universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Hyperparameter Optimization for the Application of Neural Manifold Learning Algorithm Bundle-Net on Novel Datasets

verfasst von | submitted by

Emilija Mazūraitė BA (Hons)

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 910
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Computational Science
Betreut von | Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. -Ing. Moritz Grosse-Wentrup
Mitbetreut von | Co-Supervisor:	Dr.techn. Akshey Kumar B.Sc. M.Sc.

Acknowledgements

I would like to thank my supervisor Professor Moritz Grosse-Wentrup and co-supervisor Dr. Akshey Kumar for their support and guidance during the project.

And I am very grateful for all the support I received from my family during my studies, especially my parents and my now brother-in-law Henry.

Abstract

Neural data are inherently noisy, and experimental constraints often lead to small datasets. Machine learning inference using such data can be challenging and models need to be carefully tuned to ensure robust performance. This tuning process involves adjusting variables that are needed for training but are not learned from data, and this procedure has to be done for each new dataset.

BunDLe-Net is a neuronal manifold learning algorithm that learns latent neural representations using joint neural and behavioural recordings. This project introduces a hyperparameter optimization (HPO) toolbox that sequentially identifies optimal hyperparameters for BunDLe-Net's application to novel datasets and validates the inferred manifold for behavioural information encoding using permutation testing. Hyperparameters were optimized using a combination of Bayesian optimisation algorithm, Tree-structured Parzen Estimator, grid search for hyperparameters with small search spaces to ensure robustness, and statistical testing to determine the optimal latent dimension without assuming manifold properties.

The proposed HPO method shows improved performance over the baseline method, Random Search, in five out of seven datasets. Neural manifolds inferred with BunDLe-Net, configured with optimal hyperparameters, showed significantly better-than-chance behavioral encoding for real behavioural datasets, while performing at chance level for noise datasets. This result indicates that the validation methodology can be used to confirm that the inferred manifold is not informative in the absence of meaningful behavioural signals. These findings demonstrate that the BunDLe-Net model's performance can benefit from systematic optimization, but it is robust to suboptimal tuning.

Kurzfassung

Neuronale Daten sind von Natur aus verrauscht, und experimentelle Einschränkungen führen oft zu kleinen Datensätzen. Das maschinelle Lernen mit solchen Daten kann anspruchsvoll sein, und die Modelle müssen sorgfältig abgestimmt werden, um eine robuste Leistung zu gewährleisten. Dieser Abstimmungsprozess umfasst die Anpassung von Variablen, die für das Training benötigt werden, jedoch nicht aus den Daten gelernt werden. Er muss für jeden neuen Datensatz erneut durchgeführt werden.

BunDLe-Net ist ein Algorithmus, der latente neuronale Repräsentationen anhand gemeinsamer neuronaler und verhaltensbezogener Aufzeichnungen lernt. In diesem Projekt wird eine Toolbox zur Hyperparameter-Optimierung (HPO) eingeführt, die schrittweise die optimalen Hyperparameter für die Anwendung von BunDLe-Net auf neue Datensätze bestimmt und die abgeleiteten Werte für die Kodierung von Verhaltensinformationen durch Permutationstests validiert.

Die Hyperparameter wurden mit einer Kombination aus einem Bayes'schen Optimierungsalgorithmus, Tree-structured Parzen Estimator (TPE), Gittersuche für kleine Suchräume zur Robustheitssteigerung und statistischen Tests optimiert, um die optimale latente Dimension zu bestimmen, ohne Annahmen über die Struktur der Mannigfaltigkeiten zu treffen.

Die vorgeschlagene HPO-Methode übertrifft in fünf von sieben Datensätzen die Basismethode Random Search. Die mit BunDLe-Net und optimierter Hyperparameterkonfiguration abgeleiteten neuronalen Mannigfaltigkeiten zeigten eine signifikant bessere Verhaltenskodierung für reale Verhaltensdatensätze, während sie für Rauschdatensätze auf Zufallsniveau lagen. Diese Ergebnisse zeigen, dass die Leistung des BunDLe-Net-Modells durch eine systematische Optimierung verbessert werden kann, es aber zugleich robust gegenüber suboptimalen Einstellungen bleibt.

Contents

List of Tables

List of Figures

1. Introduction

1.1. Scope of Work

Challenges associated with neural data used for machine learning inference

Biological datasets are often constrained by experimental factors such as time, cost, ethical issues, and invasiveness, resulting in small sample sizes [2]. In neuroscience research, additional challenges arise due to data quality and resolution limitations: only a fraction of the neuronal population can be recorded at a time, and when presented with repeated stimuli, neurons display varying responses [3]. Consequently, neural datasets are inherently noisy, high-dimensional, and variable across experiments, making robust model inference challenging.

Small datasets are particularly prone to overfitting, a term in machine learning that describes when a model learns patterns that fail to generalize to new or independent data [4]. The success of machine learning methods depends on the careful tuning of hyperparameters, which define the model's configuration but are not learned directly from the data. Hyperparameter Optimization (HPO) is essential for ensuring reliable model inference and requires custom tuning for each dataset [5]. Improper tuning can lead to poor generalization, causing the model to perform well on training data but fail when applied to unseen data.

Contributions to the field from the current project

Neural manifold learning algorithms are used to study neuronal dynamics, offering insights into brain function. Application of a neural learning algorithm requires considerable knowledge of model tuning which may not be within the expertise of neuroscientists. The current project aims to provide a comprehensive and automated model tuning toolbox for the BunDLe-Net neural manifold learning algorithm [1] which predicts neural dynamics in relation to behaviour. The algorithm has been shown effective for inferring neural manifolds from *Caenorhabditis elegans* data, and this project facilitates the model's applications to novel datasets with continuous or discrete-valued behavioural

variables. In addition to automated model tuning, this project establishes a methodology to validate whether the learned manifold encodes behavioral information. This is achieved through permutation testing and by comparing the manifold's performance against synthetic datasets and noise controls. The overarching goal is to describe a comprehensive procedure for optimising the BunDLe-Net neural manifold inference algorithm for novel datasets with continuous or discrete behavioural variables. The proposed framework was applied to rat, monkey and worm data.

1.2. Theoretical Background

1.2.1. Historical Context: From Single Neurons to Population-Level Computation

The primary objective of neuroscience is to understand the mechanisms underlying brain function. Early research focused on single neurons as basic units of computation in the brain [6, 7]. As the recording technologies advanced to allow simultaneous recording of multiple neurons, the computational unit paradigm shifted towards studying neural populations, as some activity patterns cannot be estimated from individual spike trains alone [8]. This shift led to the population doctrine, which posits that neural information is encoded in the coordinated activity of neuron populations rather than in individual neurons[9].

Neural activity in the brain is highly correlated, with networks exhibiting redundancy and recurring activity patterns [10]. Despite the high-dimensional nature of activity in the brain, neural trajectories—which describe the time evolution of neural population activity—tend to be confined to a low-dimensional space [11]. This suggests that neural dynamics can be effectively captured by a limited set of features, far fewer than what would be expected if each neuron operated independently [9].

This observation aligns with the Manifold Hypothesis [12], a fundamental principle in machine learning, which states that high-dimensional systems often exhibit intrinsic structure that can be captured using a lower-dimensional representation. Indeed, theoretical advancements in optimization of machine learning algorithms can facilitate neuroscientific research [13, 14]. In neuroscience, neural manifold analysis leverages this principle to study how populations of neurons encode and process information over time. The central idea is that despite the high-dimensional nature of neural activity, its essential dynamics are governed by a lower-dimensional manifold [15, 16].

1.2.2. Neural Manifold

Definition

The term manifold originates from mathematics, where it refers to a space that appears locally Euclidean but may have a more complex global structure [17]. In neuroscience, a neural manifold refers to a low-dimensional subspace embedded within an N-dimensional neural state space, which represents the collective activity of a neuronal population [15]. Within the high-dimensional neural state space, each axis corresponds to the activity of an individual neuron. However, in the low-dimensional manifold, the dimensions correspond to neural modes—correlated patterns of population activity that capture most of the variance in neural responses [18]. The neural modes span neural state space and they can be estimated from neural population recordings using dimensionality reduction techniques [16].

Neural manifolds in model organisms

In recent years, neural manifold analysis has provided crucial insights into how populations of neurons encode behavior and cognition in different model organisms. For example, motor control in monkeys has been shown to be governed by a low-dimensional representation of neural activity [19], while in mice, neural manifolds have been shown to jointly encode task-specific spatial information and cognitive variables during virtual maze navigation [20]. This approach has also revealed conserved circuit architectures in heading direction representation: a ring attractor network with local excitation and global inhibition encodes heading direction in flies [21], a mechanism later identified in zebrafish [22], demonstrating common patterns of neural dynamics across vertebrates. Together, these findings highlight how inference of neural manifolds can uncover key principles of motor control and cognition in the brain.

Neural manifold learning - related algorithms

Neural manifold learning encompasses a set of techniques designed to represent high-dimensional neural activity in a lower-dimensional space while preserving the essential structure and information of the original data [23]. These approaches range from general dimensionality reduction techniques to specialized algorithms specifically developed for neural data analysis. Linear method applications [15, 24, 25], for neural dynamics modeling include Principal Component Analysis (PCA) [26], which linearly projects data onto a lower-dimensional space by identifying orthogonal directions (principal components) that

capture the most variance, and Multidimensional Scaling (MDS) [27] which in its classical form reconstructs a low-dimensional embedding by preserving pairwise distances using eigenvalue decomposition of a transformed distance matrix. Although linear methods may be suitable for some behaviours measured in a laboratory setting, neural activity often exhibits a nonlinear structure, necessitating the use of nonlinear manifold learning techniques [11]. Recent findings confirm that neural manifolds are inherently nonlinear [28].

The foundation of nonlinear dimensionality reduction was laid by methods such as Isomap [29], which is based on geodesic distances, and Locally Linear Embedding (LLE) [30], which reconstructs each point as a weighted sum of its neighbors. Other nonlinear approaches include t-distributed Stochastic Neighbor Embedding (t-SNE) [31], which preserves local relationships by modeling pairwise similarities, and UMAP [32], which captures both local and global structure using a graph-based approach. Notably, neither UMAP nor t-SNE use temporal information available in the neural data [33].

Beyond traditional dimensionality reduction, deep learning based methods have been developed for neural data analysis. LFADS [33], a sequential variational autoencoder, infers latent neural dynamics from single rather than averaged trials. AutoLFADS [34] extends this approach by incorporating hyperparameter optimization to systematically tune model parameters for improved inference of latent dynamics. More recently, CEBRA [35] has emerged as a state-of-the-art algorithm for learning neural manifolds. It employs contrastive learning, a deep learning method that organizes similar data points closer together while separating dissimilar ones, to infer manifolds from joint neural and behavioral data by leveraging temporal structure.

The BunDLe-Net algorithm

BunDLe-Net [1] is a deep learning based neural manifold learning algorithm that maps high-dimensional neural dynamics onto a low-dimensional embedding relevant to behavioral contexts. The model treats neural trajectories as a Markov process, a type of stochastic process in which, conditional on its values at time n, the probability of transitioning to the next state $n+1$ is independent of past states [36]. This assumption ensures a causally sufficient representation of the system [1]. The model consists of an embedding function, a state transition model that predicts future manifold states, and a behavioral predictor that decodes behavior from the manifold. By adjusting its modules, BunDLe-Net can be used to infer manifolds with both linear and nonlinear methods. Its architecture integrates these components through a composite loss function that enforces both Markovian properties

and the preservation of decodable behavioral information. Given its complexity, BunDLeNet requires careful hyperparameter selection, highlighting the need for automated HPO to facilitate its use in neuroscience research.

1.2.3. Brief Overview of Hyperparameter Optimisation for Neural Networks

While neural manifold learning tools based on machine learning provide a powerful framework for inferring low-dimensional embeddings of neural activity, their effectiveness depends on optimal tuning of model hyperparameters. This challenge extends to broader applications in neural networks. Automated hyperparameter optimization plays a key role in ensuring effective model training, achieving reproducible results, and minimizing workload for model users and facilitating research [37]. HPO strategies vary in the number and order of hyperparameters tuned, as well as the algorithms used for the search. These aspects are briefly discussed below.

Hyperparameters are settings that are required for model training but are not updated during training, unlike model parameters such as weights of a neural network which are learned from the data [38]. Hyperparameters are broadly categorized into training and network structure related parameters: training parameters affect how model parameters are updated during learning and they have an effect on model's convergence and training stability, whereas the structure related parameters are related to the complexity of a function that is being learned [38]. The order in which hyperparameters are tuned matters because some parameters have a greater impact on the updates of weights during training [39]. Guidelines have been proposed for the sequence of hyperparameter tuning by machine learning experts [39].

Training related hyperparameters

Training a machine learning model involves minimizing a loss function, a process governed by optimization algorithms. In deep learning, the Adaptive Moment Estimation (Adam) optimizer [40] is often recommended as the default choice for large-scale problems or those involving numerous parameters [41]. Adam adapts learning rates for each parameter based on estimates of first-order and second-order moments of the gradients, combining the strengths of AdaGrad [42] and RMSProp [43] algorithms which are particularly suited for problems with sparse gradients and noisy problems, respectively. The primary hyperparameters to tune in Adam are the learning rate and the exponential decay rates for the moment estimates, commonly denoted as β_1 and β_2. In most cases, adjusting the learning rate suffices for optimal performance, and β_1 and β_2 only need to be tuned if the

model fails to converge or is highly unstable [38]. Notably, the AutoLFADS framework for learning neural manifolds employs the Adam optimizer.

Among training related parameters, learning rate and batch size are prioritized for optimising first, as they directly influence weight updates during training and affect the speed of convergence [38]. Learning rate is a positive scalar that determines the step size for updating weights during training, and it can be constant or dynamically adjusted by applying a learning rate decay schedule during training to reduce the step size as the training is nearing the optimal solution [41]. Small learning rate sizes need more epochs for convergence and large learning rates can lead to faster convergence but unstable behaviour. Similarly, (mini)batch size is another key training hyperparameter. It refers to the subset of training data seen before model parameters are updated. Optimization algorithms are categorized based on the batch size: batch gradient descent updates weights using the entire dataset (deterministic), stochastic gradient descent (SGD) updates after each sample (online), and mini-batch gradient descent balances both approaches by using a subset of data for updates [41]. Batch size values are recommended to be in powers of two for optimal hardware access [41]. However, batch size also affects generalization: large batches may decrease runtime but tend to obtain sharp minima, which is associated with poorer generalization [44]. Sharp minima correspond to regions in the loss landscape with high curvature, making models more sensitive to perturbations and less robust to unseen data [44]. In contrast, smaller batch sizes favor flatter minima and they are recommended as a regularization method to guard against overfitting [41]. Other major training-related hyperparameters that help prevent overfitting include weight decay and early stopping [38].

Network structure related hyperparameters

In contrast, network structure-related hyperparameters determine a model's learning capacity and ability to represent complex functions [41]. These hyperparameters include the number and width of layers, the architecture or shape of the neural network, and the choice of activation functions among others. An optimal neural network architecture may also follow a hierarchical or pyramid-like structure to better capture multi-scale features when learning patterns from biological data [45], which often exhibit hierarchical structures or organization [46]. Additionally, regularization terms may be added to decrease the risk of overfitting: examples include dropout of neurons whereby the weights are removed at random with a given probability for some neurons in the network [47], or noise injection.

The proposed hyperparameter optimization pipeline for the BunDLe-Net algorithm optimizes learning rate, epochs, batch size, and network structure parameters associated with different modules of the BunDLe-Net architecture.

Search strategies for HPO

With key hyperparameters established, we will now examine strategies for exploring the search space. Feurer and Hutter [37] outline three main paradigms of the automated HPO: black-box search, Bayesian optimization, and multi-fidelity optimization.

Baseline methods

Baseline optimization, also known as black-box optimization, refers to model-free methods that explore a prespecified hyperparameter space without leveraging prior knowledge. Grid search is the simplest strategy, systematically evaluating all possible hyperparameter combinations. While highly parallelizable, it becomes computationally expensive and infeasible in high-dimensional search spaces due to the curse of dimensionality, a phenomenon where the number of possible configurations increases exponentially with the number of dimensions, leading to an intractable optimisation problem [37].

