Latent Variables Related to Behaviour in Neural Activity

Alessandro Facchin

Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2021

Abstract

Human brain encodes information through activity patterns in populations of neurons. Recent advancements in multi-electrode interfaces allow researchers to record large clusters of neurons simultaneously with a single neuron precision. Latent variables models have been shown effective in extracting signals from these populations while disregarding the background noise, typical of in neural recordings. More recently, a few models have tried to represent both neural activity and behaviour through latent variables extracted from neural recordings. In this project, we focus our attention on LFADS [1], the state-of-the-art method for the extraction of latent variables from neural recordings. We then compare the performance of LFADS with TNDM [2], a recent extension of the same model that takes into account behaviour. We show TNDM outperforms LFADS for small training samples, that it can decode wrist electromyography (EMG) signals when applied to the primary motor cortex (M1) and that it is able of disentangling behaviour-relevant and behaviour-irrelevant latent variables. Among the contributions of this project, a new Python implementation for both algorithms based on TensorFlow2. The novel implementation offers faster training, a simpler interface and a cleaner, shorter codebase, which will be easier to maintain. This implementation was employed for all the experiments provided in this project.

Acknowledgements

I would like to deeply thank my supervisor, Dr. Matthias Hennig for his support throughout this project. His clear guidance, well-pondered feedback and insightful advice were fundamental throughout my research. I would like to extend my graditude to some of the other bright minds I had the pleasure to collaborate with during the project. Cole Hurwitz, Dr. Nina Kudryashova and Dr. Kianoush Nazarpour, I am deeply thankful for your positive encouragement and constructive comments.

I want to thank Aberdeen Standard Investments for the extraordinary opportunity they gave when they decided to welcome in their team and sponsor my education. Owen McCrossan who first believed in me, Yannis Papakonstantinou, David Sol and Dr. Roger Senior are just some of the amazing people I had the chance to meet there.

Balancing work and study requires effort from both the individual and the employer. I am thankful for the flexible approach adopted by Grigorios Papamanousakis and more in general Prometheus Technologies, my current company. Despite the dynamic environment typical of startups, they were able to understand how important this degree was for me and allow me to work on it.

Last but not least, I would like to express my love and gratitude towards my family. In particular, my grandparents, my mother, my father, my sister and my girlfriend. They have always been there for me and this would not have been possible without them.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Alessandro Facchin*)

Table of Contents

Chapter 1

Introduction

As humans, every day we sense information about the world around us, plan our future, perform some actions and feel emotions. Our brain is the organ responsible for all these actions. Studied since the 17th century BC [3], the brain was recognized as the central organ in our cognitive process only two centuries ago, thanks to the pioneering work of Santiago Ramón y Cajal [4]. The Spanish neuroscientist was the first to recognise the role of neurons and the connections between them, called synapses, in processing information.

Two centuries and many discoveries later, we know that information is encoded in networks of neurons through electrical activity. In each neuron, the electrical activity is characterised by stereotypical discrete events, called spikes, where the membrane rapidly depolarises from -50mV to +10mV and then hyperpolarises back to -70mV. Recent advancements in medicine, material engineering and electronics enabled the development of neural implants able to capture the activity of large clusters of neurons simultaneously, with single-neuron spatial precision [5, 6, 7]. At the time of writing, it is possible to monitor over 10^4 neurons using slow (10Hz) calcium imaging [8] and over 10^3 neurons through multi-electrode recordings [9, 10], which can reach sampling rates of 30KHz. The relationship between the neural activity and the encoded information has been investigated for decades [11, 12], but it is only thanks to the recent technological advances that we were able to investigate complex dynamics.

Throughout the years, several hypothesis were formulated around the encoding of information in our brain. Clusters of neighboring neurons in the primary virtual cortex (V1) were shown to activate when stripes with specific angles were displayed in front of the animal, a phenomenon called *population code* [13]. This finding seemed to suggest that distributed encoding was the preferred solution adopted by the brain, until

Quiroga et al. [14] observed that in the medial temporal lobe (MTL) some information was encoded in as little as a single neuron [14]. The amount of neurons involved in encoding a single piece of information in the brain is referred to as *sparseness*. A sparse encoding can more efficiently encode a large amount of information in a small population. In critical brain areas this comes at a cost, as sparseness increases the chances of a message being corrupted due to some background noise. Decreasing the chances of a corrupted message might be the reason underpinning redundancy in our brain [15].

The first medical and commercial applications of brain computer interfaces (BCI) [16, 17] created a growing demand for models that are able to extract from neural activity the information relevant to behaviour and perception. The challenging task of separating the signals in neural population activity from the background is of paramount importance for the development of systems downstream, whether they are prosthetic limbs [18], a system to prevent epileptic seizures [16] or a keyboard interface for your computer [19]. This task can be further complicated by sparse representations of the behaviour investigated in the neural activity, noise components in both the neural activity and behaviour or the limited availability of training samples. Improvements in the data efficiency, accuracy and speed of the models translate into faster, more precise and more robust neural decoding, which in turn enables superior systems downstream.

This project will analyse two algorithms for extracting small sets of variables that describe the variability observed in the neural activity, for this reason called *latent variables*. The first algorithm, Latent Factor Analysis via Dynamical Systems (LFADS) [1], is the *de-facto* standard solution for extracting non-linear latent variables from neural activity, producing state-of-the-art performance in multiple datasets. This algorithm will be compared to the recently developed Targeted Neural Dynamical Modelling (TNDM) [2], a derived model specialised in extracting latent variables related to behaviour. After re-implementing and updating both algorithms to a modern machine learning framework (TensorFlow 2), we will test them on both synthetic and recorded data, evaluating their accuracy, data efficiency and the differences in the latent factors extracted.

After a brief literature review in Chapter 2, we will describe how LFADS' architecture compares to TNDM in Chapter 3. Chapter 4 will describe the new TensorFlow2 implementation and compare it to the legacy code in terms of speed, resources utilisation and performance. A Lorenz system will generate the synthetic data used to compare the two models in Chapter 5, while Chapter 6 will apply both algorithms to

a dataset containing neural, muscular and behavioural activity from a primate, resembling the typical prosthetic scenario. Finally, Chapter 7 contains a summary of our findings and identifies some promising areas for further work. The code associated with this document is available on the GitHub project `alessandrofacchin/msc-project`.

Chapter 2

Background

The human brain typically encodes information through discrete events called spikes, which are generated by neurons and consist of a rapid increase in the voltage potential of a neuron's membrane, followed by a reduction beyond the resting potential, called hyper-polarization, and then a slower return to the resting potential [20, 21, 22].

Previous research has shown how neural encoding can differ substantially across brain areas [13, 14]. With few exceptions, information tends to be encoded by clusters of neurons rather than single units, a phenomenon known as *population coding*. This amount of neurons used to encode a single piece of information is associated with the concept of *redundancy*, often opposed to *sparsity*.

Patterns in spikes, or lack thereof, can be recorded across multiple channels using electrode arrays with over 10^3 channels [23, 6, 10]. Analysing populations of such size requires a systematic approach which takes into consideration the behaviour of the population as a whole, in order to filter-out noise at a single neuron level. Statistical models applied to large scale neural recordings are an active area of research. A reasonable portfolio of algorithms is already available for industrial application and improved versions of the existing models are published on a yearly basis. The wide range of algorithms developed so far can be split into *correlation* models and *latent variables* models [7].

2.1 Correlation Models

In correlation models neural activity is represented as a distribution, where the correlation structure between neurons is modelled explicitly. Macke et al. [24] apply Dichotomized Gaussian models, where the density function of a discretised multivariate

Gaussian is fitted to the population firing activity.

Savin and Tkačik [25] increase the capacity of the representation by applying a Maximum Entropy model, which can represent complex relationships, but requires additional computational resources. Generalized Linear models [26, 27] are particularly effective when the external inputs are provided and can often be formulated with convex utility function, resulting in cost-effective training procedures.

An even higher level of detail is obtained by integrating copulas [28, 29]. Such algorithms model separately the parameters of the firing distribution and the density functions themselves, allowing for complex non-linear dependencies between clusters of neurons, in line with biological properties of neural populations.

2.2 Latent Variables Models

While correlation models model the population's firing distribution by modelling single neurons individually, latent variables model extract aims at extracting a small set of variables that describe the behaviour of the population as a whole. Latent variables methods gained popularity following the discovery that neural activity was represented efficiently in a low dimensional manifold in several brain areas [30, 31, 32]. More recently, some experimental findings undermined the low-manifold assumption by showing how dimensionality can increase substantially on the primary visual cortex (V1) [33]. As a consequence, the applicability of latent variables models should not be given for granted when exploring uncharted brain areas.

Two latent variables paradigms have produced significant results: *state-space* and *temporal* models. In state-space models, the dynamics of latent variables are described by transition functions, that express the latent variables of each timestep based on the previous ones. On the other hand, temporal algorithms model time-dependent relationships, updating latent variables' trajectories as a function of time.

Several general-purpose dimensionality reduction techniques have been applied on neural populations, among all Principal Component Analisis (PCA), which has been used both to describe static relationships [34, 35] and state-space systems [36, 37]. Non-linear architectures have become popular in recent years, thanks to their ability to model more closely the non-liearities typical of brain dynamics [38].

Latent Factor Analysis via Dynamical Systems (LFADS) [1] is one of the central algorithms in this document. In LFADS, latent variables are modelled through a recurrent neural network (RNN) [39], that allows for non-linear state-space dynamics. A

bottleneck structure typical of Variational Autoencoders (VAE) [40] enforces a simple latent representation despite the non-linearities in the RNNs, resulting in state-of-art performances. The architecture of LFADS is analysed more in detail in section 3.1.

A recent framework based on evolutionary algorithms extended LFADS' usability by creating AutoLFADS, a framework that tunes hyperparameters to fit most practical applications [41]. Inspired by the popularity of transformers in natural language processing and machine translation, Ye and Pandarinath [42] developed a new model which replaces the RNN in LFADS with self-attention blocks, resulting in faster inference time thanks to the shallower network architecture. Another interesting approach from Kim et al. [43] relies on neural ordinary differential equations and claims to improve the state-of-art performance in inferring latent trajectories underlying a neural population, particularly in sparse conditions. Their method currently requires significantly longer training times than LFADS and TNDM, but could provide additional insights into the phase portraits and fixed points of neural activity.

2.3 Joint Neural-Behavioural Models

Recent improvements in the precision of electromyography (EMG) analysis [44] and the increasing importance of Brain Computer Interfaces (BCIs) in prosthetics fueled the development of a new sub-field, with researchers increasingly interested in extracting latent variables that can explain the behaviour of the animal. Preferential Subspace Identification (PSID) [45] is the most popular method in this space, extracting behaviour-relevant latent variables through a linear dynamical system which relies heavily on vector projections. Behaviour has often been shown to be linked with neural activity through non-linear relationships [46]. In addition, neural activity is rather sparse in time, therefore some degrees of integration or smoothing are necessary to understand the big picture. Algorithms that can simultaneously model non-linear relationship and extract latent variables from entire populations are naturally advantaged in extracting behaviour-relevant latent variables. Their configuration allows them to ignore the idiosyncratic noise of single neurons and integrate information both horizontally across neurons and vertically across time.

Targeted Neural Dynamical Modelling (TNDM) [2] is an adaptation of LFADS that integrates behaviour in the training process, while retaining LFADS' ability to model non-linear latent dynamics. Its peculiar architecture splits latent variables into two subspaces, for behaviour-relevant factors and behaviour-irrelevant factors. The

entire set of factors is used to reconstruct neural activity, while only behaviour-relevant latent variables are employed to reconstruct behaviour. A major advantage of TNDM compared with PSID is the flexibility of its behaviour decoder, which allows both to model same-time relationships between latent dynamics and behaviour and time lagged relationships. The structure of TNDM will be covered in further details in section 3.2.

Chapter 3

Methodology

The experiments proposed in this project focus on two architectures: Latent Factor Analysis via Dynamical Systems (LFADS) [1] and Targeted Neural Dynamical Modeling (TNDM) [2]. The former is the state-of-the-art algorithm for the extraction of latent variables from neural activity. TNDM, on the other hand, is an enhanced architecture that extends LFADS capabilities to the extraction of behaviour-relevant latent variables. This chapter explores the two architectures and the training procedure adopted throughout the project.

Both algorithms receive the activity in a neural population as an input. For a given trial, the activity is represented as a two-dimensional matrix $\mathbf{X} \in \mathbb{N}^{T \times N}$ where the first dimension T is the number of time intervals for each trial and the second dimension N represents the number of neural channels recorded. Each point of the neural activity x_{ij} corresponds to the number of spikes recorded during the time interval i on the neural channel j.

3.1 Latent Factor Analysis via Dynamical Systems

The high-level architecture of LFADS can be broken down into an *encoder* and a *decoder*, combined sequentially to form a structure that is often referred to as *auto-encoder*. Figure 3.1 illustrates the main layers in the network – dropouts, concatenation and reshaping layers have been omitted for readability reasons.

For a given trial, at each time interval the *encoder* receives the neural activity $\mathbf{x}_t \in \mathbb{R}^N$ and summarizes it into a lower dimensional latent vector $\mathbf{g}_0 \in \mathbb{R}^D$, where D is the size of the latent space. The latent vector is then fed to the *decoder*, that will de-compress the information to match as closely as possible the activity in the

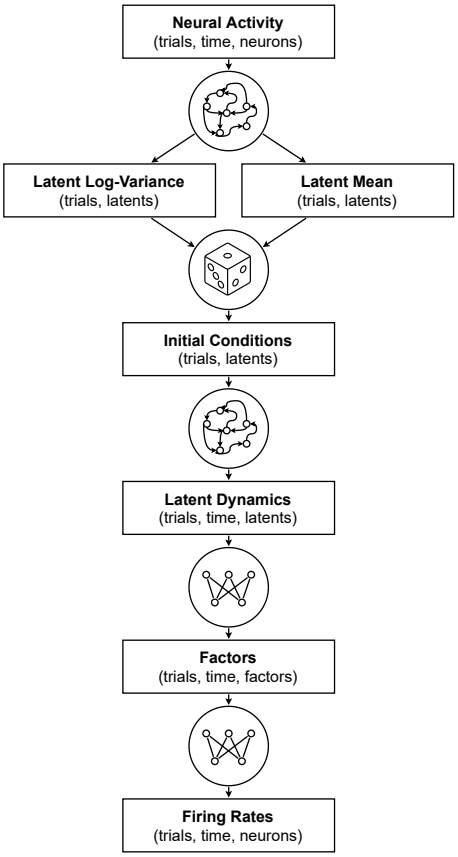


Figure 3.1: Architecture of LFADS.

input neural activity. While this structure is common to most auto-encoders, a few features distinguish LFADS from other types of *auto-encoders*, such as the presence of recurrent neural networs (RNNs) [39] and the regularization of the latent space using Kullback–Leibler divergence [47].

