et al., 2010) (Fig. 1.2D). Thus, the simplified and cleaner description of the data can hopefully, yield easier interpretations, from which one can draw conclusions on brain function.

As a method to deal with the complexity of neural data, dimensionality re-

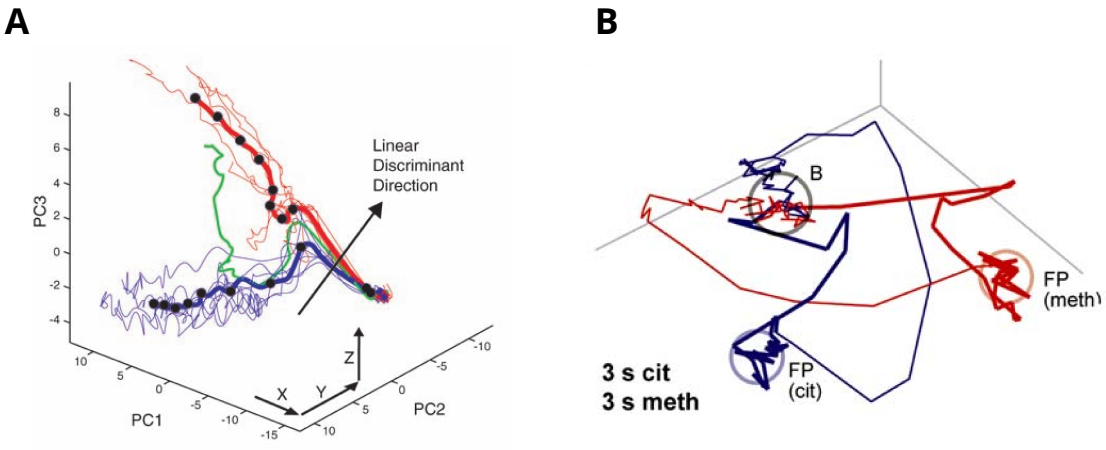

Figure 1.3: Examples where dimensionality reduction on neural population revealed functional principles of the circuit. (A) Decision making process in leeches between swimming (blue) and crawling (red) under the same sensory stimulus that could elicit either of the behaviours (Briggman et al., 2005). Analysing the population activity showed that the decision making process follows a dynamical system in the low dimensional space that branches off according to the choice to be made. The choices could be discriminated earlier from the joint neuronal activities than from any single neuronal activity. (B) Coding for odor identity using neural transients in locusts (Mazor and Laurent, 2005). When either of the two odors were presented (red or blue), the neural trajectory in the low-dimensional space would leave its baseline activity (B) and move to a steady state or fixed point (FP). When the stimulus is extinguished, the low-dimensional population trajectory returns to the baseline. Interestingly, this study revealed that the transients allow better discriminability of odor identities, long before the fixed points were reached, thus pointing to a novel emergent computational rule of the population.

duction has surprisingly been highly successful, as demonstrated by a multitude of studies (e.g. (Briggman et al., 2005; Machens et al., 2010; Mante et al., 2013; Churchland et al., 2012; Bathellier et al., 2008; Gallego et al., 2018)). This success is largely attributed to the fact that the bulk of neural activity tends to reside in a lower-dimensional subspace. Although theoretical work are now coming up to explain this phenomenon (Gao et al., 2017; Mastrogiuseppe and Ostojic, 2018), one hypothesis has been that the true underlying, or latent, signals of brain activity are in fact low-dimensional and these latent signals are then mapped onto the activities of many neurons. The brain would then be operating and computing on these low-dimensional latent signals — the so-called 'strong principle' of dimensionality reduction (Humphries, 2020). One posited advantage of this scheme is that it affords the system with robustness against the fragility of any individual neuron in the system; even if a neuron dies, or is perturbed in any other ways, this will not lead to any system failure.

According to this hypothesis, extracting the underlying signals of population activity will give insights on how the neural circuit functions. There are indeed, some experimental evidence suggesting that the dynamics of the low-dimensional signals could be a motif for brain computations. For example, Briggman et al. (2005) recorded and analysed populations of sensory neurons in leeches and showed that the low-dimensional dynamic profile of the population activity revealed the decision making process between two behavioural outputs, namely swimming and crawling (Fig. 1.3A). Interestingly, their analysis of neural population revealed a circuit computation, that would not have been accessible through the analysis of single neurons only. As another example, Mazor and Laurent (2005) showed that the low-dimensional dynamic profile of population activity in the insect's olfactory system exhibited transients that allowed to compute odor identities, long before the circuit reached a steady state of activity (Fig. 1.3B). This neural population analysis thus revealed a new computational scheme that could be used by insects to identify odors.

1.2 Interpreting results of dimensionality reduction

Although dimensionality reduction has provided key insights for understanding neural population data (Cunningham and Yu, 2014; Keemink and Machens, 2019), interpreting the results of these methods can still be fraught with difficulties. Usually, the methods will extract several features in the data in order to meet some objective, e.g. maximizing variance as in PCA (Bishop, 2006). Nevertheless, not all of these features can immediately be understood and interpreted. This is because interpretations are usually limited to the experimenters' knowledge of the experimental paradigm, e.g. task rules (what should be computed or solved) and experimentally controlled task parameters, as well as, observed behavioural responses, either elicited or spontaneous. Hence, it is still an open question what the other features in the data mean. Yet, to advance our understanding of the data, elucidating the meaning of these features is crucial.

To bring home this point, we go back to the PFC dataset mentioned earlier (Fig. 1.2). We recall that the animal needed to memorize a first stimulus, F1, during a delay period to solve the task. When PCA was applied to the data (Machens et al., 2010), we saw that $\sim$ 95% of variance of the data could be explained with the first six PCs (Fig. 1.2D, ordered according to the variance explained by each

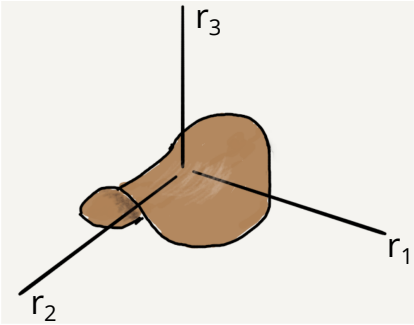

Figure 1.4: Schematic to illustrate intrinsic and embedding dimensions (figure adapted from Humphries (2020)). The manifold in brown can be curved, in which case it is embedded in a three-dimensional space. Thus, a data point on the manifold can be characterized by values along axes (r_1, r_2, r_3). However, the manifold truly has only two intrinsic dimensions and thus, the same data point can be described equally well with only two variables.

component). There, we observe that the first PC shows a ramping activity without any particular tuning to F1 (the colored traces, with each color corresponding to an F1 stimulus, overlap) and thus, one could interpret this component as reflecting the coding for time by the population. The third PC, on the other hand, shows a roughly flat activity over time, but the colored traces are now separated. This component may then be interpreted as the population coding for stimulus, F1. However, this population analysis also reveals several other PCs. Yet, it is unclear how these should be interpreted. If the PCs are truly reflecting underlying mechanisms of the neural circuit, then it is crucial that they should also be understood and interpreted for us to reach overall conclusions about brain functions.

However, we immediately face an important conundrum: how can we be sure that the features that appear upon reducing dimension reflect the functionally relevant latent signals of population data. Another possibility could be that these features emerge according to the assumptions in our analytical methods and thus, it would be misleading to attribute any functional meaning to them.

One particular confound that makes this issue of interpretations worse is when the observations of neural activity roughly lie along a surface, or manifold, which is non-linear in the state space. We show a schematic of such a manifold in Fig. 1.4 (adapted from (Humphries, 2020)). The neural manifold (in brown) can take, e.g. a saddle shape, in which case it occupies a three-dimensional space, or its embedding dimensionality is three. However, the minimum number of variables needed to describe this manifold, or its intrinsic dimensionality (Camastra, 2003), is only two.

A confound thus arises since the manifold can equally be described in two different ways, using either the embedding dimensions or the intrinsic dimensions and our methods may yield either of the descriptions. This consequently leads to the question of how do we tease apart these descriptions in order to reach correct interpretations of the data, and therefrom, come to conclusions of any underlying principles that the brain might be using.

In this thesis, we are interested in understanding the implications of the above confound due to the non-linearity of the manifold when interpreting the results of commonly used linear dimensionality reduction methods. In particular, we ask whether there are some intrinsic non-linearities along the neural manifold that would persist across datasets, irrespective of the task or stimulus and if so, can we characterize their effects on the results of these methods? Ultimately, we hope that once such confounds are elucidated, interpreting the results of dimensionality reduction will be clearer, which will get us a step closer in understanding the brain.

1.3 Organization of the thesis

In Chapter 2 of this thesis, we analyse a few example datasets of neural population recordings using some standard linear dimensionality reduction methods, e.g. PCA. We show that while some results of the analysis can immediately be understood and interpreted with regards to the experimental paradigm, several additional components are also observed (similar to PFC dataset example we considered earlier in Fig. 1.2). Interestingly though, we show that among these additional components, several display a characteristic geometry and these latter components consistently appear across datasets. For reasons that will come clearer later, we will refer to them as higher-order components (HOCs). However, it is still an open question how these HOCs emerge, what their geometry tells us about the data, and what they mean.

In Chapter 3, we argue that these HOCs do not reflect any underlying 'true' latent signals in the data, but rather appear as a reflection of the characteristic shape that the neural manifold takes. We hypothesize that this characteristic shape largely arises due to a well-known non-linearity — individual neuronal activity is constrained to be non-negative — under two crucial assumptions: (1) readouts of population activity are linear and low-dimensional and (2) population activity is limited for energy efficiency reasons. When the manifold is then described using the (linear) axes of the space in which it is embedded (e.g. as in Fig. 1.4), then

these HOCs are bound to appear. Importantly though, they have no functional meaning with regards to the neural circuit, since the true underlying signals reside in a lower-dimensional (or intrinsic) subspace.

In Chapter 4, we propose that we can get closer to the true latent signals of neural population data simply by incorporating the non-negativity constraints in our analytical methods. In this endeavour, we build a set of dimensionality reduction methods that, similar to linear dimensionality reduction methods, assume that the latent signals should be obtained through a linear mapping, but goes further by enforcing these constraints in the model predictions. Upon validating our methods on both simulated and real data, we show that indeed, incorporating these constraints allow us to find a more succinct description of the data compared to a standard linear dimensionality reduction method, namely principal component analysis.

However, it is possible that the neural manifold exhibits some additional non-linearity, besides the non-negativity constraints. We hypothesize in the rest of the thesis, that this additional non-linearity is defined according to the computations that can be done by the network. However, to understand the implications of network computations on the manifold, the computations must first be understood. In Chapter 5, we complement this thesis by addressing, within a novel framework inspired from convex optimisation theory, the computations that can be done by neural networks of biophysical spiking neurons. We show that such networks can fundamentally compute convex input-output functions, and interestingly the resulting networks can display commonly observed biological features such as asynchronous and irregular spike trains, robustness to perturbations, among others.

Chapter 2

Higher-order principal components from neural data

2.1 Introduction

Nowadays, large-scale recordings of neural populations are more widely carried out in many experimental labs yielding ever more complex datasets. Thus, making sense of these data is becoming increasingly important. One approach has been to analyse individual recorded neurons, but this has proved to be quite challenging as soon as hundreds of neurons are considered. Even in simple experimental task paradigms, for example, individual neurons often display highly heterogeneous response patterns (Machens et al., 2010; Laurent, 2002; Brody et al., 2003) which has been a big hurdle to interpret the data (see 1). An alternative approach has been to leverage the fact that the recorded neurons belong to an underlying common network and thus, must share some features across the recorded ensemble. The goal then, has been to extract these shared features. One common way to do so is through the use of dimensionality reduction methods which have yielded key insights on what several brain areas might be representing (Cunningham and Yu, 2014; Keemink and Machens, 2019). These methods try to find a lower-dimensional set of variables, which may not be directly observed, that could explain the data. The hope is that these variables can more easily be interpreted and thus, yield better insights on the population recordings.

However, such interpretations can be fraught with difficulties. Most commonly, linear methods such as principal component analysis (PCA) are used, partly due to their ease of implementation and minimal underlying assumptions. These methods work well in capturing the linear manifolds where most variance in the data reside.

At the same time, they usually display a tail of components and intriguingly, several of them often appear as higher-order functions, e.g. higher-order polynomials, of some other component. As a result, we refer to them as 'higher-order components'. Nonetheless, a clear understanding on the occurrence of these higher-order components is still missing thus, making interpretation of these results ambiguous.

In this chapter, we analyse three datasets to show the occurrence of these higher-order components in different brain areas. Since we will use linear dimensionality reduction methods here, we first start in section 2.2 with a brief overview of the principles underlying many such methods and then, consider its predictions on these datasets in section 2.3. In particular, we will show that these methods not only extract low-dimensional population structures that we interpret but also, many additional components, some of which resemble higher-order components. Finally, we discuss in section 2.4 the implications of these results in our search for neural mechanisms.

Acknowledgements. This chapter is a result of several collaborations. The author thanks the CRCNS database for providing the monkey PFC data (Romo et al., 2016), the Renart lab for providing the rat's primary auditory cortex (A1) data and the Kiani lab for providing the monkey lateral intraparietal (LIP) data. The analysis on the LIP data is a result of a fruitful collaboration with the Kiani lab, in particular, with Gouki Okazawa. He carried out the Canonical Correlation Analysis (CCA), which the author replicates in this chapter. This analysis is also part of a preprint (Okazawa et al., 2021).

2.2 Principles underlying linear dimensionality reduction methods

One condition for dimensionality reduction methods to work is that the N recorded neurons in the population covary according to a smaller number, $M < N$, of explanatory variables. These explanatory variables are not directly observed, hence are often termed latent variables, and can be some common inputs or factors that drive the network of neurons. A main problem that the methods then address is how to find and extract these variables. To do so, each method sets a specific objective supported by some underlying assumptions.

We focus throughout this chapter on the class of linear dimensionality reduction methods. A common assumption these methods make is that the latent variables, $\mathbf{z}$,

can be extracted by taking a weighted linear combination of the observed population activity, $\mathbf{r}$, or mathematically,

$$\hat{\mathbf{z}} = \mathbf{D}\mathbf{r} \tag{2.1}$$

where $\mathbf{D}$ is an $M \times N$ matrix of weights that describes how much each individual neuron contribute to the latent variable. Several studies have demonstrated that such a simple linear mapping is often sufficient to capture the stimulus set and other task parameters present during the experiment as well as both elicited and spontaneous animal behaviour (Kobak et al., 2019; Cowley et al., 2017; Stringer et al., 2019b; Churchland et al., 2012). Also, it has enabled to identify important dynamical structures in the neural populations that are experimentally relevant (Kobak et al., 2019; Cowley et al., 2017; Stringer et al., 2019b; Churchland et al., 2012). This suggests that this assumption of linearity could be an actual mechanism for downstream areas of the brain to decode important information coming from upstream areas.

A complementary view of dimensionality reduction methods is to find a set of latent variables that best reconstruct the data. Many linear methods, e.g, PCA, demixed Principal Component Analysis (dPCA) (Kobak et al., 2016; Brendel et al., 2011), factor analysis (FA), again assume that this mapping is linear. In other words, the data is reconstructed as a weighted linear combination of the latent variables, $\mathbf{z}$, or mathematically,

$$\hat{\mathbf{r}} = \mathbf{F}\mathbf{z} \tag{2.2}$$

where F is a skinny $N \times M$ matrix that contains the coupling weights of the latent variables to the reconstructed neurons.

These two views can be brought together within an autoencoder framework that re-codes its inputs to give an alternative representation. In the context here, the autoencoder would take as input the observed population activity, $\mathbf{r}$, produce an under-complete internal representation (or latent representation), in a hidden layer with a smaller number of variables, and reconstructs the population activity $\hat{\mathbf{r}}$ as its output. The goal would then be to keep the reconstructed data, given the bottleneck, somewhat faithful to the observed activity, which can achieved by minimizing some loss function, e.g. the squared-error loss, $\|\mathbf{r} - \hat{\mathbf{r}}\|_2^2$, used in PCA, with respect to the coupling weights. We illustrate such an autoencoder with linear mappings in Fig. 2.1A.

Nonetheless, the linear methods can differ by adding extra assumptions distinct to each method. PCA, for example, finds a lower dimensional description of the

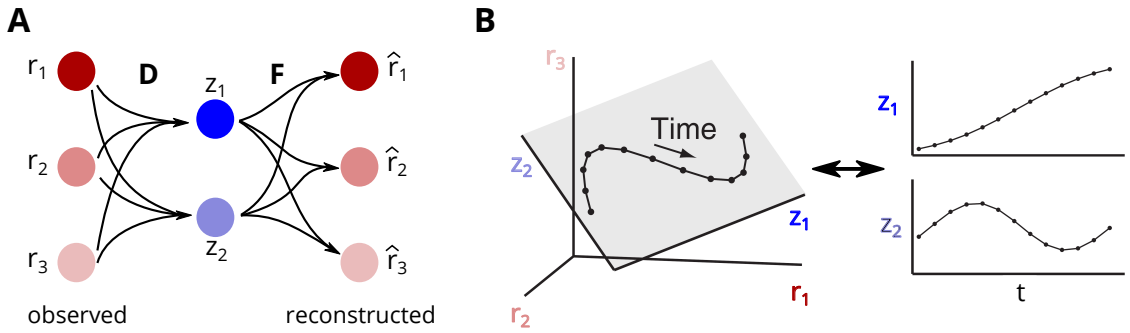

Figure 2.1: Illustration of linear dimensionality reduction methods for three neurons and two latent variables. (A) A linear autoencoder that takes a sample input of observed neuronal activities (r_1, r_2, r_3), maps it linearly (through weights $\mathbf{D}$) onto a bottleneck with latent variables (z_1, z_2) and finally reconstructs as its output a linear estimate $(\hat{r}_1, \hat{r}_2, \hat{r}_3)$ of the input (through weights $\mathbf{F}$). The sample can now be described according to (z_1, z_2). (B) This panel is adapted from Cunningham and Yu, 2014. The black dots are the samples fed to the autoencoder in (A) at different time points. They trace out a trajectory in the neural space but effectively only visit a plane (grey) in that 3D space. Thus, the samples can be equivalently be described according to a new coordinate system (z_1, z_2) that describes the plane. The coordinates along each of the two axis for different samples now describe the evolution of the latent variables. In this illustration, the linear maps are sufficient to perfectly reconstruct the data.

data in an unsupervised way while assuming that the latent variables should be uncorrelated and explain maximum variance in the data. It thus, reconstructs the data by setting $\mathbf{F} = \mathbf{D}^\top$ and assumes an orthonormal basis, i.e. $\mathbf{D}\mathbf{D}^\top = \mathbf{I}$, for the latent variables. But, methods can also be designed to find axes that best align with respect to individual task parameters. dPCA is one such method that maximizes variance related to task parameters, but is not constrained to find orthogonal axes. Thus, depending on the purpose of the study, the method needs to be chosen accordingly but, overall, linear dimensionality reduction methods will find the best linear manifold that describes the data according to some objectives.

2.3 Interpreting latent variables predicted by linear dimensionality reduction methods

In this section, we will analyse three datasets using linear dimensionality reduction methods. We will show that the above principles common to these methods yield low-dimensional descriptions of the data and we can interpret some of the predicted latent variables or components with respect to the parameters present in the exper-

imental paradigm. However, we will show that these variables only capture a small proportion of the total variance in the data while the other variables cannot be easily interpreted. In particular, several of the latter variables appear as higher-order functions of the interpretable latent variables. This consequently raises the question as to how this tail of components appears and how to interpret those higher-order components.

2.3.1 Population recordings in primary auditory cortex of rats

The first dataset that we analyse comes from a study by Kobak et al. (2019). There, the authors investigated how the neural representations for a simple auditory stimulus are affected by the global dynamics of the brain or 'brain states' which can vary along a continuum. At its lower end (or inactive states), the population activity exhibits periods of firing and of silence (up and down states, respectively) while at the other end (active states), the neural activity is more tonic and asynchronous. The authors were particularly interested in understanding how the geometry of the representations (organisation of the signal and noise subspaces) changes as a function of the brain states.

In their study, the authors evoked different brain states by administering urethane anaesthetics (Clement et al., 2008) to rats. At the same time, they recorded neurons simultaneously in area A1 of the auditory cortex while the animals were presented with noise-bursts played via miniature headphones (Pardo-Vazquez et al., 2019). See the cartoon in Fig. 2.2A(i) as an illustration of the experimental paradigm. The stimulus was two-dimensional, parameterized according to:

- **Inter-aural Level Difference (ILD)** - the difference in sound intensity in dB between the two ears, also thought as the cue used by rodents for sound localization. The ILD was varied over 12 values $[\pm 1.5, \pm 3, \pm 4, \pm 6, \pm 12, \pm 20$ db$]$.
- **Absolute Binaural Level (ABL)** - the arithmetic mean of sound intensities at the right and left ears. This was changed over the values $[20, 40, 60$ db$]$.

Fig. 2.2A(ii) shows the 36 experimental conditions (12 x 3) that were presented to the animal with noise bursts lasting 150 ms separated by ~ 850 ms.

Here, we analyse, using PCA, one exemplary recording session where the brain activity was in the active state. During this brain state, as Kobak et al. (2019)

pointed out, the signal plane, the principal noise axis and the axis along which the global population activity fluctuates (global axis) tend to become orthogonal to each other (Fig. 2.2(iii)), which is favourable for PCA.

In this session, 114 neurons were recorded simultaneously and applying PCA allows us to find a lower-dimensional description of the population activity. After smoothing the spike trains to reflect instantaneous firing rates, we computed the peri-stimulus histograms (PSTHs) by averaging the data over trials for each stimulus condition. We further chose a time window of 350ms from the onset of stimulus for this analysis. Since the temporal activity was approximately flat for each condition, we averaged it over time which resulted in 36 mean stimulus responses. We then apply PCA to the mean responses to extract a low-dimensional description of the population activity.

The first two principal components that this analysis yields, capture quite well the grid structure of the stimulus presented to the animal as shown in Fig. 2.2B. Indeed, we see that projections on the first principal axis (PC1) results in a separation of the different sized dots corresponding to the different ABLs. Conversely, PC2 separates the different colored dots, thus reflecting the ILDs. Note that this plane spanned by the first two principal axes is analogous to the signal plane that we described earlier (Fig. 2.2A(iii).

However, if we consider how much variance these two dimensions explain, they capture only around 55% of the total variance while the rest is explained by a tail of components as we see in Fig. 2.2C. Interestingly, by considering two additional components, PC3 and PC4, we see that the activity along those axes form a curvature with respect to the ILDs for each ABL (see Fig. 2.2D; a second-order polynomial is fitted to the data points to emphasize the curvature). Particularly, we see a strong curvature at low mean intensity (ABL = 20 dB) for PC3 while PC4 shows strong curvature at high mean intensity (ABL = 60 dB). Thus, PC3 and PC4 appear as 'higher-order' components in this analysis.

In summary, although the first two PCs already reflect the stimulus structure in this experiment, PCA still predicts additional components that display curvatures that resemble high-order functions of what previous PCs capture. This suggests that the neural manifold must also have a similar curvature that these additional components are reflecting. Yet, it is unclear as to what causes this curvature, which makes interpretations of this PCA analysis ambiguous. A possibility could be that these components reflect some underlying biological mechanisms, which need to be

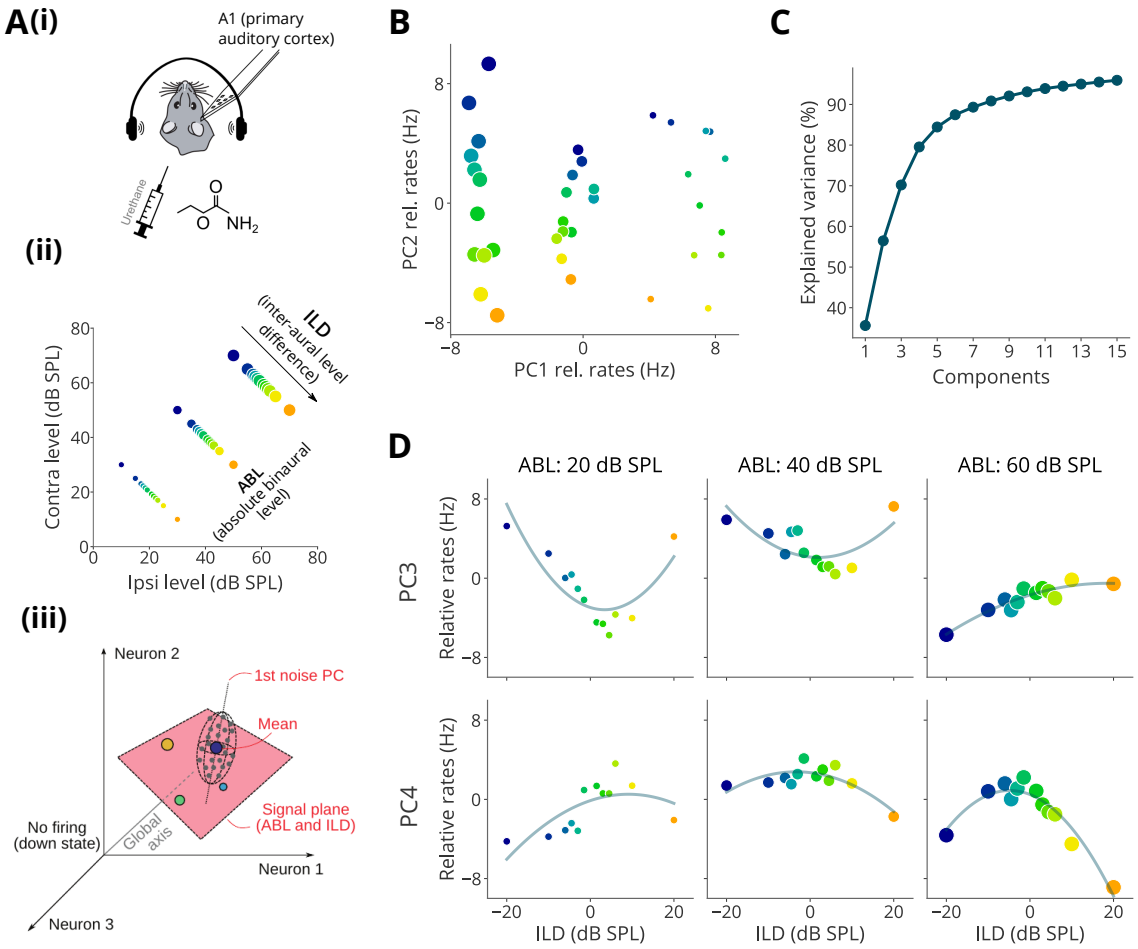

Figure 2.2: PCA applied to an exemplary recording session of neurons in the primary auditory cortex (A1) of rats (Kobak et al., 2019) anaesthetised under urethane which causes spontaneous transitions in global population dynamics (Clement et al., 2008). In this session, the population was in the active state (tonic and asynchronous neural activities). (A)(i) Cartoon of the experimental paradigm. (ii) The stimulus conditions were presented in a grid structure. The colors correspond to ILD and the dot sizes correspond to ABL. (iii) Schematic of the geometric analysis using PCA in (Kobak et al., 2019). The salient axes are the global axis (determines fluctuations in population activity during up-down transitions), signal plane (passes through mean stimulus responses) and noise axis (direction where the noise cloud stretches most). We extract this signal plane by applying PCA on the mean responses. (B) The first two PCs obtained reflect the grid-like structure of the stimulus set. PC1 captures the ABL (separation of dot sizes) while PC2 reflects the ILD (separation of colours). (C) Cumulative variance explained by the first 15 principal components. The first two PCs capture only around 55% of total variance. (D) Plotting the third and forth PCs (rows) against ILDs for each ABL (columns) shows curvatures for each ABL. We fitted a second-order polynomial - pale blue trace to emphasize the shape of the curvature. These PCs resemble second-order components of the first two PCs.

deciphered.

2.3.2 Single units recordings in prefrontal cortex of monkeys

We next reconsider a dataset of single unit recordings from the PFC of monkeys (Chapter 1, Fig. 1.2) to show that comparable higher-order components persist in other brain areas. As a reminder, this dataset was recorded while the animals performed a somatosensory working memory task (Romo et al., 1999; Brody et al., 2003). In this task, monkeys received a first vibrotactile stimulus of frequency, F1, at their fingertip followed by a second one of frequency, F2, after a delay period. They then had to discriminate whether the first frequency was stronger than the second one, i.e. F1 $>$ F2, or not. The monkeys reported their decision by pressing one of the two available buttons as shown in the cartoon in Fig. 2.3A. Here, we analyse the neural recordings from two monkeys ('RR014' and 'RR015') and for both of them, the delay period was 3 seconds. Also, the stimulus F1 was in the set $[10, 14, 18, 24, 30, 34$ Hz$]$ for both monkeys. Pooling across them resulted in 1325 recorded units.

We are interested in this analysis, similarly to (Machens et al., 2010), on the neural representation of the stimulus F1 during the delay period. However, in this task, individual neurons displayed highly heterogeneous responses (see Fig. 1.2) with mixed selectivity to the stimulus and other factors not experimentally controlled. Applying PCA on this data does not yield low-dimensional representations that can be interpreted with regards to the stimulus, unlike the previous dataset. As a result, we resort to an alternative dimensionality reduction method, namely demixed principal component analysis (dPCA) (Kobak et al., 2016), which similar to PCA, is a linear method, but has the additional constraint of predicting components that separate dependencies on task parameters in the population activity.

The population also encodes the decision information but, since we are only considering the delay period, hence before the arrival of F2, no informed decision can be made. So, we do not need to demix the decision related activity in this time window. The dPCA results we will show closely match those reported in Kobak et al. (2016) where the algorithm was developed and applied to several datasets, including this one. However, the purpose of this present analysis is different.

We observe that the population activity is dominated by the condition-independent variables; they explain most of the variance in the data (Fig. 2.3C(i)). We can also

see this by looking at the population firing rate over time, obtained by averaging the PSTHs over all recorded units: in Fig. 2.3B, we see that the population activity, when conditioned on F1 stimuli (each corresponding to a colored trace) almost overlap, thus showing that other factors and not the stimulus are dominant. However, dPCA still allows us to find a set of decoding axes in the neural space that separate the F1 dependent activity (Fig. 2.3E) from the time-varying condition-independent activity (Fig. 2.3D).

In Fig. 2.3D, E, we plot the three most prominent decoding axes for the condition-independent and stimulus-dependent activity respectively. We observe in Fig. 2.3E(i) that the F1 dependent activity persists during the delay period with F1 tuning clearly shown by the first component, as was previously noted in Machens et al. (2010), Brody et al. (2003), and Barak et al. (2010). We can average over time, here in the interval of 1 to 3 seconds where the activity looks roughly flat for components 1 and 3, to look at the mean stimulus responses. Fig. 2.3E(ii) reflects again the tuning of the activity to F1 during the delay period. The first components reflects clearly the linear tuning of the population with F1. Interestingly, however, we now observe that the third component exhibits a curvature as a function of F1 stimuli; a second-order polynomial was fitted to the mean responses to emphasize this.

We note that in order to solve this task, the monkey needs to memorize the F1 stimulus during the delay period until the F2 stimulus appears, upon which it can make the required comparison. Thus, having an accurate neural representation of F1 should be enough and indeed, this is well captured by the F1 tuning of the first demixed component in Fig. 2.3E. With this decoding axis, F1 information can then be readout by a downstream area to make the correct decision. Yet, similar to the previous dataset, we see that a tail of components is observed (Fig. 2.3C). It is again a puzzle as to what these additional components are representing, in particular, those that appear as higher-order functions.

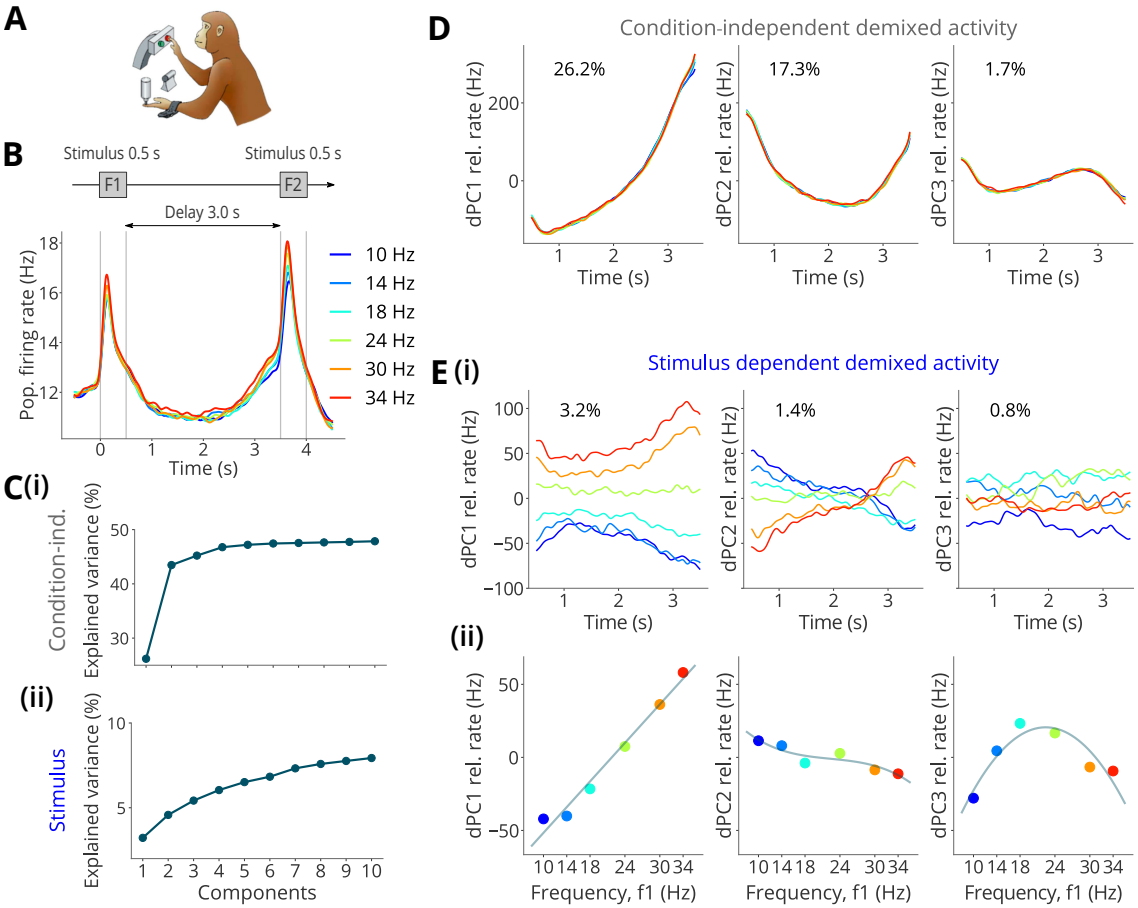

Figure 2.3: Demixed PCA (Kobak et al., 2016) applied to recordings of PFC neurons while monkeys performed a somatosensory working memory task (Romo et al., 1999). (A) Cartoon of the experimental paradigm adapted from (Romo and Salinas, 2003). (B) Population firing rate obtained by averaging the PSTHs over neurons for each F1 stimulus. Grey vertical lines correspond to the different time events in the task. The data was analysed during the delay period. The legend shows the F1 stimuli in this dataset with corresponding colour code used across the figure. (C)(i) Cumulative variance explained by the first ten components that demix condition-independent activity. (ii) Same as (i) but the components now demix F1-dependent activity. (D) First three condition-independent components across time obtained by projecting the PSTHs on the respective dPCA decoding axes. The traces overlap each other to show the condition independence. (E)(i) First three stimulus-dependent components across time. (ii) Same as (i), but now averaging the above components over a time window ranging from 1 to 3 seconds. The mean responses in coloured dots are plotted against F1. dPC1 shows the persistence of F1 tuning during the delay period which is required to solve the task. dPC3 forms a curvature to which we fitted a second-order polynomial - pale blue trace. It resembles a higher-order component given the linear tuning of the first component.

2.3.3 Single units recordings in lateral intraparietal area of monkeys

Finally, we analyse a dataset of 70 neurons recorded in the LIP of two monkeys while they performed a motion discrimination task (Kiani and Shadlen, 2009). In this task, monkeys viewed randomly moving dots and were required to report the direction of perceived motion. We illustrate this task in Fig. 2.4A. A trial started when a monkey fixated at a central point on the screen (middle red dot) after which two possible choice targets appeared; one in the response field (RF) (grey shaded region), of the neuron under study which we denote as T_{in} and the other, in the opposite hemifield, denoted as T_{out}. Then, the random-dot motion stimulus was presented, centered at the fixation point, for a variable duration (100 to 900ms) followed by a delay period (1200-1800ms) where the monkey continued fixation. Thereafter, the fixation point was extinguished, which instructed the monkeys (as a go cue) to report its decision by making a saccade to one of the direction-choice targets. The monkey was rewarded upon a correct choice. The net motion of dots was towards one of the targets and the difficulty of the stimulus was controlled by changing the percentage of coherently moving dots (or, motion coherence) which was chosen from the following values $[0, \pm 1.6, \pm 3.2, \pm 6.4, \pm 12.8, \pm 25.6, \pm 51.2\%]$. The sign of coherence indicates the direction of motion; the T_{in} choice was associated with positive coherence.

Such a motion task has extensively been used to study the neural mechanism of perceptual decision-making, e.g. (Britten et al., 1992; Gold and Shadlen, 2007; Shadlen and Newsome, 2001; Roitman and Shadlen, 2002). In the LIP area, it has been shown that when the motion of the dots supports the T_{in} target, the neuron often displays ramping activity which suggests that it might be integrating evidence into a decision variable (DV). However, in these studies, the neural population is often sub-sampled with selection biases from the experimenter and thus might not provide an overall view of the population. For example, the LIP neurons that are studied are often those that show significant persistent activity during the delay period of a memory-guided saccade task (Colby and Goldberg, 1999; Gnadt and Andersen, 1988; Shadlen and Newsome, 2001). In that task, a target is briefly presented in the visual field of the monkey and after a delay period, the monkey is required to saccade to the remembered position of the target. The target position is then varied across trials and the RF of the neuron is subsequently defined as the position in the visual field that causes the neuron to be persistently active during the delay period. It is this RF that is then set as one of the choice target (T_{in}) in the

motion discrimination task. However, it is unclear how the response of the neuron would change if the motion of dots points in other directions rather than towards the RF.

Since we are mostly interested in describing population responses here, having a homogeneous sampling of the population is crucial to reach an unbiased view of what it is representing. We go one step in this direction by constructing a surrogate dataset by assuming that whenever, for a given LIP neuron under study, the motion of the dots points towards T_{out}, there will be another neuron, not recorded, whose RF will be at that target. Thus, this latter neuron will exhibit the opposite responses, which we approximate using the reversed responses of the recorded neuron conditioned on motion coherence. For concreteness, suppose neuron 1 fires at $(x_1, \ldots, x_k)$ Hz for the k signed stimulus conditions, then we assume there is an unrecorded neuron 71 that would respond as $(x_k, \ldots, x_1)$ for the same k stimulus conditions. In other words, we build this surrogate dataset by simply flipping these conditional responses for all neurons and together with the original dataset, we analyse a population of 140 neurons.