A more efficient alternative is random search, where hyperparameters are sampled randomly from a given distribution. Compared to grid search, random search is better suited for new datasets where the relative importance of hyperparameters is unknown [48]. In deep learning applications, random search has been shown to outperform grid search, achieving comparable results with a significantly lower computational budget [49]. While black-box methods lack adaptivity, they are often preferred due to their parallelization capabilities, fault tolerance (minimizing experiment loss in case of hardware failure), and ability to explore large search spaces with minimal assumptions [49]. Additionally, black-box methods serve as useful baselines for defining search spaces before applying more complex adaptive algorithms which employ a model to pick the next configuration candidate to evaluate [37].

Bayesian Optimization

Bayesian optimization is an adaptive optimization method designed for functions that are expensive to evaluate, non-differentiable, or lack concavity and linearity [50, 51]. Instead of directly evaluating the costly objective function for each configuration, Bayesian optimization builds a probabilistic model of past evaluations to guide the search for optimal hyperparameters. An acquisition function selects promising candidates based

on this model, and the actual objective function is only computed for these selected candidates, iteratively refining the search space [51]. Bayesian Optimisation (BO)'s main advantages include its ability to handle continuous, discrete, and categorical variables, as well as its effectiveness in high-dimensional search spaces with limited computational resources [52].

BO example: Tree-structured Parzen Estimator algorithm

Tree-structured Parzen Estimator (TPE) [53] is a BO algorithm used in this study to efficiently explore complex hyperparameter spaces. Instead of directly modeling the objective function, TPE constructs probability distributions over hyperparameter configurations conditioned on performance. It maintains two separate density functions: one for high-performing configurations and another for low-performing ones. The search process begins with an initial evaluation of randomly chosen hyperparameter configurations to estimate these distributions. Thereafter, new hyperparameter configuration candidates are selected by maximizing the ratio of the high-performing to low-performing distributions, based on an estimation by a Parzen estimator. This approach allows TPE to effectively navigate search spaces containing continuous, discrete, and categorical variables without assuming a smooth functional relationship between hyperparameters and performance [52].

Multi-fidelity methods focus on dynamic resource allocation rather than candidate selection, typically by evaluating functions on small subsets to terminate poorly performing candidates early [37]. While algorithms such as BOHB [54] integrate Bayesian optimization with multi-fidelity methods to reduce runtime, they are less suitable for problems with limited data availability. Since this study prioritizes robust hyperparameter selection over runtime efficiency, we focus on Bayesian optimization using the TPE algorithm, complemented by grid search for hyperparameters with small search spaces, and an optimization process where hyperparameters are tuned in a specific order, following established guidelines [39].

2. Methods

2.1. Datasets

Worm dataset

This dataset was originally published by Kato and colleagues [55] and includes calcium imaging data recorded from five freely moving worms, with human-annotated motor behavior labeled into eight distinct states (See Figure 2.1). The recordings contain 100-200 neurons per animal and include 2500-3500 time samples with sampling frequency of around 2.9 Hz. Further details can be found at [1, 55].

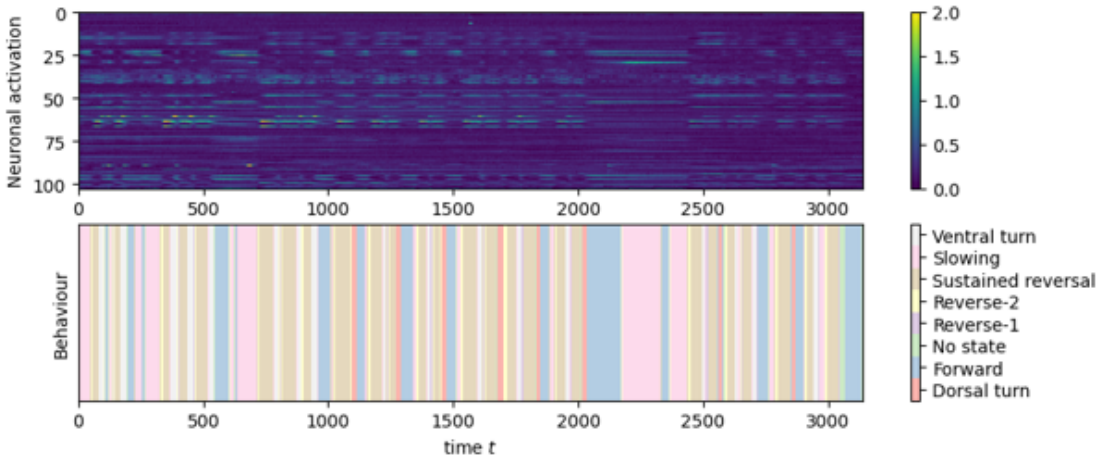

Figure 2.1.: *C.elegans* dataset, image courtesy [1]. Time t refers to sample numbers.

Neural recordings consist of calcium imaging data from the whole brain of nematode *C. elegans*, whereas behavioural data are human annotated labels describing motor states. Displayed data are from worm 0. With 3137 samples and $\sim$ 2.9 Hz sampling frequency, each time bin corresponds to $\sim$ 0.34 s.

Rat dataset

Rat hippocampal data were first introduced by Grosmark and colleagues [56, 57]. They contain electrophysiological and behavioral recordings from four male Long-Evans rats during maze running. Behavioral data include continuous variables for position of the animal during navigation of the maze and discrete variables for running direction (left or right). Neural recordings were taken from the hippocampus and they consist of spike recordings, taking discrete values. The number of neurons recorded in the dataset range between 48 and 120, whereas the number of samples range between around 6'500 and 47'000. The sampling frequency for rat datasets is 20 kHz.

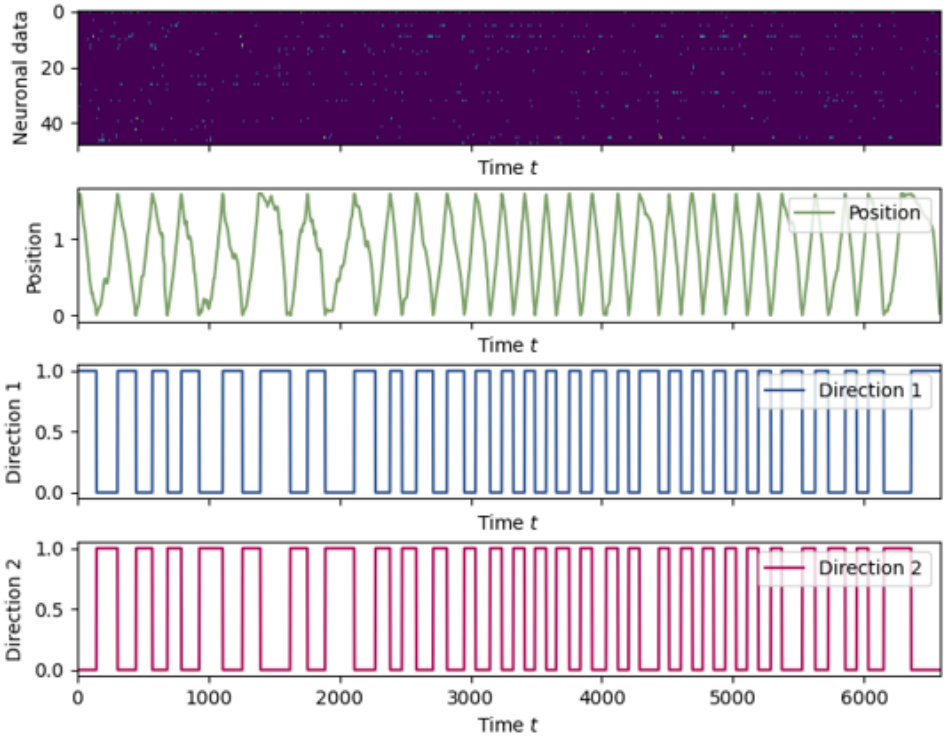

Figure 2.2.: Neural and behavioural data for rat dataset 3.

Neural data are spike recordings, whereas behavioural data include position of an animal during maze running (continuous valued) and left or right direction (discrete valued). Time t refers to sample numbers and each sample corresponds to ~ 0.32 s.

Monkey dataset

The dataset was originally introduced by Chowdhury et al. [58]. This dataset contains neural recordings from area 2 of the primary somatosensory cortex in rhesus macaques during a reaching task. Neural activity was recorded using multielectrode arrays and later processed by [35] to estimate time-varying spike rates. Simultaneously, continuous behavioral data were collected, representing cursor position during reaching movements, described as x- and y-coordinates. Neural recordings were obtained from 65 electrode sites in the primary somatosensory cortex and contain 115'800 samples where each point represents a 1 ms bin [35].

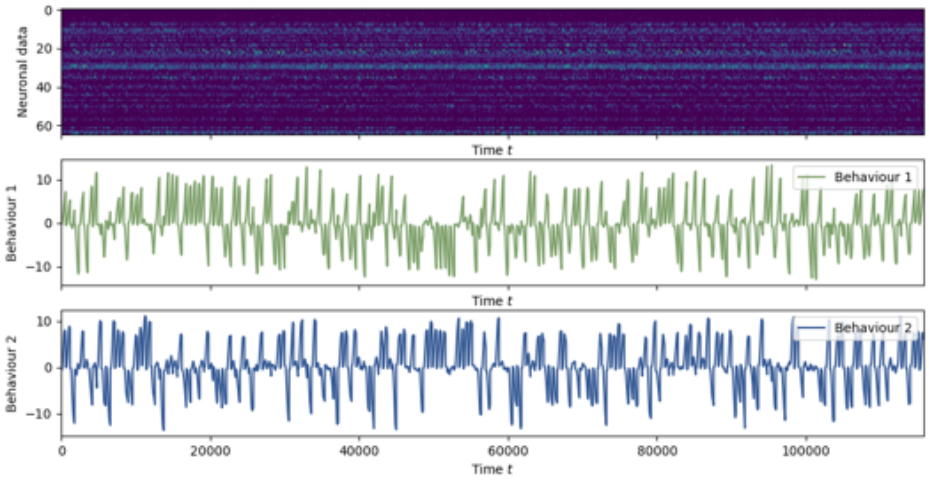

Figure 2.3.: Monkey dataset.

Neural data are estimates of spike rates, whereas Behaviour 1 and 2 represent x and y coordinates of a cursor position during reaching movements. Time t refers to sample numbers and each sample corresponds to 1 ms.

2.1.1. Implementation Details

Experiments were conducted on the Neuroinformatics Computing Cluster at the University of Vienna, utilizing a node with an Intel i7-10700K (8C/16T, 3.8GHz, 5.1GHz boost) and 62 GiB RAM (59 GiB available). The code is written in Python 3.10.12 [59].

The code utilizes NumPy [60] for numerical operations and Pandas [61] for structured data processing. Seaborn [62] and Matplotlib [63] are used for visualizations, while SciPy [64] is employed for statistical tests. Dunn's test was conducted using Scikit-Posthocs [65].

Hyperparameter optimization was performed using the Ray Tune library [66], and performance metrics, including R^2 and accuracy scores, were computed using sklearn.metrics from Scikit-Learn [67]. The network structure was visualized with TensorFlow [68] Keras layers using Graphviz [69]. The visualization of neural data and behavioral plots, and embedding plots were adapted from the code provided by [1] with minor adjustments. The continuous variant of the BunDLe-Net model [1] and associated functions for data preparation were used with minor modifications to integrate hyperparameter tuning. The original open source implementation is available at (https://github.com/akshey-kumar/BunDLe-Net-continuous/tree/main).

2.1.2. BunDLe-Net Architecture

A schematic view of the algorithm is presented in Figure 2.4. It consists of two branches: in both branches of the algorithm the embedding function τ yields an estimate for the low-dimensional embedding Y_{t+1}, but the result is achieved by using different inputs: neural data X_{t+1} for the upper branch, and neural data X_t for the lower branch. In the lower branch, once Y_t^L is obtained, a first order transition model T_Y predicts the difference ΔY_t^L which leads to an estimate for the next time step, ΔY_{t+1}^L. The difference between the two low dimensional embeddings Y_{t+1}^L and Y_{t+1}^U is evaluated by the loss function L_{Markov}. Additionally, the Y_{t+1} estimates are supplied to a *predictor* layer to obtain behavioural labels $\tilde{B}_{t+1}$. A loss function for the behavioural labels is used to estimate the reconstruction error between the true labels and those obtained from the latent representation Y.

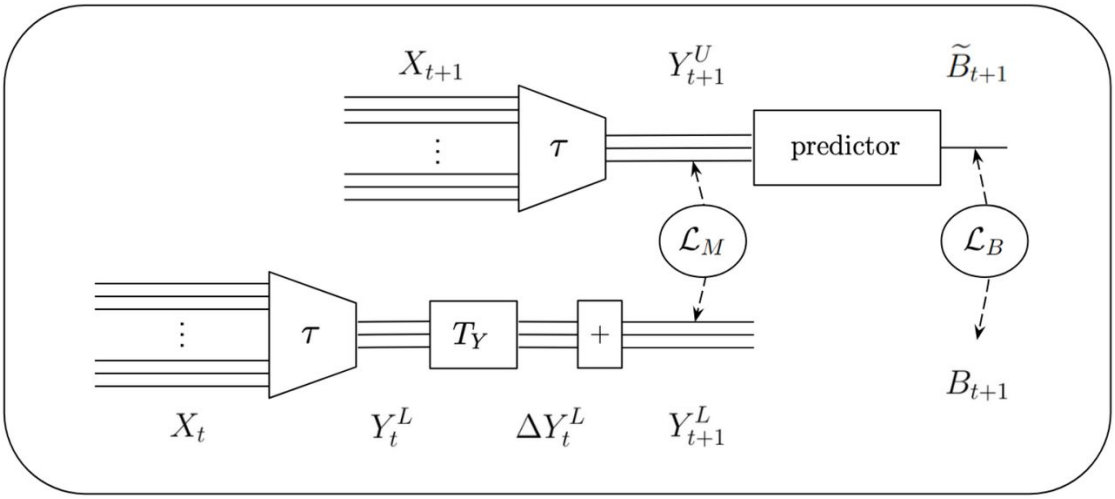

Figure 2.4.: BunDLe-Net architecture, image courtesy [1].

2.2. Hyperparameter Optimisation

2.2.1. Objective Function

Hyperparameter optimization was guided by a composite loss function, defined as a weighted sum of two components: (1) Markov loss L_{Markov}, calculated as the mean squared error (MSE) between the inferred manifold at the upper and lower branches at time $t+1$:

$$L_{Markov}(Y^U_{t+1}, Y^L_{t+1}) = \|Y^U_{t+1} - Y^L_{t+1}\|^2$$

and (2) Behavioral loss, $L_{Behaviour}$ computed as cross-entropy loss between true and predicted variables for discrete data and MSE between true and predicted behavioural target variables for continuous data, respectively:

$$L_{Behaviour(discrete)}(B_{t+1}, \tilde{B}_{t+1}) = -\sum_{j=1}^{m} B^j_{t+1} log(\tilde{B}^{(j)}_{t+1})$$

$$L_{Behaviour(continuous)}(B_{t+1}, \tilde{B}_{t+1}) = \|B_{t+1} - \tilde{B}_{t+1}\|^2$$

To ensure, that the two loss terms are of the same order, they were weighted by a γ parameter, set to 0.9 during hyperparameter tuning:

$$L = (1-\gamma)L_{Markov} + \gamma L_{Behaviour}$$

All experiments were conducted using a time window of 5, and optimization was performed using the Adam optimizer [40] with variable learning rate and $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

2.2.2. Data Splitting and Cross-Validation

The dataset was split into seven partitions, with one partition held out as a test set, which remained unused during model training and hyperparameter selection. For each hyperparameter configuration, model performance was assessed using five-fold cross-validation, with the validation loss reported as the average across the five folds.