3.1.1 Encoder

The central element in the encoding procedure of LFADS is a recurrent neural network. In line with the original LFADS codebase, our novel implementation employs the gated recurrent unit (GRU) from Cho et al. [48] as the fundamental cell of the RNN. A dropout layer is applied to the neural activity as it enters the single-layered bi-directional encoder RNN. The outputs from the last unit of the forward direction and from the first unit of the backward direction are concatenated, and dropped out a second time, resulting in the encoding vector $\mathbf{y}$.

As in other VAEs, in LFADS the latent space encodings are modelled as distributions rather than deterministic vectors. The distribution used is a multivariate Gaussian

with a diagonal covariance matrix, where each dimension is independently sampled. The output of the encoder RNN flows through two separate dense layers, W_μ and W_{σ^2}, outputting for the mean and log-variance parameters of the encoder distribution:

$$\mu = W_\mu \times \mathbf{y}$$
$$\boldsymbol{\sigma^2} = \exp(W_{\sigma^2} \times \mathbf{y})$$

The log-variance is preferred to the variance because it is expressed in $[-\infty, +\infty]$ rather then in $[0, +\infty]$, such that the output of the previous fully-connected layer can be used without the need of an activation. The resulting distribution is then used to sample the initial condition $\mathbf{g}_0$ that will be provided to the decoder:

$$\mathbf{g}_0 \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 \odot \mathbb{1}) \tag{3.1}$$

3.1.2 Decoder

The main difference between the encoding and decoding process lies in the different usage of recurrent cells. In the encoding process, the recurrent architecture is used to extract and compress the information from the neural spikes into a limited set of variables. The decoder, on the other hand, ingests the low-dimensional latent space and generates autonomous temporal dynamics. For each trial, the encoded representation $\mathbf{g}_0$ used to seed the initial state of the RNN. The inputs to the RNN at each time step are set to zero, resulting in a generative process that is solely dependent on the initial state, also called *autonomous*. The state-space model described by the RNN results in the following generative process:

$$\mathbf{g}_{t+1} = \mathrm{GRU}(\mathbf{g}_t) \tag{3.2}$$

The latent dynamics $\mathbf{g}_t \in \mathbb{R}^D$ are reduced in size through another linear transformation W_z, resulting in a limited set of latent factors $\mathbf{z}_t$:

$$\mathbf{z}_t = \mathbf{W}_z \times \mathbf{g}_t \tag{3.3}$$

One of the key assumptions underlying this architecture is the linear synchronous relationship between the latent factors and the log-firing rates. As a consequence, the low-dimensional latent factors are mapped to the log-firing rates $\mathbf{f}_t \in \mathbb{R}^N$ through a dense layer W_r:

$$\mathbf{f}_t = \mathbf{W}_r \times \mathbf{z}_t \tag{3.4}$$

3.1.3 Training

The training procedure in LFADS reflects the most common practices in VAEs. VAEs are typically trained through backpropagation by maximising the evidence lower-bound (ELBO) of the marginal log-likelihood, which is composed of a divergence loss and a reconstruction reward. The divergence loss is employed to regularize the latent space and is computed as the Kullback–Leibler (KL) divergence [47] between the *prior* distribution $\mathcal{N}(0, \sigma_p)$ of the latent space and the encoded (or *posterior*) distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 \odot \mathbb{1})$:

$$\mathrm{KL}_d = \log\left(\frac{\sigma_d^2}{\sigma_p}\right) + \frac{\sigma_p^2 + \mu_d^2}{2\sigma_d^2} - \frac{1}{2} \tag{3.5}$$

Unlike most VAEs, in LFADS the decoding procedure does not directly reconstruct the input information, but rather estimates the underlying log-firing rates $\mathbf{f}_t$, under the assumption of a Poisson spiking process. For each trial, the ELBO to maximise is computed as:

$$\mathrm{ELBO} = -\sum_{d=1}^{D} \mathrm{KL}_d + \sum_{t=1}^{T} \log p[\mathbf{x}_t | \exp(\mathbf{f}_t)] \tag{3.6}$$

where D is the size of the initial conditions and T is the number of time intervals. During training, the algorithm seeks a trade-off between a regular latent space, where the *prior* and the *posterior* are highly aligned, and a higher information throughput, which usually results in better reconstruction.

The network is regularised during training both through dropout layers and an L2 regularization on the network's weights. Dropout layers are present at the entrance and exit of the encoding RNN, as well as between each layer of the generator RNN. The L2 regularization, on the other hand, is applied to every layer in the network. The original source code of LFADS assumes a constant regularization weight across all layers, while our novel implementation provides the ability to specify different scaling factors in each layer. In general, for a given layer m, the L2 regularization loss can be written as:

$$\ell_m = k_m \sum_{i=1}^{W_m} \|w_i\|_2 \tag{3.7}$$

where k_m is the layer-specific constant and w_i the i-th weight of the layer and W_m is the number of weights associated with the layer m.

VAEs often suffer from a phenomenon called *posterior collapse*, where the posterior distribution converges to the prior and fails to learn any useful latent representation [49, 50]. In order to avoid such scenario, we introduce an hyperparameters scheduling

that gradually increases the weights of the L2 regularization and of the KL divergence. As a result, the loss function minimised throughout the experiment becomes:

$$\mathcal{L} = -\sum_{t=1}^{T} \log p[\mathbf{x}_t | \exp(\mathbf{f}_t)] + \lambda_1 \sum_{d=1}^{D} \mathrm{KL}_d + \lambda_2 \sum_{m \in M} \ell_m \tag{3.8}$$

where M is the set of all layers, $\log p[\mathbf{x}_t | \exp(\mathbf{f}_t)]$ is the Poisson log-likelihood of spiking events given firing rates and the weights λ_1 and λ_2 are updated at each back-propagation step, from an initial value of 0 to a final value of 1.

3.2 Targeted Neural Dynamical Modeling

The second algorithm explored in this document combines LFADS' ability to extract non-linear latent variables from the neural activity with the intuition of Sani, Pesaran, and Shanechi [45] to exploit behavioural information in order to extract better latent variables. The result is a sequential VAE that shares the same encoder as LFADS, but decodes separately the *behaviour-relevant* and *behaviour-irrelevant* variables. Contrary to PSID [45], TNDM allows for both same-time and delayed reactions of the behaviour relative to the neural latent variables, such that shifts in behavioural activity are not needed.

While LFADS relies uniquely on neural activity, TNDM requires behavioural information during training to disentangle *behaviour-relevant* from *behaviour-irrelevant* variables and, as a consequence, predicts behaviour at test time. For this reason, LFADS should be classified as an unsupervised method, while TNDM would more likely fall in the semi-supervised range. A high-level diagram of TNDM is presented in fig. 3.2.

3.2.1 Encoder

The encoding procedure of TNDM is extremely similar to what we discussed about LFADS in section 3.1.1, with the only exception of the post-RNN dense layers. The encoder is now in charge of producing means and log-variances for two diagonal multivariate Gaussian distributions, one for the behaviour-relevant initial values $\mathbf{g}_{r0}$ and one for the behaviour-irrelevant initial values $\mathbf{g}_{i0}$. As a consequence, there are four separate dense layer acting on the same output of the RNN:

$$\mu_r = W_{r\mu} \times \mathbf{y}\ , \quad \boldsymbol{\sigma^2}_r = \exp(W_{r\sigma^2} \times \mathbf{y})$$
$$\mu_i = W_{i\mu} \times \mathbf{y}\ , \quad \boldsymbol{\sigma^2}_i = \exp(W_{i\sigma^2} \times \mathbf{y})$$

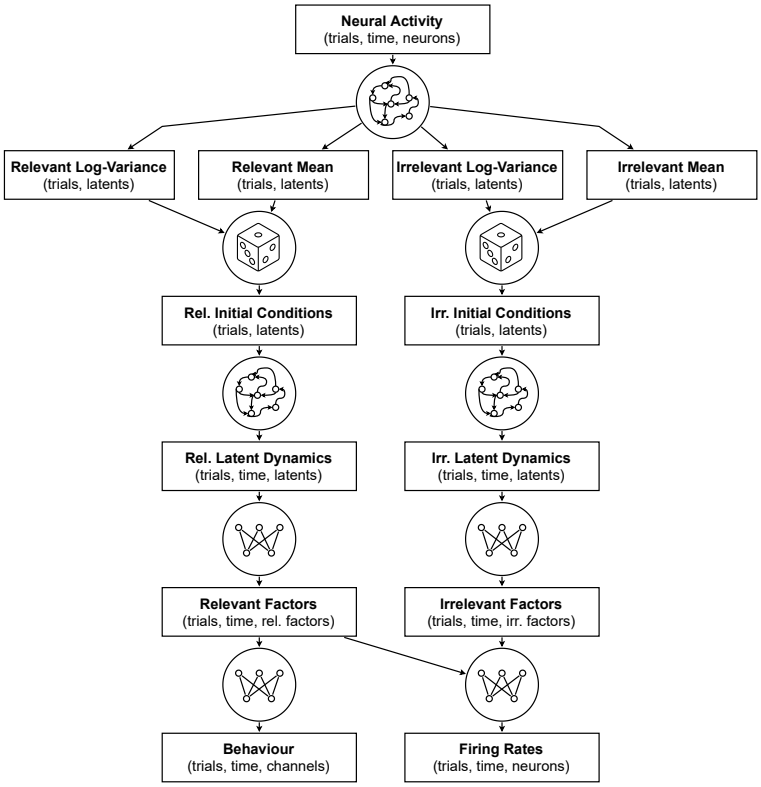


Figure 3.2: Architecture of TNDM.

Initial conditions for the two latent spaces are then sampled from the two distributions:

$$\mathbf{g}_{r0} \sim \mathcal{N}(\boldsymbol{\mu}_r, \boldsymbol{\sigma}_r^2 \odot \mathbb{1}) \ , \quad \mathbf{g}_{i0} \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2 \odot \mathbb{1})$$

3.2.2 Decoder

The decoder architecture is the most prominent difference between TNDM and LFADS. Two parallel threads are clearly identifiable in fig. 3.2, each dedicated to either behaviour-relevant or behaviour-irrelevant latent factors. While behaviour-irrelevant factors $\mathbf{z}_i t$ are merely dedicated to the reconstruction of neural firing rates, behaviour-relevant factors $\mathbf{z}_r t$ are used to reconstruct both neural activity and behaviour.

Formally, the initial conditions are fed to two distinct RNNs in order to generate the autonomous dynamics and the latent factors:

$$\mathbf{g}_{r(t+1)} = \mathrm{GRU}_r(\mathbf{g}_{rt}), \quad \mathbf{z}_{rt} = \mathbf{W}_{rz} \times \mathbf{g}_{rt}$$
$$\mathbf{g}_{i(t+1)} = \mathrm{GRU}_i(\mathbf{g}_{it}), \quad \mathbf{z}_{it} = \mathbf{W}_{iz} \times \mathbf{g}_{it}$$

Behaviour-relevant factors $\mathbf{z}_{rt}$ are used to predict behaviour using one of two methods: a single *synchronous* matrix or a set of diagonal *causal* matrices. The *synchronous*

method assumes synchronization between the $\mathbf{z}_{rt}$ and $\mathbf{b}_t$, while the *causal* matrices allow any linear relationship between $\mathbf{z}_{rt}$ and $\mathbf{b}_q$ if and only if $q \geq t$. The *synchronous* behaviour decoder is described as:

$$\hat{\mathbf{b}}_t = C \times \mathbf{z}_t \tag{3.9}$$

For the *causal* behaviour, we employ an $Z_r \times B$ set of lower triangular matrices, where Z_r is the dimension of the behaviour-relevant latent factors space and B is the number of behaviour variables recorded. Each behavioural point would be calculated as:

$$\hat{b}_{bt} = \sum_{i \in [1, Z_r]} \mathbf{C}_{ib} \times \mathbf{z}_i \ , \quad \text{with } \mathbf{C}_{ib} \text{ lower-triangular.} \tag{3.10}$$

Behaviour-relevant and irrelevant factors are eventually concatenated and used to calculate log-firing rates, in line with what happens in eq. (3.4) for LFADS.

3.2.3 Training

The loss function for TNDM is defined as the ELBO of the joint marginal log-likelihood of the neural activity and behaviour reconstruction. The same Poisson hypothesis described in section 3.1.3 is applied to the neural reconstruction, while a Gaussian distribution with fixed variance is used for the behavioural reconstruction.

A new disentanglement loss is introduced to avoid overlaps between the two latent spaces. For each trial in a given batch, a number of independent samples is generated from both $\mathcal{N}(\boldsymbol{\mu}_r, \boldsymbol{\sigma}_r^2 \odot \mathbb{1})$ and $\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2 \odot \mathbb{1})$. The sampled points are then concatenated for all the trials in the batch, to form a single matrix for each of the two latent spaces. The $\ell 2$ norm of the sample covariance matrix between the two matrices is then used as a penalty in the main loss function.

An hyperparameters scheduling procedure with a constant step updates the weights associated with the KL losses, the L2 loss and the disentanglement loss c. Fixed weights λ_n and λ_b are assigned to the neural and behavioural reconstruction losses instead, balancing the dual supervised-unsupervised nature of the algorithm.

$$\begin{aligned} \mathcal{L} = &- \lambda_n \sum_{t=1}^{T} \log p[\mathbf{x}_t | \exp(\mathbf{f}_t)] - \lambda_b \sum_{t=1}^{T} \log p[\mathbf{b}_t | \hat{\mathbf{b}}_t] + \\ &+ \lambda_1 \sum_{d=1}^{D_r} \mathrm{KL}_{rd} + \lambda_2 \sum_{d=1}^{D_i} \mathrm{KL}_{id} + \lambda_3 \sum_{m \in M} \ell_m + \lambda_4 c \end{aligned} \tag{3.11}$$

Chapter 4

Implementation

Nominally maintained and distributed as part of Tensorflow Model Garden [51] initiative, the original codebase of LFADS was developed in Python around four years ago. At the time, TensorFlow 1 [52] was the most consolidated neural networks framework together with PyTorch [53], while TensorFlow 2 was not yet released.

On September 2019, the second major release of TensorFlow introduced several improvements upon the first version. The API was simplified and consolidated, variable named scopes were replaced in favour of simpler objects and Eager Execution was introduced, resulting in a drastic reduction in startup time [54]. Since its first implementation, LFADS' source code was translated to PyTorch [55] and JAX [56], but it was never updated from TensorFlow 1 to TensorFlow 2.

Updating the codebase was the catalyst for a broader range of enhancements that improved the usability of the model and simplified the maintenance. Projects involving rigid time constraints often overlook the efforts required to maintain a codebase once the project terminates, focusing instead on short-term findings and ad-hoc analysis. This chapter will analyse the improvements introduced in the new implementation, that should simplify the usage and management of the existing LFADS model.