Similar to the previous datasets, we apply linear dimensionality reduction methods to find latent structures in the data. Proceeding as before, we obtain the trial-averaged neural responses for each neuron for each motion coherence. To avoid confounded representations with mixtures of both correct and error choices, in the following analysis we select trials with correct choices only. Following past studies (Shadlen and Newsome, 2001; Kiani and Shadlen, 2009), we group the motion coherences according to 0-3.2%, 6.4%, 12.8%, 25.6%, 51.2% to give five strength levels (thus, 10 signed stimulus conditions) that we color-coded in Fig. 2.4. At 0% coherence, the correct target was assigned randomly. Fig. 2.4B shows the populations average PSTHs after stimulus onset. In the inset, we show the population firing rate of the original dataset (70 neurons) where we see the typical ordering of firing rates of LIP neurons as previously reported in (Roitman and Shadlen, 2002; Shadlen and Newsome, 2001). The firing rates of the neurons are high for stimuli supporting the T_{in} choice (solid trace) and low for stimuli in favour of the T_{out} choice (dashed trace). Due to how we build our dataset, this split according to choices disappears upon averaging over all 140 neurons; the solid and dashed traces for each stimulus strength tend to overlap. Nonetheless, there is a general temporal trend in the population firing that is condition-independent.

Here, we analyse this data in a 250-550 ms time window. To separate the condition-independent activity from the stimulus-dependent activity, we again apply

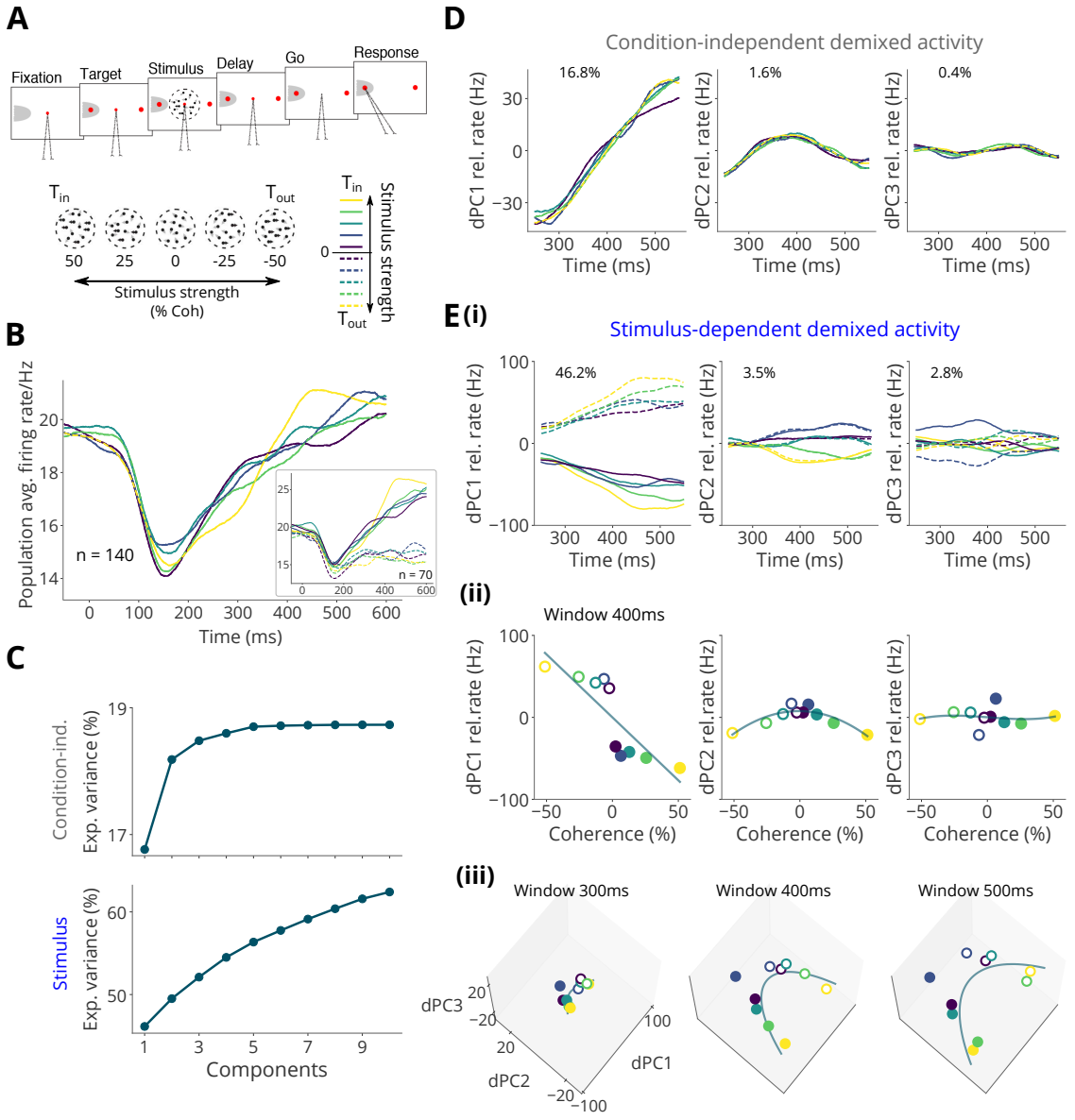

Figure 2.4: Demixed PCA applied to recordings of LIP neurons while monkeys performed a motion discrimination task (Kiani and Shadlen, 2009). (A) Cartoon of the task adapted from the original study. In the legend, solid lines correspond to T_{in} and dashed lines to T_{out}; the colors are the different stimulus strengths. (B) Population average firing rates of original and surrogate dataset with correct trials; the original data only in inset where the responses for T_{in} and T_{out} gradually diverge due to the task structure. (C) Cumulative explained variance by the first ten components that demix condition-independent and stimulus-dependent activity, respectively. (D) First three demixed principal components for condition-independent activity. (E)(i) Stimulus-dependent dPCs. (ii) Same as (i), now averaged over a time window 350-450ms. The dots are the mean responses for correct choices to different stimulus strengths. dPC1 captures the decision variable. dPC2 shows a curvature in the neural manifold, fitted with a second-order polynomial. (iii) The curvature is more prominent in a 3D space with the top three components. The fitted trajectory is obtained by fitting polynomials to each component as in (ii) but for different time windows.

dPCA. Fig. 2.4D shows the three most prominent decoding axes for the condition-independent activity (the colored traces almost overlap with each other) and similarly, Fig. 2.4E shows the dominant decoding axes for the stimulus.

We observe in this time window, that the stimulus-dependent activity dominates the population activity as measured by the amount of variance they explain (Fig. 2.4C). The first component in Fig. 2.4E, in particular, shows that this axis possibly captures choice information as the dashed and solid lines diverge over time. In supplementary figure 2.7A, we verified that the traces overlap at the onset of stimulus (t=0 ms) by projecting the data with a wider time window on the the axes obtained from this analysis. This is expected since there is no evidence to make an informed decision at that time.

We also see that the colored traces for different motion coherences diverge over time, suggesting that this axis is also capturing the DV. This is made clearer by looking at the time-averaged activity as a function of motion strength (Fig. 2.4E(ii)). Here, we averaged over a 100ms time window, centered at 400ms. We see that the first component, upon averaging, displays the separation of choice (separation of empty dots for negative coherences and full dots for positive coherences), but there is also a linear gradation of the activity conditional on stimulus strength, thus reflecting the DV, or amount of evidence in the stimulus, required to solve the task. This gradation becomes more prominent over time (see supplementary figure 2.7B) which suggests the integration of evidence.

Interestingly, however, we see that several additional components can be observed upon this dPCA analysis. In particular, the second stimulus-dependent dPC reflects a curvature in the neural manifold (Fig. 2.4E(ii); a second-order polynomial was fitted to the mean responses). This curvature is more salient when we look at the activity in a three-dimensional dPC space (see Fig. 2.4E(iii); polynomials fitted to the projected data). We observe that the major axis of the curvature seems to distinguish the difficulty associated with the stimulus — the easy stimuli (positive and negative coherence) appear towards the ends of the fitted polynomial while the harder stimuli get closer at the peak of the curvature. One may then posit that the population is explicitly encoding the stimulus difficulty in a curved manifold which may have some functional purpose in solving the task.

One possibility could be that the neural representation along a curved manifold allows to compute both the choice and the confidence associated with it since both DV and stimulus difficulty could, in principle, be readout by a downstream area

according to two distinct axes (e.g. decoding axes 1 and 2 from dPCA for stimulus-dependent activity). However, following an analysis done by Gouki Okazawa in the laboratory of Roozbeh Kiani, we will next show that intriguingly, this curvature does not bear on the confidence of the monkey.

In this motion discrimination task, the monkey also had the option on a random half of the trials to terminate a trial by making a saccade to a third target (Kiani and Shadlen, 2009). A smaller reward ($\approx 80\%$ of the reward of a correct choice) was guaranteed upon this choice. The trial would start as before and during the delay period, at least 500ms after the stimulus viewing, this 'sure target', (T_s), would appear. We illustrate this in Fig. 2.5A. As previously demonstrated (Kiani and Shadlen, 2009), the monkey would choose T_s based on certainty, choosing it more frequently on difficult trials. In other words, whenever it chose T_s, it was an indication of low confidence. Also, from the neural responses of LIP neurons, it has been shown that one could predict the monkey's T_s choices. So, if encoding stimulus difficulty explicitly had a functional role in the computation for confidence associated with a choice, then projecting the LIP responses on an axis representing stimulus difficulty (major axis of the curvature, here) would, in principle, be predictive of T_s choices.

Gouki tested whether this was the case by first identifying axes in the state space of $N = 70$ neurons (original dataset) that best captured the DV and the stimulus difficulty by doing a canonical correlation analysis (CCA) (Hotelling, 1992). CCA allowed to find a pair of axes, one most correlated with the signed stimulus strength ('DV axis') and the other, being most correlated to the unsigned stimulus strength, a proxy for stimulus difficulty ('difficulty axis'). To avoid any artefacts, the analysis was cross-validated whereby, the axes were first obtained using a random half of correct trials and prediction made on the remaining half by projecting the data on these axes (see section 2.5.2 for supplementary details on this analysis). Note that CCA is an alternative way of finding decoding axes with explicit labels, but unlike dPCA the targets for the projections need to be specified (here, signed and unsigned stimulus strengths). We replicate below the results that they obtained.

As seen in Fig. 2.5B, these two axes provide a two-dimensional description of the population data, which captured the curvature in the neural manifold as before (test data was projected on these axes and averaged over a time window 350-450ms after stimulus onset). We now show that projections on the difficulty axis were not predictive of the monkey's confidence. Following the original analysis, we projected the correct trials for all stimulus strengths, and the error and T_s trials for the three

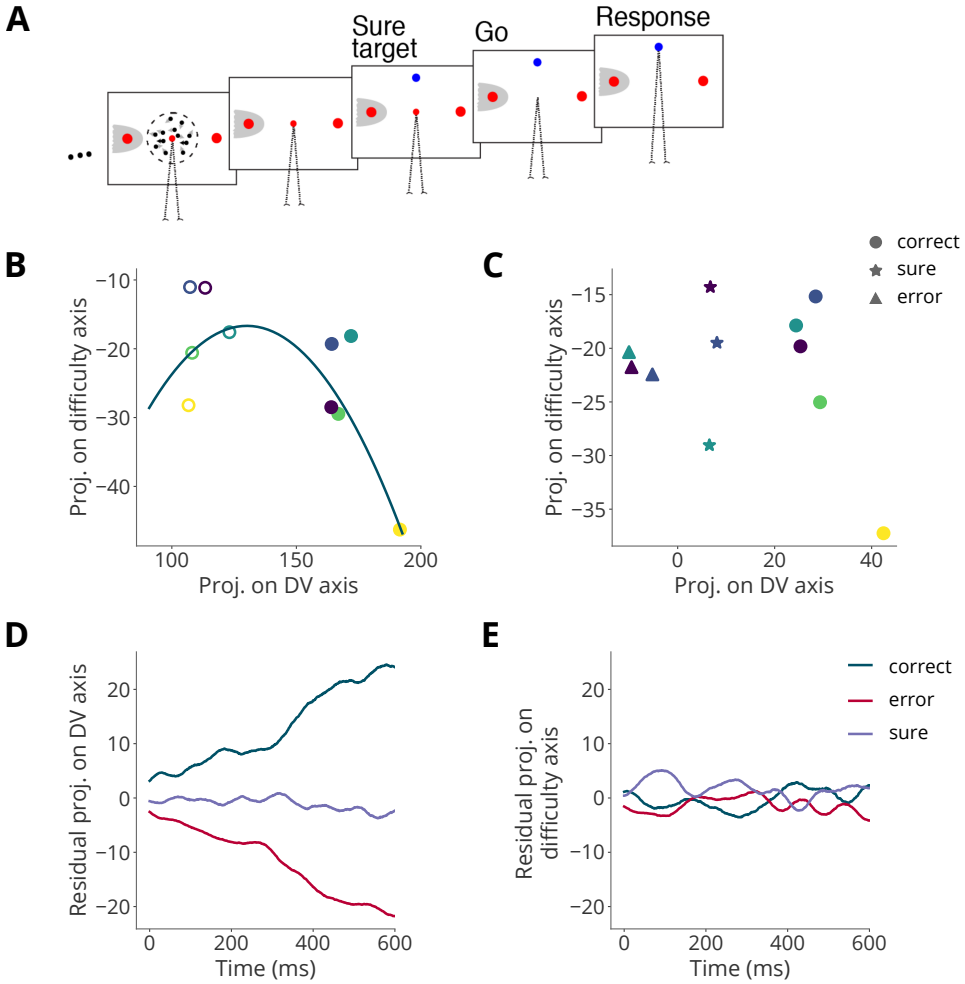

Figure 2.5: Canonical correlation analysis (CCA) on 70 LIP neurons in a motion discrimination task with confidence jugdements. The curvature in the neural manifold was not predictive of confidence. (A) The confidence of the monkey could be assessed in a variant of the task where an additional target, T_s, would appear on a random half of the trials. Upon choosing it, the monkey opted out and received a smaller reward. (B) Population activity projected on the best axes that encode DV and stimulus difficulty. These CCA axes were obtained upon cross-validation and using correct trials only. The plot show the time-averaged projected responses (test data) in a time window 350-450ms after stimulus onset. A second-order polynomial was fitted to the data to emphasize the curvature. The DV axis indicates evidence supporting T_{out} and T_{in}. (C) Projected neural responses for each stimulus strength and choice in the same time window as in (B). The DV axis is redefined here, with values indicating correct and error choices, with more positive values meaning stronger evidence for correct choices. While projections along DV axis separates the choices, projected responses for the low confidence choice (sure target, T_s) are not distinguishable from those of higher confidence choices (T_{in}, T_{out}) along the difficult axis. (D) Residuals of projected population responses projected on the DV axis. A clear separation of the choices is obtained. (E) The residuals, when projected on the difficulty axis, had no marked separations, suggesting that the curvature in the manifold is not predictive of confidence.

weakest stimulus strengths. For these latter stimulus strengths, the error and T_s choices were present in all sessions (for easier stimulus, the monkey hardly made errors or chose T_s). As shown in Fig. 2.5C, distinct projections corresponding to different choices were obtained along the DV axis. However, when projected on the stimulus difficulty axis, no clear separation of low-confidence choices, T_s, from the higher-confidence ones (T_{in} or T_{out}) was obtained (see supplementary details in section 2.5.2).

To further quantify these observations, we projected the residuals for fixed stimulus strength, as in the original study, along the DV and difficulty axis to see if they were predictive of confidence during decision formation. This residual analysis addresses the confound that a change in stimulus strength leads to changes along both DV and stimulus difficult axes due to the curved manifold. For each axis, these residuals were computed by taking the mean of the projected data over the three choices, within each coherence, and subtracting this mean from the projected data for each choice. This removed the effect of stimulus strength which was the major source of co-variation of these axes and as a result, isolated the effects of variations on one axis, while keeping projections on the other axis constant. Finally, the residuals were averaged over the stimulus strengths for each choice. We see in Fig. 2.4D that the residual projections along the DV axis showed a clear separation of the three choices. However, this was not the case when the residuals were projected along the difficulty axis (Fig. 2.4E); the residuals for the sure choice were not distinguishable from those when the animal made correct and incorrect choices, thus reinforcing that this axis does not bear on the animal's confidence.

In summary, we showed that dPCA revealed latent structures in the population data that reflected the integration of evidence which is crucial to make an informed decision in this task. Nonetheless, additional components also appeared which suggest that neural manifold has a curvature that is dependent on the motion strength. In particular, the curvature seems to encode the stimulus difficulty and one could posit that this information could be used for confidence judgements. However, replicating an analysis done by Gouki Okazawa,we showed that, in fact, this curvature does not seem to be relevant for computing the confidence associated with a choice. One possibility could be that the neural mechanism for confidence operates on a different pathway and does not use the information along the difficulty axis. In this case, it remains unclear what functional role does this curvature in the neural manifold serve.

2.4 Discussion

In this chapter, we analysed three datasets (Kobak et al., 2019; Romo et al., 1999; Kiani and Shadlen, 2009), using commonly used linear dimensionality reduction methods, namely PCA, dPCA and CCA. While in the first dataset, PCA was able to find dimensions in the data that could be interpreted with regards to the stimuli, it is not generally case that it will find low-dimensional representations of the data that correspond to the stimulus. We thus used more targeted dimensionality reduction methods, i.e. dPCA and CCA in the other datasets.

We showed that, while these methods successfully extracted low-dimensional structures in the population data that we could interpret, they also predicted a tail of components. Amongst them, some of these components resembled higher-order functions of some other components, and thus we referred to them as 'higher-order' components. This suggests that the true underlying manifold is non-linear. However, a clear understanding of how these higher-order components emerge and what they represent is still missing. We discuss below a few hypotheses.

1. **Distortions in the manifold due to noise.** One hypothesis could be that the noise in neuronal activities is sufficiently structured that it distorts an otherwise linear manifold. However, averaging across trials as we did here should have reduced the effect of the noise. Also, it seems unlikely that such structure in the noise would persist across the datasets to yield similar distortions.

2. **Curvature due to task structure.** One possibility could be that the curvature that these dimensionality reduction methods capture is due to the specific structure in the task or stimulus set that elicits neural responses only in portions of the high-dimensional neural space. As an example, consider an experiment where gratings of different orientations are presented on a screen while neurons in the early visual cortex are recorded. Then, one might expect that the responses of neurons encoding this stimulus set would also display a similar circular structure (see Fig. 2.6). In other words, we would not be measuring neural responses across the whole neural manifold, but only those elicited in a circle due to the structure in the stimulus set. Thus, the overall shape of the manifold may be different.

   To address this, one might consider a multitude of tasks and hope that the elicited responses cover more widely the neural manifold and thus, get a better parametrization of its shape. As a result, the apparent curvature, which arose due to the samples of measured neural activity from a specific task structure,

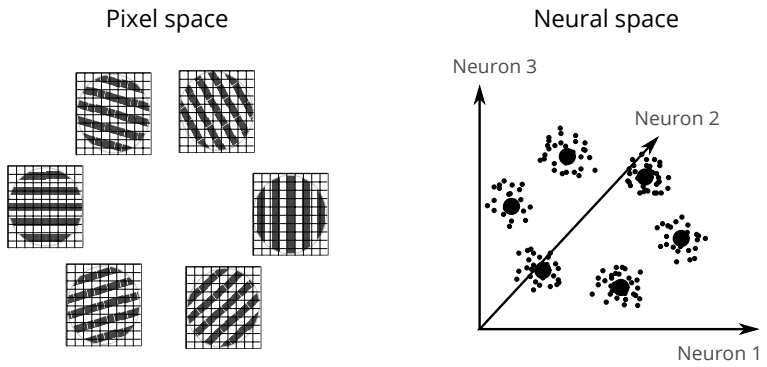

Figure 2.6: Illustration of a curved neural manifold due to a specific stimulus structure. Neurons encoding the orientation of the gratings in the pixel space are likely to have their responses lying on a circular trajectory in the neural space.

might disappear. Yet, as we showed, the higher-order components seem to persist across several tasks, which suggests that curvature may be a feature of the overall neural manifold, and is not stimulus or task specific.

3. **Curvature due to neural computations.** One other hypothesis is that the curvature in the manifold arises due to additional computations that might be required e.g., to generate accurate neural representations or to solve the tasks. The higher-order components would then be reflecting these computations along extra dimensions in the state space.

   However, as we showed in the LIP dataset (Kiani and Shadlen, 2009), the curvature which seemed to reflect stimulus difficulty, was not functional in the computation for confidence. Thus, functional purpose of this curvature remains unclear as the linear DV axis was sufficient for the animal to make a choice. In the PFC dataset (Romo et al., 1999) also, we showed that the stimulus tuning during the delay period, crucial to solve the task, was already captured by a linear component. Yet, additional higher-order components also appeared and their computational roles remain unclear. Given the experimental paradigms in the datasets that we analysed, we believe that computations done by the network might not be the dominant aspect yielding a curved manifold, although we cannot entirely neglect this hypothesis.

4. **Curvature due to coding constraints.** We propose an alternative hypothesis which is that the overall curvature in the neural manifold is due to coding constraints, and is not functionally relevant. One well known constraint underlying neural representations is that individual neuronal activity is constrained to be non-negative. We will show in the next chapter how such a simple non-linearity can change the shape of the manifold if the neural code

is to be energetically efficient and the relevant information is readout linearly. We will also how these lead to higher-order components that resemble those we described here.

2.5 Supplementary details

2.5.1 Preprocessing of neural data

- **A1 dataset in rats** (Kobak et al., 2019). First, we filtered the spike trains with a Gaussian kernel ($\sigma = 10$ ms) and sampling rate of 200 Hz to produce trial-by-trial instantaneous firing rates. By averaging over the trials for each experimental condition, we then computed the peri-stimulus histograms (PSTHs). For the plots in the figure, we analysed the PSTHs over a time window of 350ms after stimulus onset and we compute the mean responses by averaging over time.

- **PFC dataset in monkeys** (Romo et al., 1999). In this analysis, we used the same pre-processing steps as Kobak et al. (2016). Recording sessions were selected such that all the six F1 frequencies were present and only correct trials were analysed. Also, there was a selection of neurons such that for each neuron there was a minimum of 5 trials for each condition. Neurons that had mean firing rates above 50Hz were excluded to avoid biasing the variance-based analysis. This resulted in 1325 neurons. Note that this number differs from Kobak et al. (2016) as they also considered the decision of the monkey in this selection process. The spike trains were filtered with a Gaussian kernel ($\sigma = 50$ ms) and sampling rate of 100 Hz to produce instantaneous firing rates. PSTHs were computed by averaging over trials for each frequency.

- **LIP dataset in monkeys** (Kiani and Shadlen, 2009). The spike trains were filtered using a Gaussian kernel ($\sigma = 25$ ms) and window size of 100 ms to produce instantaneous firing rates. For the dPCA analysis, these were averaged over correct trials for each stimulus strength to yield PSTHs. Due to the small number of neurons in this dataset, we did not pre-select neurons as in the previous dataset, but the results were obtained after cross-validation using the same approach as in (Kobak et al., 2016). dPCA was applied on a dataset of 140 neurons (original+ surrogate). For the CCA analysis, following the original analysis, we computed PSTHs using spike counts for the 70 neurons of the original dataset.

2.5.2 Details on CCA for the LIP dataset

CCA is a linear dimensionality reduction method that, given two datasets, **R** and **P**, finds low-dimensional mappings **RA** and **PB** such that the correlation between these mappings is maximized.

To find these axes, a matrix of task parameters, **P**, that approximated DV and stimulus difficulty was first designed. Since the DV for a fixed stimulus duration is proportional to the signed stimulus strength, s, while the stimulus difficulty is defined as the negative absolute value of the stimulus strength, $-|s|$ (negative values since motion strength 51.2% is easier than motion strength, say, 25.6%), **P** was built using these values. The other matrix **R** was the PSTHs of the 70 LIP neurons but, restricted to a smaller time window (350-450ms after stimulus onset). A short time window was chosen in order to allow approximation of DV and stimulus difficulty based on the task parameters; the proportionality constant between stimulus strength and DV changes over time, as the DVs for different stimulus strengths diverge from each other. Finally, a 2D projection of the neural responses that best correlates with the task parameters was computed as $\mathbf{RAB}^{-1}$. Note that a PCA was first done on the PSTHs to attempt denoising the data prior to CCA and in our replication, we kept the first 20 PCs that explained 93.9% of the variance. This choice of number of PCs was not critical for the results obtained (the original analysis tested a range of dimensions from 10 to 40).

Note that in Fig. 2.5B, the DV axis ranges between T_{in} and T_{out} correct choices, with more positive values indicating stronger evidence for T_{in}. In Fig. 2.5C, D, however, since we are interested in comparing the projected responses across choices, the DV axis is aligned with respect to the correct target such that now, the DV axis ranges between correct and error choices. To make this point clearer, consider an example. Suppose the projected neural response along DV axis for x% coherence stimulus is α and that, for $-x$% coherence is β (assuming positive coherence is towards T_{in} target), we averaged this activity together i.e., $(\alpha - \beta)/2$. For α_1 and β_1 being the projected responses along the difficulty axis for x% and $-x$% coherences, respectively, we simply computed the average as $(\alpha_1 + \beta_1)/2$.

2.A Supplementary figure

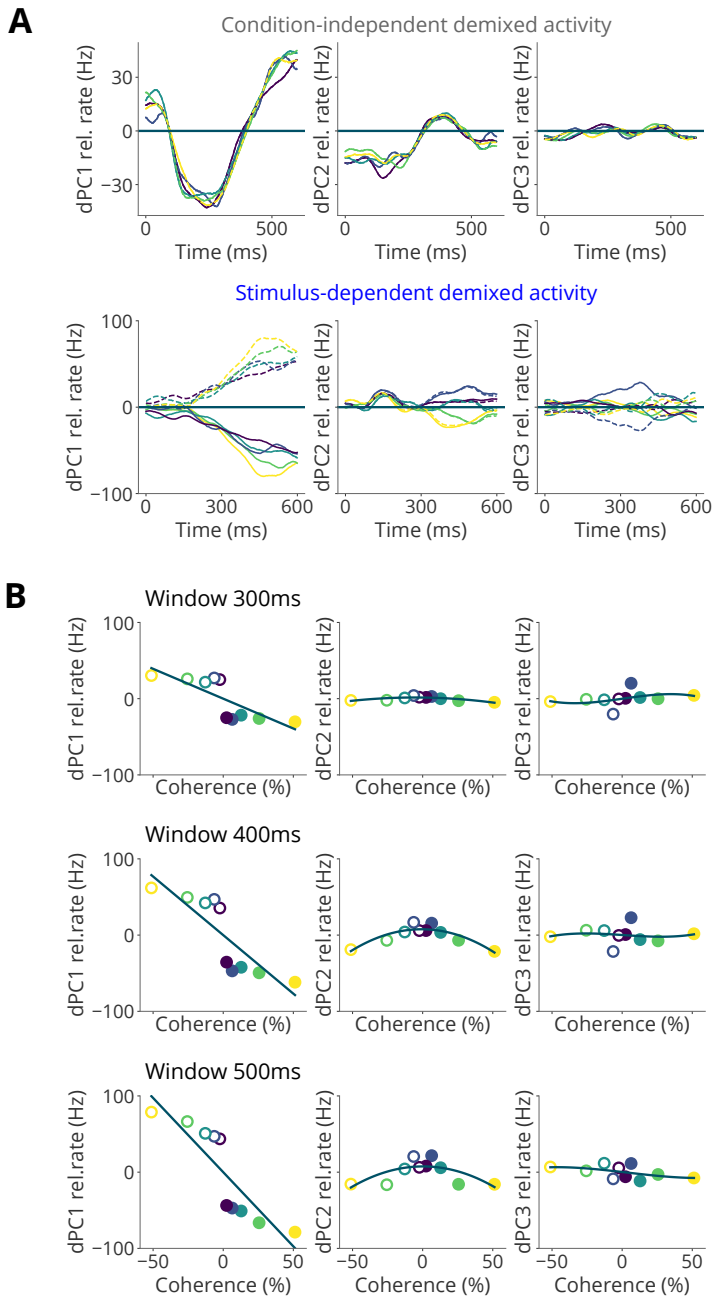

Figure 2.7: Results from demixed PCA on recordings of LIP neurons over different time windows. dPCA was fitted as before over a time window of 250-500ms after stimulus onset. (A) Data over the time window 0-600ms (with 0ms being the onset of the stimulus) is now projected to show the neural responses at stimulus onset for different components. (B) Stimulus-dependent dPCs averaged over different time windows. The first component shows that linear gradation of mean responses conditional on motion coherence becomes more evident over time. The curvature in the manifold as captured by the second component becomes more prominent over time.

Chapter 3

A theory for higher-order principal components

3.1 Introduction

Neurons in the brain communicate mainly through spikes which are discrete events. The quantification of neuronal activity as the number of spikes per unit time or firing rate is therefore, bound to be non-negative. While this simplest form of non-linearity on neuronal activity is well-known, linear dimensionality reduction methods do not explicitly take this into account and its specific effect or importance for these methods is less clear.

In this chapter, we study this effect under two crucial assumptions. First, we assume that the effective readouts from the high-dimensional population activity are linear and are low-dimensional (Assumption 1). This assumption is present in all linear dimensionality reduction methods (see Chapter 2), and the extracted latent variables from these methods would correspond, in some sense, to the readouts. Given the successes of these linear methods in uncovering various insightful population structures in neural data (Cunningham and Yu, 2014; Keemink and Machens, 2019), we hypothesize that linear readouts could be a biologically plausible mechanism employed by the brain. Our second assumption is that overall population activity is limited for reasons of energetic efficiency (Assumption 2). This assumption is motivated by the common observation that spike emissions are metabolically costly (Laughlin, 2001; Attwell and Laughlin, 2001) and thus, need to be economised while relaying maximum information in each spike (Barlow, 1969; Levy and Baxter, 1996).

We start this chapter by revisiting a classic problem in Neuroscience which is how neurons 'encode' relevant information into their firing rates (section 3.2). This will allow us to look at the implications of our assumptions on the neural code. We show that when neural activities are generated under these assumptions, the resulting neural manifold, i.e. the surface in the neural space on which the neural activities lie on, becomes curved so as not to violate the non-negativity constraints. Importantly though, the relevant signals should still be retrieved through a linear readout despite the curvature (Assumption 1). However, when a linear method such as PCA is applied to samples from this manifold, we show that many more principal components than the dimensionality of the readouts are predicted so as to compensate for the curvature in the manifold. We explain this finding geometrically and show that several of these components resemble higher-order functions, e.g. polynomials of increasing order, of some other components. We thus refer to them as higher-order components (HOC) and remark that they resemble the puzzling 'higher-order' components that we found when analysing real data as in Chapter 2. This raises the possibility that these components appear in real data due to coding constraints that bend the neural manifold.

We next turn to numerical simulations to investigate in more depth this effect of the non-negativity constraints on the predictions of PCA (sections 3.4 & 3.5). We consider two neural network models that incorporate our assumptions to simulate the ground truth data and we show, in this case, that the neural manifold is approximated by specific, piecewise-linear surfaces in the high-dimensional neural space, along which the population trajectories move. Then, PCA sometimes extracts the correct low-dimensional linear readouts, but often, also displays a tail of higher-order components due to the kinks in the manifold of piecewise-linear surfaces. Importantly though, these higher-order components do not reflect any of the underlying signals, nor any computations thereupon, and thus do not have any functional meaning. This consequently warrants caution when interpreting the results of linear methods when analysing neural data.

3.2 Coding problem revisited

In this section, we look at the implications of our aforementioned assumptions on the neural code. This will allow us to understand the effect of the non-negativity constraints on individual neuronal activities on linear dimensionality reduction methods. A common observation is that we can decode meaningful low-dimensional structures in population recordings as linear readouts (Chapter 2). We now turn

this around and ask how should the information have been 'encoded' in the first place by the population to achieve the said readouts.

A direct implication of our first assumption — population readouts are lower-dimensional — is that the same readout can be obtained by several population activity patterns in the upstream population, or in other words, the population activity is redundant. We illustrate this point in a network schematic of two neurons that encodes a one-dimensional signal. Suppose, for example, that the population activity is read out as the difference in activities of the two neurons (Fig. 3.1A). Then, as we show in Fig. 3.1B, any neural combination that falls on the dashed orange line gives the same readout. In fact, these neural combinations reside in the null-space of the decoding weights and in this illustration, forms a line. Thus, only by switching lines (i.e. going orthogonal to the null direction or along the coding direction) can a different signal be encoded. So, as the input signal changes, the population activity (red dot) will switch from one of the dashed orange line to the next. Effectively, the population activities corresponding to different input samples will lie on a surface in the neural space, or a neural manifold (e.g. the blue-green curve in Fig. 3.1B). Importantly however, this manifold is not uniquely defined since for any given input, there are many possible combinations of neuronal activities (lying on a dashed orange line) that would yield the same readout.

Given this vast set of possible network representations yielding the same readout, we ask whether this coding problem can be addressed from a normative perspective. In particular, we ask how should the network encodes its inputs given our understanding of the constraints that biological networks face. In the following, we describe several possible coding schemes and discuss their plausibility.

One of the simplest mapping from input signals to firing rates of the network is the linear one. This means that the firing rates of the neurons are obtained as linear combinations of the signals and we can also add an offset term to all neurons to capture any baseline activity. In fact, due to the mathematical convenience of such a coding scheme, it underlies many linear dimensionality reduction methods, e.g. PCA, dPCA (Kobak et al., 2016) when formulated as neural networks (see Chapter 2). However, linear encoding suffers from some strong limitations. First, firing rates are non-negative and thus, this scheme can code only a limited signal range. As we illustrate in Fig. 3.1C, for a given offset, the non-negativity constraint is violated for several input signals (crossed-out circles). To address this, the offset can simply be increased, which then moves the manifold (blue-green) up the non-negative quadrant, thus affording it a wider coding range. However, this is not ideal for two

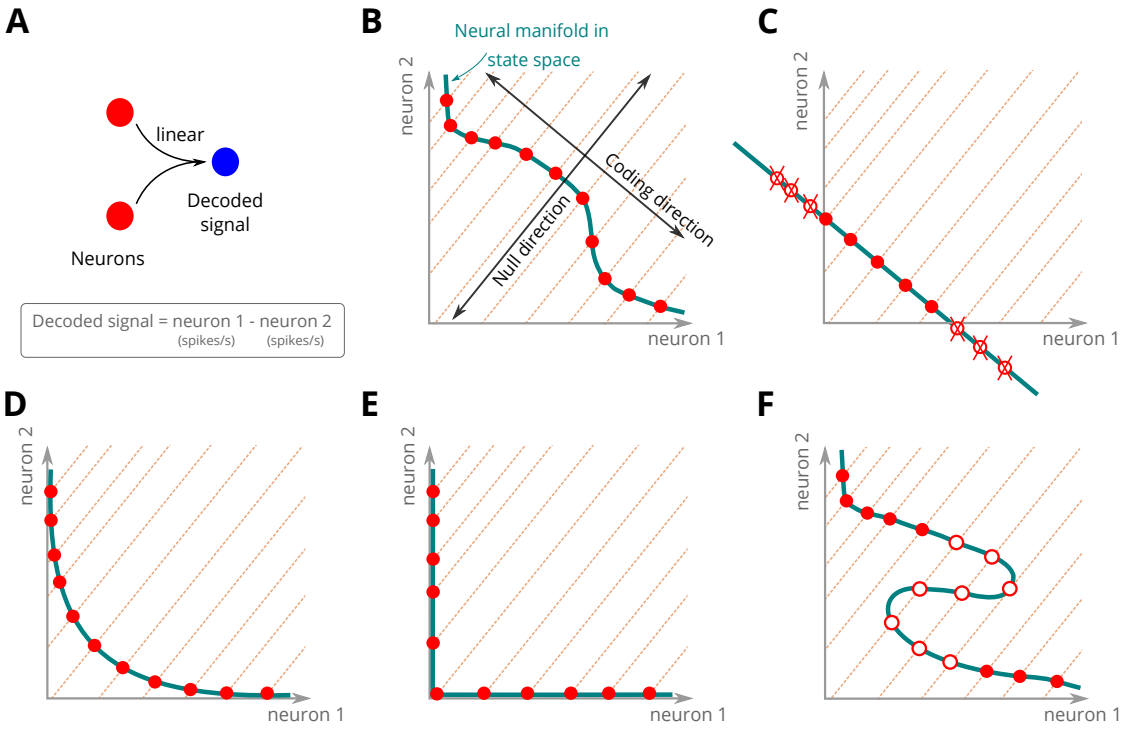

Figure 3.1: Illustrating the effects of assuming linear readouts and neural redundancy on the neural code. (A) Cartoon of a two-neuron network with a 1D readout. The decoded signal is for example, the difference in activity of the two neurons. (B) Illustrating a neural manifold (blue-green). Each neuron defines an axis in the neural space. Due to neural redundancy, combinations of network activity lying on a dashed orange line results in the same readout. Only by switching from one orange line to the next does the decoded signal change. Neural activity for a given input is illustrated as a red dot. As the input changes, the corresponding neural activity moves on the neural manifold. (C) If the manifold is linear manifold, the range for coding is limited due to the non-negativity constraints; the crossed-out red circles illustrate the network activities that violate these constraints. Moving the manifold north-east to remedy for this, however, entails an energetically expensive code. (D) A non-linear manifold can expand the range for coding. To maintain an energetically efficient code, the manifold should be bent towards the origin. (E) The curvature in (D) can be pushed further so that the manifold goes along the axes to maximise energy efficiency. But this means that the whole population can become silent (at the origin), which is experimentally unlikely. (F) Although the manifold can be twisted and tangled to further expand the coding range, this may not be desirable either. Different inputs can yield network activity patterns (empty red circles) that lie on the same dashed orange line, and thus a linear readout cannot discriminate between these inputs. A more complicated readout would then be needed, but this may not easily be implemented in biological circuits.

reasons. First, the code becomes energetically expensive as the overall population activity increases (Laughlin, 2001; Attwell and Laughlin, 2001) and second, in the middle of the quadrant, all the neurons are active which contradicts the usually observed sparsity of firing in several cortical areas (Wohrer et al., 2013).

In order to expand the dynamic range for coding, a simple solution would be to bend the neural manifold by introducing a non-linearity in the encoder. Ideally, we would want the curvature to bend towards the origin as this favours an energetically inexpensive code with low overall firing rates (Fig. 3.1D). We could push this curvature even further such that the neural manifold now goes along the axes as in Fig. 3.1E. However, situations where the whole population becomes silent (at the origin) are usually not experimentally observed, except e.g., when the animal is under anaesthesia and exhibit up-and-down states (Clement et al., 2008). Also, in such a coding scheme, whenever one neuron is active, all the other neurons in the population need to be kept below spiking threshold. This may also require energy and a tradeoff between signalling and resting costs might be required (Niven and Laughlin, 2008).