2.2.3. Hyperparameter Search Strategy

Hyperparameter tuning followed a sequential optimization approach, where key model parameters were refined in multiple stages. Each stage produced optimal values that

were then used to guide the search in subsequent steps. The order of hyperparameters optimized followed recommendations by [39]:

1. Learning rate and number of epochs.
2. Batch size.
3. Network structure related parameters (τ and T_Y function structure).
4. Latent dimension.

Defining the search space

Before optimization, a sensitivity test was performed to define the search space for learning rate and number of epochs, using Random Search. Learning rates for the sensitivity test were sampled from a uniform scale $(0.0001, 0.01)$, while epochs were sampled from the values 50, 100, 500, 1000.

Baseline comparison

A random search with $n = 100$, using the same search space as the proposed method for all hyperparameters, served as the baseline comparison. After model selection, behavioral validation was performed; however, this step was conducted only on the original dataset, excluding control groups. Specifically, permutation testing was applied solely to the original data.

Training related parameters: learning rate and epochs

Once the search space was established, the learning rate and number of epochs were optimized using the TPE algorithm, as implemented in the Hyperopt library [53]. Learning rates were sampled on a logarithmic scale, while epoch values were selected from a list of values. For the rat dataset the epoch search space was 50, 100, 250, 500, for monkey 50, 100, 150, 200, 250, and for worm datasets 50, 100, 500, 1000, 1500. The TPE search was configured with 20 initial random trials, followed by 100 trials for optimization.

During this phase, the batch size was fixed at 128, and the network architecture followed a default configuration. The τ-component consisted of five layers with 50, 30, 25, and 10 neurons, each using ReLU activation function, followed by a latent dimension of 3 neurons with a linear activation function. The T_Y function was implemented as a single-layer network with a linear activation function.

Training related parameters: batch size

After selecting the optimal learning rate and number of epochs, batch size optimization was performed using grid search. The optimal batch size was determined based on the lowest average validation loss. The search considered batch sizes of 32, 64, 128, 256, and 512, as recommended by [41]. During this process, the network structure remained unchanged.

Optimizing network structure related parameters

The network structure was optimized using the TPE algorithm, refining both the τ function and the T_Y function to achieve an optimal configuration. The τ function was implemented as a pyramid-shaped artificial neural network, where the number of neurons progressively decreased across successive layers. The optimization process focused on three structural parameters: network depth, number of neurons in the first layer, and the pyramidal factor, which determined the proportion by which neurons decreased in successive layers. The search space included network depths of 3, 5, 7, and 9 layers, initial layer sizes of 50, 100, 200, and 500 neurons, and pyramidal factors of 0.5, 0.7, and 0.9.

T_Y function

For the T_Y function, two configurations were explored. The first was a single-layer architecture with a linear activation function, where the number of neurons was set equal to the latent dimension. The second was a two-layer non-linear model, incorporating ReLU activation in the first layer, followed by a final linear activation layer.

Both the T_Y and τ functions incorporated the latent dimension, which varied between 1 and 15 during the optimization process. Once the structural parameters were optimized, the latent dimension of the embedding was further refined through statistical testing of grid search results.

Latent dimension

To avoid making assumptions about the data, a statistical testing approach was used to select an optimal dimension for the latent embedding. A grid search with 15 independent runs was conducted to collect validation loss values, with the search space ranging from 1 to the latent dimension identified during the previous step - structural optimization. The distribution of validation losses across latent dimensions was then analyzed using a non-parametric Kruskal-Wallis test to assess whether significant differences existed

between groups, obtaining a test statistic and p values. If $p > \alpha = 0.05$, indicating no statistically significant difference between the group distributions, the smallest latent dimension was selected. However, if $p \leq \alpha$, post-hoc pairwise comparisons were conducted using Dunn's test [70], with Bonferroni correction applied to control the family-wise error rate in multiple comparisons. The Bonferroni correction applied had the magnitude of the number of groups.

In the post-hoc analysis, all observations were ranked, and the mean rank for each latent dimension was computed. Dunn's test statistic measured the differences between these ranks, converting them into p-values using the standard normal distribution. The final selection criterion was to choose the smallest latent dimension whose validation loss distribution was not significantly different from that of the group with the absolute minimum validation loss. Once the latent dimension was determined, the model was trained for 1.5× the previously selected number of epochs to ensure sufficient convergence. The final number of epochs was selected based on visual analysis of the loss plot.

2.3. Behavioural Validation

R^2 score (coefficient of determination)

The predictive performance for models trained on datasets with continuous behavioral variables was evaluated using the R^2 metric, also known as the coefficient of determination. This metric quantifies the proportion of variance in the dependent variable that is explained by the independent variable. Here the independent variables were true behavioral variables, whereas the predicted behavioral variables were the dependent variables.

The R^2 metric is computed as:

$$R^2 = 1 - \frac{RSS}{TSS}$$

where RSS (Residual Sum of Squares) represents the sum of squared residuals, and TSS (Total Sum of Squares) denotes the total variance in the observed data. An R^2 value of 1 indicates a perfect fit, whereas a value of 0 corresponds to a model that only predicts the mean of the true data [71].

Evaluating Model Performance Through Controlled Comparisons

To evaluate whether the model's predictions for the target variables exceed chance levels, we compared its performance across three conditions, varying the behavioural data inputs supplied to the BunDLe-Net model, while keeping the neural data unchanged:

1. **Group 1: true target variables** – the original behavioural target variables or labels for continuous and discrete data, respectively.
2. **Group 2: synthetic data** – target variables generated as a function of neural data, here the function was mean and variance.
3. **Group 3: noise data** – behavioural data that was sampled from a uniform distribution $U(0, 10)$.

All behavioural data were preprocessed using standard scaling. The model was trained using the previously identified optimal hyperparameter configuration across all groups. The representation of behavioural information within the neural manifold was measured using the coefficient of determination R^2 between true and predicted behavioural variables for continuous data and classification accuracy for discrete variables. To evaluate whether the model's predictions were significantly better than chance, permutation testing was conducted as follows.

2.3.1. Procedure

For each group, the following steps were performed:

1. Neural and behavioural data were scaled using standard scaling.
2. The BunDLe-Net model was trained on the training set and evaluated on a held-out test set.
3. Predicted behavioural labels are obtained by applying the predictor function to the inferred manifold.
4. Performance was assessed using R^2 between true and predicted target variables if data were continuous, and classification accuracy if behavioural data were discrete.
5. The behavioural data were circularly permuted by m timepoints, ensuring that permutations remained at least 15 steps away from the original alignment.
6. For each permutation, the R^2 score was recomputed. This process was repeated for $n = 1000$ permutations.
7. A p-value was computed as the proportion of R^2 scores from permuted data exceeding the observed R^2 score, under the null hypothesis that the model's performance is unaffected by circular permutation of predicted behaviour.

3. Results

3.1. Hyperparameter optimisation

3.1.1. Optimal Hyperparameter Values

Tables 3.1 and 3.2 present the optimal hyperparameter values identified through the proposed HPO process for each dataset. The optimal hyperparameter values were selected based on the validation loss. Figure 3.1 presents search results across different steps of the proposed strategy for the Rat 3 dataset. The pairwise comparisons across different latent dimension groups are shown in Figure 3.2.

Dataset	**Epochs**	**Learning rate** (η)	**Batch size** (B)
Rat 0	100	0.0038	64
Rat 1	50	0.0003	64
Rat 2	50	0.0002	64
Rat 3	50	0.0047	256
Monkey	100	0.0005	128
Worm 0	200	0.0017	128
Worm 1	100	0.0019	64

Table 3.1.: Proposed HPO method: optimal training related hyperparameter values.

Dataset	T_Y	τ: **Layers**	τ: **1st layer**	τ: **Pyramid factor**	d	d_{opt}
Rat 0	Linear	4	200	0.9	11	6
Rat 1	Non-linear	4	50	0.7	6	3
Rat 2	Linear	4	50	0.5	4	3
Rat 3	Linear	4	200	0.7	9	3
Monkey	Linear	6	500	0.5	3	2
Worm 0	Linear	6	300	0.7	9	4
Worm 1	Non-linear	4	500	0.7	5	2

Table 3.2.: Proposed HPO method: optimal network structure related hyperparameter values.

d - latent dimension; d_{opt} - optimal latent dimension selected based on Dunn's test with Bonferroni correction, using statistical significance as a criterion. The number of layers in τ function referred in the table excludes regularization layers.

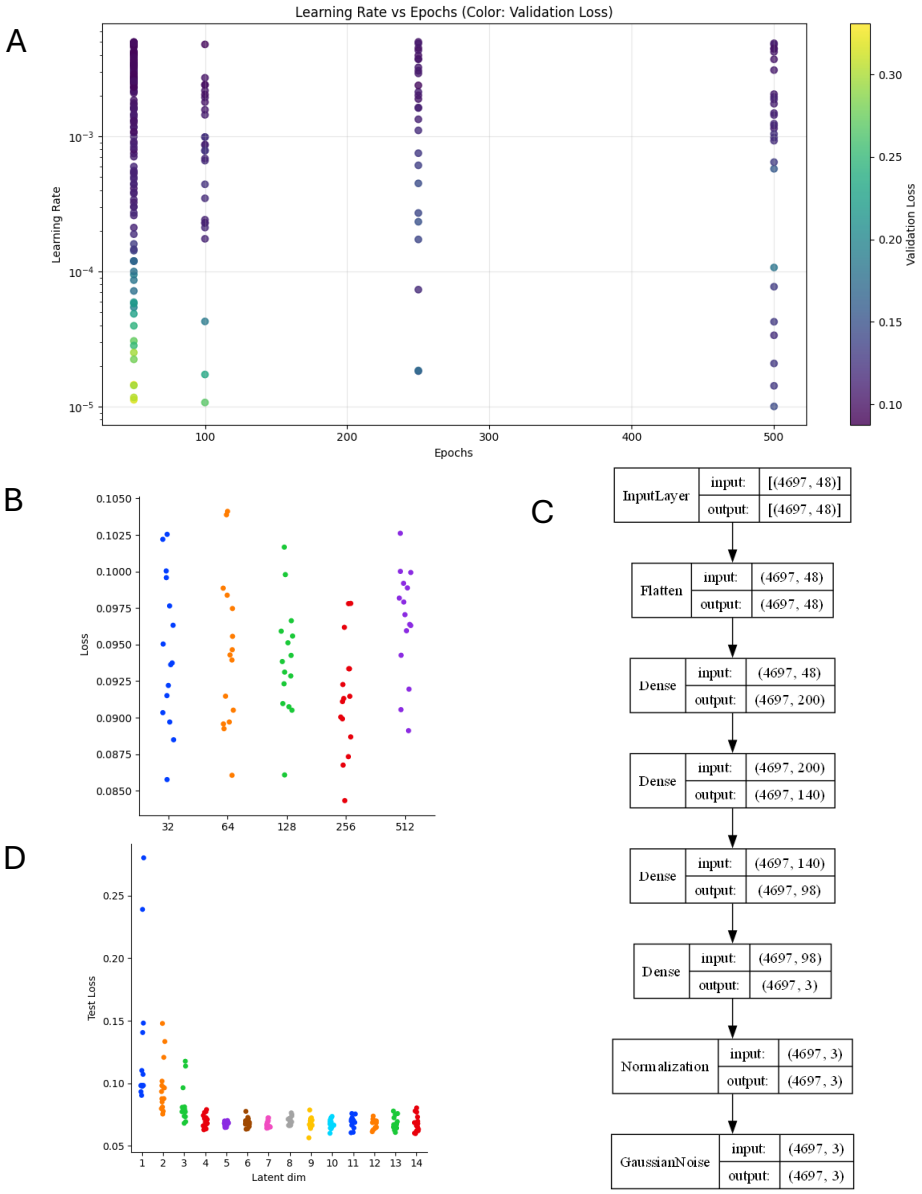

Figure 3.1.: Optimal hyperparameters

Hyperparameter optimization results for the Rat 3 dataset. (A) Results of the search using the TPE algorithm for optimal epochs versus learning rate, plotted on a logarithmic scale with validation loss as the color gradient. (B) Grid search results ($n = 15$) for optimal batch size. (C) Optimal network structure for tau function identified through structure search using TPE. (D) Grid search results ($n = 15$) over a range of latent dimension values, generating data for the next step in the optimization process, A Kruskal-Wallis test indicated significant differences among groups ($H = 102.59, p < 0.05$).

Latent dimension selection

For all datasets tested, latent dimension optimisation through statistical testing of the validation losses across different values resulted in selecting a lower value compared to the one found in the previous network structure optimisation part. When Kruskal Wallis test was applied to the grid search results for different latent dimensions, the null hypothesis that groups come from the same distribution was rejected for all datasets, with test scores presented in TableA.1.

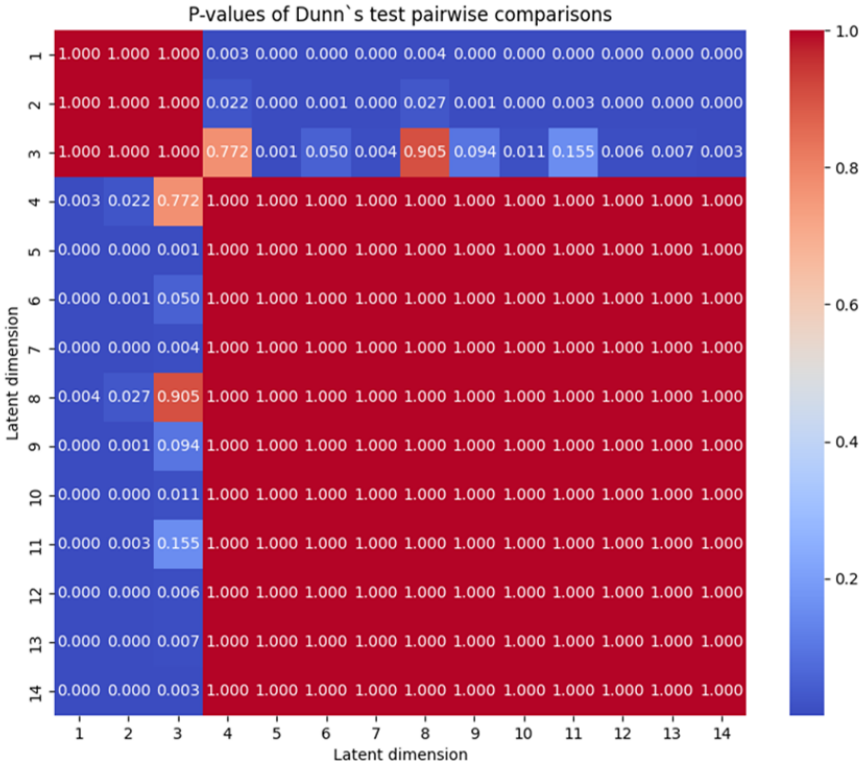

Figure 3.2.: Pairwise comparisons of validation loss for different latent dimension.

Heatmap of p-values of pairwise Dunn's test with Bonferroni of validation loss for different latent dimension values. The results are shown from Rat 3 dataset. The latent dimension with the smallest observed validation loss was 9, while the statistically selected latent dimension, based on non-significant differences from the minimum loss group, was 3.

3.1.2. Comparison with Baseline Search

Table 3.3 presents the R^2 scores for the proposed HPO method compared to Random Search across multiple datasets, the optimal configuration for random search is provided in the Appendix . The results indicate that the proposed method outperforms Random Search in most cases, achieving higher R^2 values for Rat 0, Rat 1, Rat 3, and the Worm datasets, suggesting better predictive performance. However, in the Monkey dataset, the proposed method showed no improvement over Random Search. Additionally, for Rat 2, Random Search yielded slightly better performance, though R^2 scores remained low for both methods compared to other datasets.

For the monkey dataset, the optimal configurations selected by the proposed and baseline search methods are not similar, despite achieving similar R^2 scores, for example, the optimal batch size was 32 in one case and 128 in the other.