4.1 Refactoring

An initial selection of the features worth porting to the new versions, resulted in the exclusion of an auxiliary *controller* extension, which is sometimes used in the original version. This structure provides some additional information to the generator RNN described in section 3.1, potentially leading to a weak form of trivial copying, where a single lagged time series can flow from the encoder to the decoder without the KL

divergence penalty. For smooth slow dynamics, this could mean that the bottleneck structure is not effective.

A second feature that was removed from the source code was parser for the command line interface. In the legacy implementation, runtime parameters were provided as command line arguments to the Tensorflow app in order to kick off the training or evaluation process. The flat structure of the interface, combined with the complexity of the model resulted in the proliferating of a wide range of possible command line arguments, often with counter-intuitive names (appendix A.1).

The re-factored code is structured around two interfaces: a *model* interface and a *runtime* interface. The *model* interface is the result of the adoption of TensorFlow 2's best practices for building new architectures [57]. Core methods such as `build`, `fit`, `call`, `save` and `load` can be accessed as in built-in TensorFlow architectures. Users can now experiment with the model without any sort of learning curve, through a simple script or notebook (appendix A.2). This is a significant improvement over the legacy code, which cannot be accessed easily from a notebook, due to the lack of a Python interface and the high reliance on command line tools.

The runtime interface was designed to serve a different user, who might be interested in running several experiments through batch jobs, for example from within a remote cluster. The legacy LFADS offered a solution based on command line arguments, where changes in scripts were difficult to detect and commands were hard to navigate. The ability of both JSON and YAML files to represent nested complex structure and the wide availability of explorer for these files enabled the exposure of a much wider set of parameters despite a perceived reduction in the complexity of the interface (appendix A.3).

Throughout the refactoring, a number of custom-built functions in the LFADS implementation were switched to default methods from the TensorFlow 2 Keras library, in order to improve the efficiency and reduce the long-term maintenance cost. Some examples include the implementation of the KL divergence losses, the initialisation of the weights in the layers, the Poisson log-likelihood, GRU cells and the training loop itself. The result was a 49% reduction in the project lines of code, which went from 2199 to 1117[1]. If we limit the analysis to the model code, the codebase reduction reaches -63%, from 1390 to 514 lines.

The refactored LFADS codebase the starting point for the development of a Tensor-

[1] Ad-hoc plotting libraries were excluded in both cases, as they are not necessary to run and evaluate the model.

Flow 2 TNDM implementation. The large similarities between the two models and the adoption of SOLID principles [58] throughout the refactoring of LFADS significantly simplified this task. Interfaces and loss functions were all retained, as well as the core structure of model and of the training procedure.

In order to further simplify the long-term support of the code, a suite of unit and smoke tests was developed in parallel, for a total code coverage of 91%. In line with standard coding practices, a CD/CI pipeline was developed, which tests the source code on Linux every time new changes are pushed to the repository and creates detailed code coverage reports attached to any pull request.

4.2 Benchmarking

The newly implemented algorithms were tested against the legacy version in two experiments. The first experiment compares the speed of the refactored models with the legacy version, addressing training time and resources utilisation. A second experiment compares the accuracy of the new LFADS against the old version in a simple parameter recovery task.

4.2.1 Speed and Resources Utilisation

In this first experiment, the newly developed LFADS and TNDM are tested against the legacy code in order to measure the speed improvement and any changes in the utilization of computing resources. Experienced members of the research group report similar training time for the legacy version of LFADS on CPUs and GPUs, therefore both options will be tested here. In a second stage, TNDM will be tested against the new LFADS implementation using the best performing computing solution for LFADS.

The CPU-optimized machine used for this experiment is a c2-standard-8 virtual machine (VM) on Google Cloud Platform (GCP). It is equipped with 8 virtual-CPUs (vCPUs) adding up to a total 32 GB of memory. The operating system (OS) installed on the machine is Debian 10, equipped with an Intel Cascade Lake CPU platform. According to the provider, the machine can run at a sustained clock speed of 3.8GHz.

The GPU-optimized VM is a n1-standard-4 (4 vCPUs, 15 GB memory), mounting an NVIDIA Tesla P100 GPU (3584 CUDA Cores). The operating system in this case is Ubuntu 20.04 LTS Minimal and the CPU platform is Intel Haswell. According to

the GPU producer, the P100 model can reach 4.7 TeraFLOPS on double operations.

The dataset used for the test contains synthetic neural activity simulated through a Lorenz system and encoded into spiking events on 30 channels and behaviour observations on 4 channels (more on synthetic data in chapter 5). It contains a total of 50 trials, each long 100 timesteps in total. In all models the initial conditions dimension was set to 64 and the number of factors to 3.

Table 4.1 shows the results of this first benchmark analysis. As anticipated, the legacy version of LFADS performs comparably in the two platforms (1.23 it/s and 1.44 it/s), even though the compiling time is significantly smaller in the GPU-based machine – 1m 12s against 4m 14s. In both platforms, the model shows a worrying growth in the memory allocation, hinting at a possible memory leak. For this test, the execution was limited to 1000 iterations, but longer training efforts could lead to a memory explosion and a subsequent reduction in the training speed.

Type	Machine	Startup	Speed	Max CPU	Max mem.	Max GPU	Mem. growth
LFADS-old	CPU-opt	4m 14s	1.23 it/s	69.4 %	33.3 %	–	+9.70 % / Kit
LFADS-new	CPU-opt	3s	1.53 it/s	47.9 %	4.2 %	–	–
LFADS-old	GPU-opt	1m 12s	1.44 it/s	31.2 %	33.7 %	43 %	+2.12 % / Kit
LFADS-new	GPU-opt	2s	6.78 it/s	27.0 %	28.25 %	12 %	–
TNDM	GPU-opt	2s	3.73 it/s	27.6 %	28.3 %	10 %	–

Table 4.1: Runtime statistics for legacy and new versions of LFADS, as well as for the latest version of TNDM. In the table, "it" is used to abbreviate iterations (training steps) and "Kit" for 10^3 iterations.

Thanks to TensorFlow 2's Eager Execution, the new version terminates the setup in a fraction of the time in both platforms – 3s on average against 2m 43s on average. The faster startup time is combined with a +24% speed improvement at regime on CPU and a 4.7x improvement on GPU. The increased speed did not come at a cost in terms of CPU, GPU or memory utilisation, where the new implementation achieved an average reduction of 22%, 72% and 52%.

Given the superior performance of the refactored LFADS on GPU compared to CPU, we tested TNDM on the same machine, in order to asses the performance loss due to the more complex structure of TNDM. The speed reduced 45% to 3.73 it/s, while CPU, GPU and memory remained stable despite the increase in the network size. Not only did the refactoring operation simplify the codebase, but it also significantly reduced the required training time. Moreover, the lower utilisation of resources enables the parallel execution of multiple experiments on the same machine, further decreasing

the time requirements in batch experiments.

4.2.2 Accuracy

The second experiment carried out as part of the refactoring task explores the impact on the accuracy of the model. The old and new LFADS implementation were tested on a typical parameter recovery recovery task, where synthetic latent factors are encoded into neural activity and the algorithms must recover them as accurately as possible (further details in chapter 5).

Figure 4.1 illustrates the difference in performance between the legacy and the new implementation of LFADS. Performance is calculated as the percentage of total variance explained by the model, often referred to as R2. For medium to high firing rates the performance of the two algorithms is equivalent, while the new implementation performs significantly better when the baseline firing rate is lower and encodings are sparser.

We suspect this is due to the slightly different regularisation on the two versions: the legacy codebase applies the same L2 penalty across all layers, while the new version was tuned to enforce higher L2 penalties on the recurrent weights of the RNN, in order to generate smoother trajectories. AutoLFADS [41] already demonstrated how hyperparameter tuning plays a key role in determining the performance of LFADS on sparse encodings and we believe this might be the reason for the divergence.

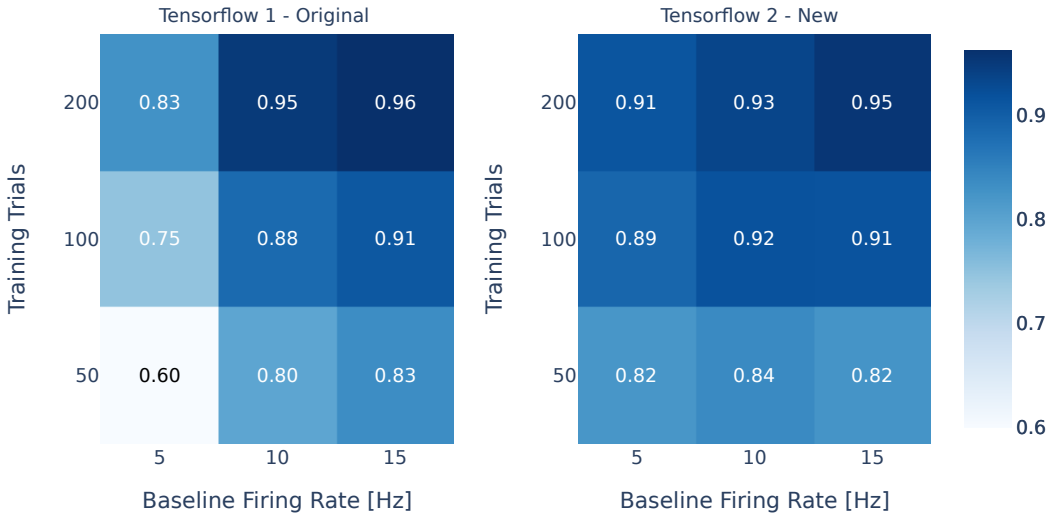


Figure 4.1: Each cell contains the average reconstruction performance (R2) on a 1000 trials test set, for 3 independent model runs.

Chapter 5

Synthetic Data

The novel implementation of LFADS and TNDM was the enabler for a set of experiments involving both synthetic and real-world data, where we explored the behaviour of the two in challenging learning conditions. Examples of challenging conditions are those where the algorithm has access to a limited number of training samples, where the signal-to-noise ratio in the population is low or where the information is encoded in few spikes, often referred to as *sparse encoding*.

For this purpose, we craft a synthetic dataset base on the Lorenz system [59]. We conjecture that TNDM will outperform LFADS in hostile conditions due to its ability to ingest both neural and behavioural data during training. The outperformance should be more significant when the behaviour is highly correlated with the neural latent factors, while high noise in behavioural data should result in less visible improvements of TNDM over LFADS.

5.1 Lorenz System Generator

The synthetic data employed throughout this chapter were generated through a Lorenz system generator. This system is commonly used for synthetic data due to its simple structure and the chaotic 3D butterfly attractor that it produces [1, 42]. Starting from an initial point (x_0, y_0, z_0), the Lorenz system updates its state according to the following dynamical system:

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= xy - \beta z \end{aligned} \tag{5.1}$$

where σ, ρ and β are three parameters, in this case equal to 10, 28 and 2.667.

For each trial, the generator samples a different initial point (x_0, y_0, z_0), where x_0, y_0, z_0 are independent and identically distributed following a uniform distribution between -10 and 10. The Lorenz system is then simulated using the Explicit Runge-Kutta method of order 5(4) [60] with a time interval $\Delta t = 10\text{ms}$ for a total trial length of 1s. For each trial, the three timeseries are then shuffled in order to increase the complexity of the recovery task, effectively resulting in 6 possible rotations of the latent space. The timeseries generated are combined in a matrix $\mathbf{U} \in \mathbb{R}^{R \times T \times 3}$, where R is the number of trials (or independent runs) and T is the total amount of timesteps, in our case $10\text{ms}/1000\text{ms} = 100$. After a standard mean-variance standardization, the matrix $\mathbf{U}$ is transformed in the latent factors $\mathbf{Z}$:

$$\mathbf{Z} = (\mathbf{U} - \bar{\mathbf{U}}) \odot \frac{1}{s_{\mathbf{U}}}, \quad \text{where } s_{\mathbf{U}} \text{ is the sample standard deviation of } \mathbf{U}. \tag{5.2}$$

Combining a high-variance initial sampling technique with the axis rotation resulted in a set of latent factors representing a highly heterogeneous set of possible trajectories (fig. 5.1).

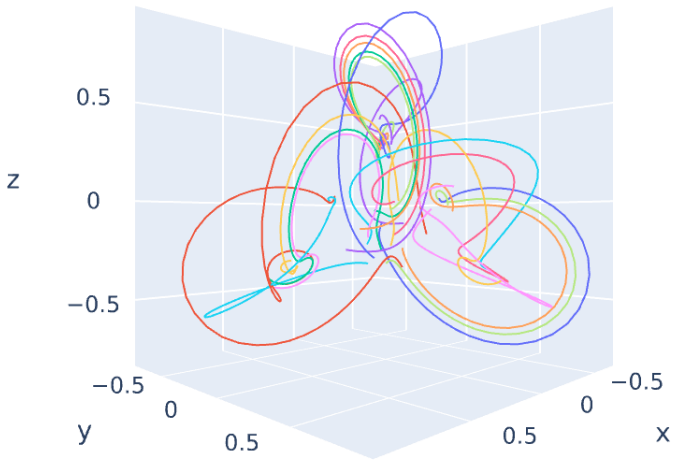


Figure 5.1: Latent factors trajectories for the first 20 trials.

For this chapter's experiment, all 3 latent factors are encoded into neural activity and only 2 of them were used in order to generate behaviour:

$$Z = Z_r + Z_i = 3, \quad Z_r = 2 \tag{5.3}$$

The factors are translated into neural activity on 30 neurons through a linear weights matrix $\mathbf{W}_N \in \mathbb{N}_+^{3 \times N}$, sampled using a Uniform$(1,2)$ distribution for the absolute value

of the weights and random signs with equal probability. Using such matrix, we compute the log-firing rates as:

$$\mathbf{F} = \mathbf{Z} \times \mathbf{W}_N + \log f_b \,, \quad \text{where } f_b \text{ is the baseline firing rate.} \tag{5.4}$$

Spikes are then sampled using a Poisson process, after being scaled by the time interval Δt:

$$s_{ijk} \sim \text{Poisson}(\exp(f_{ijk})\Delta t) \tag{5.5}$$

where f_{ijk} and s_{ijk} are the elements of $\mathbf{F}$ and $\mathbf{S}$. A typical example of the neural activity is shown in fig. 5.2. Increasing the baseline firing rate has a scaling effect on the entire activity of the population, allowing the signal to emerge from the background noise. A better signal-to-noise ratio improves the latent factors reconstruction accuracy for most algorithms, as the information is encoded in a large number of redundant events.