Moreover, not all non-linearities are desirable. For instance, if the manifold is all twisted up as in Fig. 3.1F, a linear decoder would fail to disambiguate distinct signals which have their neural representations falling in the null-space of the decoder (open red circles). Following the arguments that we laid out above, we posit that the neural manifold should resemble the one in Fig. 3.1D, under our assumptions.

Importantly, despite the curvature in the manifold, we should still be able to linearly decode linearly the underlying signal (here, a one-dimensional signal) from the neural activities. Similarly, linear methods such as PCA aim to find the latent variables in data through a linear mapping, but the non-negativity constraints on neuronal activities are not explicitly taken into account by these methods. In this chapter, we are primarily interested in understanding how these constraints, and thus the curved neural manifold, affects the results of PCA in finding the true underlying signals in the data.

Given the picture of how the neural manifold should be shaped in this illustrative example, we will start by characterising geometrically the principal components that PCA returns when applied to samples from this manifold. We will then investigate, using neural network models that incorporate our assumptions, what these constraints entail in higher-dimensional examples.

3.3 Geometry of principal components when the neural manifold is curved

From the above illustrative example, we consider the neural manifold to resemble the one in Fig. 3.1D. The samples in the manifold (red dots) are obtained as the underlying input changes, which we can imagine as some command signal to the network that varies as a function of some task parameter. For example, it could vary linearly as in Fig. 3.2E. When PCA is then applied to this data, it finds a new coordinate system consisting of orthogonal axes to represent the data. The axes are arranged in order of amount of variance they each capture when the data is projected onto them with the first axis capturing most variance.

Here, we are particularly interested in the representations that PCA gives in the new coordinate system. Fig. 3.2A shows the first axis, or dominant dimension with most variance. By projecting the data onto it (pale red), we see this results in a best linear approximation for the data. This approximation, as we recall from Chapter 2, can be formulated as an autoencoder where PCA finds an optimal linear map of the data onto a one-dimensional bottleneck (decoding step) and back up to the state space (encoding step) such that the reconstructed activity is as close as possible to the original trajectory (see inset of Fig. 3.2A). The autoencoder then yields a bottleneck representation (blue node in middle layer) which corresponds to the first principal component (PC). By plotting the latter against the task parameter, we see that it reflects the underlying input signal to the network (compare Fig. 3.2B and E). Thus, PCA would allow us to estimate, in an unsupervised way, the underlying explanatory variable for the population activity, which here is the input signal.

However, this first axis does not explain all the variance in the data. PCA finds a subsequent dimension that maximizes the variance of the projected data in the residual subspace. Fig. 3.2C shows this second axis and similarly, we can project the data onto it. A second PC or bottleneck representation corresponds to the activity along this axis which we can again plot as a function of the task parameter (Fig. 3.2D). PCA now predicts a component that looks roughly quadratic, to reflect the curvature in the manifold. Given its characteristic shape, we refer to this second component as a 'higher-order' component. Importantly, note that this second PC compensates for the fact that the population trajectory is bent due to coding constraints, but it has no functional relevance for the neural code since the true explanatory variable is already captured by the first component.

In reducing dimensionality, we would probably be fine with PCA on this toy

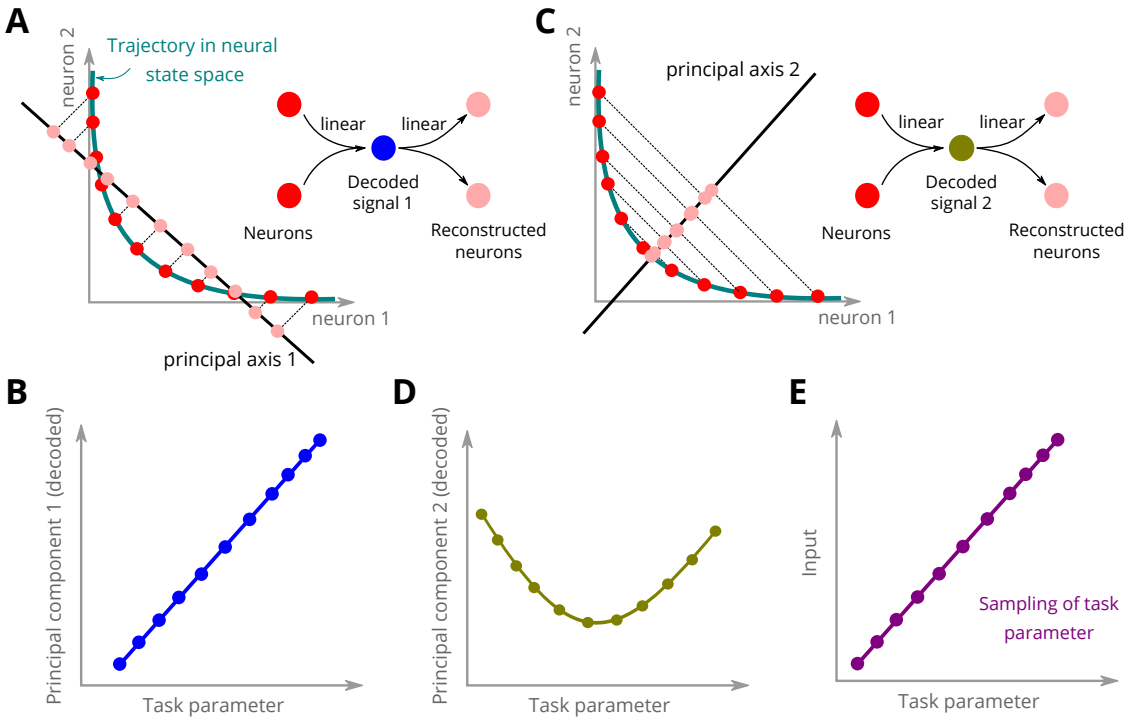

Figure 3.2: Geometry of predicted principal components (PCs) when the neural trajectory is bent (blue-green) due to coding constraints. PCA finds a new two-dimensional coordinate system to represent the data. (A) The first principal axis captures most variance when the data (dark red dots) is projected onto it. The projections give the best linear approximation (light red dots). (Inset) PCA as an autoencoder. This linear approximation is equivalent to the best linear reconstruction of the data after passing it through a bottleneck (blue node). (B) The activity of the bottleneck in (A) or first PC, when plotted against the task parameter, varies linearly, reflecting the input signal in (E). (C) A second principal axis, orthogonal to the first, is predicted to capture the remaining variance. It yields a different reconstruction of the data (light red dots). (Inset) PCA autoencoder as in (A) except that a different readout is obtained (olive). (D) The second decoded signal or PC displays a 'quadratic-like' shape against the task parameter. This PC reflects the curvature in the manifold due to coding constraints. (E) The network input, not known to PCA, is a linear function of the input and correctly captured by the first PC. The second PC has no coding function.

example as we would choose the first principal component to summarize the data as it captures most variance. However, this schematic illustrates one important point: even though there is only one input signal and that we started with an assumption of linear readout to generate the neural activities, which PCA incorporates too in its decoding step, PCA still predicts an additional component that does not reflect any underlying signal. In other words, the number of dimensions needed to describe this manifold, or its intrinsic dimensionality, is one. But PCA shows that the dimensionality of space that the manifold occupies, or its embedding dimensionality (Camastra, 2003), is higher and the extra dimensions in the embedding may not reflect anything functional in the data.

In a real experimental setting of course, the true underlying explanatory variables are unknown and thus, need to be inferred from the data. Estimating these latent variables using linear methods such as PCA leads us to a conundrum: are the predicted components truly reflecting some underlying signals or are they simply compensating for the curvature in the manifold and are thus, functionally irrelevant? We next show via network simulations that when the network's size and inputs are scaled up, the embedding dimensionality increases significantly compared to the intrinsic dimensionality, and thus these functionally irrelevant components become fairly consequential.

3.4 Modelling neural activities of the brain

To understand the effect of the non-negativity constraints on linear dimensionality reduction methods, we resort to numerical simulations, which provide us with the ground truth against which we can compare the predictions of these methods. In the following subsections, we take a small detour by first describing the two neural network models that we will use in our simulations to generate non-negative neuronal activities.

3.4.1 Neural network model with static non-linearity

A standard way to incorporate the non-negativity constraints in a neural network is by introducing a non-linearity when generating the network activities. A simple approach is to first encode the inputs linearly, followed by a static non-linearity which is the basis for a linear-nonlinear (LN) network model. Such network model dates as early as the first perceptron (Rosenblatt, 1958) and has widely been used in both artificial neural networks (LeCun et al., 2015) and biologically-inspired network

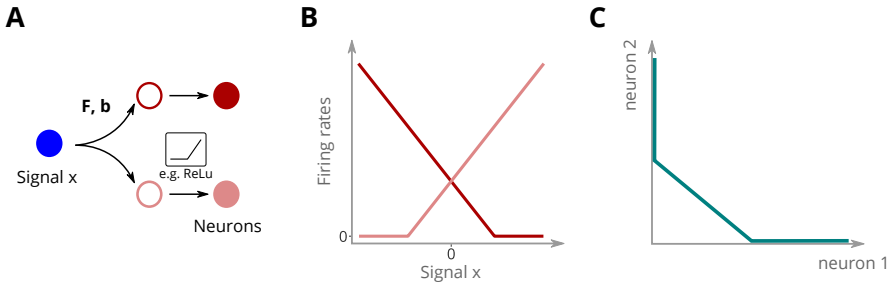

Figure 3.3: Illustration of the linear-nonlinear (LN) network model. (A) Schematic of a network of two neurons encoding an input signal, x. The network firing rates are rectified to be non-negative via a 'ReLu' non-linearity. (B) Threshold linear tuning curves are obtained. (C) In the neural space, the manifold of network activities is piecewise-linear.

models (Chichilnisky, 2001; Pillow et al., 2005). Mathematically, the LN network model encodes its inputs $\mathbf{x}$ into its firing rates $\mathbf{r}$ according to,

$$\mathbf{r} = g(\mathbf{F}\mathbf{x} + \mathbf{b}) \tag{3.1}$$

where $\mathbf{F}$ are the coefficients for the linear combination of the inputs, $\mathbf{b}$ is a bias term that captures any baseline activity in the population and $g(.)$ is a chosen non-linear function. We illustrate this transformation schematically in Fig. 3.3A.

A commonly used non-linearity is the rectified-linear unit (ReLu) which returns the element-wise maximum between some input vector $\mathbf{v}$ and 0 i.e. $g(\mathbf{v}) = \max(\mathbf{v}, 0)$. This rectifies the outputs of the network (here, firing rates of neurons) to be non-negative as demonstrated by the tuning curves schematics in Fig. 3.3B. For a given set of parameters $(\mathbf{F}, \mathbf{b})$, the neural trajectory in the neural space is bent towards the origin; in fact it has a piecewise-linear shape due to the chosen nonlinearity (rectification).

Such an LN model can be stacked up in layers to create an augmented neural network model which can do arbitrary non-linear computations. With more sophisticated learning algorithms, such networks have had tremendous success in many machine learning applications (LeCun et al., 2015). However, they have some clear shortcomings to explain biological observations. First, optimizing the parameters in such networks for effective computations does not guarantee that the internal neural representation at each layer is energetically efficient. This is because the training of the networks is usually done irrespective of the level of neuronal activities (unless, high firing rates are explicitly penalized). Second, when such layers are implemented using biophysical spiking neurons, the firing patterns of the neurons are often quite regular (Eliasmith and Anderson, 2004), which contradicts the

observed highly irregular and asynchronous spiking activity in several cortical areas (Denève and Machens, 2016).

3.4.2 Neural network model with optimal representations

We now consider an alternative modelling framework that can address the issues mentioned above. In this framework, the neural network is set as an autoencoder to find optimal neural representations for its inputs. The network aims to reach an optimal tradeoff between two competing aims: (1) the representation as determined by the readout of the population activity needs to be accurate with respect to the inputs and (2) the representation needs to be achieved at low computational costs, in particular, through limited firing rates. As a result, the network representations become energetically efficient.

Moreover, it has been shown that when such an optimality principle is incorporated into spiking networks, they usually display important biological features such as such as irregular and asynchronous spike trains, trial-to-trial variability and excitatory-inhibitory balance (Boerlin et al., 2013; Barrett et al., 2016). This arises because the neurons in the resulting networks share information through lateral connections, a feature that is not present in feedforward networks as the LN model (see an example architecture of such networks in Fig. 2.1A). Due to the neuronal interactions, the spiking network also reduces redundancy since no two spikes code for the same information, thus abiding to the 'efficient coding hypothesis' — a core idea in theoretical neuroscience (Atick and Redlich, 1990; Barlow, 1969).

Here, we build on this framework by hypothesizing that the networks not only code for some observed input signals, $\mathbf{x}$, but are also driven by a set of latent variables which yield a net modulatory effect on the global population activity. This effect can be summarized according to a global background signal, z, that we model as another input to the network. In analogy to the dataset (Kobak et al., 2019) that we analysed in Chapter 2, z would be the modulatory signal that enables fluctuations of the global population activity along up-and-down states. Recall that during up-and-down states, the population would transition from periods of firing (up-states) to periods of silence (down-states). For simplicity, we will keep in the following analysis that z is a one-dimensional signal.

We now formalize the above ideas into a loss function. First, following our assumption of linear readouts, we define an estimate, $\hat{\mathbf{x}}$, of some M-dimensional input signal, $\mathbf{x}$, as a linear combination of the firing rates generated by the network

of N neurons, i.e.,

$$\mathbf{x} \approx \hat{\mathbf{x}} = \mathbf{D}\mathbf{r} \tag{3.2}$$

where $\mathbf{D} \in \mathbb{R}^{M \times N}$ is a matrix of decoder weights. Similarly, an estimate of the background signal is obtained as $\hat{z} = \mathbf{u}^\top \mathbf{r}$ and we fix $\mathbf{u}$ to be an N-dimensional vector with only positive entries, i.e. $u_i > 0$ for all $i \in 1, \ldots, N$ so that this estimate is a weighted average of the population activity. Finally, we quantify the cost of generating the network representations via a cost function, $C(\mathbf{r})$, that depends on the population firing rates, $\mathbf{r}$. Then, the tradeoff between cost and accuracy of representations can be formulated with an optimisation problem:

$$\begin{aligned} \underset{\mathbf{r}}{\text{Minimize}} \quad & \big(E(\mathbf{r}) = \|\mathbf{x} - \mathbf{D}\mathbf{r}\|_2^2 + \beta||z - \boldsymbol{u}^\top \boldsymbol{r}||_2^2 + \mu C(\mathbf{r})\big) \\ \text{subject to} \quad & \mathbf{r} \geq 0 \end{aligned} \tag{3.3}$$

The first term in the objective quantifies the accuracy of the code for representing observed inputs, $\mathbf{x}$. Similarly, the second term determines the accuracy in decoding the background signal but note that since $\mathbf{u}$ is set to be all-positive, an increase in z necessarily leads to an increase in the overall population activity. Thus the background signal becomes a modulatory signal of the population activity. Furthermore, we introduce a parameter $\beta > 0$ that allows us to explicitly control the importance of representing this background signal in the neural code. Note that although the background signal can also be read out linearly here, we will assume that the important signals for a give task or computation remain the signals, $\mathbf{x}$, and thus the readout for $\mathbf{x}$ should remain accurate. Finally, the third term determines the cost of the representations with parameter $\mu > 0$ controlling this cost-accuracy tradeoff.

In particular, if $C(\mathbf{r}) = \|\mathbf{r}\|_1$ is the L1-norm, then a sparse code is encouraged with a small proportion of neurons in the population being active at a time (Rolls et al., 1998; Olshausen and Field, 1996). Alternatively, if $C(\mathbf{r}) = \|\mathbf{r}\|_2^2$, then the overall population activity is shrunk, whereby many neurons fire at lower rates, thus encouraging a more distributed code across the population. In this chapter, we focus on the latter cost for mathematical convenience as it yields a convex quadratic objective function. Mathematically, the optimisation problem in 3.3 then corresponds to a quadratic program (QP) (Boyd et al., 2004). As was noted by Barrett et al. (2013), the solution to this QP in fact corresponds to the time-averaged activity of the spiking networks mentioned earlier in this section, for constant inputs. Since we are primarily interested in this chapter in the input-output mappings, we will solve the QP to generate the firing rates of the network for given inputs. We reserve a more in-depth discussion of how spiking neural networks generate neural representations in Chapter 5.

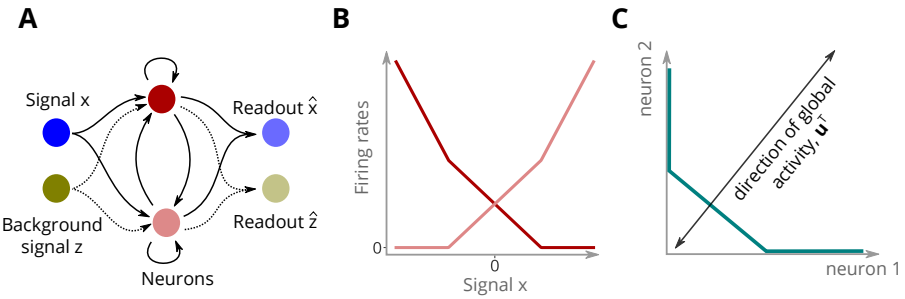

Figure 3.4: Illustration of the autoencoder network with energetically efficient neural representations. The firing rates for each constant input sample are obtained as a solution to a quadratic program (QP) (Barrett et al., 2013). (A) Schematic of the network with two neurons and two inputs — (1) a signal, x that fluctuates and (2) a constant background signal, z that modulates overall population activity. In contrast to a feedforward network, the population shares information through lateral connections. (B) Tuning curves of the neurons as a function of the varying input, x. When one neuron hits the zero-boundary, the other neuron keeps an accurate readout for x by changing its activity and thus, displays a kink in its tuning curve. (C) The neural trajectory is piecewise-linear. The background signal modulates the overall population activity by moving the middle piece of the trajectory along the $\mathbf{u}^\top$ direction.

3.4.3 Contrasting the network with firing rates computed by QP with the LN network

Owing to lateral connections, the network implementation of the previous framework (Barrett et al., 2013; Barrett et al., 2016) differs from the feedforward LN network model. In particular, the lateral connections afford the network with more versatility in the input to firing rates mapping than the LN models. For instance, when one of the neuron in the network hits the zero threshold, this information reaches the other neurons, which in turn, change their responses to maintain an accurate readout. This can also be understood through the QP optimisation perspective where the optimisation task is distributed across the network, and thus when one of the neuron hits the hard non-negativity constraint, other neurons in the network have to change their activity for an optimal solution. This effect is reflected by kinks in the tuning curves of the neurons as we see in Fig. 3.4B.

We can further show that this feature endows the network with representations that cannot easily be approximated by the LN network's output. We can already illustrate this in a network of three neurons receiving a two-dimensional input (x_1, x_2) on a 2D grid in the interval $[-1, 1]$. For the sake of illustration, we consider the tuning surface of one of the neurons (see Fig. 3.5A). We see that it exhibits creases that only partially span the input space. This arises due to the other neurons' re-

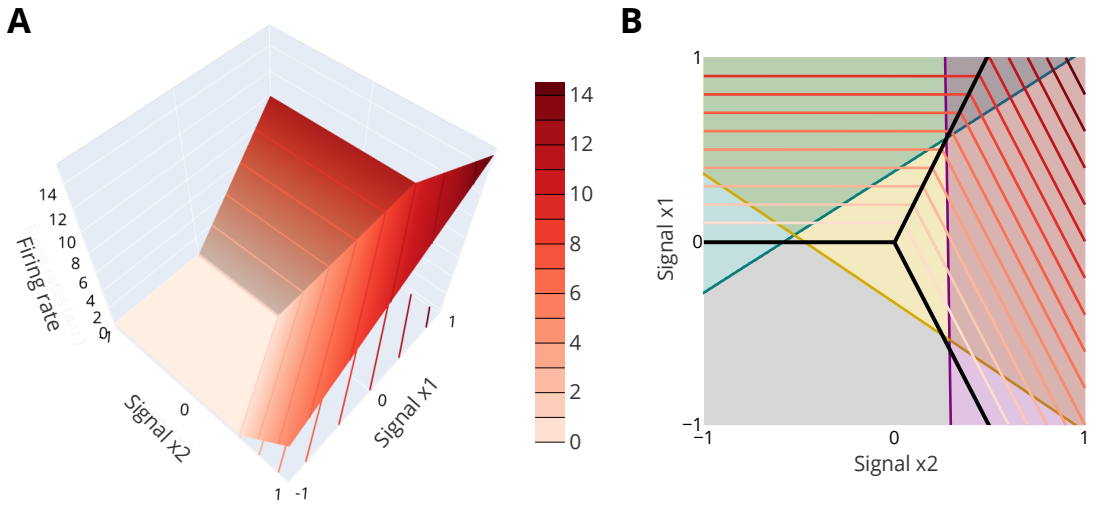

Figure 3.5: Contrasting the representations from the linear-nonlinear (LN) network and those predicted by quadratic programming (QP). (A) Tuning surface of a neuron calculated by QP when simulating a network of 3 neurons for a 2D input. It exhibits creases due to information from the other neurons, received through lateral connections in the network. (B) Contour map of the tuning surface in (A) with level curves in red. After fitting a weighted sum of three rectified-linear units (ReLu's) to this surface, each ReLu unit becomes active in a domain of the input space (colored region), with grey being the inactive or zero activity area. Each ReLu surface has a crease that cuts through the entire input domain (colored straight line) while in thick black are the creases of the tuning surface from (A). Thus, only an approximation to the QP tuning surface can be obtained using weighted combination of ReLu's, due the nature of the ReLu non-linearity.

sponses which reach our neuron under consideration. These creases are more more clearly seen in Fig. 3.5B as the thick black lines.

In contrast, when we fit the weighted sum of three ReLu units [1] to this tuning surface, each individual unit exhibits creases that span the whole input space; in Fig. 3.5B, the straight colored lines correspond to creases of individual ReLu's which cut though the whole input domain. Thus, we will need many such ReLu's to get a better approximation to this QP solution (Amos and Kolter, 2017). We will see in Chapter 4 that this feature affords the network more representational power than the LN network, although this comes at greater computation costs since an optimisation problem needs to be solved for each input sample (Amos and Kolter, 2017) and from a network perspective, would require extensive lateral wiring amongst neurons in the network.

[1] With P firing rates samples, $\mathbf{r}_i^{QP}$, from the tuning surface, we learn parameters $(\mathbf{D}, \mathbf{F}, \mathbf{b})$ that minimize the loss function $L = \sum_i^P ||\mathbf{r}_i^{QP} - \mathbf{D}g(\mathbf{F}\mathbf{x}_i + \mathbf{b})||_2^2$ where $g(\mathbf{v}) = \max(\mathbf{v}, 0)$ is the elementwise maximum.

3.5 Tail of higher-order principal components in synthetic data

Now that the neural network models that we will use are fully described, we can return to our initial problem: how are the predictions of PCA affected by the non-negativity constraints of neuronal activities in higher-dimensional examples. We will show, in this section, that when PCA is applied to synthetic data generated using these models, it sometimes extract the correct latent variables, but also predicts a tail of higher-order components. We will study this effect in various examples of increasing dimensionality of both the inputs to the network and its size.

3.5.1 Simple example with a one-dimensional varying input signal

To visualize the principal components that PCA predicts, we start with a simple example of a network of 30 neurons that encodes a varying input signal, x, and a constant background signal, z, with rates predicted by quadratic programming (QP) (equation (3.3)). Due to the chosen decoder weights and the constant background signal, the network displays inhomogeneous responses as seen by the tuning curves in Fig. 3.6B with mixed selectivity to both positive and negative parts of the input signal, x, and non-monotonicity of these functions. Note that without the background signal in this one-dimensional input example, the tuning curves remain fairly regular (Fig. 3.6D).

We now apply PCA to the firing rates generated (Fig. 3.6B), bearing in mind that there is a linear readout of the population activity that recovers accurately the varying input signal in the simulation (see Fig. 3.6C). PCA predicts a first component that correctly reflects the underlying input signal, x (although of a different scale). However, it also finds two additional higher-order components that resemble polynomials of increasing order (Fig. 3.6E). Interestingly, they also resemble the higher-order components that we found in real data (Chapter 2). Had we not known the ground truth in this simulation, these higher-order components can mislead us to overestimate the (intrinsic) dimensionality of the data, and make interpretations less clear.

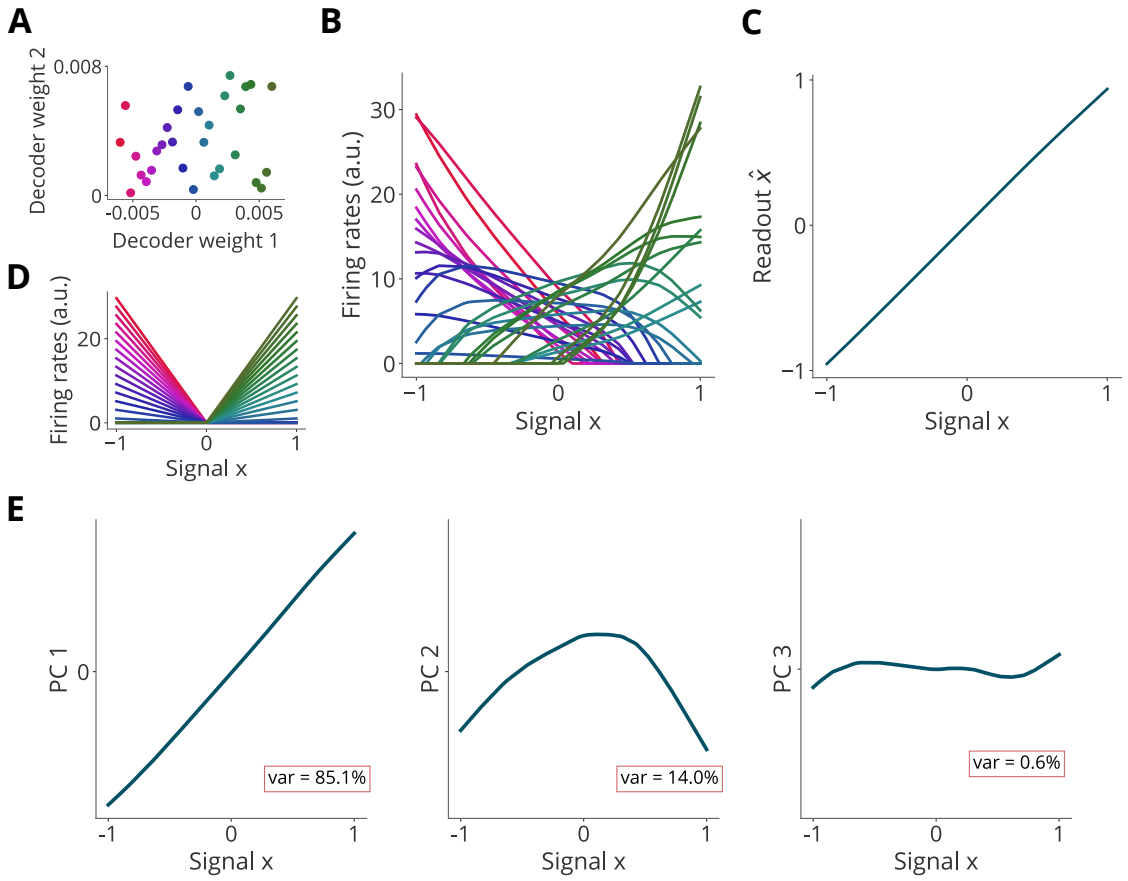

Figure 3.6: Simple network simulation with a varying 1D signal x and a constant background signal, $z = 0.7$, with firing rates calculated by quadratic programming (QP). PCA predicts higher-order components due to the non-negativity constraints on firing rates. (A) Decoder weights for the simulation. (B) Tuning curves of $N = 30$ neurons in the network as x varies in the interval $[-1, 1]$. (C) A one-dimensional readout recovers this input signal accurately despite the inhomogeneity in the network responses. (D) In the absence of a background signal, the inhomogeneity of the tuning curves disappears. (E) PCA predicts 3 components to explain the variance in this data; explained variance by each component shown as a percentage in the red box. The first principal component (PC) correctly reflects input signal x, but predicts two additional higher-order components to compensate for the curvature in the neural manifold due to the non-negativity constraints. These are functionally irrelevant. **Additional parameters:** background cost, $\beta = 1$, quadratic cost, $\mu = 1e - 05$.

3.5.2 Network with higher-dimensional inputs

We next show that these higher-order components can become fairly consequential when the dimensionality of the inputs increases. As an example, we consider a network model that encodes a three-dimensional input. We model this input as follows: for each dimension of the signal, we draw $P = 1000$ random samples from a standard Gaussian distribution with zero mean and unit variance, i.e. $\mathcal{N}(0, 1)$, and apply a Gaussian filter that smooths over adjacent points, thus mimicking a continuous temporal signal. We illustrate each dimension of the input signal as thick and pale colored trace in Fig. 3.7A.

We then fed this input to the neural network with population activity calculated by QP. As in the previous example, we add a constant background signal, $z = 2$, with an all-positive decoder weight for its readout that modulates the global population activity. Once the population data has been generated, we checked that a linear readout is sufficient to recover the true input signals. As shown in Fig. 3.7A, each dimension of the readout (thin, dark colored traces) overlaps with the corresponding dimension of the input signal (think, paler traces of corresponding colour). Given that we used the same decoder weights, $\mathbf{D}$, used in the generative network (see equation 3.3), this observation is not surprising.

However, when PCA was applied to this data, it predicted several additional components than the true number of dimensions in the input signal. To capture $\sim 95\%$ of the variance in the data, it required nine components (Fig. 3.7B), thus overestimating the (intrinsic) dimensionality of the data. Furthermore, the first three PCs failed to match the input signals (Fig. 3.7A(ii); the input signals and PCs were normalized for better comparison). To further quantify this difference, we computed the principal angles between the subspaces for the first three PCs and the input signal, $\mathbf{x}$. These angles were computed using the same algorithm proposed in Knyazev and Argentati (2002); the minimum angle between pairs of basis vectors (defined by the decoder weights from PCA and those used to decode signal $\mathbf{x}$ in the QP algorithm) was computed recursively, with each basis vector coming from an individual subspace. In descending order, the angles were $[55.1, 48.3, 42.8]$ degrees, showing the mismatch between the subspaces. Hence, with the first three PCs, we would end with an incorrect representation of the data.

Next we check if these observations hold when the background signal was not encoded (by setting $\beta = 0$ in equation (3.3)). In this case, PCA again predicted more components (see Fig. 3.7C), although the misrepresentation of the underlying input

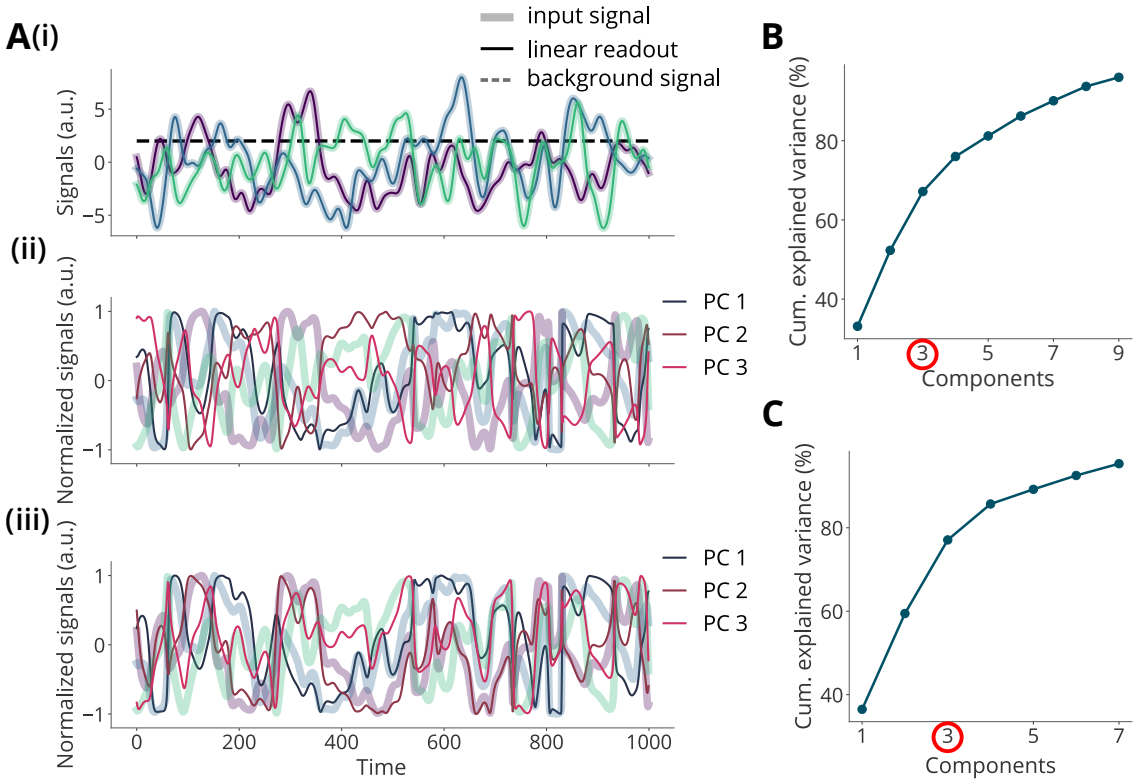

Figure 3.7: PCA predicts several additional principal components (PCs) than the dimensionality of inputs to a network of $N = 100$ neurons. Firing rates are computed using QP. (A)(i) The 3D input, $\mathbf{x}$ (think and pale traces) and a constant background signal, $z = 2$ (dashed line) fed to the network. The background signal was turned off in one of the simulations. A linear readout of the population activity with the right decoder weights recovers the inputs (overlapping thin and dark traces). (ii) First three PCs compared to the input signals in a simulation with the constant background signal. They were normalized for better comparisons. (iii) Same as (ii) but without the background signal. (B) Cumulative variance explained upto a threshold of 95%. Although there were 4 inputs ($\mathbf{x}$ and z), PCA needs 9 dimensions in the neural state space to reach the threshold. Since z is fixed, the intrinsic dimensionality is 3 (red circle). (C) Same as (B) but, for simulation without the background signal. **Simulation parameters**: quadratic cost $\mu = 1e-05$; background cost $\beta = 1e-05$ for A(i)(ii), B(i) and $\beta = 0$ for A(iii), C; decoding weights, $\mathbf{D}$, were randomly sampled from a standard Gaussian distribution and normalized over neurons (length of decoder weight for each neuron set to 0.1; background weights, $\mathbf{u}$, were randomly sampled from a uniform distribution.

signals was less strong (Fig. 3.7A(iii)) with principal angles being $[19.9, 16.5, 5.4]$ degrees. While coding for the background signal leads to more principal components, a phenomenon which we will explore further in a later section, in all cases the non-negativity constraints on firing rates can lead to misrepresentations of the data when using linear methods such as PCA.

Effect of energy efficiency constraints on PCA predictions

In the above simulation, the network model we considered generated neural representations that were also energetically efficient. Here we study the effect of this energy constraint, together with the non-negativity constraints, on the predictions by PCA. One way to do this analysis is to consider neural representations that incorporate the non-negativity constraints, without necessarily abiding to an energy efficiency principle and contrast the PCA results on such data. In fact, the alternative network modelling approach we discussed earlier, namely the LN model, allows us to address this. To enable comparisons with the previous model, we set the LN network as an autoencoder so that the input can be read out linear using weights $(\mathbf{D}, \mathbf{u})$ (same as in the previous model) from the population activity. Then, we learn the model's parameters, namely feedforward weights, $\mathbf{F}$, and biases, $\mathbf{b}$ so that the readout is accurate. This can be formalized as minimizing the following loss,

$$L(\boldsymbol{F}, \boldsymbol{b}) = \sum_{i}^{P} \left(||\boldsymbol{x}_i - \boldsymbol{D}g(\boldsymbol{F}\boldsymbol{x}_i + \boldsymbol{b})||_2^2 + \beta ||z - \boldsymbol{u}^T g(\boldsymbol{F}\boldsymbol{x}_i + \boldsymbol{b})||_2^2 \right) \tag{3.4}$$

with respect to parameters $(\mathbf{F}, \mathbf{b})$ over all the P-samples. Note that this loss function makes the readouts for both signal $\mathbf{x}$ and fixed background signal z explicit.

After learning the parameters using same inputs and network size as in the previous simulation, we verify that a linear readout with weights, $\mathbf{D}$, recovers the relevant input signal, $\mathbf{x}$. Fig. 3.8A(i) shows that the readout is accurate. We then apply PCA to the generated firing rates. Although more principal components than the number of dimensions in the input can again be observed (see Fig. 3.8B), we find that fewer additional components than in the previous simulation are required. This occurs even though the distribution of mean firing rates (computed by averaging over the P samples) across neurons shows that many neurons have close to zero firing rates (see Fig. 3.8C) i.e. the neurons hit the zero-threshold in this simulation also. The energetic constraints in the previous model thus seem to change the nature of the neural representations which entail in a longer tail of 'higher-order' components. We explore this effect in more depth in the next section.

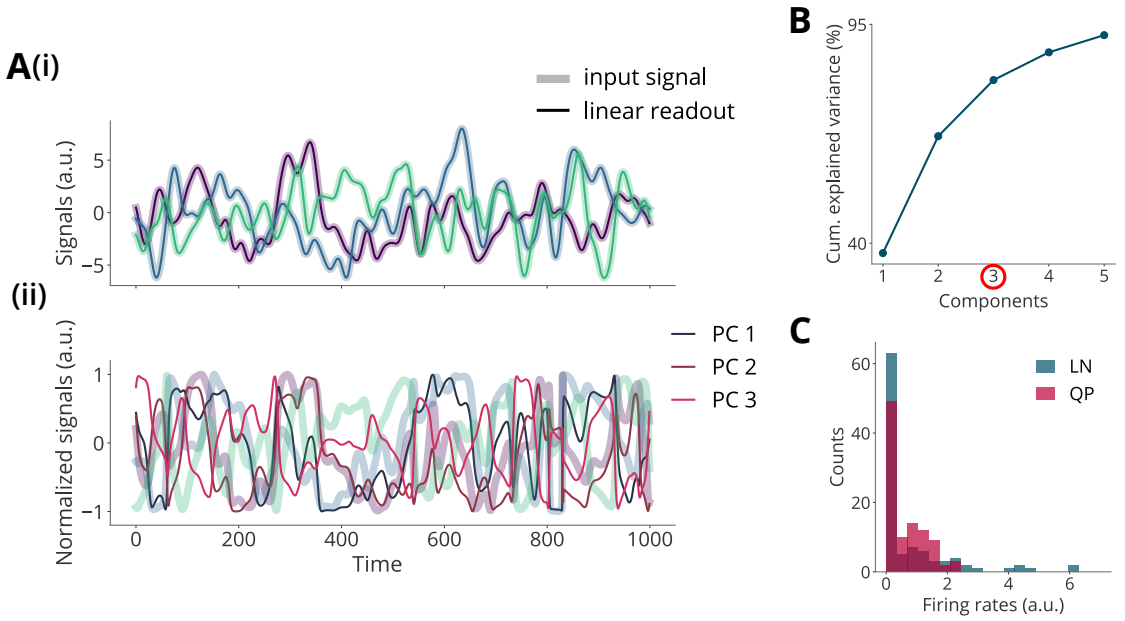

Figure 3.8: Generating firing rates in an auto-encoder with rectified-linear units (LN model). For given decoding weights, the auto-encoder optimizes its feed-forward weights and biases so that the readout of population activity matches the inputs. (A)(i) Same varying inputs as in Fig. 3.7. Linear readouts of the firing rates (thin and dark traces) match the input signals (thick and pale traces) (ii) The first three principal components (thin traces) plotted against the inputs (think pale traces). This linear projection of the data fails at recovering the input signals. (B) Cumulative explained variance upto a threshold of 95%. (C) Distribution of mean firing rates obtained by averaging over samples for each neuron. For both models (QP and LN), several neurons hit the zero-threshold, but less 'higher-order' components are observed here.