Dataset	Proposed method R^2	Random Search R^2
Rat 0	0.844	0.810
Rat 1	0.679	0.619
Rat 2	0.089	0.100
Rat 3	0.663	0.601
Monkey	0.831	0.838
	Proposed method, accuracy	Random Search, accuracy
Worm 0	0.808	0.731
Worm 1	0.756	0.711

Table 3.3.: Proposed method and baseline performance

R^2 score comparison between HPO with the proposed method and Random Search across different datasets.

Model Convergence Across Datasets

Training and validation loss curves for models trained with optimal hyperparameter configuration for rat and monkey datasets are displayed in Figure 3.3.

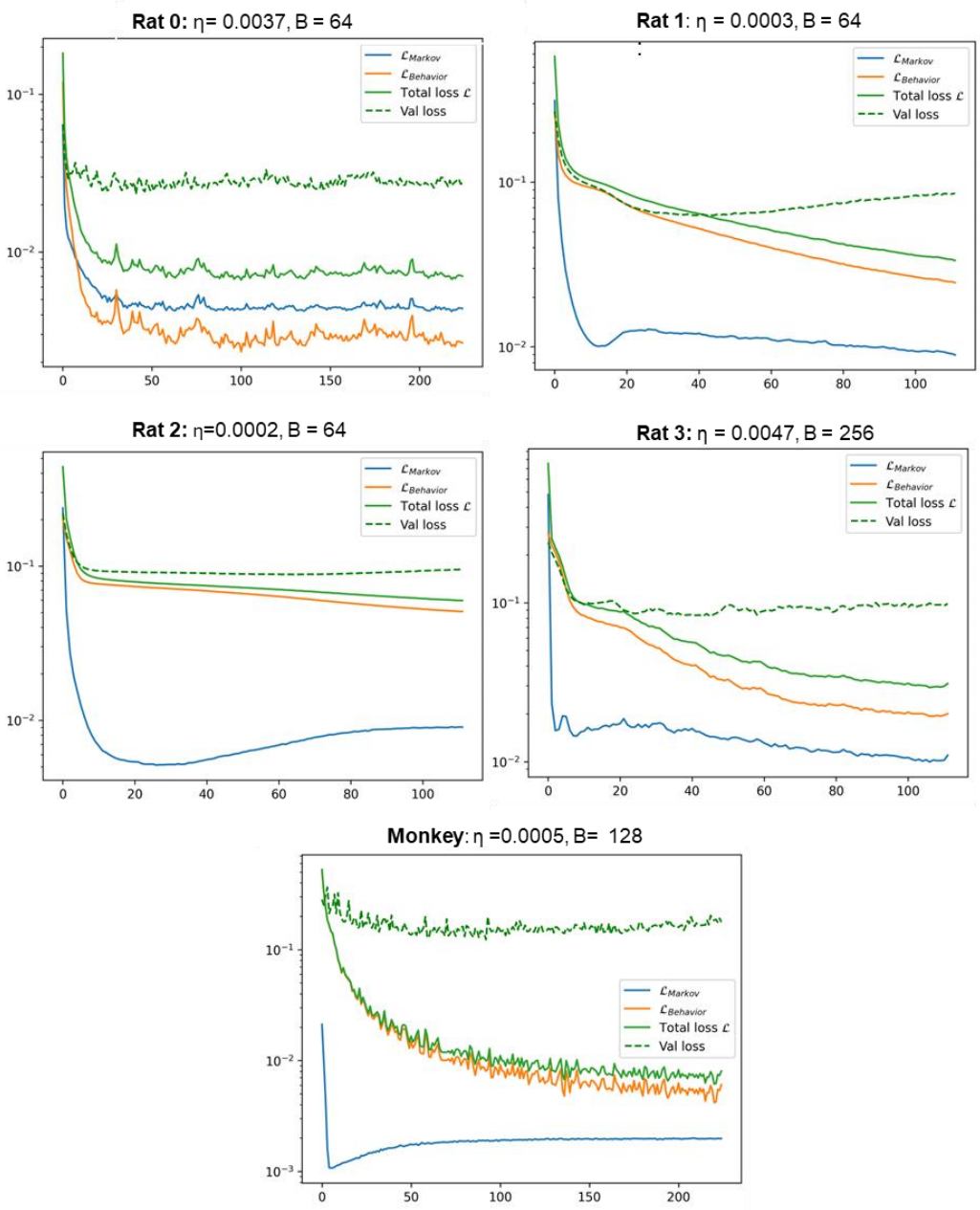

Figure 3.3.: Training and Validation Loss for Optimized Hyperparameters.

Loss curves from hyperparameter optimization, showing training and validation loss across epochs using the optimal training and structural hyperparameters.

3.1.3. Batch Size Effect on Model Convergence

For the Rat 0 dataset, the model was trained using the optimal hyperparameter configuration while varying batch sizes to assess their impact on convergence and generalization. The training and validation loss curves are displayed in Figure 3.4. For each batch value, the model was trained once and all other hyperparameters were held constant. The loss curves displayed in Figure 3.4 indicate that smaller batch sizes resulted in noisier convergence curves, mid-sized batch sizes exhibited large peaks, and larger batch sizes had smoother curves.

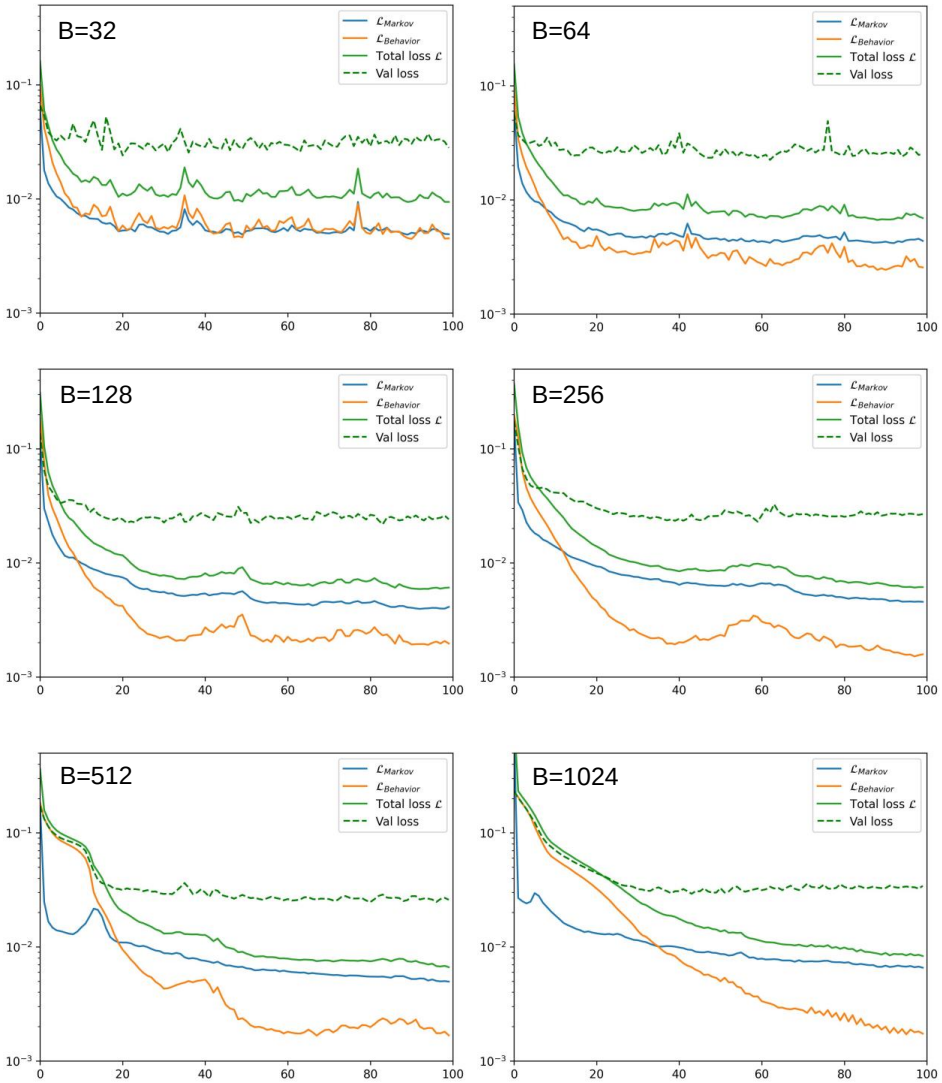

Figure 3.4.: Training and Validation Loss for Varying Batch Sizes.

Batch Size	R^2 Score	Validation Loss
32	0.5137	0.0280
64	0.5343	0.0254
128	0.5713	0.0241
256	0.5411	0.0267
512	0.5773	0.0258
1024	0.567	0.034

Table 3.4.: R^2 scores and validation loss for different batch sizes on the Rat 0 dataset.

3.2. Behavioural Validation

3.2.1. Hypotheses

To evaluate whether the inferred manifold encoded behavioural information, we applied permutation testing to assess the effect of circularly shifting behavioural data on model performance. Table 3.5 summarizes the permutation testing results across all datasets.

We tested the following hypotheses:

- Null Hypothesis (H): Shifting the behavioural dataset does not impact the model's performance.
- Alternative Hypothesis (H): Shifting the behavioural dataset results in decreased model performance.

We expected the following outcomes:

1. The model's performance should remain unchanged for the noise dataset after the predicted behavioural data are permuted, as no meaningful structure exists in the data.
2. The model should perform better than the shifted permutations for behavioural labels that were derived as a function of neural data (mean and variance), reflecting meaningful structure in the dataset.

Permutation testing confirmed these expectations. Specifically, for datasets where behavioural labels were generated as a function of neural data, the model's performance (measured by R^2 for continuous data and accuracy for discrete labels) significantly

decreased when labels were circularly permuted. This suggests that the inferred manifold encoded meaningful information related to the neural signal. Conversely, when the model was supplied with noise data, permutation testing showed that shifting had no impact on performance. This indicates that in the absence of meaningful behavioural structure, the model does not extract useful information. Permutation testing results for Rat 3 dataset are displayed in 3.5 and 3.6 shows true and predicted behavioural variables.

The results 3.5 show that the model achieved positive R^2 scores across all datasets when trained on true behavioural data ($p = 0.0$). For discrete data, accuracy scores were consistently higher than the random guessing baseline ($p = 0.125$ for 8 unique labels in *C.elegans* data). Performance on synthetic data (Group 2) varied, with some datasets (e.g., Monkey: $R^2 = 0.9344$) exhibiting higher performance than others (e.g., Rat 0: $R^2 = -0.32$). However, across all datasets, the inferred manifold had higher R^2 scores than permuted datasets ($p = 0.0$), indicating that the model captured patterns present in the input data. For noise data, R^2 scores were negative across all animals, with p-values > 0.05, indicating that permutation did not impact performance.

3.2.2. Permutation Testing Summary

Dataset	R^2, **Original dataset**	R^2, **Synthetic data**	R^2, **Noise**
Rat 0	0.844, $p = 0.0$	-0.3282, $p = 0.0$	-0.3346, $p = 0.373$
Rat 1	0.679, $p = 0.0$	0.4469, $p = 0.0$	-0.8785, $p = 0.416$
Rat 2	0.089, $p = 0.0$	-0.3680, $p = 0.0$	-0.292, $p = 0.063$
Rat 3	0.663, $p = 0.0$	-0.1833, $p = 0.0$	-0.226, $p = 0.176$
Monkey	0.831, $p = 0.0$	0.9344, $p = 0.0$	-0.0265, $p = 0.7$
Dataset	**Accuracy, Original dataset**	R^2, **Synthetic data**	R^2, **Noise**
Worm 0	0.808, $p = 0.0$	0.846, $p = 0.0$	-0.005, $p = 0.09$
Worm 1	0.756, $p = 0.0$	0.389, $p = 0.0$	-0.0038, $p = 0.32$

Table 3.5.: Encoding of behavioural information in the inferred manifold across groups

R^2 values for the original dataset, synthetic data, and noise across different datasets.

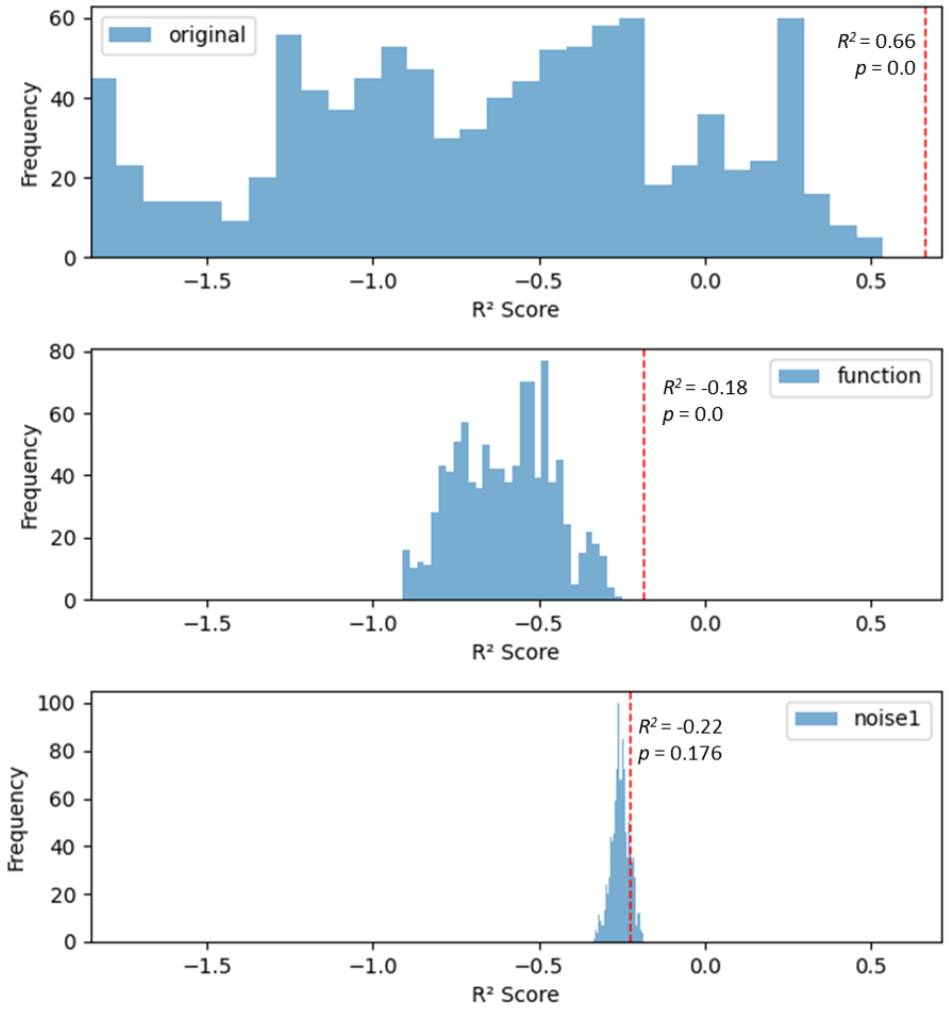

Figure 3.5.: Permutation testing results

The red line represents the R^2 score between the test and predicted target variables, while the blue histogram shows the distribution of R^2 scores obtained from circularly permuted predicted target variables (n = 1000). The graph displays results for Rat 3; additional experiments are provided in the appendix.