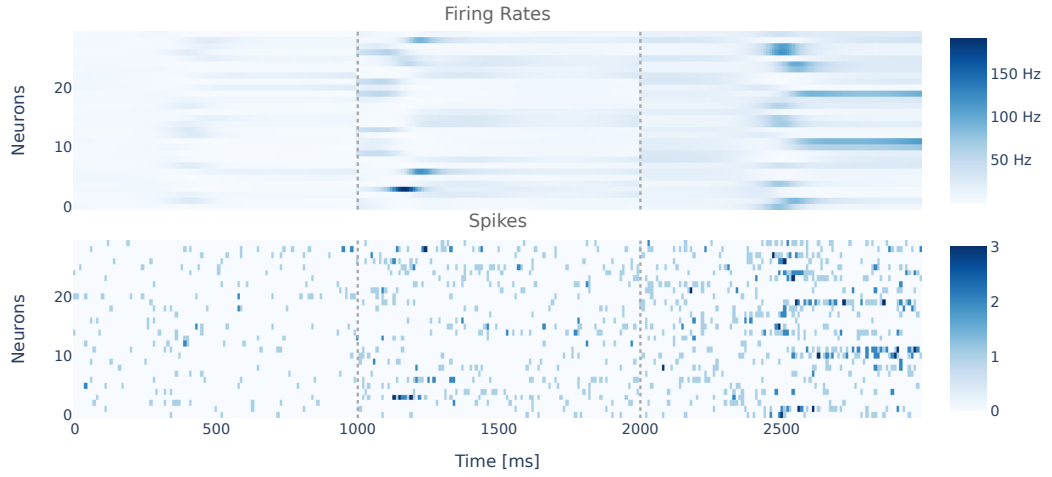


Figure 5.2: Encoded neural activity of the Lorenz system in 3 independent trials sorted by sparseness. On the left-hand side, the baseline firing rate was set to 5Hz, in the middle to 10Hz and on the right-hand side to 15Hz

In order to evaluate TNDM's ability to leverage behavioural information in order to disentangle behaviour relevant variables and improve reconstruction performance, we designed a second linear transformation $\mathbf{W}_B \in \mathbb{R}^{Z_r \times B}$, where single weights are sampled from a Gaussian distribution with zero mean and variance of 5. Only the behaviour-relevant latent factors Z_r were used in the construction of the four (B) behavioural timeseries. Excluding one latent factor from the encoding will turn out useful in the experiments in order to evaluate the ability of TNDM to partition the latent space correctly. Each element of the behavioural timeseries $\mathbf{B} \in \mathbb{R}^{R \times T \times B}$ was calculated as:

$$b_{ijk} \sim \mathcal{N}(\mathbf{Z}^{[:,:,1:Z_r]} \times \mathbf{W}_B, \sigma_B^2) \tag{5.6}$$

where σ_B is the behavioural noise, ranging from 0.5 to 2 depending on the specific experiment. Figure 5.3 illustrates the behavioural variables for three trials with increasing behavioural noise σ_B.

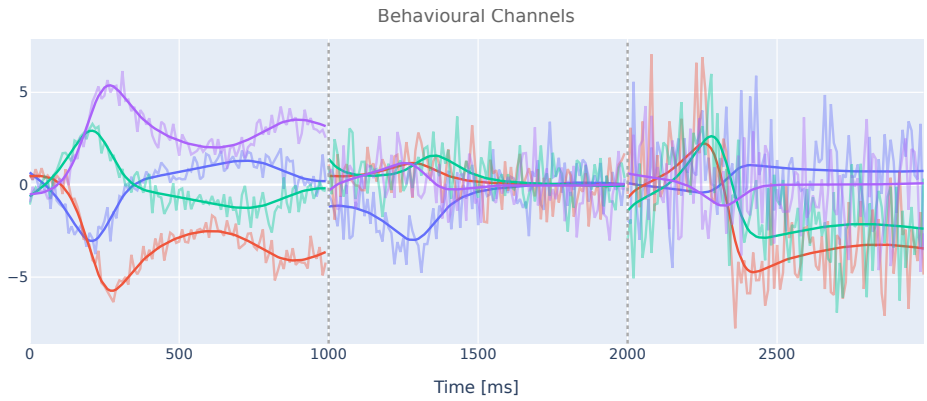


Figure 5.3: Encoded behavioural activity of the Lorenz system in 3 independent trials sorted by behavioural noise $(0.5, 1, 2)$. Heavier lines represent the noiseless behavioural variables, before adding the noise term. The more transparent lines represent the actual behavioural variables.

5.2 Experiment Setup

A single experiment was designed to answer three separate research questions:

- Is TNDM outperforming LFADS on latent factors recovery?
- To what extent is TNDM's performance affected by the noise in the behaviour?
- Can TNDM disentangle behaviour-relevant variables?

In order to answer these questions, we designed a $3 \times 3 \times 3$ grid of experimental settings, spanning across different training sample sizes (50, 100, 200), baseline firing rates (5Hz, 10Hz, 15Hz) and behaviour noises (0.5, 1, 2). Each setting was associated with different $\mathbf{W}_N$ and $\mathbf{W}_B$, simulating a total of 27 independent conditions.

An architecture was chosen for LFADS based on the hyperparameters provided in the original implementation and a brief manual fine-tuning, to adapt the parameters to the new implementation. The same parameters were applied to TNDM, with the exception of latent factors, modelled jointly in LFADS and here separated between the relevant and irrelevant subspace ($Z_i = 1, Z_r = 2$). Appendix B.1 contains the full set of

hyperparameters used for the two models. The experiments were run on GCP, using the GPU-optimized machine mentioned in section 4.2.1.

5.3 Results

Both LFADS and TNDM displayed some degrees of learning across all the experiments in the grid. Throughout this analysis, we will use R2 as the prime metric of accuracy in retrieving the latent factors. The average R2 values were 0.76 and 0.81 for LFADS and TNDM respectively, with lows of 0.43 and 0.60, and highs of 0.94 on both. Figure C.1 displays the encoded and reconstructed latent factors for some typical scenarios.

The experiments show that both LFADS and TNDM were able to reconstruct latent variables reliably when they were trained with more than 50 samples. Both algorithms reached their top performance when trained on with the maximum number of training samples and on the least sparse dataset (fig. 5.4). TNDM significantly outperformed LFADS in setting with small training samples, leveraging on its dual backpropagation flow to extract useful information from both the neural and the behavioural activity. On average, TNDM was able to explain 16% more of the total variance compared to LFADS on the 50 trials training sample, generating latent variables that more closely tracked the encoded dynamics.

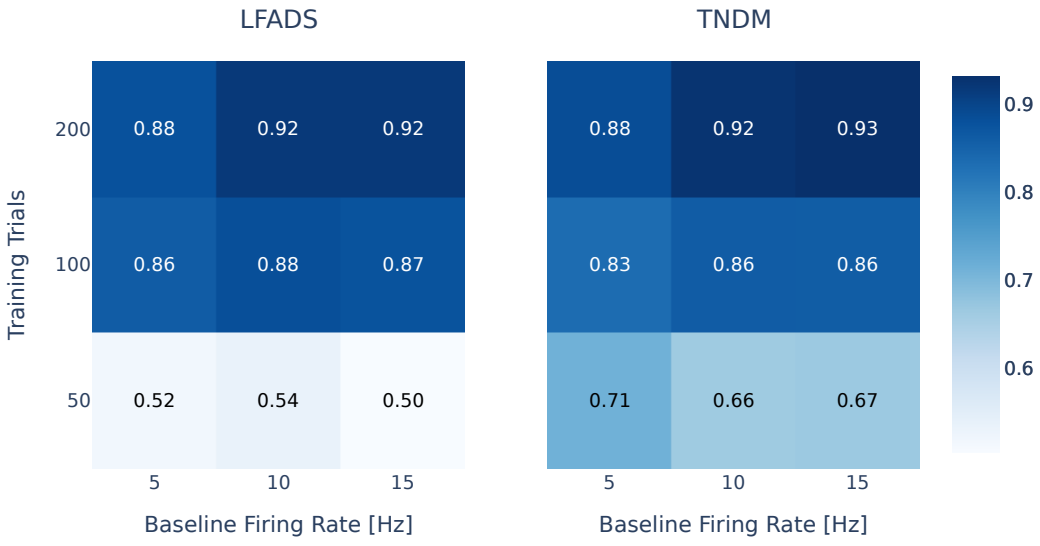


Figure 5.4: Reconstruction accuracy (R2) for LFADS and TNDM as a function of the training trials and the baseline firing rate.

On an aggregated level, both LFADS and TNDM displayed a good resistance to low firing rates, with the average R2 decreasing of just 0.01 from the maximum baseline firing rate (15Hz) to the minimum one (5Hz), despite a large difference in the resulting activity, as shown in fig. 5.2. A similar experiment with more extreme changes in the baseline firing rates or with more independent runs per setup could unveil some minor impacts that we were not able to observe.

In contrast with our initial assumption, the noise in the behaviour did not impact the learning outcome for TNDM (fig. 5.5). This result seems counter-intuitive and we believe it could be due to the fact that the behavioural weight in the loss function was significantly lower than the neural reconstruction weight, shifting the attention away from behavioural performance. We conjecture that a higher behavioural weight could have resulted in some overfitting in the high-noise datasets, where the model has high risk of picking up spurious relationships between latent and the noisy behaviours, and a more precise reconstruction in the low-noise datasets. Further work should explore this relationship in more detail, possibly leveraging on the automated hyperparameters tuning developed for AutoLFADS [41].

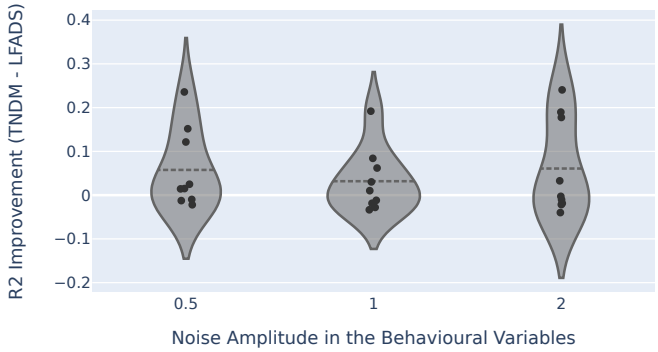


Figure 5.5: Difference in reconstruction accuracy (R2) between LFADS and TNDM as a function of the behavioural noise.

The last research question we posed involves the ability of TNDM to isolate behaviour-relevant information in the appropriate side of its architecture. In order to verify to what extent TNDM managed to isolate the behaviour-relevant latent variables, we calculate for each experiment a confusion matrix of how each partition of the recovered factors can predict each partition of the actual factors. Results are summarized in Figure 5.6. It is possible to observe how the matrix in the diagonal contain the higher accuracy, meaning that factors were disentangled on average correctly. As for the

general accuracy, disentanglement seems to be working more successfully for large training datasets.

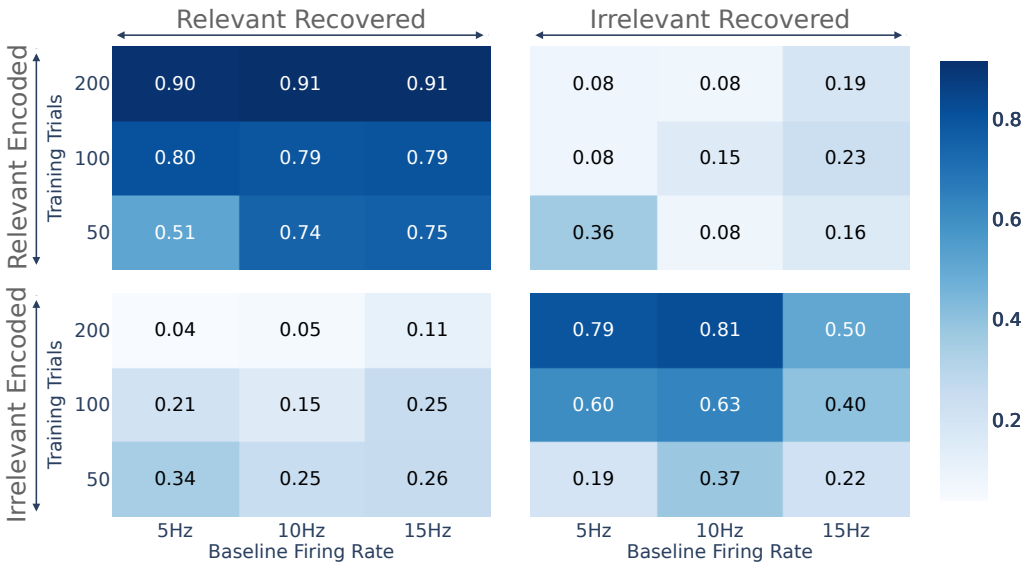


Figure 5.6: Accuracy in predicting one of the two sets of embedded factors from any of the two sets of recovered factors.

The experiments proposed in this chapter analysed the two models' ability to recover parameters embedded in neural and behavioural activity. The ability of TNDM to consume behavioural and neural information jointly resulted in a significantly better generalisation for small training dataset, but did not show any advantage for sparse encoding, even when using extremely informative behavioural variables. We hypothesize this is due to the small weight associated to behaviour likelihood in the loss function and that a larger weight could further improve TNDM's performance when behavioural timeseries are clean. Finally, we demonstrated TNDM's ability to disentangle behaviour-relevant latent factors from irrelevant ones, an extremely useful property for prosthetic applications and BCI in general.

Chapter 6

Primate Motor Cortex Data

6.1 Dataset

The previous chapter demonstrated how both LFADS and TNDM can recover latent variables when applied on synthetic data. The two models will now be applied to a real-world dataset containing neural and behavioural recordings from a rhesus monkey (*Macaca mulatta*, 10.5 kg) [46], in order to evaluate their performances when applied on the field. In addition to the neural activity, the dataset analysed contains both electrical activity from the monkey's wrist muscles (electromyography, EMG) and the forces applied by the monkey on a wrist-operated manipulandum. TNDM's ability to extract behavioural-relevant latent variables will be tested on each of the two behavioural datasets, measuring the accuracy in predicting the behavioural variables while reconstructing accurately the neural activity.

The task involves the monkey sitting in rest position with the forearm restrained, holding with one hand the manipulandum. A computer screen displays the forces applied by the monkey on the manipulandum, which does not move. A trial begins when no forces are applied and the cursor stays in the middle of the screen centre for at least 500ms. A second target is then showed at one of 8 directions equidistant from the centre, positioned with radial symmetry each $\frac{\pi}{4}$. The monkey is rewarded when the cursor reaches the target and is maintained within the target for at least 500ms, with a maximum trial length of 5s. The experimental session used throughout this chapter lasts 25 minutes in total and contains 435 trials with a success rate just short of 85%. The maximum time to reach the second target once shown is 3.5s, while the minimum time is as low as 0.74s. For each trial, we trim the timeseries from the moment the second target is displayed until the minimum trial length across the experimental session.