3.5.3 Consequence of a modulating background signal on linear dimensionality

As we observed earlier, when the network's representations are optimized under energy constraints, PCA tends to significantly overestimate the (intrinsic) dimensionality of the data, despite the fact that a linear readout with the correct decoder accurately retrieves the underlying input signals. Recall that the dimensionality of the inputs to the network defines the intrinsic dimensionality of the manifold. Thus, this raises the question as to how many dimensions PCA predicts as the energy requirements fluctuate. In fact, we can probe this question explicitly as the network codes for a background signal, z, which effectively modulates the global population activity and hence, the energy requirements. An increased z should increase the overall population activity, thus yielding a less energetically efficient code (also, corresponding to moving the population activity north-east along the non-negative

orthant, as in Fig. 3.4C). As a result, we can use variations of the background signal and its importance in the code, to determine how the energy requirements affect the shape of the neural manifold, and thus the results of PCA.

We first consider a network of three neurons encoding a one-dimensional signal. This allows us to visualize the resulting 1D manifold as the energy requirements vary. We consider three regimes of our model with optimal neural representations: (i) no background signal is coded by the network i.e. $\beta = 0$, (ii) a moderate background signal is coded i.e. (z moderate, $\beta > 0$) and (iii) the coded background signal is large i.e. (z large, $\beta > 0$). Fig. 3.9 shows the resulting manifold across the three different regimes. When the network's activity is not modulated by a background signal, the resulting manifold is bent all the way to the origin (Fig. 3.9A) and is embedded in a lower-dimensional subspace. However, in the presence of a moderate background activity, the manifold becomes twisted in the ambient space (Fig. 3.9B) since the network now needs to code for the relevant 1D signal while having its the overall activity moved towards a set-point, z. To yield an optimal representation under the non-negativity constraint that each neuron faces, the manifold then exhibits several kinks. As a result, the embedding becomes higher-dimensional; in this example, the manifold explores all the dimensions of the ambient space. Finally, when the background signal becomes increasingly strong, the manifold moves further north-east the non-negative orthant, such that for the range of the input signal, individual neurons hardly hit the zero-threshold. Hence, the manifold has fewer kinks (Fig. 3.9B) and its embedding becomes lower-dimensional.

To further quantify these observations in higher-dimensional networks, we explore the parameter-space defined by (β, z) for the background activity and measure the resulting linear dimensionality, K, given by PCA. The optimal firing rates are again computed by QP (equation 3.3). Recall that β controls the importance of coding for the background signal while z determines the level of the signal. To quantify the linear dimensionality, K, we computed the cumulative variance explained by the components and determine the number of components needed to reach an arbitrary cutoff threshold, $T = 80\%$, of explained variance, or mathematically,

$$K(T) = \arg\min_{\tilde{K}} \ \tilde{K} \text{ s.t. } \frac{\sum_i^{\tilde{K}} \lambda_i}{\sum_i^{N} \lambda_i} \geq T \tag{3.5}$$

where λ_i is the i^{th} eigenvalue of the N-dimensional neural population covariance matrix used in the PCA algorithm.

However, it is possible that sufficiently strong background signals would overwhelm the code (for $\beta > 0$), in which case, the readout for the relevant signals, $\mathbf{x}$

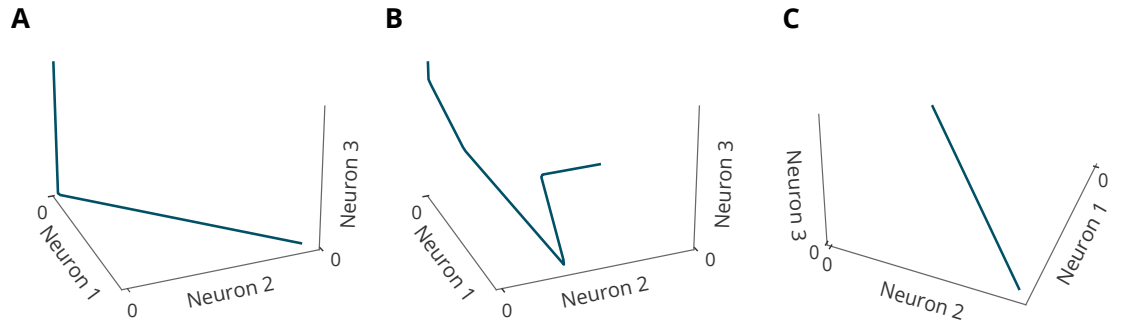

Figure 3.9: 1D neural manifold resulting from a network of 3 neurons encoding a 1D signal. The manifold is shaped differently according to a background signal that modulates overall population activity. (A) **No background signal**. The background signal is not important in the code, and this results in a manifold that bends all the way to the origin while exploring only a few dimensions of the ambient neural space. (B) **Moderate background signal**. To maintain optimal neural representations in the presence of a moderate background signal, the manifold exhibits many more kinks and thus explores more dimensions within the ambient space. (C) **Strong background signal**. For increasingly strong background signal, the manifold moves up the non-negative orthant. For the range of input signal encoded, individual neurons do not hit the zero-threshold and thus, yield optimal representations without kinks along the manifold.

would be poor. Thus, to check this, we further quantify the fitness of the readout, $\hat{\mathbf{x}} = \mathbf{D}\mathbf{r}$ (where $\mathbf{D}$ is also used in QP algorithm when generating firing rates) by computing its distance from the inputs, $\mathbf{x}$. In particular, we will use the R^2 metric obtained by comparing the readout value, $\hat{x}_i^j$, for dimension j and for sample i with the true input sample, x_i^j, and then averaging over all input dimensions as follows,

$$R^2 = \frac{1}{M}\sum_j^M \left(1 - \frac{\sum_i^P (x_i^j - \hat{x}_i^j)^2}{\sum_i^P (x_i^j - \bar{x}^j)^2}\right) \tag{3.6}$$

where $\bar{x}^j$ is the average input value for a dimension, j. Note that the closer the R^2 value is to its maximum possible value, i.e. 1, the more accurate is the readout.

In the following simulations, we will fix for simplicity, the decoding weights of the background signal as a vector of ones i.e., $\mathbf{u} = \mathbf{1}$, instead of being random positive numbers. Then, the background signal can be decoded as the sum of the neuronal activities. As before, each dimension of the input signal is modelled by taking random samples from a standard normal Gaussian, $\mathcal{N}(0, 1)$, but the filtering step is omitted as this would entail further parameter choices. Then, for each input sample, we generate firing rates for a network of $N = 200$ neurons using quadratic programming. Furthermore, we vary the dimensionality of the input signals from $M = 5$ to $M = 40$ for the same network size.

In Fig. 3.10, we show the resulting dimensionality upon applying PCA on the synthetic firing rates (contours on the left column) and R^2 values of the linear readouts (contours on the right column) as a function of the parameters (β, z). We observe that when the background signal has little importance in the code (low β), PCA needs fewer dimensions to explain the data. This would correspond to the unlikely regime where the neural manifold has a curvature that almost goes to the origin as in Fig. 3.9A. Conversely, when β is highest, the readouts for the relevant signals (across a wide range of z) are poor (low R^2), especially as the input dimensionality, M, increases. At the same time, when the background signal, z, becomes sufficiently big, the (embedding) dimensionality predicted by PCA gets closer to the true input (intrinsic) dimensionality (observed more clearly for $M = 5, 10$). This is because when z is sufficiently high, the neural manifold gets closer to a linear regime for the range of input values being coded by the network (the manifold would be substantially up in the non-negative orthant as in Fig. 3.9C). However, this regime, as we discussed earlier, is again unlikely as the code becomes energetically very costly.

We can in turn, consider the more realistic regime where both β and z take moderate values. Then, the background activity will have relative importance, but the readouts for the input signals, $\mathbf{x}$, would remain accurate. In a low-dimensional network (i.e. few neurons), the manifold would resemble the one in Fig. 3.9B, but this picture becomes more complex in bigger networks. This can be seen by the sharp increase in dimensionality that PCA predicts in Fig. 3.10 as compared to the dimensionality, M, of the inputs. This can be explained by the fact that the neural manifold becomes tiled with a multitude of piecewise-linear surfaces to maintain an optimal code where the inputs can still be read out linearly while the firing rates are kept as low as possible. As a result, PCA compensates for these kinks in the manifold by predicting many more principal components.

This simulation is of course, a very rough sketch of what could be happening in reality. For example, the input signals, e.g. sensory signals, would be smoother with temporal correlations and each dimension might be correlated with each other. Also, the connectivity of the network may be more structured, instead of just random, to generate neural representations. As a result, we can expect that the dimensionality predicted by PCA would be lower. However, this numerical analysis still reveals an important insight: when the neural code is closed to being energetically efficient, a regime which we discussed earlier as being more plausible, as well as is bounded from below to be non-negative, the neural manifold can become non-trivially shaped so

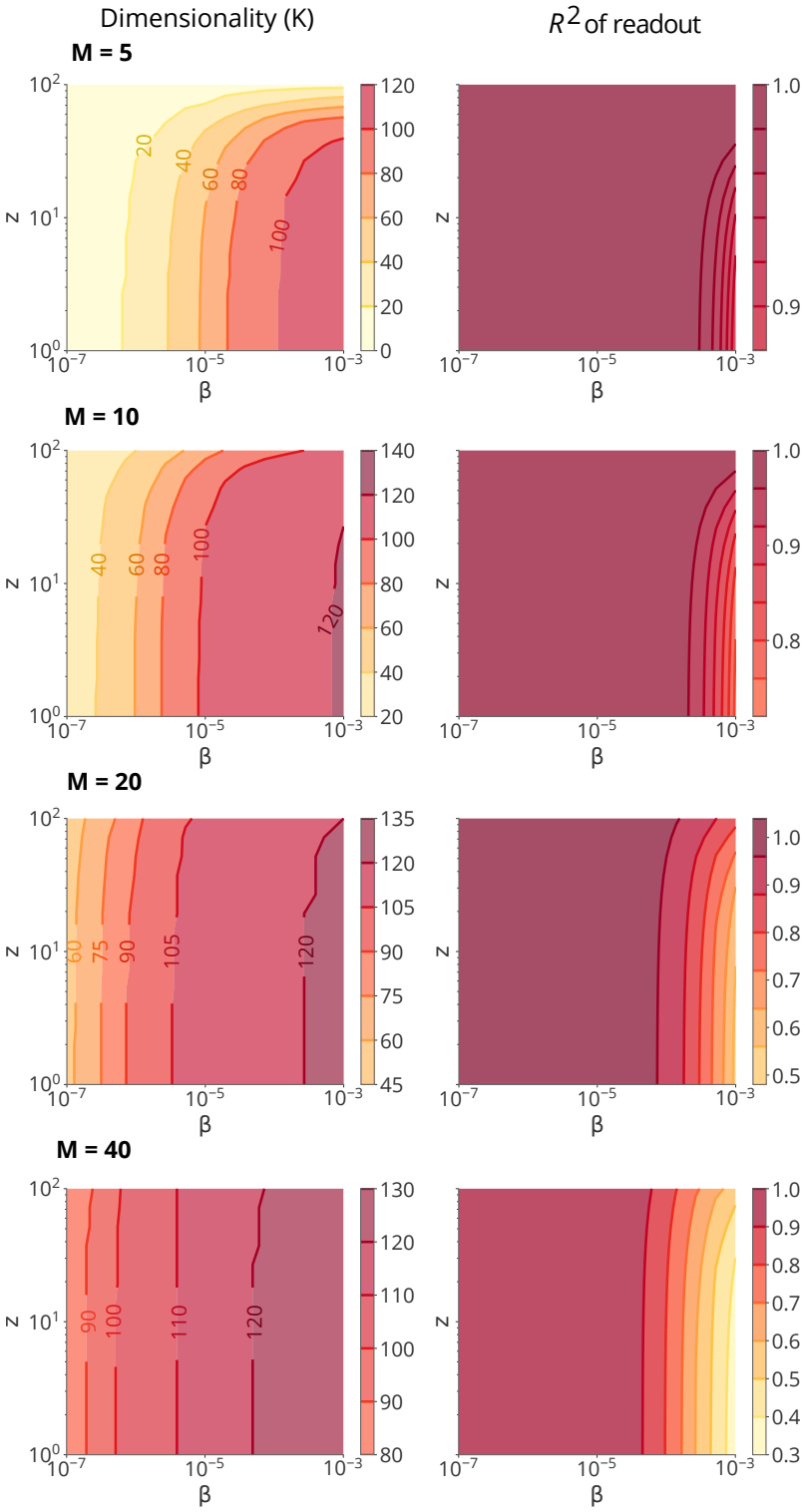

Figure 3.10: The dimensionality predicted by PCA is affected by the energy constraints of the neural code together with the non-negativity constraints. Population firing rates of $N = 200$ neurons are simulated using quadratic programming for inputs of varying dimensionality, $M = 5, 10, 20, 40$. (Left column) Contour plots of dimensionality as a function of parameters (β, z). The dimensionality as predicted by PCA is significantly overestimated for moderate values of the parameters. (Right column) Contour plots of the accuracy of the readout, measured by R^2. The accuracy gets worse for higher β.

as to maintain an accurate readout. As a result, linear methods that are agnostic to such constraints might give flawed interpretations of the data and in the context of dimensionality reduction, can significantly overestimate the intrinsic dimensionality of the neural manifold.

Additional details for the simulations: In the above analysis, $P = 1500$ points were sampled from a standard normal distribution for each input dimension. Similarly, the decoding weights, $\mathbf{D}$, were sampled from a standard normal distribution, but normalized and scaled across neurons so that all neurons had the same decoding vector length of 0.05 which yielded reasonable magnitudes for the firing rates. The firing rates were computed using quadratic programming for each input sample (an M-dimensional vector) while the parameters (z, β) were varied between 10^0 to 10^2 and between 10^-7 to 10^-3, respectively, on a logarithmic grid. The quadratic cost, μ that limits overall firing rates was fixed at $1e-05$ throughout.

3.6 Discussion

In this chapter, we showed that when linear readouts of population activity are low-dimensional and that the neural code faces energy constraints, the non-negativity constraints on neuronal activities lead to a curvature in the neural manifold. We explored geometrically the implications of such a curvature on the results of linear methods such as PCA. We showed in simulations using two different neural network models that, although the input signals to a neural network can be read out accurately with a linear decoder from the population activity, PCA has difficulty to extract the correct latent structures in the data, and often compensates for the curvature in the manifold with a set of 'higher-order' components which are functionally irrelevant.

While both network models we considered here incorporate the non-negativity constraints, they nonetheless have different predictions of how such constraints shape the neural manifold. In particular, one of the models we considered, generate firing rates according to an optimality principle between accurate representations and computational costs which can yield more complex neural representations. Upon a parameter space exploration, we showed that this model can yield non-trivially shaped neural manifolds embedded in the neural space. PCA consequently predicted a long tail of components to capture the non-linearities of the manifold, although a correct decoder could retrieve the inputs that generated the manifold. As such, these inputs defined the intrinsic dimensionality of the manifold, which was much

lower than the number of components predicted by PCA. Interestingly, a recent work by Stringer et al. (2019a) showed that when a variant of PCA was applied to recordings of large populations of neurons in the visual cortex of mice, a long tail of components was similarly observed. However, an explanation for how the network generates this potentially high-dimensional representation is still missing. It may be possible that the simple coding constraints as we discussed here could be a contributing factor to this result, although according to our theory the true (intrinsic) dimensionality would in fact be lower. This may be an interesting avenue for future research work.

However, as we discussed in Chapter 2, there may be other possible explanations besides coding constraints that would lead to curvatures in the neural manifold and hence, the observed 'higher-order' components in data. To address the validity of our theory, we seek to test it by considering dimensionality reduction methods that incorporate the coding constraints mentioned here and compare the results to a linear method such as PCA. In the next chapter, we set on this purpose by building dimensionality reduction methods that, similarly to PCA, assume that the latent variables can be estimated through a linear map, but predict the neural data with the non-negativity constraints incorporated in a meaningful way. If these constraints are indeed present in real data, these methods should be able to find a more succinct representation of the data as they would be able explain the neural manifold better with fewer latent variables.

Chapter 4

The need to incorporate coding constraints in dimensionality reduction methods

4.1 Introduction

The goal of dimensionality reduction methods when applied to neural data is to find a description of the recorded population activity of many neurons with a smaller number of explanatory variables. The hope is that these variables, often termed as latent variables, capture the essential structures in population data and are more easily interpretable. Most commonly, linear methods such as principal component analysis (PCA) are used and have successfully revealed important low-dimensional structures across several datasets through linear mappings of the measured neuronal activities, e.g. (Mazor and Laurent, 2005; Briggman et al., 2005; Machens et al., 2010; Bathellier et al., 2008). While neuronal activity (as measured, e.g. in firing rate) is non-negative by definition, these methods however, do not explicitly take this simplest form of non-linearity into account. As we discussed in Chapter 3, this non-linearity can lead to a tail of higher-order components which are functionally irrelevant. This is problematic as we risk overestimating significantly the (intrinsic) dimensionality of the data as well as misinterpreting what is being represented.

One way to address these issues is through the use of non-linear dimensionality reduction methods, but they usually come with difficulties of their own (Cunningham and Yu, 2014). For example, general-purpose methods such as Isomap (Tenenbaum et al., 2000) and Locally Linear Embedding (LLE) (Roweis and Saul, 2000) use local neighbours to compute low-dimensional embeddings of high-dimensional data

onto non-linear manifolds. However, for an accurate estimation of this manifold, the high-dimensional neural space needs to be evenly sampled, which is typically not the case in current neural datasets due to the restricted nature of standard experimental paradigms; e.g. the stimulus set or the elicited behaviour during the experiment may not be rich enough and thus, the evoked neural responses would more likely visit only part of the space. As a result, differences in the samples of neural activity may become overly emphasized in the low-dimensional embeddings and thus, can result in a distorted view of the manifold. Also, these methods are fragile in the presence of noise (Boots and Gordon, 2012) which is usually ubiquitous in neural data. For these reasons, they may not be ideal for analysing neural data.

In recent years, nonetheless, there has been significant effort to develop non-linear dimensionality reduction methods targeted for neural data, e.g. (Low et al., 2018; Pandarinath et al., 2018; Whiteway and Butts, 2017). Among these methods, several can be understood as autoencoder neural networks where the latent variables are estimated through a deterministic mapping from the measured population activity down to a bottleneck (decoding step). Then, for the given latent variables in the bottleneck, the population activity is reconstructed (encoding step) with the goal that it remains as closed as possible to the observed data. This consequently, results in bottlenecks that optimally compress the data given the assumptions of the model on the decoder and encoder. A common approach is to approximate the decoding and encoding functions e.g. by using recurrent neural networks (Pandarinath et al., 2018) or deep neural networks (Whiteway et al., 2019; Hinton and Salakhutdinov, 2006). From a machine learning perspective, this approach works well but does not not necessarily reveal the mechanisms that the brain could be using to give rise to these representations. One way to address this, is to add biology-inspired constraints in the methods and determine how well these constraints help in explaining the data.

In this chapter, we remain within the modelling framework of autoencoders for dimensionality reduction, but we go further by incorporating explicit constraints on the mappings of our autoencoders. For reasons we discussed in previous chapters of this thesis, we will assume that the latent variables can be estimated as low-dimensional linear readouts of the population activity, i.e. the decoder is linear similar to the PCA algorithm. Then, we will enforce the reconstruction of the data (encoder) to be non-negative. We will consider two ways of doing so based on network models in Chapter 3. As a reminder, the first approach will build on the standard linear-nonlinear (LN) network model that first maps the latent variables

linearly, followed by a non-linearity while the second approach explicitly enforces the reconstructed activities to be energetically efficient.

Finally, we check the predictions of incorporating such constraints in dimensionality reduction methods by validating them on both synthetic and and real data. We show that they can significantly outperform linear methods such as PCA simply because of the non-negativity constraints. Although both methods incorporate the non-negativity constraints, we further show that they can have different performances when fitting data as they make different predictions of how the non-negativity constraints shape the neural manifold.

4.2 Dimensionality reduction and latent variable models

One motivation for using dimensionality reduction methods when analysing neural data is that one suspects that the high-dimensional activity of many recorded neurons vary according to a smaller number of explanatory but unobserved variables, or latent variables. Successfully extracting these latent variables would then provide a succinct representation of the data. A classical approach for dimensionality reduction is through latent variable modelling, which uses probabilistic models to explain complicated statistical structures in the data according to an informative lower-dimensional set of latent variables. More concretely, a latent variable model assumes that observations of population activity, $\mathbf{r}$, are distributed conditionally to some unknown latent variables, $\mathbf{z}$, i.e. $p(\mathbf{r}|\mathbf{z})$ where p is some probability distribution.

Since the latent variables are not directly observed, they need to be estimated from the data. Latent variable models commonly approach this inference problem within a Bayesian framework. After defining a probabilistic model of the observations given the latent variables, i.e. $p(\mathbf{r}|\mathbf{z})$, the posterior distribution of the latent variables given the observations, i.e. $p(\mathbf{z}|\mathbf{r})$ is then inferred. This inference will depend, of course, on the chosen prior distribution of the latent variables, $p(\mathbf{z})$. However, defining the probability distributions to model neural data, especially the priors of the latent variables, and subsequently, fitting the model parameters and inferring the latent variables within this Bayesian framework may not be an easy task. In this chapter, we consider instead a simpler modelling approach, namely the autoencoder neural network. In particular, such networks need not impose any particular distributional assumptions on the latent variables.

4.3 Model formulation as autoencoders

The autoencoder neural network aims to reproduce its input after building an internal representation for it through a set of hidden layers. When unconstrained, the network learns a trivial identity transformation without necessarily capturing any interesting features in its input. However, one way to prevent the network to simply learn the identity function is to force the network to learn an under-complete representation of its input, so that the autoencoder then becomes a dimensionality reduction technique.

In the models that we consider here, the network receives as input some observed population activity, $\mathbf{r}$, and is constrained to produce an under-complete internal representation with a smaller number of latent variables, $\mathbf{z}$, than the number of recorded neurons. From this internal representation, the network is then tasked to yield as its output, a prediction of the population activity, $\hat{\mathbf{r}}$. Importantly, in contrast to latent variable models, the mapping from the observed activity to latent variables i.e. $\mathbf{z} = f_{dec}(\mathbf{r})$ (decoding step) and that from latent variables to reconstructed activity i.e. $\hat{\mathbf{r}} = f_{enc}(\mathbf{z})$ (encoding step) are deterministic functions f_{dec} and f_{enc}. Hence, the latent variables will no longer be random variables. Interestingly however, in many cases the latent variables estimated in this approach can be interpreted as the mean of some posterior distribution, $p(\mathbf{z}|\mathbf{r})$, in the latent variable model (Roweis and Ghahramani, 1999; Bengio et al., 2017).

4.3.1 Modelling latent variables

In the models that we build here, we will simply assume, in accordance with previous chapters, that the latent variables can be estimated through a linear map, i.e. f_{dec} is a linear function of the observed population activity. Note that in PCA, the latent variables or principle components are estimated similarly. Mathematically, for an observed data point, $\mathbf{r}_k$ where k is the index of the sample, we assume that the corresponding latent variables, $\mathbf{z}_k$ for that sample is given as,

$$\mathbf{z}_k = \mathbf{D}\mathbf{r}_k \tag{4.1}$$

With N recorded neurons of the population and M latent variables, $\mathbf{D}$ then corresponds to an $M \times N$ matrix of weights that determines how each observed neuronal activity needs to be combined to give rise to the latent variables. The number of latent variables, M, is a free parameter of the model and to reduce dimensionality, we need to set $M < N$. Since we mainly concern ourselves with electrophysiological data here, $\mathbf{r}$ can be interpreted as a vector of spike counts or firing rates of the N

recorded neurons and a data point can be for example, the measured activity of the population at a time point.

In this framework, the latent variables, $\mathbf{z}$, should correspond to all factors that drive neural activity. They could be related to observables in the experiment, e.g., stimulus, behaviour, pupil dilation (Stringer et al., 2019b; Musall et al., 2019; Kobak et al., 2019; Vinck et al., 2015) or to hidden variables, e.g., internal states of the animal such as thirst, motivation, attention that we cannot directly measure. However, we will not make any assumption here, about how they relate to experimental variables when finding them.

4.3.2 Modelling neural activity given the latent variables

Once some latent variables, $\mathbf{z}$, are estimated, the autoencoder then attempts to predict the population activity according to some function $f_{enc}(\mathbf{z})$. For mathematical convenience, methods such as PCA assume that f_{enc} is linear (Fig. 4.1A(i)). However, as we discussed in Chapter 3, this does not guarantee that the non-negativity constraints on neuronal activities would be satisfied. Thus, inspired by the neural network models we discussed earlier (see Chapter 3), we propose instead two different non-linear transformations during the encoding step such that the non-negativity constraints are satisfied. We describe these approaches in a dimensionality reduction context below.

Predicting firing rates from linear-nonlinear networks

A standard approach to satisfy the non-negativity constraints would be to apply a non-linearity to the predictions in the encoding step. A simple way would be to first map the latent variables linearly, followed by some non-linearity, e.g. a rectification. This guarantees that the modelled firing rates will remain above the zero-threshold. Thus, for latent variables, $\mathbf{z}_k$, (k being the index of the sample) the reconstructed neural activity can be modelled as,

$$\hat{\mathbf{r}}_k = \max(\mathbf{F}\mathbf{z}_k + \mathbf{b}, 0) \tag{4.2}$$

where $\mathbf{F}$ is an $N \times M$ matrix that determines the coupling weights between each latent variable and each neuron and $\mathbf{b}$ is a vector of bias terms that determines the baseline activity. The modelled firing rates are now threshold-linear functions of the latent variables, or f_{enc} will be defined according to equation 4.2. We illustrate these functions in Fig. 4.1B(ii).

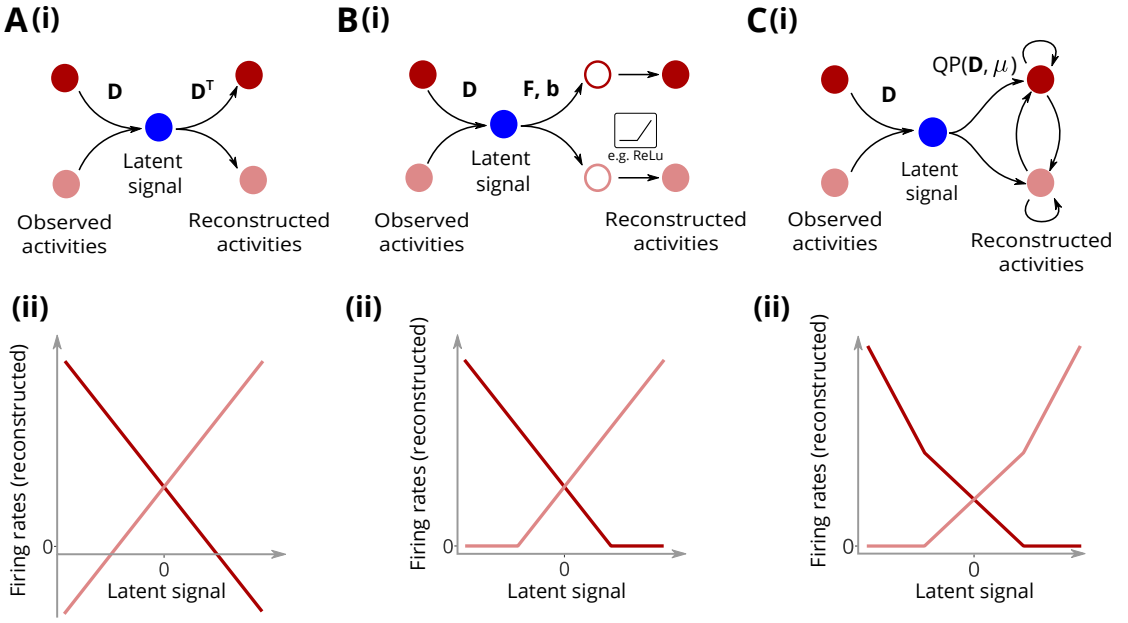

Figure 4.1: Illustration of the different autoencoders considered in this chapter. They all have a linear mapping from the observed population activity to the latent space, but differ according to their reconstructions of the data. (A)(i) PCA reconstructs the data linearly by taking the transpose of its decoder weights, $\mathbf{D}$. (ii) The resulting 'tuning curves' of the reconstructed neurons are linear as a function of some underlying latent signal and violate the non-negativity constraints on firing rates. (B)(i) The LN autoencoder adds a static non-linearity, e.g. 'ReLu', in its reconstruction. (ii) The resulting 'tuning curves' are now threshold-linear as the firing rates are rectified to be non-negative. (C)(i) The QP autoencoder reconstructs the population activity using an optimization procedure. The equivalent neural network approach for this reconstruction mediates shared information through lateral connections such that the predicted population activity remains optimal. (ii) Due to the lateral connections, kinks can appear in the 'tuning curve' of a neuron whenever some other neuron in the population hits the zero-boundary.

Combining this non-linear encoder with the linear decoder[1] (Fig. 4.1B(i)), this method then falls within the class of PCA-type non-linear dimensionality reduction methods, with explicitly defined non-linearities. Methods in this class have previously been shown capable of extracting features in the data that a standard PCA algorithm would not. For example, Karhunen and Joutsensalo (1994) showed that by just adding one non-linearity to the internal representations of the autoencoder, the resulting algorithm can also capture higher-order moments in the data, rather than just the first two moments. This consequently, enabled the method to find la-

[1]Note that we are departing slightly from the terminologies commonly used for autoencoders. We are defining, within a neuroscience context, the decoder as the transformation from observed neural activity to the latent variables and the encoder as the transformation from latent variables to predicted neural activity.

tent variables that were more independent, and not just uncorrelated. More recently, Whiteway and Butts (2017) proposed a method that rectifies the latent variables to be non-negative. Interestingly, they showed that this allowed the method to extract latent variables that could more easily be interpreted with regards to the experimental paradigm, as compared to PCA when applied to the same dataset. Note that in our method though, the non-linearity is only applied during data reconstruction.

However, modelling the firing rates in this way do not explicitly take into account possible energy constraints that the network faces. As we discussed previously (Chapter 3), these constraints can effectively change the nature of the neural representations. We next consider an alternative model for reconstructing the data. In particular, the model will not only satisfy the non-negativity constraints, but also limit overall network activities for energy efficiency reasons.

Predicting firing rates with energy efficiency constraints

In this model, the data is reconstructed according to an optimality principle. Similar to the neural network model we described in Chapter 3, the reconstructed activity should accurately represent the latent variables, but with limited firing rates. So, once some latent variables $\mathbf{z}_k$ have been estimated for a data point indexed k, the encoder attempts at reconstructing the observed sample of neural activity as the solution to the following optimization problem,

$$\hat{\mathbf{r}}_k = \underset{\tilde{\mathbf{r}} \geq 0}{\arg\min} \, \|\mathbf{z}_k - \mathbf{D}\tilde{\mathbf{r}}\|_2^2 + \mu \, \|\tilde{\mathbf{r}}\|_2^2 \tag{4.3}$$

Note that $\tilde{\mathbf{r}}$ is the optimization variable. So this optimization problem aims to find optimal firing rates, $\hat{\mathbf{r}}_k$, that would recover the previously estimated latent variables $\mathbf{z}_k$ (estimated from recorded neural activity, see above) through a linear readout, but are constrained by a quadratic cost so that high population firing rates are penalized. The cost-accuracy tradeoff is controlled by parameter $\mu > 0$. We point out that in this model, the f_{enc} function is defined implicitly within this optimization problem.

For given latent variables, $\mathbf{z}_k$, and parameters $(\mathbf{D}, \mu)$, the reconstructed firing rates are obtained by solving the corresponding quadratic program (QP) (equation 4.3). We can in turn think of a network implementation as in Chapter 3 to solve this QP. In particular, the properties of the neural representations are preserved, i.e. the firing rates of each reconstructed neuron will be piecewise-linear functions of the latent variables (see Fig. 4.1C(ii)). This is because, as we can recall, the optimization task is distributed across the network such that whenever a neuron hits the zero-

threshold, the other neurons in the network subsequently adjust their activities so as to maintain an optimal solution. This is possible due to the shared information mediated through lateral connections. We illustrate this network architecture in Fig. 4.1C(i). However, we point out that here, we will simply use a QP solver (Wright and Nocedal, 1999)) to find the optimal neuronal activities.

4.3.3 Learning model parameters and inferring latent variables

The complete autoencoders are obtained by combining the decoding and encoding steps. We will refer to the autoencoder with explicit non-linearity as the LN autoencoder (Fig. 4.1B) and the one with implicit non-linearity defined in an optimization problem as the QP autoencoder (Fig. 4.1C). We now turn to the problem of how to learn the model parameters and infer the latent variables.

In latent variable models, a common approach to do so is through the expectation-maximisation (EM) algorithm (Bishop, 2006), which is an iterative algorithm that cycles between two modes. In the first mode (E-step), the latent variables are inferred for some random initial set of model parameters. This is usually done through Bayesian inference as we discussed earlier. The second mode (M-step) attempts to optimize the model parameters to best explain the data, given the estimated latent variables in the E-step. The algorithm alternates between these two modes until some convergence criterion is met. In our deterministic autoencoder network models however, we do not need to resort to probabilistic inference. Rather, we will learn the model parameters through direct optimization of some loss function.

Here, we will use a standard loss function, similar to PCA, defined as squared reconstruction error between the predicted firing rates, $\hat{\mathbf{r}}$ and the observed data, $\mathbf{r}$, summed over all P samples. This loss function becomes,

$$L(\theta) = \sum_{k}^{P} \|\hat{\mathbf{r}}_k(\theta) - \mathbf{r}_k\|_2^2 \tag{4.4}$$

where θ denotes the set of parameters for each model. The autoencoder then aims to minimize this loss by optimizing the parameters. We describe next how this can be done.

For some initial decoder weights, $\mathbf{D}$, the latent variables, $\mathbf{z}$ (we omit the sample index here for simplicity), can be estimated from the observed data according to equation 4.1 (somewhat akin to the E-step). In fact, we will assume that the latent

variables should be uncorrelated similar to PCA, and thus we enforce the decoder weight matrix, $\mathbf{D}$, to be orthonormal i.e. $\mathbf{D}\mathbf{D}^\top = \mathbf{I}$. Note that for PCA, this constraint is necessary to find uncorrelated latent variables that explain the greatest amount of variance.

Then, the neuronal activities can be predicted according to encoding models (f_{enc}) that we described earlier, for the estimated latent variables. For the LN autoencoder, we can start with some random parameters $(\mathbf{F}, \mathbf{b})$ to do this computation (equation 4.2). This yields predicted neuronal activities, $\hat{\mathbf{r}}$, and thus, an error, $\mathbf{e} = \hat{\mathbf{r}} - \mathbf{r}$, can be determined. This error can then be back-propagated to update the model parameters, here $\theta = \{\mathbf{D}, \mathbf{F}, \mathbf{b}\}$, to minimize the loss function, and thus explain the observed data better (somewhat akin to the M-step). With the updated parameters, a new estimate of the latent variables is obtained, which gives a new prediction for the neuronal activities and hence, a new error that is back-propagated. This process is repeated until the global error over all samples is minimized. Note that we enforce the orthogonality constraint on $\mathbf{D}$ at each update for all models. We describe in more depth how the updates are computed to optimize the parameters in the Methods section below.

4.4 Results

4.4.1 Validation on synthetic data

To understand the nature of the solutions when incorporating a non-linearity in the reconstruction step of the autoencoders, we resort to synthetic data where the ground truth is known. We will simulate the data in two ways using the neural network models we described in Chapter 3: we first simulate firing rates with energy efficiency constraints, i.e. compute them using quadratic programming and second, we generate them through a linear-nonlinear mapping of the network's inputs.

In particular, we are interested in whether our methods can eliminate the need for the higher-order components that PCA produces to explain the data (see chapter 3), and thus get us closer to the true latent variables. Thus, we will compare the performance of our methods to PCA. Recall that all these methods, including PCA, estimate the latent variables through a linear map of the observed population activity, but only differ by how they reconstruct the data.

Since the autoencoders try to simultaneously estimate the latent variables (decoding step) and reconstruct the population activity (encoding step), we will test

the following two things:

1. We will determine whether our methods can correctly estimate the low-dimensional subspace for the true latent variables. We do so by measuring the principal angles between the true and estimated subspaces. Since each subspace will be characterized by a set of basis vectors, we compute the principal angles by measuring the angles between these vectors (see Methods for more details).

   We choose this measurement rather than, for example measuring the error between the predicted and true latent variables, since our methods can suffer from a rotational degeneracy, characteristic of models that involve matrix-vector multiplication as in equation 4.2 (for any orthogonal matrix $\mathbf{A}$, the solution $\mathbf{Fz}$ is equivalent to $(\mathbf{FA}^\top)(\mathbf{Az})$ since $\mathbf{A}^\top\mathbf{A} = \mathbf{I}$). Thus our methods can predict latent variables in the correct subspace, but be off from the true latent variables by a rotation. As a result, using a metric based on error can yield poor performance.

2. We will determine whether our methods can accurately reconstruct the observed data. To this end, we will measure the amount of variance that the model can explain from the data for a given number, M, of latent variables defining the bottleneck size of the autoencoder. By increasing this size and measuring the corresponding amount of variance explained, we can determine the number of latent variables needed to recover the full data. This number would then correspond to the dimensionality predicted by the methods, which can then be compared with the true dimensionality of the simulated data.