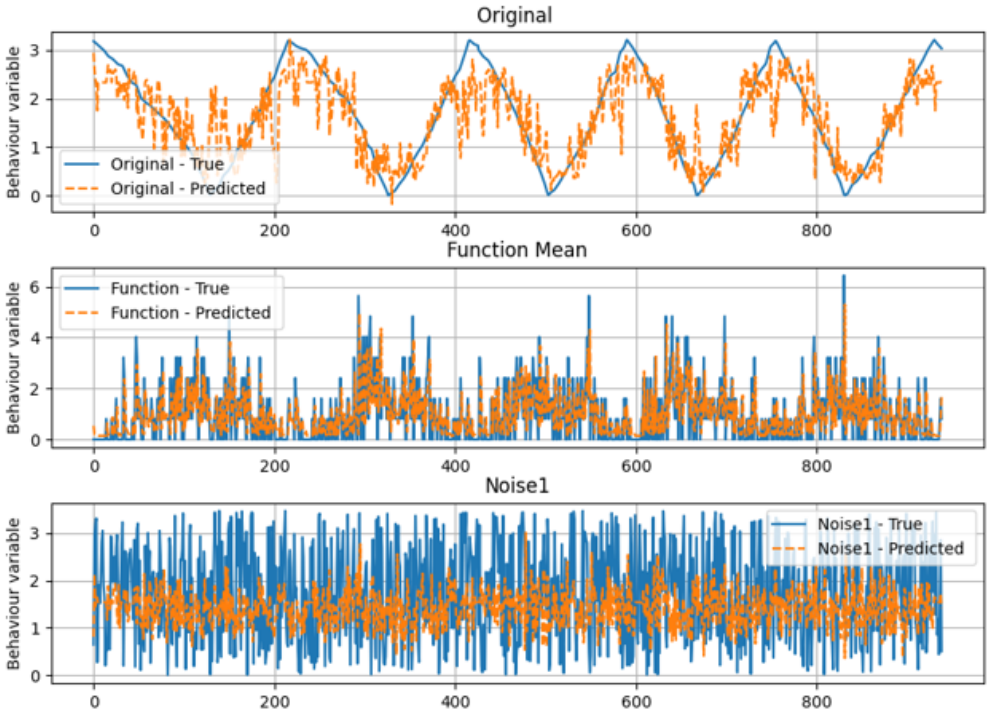

Figure 3.6.: True and predicted behavioural variables

Behavioural variables for true and predicted behavioural data across the three groups (true data, synthetic data, and noise data) for Rat 3. Predicted behavioural data were obtained by applying the predictor function to the inferred manifold. Additional plots for other datasets can be found in the Appendix.

3.2.3. Embeddings across Different Groups

The low dimensional embeddings inferred by the model for are shown in 3.7, only the embeddings for datasets with 2 or 3 latent dimension are displayed. Embeddings for noise group were either line shaped or spherical.

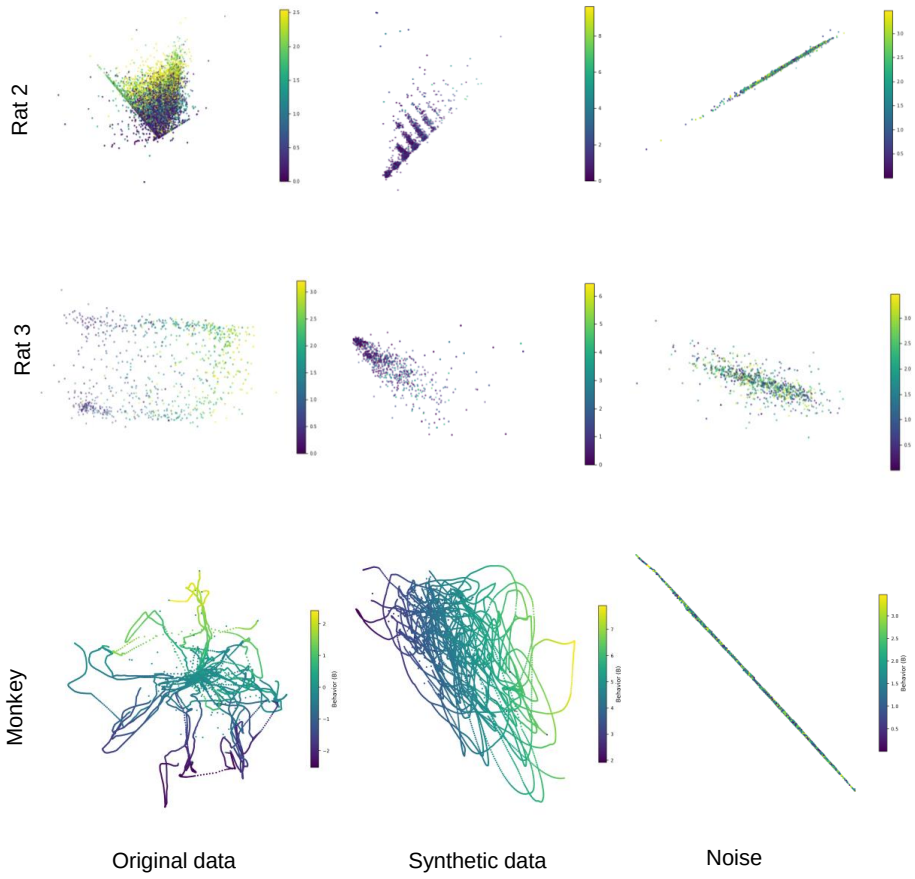

Figure 3.7.: Inferred manifolds.

Inferred embeddings for Rat 2, Rat 3, and Monkey across three behavioural data conditions (true data, synthetic data, and noise data). Each subplot visualizes the low-dimensional representation of neural activity as learned by BunDLe-Net. Monkey dataset is two-dimensional, whereas, the embeddings shown for rat data are three-dimensional.

The training and validation loss curves for different groups and datasets are displayed in Figure 3.8. Visual inspection suggests that the noise group exhibits the largest relative gap between validation and training loss, while the synthetic data group shows the smallest, though differences in axis scales should be considered when comparing gaps across plots.

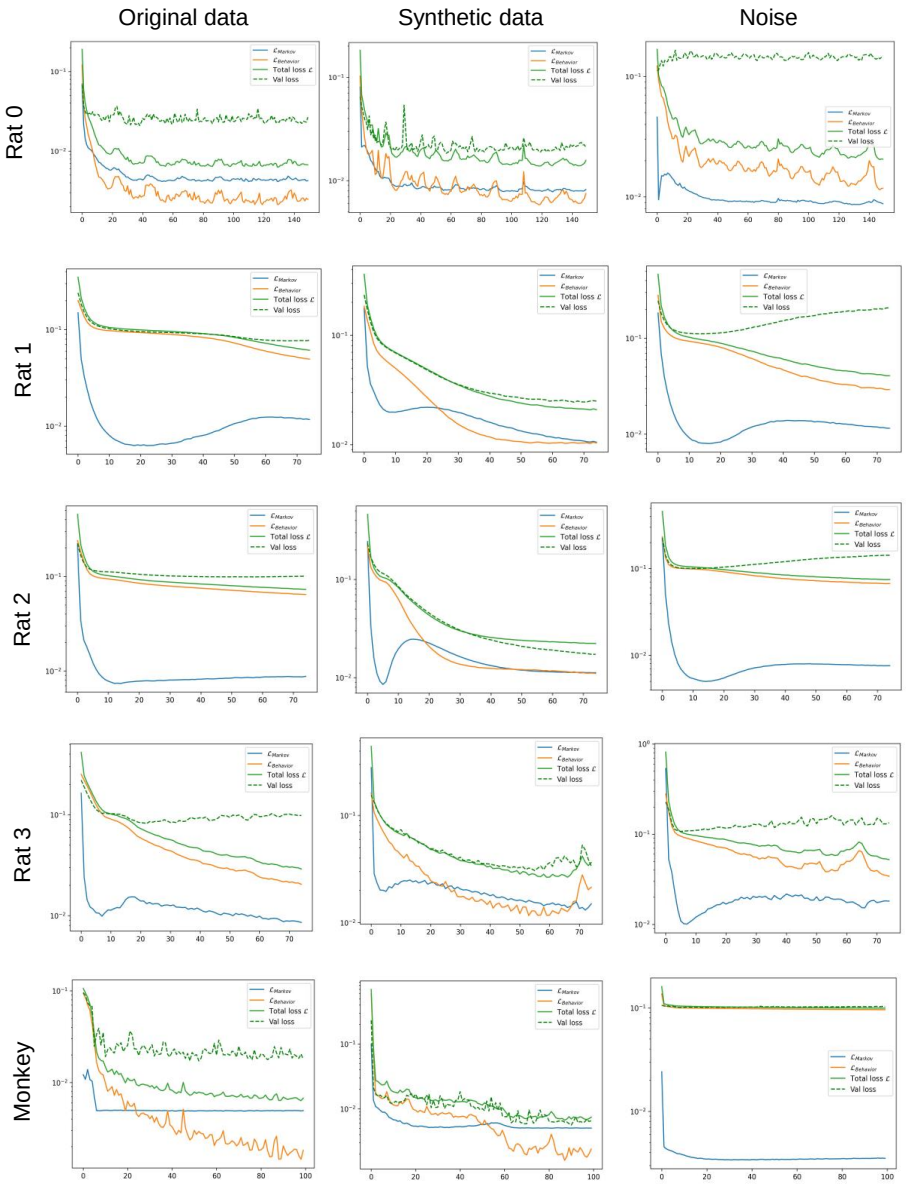

Figure 3.8.: Training and validation loss plots for models learned during across the three groups

Training and validation loss curves for BunDLe-Net across different behavioural groups (true data, synthetic data, and noise data) and datasets (Rats 0–3 and Monkey).

4. Discussion

4.1. Hyperparameter Tuning

In this thesis, I have shown that for five out of seven datasets the proposed HPO method of hyperparameter tuning led to an improvement in performance compared to the baseline model, Random Search, where the performance was measured as behaviour decoding performance.

While the proposed HPO approach generally outperformed the baseline across most datasets, there were two exceptions. One of these, the monkey dataset, which is used as a benchmark [72, 73] in neural manifold research, achieved high R^2 scores under both search strategies, despite having rather different optimal hyperparameter configurations. This could indicate that the monkey dataset may be robust to variations in hyperparameter tuning.

The second dataset, where the proposed HPO strategy underperformed compared to the Random Search, exhibited relatively low R^2 scores across both methods. This outcome could indicate that the dataset itself posed a challenge for modeling, the R^2 score's interpretability was affected by the problem's complexity, or there was high variability in the behavioural data [71]. Additionally, acceptable R^2 values vary and they are interpreted relative to a constant mean model, so the absolute R^2 score values should be interepreted with caution [71].

Nevertheless, the improvements in R^2 scores achieved by the proposed method were relatively modest. Additionally, both methods were implemented only once, whereas similar studies evaluating optimization effectiveness have repeated model runs multiple times (n=100) to compare against randomized grid search [51]. This difference in experimental setup may have influenced the observed outcomes, as a single run may not fully capture the variability in optimization performance.

One possible explanation for the modest improvements in R^2 cores is that Random Search, which explored all tuned hyperparameters simultaneously, has been shown to perform well in high-dimensional optimization problems [49]. In contrast, the proposed sequential approach adjusted only one (or a small subset) of hyperparameters at a time,

potentially missing important interactions among them. For instance, learning rate and batch size—two parameters often recommended to be scaled in tandem [74]—were tuned separately, which may have limited the method's ability to find a globally optimal configuration. A potential improvement could have been achieved by reducing the number of distinct hyperparameter groupings, allowing for better interaction modeling. Given that TPE has been shown to perform well even with hundreds of hyperparameters and a comparable number of trials [52], a more integrated tuning approach may have yielded better results.

Experiments exploring batch size effect on the model generalization

Small batch sizes resulted in noisier loss curves due to the increased number of weight updates and the inherent stochasticity in training, whereas larger batch sizes produced smoother curves (see Figure 3.4). However, when BunDLe-Net model was trained for Rat 0 dataset with varying batch sizes, the highest R^2 score was achieved with a batch size of 512 (see Table 3.4, while the hyperparameter optimization pipeline selected a batch size of 64 (see Table 3.1). This discrepancy highlights significant variability across training runs and suggests that a single trial may not capture the true optimal hyperparameter setting; averaging over multiple trials could yield different results.

Moreover, the generally smaller optimal batch sizes (B=64) for most rat datasets are consistent with the findings of [75], which demonstrated that small batches tend to produce flatter minima and better generalization. Thus, it is possible that the grid search approach—by aggregating performance over multiple trials—provided a more robust estimate of the optimal batch size, or alternatively, that the batch size selected during grid search was influenced by the previously chosen learning rate. In either case, considering these interactions might lead to further improvements in model performance.

Latent Dimension

For all datasets, the latent dimension refinement step resulted in a lower optimal latent dimension. This finding shows that selecting the latent dimension solely based on achieving the minimum validation loss is insufficient. Instead, an additional optimization step—focused on reducing the latent dimensionality—is necessary to accurately identify the intrinsic dimension of the latent representation.

4.2. Behavioural Validation

For all groups except noise the null hypothesis, that the circular shifting of behavioural data does not affect the model's performance, was rejected. This finding indicates that the model failed to learn a meaningful manifold when presented with data lacking any behaviorally relevant information, as expected, and confirms that in the absence of meaningful behavioral signals, the learned embedding remains at chance level.

In contrast, in the second control group using synthetic data, all manifolds were statistically significant. However, performance varied among datasets with some datasets yielding negative R^2 scores (see Table 3.5). This could be due to the fact that by construction the synthetic dataset may not have contained behaviourally relevant information. Synthetic behavioral data were derived from neural recordings, which inherently contain noise and signals unrelated to the behavior of interest. Since the actual behavioral signal in neural recordings is unknown, generating synthetic behaviorally relevant data is a challenging task.

Other studies have proposed methods for constructing synthetic neural signals encoding behaviour [73]. This method involved selecting neurons based on their behaviour decoding performance and training a deep learning model to extract the behaviorally relevant signal.

Loss curves for different groups

The gap between training and validation loss varied among the groups (see Figure 3.8. Notably, the noise dataset exhibited the largest gap, while the synthetic dataset had the smallest. A larger gap may indicate poorer generalization, whereas a smaller gap suggests better generalization.

4.3. Limitations and Future Work

Hyperparameter Optimization

The performance of the proposed HPO pipeline was compared to a baseline model, but it was not benchmarked against state-of-the-art HPO toolboxes such as Auto-LFADS [34]. This decision was made because Auto-LFADS is a low-fidelity model based on population-based training [76], and BunDLe-Net's loss function is composite. Therefore, low-fidelity model may affect the objective function that is being optimised [77] Notably, the state-of-the-art neural manifold learning model CEBRA [35] was optimized using

Auto-LFADS, with a publicly available script.

Validation

Another limitation of the proposed methodology is that the validation of the learned manifolds addresses only the behavioral component of the composite loss function, excluding the assessment of their dynamical properties. The manifolds learned by the BunDLe-Net model are Markovian by construction, and this property could be tested. One potential approach would be to compare the dynamical properties of manifolds learned by the BunDLe-Net model against those of scrambled manifolds, using deep learning models resembling the structure of the T_Y module.

Bibliography

[1] A. Kumar, A. Gilra, M. Gonzalez-Soto, A. Meunier, and M. Grosse-Wentrup, "Bundle-net: Neuronal manifold learning meets behaviour," *bioRxiv*, 2023.

[2] "Consideration of sample size in neuroscience studies," *Journal of Neuroscience*, vol. 40, no. 21, pp. 4076–4077, 2020.

[3] A. A. Faisal, L. P. J. Selen, and D. M. Wolpert, "Noise in the nervous system," *Nature Reviews Neuroscience*, vol. 9, pp. 292–303, 2008.

[4] S. Theodoridis and K. Koutroumbas, *Pattern Recognition.* Elsevier, 4th ed., 2009.

[5] R. N. D'souza, P.-Y. Huang, and F.-C. Yeh, "Small data challenge: Structural analysis and optimization of convolutional neural networks with a small sample size," *bioRxiv*, 2018.

[6] D. R. Humphrey, E. M. Schmidt, and W. D. Thompson, "Predicting measures of motor performance from multiple cortical spike trains," *Science*, vol. 170, pp. 758–762, 1970.

[7] A. P. Georgopoulos, J. F. Kalaska, R. Caminiti, and J. T. Massey, "On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex," *Journal of Neuroscience*, vol. 2, no. 11, pp. 1527–1537, 1982.

[8] S. Saxena and J. P. Cunningham, "Towards the neural population doctrine," *Current Opinion in Neurobiology*, vol. 55, pp. 103–111, 2019.

[9] R. B. Ebitz and B. Y. Hayden, "The population doctrine in cognitive neuroscience," *Neuron*, vol. 109, no. 19, pp. 3055–3068, 2021.