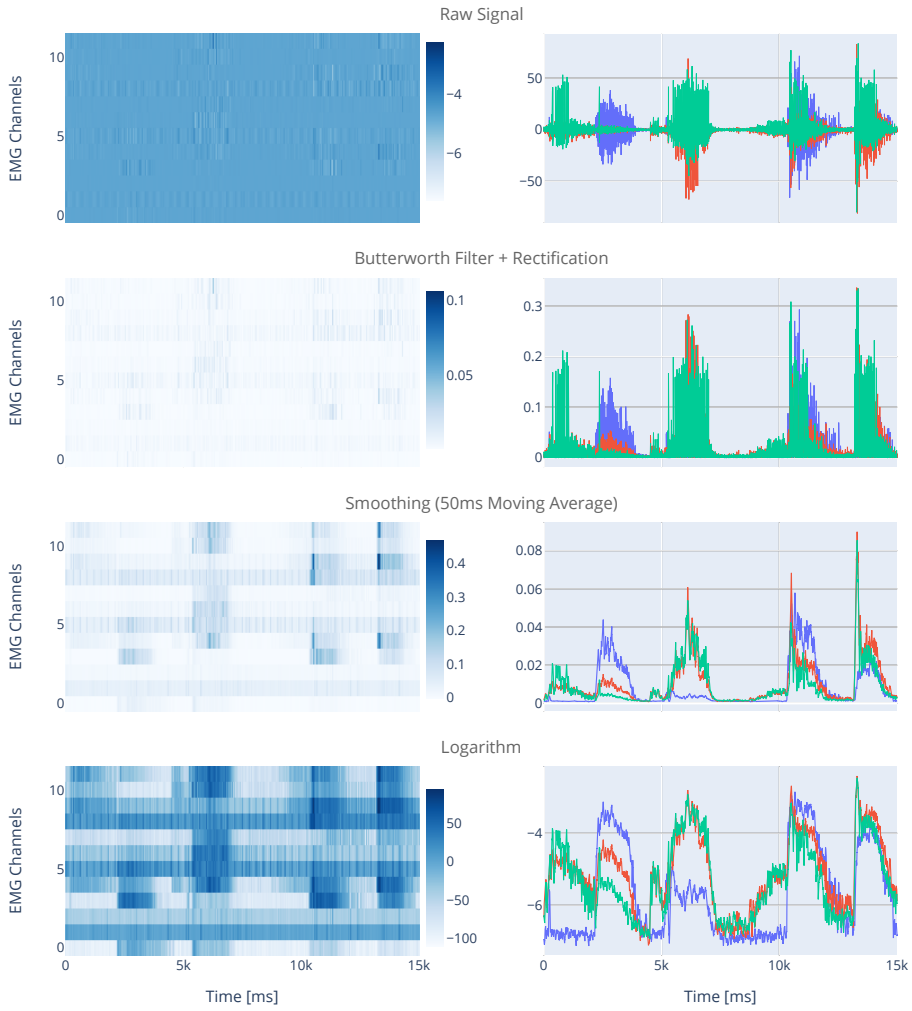


Figure 6.1: On the left-hand side, EMG activity on all 12 channels recorded. On the right-hand side, EMG activity on a subset of channels (4, 5, 12).

Unsuccessful trials or trials with equipment failures are removed, resulting in a total number of trials $R = 412$. The experiments were approved by the Institutional Animal Care and Use Committee of Northwestern University, in line with the Guide for the Care and Use of Laboratory Animals.

A 96-channel microelectrode array from Blackrock Microsystems (Salt Lake City, UT) is connected to the primary motor cortex (M1), extracting neural activity time-series with a 1ms precision after spike-sorting with Offline Sorter (Plexon, Dallas, TX). In the experimental session analysed, 92 of the 96 channels were recorded successfully ($N = 92$). The forces applied by the monkey are measured from the manipulandum itself and result in two behavioural variables ($B_{\text{forces}} = 2$): the force on the

flexion/extension of the hand and the force on urnal/radial rotations of the hand. All forces were sampled with a 1000Hz frequency.

A total of 24 pairs of intramuscular electrodes was implanted in a second procedure, monitoring among others four wrist muscles involved in the interaction with the manipulandum. Only 12 of the 24 total EMG channels were recorded in the experimental session analysed here ($B_{\text{emg}} = 2$). At recording time, the electromyography signal from these electrodes was amplified (x500), bandpass filtered between 80Hz and 500Hz and sampled at 2000Hz. EMG timeseries were then cleaned offline using a bandpass Butterworth filter of order 3 between 30Hz and 350Hz, followed by a rectification (absolute value) and some smoothing (moving average with 50ms window size). The right-skewed distribution of the resulting timeseries was then mitigated by applying the logarithm and subsampled at 1000Hz, in order to match the neural activity. Figure 6.1 illustrates the effect of the post-processing of the EMG data. The resulting timeseries for neural data, forces and EMG were further reduced in granularity by applying a binning with $\Delta t = 10$ms. For neural data, spikes within each bin were summed, while for the two behavioural datasets the values were summarised through their mean.

6.2 Decoding Forces

In this first experiment, we explore the relationship between the activity of the primary motor cortex (M1) and the forces imposed on the manipulandum by the monkey. Forces are a result of the contraction of the wrist muscles, which in turn are controlled by the neural activity in M1. Several hypothesis have been tested regarding the relationship between these two variables. Some linear relationships between the activity of single motor neurons and wrist dynamics were discovered as early as in 1968 by the pioneering work of Evarts [61]. Humphrey, Schmidt, and Thompson [62] noticed for the first time that the inertial load could significantly alter the linear coefficients, while two working groups including the same Evarts noticed weak finely graded movements were associated with unexpectedly high neural activations [63].

In contrast with the experiments in chapter 5, in this section we will replace the *synchronous* decoder for TNDM with the *causal* decoder described in eq. (3.10). This corresponds to the assumption that behaviour at time t can be expressed as a linear combination of the latent factors at times $t-k$, with $k \geq 0$.

We will investigates whether TNDM can outperform LFADS in extracting latent

variables that represent accurately neural activity and how the two models evolve as a function of the number of factors allowed. TNDM's ability to disentangle behaviour-relevant and irrelevant factors will be analysed qualitatively and quantitatively, as well as its accuracy in inferring behaviour. A simple dimensionality reduction on the information encoded as initial conditions in the bottleneck will unveil a circular latent representation, aligned with the target directions in the reach task.

6.2.1 Experiment Setup

The dataset was randomly shuffled and split using a 5-fold cross-validation framework into 5 folds, each with 64% of samples dedicated to training, 16% for validation and the rest for testing purposes. A default implementation of TNDM and LFADS involving 4 factors was applied to all 5 folds in order to obtain an estimation of the performance distribution. TNDM and LFADS models of varying capacities – from 2 to 5 total factors – were applied to the first folds to observe the relationship between the capacity of the model and its performance. The hyperparameters template used for both LFADS and TNDM is available in appendix B.2.

6.2.2 Results

LFADS and TNDM performed equally as good across the 5 folds in terms of neural reconstruction, achieving a neural negative log-likelihood of 1379.7 and 1379.9 respectively (p-value $= 0.98$, no significant difference). Figure 6.2 displays the inferred firing rates from TNDM, comparing them with actual neural activity. The reconstructed firing rates appears smoother and less variables than the neural activity, which could hint at a high amount of noise in the population activity or at an excessive regularisation in the generator RNN. In the previous chapter, the regularization of the weights in the generator RNN could be tuned such that the smoothness of the reconstructed dynamics mimicked that of the encoded latent factors, which were known. Evaluating how smooth latent variables should ideally be in this case is much more complex. On one hand, having firing rates that correctly track single spikes would result in a black-box solution that is too hard to decipher. At the same time, observing excessively smooth latent factors could mean that the model is not reactive enough, or that it fails to capture faster dynamics. The ideal reactiveness could be ideally an external parameter, determined for instance as a function of the downstream processes, which consume the latent factors.

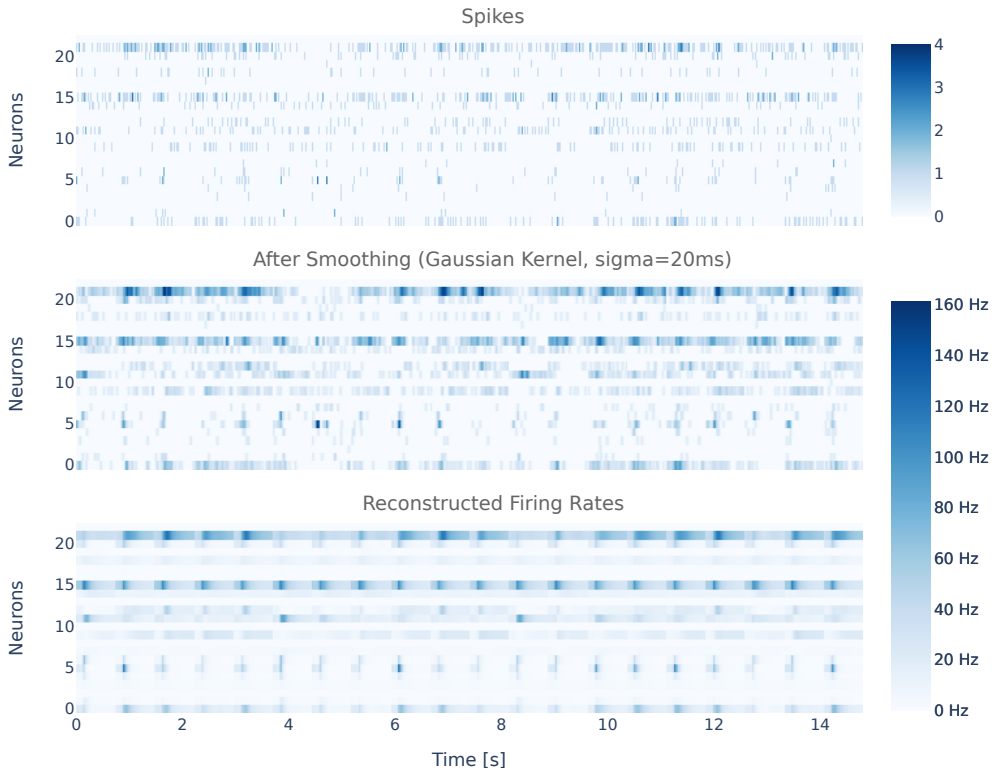


Figure 6.2: Neural activity, sample firing rates and estimated firing rates for a sumb-sample of neurons (1:4) and trials (the first 20) using TNDM with 2 relevant and 2 irrelevant factors.

Figure 6.3 shows the evolution on the performance for the two algorithms when the number of factors changes. In the interval explored, neural log-likelihood improves linearly with the addition of new factors on both models, regardless the split between relevant and irrelevant factors. LFADS appears to be performing marginally better than TNDM, particularly for 5 factors. This could be due to a lack of compatibility between the modelling of neural activity and forces, resulting in a trade-off between neural likelihood and behaviour likelihood without any significant synergy.

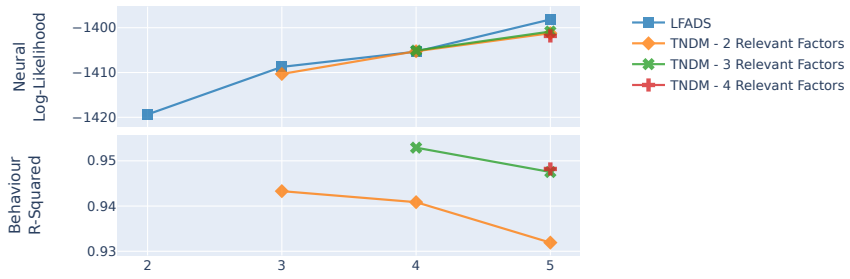
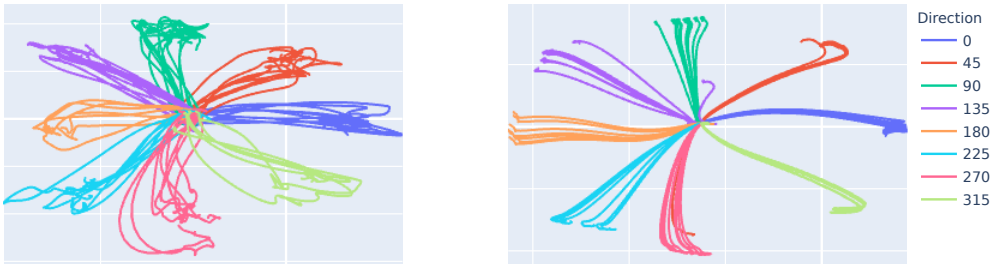


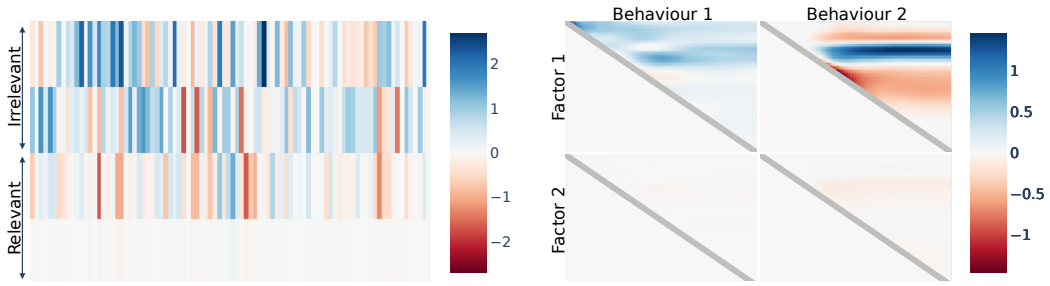
Figure 6.3: Reconstruction performance for LFADS and TNDM for varying factors.



(a) Original behaviour. (b) Reconstructed behaviour.

Figure 6.4: Original and reconstructed behaviour using TNDM with 2 relevant and 2 irrelevant factors. Directions are expressed in degrees.

While we were not able to observe any improvement of TNDM over LFADS in neural reconstruction, TNDM managed to obtain a staggering 0.95 average R2 performance on behaviour reconstruction, with very low variability across the five folds ($s_{R2} \leq 0.01$). Increasing the number of relevant factors from 2 to 3 resulted on average in a 0.01 improvement on the R2, while no improvement was detected from 3 to 4. Figure 6.3 highlights how TNDM models with 5 factors underperformed model with 4 factors on the test set, despite a similar performance on the training set, where the average R2 was 0.96 in both scenarios. This might be an indication of overfit, which could be addressed by increasing the regularization or the dropout on higher-capacity models. Figure 6.4 shows the difference between the original and the reconstructed behaviour as a function of the target angle. The low trial-to-trial variability in the reconstructed behaviour for a given direction seems to suggest that this variability is not encoded clearly in the neural activity and it might derive from the interaction between the monkey and the manipulandum.



(a) Neural Weights. (b) Causal Behaviour Weights.

Figure 6.5: Weights mapping the latent factors to the neural and behavioural activity.