Data simulation using quadratic programming

In a first simulation, we generated firing rates with $M = 10$ latent variables which provided inputs to a network of $N = 100$ neurons. The first nine of these latent variables were modelled as filtered Gaussian signals while the last one was kept as a roughly constant positive signal corrupted with Gaussian noise to model the level of background activity. The firing rates were computed using quadratic programming (equation 4.3). The decoder weights used in the QP algorithm were also drawn randomly from a Gaussian distribution for the first nine dimensions and was kept all positive for the last dimension. This ensures that that background signal will be read out as a weighted average of the population activity (see Chapter 3). Further, the decoder weights were set to be orthonormal. See the Methods section for further details on this simulation.

We then fitted the parameters of PCA, LN and QP autoencoders to the synthetic data for different sizes of the bottlenecks in the models. Fig. 4.2A shows the relative performance of the methods in terms of the amount of variance they explain as a function of the number of components or latent variables. We observe that both the LN and QP autoencoders outperform PCA by explaining more variance in the data with fewer components. Although all the methods estimate the latent variables as linear projections of the data, the increase in performance by the LN and QP autoencoders is explained by the fact that they incorporate a non-linearity to satisfy the non-negativity constraints on firing rates when reconstructing the data.

The QP autoencoder inferred the correct number of latent variables, i.e. $M = 10$, which allowed it to reconstruct the data almost perfectly ($\sim 99\%$ explained variance). This is not surprising since the data was generated using the same mathematical framework in the first place. But, with its inferred ten latent variables, the LN autoencoder was also able to explain $\sim 80\%$ of the variance in the data, outperforming PCA which only explained $\sim 40\%$.

This simulation emphasizes the need of incorporating the non-negativity constraints in dimensionality reduction methods. Due to these constraints, the manifold was non-trivially shaped with high (linear) dimensionality, as determined by PCA, for its embedding in the neural space. However, incorporating the non-negativity constraints in our methods allowed to find a significantly better estimate of its intrinsic dimensionality, i.e. the number input variables used to simulate the data.

We remark nonetheless, that the QP autoencoder had worse performance than PCA and the LN autoencoder for models with one latent variable. This is because these latter methods have more degrees of freedom in fitting the data for each bottleneck. In the case of PCA, the data is centered beforehand and thus the PCA algorithm does not need to estimate this mean. We cannot use this centering procedure in the other autoencoders as this would lead to positive and negative values in the centered data, while the autoencoders try to reconstruct non-negative neuronal activities. In the case of the LN autoencoder, besides the weight parameters, it also learns a set of bias parameters, $\mathbf{b}$, when fitting the data and this equips the method with enough flexibility to capture any baseline activity or global mean of the data, even for models with one latent variable. Interestingly, despite having fewer parameters, the QP autoencoder quickly catches up with the LN autoencoder and even outperforms it as the number of components increases.

We next checked whether the different methods found the correct subspace for

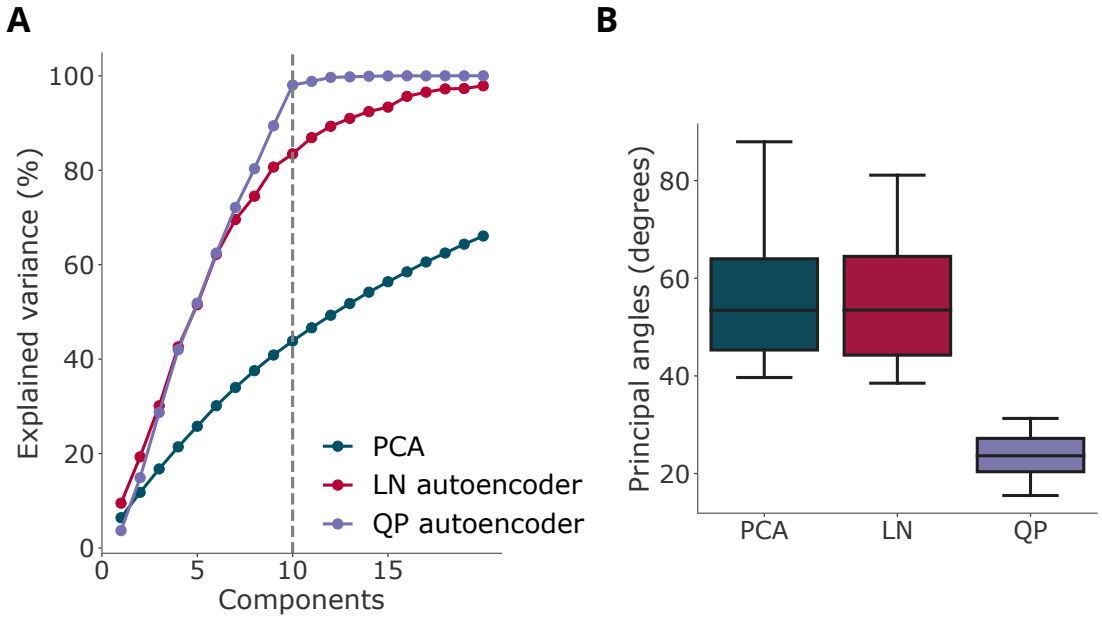

Figure 4.2: Validating the autoencoders on data simulated using quadratic programming (QP). The network of 100 neurons receives inputs from 10 latent variables. (A) Performance of the methods in explaining the data. By incorporating the non-negativity constraints, our methods significantly outperform PCA. The QP autoencoder finds the correct number of latent variables (i.e. $M = 10$) to fully explain the data, while the LN autoencoder explains 80% of variance with a bottleneck of 10 dimensions. (B) Comparing the subspaces found by the methods to the true subspace by measuring the principal angles between them. The decoder weights learnt by the model provided a basis for the estimated subspace, and thus were used to compute the angles. The QP autoencoder was able to find a subspace close to the true subspace of latent variables.

the ten latent variables. To do so, we measured the principal angles between the inferred subspaces by each method and the true subspace (Fig. 4.2B). We used the basis given by the fitted and true decoder weights to characterize these subspaces and measure the angles (see Methods). We observe that the QP autoencoder found a subspace that was close to the true one (although, not perfect) showing that the method can find a rough estimate of the true latent subspace as well as reconstruct the population data. Although the LN autoencoder and PCA found roughly similar subspaces (similar principal angles), the LN autoencoder reconstructed better the data as it satisfied the non-negativity constraints.

Data simulation using the linear-nonlinear network model

In a second simulation, we look at whether the autoencoders can explain the data when it is generated using the LN network model (see equation 4.2). The same

ten latent variables as in the previous simulations, provided inputs to this network (same size as before). For a better comparison to the previous simulation, we set the LN network model as an autoencoder. Instead of generating random firing rates, we trained the network parameters, i.e. encoding weights, $\mathbf{F}$, and bias, $\mathbf{b}$, such that the inputs should be retrieved through a linear readout of the generated neural activity according to the decoder weights, $\mathbf{D}$, used in the previous simulation (see Methods). We then applied the different autoencoders to this synthetic data.

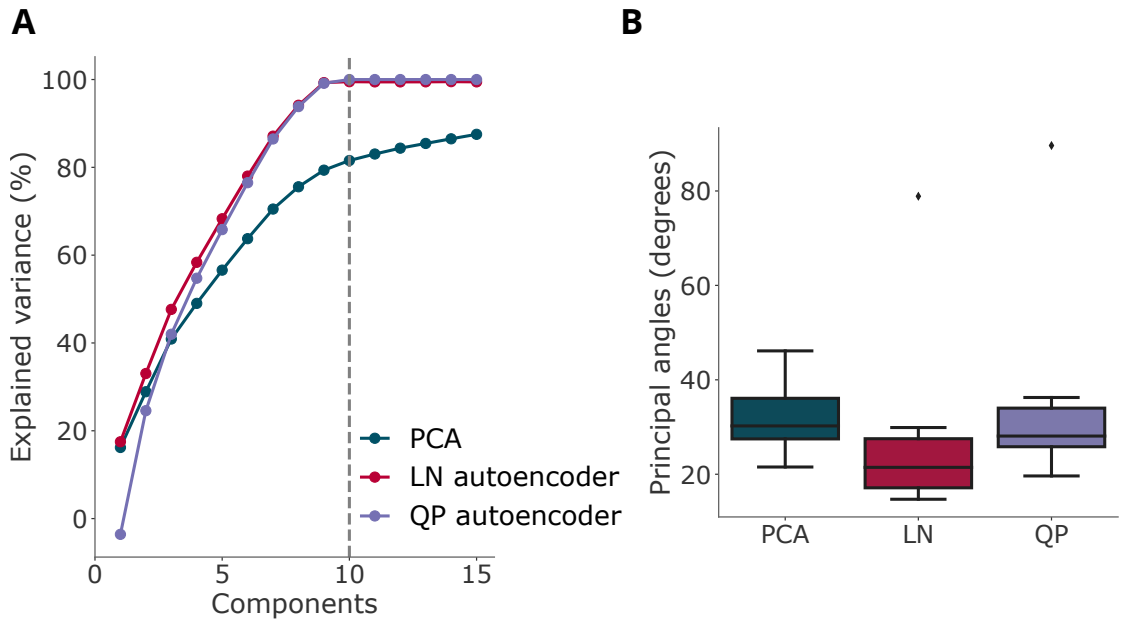

Figure 4.3: Validating the autoencoders on data simulated using the LN network model. The same 10 latent variables as in the simulation in (Fig. 4.2) provided inputs to a network 100 neurons. The network parameters were learnt so that the latent variables could be linearly read out from the population activity using the same decoder weights as in the previous simulation. (A) Model performance in explaining the data. Both the LN and QP autoencoders required the correct number of latent variables to explain the data, thus outperforming PCA. (B) Comparing inferred and true subspaces by measuring the principal angles between them.

In Fig. 4.3A, we observe unsurprisingly, that the LN autoencoder explains the data with the correct number of latent variables (i.e. $M = 10$). Interestingly however, the QP autoencoder was also able to explain fully the data with its ten inferred latent variables, despite its initial disadvantage that it has fewer free parameters compared to the other methods to fit the data. This observation, together with the fact that in the previous simulation the LN autoencoder did not manage to fully explain the data with its $M = 10$ latent variables (Fig. 4.2A), suggest that the QP autoencoder potentially has more representational power than the LN autoencoder. Although this still needs to be proven formally, which we keep for future

work, part of the explanation for this feature can be understood with respect to the geometric picture we obtained in Chapter 3 (Fig. 3.5) where we saw that the network model with neural activity computed by QP can display solutions that can only be approximated by the LN network model.

4.4.2 Validation on a dataset from the auditory cortex of rats

We now check if we can get more condensed representations for real population recordings simply by incorporating the non-negativity constraints in our methods while keeping the assumption that the latent variables should be obtained through a linear decoder. We analyse here an example experimental dataset (Kobak et al., 2019) that we presented earlier in Chapter 2. We choose this dataset as the neurons were recorded simultaneously and thus, the neural responses measured in a recording session should belong to the same underlying neural manifold. This can then be leveraged to capture any coordinated responses across the neurons in the population, and possibly lead to a better estimate of the neural manifold. Note that the other datasets that we considered in chapter 2 were obtained using single-unit recordings on multiple sessions and days and thus, the datasets may not reflect a common manifold as it could be changing e.g. due to cell death or re-wiring of the circuit.

In this dataset, we recall that population recordings were carried out in the auditory cortex (area A1) of anaesthetised rats while they were presented with noise bursts. The stimulus was parameterized according to inter-aural level difference (ILD) and absolute binaural level (ABL). Roughly $N \sim 100$ isolated neurons were recorded in a session and we analyse here 8 example sessions. In a session, a trial consisted of an ILD-ABL pair that was presented to the animal for 150ms. We choose to analyse the spike counts in a time window from 50ms to 100ms after stimulus onset, which results in an N-dimensional vector for each stimulus presentation. Since there were 36 different ILD-ABL combinations in this experiment, each presented 100 times, we thus obtain 3600 (36 x 100) trials. We then determine how well the different autoencoders can reconstruct the cross-validated data according to different number of latent variables which we vary between 1 and 40. As before, we choose the fraction of variance explained as the metric to evaluate performance. Since we do not know the number of 'true' latent variables in real data, we compare the performance of the different methods for each size of the bottleneck.

Fig. 4.4 shows the results of this analysis. We found that incorporating the

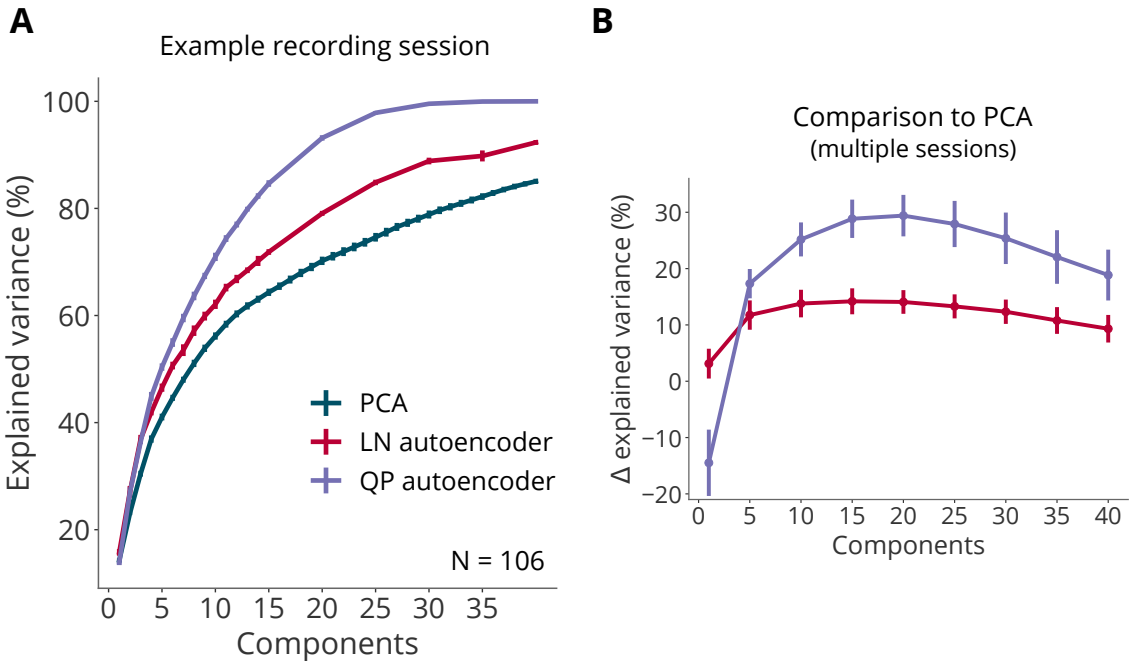

Figure 4.4: Validating the autoencoders with non-negativity constraints on an example dataset (Kobak et al., 2019). Neurons were recorded simultaneously in the auditory cortex of anaesthetized rats while noise bursts were played. We analysed 8 example recording sessions. (A) Performance on an example recording session of the methods in reconstructing the spike counts in a time window 50-100ms after the onset of the stimulus. The data was cross-validated (5-fold c.v.). Traces correspond to the mean over the folds (error bars are s.e.m.). Due to the predictions of the methods when incorporating the non-negativity constraints when reconstructing the data, they outperform PCA although they similarly estimated the latent variables as linear readouts of the population activity. Owing to the energy efficiency constraint in the QP autoencoder, it finds a closer match to the data. (B) Percentage difference in variance explained by the autoencoders over PCA as a function of the number of components in the models. The traces correspond to the mean across the 8 recording sessions (error bars are the s.e.m.). For models of larger bottlenecks, the QP autoencoder outperforms the LN autoencoder consistently across these recording sessions.

non-negativity constraints in the autoencoders generally allows them to explain more variance in the data with fewer number of latent variables, i.e. they can find a better match to the data with a fewer number of explanatory variables. We show the results on an example recording session in Fig. 4.4A. We see that the methods show roughly similar performance with models of less than 5 components, but a marked difference over PCA appears for higher dimensions. Interestingly, despite an initially similar performance, the QP autoencoder departs from the LN autoencoder as the number of latent variables increases ($M > 5$) and can almost perfectly reconstruct the data with roughly 25 components, thus reducing the dimensionality to around a quarter of the number of recorded neurons. This improvement in performance over the LN autoencoder was observed across several sessions (see supplementary Fig. 4.6).

We summarize the results of our analysis across multiple sessions in Fig. 4.4B where we computed the difference in the amount of variance explained by the non-linear autoencoders and PCA as a function of the number of components. By normalizing this difference by the variance explained by PCA, we obtain the percentage change in performance of the LN and QP autoencoders over PCA. The error bars are the standard error of the mean (s.e.m.) across sessions. We see that in general, incorporating the non-negativity constraints yields an improvement over PCA, although the QP autoencoder has negative performance for small bottlenecks. As we discussed previously, this is because the latter method faces the initial disadvantage that it has less degrees of freedom (fewer parameters) to fit the data compared to the other methods. Nonetheless, it quickly catches up and substantially outperforms both PCA and the LN autoencoder as the number of components in the model increases. We also observe that the percentage increase in performance eventually goes down for both methods since the additional amount of variance they explain with increasing number of dimensions saturates.

These results overall emphasize the importance of incorporating the non-negativity constraints in our methods when analysing neural data. Both the LN and QP autoencoders were able to explain the data better with fewer latent variables compared to PCA, although all the methods estimated the latent variables through a linear mapping. This shows that the (intrinsic) dimensionality of the manifold is in fact, much smaller than what PCA would predict and interestingly, the meaningful variables to explain the data can potentially still be obtained via a linear transformation.

Furthermore, similar to the results from the first simulated dataset, the QP autoencoder outperformed the LN autoencoder by explaining more variance in the data

for a given number of latent variables. As we showed previously, modelling the neuronal activities according to an optimality principle where an efficiency constraint is imposed, can yield different predictions of how the non-negativity constraints shape the neural manifold. Given that the QP autoencoder provided a better estimate to the data collected from the rats' auditory cortex, it might be the case that this brain region faces similar constraints when generating its representations.

4.5 Discussion

While each recorded neuron adds a dimension to the state space, the measured neural population activities usually lie along a manifold, which can usually be described according to a smaller set of latent variables. Dimensionality reduction methods aim to extract these variables for a succinct description of the data. In this chapter, we argued that these methods should incorporate the natural non-negativity of neuronal activities in order to find more accurate estimates of the latent variables underlying the data.

We considered two PCA-like dimensionality reduction methods that incorporate these constraints in a meaningful way. We described our methods within an autoencoder framework. Similar to the standard PCA algorithm, we assumed that the methods should estimate the latent variables as a linear combination of observed neuronal activities. However, unlike PCA, the methods should satisfy the non-negativity constraints when reconstructing the data given some estimated latent variables. In this chapter, we investigated two approaches of doing this reconstruction and hence, the two methods. The first method, namely the LN autoencoder, predicts non-negative neural activity by first mapping the latent variables linearly, followed by an explicit non-linearity (a rectification, here). A similar approach has previously been used in a dimensionality reduction context (Whiteway and Butts, 2017; Karhunen and Joutsensalo, 1994). The second method we proposed, namely the QP autoencoder, predicts the neural activity as a solution to a quadratic programming problem that aims to generate optimal neural representations that are energetically efficient. The non-negativity constraints are implemented as a hard constraints in the optimization problem, yielding an implicitly defined non-linearity.

We showed, in simulations, that incorporating the non-negativity constraints in our methods resulted in marked improvements in performance compared to the PCA algorithm. Although all the methods estimated the latent variables through a linear mapping, the LN and QP autoencoders were better at estimating the true (intrinsic)

dimensionality of the manifold. PCA, on the other hand, predicted a tail of principal components, significantly overestimating the dimensionality of the manifold and thus, pointing to a potential ambiguity when interpreting these components; had we not known the ground truth, it would be unclear how these components should be interpreted. It would be misleading to attribute functional meaning to them, since many of these components arise to compensate for the curvature of in the manifold. Put another way, these components provide a linear description of the space in which the manifold is embedded, but may not reflect the true underlying signal of the manifold.

We also validated our methods in an example dataset (Kobak et al., 2019) of simultaneously recorded neurons in the rats' auditory cortex. We showed that again the LN and QP autoencoders outperformed PCA. Interestingly though, we observed that the QP autoencoder could explain more variance in the data than the LN autoencoder, for comparative number of latent variables. Given the results from our numerical simulations, we suspect that this is because the QP autoencoder has more representational power than the LN autoencoder. This has previously been demonstrated but for more general quadratic programs (Amos and Kolter, 2017). It would be interesting in future work, to check whether the QP autoencoder generally outperforms the LN autoencoder across several datasets. This would then suggest that the brain might indeed be facing in its code, similar energy efficient constraints, which delineated the non-linearity of the QP autoencoder.

Finally, coming back to the experimental dataset that we analysed, we recall that the stimulus set presented to the animal during the neural recordings was only two-dimensional. However, even though our methods provided a more succinct description of the data than PCA, they still required many more latent variables than the dimensionality of the stimulus set to capture most variance. Thus, it is intriguing what these latent variables mean. One possibility could be that they represent other factors such as the internal states of the animal that are not directly observed during the experiment. These factors would still lead to variability in the neural recordings and thus, are captured as these additional latent variables.

However, it might also be the case that the neural manifold displays additional non-linearities, stemming for example from the dendritic tree (Stuart et al., 2016) or synapses (Markram, 2003) of individual neurons or from the connectivity structure of the network. As a result, the non-negativity constraints alone would be insufficient to explain these non-linearities, and consequently our methods would require additional (or functionally irrelevant) components to compensate for them. In other

words, had our methods been able to capture these additional non-linearities when reconstructing the data, it would require even fewer latent variables to explain the data. One way to remedy for this, could be to use a 'black-box' approach, e.g. a deep autoencoder (Hinton and Salakhutdinov, 2006), to find the smallest possible set of latent variables to explain the data. A downside of this approach however, is that the resulting mappings may not have any functional relevance for the neural circuit, and thus does not add to our understanding of how these non-linearities in the manifold emerge.

Alternatively, a hypothesis-driven approach can be used: we hypothesize in the rest of this thesis that, besides the non-negativity constraints, the network exhibits a non-linearity which allows it to perform some non-linear computations. This in turn, should be reflected in the shape of the neural manifold. However, to be able to explain how such computations shape the manifold, one must first understand the nature of computations that can be done by neural networks. In other words, we first need to characterize the specific non-linearity (if any) that emerges according to these computations. Yet, even today, a clear understanding of the computations done by networks of biophysical spiking neurons is still missing, although spikes are ubiquitous in the brain. In the next chapter, we switch gears and propose a new framework of how to understand some of these computations.

4.6 Methods

We describe our dimensionality reduction methods as autoencoder neural networks. The autoencoder network functions in two steps. It first estimates the latent variables, $\mathbf{z}_k \in \mathbb{R}^M$ through a deterministic map, f_{dec}, of its input sample of observed population activity, $\mathbf{r}_k \in \mathbb{R}^N$ where k is the index of the sample. A sample could be the measured population activity at a time point. N is the number of neurons recorded in the population and M is a free parameter determining the number of latent variables in the model. To reduce dimensionality, we set $M < N$ and thus M becomes the size of the bottleneck of the autoencoder. The second step in the functioning of the autoencoder is to predict as its output the neural activity, $\hat{\mathbf{r}}_k \in \mathbb{R}^N$ from the estimated latent variables, again through a deterministic map, f_{enc}.

In all the methods we consider here, we will assume that the latent variables are estimated through a linear mapping of the observed population activity, i.e. f_{dec} is linear. Specifically, we set

$$\mathbf{z}_k = \mathbf{D}\mathbf{r}_k \tag{4.5}$$

where $\mathbf{D} \in \mathbb{R}^{M \times N}$ is the decoder matrix. We consider next the two different models we used for reconstructing the data and describe in depth how to optimize their parameters to infer the latent variables and fit the neural data.

4.6.1 Fitting the LN autoencoder

Once some latent variables, $\mathbf{z}_k \in \mathbb{R}^M$, have been estimated ($k$ being the index of the data sample), the LN autoencoder attempts to reconstruct the observed sample of neural activities. It does so by using the linear-nonlinear (LN) model where the latent variables are first mapped linearly, followed by a non-linearity (i.e. f_{enc} mapping is given by the LN model). Here, we use a rectification function, $g(\mathbf{a}) = \max(\mathbf{a}, 0)$, which returns the element-wise maximum, as the non-linearity and thus, the predicted neural activity will be non-negative. The reconstructed neural activity is given as,

$$\hat{\mathbf{r}}_k = g(\mathbf{F}\mathbf{z}_k + \mathbf{b}). \tag{4.6}$$

where $\mathbf{F} \in \mathbb{R}^{N \times M}$ is a coupling matrix and $\mathbf{b} \in \mathbb{R}^N$ is a bias vector. Thus, an error, $\mathbf{e}_k = \hat{\mathbf{r}}_k - \mathbf{r}_k$, between the predicted and observed neuronal activities can be calculated.

The parameters of the autoencoder should be optimized to minimize this error over all data samples. We do so by minimizing the following loss function with

respect to the parameters:

$$\begin{aligned} L(\mathbf{F}, \mathbf{D}, \mathbf{b}) &= \sum_{k}^{P} \|\hat{\mathbf{r}}_k - \mathbf{r}_k\|_2^2 \\ &= \sum_{k}^{P} \|g(\mathbf{F}\mathbf{D}\mathbf{r}_k + \mathbf{b}) - \mathbf{r}_k\|_2^2 \\ &= \sum_{k}^{P} \|g(\mathbf{F}\mathbf{z}_k + \mathbf{b}) - \mathbf{r}_k\|_2^2 \end{aligned} \tag{4.7}$$

where $\|\mathbf{v}\|_2$ is the L2-norm of the vector, $\mathbf{v}$, and P is the total number of observed samples. We also include regularization terms for the model parameters, which prevents overfitting the training dataset and thus, allows to generalize better to the test data (Bishop, 2006). As a result, the loss function with regularization is augmented as follows:

$$L(\mathbf{F}, \mathbf{D}, \boldsymbol{b}) = \sum_{k}^{P} \|g(\mathbf{F}\mathbf{z}_k + \mathbf{b}) - \mathbf{r}_k\|_2^2 + \frac{\lambda_1}{2} \|\mathbf{F}\|_F^2 + \frac{\lambda_2}{2} \|\mathbf{D}\|_F^2 + \frac{\lambda_3}{2} \|\mathbf{b}\|_2^2 \tag{4.8}$$

where $\|\mathbf{A}\|_F$ is the Frobenius norm of the matrix $\mathbf{A}$ defined as $\|\mathbf{A}\|_F = \sqrt{\sum_i \sum_j (A_{ij}^2)}$. The hyper-parameter, λ_i, determines the strength of the regularization; as λ_i increases, it encourages the values in the parameters to be smaller. These hyper-parameters, λ_i, are determined using cross-validation as we describe in section 4.6.3 below.

Computing gradients for optimization

We now consider how this loss function can be minimized. We use a stochastic gradient descent method to optimize the parameters, $\theta = \{\mathbf{F}, \mathbf{D}, \mathbf{b}\}$, of the model to do so. Instead of computing the full gradient with respect to all data samples which is computationally costly, we consider only a few random samples or mini-batch in a learning epoch to compute a descent direction for this minimization. For a single data point (omitting the sample index for simplicity), it can be shown that the gradients will satisfy:

$$\frac{\partial L}{\partial \mathbf{F}} = \operatorname{diag}'(\mathbf{F}\mathbf{z} + \mathbf{b})\, \mathbf{e}\mathbf{z}^T + \lambda_1 \mathbf{F} \tag{4.9}$$

$$\frac{\partial L}{\partial \mathbf{D}} = \mathbf{F}^\top \operatorname{diag}'(\mathbf{F}\mathbf{z} + \mathbf{b})\mathbf{e}\mathbf{r}^\top + \lambda_2 \mathbf{D} \tag{4.10}$$

$$\frac{\partial L}{\partial \mathbf{b}} = 2 \operatorname{diag}'(\mathbf{F}\mathbf{z} + \mathbf{b})\, \mathbf{e} + \lambda_3 \mathbf{b} \tag{4.11}$$

We recall that $\mathbf{e}$ is the error between predicted and observed activity. Thus, to update the parameters, we are back-propagating the error for a given prediction.

diag' is a differential operator that takes an input vector of dimensionality, N, and returns a diagonal matrix of size $N \times N$ with diagonal entries, g_i', which is the derivative of the non-linearity function g, evaluated at each entry, i, of the input vector. Since g is the 'ReLu' function here, it is not differentiable at zero. We use a sub-differentiable approach common in neural networks literature by setting the derivative to be zero at zero (Hara et al., 2015). As a result,

$$g_i'(\mathbf{v}) = \begin{cases} 1, & \text{if } v_i > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.12}$$

Learning the parameters of the LN autoencoder

To start the learning, we first run a PCA on the data and initialise the weight parameters $(\mathbf{F}, \mathbf{D})$ with the weights obtained from the PCA (i.e. choosing the first M eigenvectors and its transpose). The bias parameters, $\mathbf{b}$ are initialised randomly. Given this initial decoder weights, a first estimate of the latent variables, $\mathbf{z}$ can be made according to equation 4.5, and thus an initial prediction can be computed (equation 4.6).

In a learning epoch, indexed n, the training data is shuffled and the samples are divided into mini-batches. For a given mini-batch we compute the gradient by taking sum of the partial derivatives (equations 4.9 to 4.11) over the samples in the mini-batch. We then update the parameters of the model according to the following rules:

$$\begin{aligned} \mathbf{D} &\leftarrow \mathbf{D} - \epsilon_1(n) \sum_j \frac{\partial L_j}{\partial \mathbf{D}} \\ \mathbf{F} &\leftarrow \mathbf{F} - \epsilon_2(n) \sum_j \frac{\partial L_j}{\partial \mathbf{F}} \\ \mathbf{b} &\leftarrow \mathbf{b} - \epsilon_3(n) \sum_j \frac{\partial L_j}{\partial \mathbf{b}} \end{aligned} \tag{4.13}$$

where the index, j runs over samples in the mini-batch. ϵ_i is the learning rate. To guarantee convergence of the algorithm, the learning rates need to decay over epochs. We use an adaptive learning rate, following the ADAM optimiser (Kingma and Ba, 2014), to do so.

Furthermore, we constrain the decoder weights, $\mathbf{D}$, to be orthonormal; after each update as in equation 4.13, we further project $\mathbf{D}$ on the closest point on the manifold of orthogonal matrices (Stiefel manifold). Thus, the decoder weights are

finally updated according to,

$$\mathbf{D}^\top \leftarrow \mathbf{D}^\top(\mathbf{D}\mathbf{D}^\top)^{-0.5} \tag{4.14}$$

Finally, with the updated parameters, we can make new inferences of the latent variables (equation 4.5), and new predictions of neural activity (equation 4.6) for another mini-batch of data points. We refer to this as the forward pass of the algorithm. Then, for these predictions, we can back-propagate the errors accrued to re-update the parameters. We call this second step the backward pass. We alternate between these forward and backward passes across all the mini-batches and then reshuffle the data on the next learning epoch. We repeat the above procedure over many epochs until the algorithm converges.

4.6.2 Fitting the QP autoencoder

Similar to the LN autoencoder, the QP autoencoder attempts at reconstructing the observed population activity after the latent variables, $\mathbf{z}_k$, have been estimated for a data point indexed, k. Instead of using an explicit non-linearity however, the QP autoencoder models the reconstructed neural activity as the solution to a quadratic program (i.e., the f_{enc} mapping is implicitly defined). Thus, the predicted neural activity is given as,

$$\hat{\mathbf{r}}_k = \arg\min_{\tilde{\mathbf{r}} \geq 0} E(\tilde{\mathbf{r}}) \tag{4.15}$$

where

$$E(\tilde{\mathbf{r}}) = \|\mathbf{z}_k - \mathbf{D}\tilde{\mathbf{r}}\|_2^2 + \mu \|\tilde{\mathbf{r}}\|^2 \tag{4.16}$$

is a quadratic objective function. Note that $\tilde{\mathbf{r}}$ is the optimization variable. Also, the non-negativity constraints are set as hard constraints in this optimization problem. To get a prediction, we solve the QP using a standard Python QP solver 'Quadprog' (Goldfarb and Idnani, 1983). As before, an error, $\mathbf{e}_k = \hat{\mathbf{r}}_k - \mathbf{r}_k$, between the predicted and observed neural activity can then be calculated.

To optimize the parameters of the autoencoder, we will use a similar loss function as in the LN autoencoder,

$$L(\mathbf{D}, \mu) = \sum_k^P \|\hat{\mathbf{r}}_k - \mathbf{r}_k\|_2^2$$

Note that this loss function implicitly depends on the parameters in the set $\theta = \{\mathbf{D}, \mu\}$ through the QP solution $\hat{\mathbf{r}}$. We next consider how these parameters can be learnt to minimize the loss.

Computing gradients for optimization

We learn the weight parameters $\mathbf{D}$ using again a stochastic gradient descent method. Since μ is only a positive scalar here, we optimize it using a cross-validation method instead (see section 4.6.3). This allowed us to minimize numerical instabilities. We describe next how we optimize $\mathbf{D}$ for a fixed μ.

Computing the gradients in this framework is a bit more involved than the LN autoencoder since the predicted activity is only obtained through an optimization problem and cannot easily be expressed analytically. To approach this learning problem, we first phrase it as a bi-level optimization problem where the upper level involves minimising the loss function with respect to the parameters $\mathbf{D}$ and the lower level is our QP, which is a minimisation problem of the function $E(\mathbf{r})$ with respect to non-negative arguments $\mathbf{r} > 0$. This can be written as:

$$\begin{aligned} &\underset{\mathbf{D}}{\text{Minimize}} \quad \left(L(\mathbf{D}) = \sum_{k}^{P} \|\hat{\mathbf{r}}_k(\mathbf{D}) - \mathbf{r}_k\|_2^2 \right) \\ &\text{subject to} \quad \hat{\mathbf{r}}_k(\mathbf{D}) = \arg\min_{\tilde{\mathbf{r}} \geq 0} E(\tilde{\mathbf{r}}) \end{aligned} \tag{4.17}$$

Recall that $\mathbf{r}_k$ is the observed data sample, $\hat{\mathbf{r}}_k$ is the predicted activity by the autoencoder and $\tilde{\mathbf{r}}$ is the optimization variable of the QP.

We consider now how the derivatives of the loss function with respect to $\mathbf{D}$ can be computed for a data point (omitting the index for simplicity). Given the loss function in the upper level problem, we can write its derivative as,

$$\frac{\partial L}{\partial \mathbf{D}} = 2\mathbf{e}\frac{\partial \hat{\mathbf{r}}}{\partial \mathbf{D}} \tag{4.18}$$

where $\mathbf{e}$ is the error as defined above for a sample. Thus, to compute this derivative, we will need to compute the derivative of the model prediction, $\hat{\mathbf{r}}$ with respect to $\mathbf{D}$. Since $\hat{\mathbf{r}}$ is the solution (fixed point) of the lower level problem (or QP), computing this derivative involves differentiating through an argmin operator.

While there are several ways to reach some form of differentiation through the argmin operator, we take a simple approach inspired from Gould et al. (2016). We will use an interior point method (Boyd et al., 2004) to reach an equation that an approximate QP solution would satisfy and then use ideas of implicit differentiation to compute the derivatives at the fixed points (QP solution) (Dontchev and Rockafellar, 2009).

In this interior point method, we aim to augment the loss function of the QP with its N inequality constraints, $\tilde{\mathbf{r}} \geq 0$, by means of a barrier function. We use

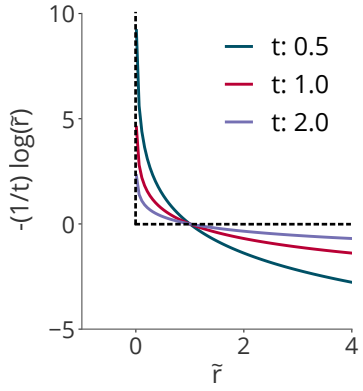

Figure 4.5: Illustration of the log-barrier function for one neuron. Figure adapted from Boyd et al. (2004). The inequality constraint, $r > 0$, can be made implicit in the objective of the QP via an indicator function which returns 0 whenever $r > 0$, else it is infinite. This is shown by the dashed line. The log-barrier relaxes this function. The bigger the hyper-parameter, t, is, the closer it is to the indicator function and thus, the closer we are to the original QP.

the following log-barrier function $-\frac{1}{t}\log(\tilde{r})$ to relax an inequality constraint; $\tilde{r}$ is an element of the vector, $\tilde{\mathbf{r}}$. We illustrate this function in Fig. 4.5). The parameter, t, controls the steepness of the function. The bigger the value of t, the closer is the function to an indicator function, and thus becomes a better approximation to the inequality constraint. By introducing a log-barrier for each neuron, we can approximate the QP problem as,

$$\underset{\tilde{\mathbf{r}}}{Minimize} \quad \left(tE(\tilde{\mathbf{r}}) - \sum_{j}^{N} \log(\tilde{r}_j) \right) \tag{4.19}$$

where we multiplied the objective by t to simplify notations (this does not change the argmin solution). Note that this optimization problem will favour $\tilde{r} > 0$ as when $\tilde{r}$ approaches zero, the value of the objective rapidly increases. The parameter, t, can also be interpreted as controlling the penalty in violating the non-negativity constraints.

Given our new objective function, we can now find an equation for its fixed point. We do so by taking its derivative w.r.t. $\tilde{\mathbf{r}}$ and setting to zero. For the sake of clarity, we will use the notation $\boldsymbol{v}^{-1}$ to denote a vector with element $\frac{1}{v_i}$ at the i^{th} entry. Then, we can show that the fixed point, $\hat{\mathbf{r}}$ of the approximate QP problem (equation 4.19) satisfies,

$$t\Big([\boldsymbol{D}^T\boldsymbol{D} + \mu\boldsymbol{I}]\hat{\boldsymbol{r}} - \boldsymbol{D}^T\boldsymbol{z}\Big) - \hat{\boldsymbol{r}}^{-1} = 0 \tag{4.20}$$

Note that we made a slight abuse of notation since $\hat{\mathbf{r}}$ now corresponds to the approximate QP solution that will depend on t, and thus may slightly differ from the

true QP solution.

Finally, we can take derivatives on both side of equation 4.20 with respect to $\mathbf{D}$ to evaluate $\frac{\partial \hat{\boldsymbol{r}}}{\partial D}$. We can then plug this back in equation 4.18 to get the gradients $\frac{\partial L}{\partial \mathbf{D}}$ that we use to optimize the parameters, $\mathbf{D}$. We show in Appendix A the full derivation for this gradient.

Learning the weights of the QP autoencoder

As before, to start the learning, we initialise the weight parameters with those obtained from a PCA on the dataset. This gives us a first estimate of the latent variables (equation 4.5) and a prediction for the neural activity (equation 4.15).