[10] B. B. Averbeck, P. E. Latham, and A. Pouget, "Neural correlations, population coding and computation," *Nature Reviews Neuroscience*, vol. 7, no. 5, pp. 358–366, 2006.

[11] J. P. Cunningham and B. M. Yu, "Dimensionality reduction for large-scale neural recordings," *Nature Neuroscience*, vol. 17, no. 11, pp. 1500–1509, 2014.

[12] L. Cayton, "Algorithms for manifold learning," Tech. Rep. 12(1-17):1, University of California at San Diego, 2005.

[13] B. A. Richards, T. P. Lillicrap, P. Beaudoin, Y. Bengio, R. Bogacz, A. Christensen, C. Clopath, R. P. Costa, A. de Berker, S. Ganguli, C. J. Gillon, D. Hafner, A. Kepecs, N. Kriegeskorte, P. Latham, G. W. Lindsay, K. D. Miller, R. Naud, C. C. Pack, P. Poirazi, P. Roelfsema, J. Sacramento, A. Saxe, B. Scellier, A. C. Schapiro, W. Senn, G. Wayne, D. Yamins, F. Zenke, J. Zylberberg, D. Therien, and K. P. Kording, "A deep learning framework for neuroscience," *Nature Neuroscience*, vol. 22, no. 11, pp. 1761–1770, 2019.

[14] e. a. Kuoch, "Neural manifold learning in behavioral neuroscience," in *Proceedings of Machine Learning Research*, 2024.

[15] P. T. Sadtler, K. M. Quick, M. D. Golub, S. M. Chase, S. I. Ryu, E. C. Tyler-Kabara, B. M. Yu, and A. P. Batista, "Neural constraints on learning," *Nature*, vol. 512, no. 7515, pp. 423–426, 2014.

[16] J. A. Gallego, M. G. Perich, R. H. Chowdhury, *et al.*, "Long-term stability of cortical population dynamics underlying consistent behavior," *Nature Neuroscience*, vol. 23, pp. 260–270, 2020.

[17] J. M. Lee, *Introduction to Smooth Manifolds*. Springer, 2nd ed., 2013.

[18] J. A. Gallego, M. G. Perich, L. E. Miller, and S. A. Solla, "Neural manifolds for the control of movement," *Neuron*, vol. 94, no. 5, pp. 978–984, 2017.

[19] J. A. Gallego, M. G. Perich, S. N. Naufel, *et al.*, "Cortical population activity within a preserved neural manifold underlies multiple motor behaviors," *Nature Communications*, vol. 9, p. 4233, 2018.

[20] E. H. Nieh, M. Schottdorf, N. W. Freeman, R. J. Low, S. Lewallen, S. A. Koay, L. Pinto, J. L. Gauthier, C. D. Brody, and D. W. Tank, "Geometry of abstract learned knowledge in the hippocampus," *Nature*, vol. 595, no. 7865, pp. 80–84, 2021.

[21] S. S. Kim *et al.*, "Ring attractor dynamics in the drosophila head direction circuit," *Science*, vol. 356, no. 6340, pp. 849–853, 2017.

[22] L. Petrucco, H. Lavian, Y. K. Wu, *et al.*, "Neural dynamics and architecture of the heading direction circuit in zebrafish," *Nature Neuroscience*, vol. 26, pp. 765–773, 2023.

[23] R. Mitchell-Heggs, S. Prado, G. P. Gava, M. A. Go, and S. R. Schultz, "Neural manifold analysis of brain circuit dynamics in health and disease," *Journal of Computational Neuroscience*, vol. 51, no. 1, pp. 1–21, 2023.

[24] M. T. Kaufman, M. M. Churchland, S. I. Ryu, and K. V. Shenoy, "Cortical activity in the null space: permitting preparation without movement," *Nature Neuroscience*, vol. 17, pp. 440–448, 2014.

[25] A. Luczak, P. Barthó, and K. D. Harris, "Spontaneous events outline the realm of possible sensory responses in neocortical populations," *Neuron*, vol. 62, no. 3, pp. 413–425, 2009.

[26] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, no. 11, pp. 559–572, 1901.

[27] W. S. Torgerson, "Multidimensional scaling: I. theory and method," *Psychometrika*, vol. 17, pp. 401–419, 1952.

[28] C. Fortunato, J. Bennasar-Vázquez, J. Park, J. C. Chang, L. E. Miller, J. T. Dudman, M. G. Perich, and J. A. Gallego, "Nonlinear manifolds underlie neural population activity during behaviour," *bioRxiv [Preprint]*, Apr 2024.

[29] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[30] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[31] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[32] L. McInnes and J. Healy, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint*, 2018.

[33] C. Pandarinath, D. J. O'Shea, J. Collins, R. Jozefowicz, S. D. Stavisky, J. C. Kao, E. M. Trautmann, M. T. Kaufman, S. I. Ryu, L. R. Hochberg, J. M. Henderson,

K. V. Shenoy, L. F. Abbott, and D. Sussillo, "Inferring single-trial neural population dynamics using sequential auto-encoders," *Nature Methods*, vol. 15, no. 10, pp. 805–815, 2018.

[34] M. R. Keshtkaran, A. R. Sedler, R. H. Chowdhury, *et al.*, "A large-scale neural network training framework for generalized estimation of single-trial population dynamics," *Nature Methods*, vol. 19, pp. 1572–1577, 2022.

[35] S. Schneider, J. H. Lee, and M. W. Mathis, "Learnable latent embeddings for joint behavioural and neural analysis," *Nature*, vol. 617, no. 7960, pp. 360–368, 2023.

[36] G. Grimmett and D. Stirzaker, *Probability and Random Processes*. Oxford University Press, 3rd ed., 2001.

[37] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning* (F. Hutter, L. Kotthoff, and J. Vanschoren, eds.), The Springer Series on Challenges in Machine Learning, pp. 3–33, Springer, Cham, 2019.

[38] T. Yu and H. Zhu, "Hyper-parameter optimization: A review of algorithms and applications," *arXiv*, 2020.

[39] A. Ng, "Improving deep neural networks: Hyperparameter tuning, regularization and optimization." Deeplearning.ai on Coursera, 2017.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[41] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[42] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[43] T. Tieleman and G. Hinton, "Lecture 6.5—rmsprop: Divide the gradient by a running average of its recent magnitude." COURSERA: Neural Networks for Machine Learning, 2012.

[44] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint*, 2017.

[45] I. Ullah and A. Petrosino, "About pyramid structure in convolutional neural networks," *arXiv preprint*, 2016.

[46] P. M. Rezende, J. S. Xavier, D. B. Ascher, G. R. Fernandes, and D. E. V. Pires, "Evaluating hierarchical machine learning approaches to classify biological databases," *Briefings in Bioinformatics*, vol. 23, no. 4, p. bbac216, 2022.

[47] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[48] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pp. 754–762, 2014.

[49] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[50] J. Mockus, V. Tiesis, and A. Zilinskas, "The application of bayesian methods for seeking the extremum," vol. 2, pp. 117–129, 1978.

[51] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems (NIPS)*, vol. 25, 2012.

[52] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. Cox, "Hyperopt: A python library for model selection and hyperparameter optimization," *Computational Science Discovery*, vol. 8, p. 014008, 07 2015.

[53] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*, pp. I–115–I–23, 2013.

[54] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[55] S. Kato, H. S. Kaplan, T. Schrödel, S. Skora, T. H. Lindsay, E. Yemini, S. Lockery, and M. Zimmer, "Global brain dynamics embed the motor command sequence of caenorhabditis elegans," *Cell*, vol. 163, no. 3, pp. 656–669, 2015.

[56] A. D. Grosmark, J. Long, and G. Buzsáki, "Recordings from hippocampal area ca1, pre, during and post novel spatial learning," tech. rep., CRCNS.org, 2016.

[57] A. D. Grosmark and G. Buzsáki, "Diversity in neural firing dynamics supports both rigid and learned hippocampal sequences," *Science*, vol. 351, pp. 1440–1443, 2016.

[58] R. H. Chowdhury, J. I. Glaser, and L. E. Miller, "Area 2 of primary somatosensory cortex encodes kinematics of the whole arm," *eLife*, vol. 9, p. e48198, 2020.

[59] Python Software Foundation, *Python 3.10.12 Documentation*, 2023. Accessed: Feb. 10, 2025.

[60] C. R. Harris *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[61] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, pp. 51–56, 2010.

[62] M. Waskom, "Seaborn: Statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021.

[63] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[64] P. Virtanen *et al.*, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020.

[65] M. A. Terpilowski, "scikit-posthocs: Pairwise multiple comparison tests in python," *Journal of Open Source Software*, vol. 4, no. 36, p. 1169, 2019.

[66] P. Moritz *et al.*, "Ray: A distributed framework for emerging ai applications," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[67] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[68] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[69] J. Ellson, E. R. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, "Graphviz—open source graph drawing tools," in *Graph Drawing*, pp. 483–484, Springer, 2002.

[70] O. J. Dunn, "Multiple comparisons using rank sums," *Technometrics*, vol. 6, no. 3, pp. 241–252, 1964.

[71] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*. Wiley, 5th ed., 2012.

[72] M. Safaie, J. C. Chang, J. Park, *et al.*, "Preserved neural dynamics across animals performing similar behaviour," *Nature*, vol. 623, pp. 765–771, 2023.

[73] Y. Li, X. Zhu, Y. Qi, and Y. Wang, "Revealing unexpected complex encoding but simple decoding mechanisms in motor cortex via separating behaviorally relevant neural signals," *eLife*, vol. 12, p. RP87881, 2023.

[74] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," in *International Conference on Learning Representations*, 2018.

[75] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint*, vol. arXiv:1609.04836v2, 2017.

[76] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, D. Hassabis, K. Kavukcuoglu, and R. Hadsell, "Population based training of neural networks," *arXiv*, 2017.

[77] F. Hutter, L. Kotthoff, and J. Vanschoren, eds., *Automated Machine Learning*. The Springer Series on Challenges in Machine Learning, Springer, 2019.

Acronyms

BO Bayesian Optimisation. 8

HPO Hyperparameter Optimization. 1, 5, 7, 19, 23

TPE Tree-structured Parzen Estimator. 8, 14, 15, 21, 34

A. Appendix

A.1. Hyperparameter optimisation results for all animal datasets

A.1.1. Rat 0

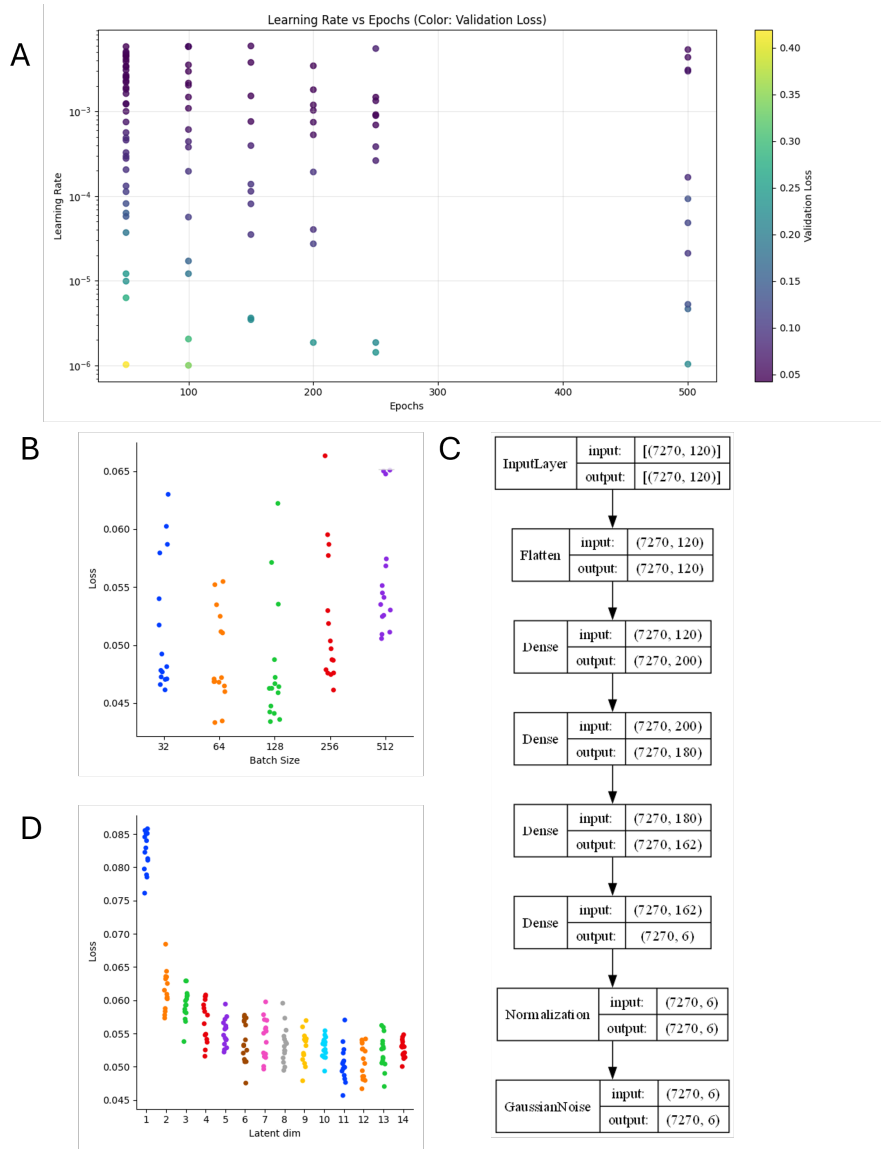

Figure A.1.: Optimal hyperparameter values for rat 0 dataset

A - learning rate and epochs using TPE algorithm, B- batch size search using grid search, C- network structure using TPE algorithm, D - latent dimension using grid search

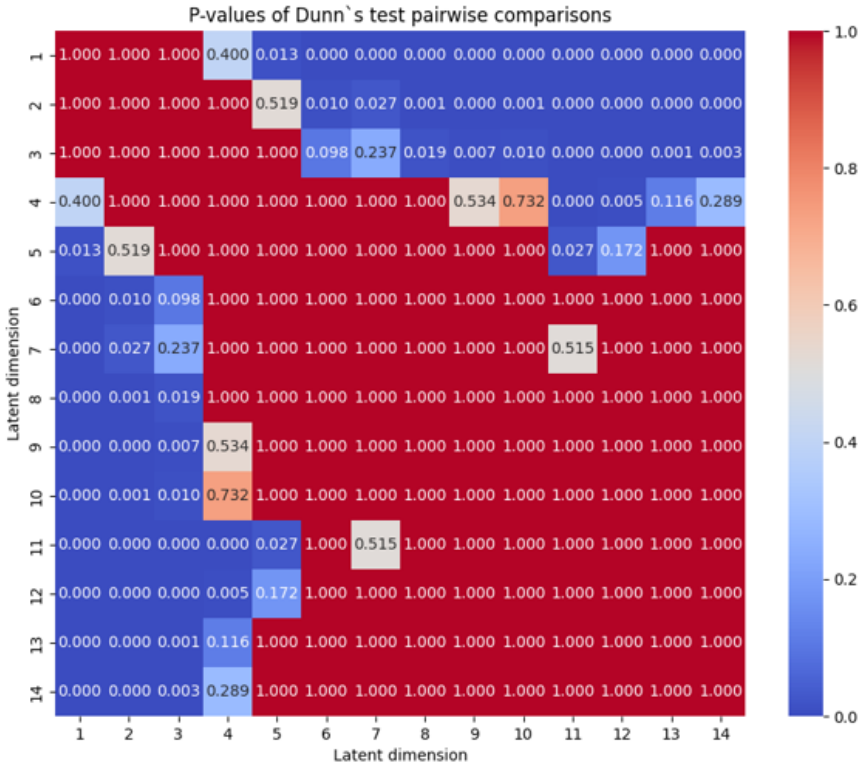

Figure A.2.: Pairwise comparisons of validation loss for different latent dimension - Rat 0

Heatmap of p-values of pairwise Dunn's test with Bonferroni of validation loss for different latent dimension values. The results are shown from Rat 0 dataset.