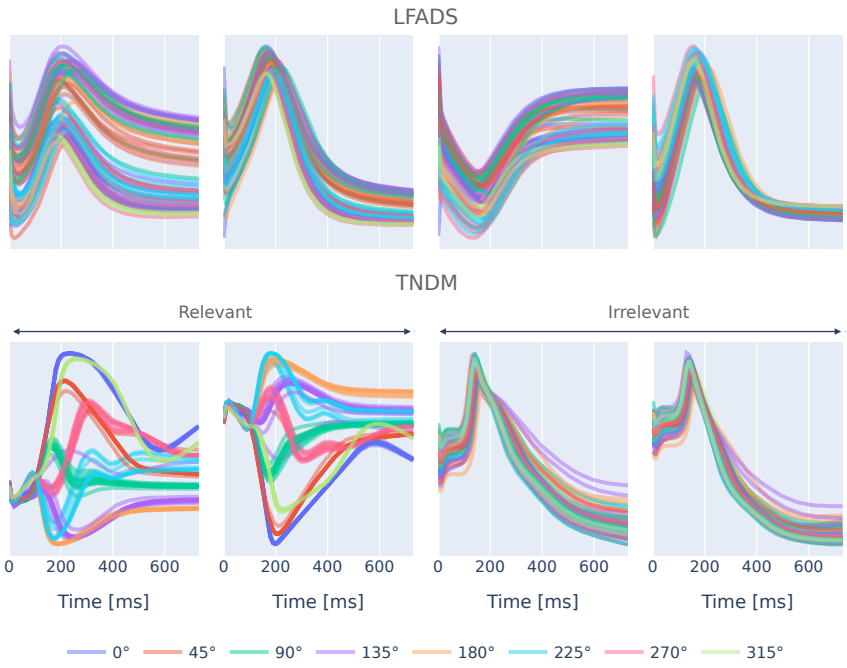


Figure 6.6: Latent variables for LFADS and TNDM.

A useful source of information to understand the behaviour of TNDM are the weights mapping the latent factors to the neural and behavioural activity (fig. 6.5). The network seems to be partitioning the relevant factors subspace neatly in two: the first factor is solely used to reconstruct behaviour, while the second is fully dedicated to neural activity. Considering that the performance of TNDM was equivalent to LFADS, we can comfortably say that TNDM managed to encode the neural activity more efficiently, using only 3 factors to achieve the same precision as a four factors LFADS. Despite this positive remark, there does not seem to be any interesting interaction between the two relevant factors. This observation, combined with the lack of outperformance against LFADS and the suspicious behavioural reconstruction, seems to suggest that wrist forces are not linearly encoded into neural activity.

So far, TNDM was able to demonstrate an improvement over LFADS in encoding efficiency of the encoding and a promising behaviour reconstruction performance. An area where TNDM excels despite the lack of linear relationships between M1 activity and the forces is the disentanglement of behaviour-relevant and behaviour-irrelevant activity. In fig. 6.6, the different trajectories for the eight behavioural directions are clearly identified in both behaviour-relevant variables, despite the minor impact the second one has on behaviour reconstruction. The connection between

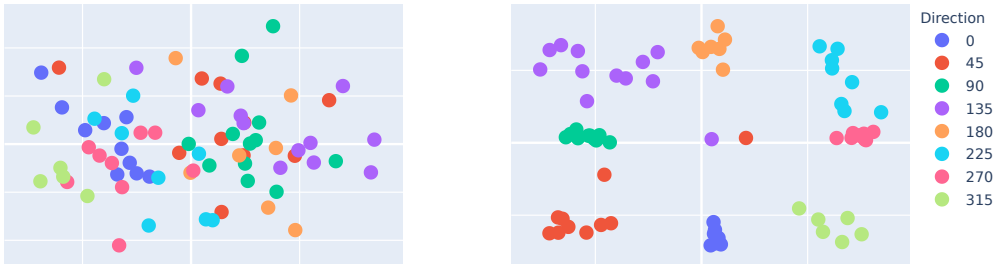


Figure 6.7: First two PCA components for LFADS initial conditions (left-hand side) and TNDM behaviour-relevant initial conditions (right-hand side). Direction is expressed in degrees.

behaviour-relevant variables and the target angles is even more evident by projecting the behaviour-relevant initial conditions into a low-dimensional with PCA (fig. 6.7). While in LFADS no patterns are detectable, TNDM organises the initial conditions according to the target angle, forming eight clusters where encoded angles increase proceeding clockwise.

6.3 Decoding Electromyography

The previous section illustrated how wrist forces do not appear to be linearly encoded into neural activity. The remaining two experiments of this chapter explore the relationship between M1 and EMG signals, a problem of major importance for BCI applications.

6.3.1 Experiment Setup

The first experiment follows the structure of section 6.2.1. The dataset containing neural and EMG data is re-organised into five derived datasets, containing different partitions of the same trials, following standard cross-validation practices. The capacity of TNDM was increased from 2 behaviour-relevant latent factors to 4, in order to allow for a better modelling of the high dimensional EMG behaviour. LFADS total factors were consequently increased from 4 to 6 to ensure a fair comparison – all the other hyperparameters are reported in table B.3. The goal is to evaluate the difference in neural reconstruction performance between TNDM and LFADS, as well as the precision of TNDM to infer behavioural dynamics.

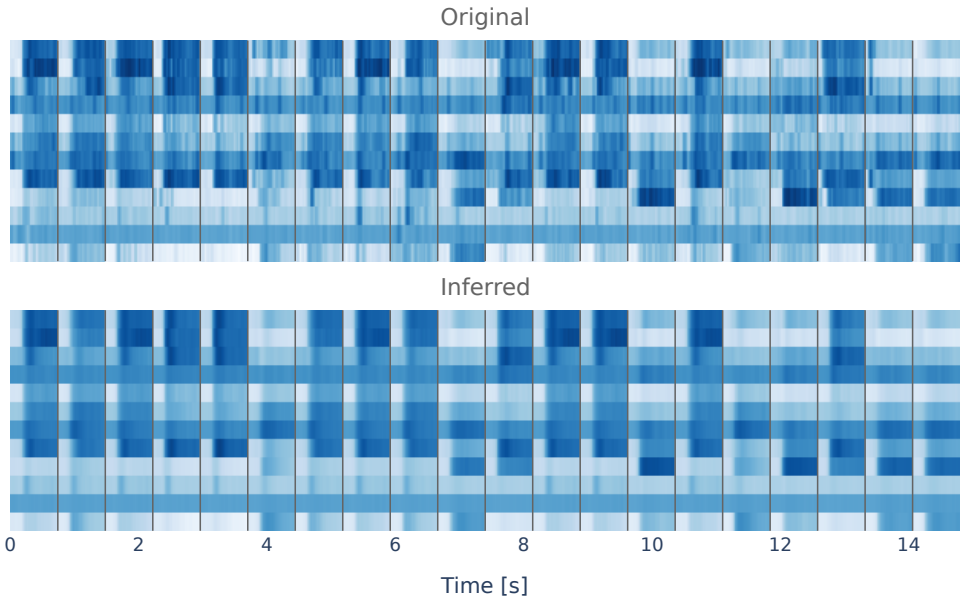


Figure 6.8: Original and reconstructed EMG behaviour using TNDM. First 20 trials shown.

The experiment described above tests the ability of LFADS and TNDM to generalise to unseen trials drawn from the same distribution as the training samples. In the second experiment, we stress-test the two models and evaluate their ability to generalise to trials drawn from a different distribution than the training set. In order to do so, we train both LFADS and TNDM on 7 target directions and test their neural reconstruction and behaviour accuracy on the unseen direction.

6.3.2 Results

In the first experiment, TNDM marginally underperforms LFADS, obtaining an average negative log-likelihood of 1371.8 against 1369.7 (p-value $= 0.70$, no significant difference). At the same time, the semi-supervised nature of TNDM enables the reconstruction of both behaviour and neural activity, with an average R2 of 0.82. From the perspective of a prosthetic application, this is a remarkable finding. By monitoring 92 neurons we were able to extract 4 behaviour-relevant latent variables, that alone could be used to control the torques and forces in a synthetic wrist junction (fig. 6.8).

Figure 6.9 highlights the ability of the two models to disentangle the target directions in the target variables, even though this information was never provided during the training of the algorithm. By solely processing the neural activity and the EMG data, TNDM and LFADS were able to recognise eight typical clusters of activity and

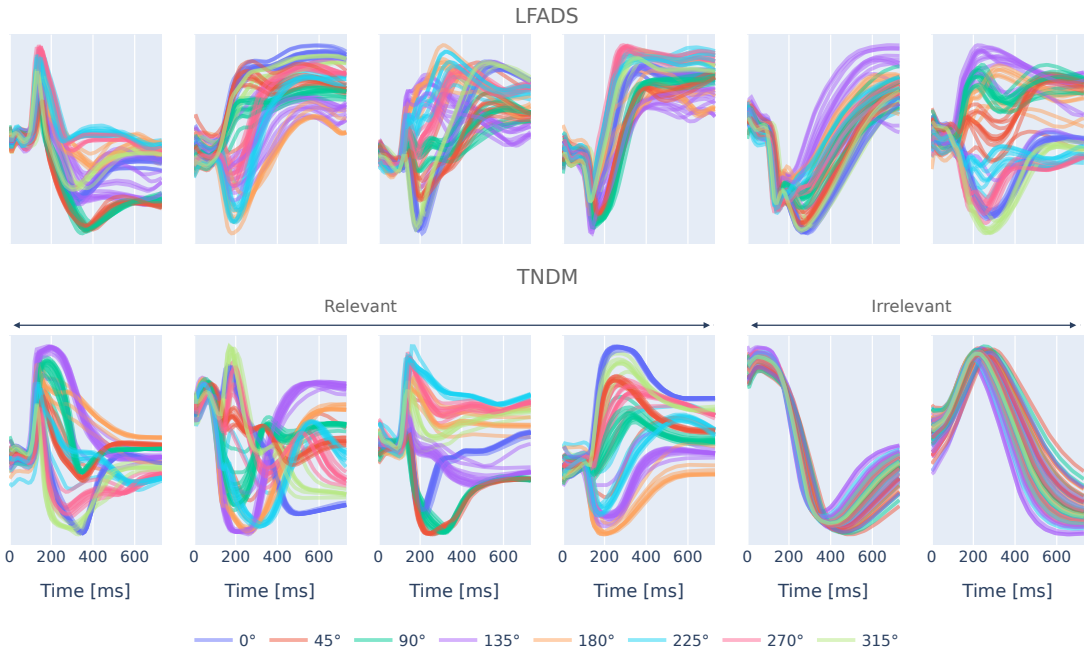


Figure 6.9: Latent variables for LFADS and TNDM.

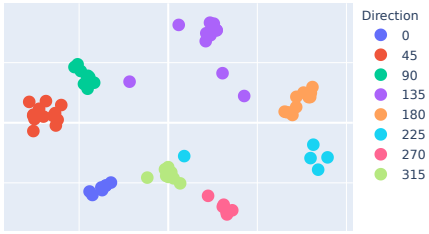


Figure 6.10: First two PCA components for TNDM behaviour-relevant initial conditions.

to represent them as different trajectories. This behaviour is observable in the latent variables of both algorithms, although much more visible in TNDM, where behaviour-relevant and behaviour-irrelevant information is clearly demixed. In contrast with what observed in the previous experiment for forces, this time the weights from factors to neural activity and behaviour did not show any partitioning of the behaviour-relevant subspace (fig. C.2 and fig. C.3 in appendix C).

TNDM's ability to capture behaviour-relevant information is clear if we analyse the representation of the encoded initial conditions. In fig. 6.10, TNDM represent behaviour-relevant information in the neural activity in the same circular structure observed in fig. 6.7. The main difference between the two experiments is that here the network was never shown the notion of the target direction, not even during training.

While the target direction is encoded in the terminal point of the forces trajectories, provided as target during training in section 6.2, the direction is not encoded explicitly in EMG trajectories, which were used in this experiment. Despite the absence of such information in the behaviour, TNDM recognise this circular manifold as the most information-efficient way to describe the behaviour-related latent variable through the bottleneck.

In the second experiment, we test the ability of the two algorithms to generalise to data drawn from a different distribution then the training set. LFADS and TNDM continue to perform similarly from a neural reconstruction point of view, scoring 1367.4 and 1374.4 in the average log-likelihood (p-value $= 0.40$, no significant difference). TNDM improves the behaviour reconstruction performance from 0.82 in the previous experiment to 0.85 now, showing robustness to unseen behaviours and good generalisation. The encoding of the test trials compared to the encoding of training samples in the initial conditions space confirms the ability of the algorithm to generalised to unseen angles (Figure 6.11).

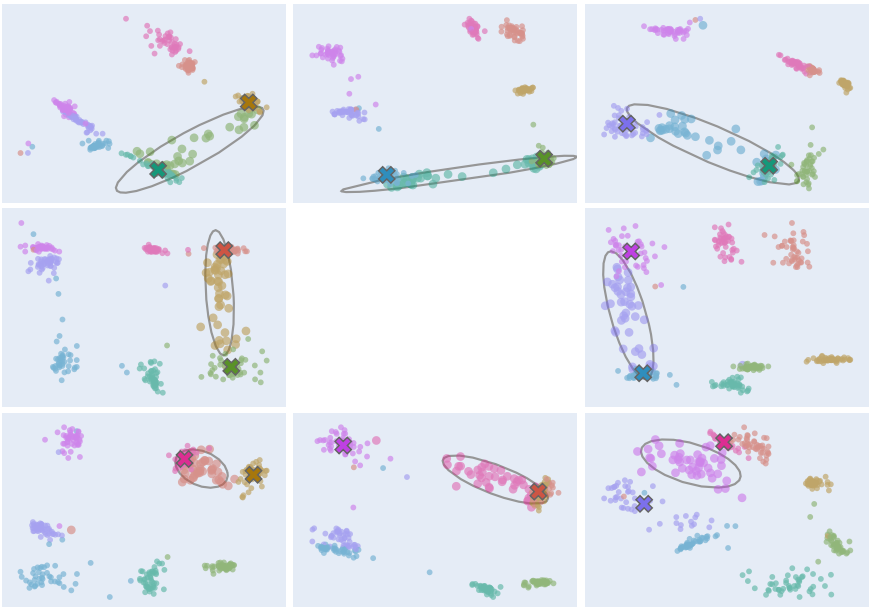


Figure 6.11: First two PCA components for TNDM behaviour-relevant initial conditions on leave-one-out experiment with target angles. The crosses represent the center of mass for the adjacent angles. The circle represents the a multivariate Gaussian distribution fitted to the data.

Chapter 7

Conclusions and Further Work

The extraction of latent variables from neural activity is an active research area, which is rapidly developing thanks to advancements in recording tools and the parallel progress in machine learning and pattern recognition. The main contributions of this project to the field can be divided in three areas.

In chapter 4, we provide an updated implementation for the current state-of-the-art algorithm. Startup time is reduced from minutes to seconds, the processing of batches was sped up of a factor of five, cleaner interfaces were developed for the end user and the codebase was more than halved in size, reducing maintenance costs.

The second major contribution involved the systematic testing of TNDM against LFADS. For the first time TNDM was validated on a synthetic dataset against a state-of-the-art algorithm. Results displayed a significant improvement in performances on small batch sizes and convergence in the neural reconstruction performance for favourable conditions. No clear trends were detected for varying behavioural noises and/or baseline firing rates, in contrast with our initial assumptions. The study should be repeated using wider ranges for these two parameters. In addition to the performance comparison, TNDM's disentanglement of behaviour-relevant from behaviour-irrelevant factors was observed and quantified.