In a learning epoch, we shuffle the data and divide it into mini-batches. For a given mini-batch, we compute the gradients (equation 4.18) and update the parameters $\mathbf{D}$ as follows,

$$\mathbf{D} \leftarrow \mathbf{D} - \epsilon(n) \sum_k \frac{\partial L_k}{\partial \mathbf{D}} \tag{4.21}$$

We then orthogonalize $\mathbf{D}(n+1)$ by projecting it on the Stiefel manifold according to equation 4.14. We again use an adaptive learning rate, ϵ, that changes with learning epoch, n.

Finally, with the updated parameters, we can make new inferences of the latent variables (equation 4.5)and new predictions of the neural activity (equation 4.15). This allows us to compute the new error which we can back-propagate to re-update the parameters. As in the LN autoencoder, we repeat this cycle between forward pass and backward pass over many epochs until convergence on the training data.

4.6.3 Models evaluation

Cross-validation

To select the regularization parameters in the LN autoencoder and the quadratic cost term, μ, in the QP model, we use a 5-fold cross-validation method. After shuffling the data, it is divided into 5 chunks of roughly equal size, with 4 used for training and 1 for testing the model predictions. By taking different combinations of train and test sets, we get 5 non-overlapping testing blocks.

For the LN autoencoder, we assume that $\lambda_1 = \lambda_2 = \lambda_3 = \lambda$ and run the autoencoder for 5 values of λ between 10^{-7} to 10^{-3} on a logarithmic grid. For each

λ, we learnt the parameters on the training set and measured mean squared error of the reconstructed population activity on the test set. We repeated this procedure five times for the different train-test fold and averaged the resulting mean squared errors. We selected the optimal λ which gave the minimum error.

Similarly, for the QP autoencoder, we optimize the parameter μ using cross-validation. μ was chosen from 5 values between 10^{-7} to 10^{-3} on a logarithmic grid. Note that a model is defined for each dimensionality, M, and thus, this procedure is repeated across models with different sizes of the bottleneck.

Measuring performance

To assess the quality of the model fits, we first measured how well the models can explain the observed population activity. For all models, the parameters were first learnt on the training data. Then, using these learnt parameters and cross-validated hyper-parameters, the predicted neural activities on the test data, $R_{recons.}$ were computed. Note that we also did the train-test splitting of the data when running PCA, where the mean of the data and weight parameters were evaluated on the training set.

To evaluate the goodness-of-fit of a model with bottleneck size M, we measured how much variance, V, it explains, which is defined as,

$$\text{explained variance}, V = \frac{\|R_{test}\|_F^2 - \|R_{recons.}\|_F^2}{\|R_{test}\|_F^2} \tag{4.22}$$

where R_{test} is a $P_{test} \times N$ matrix containing the test set. P_{test} is the number of data points in the test set. Note that $R_{recons.}$ changes as a function of the size of the bottleneck in the model and thus, the explained variance, V is a function of M. We do this evaluation for both synthetic and real data over different values of M.

To determine the change in performance of either the LN or QP autoencoder over PCA, we computed the percentage difference in explained variance captured by the LN or QP autoencoder and PCA as follows:

$$\Delta V = 100 \times (V_{LN/QP} - V_{PCA})/V_{PCA} \tag{4.23}$$

A positive ΔV implies that the LN or QP autoencoder explains more variance than PCA in the data for a given bottleneck of size, M.

Since the simulated data provided us with the ground truth, we also evaluated whether the methods were able to retrieve the correct subspace. Since the methods

faced a rotational degeneracy, similar to PCA, we did not directly compare the predicted latent variables with the true latent variables in the simulations. First, we determined whether the correct dimensionality was predicted, simply by looking at how much variance each model was able to capture and then determining if the model with the right bottleneck size explained all the variance in the data. Second, to determine how close the subspaces were, we measured the principal angles between them. We characterized a subspace by the row space defined by fitted decoder weights of each method. The dimensionality of that subspace corresponds to the rank of the decoder weights. Then, the principal angles between two subspaces were determined by finding recursively the minimum angle between pairs of basis vectors from either subspace (Knyazev and Argentati, 2002). This was implemented using Python's Scipy module (scipy.linalg.subspace_angles).

4.6.4 Simulating synthetic data

To validate our methods, we simulated two sets of ground truth data, each using a different network model described in Chapter 3. Both networks consisted of $N = 100$ neurons receiving as inputs $M = 10$ latent variables, $\mathbf{z}$ which we modelled as follows: Each of the first nine dimensions was constructed by drawing $P = 2500$ random samples from a Gaussian distribution with zero mean and unit variance, i.e. $\mathcal{N}(0, 1)$, and then filtered. The last latent variable was set at a constant $z = 1$, but corrupted with white noise (samples drawn from the following Gaussian $\mathcal{N}(0, 0.3)$).

Then, we designed the readout weights $\mathbf{D} \in \mathbb{R}^{M \times N}$ as follows: each dimension (or row of $\mathbf{D}$) was drawn from a Gaussian distribution with the first nine from $\mathcal{N}(0, 1)$ while the last one from $\mathcal{N}(1, 0.2)$. The latter decoder weights were all positive. As a result, the last latent variable can be interpreted as a background signal obtained as the weighted average of the population activity. Finally, the weights $\mathbf{D}$ were orthonormalized according to equation 4.14. The decoder weights for the background signal remained all positive after this procedure.

In the first simulation, the firing rates were computed using quadratic programming (QP) (see equation 4.3) with decoder weights, $\mathbf{D}$. The parameter controlling the quadratic cost to penalize high firing rates was set as $\mu = 1e-05$. In the second simulation, the firing rates were generated using the linear-nonlinear (LN) network model (see equation 4.2. The parameters of the model, namely the feedforward weights $\mathbf{F}$ and bias, $\mathbf{b}$, were learnt such that the latent variables, $\mathbf{z}$ could be read out linearly from the generated population activity using decoder weights, $\mathbf{D}$. Split-

ting the decoder weights as $\mathbf{D} = \begin{bmatrix} \tilde{\mathbf{D}} \\ \mathbf{u}^\top \end{bmatrix}$ where $\mathbf{u}$ is a vector containing the decoder weights for the background signal, these parameters were learnt by minimizing the following loss function

$$L(\boldsymbol{F}, \boldsymbol{b}) = \sum_{i}^{P} \left(||\boldsymbol{z}^i_{1:M-1} - \tilde{\boldsymbol{D}} g(\boldsymbol{F}\boldsymbol{z}^i + \boldsymbol{b})||_2^2 + \beta ||z^i_M - \boldsymbol{u}^T g(\boldsymbol{F}\boldsymbol{z}_i + \boldsymbol{b})||_2^2 \right) \quad (4.24)$$

Superscript i corresponds to the index of the data point while the subscript $1 : M-1$ selects the first $M-1$ elements of the M-dimensional vector $\mathbf{z}$.

4.A Supplementary figure

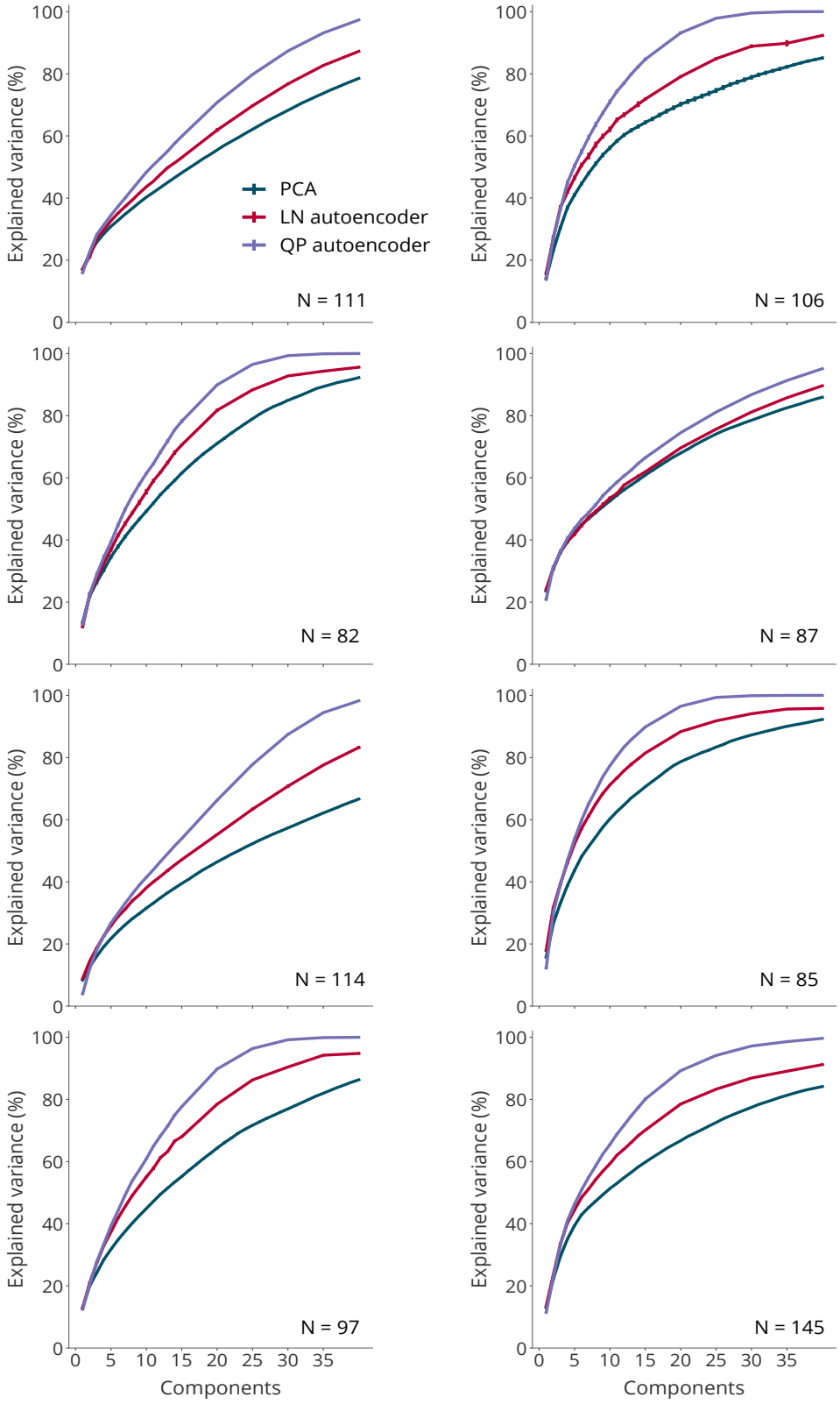

Figure 4.6: Performance of the dimensionality reduction methods on the experimental dataset (Kobak et al., 2019), across eight different recording sessions.

Chapter 5

Understanding computations by spiking neural networks

5.1 Introduction

The brain can generate internal representations of features in the world and compute over these representations to yield meaningful outputs, e.g. behavioural outcomes. These representations and computations are essentially mediated by sparse and irregular spiking activity of networks of neurons. In the previous chapters, we primarily investigated neural manifolds generated by network models using rate-based units (that give continuous-valued approximations to spiking activities), and showed that the manifolds can exhibit curvatures due to coding constraints. However, the coding constraints might not be the only source for the non-linearity along the neural manifold. Another possibility could be that the manifold reflects some additional non-linearity that arises according to the underlying computations done by the network. However, to start understanding the implications of network computations on the shape of neural manifolds, one must first understand the nature of computations that can be done by the network.

Although spikes are ubiquitous in the brain, even today, much of neuroscience theory still relies on models using rate-based units to understand core network computations (e.g. (Rubin et al., 2015; Sussillo et al., 2015; Mastrogiuseppe and Ostojic, 2018)). Understanding the computations in spike-based network models, or Spiking Neural Networks (SNNs), has proven more challenging. While SNNs have long been established as computationally powerful (Maass, 1997), only recently have they started to display competitive results to standard rate-based networks in machine learning applications. This has partly been due to novel and efficient spike-based

learning rules (Bohte et al., 2002; Huh and Sejnowski, 2018; Zenke and Ganguli, 2018), but mainly through transferring insights from rate to spiking networks (Eliasmith and Anderson, 2004; Stöckl and Maass, 2020; Sorbaro et al., 2020). Thus, even if these SNNs can now be trained to achieve high performance in solving some tasks, this approach does not necessarily yield any further insights on the underlying computations compared to their rate-based counterparts.

A key hurdle has been that spiking networks are hard to treat analytically due to the discrete nature of spike events. Major insights have often been limited to bottom-up approaches based on randomly connected networks (Vreeswijk and Sompolinsky, 1996; Brunel, 2000; Maass et al., 2002), or to single neuron computations based on spike-timing (Hopfield, 1995; Gütig and Sompolinsky, 2006; Gütig, 2016). An alternative way to understand SNN computations has been to derive spiking behavior directly from some loss functions with biology-inspired constraints (Hu et al., 2012; Boerlin et al., 2013; Tang et al., 2017; Pehlevan, 2019). Using this approach, it has been shown that the time-averaged activity of some SNNs can solve some underlying convex optimization problems. For instance, the networks can be designed to efficiently represent their inputs as an auto-encoder by solving a quadratic program (QP) (Barrett et al., 2013; Moreno-Bote and Drugowitsch, 2015) or learn sparse representations by solving a linear program (LP) (Chou et al., 2018).

In this chapter, we attempt at reconciling the computational power of SNNs with an understanding of the nature of these computations. We build on the aforementioned approaches, as well as some recent geometric insights (Calaim et al., 2020), to propose a new framework based on convex optimization theory to understand SNN computations. We show that virtually all input-driven (or inhibition-dominated) SNNs (Abbott et al., 2016) are intimately tied to convex optimization problems (Boyd et al., 2004), with network connectivity, timescales, and firing thresholds being intricately linked to the parameters of the underlying optimization problems (section 5.2). In particular, we show that our framework can capture as a specific parametrization, the autoencoder neural network with optimal representations that we discussed earlier (see Chapter 3, section 3.4), but with the optimization now carried out using spikes. As a result, the arguments on how the neural manifold is shaped due to coding constraints would still hold in this framework, albeit the presence of noise due to spikes.

However, we can go beyond computing the identity function of the autoencoder (since we would only be representing the input) and show that this perspective from convex optimization affords new geometric insights into more general computations

performed by SNNs. Using these insights, we clarify the input-output functions of input-driven SNNs that solve convex optimization problems (section 5.3.1). We show geometrically that such SNNs effectively compute convex, piecewise-linear functions according to their connectivity, and thus can approximate any convex input-output function. The computational power of convex input-output transformations in a network layer has previously been demonstrated in a pure machine-learning context (Amos and Kolter, 2017; Agrawal et al., 2019), and without a specific network implementation in mine. We here show that SNNs provide a natural implementation of such transformations.

Furthermore, we show that the geometric perspective of our framework enables us to understand SNNs beyond solving convex optimization problems. First, we show that we can understand more general network dynamics, e.g. bi-stability and periodicity, of SNNs which are not necessarily input-driven (section 5.3.2). Second, we illustrate how local learning rules to implement a given computation can be derived (section 5.4). Ultimately, we show that the resulting SNNs display many features from biological networks such as irregular and asynchronous firing patterns and robustness to various perturbations. We therefore, propose that such networks are closer to biology than standard rate-based networks and point to the possibility that biological networks could be doing similar computations.

Notes and acknowledgements. This chapter is largely adapted from a previously published work (Mancoo et al., 2020). The author points out that he has worked closely with Sander Keemink, in the lab of Christian Machens, to arrive at the results presented here. In particular, Sander has provided important insights on the computations done by SNNs (section 5.3) and showed how the network can solve a classification task, whilst exhibiting biological features (Fig. 5.8). After a joint effort to understand different network behaviours in this framework (section 5.3.2), the simulations for Fig. 5.5 were originally designed by him.

5.2 Spiking neural networks and convex optimization

In this section, we aim to provide the link between spiking neural networks (SNNs) and convex optimization problems. SNNs can be modeled with a broad range of neuron models. In this chapter, we focus on arguably the most common model, namely Leaky Integrate-and-Fire (LIF) neurons.

5.2.1 Leaky integrate-and-fire neurons

The LIF neuron is one of the simplest models to capture the core principles underlying spiking biophysical neurons. A network of N such neurons is governed according to following two key ingredients:

- First, the membrane potentials, $V(t) \in \mathbb{R}^N$, of the neurons are described as evolving according to a differential equation, where t stands for time,

$$\dot{\mathbf{V}}(t) = -\lambda \mathbf{V}(t) + \mathbf{F}\mathbf{c}(t) + \mathbf{\Omega}\mathbf{s}(t) + \mathbf{I}_{bg}(t), \tag{5.1}$$

  Parameter λ determines the membrane leak time-constant, $\mathbf{c}(t) \in \mathbb{R}^K$ is a K-dimensional time-varying input, $\mathbf{F} \in \mathbb{R}^{N \times K}$ are the feed-forward weights, $\mathbf{\Omega} \in \mathbb{R}^{N \times N}$ are the recurrent weights, $\mathbf{s}(t) \in \mathbb{R}^N$ are the neural spike trains described as a sequence of firing in time and modelled as a sum of delta-functions, $s_i(t) = \sum_{t_j} \delta(t - t_j)$, and $\mathbf{I}_{bg}(t) \in \mathbb{R}^N$ are background currents or noise.

- Second, each neuron has a threshold, T_i, and it emits a spike whenever its membrane potential, V_i, crosses its threshold. Then, immediately after the neuron spikes, its voltage is reset to a resting potential, R_i, which here is implicitly implemented in the diagonal terms of $\mathbf{\Omega}$ which are all negative, so that $R_i = T_i - \Omega_{ii}$.

We illustrate this process for an integrate-and-fire neuron in Fig. 5.1.

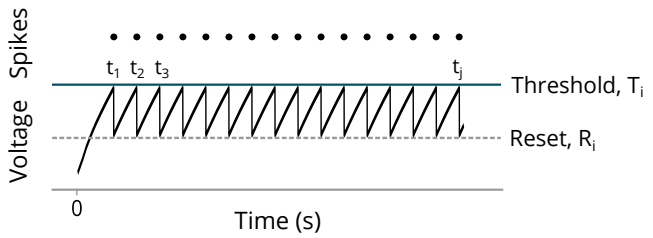

Figure 5.1: Illustration of an integrate-and-fire neuron. Its membrane potential or voltage is integrated over time in the presence of a constant driving input until it reaches a threshold, T_i. A spike is subsequently emitted that resets the membrane potential to R_i. The spike train (black dots) is a record of spiking events at different time points, t_j.

5.2.2 Linear and quadratic programming and their geometry

The key insight that we bring forward here is that the dynamics described above and resulting computations of SNNs can be examined through the lens of a convex optimization problem with inequality constraints. In fact, inequality constraints have a natural correlate in spiking networks since the membrane potentials of neurons are bounded from above by their thresholds.

We start with a fairly generic optimization problem,

$$\begin{aligned} &\underset{\mathbf{y}}{\text{Minimize}} \quad \left(E(\mathbf{y}) = \frac{\lambda}{2}\mathbf{y}^\top\mathbf{y} + \mathbf{b}^\top\mathbf{y} \right) \\ &\text{subject to} \quad \mathbf{F}\mathbf{x} - \mathbf{G}\mathbf{y} \leq \mathbf{T} \end{aligned} \tag{5.2}$$

where $\mathbf{y}$ is an M-dimensional optimization variable , $\mathbf{b}$ is a bias, and $\mathbf{x}$ is a K-dimensional input to the problem. The remaining variables, $\mathbf{F} \in \mathbb{R}^{N\times K}, \mathbf{G} \in \mathbb{R}^{N\times M}, \mathbf{T} \in \mathbb{R}^N$ and the positive scalar λ are parameters of the optimization problem. This optimization problem is convex as both the objective function and the constraints form convex sets. In particular, this type of optimization problem is generally referred to as a quadratic program (QP) — a quadratic objective function with affine constraints. If $\lambda = 0$, then the optimization problem becomes a linear program (LP) — a linear objective function with affine constraints (Boyd et al., 2004). Importantly, due to the hard inequality constraints, not all minimizers of the objective function are permissible; the inequality constraints define a feasible set (see Fig. 5.2 as an example) and the minimizer must belong to that set.

Before mapping this optimization problem onto the spiking network, equation (5.1), we first review its geometric interpretation. We can rewrite the i-th inequality constraint as $\mathbf{G}_i^\top\mathbf{y} \geq \mathbf{F}_i^\top\mathbf{x} - T_i$, where the (column) vectors $\mathbf{F}_i$ and $\mathbf{G}_i$ refer to the i-th rows of $\mathbf{F}$ and $\mathbf{G}$, respectively. Each inequality constraint thereby divides the $\mathbf{y}$-space (where the optimization variable resides) into two half-spaces. The boundary is defined at equality. Fig. 5.2A shows an example for a two-dimensional optimization variable ($M = 2$) with three constraints ($N = 3$). Each colored region shows the half-space where the corresponding inequality constraint is violated. The union of colored half-spaces thus defines the infeasible set for the optimization problem, and the white area is the feasible set. The QP solution, $\mathbf{y}^\star$, should correspond to the closest point to the unconstrained solution $\frac{\mathbf{b}}{\lambda}$ but within the feasible set (Fig. 5.2A, red star.)

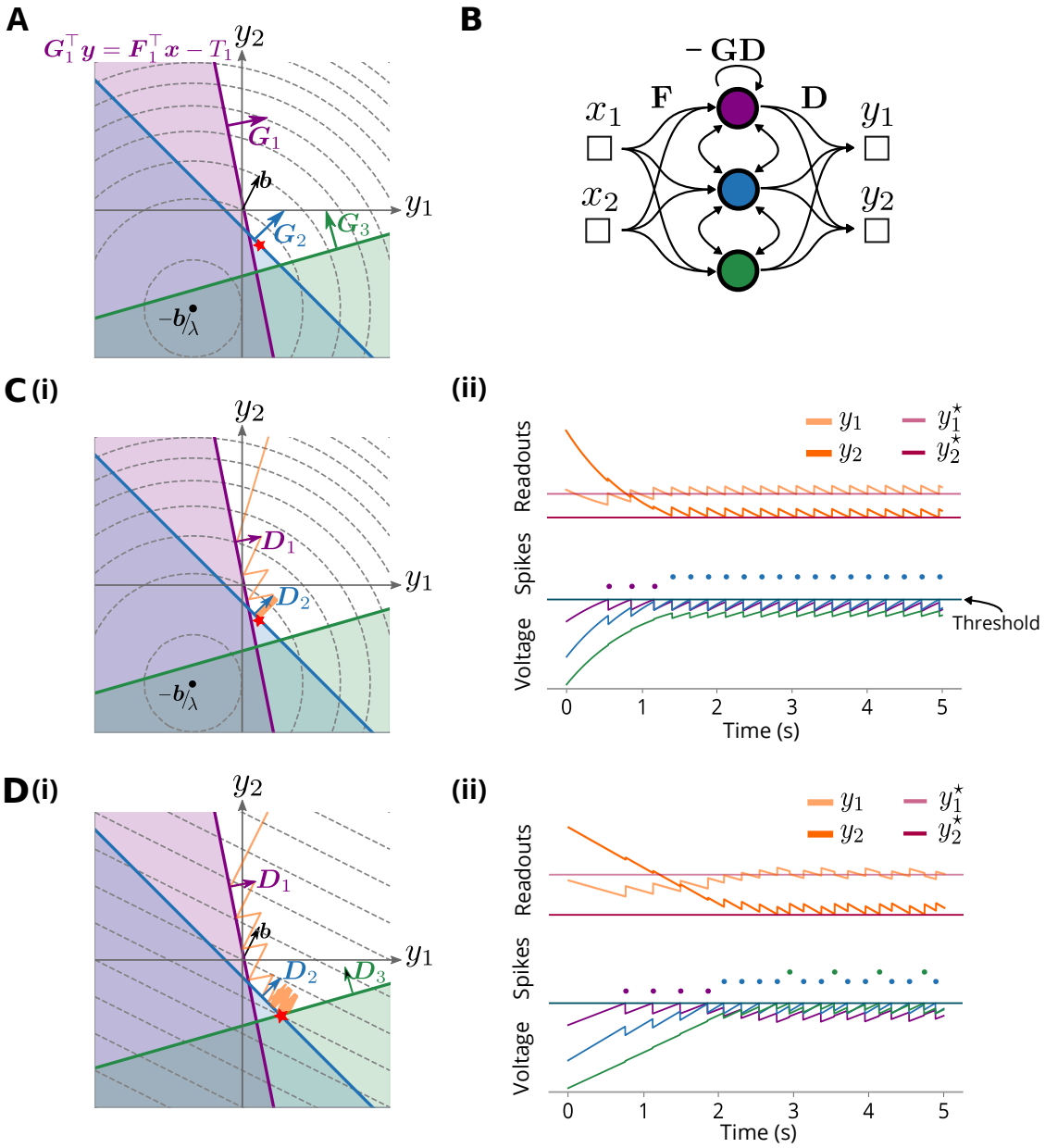

Figure 5.2: Convex optimization and SNNs. (A) Illustration of a quadratic program (QP). Dashed grey circles are the contours of the objective with unconstrained minimum at $-\frac{\mathbf{b}}{\lambda}$. Each neuron or inequality constraint divides the space into half-spaces with shaded regions being infeasible. The QP solution (red star) is attained within the feasible set (unshaded). (B) Configuration of a network of 3 neurons with connectivity matrices labelled. For inputs (x_1, x_2), the network produces the outputs (y_1, y_2) that can be readout linearly (C) Dynamics of a spiking network solving a QP. (i) The trajectory of the network's readout (orange) bounces back into the feasible set each time it hits a boundary. Gradually, the readout approaches the optimum point. (ii) Neural activity over time with corresponding readouts jumping about the QP solution. The corresponding spikes and voltages are shown below. (D) Same as (C) but the spiking network is now solving a linear program. The dynamics is now driven by $\mathbf{b}$ and does not leak to the unconstrained minimum.

5.2.3 Gradient descent under constraints

Now that the optimization problem is defined, we can ask how would an algorithm proceed to solve it. Several highly efficient algorithms exist to solve QPs (Boyd et al., 2004; Nocedal and Wright, 2006). Here, we start with one of the simplest: gradient descent. We can take the derivative of the objective function in problem 5.2 to find the descent direction and this yields in continuous time, the following differential equation

$$\dot{\mathbf{y}} = -\frac{\partial E}{\partial \mathbf{y}} = -\lambda \mathbf{y} - \mathbf{b}. \tag{5.3}$$

However, following this trajectory alone can drive the optimization variable outside the feasible set. To avoid this, we introduce a bouncing mechanism whereby each boundary reflects $\mathbf{y}$ into a direction $\mathbf{D}_i$ whenever $\mathbf{y}$ hits it [1]. By modelling each such bounce as a delta function, we then obtain the following optimization dynamics

$$\dot{\mathbf{y}} = -\lambda \mathbf{y} - \mathbf{b} + \mathbf{D}\mathbf{s}(t), \tag{5.4}$$

where $\mathbf{s}(t)$ is the N-dimensional vector of bouncing events, similar to the spike trains introduced above (equation 5.1), and $\mathbf{D} \in \mathbb{R}^{M \times N}$ is the matrix of bounce directions.

To achieve the correct solution to the optimization problem, the direction of each bounce is crucial. For now, we will simply assume that $\mathbf{y}$ bounces back orthogonal to the respective boundary. This means that bounce direction, $\mathbf{D}_i$ should be parallel to the normal vector of the boundary, $\mathbf{G}_i$ and into the feasible, or mathematically $\mathbf{D}_i \propto \mathbf{G}_i$ (with non-negative constant of proportionality). This condition is sufficient, although not necessary, to guarantee that the optimization variable approaches a minimum of the loss function within the feasible set. Eventually, $\mathbf{y}$ approximates the true solution to the QP with a discretization error, η, which depends on the size of the jumps as defined by the length of $\mathbf{D}_i$. We illustrate this in Fig. 5.2(C, D), where the orange trajectory shows the dynamics of $\mathbf{y}$ due to descent on the loss function, interspersed with bouncing events, whenever one of the boundaries is hit. Once close to the solution (red star), the dynamics jumps back and forth around the true solution. If we set the parameter $\lambda = 0$, the dynamics now moves $\mathbf{y}$ close to the solution of a linear program (LP) which lies at a vertex as seen in Fig. 5.2D (Boyd et al., 2004). In contrast to Fig. 5.2C, the optimization dynamics is now driven by a constant drift, $-\mathbf{b}$, rather than an exponential decay.

[1] Note that $\mathbf{D}_i$ refers to the i-th column of $\mathbf{D}$ while $\mathbf{F}_i$ and $\mathbf{G}_i$ are the i-th rows of $\mathbf{F}$ and $\mathbf{G}$, but expressed as column vectors.

5.2.4 From convex optimization to the voltage dynamics of LIF neurons

To link this optimization problem to SNNs, we first define the left-hand-side of the inequality constraints in optimization problem 5.2 as the voltages of the neurons, $\mathbf{V} = \mathbf{F}\mathbf{x} - \mathbf{G}\mathbf{y}$, and the parameters, $\mathbf{T}$, on the right-hand-side as their thresholds. The dimensionality of the network is then set by the number of such inequality constraints. Furthermore, we assume that the input $\mathbf{x}$, just as the optimization variable $\mathbf{y}$, are time-dependent variables. By taking the temporal derivative of the voltages, we then obtain

$$\dot{\mathbf{V}} = \mathbf{F}\dot{\mathbf{x}} - \mathbf{G}\dot{\mathbf{y}} \tag{5.5}$$

$$= \mathbf{F}\dot{\mathbf{x}} + \lambda\mathbf{G}\mathbf{y} + \mathbf{G}\mathbf{b} - \mathbf{G}\mathbf{D}\mathbf{s}(t) \tag{5.6}$$

$$= -\lambda\mathbf{V} + \mathbf{F}(\lambda\mathbf{x} + \dot{\mathbf{x}}) - \mathbf{G}\mathbf{D}\mathbf{s}(t) + \mathbf{G}\mathbf{b}. \tag{5.7}$$

Equation (5.6) follows from equation (5.5) by replacing $\dot{\mathbf{y}}$ by its dynamics (5.4), and equation (5.7) follows from (5.6) by using the definition of the voltage.

Interestingly, we have now recovered the voltage dynamics of a network of LIF neurons, similar to equation (5.1) we introduced earlier. The parameter λ, tied to the quadratic loss function in the optimization problem, determines the membrane leak time-constant, the matrix $\mathbf{F}$ has become the feed-forward weight matrix with inputs $\mathbf{c}(t) = \lambda\mathbf{x} + \dot{\mathbf{x}}$, the matrix $\mathbf{G}$, together with the matrix of bouncing directions, $\mathbf{D}$, has become the recurrent weight matrix, $\mathbf{\Omega} = -\mathbf{G}\mathbf{D}$, the bouncing events, $\mathbf{s}(t)$, are now the spike trains, and the bias $\mathbf{b}$ is an external current fed to the network via the weights $\mathbf{G}$. Since there is a direct correspondence from the voltage dynamics to the dynamics of the optimization variable, the resulting SNN thus implements the gradient optimization described above. It solves quadratic programs if the voltages of the neurons leak i.e. $\lambda > 0$ or linear programs in the absence of such leaks i.e. $\lambda = 0$, up to a discretization error, η.

If we define the instantaneous firing rates of neurons, $\mathbf{r}(t)$, as the filtered versions of their spike trains, i.e., $\dot{\mathbf{r}} = -\lambda\mathbf{r} + \mathbf{s}(t)$, then we recover our optimization variable as the instantaneous linear readout of a downstream layer since it satisfies $\mathbf{y} = \mathbf{D}\mathbf{r} - \frac{\mathbf{b}}{\lambda}$ (this can be checked simply by taking the time-derivative of $\mathbf{y}$, from which we recover equation 5.4). We illustrate the behavior of the network in Fig. 5.2(C and D) while solving a QP and LP problem, respectively; the network's instantaneous output ultimately provides a solution to the QP and LP.

5.2.5 Example networks

We now show that several previously proposed SNNs fit into this framework. Here, we list a few (in all cases setting the bias, $\mathbf{b} = 0$, for simplicity):

- **ReLu Layer.** If we assume independent neurons, we get $M = N$, and $\mathbf{G} = \mathbf{D}^\top = \mathbf{I}$. In this case, $\mathbf{y} = \mathbf{r}$ and we can integrate the voltage dynamics, equation 5.7, over time to get the following equation for the voltages $\mathbf{V} = \mathbf{F}\mathbf{x} - \mathbf{r} \leq \mathbf{T}$, which we can re-write as $\mathbf{r} \geq \mathbf{F}\mathbf{x} - \mathbf{T}$. Since the objective of the optimization problem is quadratic, we find that the SNN dynamics leads to the solution

$$\mathbf{r} = \max(\mathbf{F}\mathbf{x} - \mathbf{T}, \mathbf{0}) + \eta. \tag{5.8}$$

  Thus, the input-output function of the network is essentially a 'ReLu' function besides the additional term η that captures the error due to the jumps about the fixed point.

- **Spike Coding networks.** If we assume $\mathbf{G} = \mathbf{F}$, the integrated voltage equation then becomes $\mathbf{V} = \mathbf{F}\mathbf{x} - \mathbf{F}\mathbf{D}\mathbf{r} \leq \mathbf{T}$, which is the voltage definition of the 'spike coding' networks learnt in (Brendel et al., 2020). These networks have been shown to generate biologically realistic activity (asynchronous, irregular spiking, balance of excitation and inhibition).

- **Sparse Coding networks.** If we additionally assume $\mathbf{G} = \mathbf{F} = \mathbf{D}^\top$, the voltages become $\mathbf{V} = \mathbf{D}^\top\mathbf{x} - \mathbf{D}^\top\mathbf{D}\mathbf{r} \leq \mathbf{T}$, and we recover the voltage equation in previously proposed auto-encoder SNNs (Barrett et al., 2013; Boerlin et al., 2013). In fact, these SNNs find energy-efficient neural representations of their inputs, similar to the networks we discussed in Chapter 3, although in the latter case the representations were generated using firing rates.

  To clarify this link of energy efficiency between the QP that optimizes over firing rates in Chapter 3 and the optimization problem here, we resort to the latter's Lagrange dual formulation (Boyd et al., 2004). As we show in the derivations below, with the weight parameter choices made above (and using Lagrangian multipliers $\mathbf{r} \geq 0$), the 'argmin' solution to the resulting dual optimization problem becomes equivalent to the classical sparse coding (Olshausen and Field, 1996) firing rates solution, given as,

$$\mathbf{r}^\star = \underset{\mathbf{r} \geq \mathbf{0}}{\arg\min} ||\mathbf{x} - \mathbf{D}\mathbf{r}||^2 + \mathbf{r}^\top\mathbf{T}, \tag{5.9}$$

  We note that the thresholds, $\mathbf{T}$, now explicitly appear in the objective function and effectively implement a sparsity cost on the firing rates. Although this fea-

ture was implicitly captured when simulating previous SNNs, we here provide a more formal understanding of the sparsity effects of the spiking thresholds.

Derivation of the dual problem. With the above parameter choices and $\lambda = 1$, using Lagrange multipliers $\mathbf{r} \geq 0$, the Langragian, L, is obtained by augmenting the loss function in problem 5.2 with its constraints, i.e.,

$$L(\mathbf{y}, \mathbf{r}) = \frac{1}{2}\mathbf{y}^\top \mathbf{y} + \mathbf{r}^\top \Big(\mathbf{D}^\top(\mathbf{x} - \mathbf{y}) - \mathbf{T}\Big) \tag{5.10}$$

The Lagrange dual function is then defined as the minimum value of the Lagrangian over $\mathbf{y}$:

$$g(\mathbf{r}) = \inf_{\mathbf{y}} \Big(\frac{1}{2}\mathbf{y}^\top \mathbf{y} + \mathbf{r}^\top (\mathbf{D}^\top(\mathbf{x} - \mathbf{y}) - \mathbf{T})\Big) \tag{5.11}$$

At its minimum, we obtain $\mathbf{y} = \mathbf{D}\mathbf{r}$. Plugging this back into $g(\mathbf{r})$ yields the dual problem:

$$\begin{aligned} \underset{\mathbf{r}}{\text{Minimize}} \quad & \left(\frac{1}{2}\mathbf{r}^\top \mathbf{D}^\top \mathbf{D}\mathbf{r} - \mathbf{r}^\top \mathbf{D}^\top \mathbf{x} + \mathbf{r}^\top \mathbf{T}\right) \\ \text{subject to} \quad & \mathbf{r} \geq 0 \end{aligned} \tag{5.12}$$

which has the same 'argmin' solution as equation 5.9.

5.2.6 Configuration for input-driven networks

However, we can also free ourselves from strictly solving convex optimization problems (QPs and LPs), and consider more generally, what happens if we choose the bouncing directions $\mathbf{D}$ differently, i.e., not necessarily orthogonal to the boundaries.

For instance, we can simply choose $\mathbf{D}$ such that the optimization variable $\mathbf{y}$ remains within the feasible set. One way to enforce this is to have the bounce direction, $\mathbf{D}_i$ to make an angle of at most 90° to the normal vector, $\mathbf{G}_i$, for each boundary i, or mathematically, we need $\mathbf{G}_i^\top \mathbf{D}_i \geq 0$. This necessary condition means that the diagonal elements of the recurrent connectivity need to be negative, i.e., $\Omega_{ii} = -\mathbf{G}_i^\top \mathbf{D}_i \leq 0$, which simply corresponds to the requirement that a neuron's self-reset after it spikes (which we model through the diagonal entries of $\mathbf{\Omega}$) to be negative.

However, this is not a sufficient condition for $\mathbf{y}$ to remain in the feasible set after the jump. We illustrate this in a two-neurons schematic in Fig. 5.3A. From the initial position, $\mathbf{y}_a$, a jump along $\mathbf{D}_1$ keeps the optimization variable within the feasible set (unshaded region). But, the same jump is infeasible if the initial position

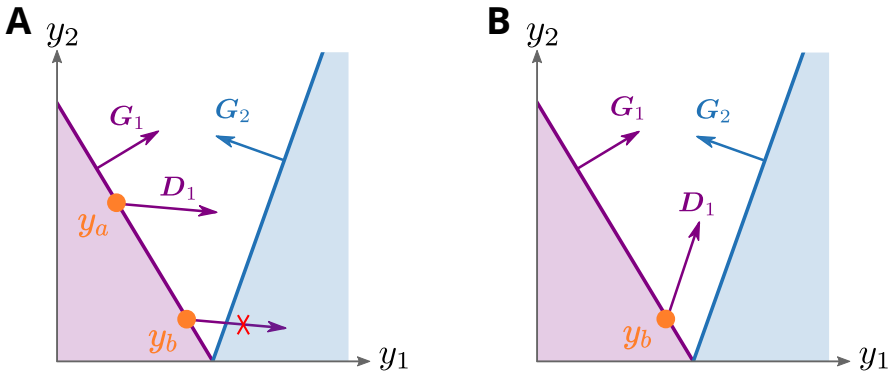

Figure 5.3: Conditions to keep the optimization variable, $\mathbf{y}$, within the feasible set (unshaded area) using a two-neurons schematic. (A) When the bounce direction, $\mathbf{D}_1$, of neuron 1 is less than 90° to the normal vector of its boundary, $\mathbf{G}_1$, this can prevent $\mathbf{y}$ from jumping into its infeasible region (pink), e.g. the jump from $\mathbf{y}_a$. But this is not sufficient as shown by the jump from $\mathbf{y}_b$. (B) A sufficient but not necessary condition is to keep $\mathbf{D}_1$ angled less than 90° to the normal vector of the other neuron's boundary, $\mathbf{G}_2$. Thus, the jump from $\mathbf{y}_b$ does not cross into the infeasible half-space defined by the second neuron (pale blue).

is $\mathbf{y}_b$. A solution would be to constrain the bounce direction for a given neuron to be angled at most 90° to the normal vectors of the boundaries of all other neurons. Thus, a sufficient (but not necessary) for each jump to remain in the feasible set is to set $\Omega_{ij} = -\mathbf{G}_i^\top \mathbf{D}_j \leq 0$ (see Fig. 5.3B).