A.1.2. Rat 1

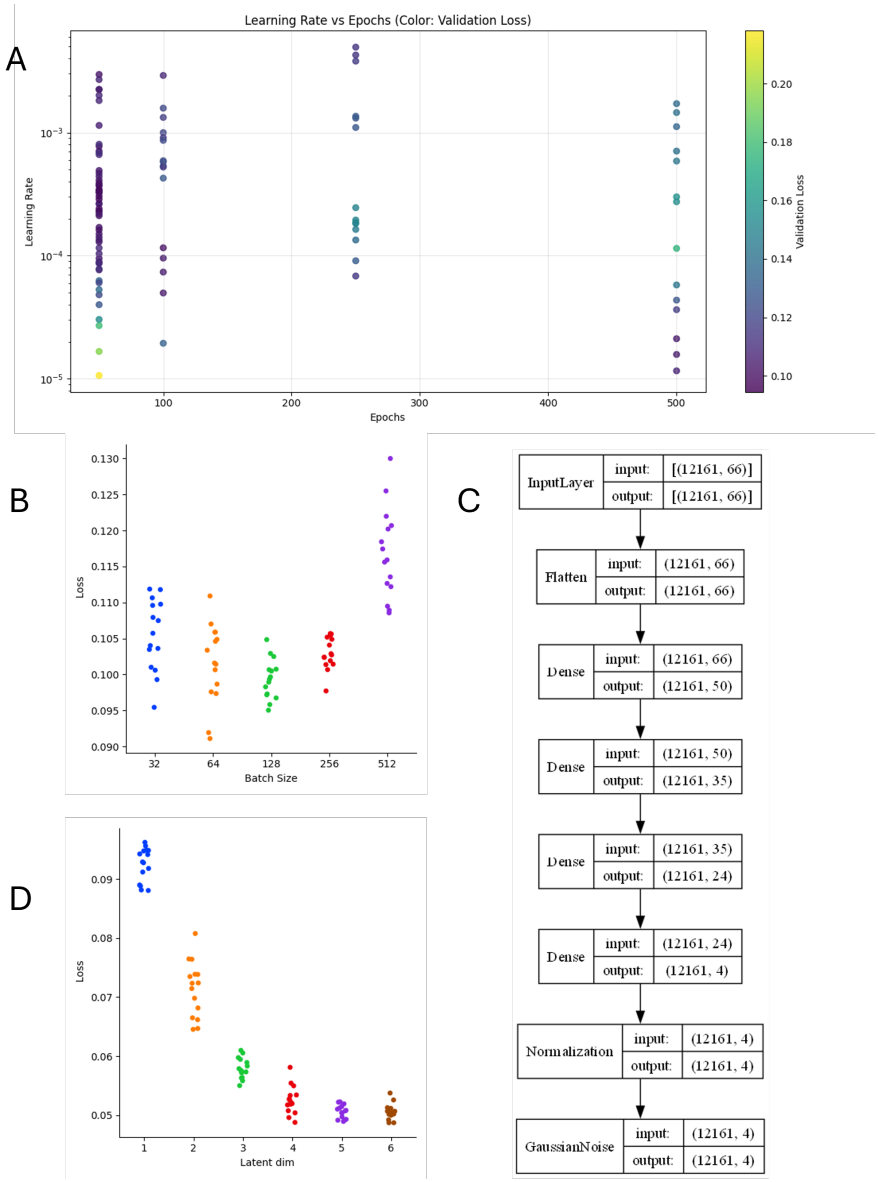

Figure A.3.: Optimal hyperparameter plots for rat 1 dataset

A - learning rate and epochs using TPE algorithm, B- batch size search using grid search, C- network structure using TPE algorithm, D - latent dimension using grid search

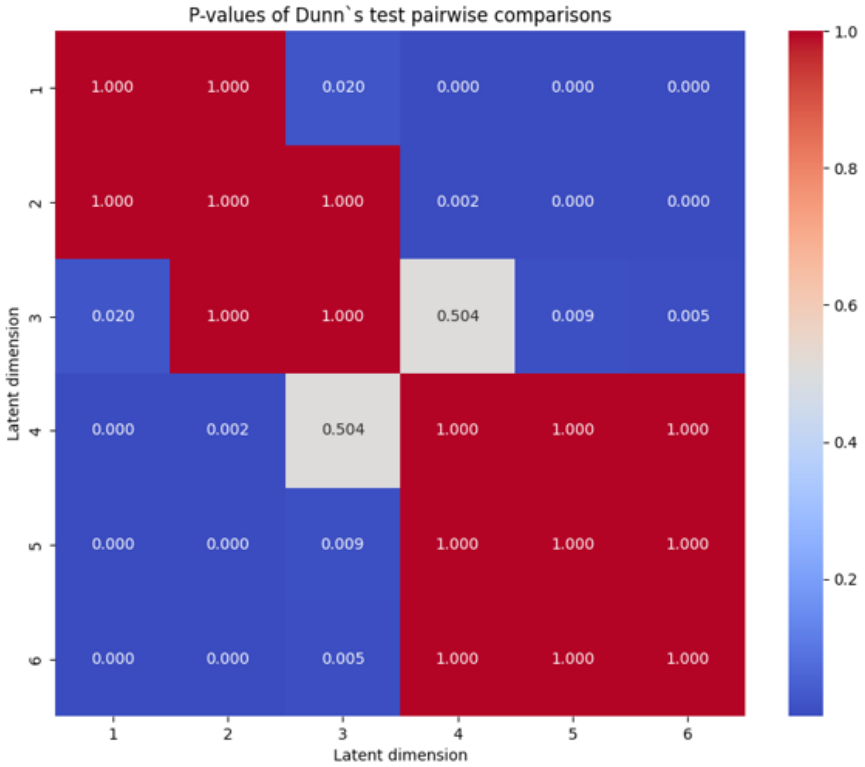

Figure A.4.: Pairwise comparisons of validation loss for different latent dimension - Rat 1

Heatmap of p-values of pairwise Dunn's test with Bonferroni of validation loss for different latent dimension values. The results are shown from Rat 1 dataset.

A.1.3. Rat 2

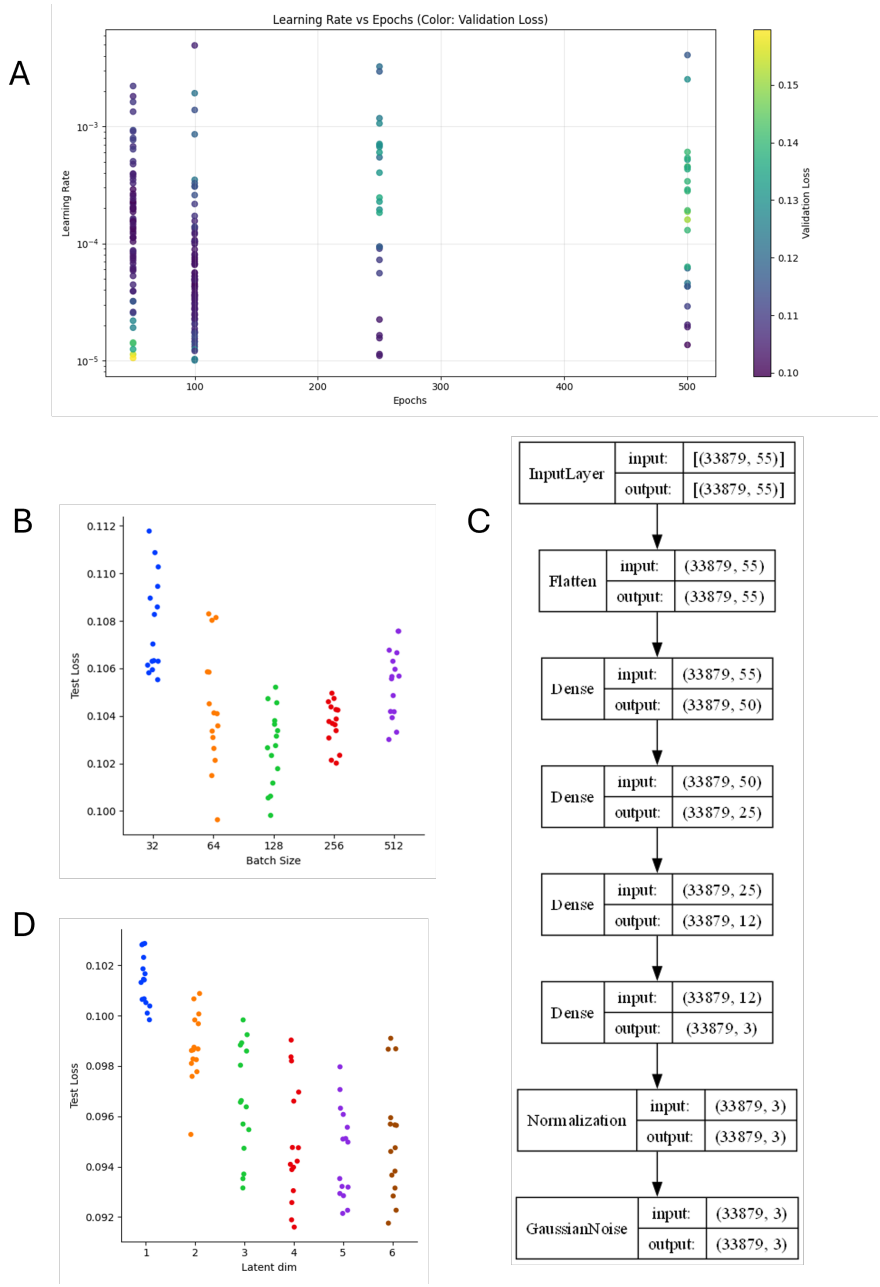

Figure A.5.: Optimal hyperparameter plots for rat 2 dataset

A - learning rate and epochs using TPE algorithm, B- batch size search using grid search, C- network structure using TPE algorithm, D - latent dimension using grid search

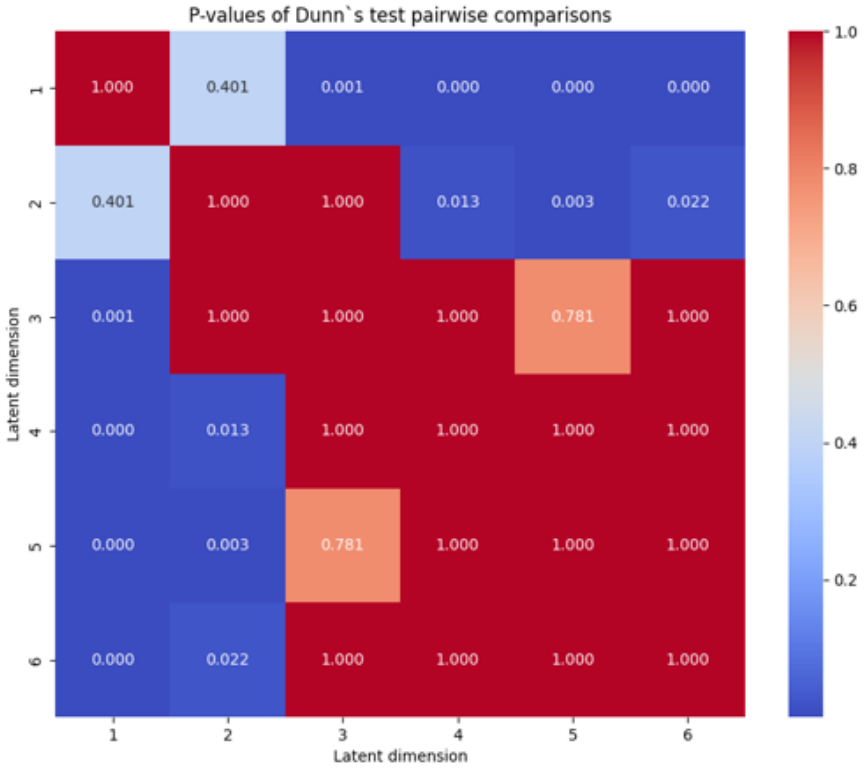

Figure A.6.: Pairwise comparisons of validation loss for different latent dimension - Rat 2

Heatmap of p-values of pairwise Dunn's test with Bonferroni of validation loss for different latent dimension values. The results are shown from Rat 2 dataset.

A.1.4. Monkey

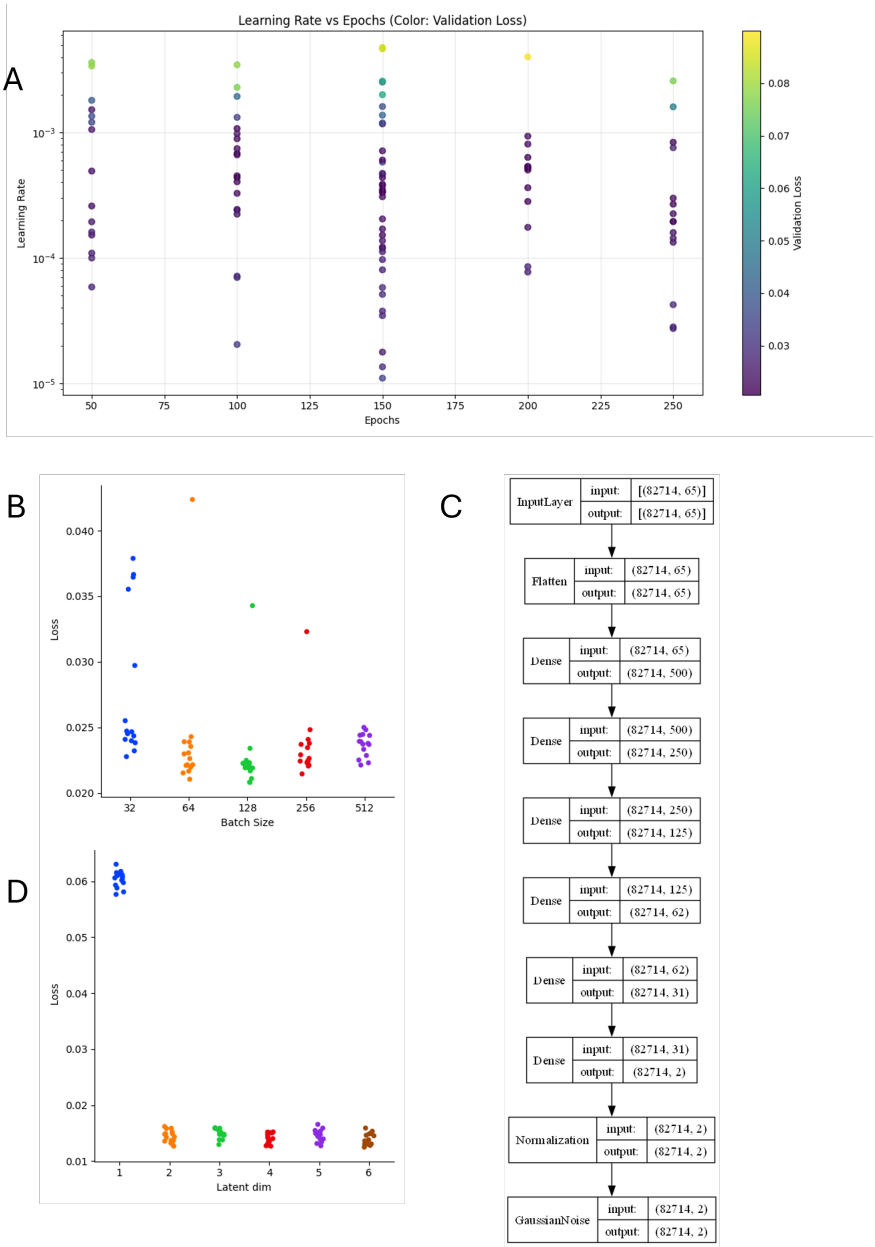

Figure A.7.: Optimal hyperparameter plots for monkey dataset

A - learning rate and epochs using TPE algorithm, B- batch size search using grid search, C- network structure using TPE algorithm, D - latent dimension using grid search

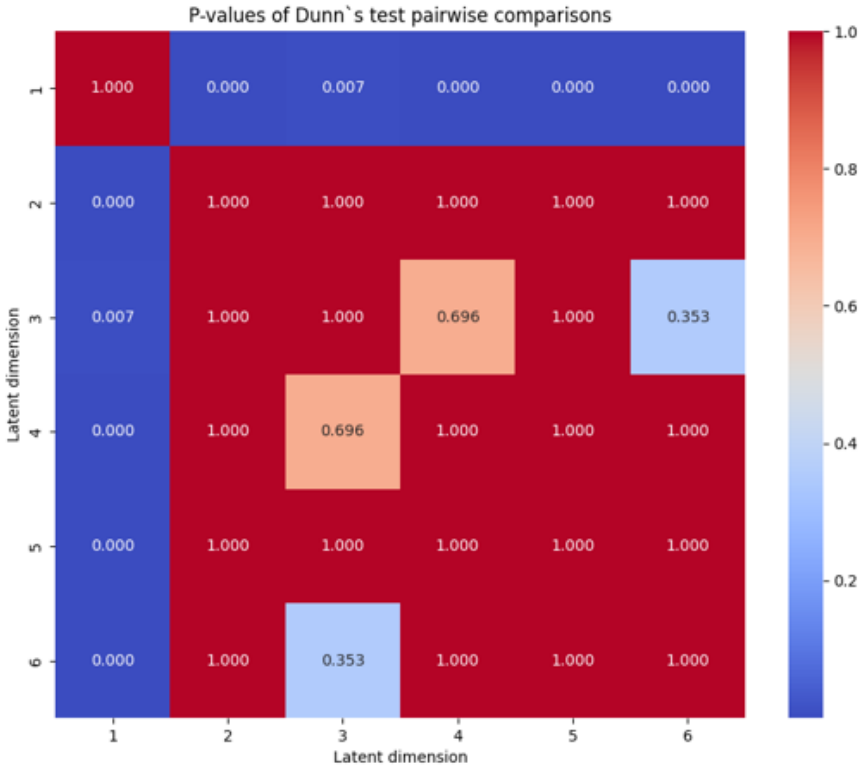

Figure A.8.: Pairwise comparisons of validation loss for different latent dimension -Monkey

Heatmap of p-values of pairwise Dunn's test with Bonferroni of validation loss for different latent dimension values. The results are shown from the monkey dataset.