In chapter 6, the two algorithms were applied to real-world data from the primary motor cortex of a primate. TNDM's ability to decode signals from non-linear high-dimensional data – such as electromyography – was tested for the first time and results were successful. TNDM and LFADS models with the same capacity produced similar neural reconstruction performances. However, TNDM proved able to encode at the same time both behaviour and neural activity, reconstructing forces with an R-squared of 0.95 and EMG signals with a 0.82 R-squared. In the last experiment, TNDM was

shown to decode successfully also from behaviours observed from the first time, thanks its ability to encode information in a manifold aligned with the target directions during the task.

From a neural reconstruction perspective, TNDM proved to perform better than LFADS in experimental setups with few trials and to perform in line with LFADS in the remaining setups. Its ability to decode both neural activity and behaviour at the same time, the creation of meaningful partitions between latent factor sub-spaces and its tendency to cluster latent trajectories associated with similar behaviours are the core features that this model offers beyond LFADS.

We identified two main areas of research connected with TNDM: *discrete behaviours* and *online decoding*. The current model is limited by the fact that it decodes continuous timeseries. Integrating a discrete behavioural decoder would enable the study of a new set of tasks, ranging from decision making to typing on a virtual keyboard. Online decoding, on the other hand, is a key problem that needs to be solved if we want to be able to predict the future behaviour at any point in time, and not only upon trial end. Solving the online learning problem would enable the application of TNDM to brain computer interface, with substantial performance improvements compared to all other existing algorithms.

Bibliography

[1] Chethan Pandarinath et al. *Inferring single-trial neural population dynamics using sequential auto-encoders*. June 2017. DOI: 10.1101/152884.

[2] Cole Hurwitz et al. "Targeted Neural Dynamical Modeling". In: *Advances in Neural Information Processing Systems 34*. Unpublished, submitted.

[3] M. Kamp et al. "Traumatic Brain Injuries in the Ancient Egypt: Insights from the Edwin Smith Papyrus". In: *Journal of Neurological Surgery Part A: Central European Neurosurgery* 73.04 (Aug. 2012). ISSN: 2193-6315. DOI: 10.1055/s-0032-1313635.

[4] Francisco López-Muñoz, Jesús Boya, and Cecilio Alamo. "Neuron theory, the cornerstone of neuroscience, on the centenary of the Nobel Prize award to Santiago Ramón y Cajal". In: *Brain Research Bulletin* 70.4-6 (Oct. 2006). ISSN: 03619230. DOI: 10.1016/j.brainresbull.2006.07.010.

[5] Ryan C. Williamson et al. "Bridging large-scale neuronal recordings and large-scale network models using dimensionality reduction". In: *Current Opinion in Neurobiology* 55 (Apr. 2019). ISSN: 09594388. DOI: 10.1016/j.conb.2018.12.009.

[6] Elon Musk. "An Integrated Brain-Machine Interface Platform With Thousands of Channels". In: *Journal of Medical Internet Research* 21.10 (Oct. 2019). ISSN: 1438-8871. DOI: 10.2196/16194.

[7] Cole Hurwitz et al. "Building population models for large-scale neural recordings: opportunities and pitfalls". In: *arXiv* (Feb. 2021). URL: http://arxiv.org/abs/2102.01807.

[8] Marius Pachitariu et al. "Suite2p: beyond 10,000 neurons with standard two-photon microscopy". In: *bioRxiv* (2016). DOI: 10.1101/061507.

[9] Vito Paolo Pastore et al. “Identification of excitatory-inhibitory links and network topology in large-scale neuronal assemblies from multi-electrode recordings”. In: *PLoS Computational Biology* 14.8 (2018). ISSN: 15537358. DOI: 10.1371/journal.pcbi.1006381.

[10] Gustavo Rios et al. “Nanofabricated Neural Probes for Dense 3-D Recordings of Brain Activity”. In: *Nano Letters* 16.11 (2016). ISSN: 15306992. DOI: 10.1021/acs.nanolett.6b02673.

[11] D. H. Hubel and T. N. Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. In: *The Journal of Physiology* 160.1 (1962). ISSN: 14697793. DOI: 10.1113/jphysiol.1962.sp006837.

[12] E. D. Adrian and Yngve Zotterman. “The impulses produced by sensory nerve-endings: Part II. The response of a Single End-Organ”. In: *The Journal of Physiology* 61.2 (1926). ISSN: 14697793. DOI: 10.1113/jphysiol.1926.sp002281.

[13] Thomas P. Trappenberg. *Fundamentals of Computational Neuroscience*. Oxford University Press UK, 2002.

[14] R. Quian Quiroga et al. “Invariant visual representation by single neurons in the human brain”. In: *Nature* 435.7045 (2005). ISSN: 00280836. DOI: 10.1038/nature03687.

[15] Horace Barlow. “Redundancy reduction revisited”. In: *Network: Computation in Neural Systems* 12.3 (2001). ISSN: 0954898X. DOI: 10.1088/0954-898X/12/3/301.

[16] “NeuroPace RNS System”. In: ().

[17] Biao Zhang, Jianjun Wang, and Thomas Fuhlbrigge. “A review of the commercial brain-computer interface technology from perspective of industrial robotics”. In: *2010 IEEE International Conference on Automation and Logistics, ICAL 2010*. 2010. DOI: 10.1109/ICAL.2010.5585311.

[18] Gernot R. Müller-Putz and Gert Pfurtscheller. “Control of an electrical prosthesis with an SSVEP-based BCI”. In: *IEEE Transactions on Biomedical Engineering* 55.1 (2008). ISSN: 00189294. DOI: 10.1109/TBME.2007.897815.

[19] Edward F. Melcer et al. “CTRL-labs: Hand activity estimation and real-time control from neuromuscular signals”. In: *Conference on Human Factors in Computing Systems - Proceedings*. Vol. 2018-April. 2018. DOI: 10.1145/3170427.3186520.

[20] D H Perkel and T H Bullock. *Neural Coding - A Report based on an NRP Work Session*. 2008.

[21] Kenneth O Johnson. "Neural coding". In: *Neuron* 26.3 (2000), pp. 563–566.

[22] Emery N. Brown, Robert E. Kass, and Partha P. Mitra. *Multiple neural spike train data analysis: State-of-the-art and future challenges*. 2004. DOI: 10.1038/nn1228.

[23] Guosong Hong and Charles M. Lieber. *Novel electrode technologies for neural recordings*. 2019. DOI: 10.1038/s41583-019-0140-6.

[24] Jakob H. Macke et al. *Generating spike trains with specified correlation coefficients*. 2009. DOI: 10.1162/neco.2008.02-08-713.

[25] Cristina Savin and Gašper Tkačik. *Maximum entropy models as a tool for building precise neural controls*. 2017. DOI: 10.1016/j.conb.2017.08.001.

[26] Jonathan W. Pillow et al. "Spatio-temporal correlations and visual signalling in a complete neuronal population". In: *Nature* 454.7207 (2008). ISSN: 00280836. DOI: 10.1038/nature07140.

[27] Wilson Truccolo et al. "A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects". In: *Journal of Neurophysiology* 93.2 (2005). ISSN: 00223077. DOI: 10.1152/jn.00697.2004.

[28] Pietro Berkes, Frank Wood, and Jonathan Pillow. "Characterizing neural dependencies with copula models". In: *Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*. 2009.

[29] Nina Kudryashova et al. *Parametric Copula-GP model for analyzing multidimensional neuronal and behavioral relationships*. 2020.

[30] Valerio Mante et al. "Context-dependent computation by recurrent dynamics in prefrontal cortex". In: *Nature* 503.7474 (2013). ISSN: 00280836. DOI: 10.1038/nature12742.

[31] Patrick T. Sadtler et al. "Neural constraints on learning". In: *Nature* 512.7515 (2014). ISSN: 14764687. DOI: 10.1038/nature13665.

[32] Thomas M. Hall, Kianoush Nazarpour, and Andrew Jackson. "Real-time estimation and biofeedback of single-neuron firing rates using local field potentials". In: *Nature Communications* 5 (2014). ISSN: 20411723. DOI: 10.1038/ncomms6462.

[33] Carsen Stringer et al. "High-dimensional geometry of population responses in visual cortex". In: *Nature* 571.7765 (2019). ISSN: 14764687. DOI: 10.1038/s41586-019-1346-5.

[34] Juan A. Gallego et al. "Long-term stability of cortical population dynamics underlying consistent behavior". In: *Nature Neuroscience* 23.2 (Feb. 2020), pp. 260–270. ISSN: 15461726. DOI: 10.1038/s41593-019-0555-4.

[35] Sam Roweis and Zoubin Ghahramani. *A unifying review of linear gaussian models*. 1999. DOI: 10.1162/089976699300016674.

[36] Anne C. Smith and Emery N. Brown. "Estimating a state-space model from point process observations". In: *Neural Computation* 15.5 (2003). ISSN: 08997667. DOI: 10.1162/089976603765202622.

[37] Mark M. Churchland et al. "Neural population dynamics during reaching". In: *Nature* 487.7405 (2012). ISSN: 00280836. DOI: 10.1038/nature11129.

[38] Abigail A. Russo et al. "Neural Trajectories in the Supplementary Motor Area and Motor Cortex Exhibit Distinct Geometries, Compatible with Different Classes of Computation". In: *Neuron* 107.4 (2020). ISSN: 10974199. DOI: 10.1016/j.neuron.2020.05.020.

[39] David E. Rumelhart and David Zipser. "Feature discovery by competitive learning". In: *Cognitive Science* (1985). ISSN: 03640213. DOI: 10.1016/S0364-0213(85)80010-0.

[40] Diederik P Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *arXiv* (Dec. 2013). URL: http://arxiv.org/abs/1312.6114.

[41] Mohammad Reza Keshtkaran et al. "A large-scale neural network training framework for generalized estimation of single-trial population dynamics 1". In: *bioRxiv* (2021).

[42] Joel Ye and Chethan Pandarinath. "Representation learning for neural population activity with Neural Data Transformers". In: *bioRxiv* (2021).

[43] Timothy D Kim et al. “Inferring Latent Dynamics Underlying Neural Population Activity via Neural Differential Equations”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 5551–5561. URL: http://proceedings.mlr.press/v139/kim21h.html.

[44] Irene Mendez et al. “Non-invasive real-time access to the output of the spinal cord via a wrist wearable interface”. In: *bioRxiv* (2021).

[45] Omid G Sani, Bijan Pesaran, and Maryam M Shanechi. “Modeling behaviorally relevant neural dynamics enabled by preferential subspace identification (PSID)”. In: *bioRxiv* (Jan. 2019), p. 808154. DOI: 10.1101/808154. URL: http://biorxiv.org/content/early/2019/10/17/808154.abstract.

[46] Stephanie Naufel et al. “A muscle-activity-dependent gain between motor cortex and emg”. In: *Journal of Neurophysiology* 121.1 (2019). ISSN: 15221598. DOI: 10.1152/jn.00329.2018.

[47] S. Kullback and R. A. Leibler. “On Information and Sufficiency”. In: *The Annals of Mathematical Statistics* 22.1 (1951). ISSN: 0003-4851. DOI: 10.1214/aoms/1177729694.

[48] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*. 2014. DOI: 10.3115/v1/d14-1179.

[49] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. “A hybrid convolutional variational autoencoder for text generation”. In: *EMNLP 2017 - Conference on Empirical Methods in Natural Language Processing, Proceedings*. 2017. DOI: 10.18653/v1/d17-1066.

[50] Junxian He et al. “Lagging inference networks and posterior collapse in variational autoencoders”. In: *7th International Conference on Learning Representations, ICLR 2019*. 2019.

[51] Hongkun Yu et al. *TensorFlow Model Garden*. https://github.com/tensorflow/models. 2020.

[52] Martìn Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[53] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[54] Open Source. *Tensorflow 2.0.0*. https://github.com/tensorflow/tensorflow/releases/tag/v2.0.0. 2019.

[55] Luke Yuri Prince. *Hierarchical LFADS*. https://github.com/lyprince/hierarchical_lfads. 2020.

[56] David Sussillo et al. *Computation Through Dynamics*. https://github.com/google-research/computation-thru-dynamics. 2020.

[57] Tensorflow. *Making new Layers and Models via subclassing*. https://www.tensorflow.org/guide/keras/custom_layers_and_models. Accessed: 2021-08-07. 2021.

[58] Robert C. Martin. *Getting a SOLID start. - Clean Coder*. https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start. Accessed: 2021-08-07. 2009.

[59] Edward N. Lorenz. "Deterministic Nonperiodic Flow". In: *Journal of the Atmospheric Sciences* 20.2 (1963). ISSN: 0022-4928. DOI: 10.1175/1520-0469(1963)020<0130:dnf>2.0.co;2.

[60] J. R. Dormand and P. J. Prince. "A family of embedded Runge-Kutta formulae". In: *Journal of Computational and Applied Mathematics* 6.1 (1980), pp. 19–26.

[61] E. V. Evarts. "Relation of pyramidal tract activity to force exerted during voluntary movement." In: *Journal of neurophysiology* 31.1 (1968). ISSN: 00223077. DOI: 10.1152/jn.1968.31.1.14.

[62] Donald R. Humphrey, E. M. Schmidt, and W. D. Thompson. "Predicting measures of motor performance from multiple cortical spike trains". In: *Science* 170.3959 (1970). ISSN: 00368075. DOI: 10.1126/science.170.3959.758.

[63] E. V. Evarts et al. “Motor cortex control of finely graded forces”. In: *Journal of Neurophysiology* 49.5 (1983). ISSN: 00223077. DOI: 10.1152/jn.1983.49.5.1199.