Note that due to these conditions, the recurrent connections become all-inhibitory, in which case the excitation in the network comes from external inputs and hence, input-driven (Abbott et al., 2016). In all cases, the dynamics of the network will effectively wander along the boundary of the feasible set, and this boundary can therefore be thought of as the manifold on which the SNN dynamics evolves.

5.3 Understanding computations in SNN layers

5.3.1 Input-driven networks

Given the link between SNNs and convex optimization, we now consider the possible computations done by such networks. More specifically, we will study how the output, $\mathbf{y}$, read out from the network's activities, changes as a function of the input, $\mathbf{x}$. For simplicity, we will assume $\mathbf{b} = 0$. For now, we will consider input-driven networks where the optimization variable remains in the feasible set (see above). To guarantee that the SNNs approximate the solution of the underlying optimization problem, we choose $\mathbf{D}_i \propto \mathbf{G}_i$ and $G_{ij} \geq 0$, which together keep the

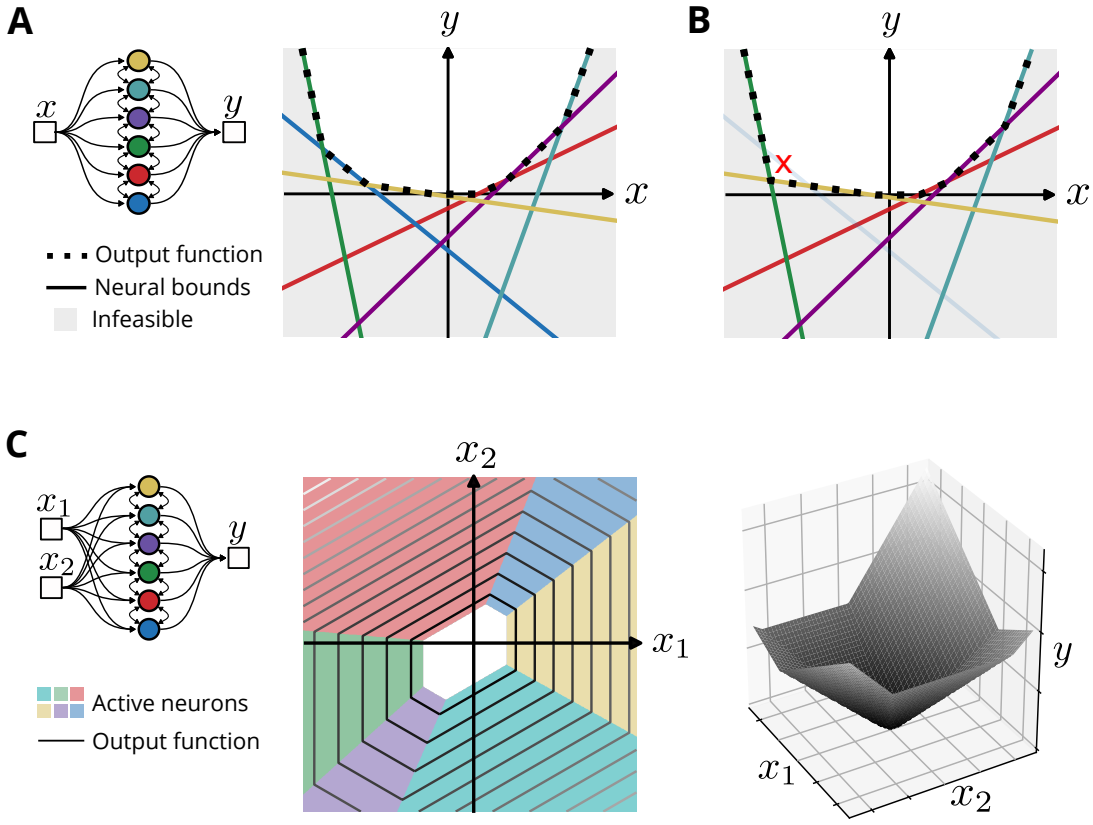

Figure 5.4: Interpreting spiking network computations. (A) Example network of six interconnected neurons performing one-dimensional transformations. The corresponding input-output function is illustrated on the right in (x, y)-space. Solid lines illustrate neural boundaries, and the dotted line corresponds to the resulting input-output functions (minus the discretization error η) generated by the network. (B) Changing or silencing a single neuron (red cross) does not substantially change the overall input-output function, thus showing the locality of the representation. (C) Same as in (A), but for a 2D input. Now each neuron's inequality boundary corresponds to a plane in 3D space. Different neurons are active in different parts of this space (colored patches), and locally determine the slope of the output function (as shown by the contours). White corresponds to having no neurons active. The output function is shown in 3D on the right.

recurrent connectivity all-inhibitory.

One-dimensional outputs

When the output of the network has dimensionality $M = 1$, we can write each neuron's voltage inequality as $\mathbf{F}_i^\top \mathbf{x} - G_i y \leq T_i$. Solving for y, we then obtain the set of inequalities $y \geq (\mathbf{F}_i^\top \mathbf{x} - T_i)/G_i$. Since the dynamics of the spiking network minimizes the loss $L = \lambda y^2/2$, the final solution will either be $y = 0$ which is the unconstrained minimum, or for a given $\mathbf{x}$ be bouncing off at a point on the boundary

of some active neuron j, given by $y = (\mathbf{F}_j^\top \mathbf{x} - T_j)/G_j$.

As $\mathbf{x}$ changes, if a different neuron now has its boundary overshadowing the other boundaries and is above the origin, then that neuron becomes active. The output function is thus determined locally by each neuron's boundary, until either $y = 0$ or another neuron becomes active instead. We can therefore write the input-output function as

$$y = f(\mathbf{x}) = \texttt{max}\Big(0, \frac{\mathbf{F}_1^\top \mathbf{x} - T_1}{G_1}, \frac{\mathbf{F}_2^\top \mathbf{x} - T_2}{G_2}, ...\Big) + \eta, \tag{5.13}$$

which is effectively a piecewise-linear function which partitions the space according to a set of linear equations (see Fig. 5.4A). Importantly, each neuron only acts within a limited local range, which endows the network with a certain inherent robustness to single neuron changes in parameters and even cell death, as this would only locally affect the input-output function (Fig. 5.4B). In this one-dimensional case, the network then behaves as a 'maxout' unit, which has previously been shown to be universal approximators (Goodfellow et al., 2013).

M-dimensional outputs

Conceptually, the extension to M-dimensional outputs is not difficult. A given neuron's spiking boundary is described by all points satisfying $\mathbf{F}_i^\top \mathbf{x} - \mathbf{G}_i^\top \mathbf{y} = T_i$, which can be viewed as a $(K+M-1)$-dimensional hyperplane in the $(K+M)$-dimensional space defined by $\mathbf{u} = (\mathbf{x}, \mathbf{y})$. As before, each neuron's hyper-plane divides the space into a feasible and infeasible region. The solution is constrained to lie either at $\mathbf{y} = 0$ or be bouncing off one (or more) of these hyperplanes. Consequently, the solution comes from a piecewise-linear, convex function of the input $\mathbf{x}$.

5.3.2 Moving beyond solving convex optimization problems

By considering different alignments of the bouncing directions and neuronal boundaries and not restricting the networks to solve QPs and LPs, we show here that this framework can also give geometric insights of more general network dynamics e.g. bi-stability or periodicity.

We consider in a first example a scenario where the SNN exhibits bi-stable dynamics due to a specific configuration of the bounce directions (Fig. 5.5A). With a small nudge, e.g. noise, that can change the initial state from either points A or B, the dynamics lead the network's output, $\mathbf{y}$, to systematically different equilibrium

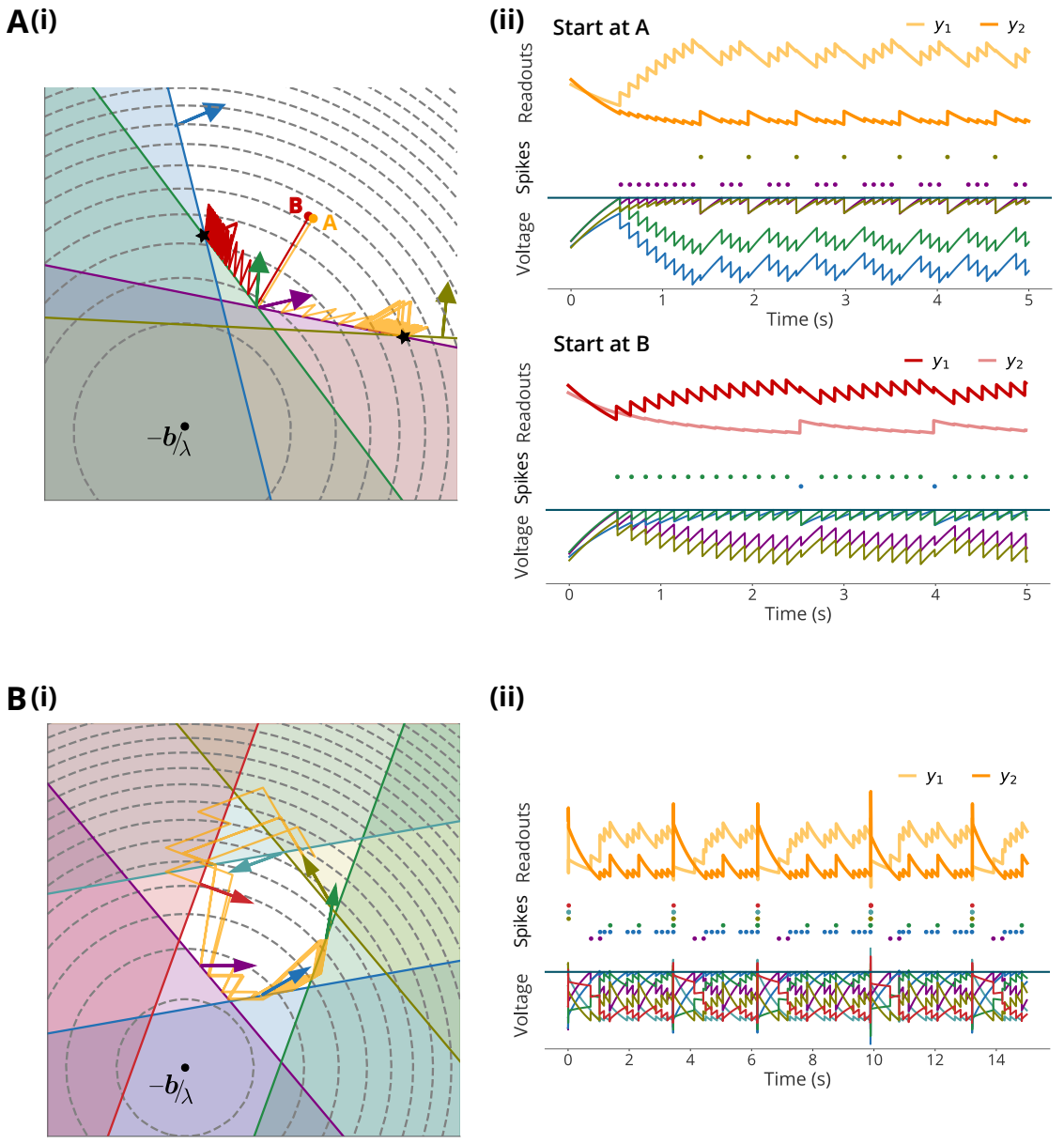

Figure 5.5: Other dynamics e.g. bi-stable or periodic dynamics are obtained under different configurations of the network parameters. These can be understood geometrically within the same framework. (A) Bi-stable network dynamics. (i) Starting from point A or B leads to different equilibrium points with the readout trajectory following different paths. (ii) Behaviour of the networks under different initial conditions. (B) Periodic network dynamics. (i) The readout trajectory, due to some excitation, follows a cyclical path when the feasible set is closed. The excitation is due to the green and yellow neurons allowing the solution to move outside the feasible set. (ii) Network behaviour that leads to this dynamics. The readouts follow a cyclical pattern over time.

points (black stars). The network's output is driven close to one of the two attractors about which it then jumps back and forth. We illustrate the behaviour of the network in Fig. 5.5A(ii).

In a general setting, the recurrent connectivity, $\Omega = -\mathbf{GD}$, needs not be all-inhibitory. We can thus design the connectivity such that excitation recurrent connections drive $\mathbf{y}$ outside of the feasible set. Such excitations can either decay quickly if the next neuron moves the solution back into the feasible set, or they may self-sustain if subsequently active neurons keep pushing the solution out of the feasible set. We show in Fig. 5.5B an example network that displays periodic dynamics due to a small amount of excitation. In this case, the feasible set is closed forming a polytope (Fig. 5.5B(i)) and the green and olive neurons push the solution outside the feasible set. As a result, the solution follows a cycle (orange trajectory) not reaching a fixed point. Periodic patterns in the network's readouts can also be seen in Fig. 5.5B(ii).

5.4 A geometric view on supervised learning in SNNs

The above considerations have given us a better understanding of the computations performed by a single layer of spiking neurons. An immediate question that arises next is whether the parameters of the network can be learnt for a desired computation. Although a full-fledged set of learning rules is beyond the scope of this thesis, we will show next that the geometric insights in this framework provide us with a new avenue to address the learning problems in a supervised setting with biological constraints. In particular, we will concern ourselves with input-driven networks where the input-output functions are clearly defined (see section 5.3.1).

5.4.1 Learning through basis functions

Let us first consider how the network parameters can be learnt in a classical sense. One simple approach is to consider the outputs of the neurons at a given layer to form a set of basis functions, which can then be combined linearly for arbitrary input-output transformations by training the output weights (LeCun et al., 2015; Eliasmith and Anderson, 2004). The same applies within our framework. By using random parameters $\mathbf{F}$, $\mathbf{G}$, $\mathbf{D}$, and $\mathbf{T}$, a rich set of (convex) basis functions as the M-dimensional output of the network can be generated. If the set is rich enough, these basis functions can then be combined to approximate arbitrary, non-convex

input-output functions. Geometrically, this approach corresponds to having a fixed set of N neural hyper-planes in an M-dimensional space (Fig. 5.4C), and requires a high dimensionality (in both the number of neurons and the readout space) to generate a rich enough basis-set. However, instead of following this standard route, we will investigate here the problem of directly adjusting the neural hyper-planes themselves in order to learn specific (convex) input-output functions.

5.4.2 Learning the neural hyper-planes

How can the neural hyperplanes (each hyperplane corresponding to an inequality constraint in the optimization problem) be adjusted to make the network's output, $\mathbf{y}$, match a desired target, $\tilde{\mathbf{y}}$? One approach would be to define a loss function over the global error, $\mathbf{e} = \mathbf{y} - \tilde{\mathbf{y}}$, and then minimize it e.g. using gradient descent to find the best set of parameters (Amos and Kolter, 2017; Agrawal et al., 2019; Gould et al., 2016). However, this approach yields highly non-local parameter updates and thus, it is not feasible for biological networks.

From the geometric interpretation of the problem, we note three facts. First, individual neurons do not have direct access to the global variable, $\mathbf{y}$, or to the error, $\mathbf{e} = \mathbf{y} - \tilde{\mathbf{y}}$, but rather get indirect information through a projection on their encoder weights, $\mathbf{G}_i$. Second, at spike-time, this projection satisfies $\mathbf{G}_i^\top \mathbf{y} = \mathbf{F}_i^\top \mathbf{x} - T_i$ which is the boundary equation (see Fig. 5.2). So, a neuron can get information about $\mathbf{y}$ only when there is a spike event. Third, the end result of learning should be that each boundary locally supports the target function from below, or becomes tangential to its epigraph[2] in $(\mathbf{x}, \mathbf{y})$-space, and that the output function is properly distributed across neurons (e.g. see Fig. 5.4A).

From these insights, we propose the following learning scheme. To prevent overshadowed neurons from ever participating, we allow neurons to drift their boundaries slowly to the epigraph by lowering their thresholds, $\dot{T}_j = -\lambda_T, \ \forall j \in 1, \ldots, N$, where λ_T determines the speed of the drift. This encourages neurons whose boundaries are far away from the target to eventually approach the epigraph from below, thereby compete with the other neurons, start spiking, and thus take part in the learning. At the same time, we let all neurons that emit spikes to adjust their boundaries and redistribute them along the epigraph. To do so, we take a simple approach and minimize the projected error on each neuron's encoding weights. For one data

[2]The epigraph of a function $f : \mathbb{R}^K \to \mathbb{R}$ is defined as $\mathbf{epi}\ f = \{(x,t) | x \in \mathbf{dom}\ f, f(x) \leq t\}$ (Boyd et al., 2004)

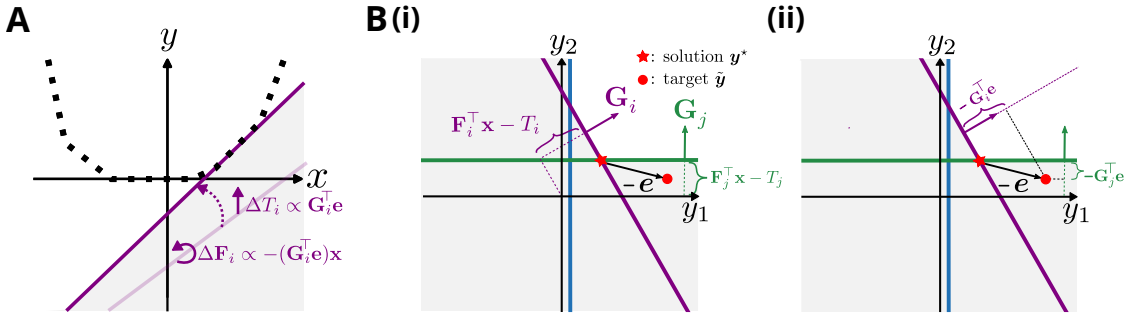

Figure 5.6: Supervised learning of spiking networks. (A) Illustrating the effect of learning the feed-forward weights, $\mathbf{F}$ and thresholds $\mathbf{T}$ in (x, y)-space. Learning T_i for neuron i translates its spiking boundary as an offset and learning $\mathbf{F}_i$ changes its slope. With competition across neurons, the violet neuron only learns to fit data points in its partition. (B) Illustrating the locality of the learning rules in $\mathbf{y}$-space (assuming unit vectors for the sake of illustration). (i) For a given input $\mathbf{x}$, the parameters $\mathbf{F}$ and $\mathbf{T}$ effectively determine by how much each spiking boundary is translated from the center of coordinates. (ii) At the solution (red star), two neurons are active and they individually try to minimize the projected error. Either a change $\Delta\mathbf{F}_i$ or $\Delta\mathbf{T}_i$ in (A) leads to a translation of the boundary in $\mathbf{y}$-space. According to the learning rules, the size of the translation will be $\mathbf{G}_i^\top \mathbf{e}$ which brings each spiking boundary closer to the target (red dot).

point, we therefore minimize the loss

$$L = \frac{1}{2}\sum_i^{N_{act}} ||\mathbf{G}_i^\top (\mathbf{y} - \tilde{\mathbf{y}})||^2 = \frac{1}{2}\sum_i^{N_{act}} ||\mathbf{F}_i^\top \mathbf{x} - T_i - \mathbf{G}_i^\top \tilde{\mathbf{y}}||^2 \tag{5.14}$$

where N_{act} is the subset of neurons that are active for a given input-output pair.

Here, we focus only on learning the feed-forward weights, $\mathbf{F}$, and thresholds, $\mathbf{T}$, which will be sufficient to guarantee some distribution along the epigraph; more general and efficient rules should also include changes in the recurrent connectivity $\mathbf{\Omega} = -\mathbf{GD}$ as well (e.g. along the same lines as in (Brendel et al., 2020)) but this is beyond the scope of this thesis. By computing the corresponding gradients, we get the following rules for $i \in N_{act}$:

$$\Delta T_i = -\alpha \frac{\partial L}{\partial T_i} = \alpha \left(\mathbf{F}_i^\top \mathbf{x} - T_i - \mathbf{G}_i^\top \tilde{\mathbf{y}}\right) = \alpha \mathbf{G}_i^\top \mathbf{e} \tag{5.15}$$

$$\Delta \mathbf{F}_i = -\alpha \frac{\partial L}{\partial \mathbf{F}_i} = -\alpha \left(\mathbf{F}_i^\top \mathbf{x} - T_i - \mathbf{G}_i^\top \tilde{\mathbf{y}}\right) \mathbf{x}^\top = -\alpha (\mathbf{G}_i^\top \mathbf{e})\mathbf{x}, \tag{5.16}$$

where α is the learning-rate. To ensure that the drift in thresholds does not dominate the learning, we set $\lambda_T \ll \alpha$. Additionally, to encourage a proper distribution of the code across neurons, we introduce a cost, μ, on spikes which further hyperpolarizes a neuron after it spikes, i.e., it lowers the reset after a spike to $R_i = T_i - \Omega_{ii} - \mu$. This

cost limits arbitrarily high activity of individual neurons and can be thought of as a type of regularization similar to what has previously been used in SNNs (Boerlin et al., 2013).

We illustrate the effects of these rules in Fig. 5.6. Whenever a neuron spikes, it shifts and rotates its spiking boundary by ΔT_i and $\Delta \mathbf{F}_i$, respectively, in (x, y)-space, thereby reaching a partition of the input-target $(x, \tilde{y})$ samples. Assuming that different neurons start out with different random parameters, by presenting different input-target pairs many times, neurons can thus jointly find a piecewise-linear fit of a target function, e.g. as in Fig. 5.4A. To get further intuition of these rules, we consider its effect in a two-dimensional output space, for a single input-target sample. The parameters $\mathbf{F}$ and $\mathbf{T}$, for a given input, $\mathbf{x}$, effectively determine the perpendicular distance of each boundary from the center of coordinates (Fig. 5.4B(i)). When the solution (red star) sits at a corner of two boundaries, the corresponding neurons spike and use their local information about the projected error to individually minimize it (Fig. 5.4B(ii)). This results in changes ΔT_i and $\Delta \mathbf{F}_i$ $(i \in N_{act})$ causing each active neuron to translate its boundary closer to the target (red dot).

5.4.3 Simple paraboloid example

To demonstrate the effect of the learning rule in a simple toy example, we trained an SNN with $N = 50$ neurons to reproduce a paraboloid in 3D-space defined as $\tilde{y} = 0.3(x_1^2 + x_2^2)$. We show its contour plot in Fig. 5.7C. During a learning epoch, the inputs were randomly sampled from one hundred equally spaced points in the range, $[-4, 4]$, for each x-dimension. In each trial, a sampled input-target pair $(\mathbf{x}, \tilde{y})$ was fed to the SNN for four seconds of simulation time (using the forward Euler method). To reduce the effect of spikes due to transients as the input changed across trials, we only started training one second after the onset of the trial. We ran the algorithm for 100 epochs with each epoch covering the whole input space. Finally, we turned off the teaching signal and ran the network with learnt parameters. Fig. 5.7B shows the contours of the network readout (averaged over the last few time bins) which is a piecewise linear fit to the paraboloid. By color coding the background of the contour plot based on which neurons spike, we see a more distributed but still distinct partitioning of the input space after learning (contrast Fig. 5.7A and B).

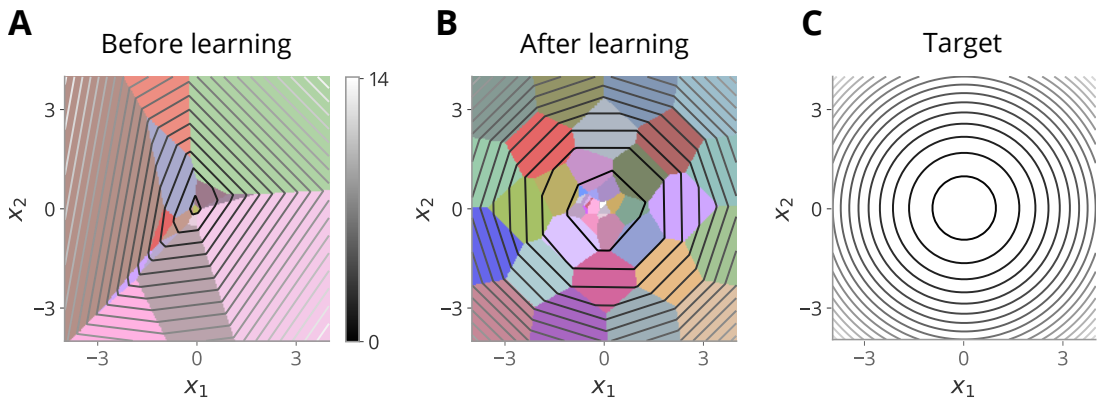

Figure 5.7: Learning a paraboloid for N=50 neurons. (A) The contour plot of network output before learning with background color coded according to which neurons are active. If no neuron participates, the partition is colored white. Each neuron locally determines the slope of the output function (as shown by the contours). (B) Contour plot after learning. It is a piecewise-linear approximation of the target function shown in (C). The more diverse colored patches illustrates the distribution of the code after learning. (C) Contour plot of the target function.

5.5 A larger spiking network

Finally, we demonstrate that the resulting networks exhibit many features of biological systems when they are scaled up. Our examples so far have focused on simple toy scenarios with only a one-dimensional output ($M = 1$), which leaves mostly one neuron active at a time. As a result, the neurons fire regularly (see Fig. 5.2C for an illustration) which is biologically unrealistic (Denève and Machens, 2016). But, this is no longer true when the readout is higher-dimensional. For example, in a two-dimensional output setting ($M = 2$), we can already find solutions that sit at corner points where boundaries intersect, in which case two neurons are active simultaneously (as in Fig. 5.2D). For higher dimensions, this quickly leads to more realistic and complex spiking behavior.

We demonstrate this observation with a network trained to classify a vector of three input pixels ($K = 3$). The inputs consist of three dimensional vectors with each element being either a zero or one, i.e. $\mathbf{x} = [x_1, x_2, x_3]$ with $x_i \in \{0, 1\}$. This creates a set of seven stimulus possibilities and the zero input ($\mathbf{x} = 0$) (Fig. 5.8A(i)). Then, for classification of these inputs, we create a 7-dimensional readout ($M = 7$). Whenever one of the seven possible pixel combinations is detected, the corresponding dimension of the readout should jump up from a baseline level. In the absence of any inputs, all dimensions of the readout are equal and set at that baseline level (see Fig. 5.8A(ii) as an example). Throughout all simulations, we added small amounts

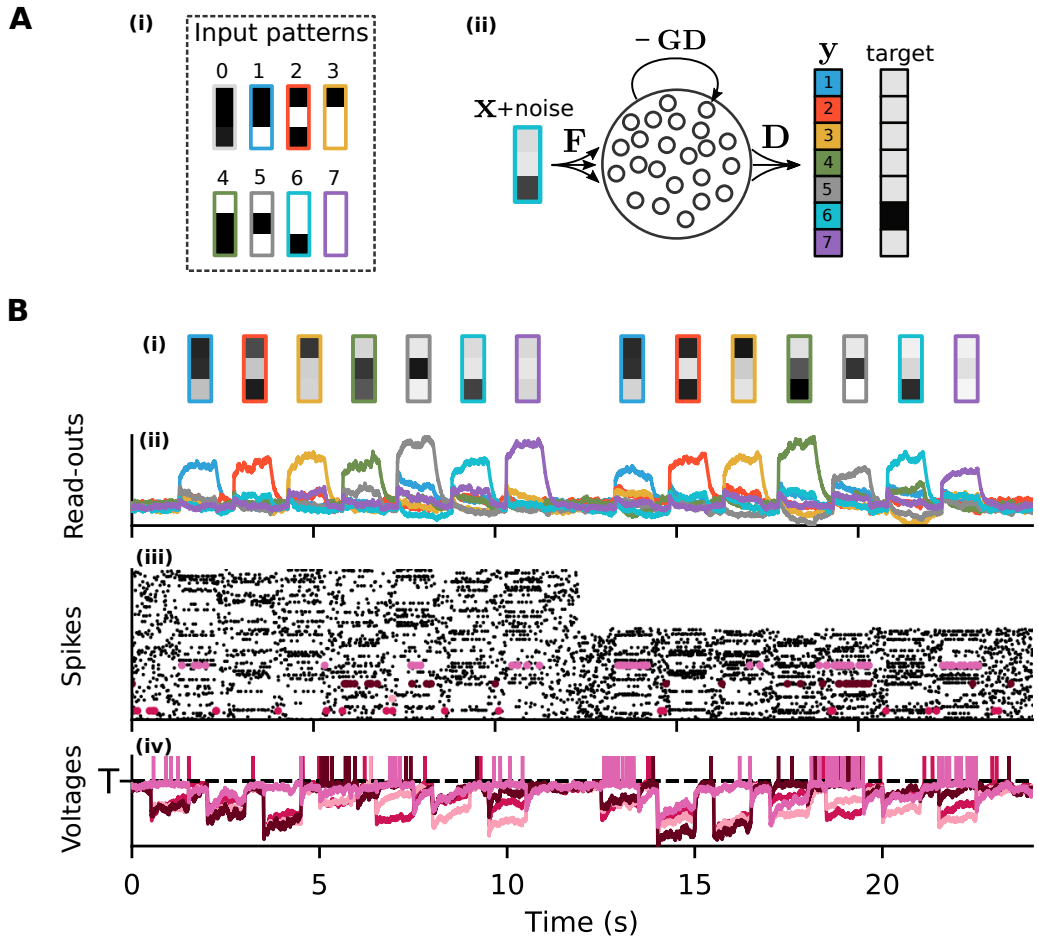

Figure 5.8: Spiking behavior in larger networks, illustrated with a simple classification network. (A) (i) Set of 3D stimulus inputs to the network corresponding to combinations of pixels of zeros and ones. (ii) Example classification by the network. In a trained network, noise was further added on the stimulus inputs (hence, the grey shade). The network classifies the input as a jump from the baseline in one of its 7-dimensional readout. (B) Network activity after training as a function of different input combinations. (i) Stimulus inputs presented to the network. (ii) Corresponding network readouts, with colors corresponding to the possible input combinations. The network correctly classifies the inputs despite the noise. (iii) Asynchronous and irregular underlying spike patterns. Halfway through the simulation, 40% of the population is knocked out, but the computations are preserved. (iv) Voltages of three neurons highlighted in (iii). Spikes are illustrated by vertical lines. Network parameters: $N = 300$, $\lambda = 2$, $\mathbf{D} = 0.1\mathbf{G}^\top$, $\mu = 0.1$, $\sigma_\mathrm{V} = 0.1$. Learning parameters: $\alpha = 0.1$, $\lambda_T = 0.001$, both decaying across epochs according to $\exp(-0.001 n_\mathrm{epoch})$, for 750 epochs. Perturbation noise parameters: $\sigma_\mathrm{stim} = 0.1$, $\sigma_\mathrm{OU} = 0.1$, $\lambda_\mathrm{OU} = 10$.

of Gaussian noise to the voltages. The network is then trained using the above rules to solve this simple classification task.

After training, we test the robustness of the resulting network by adding several perturbations not present during learning. First, each input pixel was corrupted by a random offset drawn from a Gaussian distribution on each trial (leading to different shades of grey in Fig. 5.8B(i)), as well as a drifting noise process over time (simulated by an Ornstein-Uhlenbeck (OU) process). Second, we silenced 40% of the neural population halfway through the simulation.

The resulting network after training, operates in a biologically realistic regime with irregular spiking patterns, robustness to perturbations, and low firing rates (see Fig. 5.8B). The irregularity of spiking stems largely from the complex spiking dynamics around the idealized solution point, rather than the voltage noise alone. Also, the network computation is robust to cell death; the network is still able to classify the input patterns despite 40% of the neurons knocked-out. This robustness emerges due to the redundancy in the neural representations since eliminating neurons, and thereby their boundaries, does not substantially change the feasible set (see Fig. 5.4B for an illustration and also (Barrett et al., 2016)). Finally, whereas eliminating neurons increases firing rates in the remaining neurons to compensate, adding neurons decreases population firing rates, and each neuron's activity can thereby be made arbitrarily sparse. In our published work (Mancoo et al., 2020), we also show that the network operates in a regime of balanced excitation and inhibition and is further robust to synaptic delays, a feature that has not been captured in previous work on similar networks (Boerlin et al., 2013; Barrett et al., 2016).

5.6 Discussion

In this chapter, we used insights from convex optimization theory to develop a new framework to understand SNNs. We showed how a broad class of SNNs fundamentally solve QPs and LPs, expanding on what had previously been shown (Barrett et al., 2013; Moreno-Bote and Drugowitsch, 2015; Chou et al., 2018), but with the solution obtained as an instantaneous readout (i.e. not requiring temporal integration). As a result, we gained key geometric insights on the dynamics as well as computations performed by SNNs. We showed that when the network is inhibition-dominated (or input-driven), it can generate convex input-output functions, or convex sets defined by their epigraphs. Thus, we can now understand SNNs as computing arbitrary, convex transformations of their inputs, according to

the connectivity of the neurons.

It may also be possible to think of SNNs and the brain as fundamentally solving convex optimization problems. Since many optimization problems can in fact, be relaxed to convex ones, this may not be such a crazy thought. In the framework here, the epigraph of the input-output function was obtained through a linear transformation of the neuronal activities (i.e. the output was obtained as a linear readout). Consequently, the epigraph of the input to neuronal activities function should also be convex since it is the 'inverse image' of the input-output epigraph under a linear transformation (Boyd et al., 2004). It may then be the case that the neural manifold would display some additional non-linearity to preserve the convexity property underlying the network computations. It would be interesting to see in future work if signatures of such computations can be found in neural data.

In this work, we primarily focused on input-output functions by restricting the SNNs to the input-driven (inhibition-dominated) regime. This was achieved by constraining the connectivity to be all-inhibitory (i.e. $-\mathbf{G}_i^\top \mathbf{D}_j \leq 0$), which we believe is realistic from a functional point of view, since inhibition is thought to dominate during awake behaviour (Haider et al., 2013). However, we also showed that this constraint can be relaxed and the geometric picture of our framework is still useful to understand the resulting network dynamics. In particular, we showed that sustained network activity can arise when each neuron violates in succession some other neuron's inequality constraint. Perhaps, this framework can be extended to understand geometrically the role of recurrent excitation in self-driven network regimes. Finally, we point out that we restricted our analysis here to the leaky-integrate-and-fire neural network model as it is one of the most commonly used models. Yet, it may be possible that the dynamics of other spiking neural network models can similarly, be understood through an optimization lens, where their dynamics would be minimizing some objective function. We leave this as an avenue for future research work.

Chapter 6

Conclusions and perspectives

With major ongoing advances in recording techniques, the neural data that we are collecting are becoming increasingly high-dimensional and complex. If we aim to elucidate the mechanisms of the brain through such data, the latter must first be understood. A promising route to this end has been through dimensionality reduction. This thesis was meant to contribute to the quest of understanding neural data, via the dimensionality reduction pathway. We summarize our findings below.

6.1 Summary of the thesis

While dimensionality reduction has been very useful in understanding some aspects of neural data (Cunningham and Yu, 2014), interpreting the results, thereof, may not always be straightforward. In Chapter 2, we analysed some examples datasets using some common linear dimensionality reduction methods. We showed that when methods such as principal component analysis (PCA) were applied, they extracted low-dimensional structures in the data that we could interpret with regards to the stimulus or task of the experiment. At the same time, they also displayed many additional components to capture the remaining variance in the data. Interestingly, we observed that several of these extra components resembled higher-order functions, e.g. polynomials of increasing order, of some other component, or 'higher-order' components (HOCs). While these HOCs persisted across datasets, there is still no clear explanation of what they mean. This thesis aimed to shed some light on these HOCs.

In Chapter 3, we argued that these HOCs were largely a reflection of the characteristic non-linear shape that the neural manifold takes under a very simple non-linearity: individual neuronal activity, as measured in its firing rate, is non-negative.

Our argument rested on two key assumptions: first, we assumed that readouts of population activity should be linear and low-dimensional and second, that population activity should be limited for reasons of energy efficiency. We explained geometrically how the neural manifold is then bound to be bent to preserve a suitable dynamic range for coding. We explored, using numerical simulations, the implications of this non-linearity on the results of linear methods such as PCA. We showed that while the correct latent variables could be extracted by means of a linear decoder, PCA had difficulty to do the same. PCA sometimes extracted the correct latent variables, but often displayed a tail of HOCs to compensate for the curvature in the manifold. Hence, these HOCs in the simulated data were functionally irrelevant as they did not represent any of the inputs, nor any underlying computations. Had we not known the ground truth, then these HOCs could mislead us to overestimate the dimensionality of the data and incorrectly assign some function to them.

To address this issue, we proposed in Chapter 4, a set of dimensionality reduction methods that incorporate the above coding constraints. These methods retain the assumption that the latent variables should be estimated as a linear mapping of the observed population activity, but enforce the coding constraints when predicting the neural activity given the latent variables. We showed that when these methods were applied to simulated data, they could find better estimates of the true (intrinsic) dimensionality of the data, compared to PCA. We further validated our methods on an example experimental dataset and showed that they could find a more succinct description by explaining more variance in the data with fewer latent variables, than PCA.

However, the neural manifold could exhibit additional non-linearities, besides the non-negativity constraints. We hypothesized in this thesis that the neural manifold can display some additional non-linearity that reflects the underlying computations done by the network. But to be able to understand the implications of such computations on the shape of the neural manifold and on data analysis, one must first understand the nature of these computations. In Chapter 5, we complemented this thesis by investigating the computations done by spiking neural networks (SNNs). We developed a new framework inspired from convex optimisation theory to show that a broad class of SNNs can fundamentally compute convex input-output functions, where the outputs are obtained through a linear mapping of the neural activities. We explained geometrically how such non-linear transformations occur according to the connectivity structure of the network. We further, showed that the

resulting networks display many features observed in the cortex such as irregular and asynchronous spike trains and robustness to perturbations.

6.2 Perspectives and future work

Finally, we conclude this thesis by putting in perspectives some of the keys findings of this work. We highlight their implications to our future endeavours of understanding high-dimensional neural data. We also add some criticisms to our approaches, and discuss what is still missing and what should be done in future work.