Appendix A

Interfaces

A.1 Legacy Interface

In the legacy interface, command line arguments are the only method for providing parameters to the algorithm. This often results in commands that are hard to read due to the absence of structure in the parametrization and the counter-intuitive names. For example, the snippet below is a typical command used for one of the experiments in section 4.2. One noticeable shortcoming of this interface is that training and evaluation, which are typically performed jointly, are here delegated to two different commands, both containing similar arguments in order to define the same network architecture.

```
#!/bin/bash
echo "Training LFADS on data"
python latentneural/legacy/lfads/original/run_lfads.py --kind=train --data_dir="datadir" \
--data_filename_stem=dataset.h5 --lfads_save_dir="datadir/results" --co_dim=0 \
--factors_dim=3 --ext_input_dim=0 --controller_input_lag=1 --output_dist=poisson \
--do_causal_controller=false --batch_size=16 --learning_rate_init=.01 \
--learning_rate_stop=1e-05 --learning_rate_decay_factor=.95 --learning_rate_n_to_compare=6 \
--do_reset_learning_rate=false --keep_prob=0.95 --gen_dim=64 --ci_enc_dim=128 \
--ic_dim=64 --ic_enc_dim=64 --ic_prior_var_min=0.1 --gen_cell_input_weight_scale=1.0 \
--cell_weight_scale=1.0 --do_feed_factors_to_controller=true --kl_start_step=1000 \
--kl_increase_steps=1000 --kl_ic_weight=1.0 --l2_gen_scale=2000 --l2_con_scale=0.0 \
--l2_start_step=0 --l2_increase_steps=1000 --ic_prior_var_scale=0.1 \
--ic_post_var_min=0.0001 --kl_co_weight=1.0 --prior_ar_nvar=0.1 --cell_clip_value=5.0 \
--max_ckpt_to_keep_lve=5 --do_train_prior_ar_atau=true --co_prior_var_scale=0.1 \
--csv_log=fitlog --feedback_factors_or_rates=factors --do_train_prior_ar_nvar=true \
--max_grad_norm=200.0 --device=cpu:0 --num_steps_for_gen_ic=100000000 \
--ps_nexamples_to_process=100000000 --checkpoint_name=lfads_vae \
--temporal_spike_jitter_width=0 --checkpoint_pb_load_name=checkpoint \
--inject_ext_input_to_gen=false --co_mean_corr_scale=0.0 --gen_cell_rec_weight_scale=1.0 \
--max_ckpt_to_keep=5 --output_filename_stem="" --ic_prior_var_max=0.1 \
--prior_ar_atau=10.0 --do_train_io_only=false --do_train_encoder_only=false \
--seed=0
```

```
echo "Evaluating LFADS on data"
python latentneural/legacy/lfads/original/run_lfads.py --kind=train --data_dir="datadir" \
--kind=posterior_sample_and_average \
[...] # continues with the same arguments as the previous command
```

Code A.1: Typical legacy LFADS command.

A.2 Model Interface

The Python interface provides a simple solution for developers who are interested in running one-off experiments without significant learning curves. In a few lines the model can be initialised, run and evaluated:

```
import tensorflow as tf
import numpy as np
from latentneural import LFADS
from latentneural.utils import AdaptiveWeights

train = np.random.binomial(1, 0.5, (80, 100, 50)).astype(float)
valid = np.random.binomial(1, 0.5, (10, 100, 50)).astype(float)
test = np.random.binomial(1, 0.5, (10, 100, 50)).astype(float)

adaptive_weights = AdaptiveWeights(
    initial=[0.5, 1, 1],
    min_weight=[0., 0., 0.],
    max_weight=[1., 1., 1.],
    update_step=[1, 2, 1],
    update_start=[2, 1, 1],
    update_rate=[-0.05, -0.1, -0.01]
)

model = LFADS(neural_dim=50, max_grad_norm=200)
model.build(input_shape=[None] + list(train.shape[1:]))
model.compile(optimizer=tf.keras.optimizers.Adam(1e-3),
              loss_weights=adaptive_weights.w)

model.fit(x=train, y=None, callbacks=[adaptive_weights], shuffle=True,
    epochs=4, validation_data=(valid,None))
test_out, _ = model(test, training=False)
model.save(save_location)
```

Code A.2: LFADS Python interface.

A.3 Runtime Interface

The second interface was designed around scientists running batch jobs and/or executing experiments in remote clusters. A single YAML or JSON configuration file contains all the necessary information to run the task, including some dataset properties, the architecture of the network, the optimizer and the hyperparameters schedule. The execution is triggered through a simple command from the terminal:

```
python latentneural -r "\$FILENAME"
```

The code can be run in a *debug* mode, which logs additional metrics (i.e. the gradient flow and the weights norm in each layer) or in the *standard* mode, which is marginally faster. Training metrics are logged in a format accessible through TensorBoard, as in the legacy code. A complete description of the experiment, including environment and timing information, settings used and the code version is also saved in a target folder. At the end of the training, the model is evaluated on test data and saved in the same target directory. Below, a YAML example of a settings file:

```yaml
data: {directory: datadir}
model:
  settings:
    full_logs: False
    default_layer_settings:
      kernel_initializer:
        arguments: {distribution: normal, mode: fan_in, scale: 1.0}
        type: variance_scaling
      kernel_regularizer: {arguments: {l: 0.1}, type: l2}
    encoded_dim: 64
    factors: 3
    layers:
      decoder: {kernel_regularizer: {arguments: {l: 3}, type: l2}, original_cell: false}
      encoder: {dropout: 0.05, var_trainable: false, var_min: 0.1}
   max_grad_norm: 200
    timestep: 0.01
  type: lfads
output: {directory: outputdir}
runtime:
  batch_size: 16
  epochs: 10000
  learning_rate: {factor: 0.95, initial: 0.01, patience: 30, terminating: 1.0e-05}
  optimizer: {arguments: {beta_1: 0.9, beta_2: 0.999, epsilon: 0.1}, type: adam}
  weights:
    initial: [1.0, 0.0, 0.0]
    update_rate: [0.0, 0.0005, 0.0005]
seed: 0
```

Code A.3: LFADS YAML interface.

The nested structure enhances readability, while allowing for a more granular parameter definition. For example, while in the legacy LFADS implementation the L2 regularization is applied to all weights as a whole, in the interface proposed it can be tuned to individual layers.

Appendix B

Hyperparameters

B.1 Synthetic Data Experiment

Name	LFADS	TNDM
Factors	3	2 Relevant, 1 Irrelevant
Initial Conditions Size	64	64
Behaviour Dense Type	–	Synchronous
Initial Conditions Size	64	64
Sampling Distribution Minimum Variance	0.1	0.1
Initialisation Type	Variance Scaling	Variance Scaling
Optimizer	Adam	Adam
Maximum Epochs	10000	10000
Batch Size	16	16
Initial Learning Rate	0.1	0.1
Learning Rate Decay	0.95	0.95
Learning Rate Patience	6	6
Terminating Learning Rate	0.00001	0.00001
Neural Weight	1	1
Behavioural Weight	–	0.1
KL Divergence Step-size	0.0002	0.0002
L2 Weight Step-size	0.0002	0.0002
Disentanglement Weight Step-size	–	0.0002
Standard L2	0.1	0.1
Generator GRU L2 (Kernel)	3	3
Generator GRU L2 (Recurrent)	3	3
Behaviour Dense L2	–	0.000001
Dropout	0.15	0.15

Table B.1: Hyperparameters used for the synthetic data experiment.

B.2 Forces Data Experiment

Name	LFADS	TNDM
Factors	Various	Various
Initial Conditions Size	64	64
Behaviour Dense Type	–	Causal
Initial Conditions Size	64	64
Sampling Distribution Minimum Variance	0.01	0.01
Initialisation Type	Variance Scaling	Variance Scaling
Optimizer	Adam	Adam
Maximum Epochs	10000	10000
Batch Size	16	16
Initial Learning Rate	0.1	0.1
Learning Rate Decay	0.95	0.95
Learning Rate Patience	15	15
Terminating Learning Rate	0.0001	0.0001
Neural Weight	1	1
Behavioural Weight	–	0.00001
KL Divergence Step-size	0.0005	0.0005
L2 Weight Step-size	0.0005	0.0005
Disentanglement Weight Step-size	–	0.0005
Standard L2	0.01	0.01
Generator GRU L2 (Kernel)	1	1
Generator GRU L2 (Recurrent)	1	1
Behaviour Dense L2	–	0.01
Dropout	0.15	0.15

Table B.2: Hyperparameters used for the forces data experiment.

B.3 EMG Data Experiment

Name	LFADS	TNDM
Factors	6	4 Relevant, 2 Irrelevant
Initial Conditions Size	64	64
Behaviour Dense Type	–	Causal
Initial Conditions Size	64	64
Sampling Distribution Minimum Variance	0.01	0.01
Initialisation Type	Variance Scaling	Variance Scaling
Optimizer	Adam	Adam
Maximum Epochs	10000	10000
Batch Size	16	16
Initial Learning Rate	0.1	0.1
Learning Rate Decay	0.95	0.95
Learning Rate Patience	15	15
Terminating Learning Rate	0.0001	0.0001
Neural Weight	1	1
Behavioural Weight	–	0.001
KL Divergence Step-size	0.0005	0.0005
L2 Weight Step-size	0.0005	0.0005
Disentanglement Weight Step-size	–	0.0005
Standard L2	0.02	0.02
Generator GRU L2 (Kernel)	1	1
Generator GRU L2 (Recurrent)	1	1
Dropout	0.2	0.2

Table B.3: Hyperparameters used for the EMG data experiment.

Appendix C

Supplementary Plots

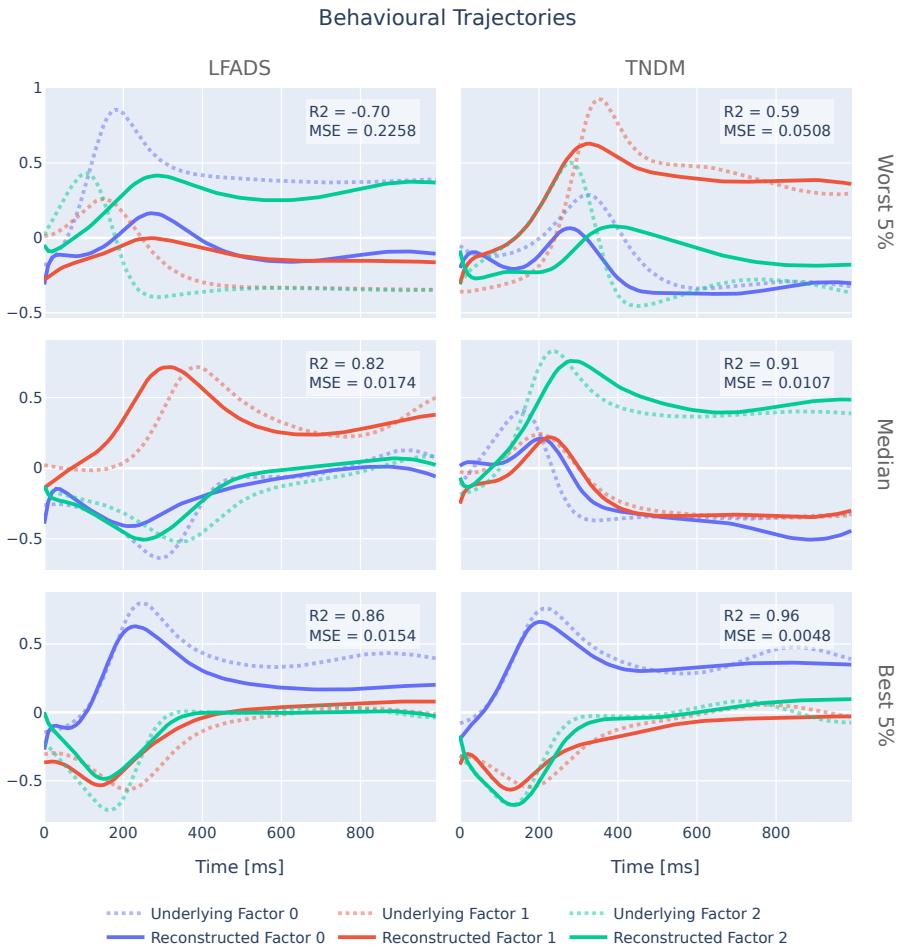


Figure C.1: Encoded ("underlying") and reconstructed trajectories for the bottom 5% accuracy, median accuracy and top 5% accuracy in the experiment with 50 training trials, a baseline rate of 5Hz and behavioural noise set to 2.

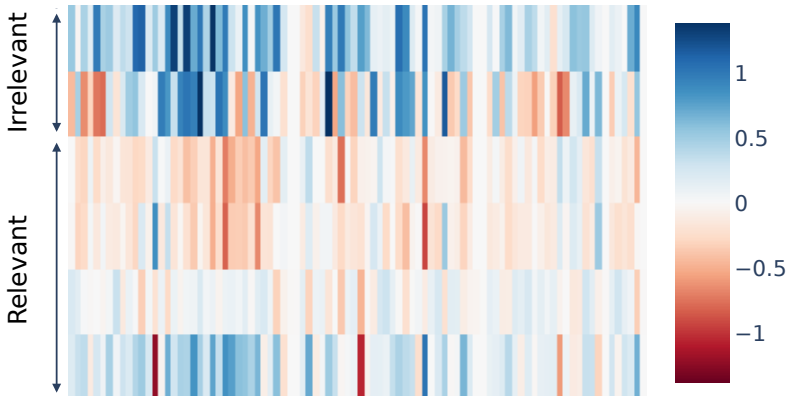


Figure C.2: Weights mapping the latent factors to the behavioural activity for the EMG task.

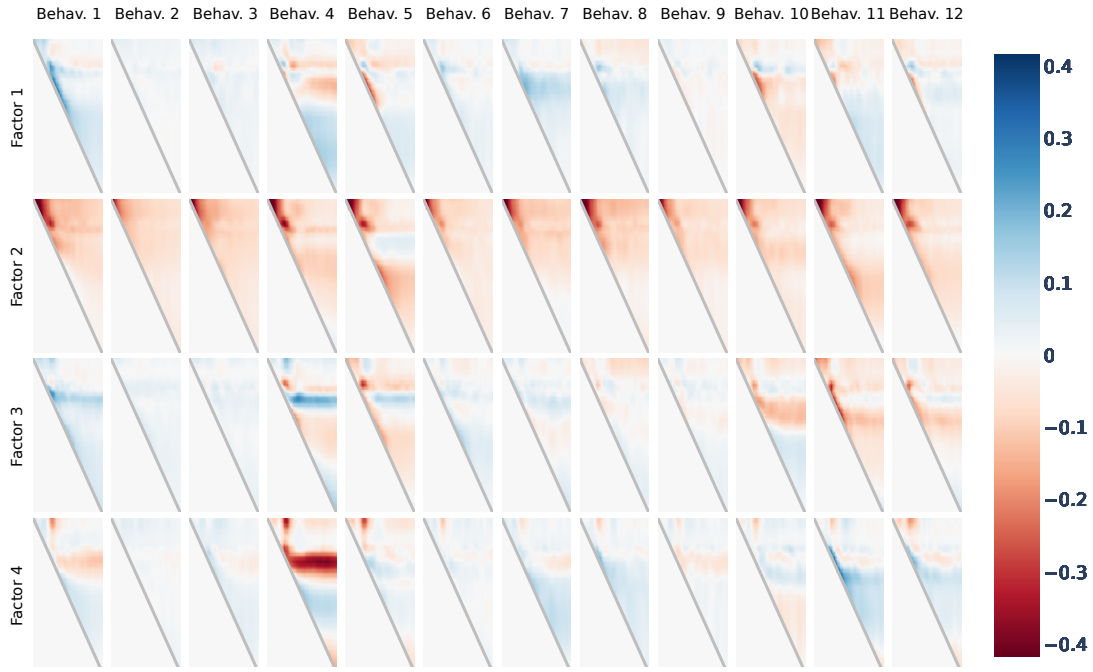


Figure C.3: Weights mapping the latent factors to the neural activity for the EMG task.