Functionally irrelevant higher-order components

Even today, most experiments employ highly-simplified stimuli or task structures to probe specific mechanisms of the brain. However, the brain arguably, has evolved to solve more complex tasks and thus, we might expect in the future, that experimental designs will become more sophisticated to explore in more depth the intricacies of brain functions.

However, despite the simplicity of current experimental paradigms, interpreting the neural data can still be fraught with difficulties. As we saw in Chapter 2, although the stimuli and behavioural task (if any) were quite simple, analysing the population data yielded patterns, in particular, higher-order components (HOCs), that are not immediately interpretable. Thus, before moving to more sophisticated experiments where the number of these HOCs may become more significant, it would be critical that they are first elucidated so as to understand the resulting neural data.

In this thesis, the key prediction of our theory (Chapter 3) is that these HOCs are functionally irrelevant and thus, ignoring them when interpreting neural data would not be harmful. For example, going back to the lateral intraparietal (LIP) data (Kiani and Shadlen, 2009) that we analysed in Chapter 2, we found that the activity of one of the HOCs was correlated with the difficulty associated with the stimulus. Since this stimulus difficulty information could be read out linearly (i.e. could be retrieved easily), one may have expected the neural circuit that computes confidence associated with a choice to use that information. However, we found surprisingly, that this HOC did not bear on the monkey's confidence. One possibility may then be that the correlation between stimulus difficulty and the HOC was, in fact, coincidental: according to our theory, the HOC simply appeared to compensate

for the curvature of the manifold and thus, does not reflect any underlying function of the system.

Non-linear neural manifold despite linear readouts

One assumption we made throughout this thesis was that the important underlying signals of neural activity should be obtained through a linear mapping — a transformation also present in all linear dimensionality reduction methods. We motivated this assumption by the numerous successes of these methods in finding important structures in neural population data that are experimentally relevant (e.g. as reviewed in (Cunningham and Yu, 2014; Keemink and Machens, 2019)), which suggest that the brain might actually, be implementing such a mechanism to decode relevant information.

An important finding we bring forth in this work is that, even if the true underlying low-dimensional variables, or latent variables, of the manifold can be obtained through a linear mapping, the neural manifold can still be non-linear. We explained that the non-linearity can emerge due to non-negativity constraints on individual neuronal activities when the neural code faces energetic constraints. Importantly though, a clear distinction should be made between describing the manifold using its true (intrinsic) underlying variables and describing it using the axes that span the space in which it is embedded. Linear methods such as PCA, even if they incorporate a linear decoder, can only describe the manifold in terms of its embedding space. However, this description, as we highlighted in this thesis, may not correspond to the underlying signals and thus, care should be taken when interpreting the results.

Estimating latent variables with biology-inspired constraints

If the brain effectively operates and computes on some lower dimensional set of latent variables, then extracting and interpreting them is crucial to enhance our understanding of brain functions. However, since these variables are not directly observed, estimating them ultimately relies on the assumptions or constraints in our methods when analysing neural data. Thus, any interpretations of the results, in particular when extrapolating to give functional meaning to the estimated latent variables, should be done with care (e.g. we should distinguish components that appear in neural data due to the curvature of the manifold from latent variables that reflect input signals or underlying computations).

In Chapter 4, we proposed dimensionality reduction methods that similar to PCA, estimate the latent variables through a linear mapping of the observed neural activity, but further incorporate some biology-inspired constraints when predicting the data. One advantage of constructing analytical methods in this way is that we can hypothesize on the mechanistic principles of the brain (e.g. linear readouts and energy efficiency in coding, in our case) and then test these hypotheses when applying the methods to neural data. Thus, not only will we find an estimate of the latent variables, but also provide some understanding of how these internal representations emerge.

However, a main criticism of our approach is that it remains unclear whether the estimated latent variables correspond to the 'true' latent variables (even though our methods did better than PCA, thus giving us some guarantee that we are on the right track). Of course, we can never be sure of what the true latent variables are, but we can still check how our methods fare against what may be possible. To this end, modern machine-learning techniques, e.g. deep autoencoders (Hinton and Salakhutdinov, 2006), or recurrent neural networks (Pandarinath et al., 2018), could be useful. Even if, these methods only approximate the function that yields the latent variables without necessarily shedding light on the biological or functional purpose of this transformation, they can hopefully extract the smallest set of estimated latent variables that would explain the data. By comparing these predictions with the results of our methods, this will then inform us if our methods missed on some keys aspects of the data. Thereupon, we can refine the assumptions or constraints in our methods if needed. We point out that such a comparison needs to be done in future work in order to put into perspective the importance of incorporating coding constraints in our dimensionality reduction methods.

Additional intrinsic non-linearities that shape the neural manifold

Since the HOCs were observed across several datasets, they likely emerged according to some inherent non-linearity in the system. This may arise, e.g. from the biophysical properties of individual neurons. In this thesis, we posited that a likely candidate leading to the characteristic shape of the neural manifold is the non-negativity constraint on individual neuronal activity. We showed that indeed, this constraint can produce a curved neural manifold, that would yield principal components resembling HOCs (Chapter 3).

However, there may be other intrinsic non-linearities in the system. For example,

synaptic connections between individual neurons or the integration of synaptic inputs in the dendritic tree of a neuron can be characterised by diverse non-linearities (Markram, 2003; Poirazi et al., 2003). These non-linearities could also have a functional purpose (e.g. for computations). But overall, they can contribute to an overall non-linear neural manifold, and consequently would affect of results of analytical methods, especially linear dimensionality reduction methods. It would be interesting in future work to investigate if, similarly to the non-negativity constraints, whether such intrinsic non-linearities in the system contribute to a shape of the manifold that can be characterised geometrically and see if they can also be incorporated in a meaningful way in our dimensionality reduction methods.

Bibliography

Abbott, Larry F, Brian DePasquale, and Raoul-Martin Memmesheimer (2016). "Building functional networks of spiking model neurons". In: *Nature neuroscience* 19.3, pp. 350–355.

Agrawal, Akshay et al. (2019). "Differentiable Convex Optimization Layers". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 9562–9574.

Ahrens, Misha B et al. (2012). "Brain-wide neuronal dynamics during motor adaptation in zebrafish". In: *Nature* 485.7399, pp. 471–477.

Amos, Brandon and J. Zico Kolter (Aug. 2017). "OptNet: differentiable optimization as a layer in neural networks". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, pp. 136–145.

Atick, Joseph J and A Norman Redlich (1990). "Towards a theory of early visual processing". In: *Neural computation* 2.3, pp. 308–320.

Attwell, David and Simon B Laughlin (2001). "An energy budget for signaling in the grey matter of the brain". In: *Journal of Cerebral Blood Flow & Metabolism* 21.10, pp. 1133–1145.

Barak, Omri, Misha Tsodyks, and Ranulfo Romo (2010). "Neuronal population coding of parametric working memory". In: *Journal of Neuroscience* 30.28, pp. 9424–9430.

Barlow, HB (1969). "Trigger features, adaptation and economy of impulses". In: *Information Processing in the Nervous System*. Springer, pp. 209–230.

Barlow, Horace B (1953). "Summation and inhibition in the frog's retina". In: *The Journal of physiology* 119.1, p. 69.

Barrett, David G, Sophie Denève, and Christian K Machens (2013). "Firing rate predictions in optimal balanced networks". In: *Advances in Neural Information Processing Systems*, pp. 1538–1546.

Barrett, David GT, Sophie Deneve, and Christian K Machens (2016). "Optimal compensation for neuron loss". In: *Elife* 5, e12454.

Bathellier, Brice et al. (2008). "Dynamic ensemble odor coding in the mammalian olfactory bulb: sensory information at different timescales". In: *Neuron* 57.4, pp. 586–598.

Bengio, Yoshua, Ian Goodfellow, and Aaron Courville (2017). *Deep learning*. Vol. 1. MIT press Massachusetts, USA:

Bishop, Christopher M (2006). *Pattern recognition and machine learning*. springer.

Boerlin, Martin, Christian K. Machens, and Sophie Denève (Nov. 2013). "Predictive Coding of Dynamical Variables in Balanced Spiking Networks". en. In: *PLoS Computational Biology* 9.11. Ed. by Olaf Sporns, e1003258. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1003258.

Bohte, Sander M., Joost N. Kok, and Han La Poutré (2002). "Error-backpropagation in temporally encoded networks of spiking neurons". en. In: *Neurocomputing* 48.1, pp. 17–37. ISSN: 0925-2312. DOI: 10.1016/S0925-2312(01)00658-0.

Boots, Byron and Geoff Gordon (2012). "Two-manifold problems with applications to nonlinear system identification". In: *arXiv preprint arXiv:1206.4648*.

Boyd, Stephen, Stephen P Boyd, and Lieven Vandenberghe (2004). *Convex optimization*. Cambridge university press.

Brendel, Wieland, Ranulfo Romo, and Christian K Machens (2011). "Demixed principal component analysis". In: *Advances in neural information processing systems*, pp. 2654–2662.

Brendel, Wieland et al. (Mar. 2020). "Learning to represent signals spike by spike". en. In: *PLOS Computational Biology* 16.3. Publisher: Public Library of Science, e1007692. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1007692.

Briggman, Kevin L, Henry DI Abarbanel, and William B Kristan (2005). "Optical imaging of neuronal populations during decision-making". In: *Science* 307.5711, pp. 896–901.

Britten, Kenneth H et al. (1992). "The analysis of visual motion: a comparison of neuronal and psychophysical performance". In: *Journal of Neuroscience* 12.12, pp. 4745–4765.

Brody, Carlos D et al. (2003). "Timing and neural encoding of somatosensory parametric working memory in macaque prefrontal cortex". In: *Cerebral cortex* 13.11, pp. 1196–1207.

Brunel, Nicolas (May 2000). "Dynamics of Sparsely Connected Networks of Excitatory and Inhibitory Spiking Neurons". en. In: *Journal of Computational Neuroscience* 8.3, pp. 183–208. ISSN: 1573-6873. DOI: 10.1023/A:1008925309027.

Cajal, S Ramon y (1906). *Nobel lecture—The structure and connexions of neurons*. Nobelprize.org.

Calaim, Nuno et al. (2020). "Robust coding with spiking networks: a geometric perspective". In: *bioRxiv*, p. 2020.06.15.148338. DOI: 10.1101/2020.06.15.148338.

Camastra, Francesco (2003). "Data dimensionality estimation methods: a survey". In: *Pattern recognition* 36.12, pp. 2945–2954.

Chichilnisky, EJ (2001). "A simple white noise analysis of neuronal light responses". In: *Network: Computation in Neural Systems* 12.2, pp. 199–213.

Chou, Chi-Ning, Kai-Min Chung, and Chi-Jen Lu (Mar. 2018). "On the Algorithmic Power of Spiking Neural Networks". In: *arXiv:1803.10375 [cs]*.

Churchland, Mark M et al. (2012). "Neural population dynamics during reaching". In: *Nature* 487.7405, pp. 51–56.

Clement, Elizabeth A et al. (2008). "Cyclic and sleep-like spontaneous alternations of brain state under urethane anaesthesia". In: *PloS one* 3.4, e2004.

Colby, Carol L and Michael E Goldberg (1999). "Space and attention in parietal cortex". In: *Annual review of neuroscience* 22.1, pp. 319–349.

Cowley, Benjamin et al. (2017). "Distance covariance analysis". In: *Artificial Intelligence and Statistics*, pp. 242–251.

Cunningham, John P and M Yu Byron (2014). "Dimensionality reduction for large-scale neural recordings". In: *Nature neuroscience* 17.11, pp. 1500–1509.

Denève, Sophie and Christian K. Machens (Mar. 2016). "Efficient codes and balanced networks". en. In: *Nature Neuroscience* 19.3, pp. 375–382. ISSN: 1546-1726. DOI: 10.1038/nn.4243.

Desimone, Robert et al. (1984). "Stimulus-selective properties of inferior temporal neurons in the macaque". In: *Journal of Neuroscience* 4.8, pp. 2051–2062.

Dontchev, Asen L and R Tyrrell Rockafellar (2009). *Implicit functions and solution mappings*. Vol. 543. Springer.

Eichenbaum, Howard (2018). "Barlow versus Hebb: When is it time to abandon the notion of feature detectors and adopt the cell assembly as the unit of cognition?" In: *Neuroscience letters* 680, pp. 88–93.

Eliasmith, Chris and Charles H. Anderson (Aug. 2004). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. en. MIT Press. ISBN: 978-0-262-55060-4.

Gallego, Juan A et al. (2018). "Cortical population activity within a preserved neural manifold underlies multiple motor behaviors". In: *Nature communications* 9.1, pp. 1–13.

Gao, Peiran et al. (2017). "A theory of multineuronal dimensionality, dynamics and measurement". In: *BioRxiv*, p. 214262.

Georgopoulos, Apostolos P et al. (1982). "On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex". In: *Journal of Neuroscience* 2.11, pp. 1527–1537.

Gnadt, James W and Richard A Andersen (1988). "Memory related motor planning activity in posterior parietal cortex of macaque". In: *Experimental brain research* 70.1, pp. 216–220.

Gold, Joshua I and Michael N Shadlen (2007). "The neural basis of decision making". In: *Annual review of neuroscience* 30.

Goldfarb, Donald and Ashok Idnani (1983). "A numerically stable dual method for solving strictly convex quadratic programs". In: *Mathematical programming* 27.1, pp. 1–33.

Goodfellow, Ian J. et al. (2013). "Maxout Networks". In: *arXiv:1302.4389 [cs, stat]*.

Gould, Stephen et al. (2016). "On differentiating parameterized argmin and argmax problems with application to bi-level optimization". In: *arXiv:1607.05447*.

Gütig, Robert (2016). "Spiking neurons can discover predictive features by aggregate-label learning". en. In: *Science* 351.6277. Publisher: American Association for the Advancement of Science Section: Research Article. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aab4113.

Gütig, Robert and Haim Sompolinsky (2006). "The tempotron: a neuron that learns spike timing–based decisions". en. In: *Nature Neuroscience* 9.3. Number: 3 Publisher: Nature Publishing Group, pp. 420–428. ISSN: 1546-1726. DOI: 10.1038/nn1643.

Haider, Bilal, Michael Häusser, and Matteo Carandini (2013). "Inhibition dominates sensory responses in the awake cortex". en. In: *Nature* 493.7430, pp. 97–100. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature11665.

Hara, Kazuyuki, Daisuke Saito, and Hayaru Shouno (2015). "Analysis of function of rectified linear unit used in deep learning". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8.

Hinton, Geoffrey E and Ruslan R Salakhutdinov (2006). "Reducing the dimensionality of data with neural networks". In: *science* 313.5786, pp. 504–507.

Hopfield, J. J. (July 1995). "Pattern recognition computation using action potential timing for stimulus representation". en. In: *Nature* 376.6535, pp. 33–36. ISSN: 1476-4687. DOI: 10.1038/376033a0.

Hotelling, Harold (1992). "Relations between two sets of variates". In: *Breakthroughs in statistics*. Springer, pp. 162–190.

Hu, Tao, Alexander Genkin, and Dmitri B Chklovskii (2012). "A network of spiking neurons for computing sparse representations in an energy-efficient way". In: *Neural computation* 24.11, pp. 2852–2872.

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

Hubel, David H et al. (1957). "Tungsten microelectrode for recording from single units". In: *Science* 125.3247, pp. 549–550.

Huh, Dongsung and Terrence J Sejnowski (2018). "Gradient Descent for Spiking Neural Networks". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., pp. 1433–1443.

Humphries, Mark D (2020). "Strong and weak principles of neural dimension reduction". In: *arXiv preprint arXiv:2011.08088*.

Karhunen, Juha and Jyrki Joutsensalo (1994). "Representation and separation of signals using nonlinear PCA type learning". In: *Neural networks* 7.1, pp. 113–127.

Keemink, Sander W and Christian K Machens (2019). "Decoding and encoding (de) mixed population responses". In: *Current Opinion in Neurobiology* 58, pp. 112–121.

Kiani, Roozbeh and Michael N Shadlen (2009). "Representation of confidence associated with a decision by neurons in the parietal cortex". In: *science* 324.5928, pp. 759–764.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv:1412.6980*.

Knyazev, Andrew V and Merico E Argentati (2002). "Principal angles between subspaces in an A-based scalar product: algorithms and perturbation estimates". In: *SIAM Journal on Scientific Computing* 23.6, pp. 2008–2040.

Kobak, Dmitry et al. (2016). "Demixed principal component analysis of neural population data". In: *Elife* 5, e10989.

Kobak, Dmitry et al. (2019). "State-dependent geometry of population activity in rat auditory cortex". In: *Elife* 8, e44526.

Laughlin, Simon B (2001). "Energy as a constraint on the coding and processing of sensory information". In: *Current opinion in neurobiology* 11.4, pp. 475–480.

Laurent, Gilles (2002). "Olfactory network dynamics and the coding of multidimensional signals". In: *Nature reviews neuroscience* 3.11, pp. 884–895.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". en. In: *Nature* 521.7553. Number: 7553 Publisher: Nature Publishing Group, pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539.

Levy, William B. and Robert A. Baxter (1996). "Energy Efficient Neural Codes". In: *Neural Computation* 8.3, pp. 531–543. ISSN: 08997667. DOI: 10.1162/neco.1996.8.3.531.

Low, Ryan J et al. (2018). "Probing variability in a cognitive map using manifold inference from neural dynamics". In: *bioRxiv*, p. 418939.

Maass, Wolfgang (1997). "Networks of spiking neurons: the third generation of neural network models". In: *Neural networks* 10.9, pp. 1659–1671.

Maass, Wolfgang, Thomas Natschläger, and Henry Markram (Nov. 2002). "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations". In: *Neural Computation* 14.11. Publisher: MIT Press, pp. 2531–2560. ISSN: 0899-7667. DOI: 10.1162/089976602760407955.

Machens, Christian K, Ranulfo Romo, and Carlos D Brody (2010). "Functional, but not anatomical, separation of "what" and "when" in prefrontal cortex". In: *Journal of Neuroscience* 30.1, pp. 350–360.

Mancoo, Allan, Sander Keemink, and Christian K Machens (2020). "Understanding spiking networks through convex optimization". In: *Advances in Neural Information Processing Systems* 33.

Mante, Valerio et al. (2013). "Context-dependent computation by recurrent dynamics in prefrontal cortex". In: *nature* 503.7474, pp. 78–84.

Markram, Henry (2003). "Elementary principles of nonlinear synaptic transmission". In: *Computational models for neuroscience*. Springer, pp. 125–169.

Mastrogiuseppe, Francesca and Srdjan Ostojic (Aug. 2018). "Linking Connectivity, Dynamics, and Computations in Low-Rank Recurrent Neural Networks". In: *Neuron* 99.3, 609–623.e29. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2018.07.003.

Mazor, Ofer and Gilles Laurent (2005). "Transient dynamics versus fixed points in odor representations by locust antennal lobe projection neurons". In: *Neuron* 48.4, pp. 661–673.

Moreno-Bote, Rubén and Jan Drugowitsch (Dec. 2015). "Causal Inference and Explaining Away in a Spiking Network". en. In: *Scientific Reports* 5, p. 17531. ISSN: 2045-2322. DOI: 10.1038/srep17531.

Mountcastle, Vernon B (1957). "Modality and topographic properties of single neurons of cat's somatic sensory cortex". In: *Journal of neurophysiology* 20.4, pp. 408–434.

Musall, Simon et al. (2019). "Single-trial neural dynamics are dominated by richly varied movements". In: *Nature neuroscience* 22.10, pp. 1677–1686.

Niven, Jeremy E and Simon B Laughlin (2008). "Energy limitation as a selective pressure on the evolution of sensory systems". In: *Journal of Experimental Biology* 211.11, pp. 1792–1804.

Nocedal, Jorge and Stephen Wright (2006). *Numerical optimization*. Springer Science & Business Media.

Okazawa, Gouki et al. (2021). "The geometry of the representation of decision variable and stimulus difficulty in the parietal cortex". In: *bioRxiv*. DOI: 10.1101/2021.01.04.425244.

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, pp. 607–609.

Pandarinath, Chethan et al. (2018). "Inferring single-trial neural population dynamics using sequential auto-encoders". In: *Nature methods* 15.10, pp. 805–815.

Pang, Rich, Benjamin J Lansdell, and Adrienne L Fairhall (2016). "Dimensionality reduction in neuroscience". In: *Current Biology* 26.14, R656–R660.

Pardo-Vazquez, Jose L et al. (2019). "The mechanistic foundation of Weber's law". In: *Nature neuroscience*, pp. 1–10.

Pehlevan, Cengiz (May 2019). "A Spiking Neural Network with Local Learning Rules Derived from Nonnegative Similarity Matching". In: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. ISSN: 2379-190X, pp. 7958–7962. DOI: 10.1109/ICASSP.2019.8682290.

Pillow, Jonathan W et al. (2005). "Prediction and decoding of retinal ganglion cell responses with a probabilistic spiking model". In: *Journal of Neuroscience* 25.47, pp. 11003–11013.

Poirazi, Panayiota, Terrence Brannon, and Bartlett W Mel (2003). "Pyramidal neuron as two-layer neural network". In: *Neuron* 37.6, pp. 989–999.

Raposo, David, Matthew T Kaufman, and Anne K Churchland (2014). "A category-free neural population supports evolving demands during decision-making". In: *Nature neuroscience* 17.12, p. 1784.

Rigotti, Mattia et al. (2013). "The importance of mixed selectivity in complex cognitive tasks". In: *Nature* 497.7451, pp. 585–590.

Roitman, Jamie D and Michael N Shadlen (2002). "Response of neurons in the lateral intraparietal area during a combined visual discrimination reaction time task". In: *Journal of neuroscience* 22.21, pp. 9475–9489.

Rolls, Edmund T, Alessandro Treves, and Edmund T Rolls (1998). *Neural networks and brain function*. Vol. 572. Oxford university press Oxford.

Romo, Ranulfo and Emilio Salinas (2003). "Flutter discrimination: neural codes, perception, memory and decision making". In: *Nature Reviews Neuroscience* 4.3, pp. 203–218.

Romo, Ranulfo et al. (1999). "Neuronal correlates of parametric working memory in the prefrontal cortex". In: *Nature* 399.6735, pp. 470–473.

— (2016). "Single-neuron spike train recordings from macaque prefrontal cortex during a somatosensory working memory task". In: *CRCNS. org*.

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Roweis, Sam and Zoubin Ghahramani (1999). "A unifying review of linear Gaussian models". In: *Neural computation* 11.2, pp. 305–345.

Roweis, Sam T and Lawrence K Saul (2000). "Nonlinear dimensionality reduction by locally linear embedding". In: *science* 290.5500, pp. 2323–2326.

Rubin, Daniel B., Stephen D. Van Hooser, and Kenneth D. Miller (Jan. 2015). "The Stabilized Supralinear Network: A Unifying Circuit Motif Underlying Multi-Input Integration in Sensory Cortex". In: *Neuron* 85.2, pp. 402–417. ISSN: 0896-6273. DOI: 10.1016/j.neuron.2014.12.026.

Saxena, Shreya and John P Cunningham (2019). "Towards the neural population doctrine". In: *Current opinion in neurobiology* 55, pp. 103–111.

Shadlen, Michael N and William T Newsome (2001). "Neural basis of a perceptual decision in the parietal cortex (area LIP) of the rhesus monkey". In: *Journal of neurophysiology* 86.4, pp. 1916–1936.

Sorbaro, Martino et al. (2020). "Optimizing the energy consumption of spiking neural networks for neuromorphic applications". In: *arXiv:1912.01268 [cs, q-bio]*.

Stevenson, Ian H and Konrad P Kording (2011). "How advances in neural recording affect data analysis". In: *Nature neuroscience* 14.2, pp. 139–142.

Stringer, Carsen et al. (2019a). "High-dimensional geometry of population responses in visual cortex". In: *Nature* 571.7765, pp. 361–365.

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

Stuart, Greg, Nelson Spruston, and Michael Häusser (2016). *Dendrites*. Oxford University Press.

Stöckl, Christoph and Wolfgang Maass (2020). "Classifying Images with Few Spikes per Neuron". In: *arXiv:2002.00860 [cs]*.

Sussillo, David et al. (July 2015). "A neural network that finds a naturalistic solution for the production of muscle activity". en. In: *Nature Neuroscience* 18.7, pp. 1025–1033. ISSN: 1546-1726. DOI: 10.1038/nn.4042.

Tang, Ping Tak Peter, Tsung-Han Lin, and Mike Davies (May 2017). "Sparse Coding by Spiking Neural Networks: Convergence Theory and Computational Results". In: *arXiv:1705.05475 [cs, q-bio]*.

Tenenbaum, Joshua B, Vin De Silva, and John C Langford (2000). "A global geometric framework for nonlinear dimensionality reduction". In: *science* 290.5500, pp. 2319–2323.

Vinck, Martin et al. (2015). "Arousal and locomotion make distinct contributions to cortical activity patterns and visual encoding". In: *Neuron* 86.3, pp. 740–754.

Vreeswijk, C. van and H. Sompolinsky (Dec. 1996). "Chaos in Neuronal Networks with Balanced Excitatory and Inhibitory Activity". In: *Science* 274.5293. Publisher: American Association for the Advancement of Science Section: Reports, pp. 1724–1726. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.274.5293.1724.

Whiteway, Matthew R and Daniel A Butts (2017). "Revealing unobserved factors underlying cortical activity with a rectified latent variable model applied to neural population recordings". In: *Journal of neurophysiology* 117.3, pp. 919–936.

Whiteway, Matthew R et al. (2019). "Characterizing the nonlinear structure of shared variability in cortical neuron populations using latent variable models". In: *Neurons, behavior, data analysis and theory* 3.1.

Wohrer, Adrien, Mark D Humphries, and Christian K Machens (2013). "Population-wide distributions of neural activity during perceptual decision-making". In: *Progress in neurobiology* 103, pp. 156–193.

Wright, Stephen and Jorge Nocedal (1999). "Numerical optimization". In: *Springer Science* 35.67-68, p. 7.

Yuste, Rafael (2015). "From the neuron doctrine to neural networks". In: *Nature reviews neuroscience* 16.8, pp. 487–497.

Zenke, Friedemann and Surya Ganguli (Apr. 2018). "SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks". In: *Neural Computation* 30.6. Publisher: MIT Press, pp. 1514–1541. ISSN: 0899-7667. DOI: 10.1162/neco_a_01086.

Appendix A

Gradient derivations for QP autoencoder

To fit the parameters of the autoencoder, we first phrased it as a bi-level optimisation problem (equation 4.17) and minimize the reconstruction error of the model using gradient descent. We now show how the gradient with respect to (w.r.t.) parameters, $\mathbf{D} \in \mathbb{R}^{M \times N}$ can be computed where N is the number of neurons and M is the number of latent variables. Note that these parameters are implicit in the lower level problem (QP) and thus, to compute the gradient we will need to differentiate through the lower-level problem. We will use an implicit differentiation method at the fixed points of the QP which we approximated using a log-barrier function. For a data point, recall that the following equation is satisfied:

$$t\Big([\boldsymbol{D}^\top \boldsymbol{D} + \mu \boldsymbol{I}]\hat{\boldsymbol{r}} - \boldsymbol{D}^\top \boldsymbol{z}\Big) - \hat{\boldsymbol{r}}^{-1} = 0 \tag{A.1}$$

where $\mathbf{z} = \mathbf{D}\mathbf{r}$ are the estimated latent variables obtained through a linear mapping of the observed population activity, $\mathbf{r}$. $\hat{\mathbf{r}}$ is the model prediction (QP solution), parameter t controls the log-barrier penalty and $\hat{\mathbf{r}}^{-1}$ denotes an N-dimensional vector with element $\frac{1}{\hat{r}_i}$ at the i^{th} entry. To compute the gradient, we need to reach an expression for the 3D tensor $\frac{\partial \hat{\boldsymbol{r}}}{\partial \mathbf{D}}$ (see equation 4.18) and we do so by taking the implicit derivative w.r.t. $\mathbf{D}$ on both sides of the above equation A.1. We first consider part $\Big(\boldsymbol{D}^\top \boldsymbol{D} + \lambda \boldsymbol{I}\Big)\hat{\boldsymbol{r}} - \boldsymbol{D}^\top \boldsymbol{z}$ of equation 4.18, and we would like to find an expression of its derivative w.r.t. $\mathbf{D}$. In sum notations, this is:

$$\frac{\partial}{\partial D_{mn}} \left(\underbrace{\sum_j^N \sum_l^M D_{il}^\top D_{lj} \hat{r}_j}_{1} + \lambda \hat{r}_i - \underbrace{\sum_l^M D_{il}^\top z_l}_{2} \right) \tag{A.2}$$

Computing the derivatives w.r.t. terms 1 and 2:
Term 1 (using product rule):

$$\frac{\partial}{\partial D_{mn}}\left(\sum_{j}^{N}\sum_{l}^{M} D_{il}^{\top} D_{lj}\hat{r}_j\right) = \sum_{j}^{N} D_{mj}\hat{r}_j\delta_{i,n} + \sum_{j}^{N}\sum_{l}^{M} D_{il}^{\top} D_{lj}\frac{\partial \hat{r}_j}{\partial D_{mn}} + D_{im}^{\top}\hat{r}_n$$

Term 2 (using product rule):

$$\frac{\partial}{\partial D_{mn}}\sum_{l}^{M} D_{il}^{\top} z_l = z_m\delta_{i,n} + D_{im}^{\top} r_n$$

Joining these together, expression (A.2) is equivalent to:

$$\sum_{j}^{N} D_{mj}\hat{r}_j\delta_{i,n} + \sum_{j}^{N}\sum_{l}^{M} D_{il}^{\top} D_{lj}\frac{\partial \hat{r}_j}{\partial D_{mn}} + D_{im}^{\top}\hat{r}_n + \lambda\frac{\partial \hat{r}_i}{\partial D_{mn}} + \sum_{j}^{N}\beta u_i u_j\frac{\partial \hat{r}_j}{\partial D_{mn}} - z_m\delta_{i,n} - D_{im}^{\top} r_n \tag{A.3}$$

Then, taking the derivatives of the remaining terms in equation A.1 and plugging in the above expression, we can show that, after moving terms around, we get:

$$\sum_{j}^{N}\left(\sum_{l}^{M} tD_{il}^{\top} D_{lj} + t\lambda\delta_{i,j} + \frac{1}{\hat{r}_j^2}\delta_{i,j}\right)\frac{\partial \hat{r}_j}{\partial D_{mn}} = t\left(z_m\delta_{i,n} + D_{im}^{\top} r_n - \sum_{j}^{N} D_{mj}\hat{r}_j\delta_{i,n} - D_{im}^{\top}\hat{r}_n\right)$$

Using the following shorthand notations $\mathbf{z} - \mathbf{D}\hat{\mathbf{r}} = \mathbf{b}$, $\mathbf{H} = \mathbf{D}^{\top}\mathbf{D} + \lambda\mathbf{I}$ and reconstruction error, $\mathbf{e} = \mathbf{r} - \hat{\mathbf{r}}$, the above equation simplifies to:

$$\sum_{j}^{N}\left(tH_{ij} + \frac{1}{r_j^2}\delta_{i,j}\right)\frac{\partial \hat{r}_j}{\partial D_{mn}} = t\left(b_m\delta_{i,n} + D_{im}^{\top} e_n\right) \tag{A.4}$$

Note that $\frac{\partial \hat{\boldsymbol{r}}}{\partial \mathbf{D}}$ is a stack of matrices and we can rearrange it such that a matrix in this stack, denoted as $\mathbf{J}^m$, is the Jacobian matrix $\frac{\partial \hat{\boldsymbol{r}}}{\partial \mathbf{D}_m}$ where $\mathbf{D}_m$ is the m^{th} row of $\mathbf{D}$. $\boldsymbol{J}^m$ will be an $N \times N$ matrix, with its (j, n)-entry being the derivative of neuron j w.r.t. the weight of neuron n for given latent variable indexed, m. Using the additional shorthand notation of $\boldsymbol{A}$ being an $N \times N$ diagonal matrix with j^{th} diagonal element to be $\frac{1}{\hat{r}_j^2}$, we can write the derivative with respect to $\mathbf{D}_m$ in matrix-vector format as:

$$(t\boldsymbol{H} + \boldsymbol{A})\,\boldsymbol{J}^m = t\left(b_m\boldsymbol{I} + (\boldsymbol{D}_m)^{\top}(\boldsymbol{e})^{\top}\right) \tag{A.5}$$

Note that only the right hand side (RHS) of this equation changes for different data points since only the reconstruction error $\mathbf{e}$ would change. We introduce the sample index, k, to emphasize this dependency and rewrite the RHS in shorthand format,

$\boldsymbol{C}_k^m = t\left(b_m \boldsymbol{I} + (\boldsymbol{D}_m)^\top (\boldsymbol{e}_k)^\top\right)$. Thus, our Jacobian matrix could be evaluated as follows for a data point:

$$\boldsymbol{J}^m = (t\boldsymbol{H} + \boldsymbol{A})^{-1} \boldsymbol{C}_k^m \tag{A.6}$$

However, we note that we do not need an explicit form for the Jacobian matrix, but rather want to compute its matrix-vector product with the error, i.e., $\boldsymbol{e}_k^\top \boldsymbol{J}^m$, as in equation 4.18. This means that for every latent variable indexed $m \in \{1, \ldots, M\}$, we need to compute the following dot product:

$$\boldsymbol{e}_k^\top \boldsymbol{J}^m = \boldsymbol{e}_k^\top (t\boldsymbol{H} + \boldsymbol{A})^{-1} \boldsymbol{C}_k^m = \boldsymbol{\kappa}^\top \boldsymbol{C}_k^m \tag{A.7}$$

Thus, we do not need to invert the matrix $(t\boldsymbol{H} + \boldsymbol{A})$, but rather solve the following system of linear equations for $\boldsymbol{\kappa}$:

$$(t\boldsymbol{H} + \boldsymbol{A})\, \boldsymbol{\kappa} = \boldsymbol{e}_k \tag{A.8}$$

Hence, to compute a row in the matrix of derivatives, $\frac{\partial L}{\partial \mathbf{D}} \in \mathbb{R}^{M \times N}$ (equation 4.18), we need to solve for $\boldsymbol{\kappa}$ in equation A.8 and compute $\boldsymbol{C}_k^m$ for a data point, indexed k.

RÉSUMÉ

Les enregistrements à grande échelle de l'activité neuronale sont maintenant largement réalisés dans de nombreux laboratoires, ce qui soulève l'importante question de comment extraire les structures essentielles des données. Une approche courante consiste à réduire leur dimensionnalité. Cependant, l'interprétation des résultats peut être parsemée de difficultés. Le plus souvent, les méthodes linéaires telles que l'analyse en composantes principales (ACP) fonctionnent bien pour trouver des projections linéaires des données qui expliquent la majorité de la variance, mais généralement, elles présentent également une suite de composantes, dont plusieurs ressemblent à des fonctions d'ordre supérieur de certaines autres composantes. On les appellera "composantes d'ordre supérieur" (COS). Bien que ces COS suggèrent que la variété neurale est non linéaire, il n'est pas encore clair comment elles apparaissent et ce qu'elles signifient.

Nous soutenons ici que ces COS apparaissent en grande partie à cause d'une non-linéarité bien connue — l'activité neuronale est non-négative, ce qui fait plier la variété neurale, mais les COS résultants ne sont pas pertinents d'un point de vue fonctionnel. Nous menons notre analyse en partant de deux hypothèses essentielles: la lecture de l'activité de la population doit être linéaire et à faible dimension, et le taux d'activité global doit être limité pour des raisons énergétiques. Nous montrons, dans des simulations, que lorsque des activités neurales sont générées sous ces hypothèses, alors l'ACP extrait parfois les vrais signaux sous-jacents, mais affiche souvent une suite de COS pour compenser la courbure de la variété. Nous expliquons ces résultats de manière géométrique et proposons des méthodes de type ACP qui incorporent les contraintes de non-négativité de manière constructive. Nous validons nos méthodes avec des données de simulation, mais nous montrons aussi, sur un exemple de données expérimentales, que l'incorporation de cette simple non-linéarité permet une description plus concise que l'ACP.

Cependant, il est possible qu'en plus des contraintes de non-négativité, la variété neurale présente une certaine non-linéarité supplémentaire. Nous supposons, en dernier lieu, que cette non-linéarité peut apparaître en fonction des computations effectuées par le réseau. Cependant, une compréhension claire de ces computations par des réseaux neuronaux proches de la biologie est toujours manquante. Nous complétons cette thèse en examinant les computations accomplies par les réseaux de neurones à impulsions (SNNs), dans un nouveau cadre inspiré de la théorie de l'optimisation convexe. Nous montrons qu'une large gamme de réseaux neuronaux calculent essentiellement des fonctions d'entrée-sortie convexes. De plus, ces réseaux peuvent également afficher plusieurs caractéristiques biologiques telles que des suites de potentiels d'action asynchrones et irréguliers, la robustesse aux perturbations, entre autres.

MOTS CLÉS

Réduction de la dimensionnalité, contraintes de non-négativité, réseaux de neurones à impulsions

ABSTRACT

Large-scale recordings of neural activity are now widely carried out in many experimental labs, leading to the question of how to extract the essential structures in population data. One common approach is to use dimensionality reduction methods. However, interpretation of the results of these tools can be fraught with difficulties. Most commonly, linear methods such as principal component analysis (PCA) work well in finding linear projections of the data that explain most variance, but usually, also display a tail of components, among which several resemble higher-order functions of some other components, or 'higher-order' components (HOCs). While these HOCs suggest that the true neural manifold is non-linear, it is still unclear how they emerge and what they mean.

In this thesis, we argue that these HOCs largely arise due to a well-known non-linearity — individual neuronal activity is non-negative, which bends the neural manifold, but the resulting HOCs are otherwise functionally irrelevant. We lead our investigation with the crucial assumptions that readouts of population activity should be linear and lower-dimensional, and that overall firing rates should be limited for energetics reasons. We show, in simulations, that when neural activities are generated under these assumptions, then PCA sometimes extracts the true underlying signals, but often displays a tail of HOCs to compensate for the curvature of the manifold. We explain these findings geometrically and propose a set of PCA-like methods that incorporate the non-negativity constraints in a meaningful way. We validate our methods against ground truth data, but also show, in an example experimental dataset, that incorporating this simple non-linearity affords more succinct representations of the data than PCA.

However, it is possible that the neural manifold exhibits some additional non-linearity besides the non-negativity constraints. We hypothesize lastly, that this non-linearity may emerge according to the computations done by the network. Yet, a clear understanding of such computations by biologically realistic neural networks is still missing. We complement this thesis by investigating the computations done by spiking neural networks (SNNs), within a new framework inspired from convex optimisation theory. We show that a broad class of SNNs fundamentally compute convex input-output functions. Interestingly, these networks can also display several biological features such as asynchronous and irregular spike trains, robustness to perturbations, among others.

KEYWORDS

Dimensionality reduction, non-negativity constraints, spiking neural networks