Latent dimension (refinement) - Kruskal Wallis test results

Subject	Latent Dimension with Minimal Observed Validation Loss Value	Test Statistic	p-value
Rat 0	11	130.1459	1.96×10^{-21}
Rat 1	6	78.3994	1.81×10^{-15}
Rat 2	4	54.2564	1.86×10^{-10}
Rat 3	9	102.5965	5.20×10^{-16}
Worm 0	9	93.2936	3.55×10^{-16}
Worm 1	5	56.5720	2.23×10^{-10}
Monkey	3	26.8748	6.03×10^{-5}

Table A.1.: Test results for latent dimension refinement step

HPO toolbox latent dimension refinement step. The table presents results for each dataset after validating whether latent dimension groups originate from the same distribution using a statistical test. If $p > 0.05$, a post-hoc analysis was conducted (pairwise Dunn's test with Bonferroni correction).

A.1.5. Optimal Hyperparameter Configuration Obtained by Random Search

Dataset	**Epochs**	**Learning rate** (η)	**Batch size** (B)
Rat 0	50	0.0022	64
Rat 1	100	1.456e-05	32
Rat 2	200	1.268e-05	256
Rat 3	300	0.0012	32
Monkey	100	5.919e-05	32
Worm 0	100	0.0015	128
Worm 1	200	0.0017	128

Table A.2.: Random search: training related hyperparameters

Dataset	T_Y	τ: **Layers**	τ: **1st layer**	τ: **Pyramid factor**	d
Rat 0	Non-linear	6	500	0.5	14
Rat 1	Linear	8	50	0.9	6
Rat 2	Linear	8	100	0.9	10
Rat 3	Non-linear	4	500	0.7	10
Monkey	Non-linear	6	200	0.9	4
Worm 0	Non-linear	4	50	0.7	3
Worm 1	Non-linear	4	200	0.9	2

Table A.3.: Random search: network structure related hyperparameters

d - latent dimension; The number of layers in τ function referred in the table excludes regularization layers.

A.2. Behavioural validation

A.2.1. True versus Predicted Behavioural Variables

Rat 0

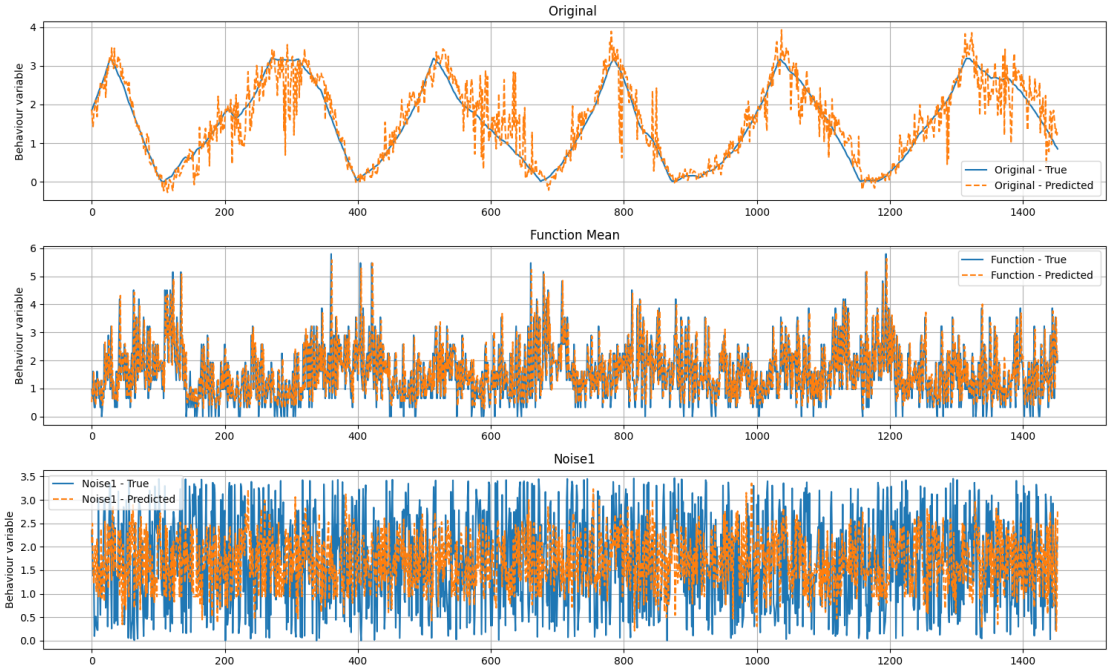

Figure A.9.: True and predicted behavioural variables across groups for Rat 0

$R^2 = 0.84 (p = 0.0) for group 1, R2 = -0.328 (p = 0.0) for group 2, R2 = -0.33 (p = 0.373) for group 3.$

Rat 1

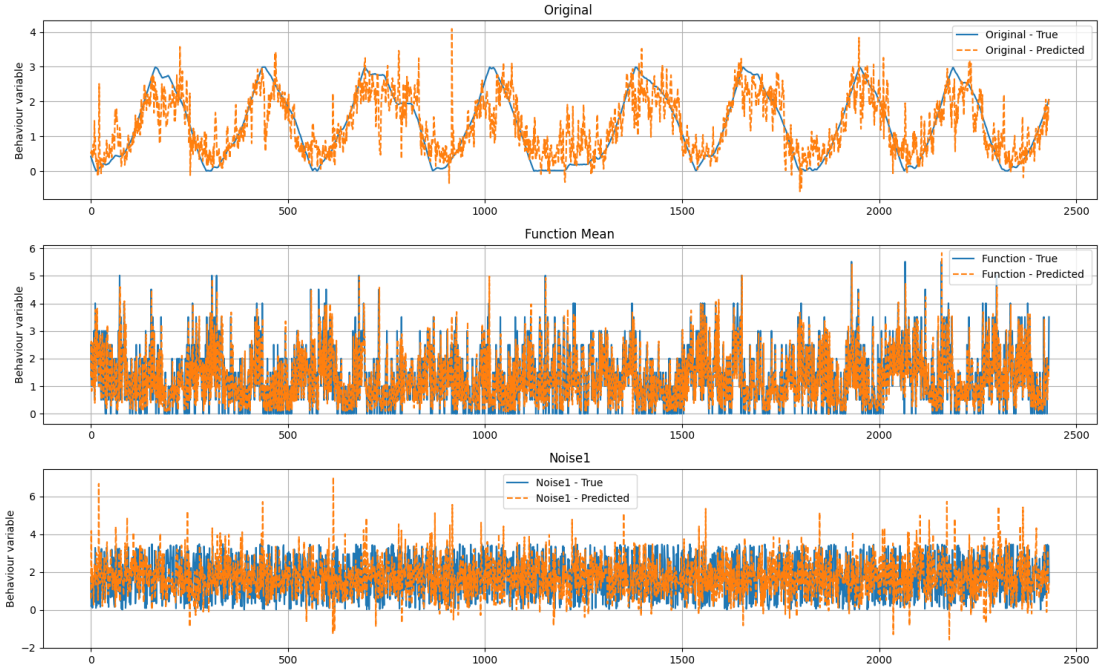

Figure A.10.: True and predicted behavioural variables across groups for Rat 1

R2 = 0.67 (p=0.0) for group 1, R2 = -0.44 (p=0.0) for group 2, R2=-0.87 (p=0.41) for group 3.

Rat 2

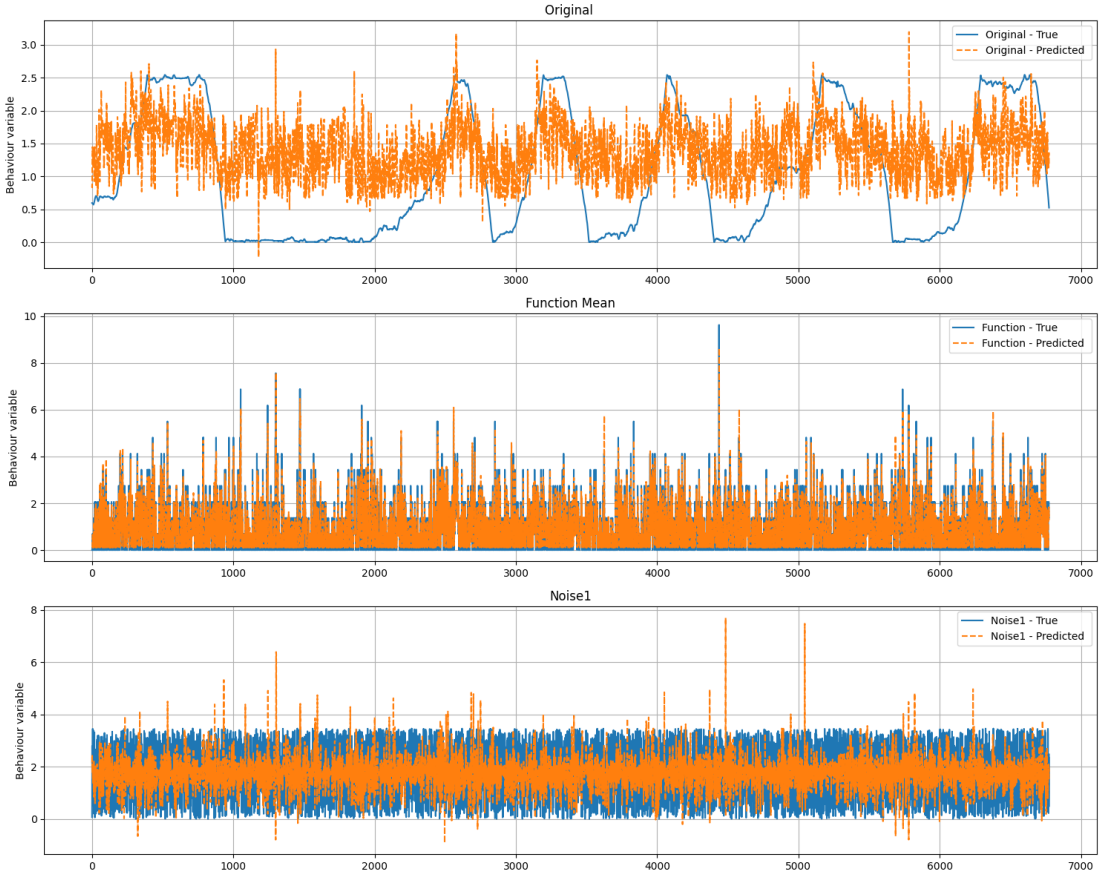

Figure A.11.: True and predicted behavioural variables across groups for Rat 2

R2 scores were R2 = 0.089 (p=0.0) for group 1, R2 = -0.36 (p=0.0) for group 2, R2=-0.29 (p=0.06) group 3.

Rat 3

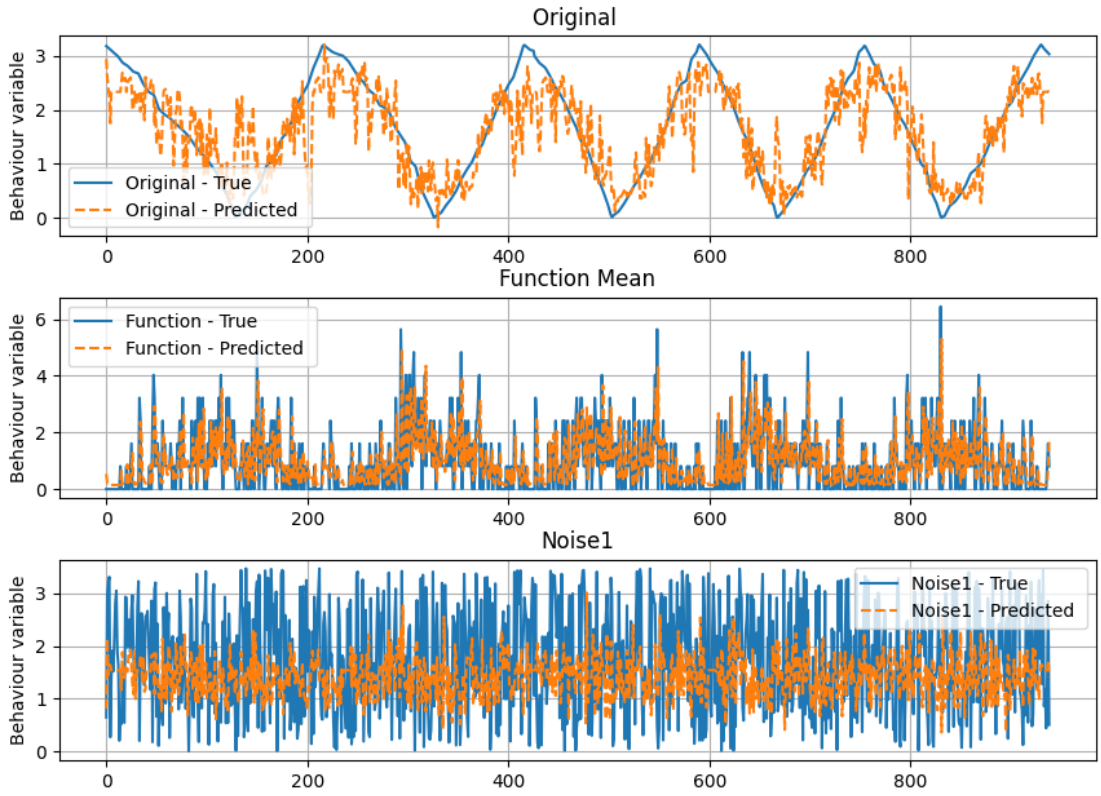

Figure A.12.: True and predicted behavioural variables across groups for Rat 3

The R2 values were R2 = 0.66 (p=0.0) for group 1, R2 = -0.18 group 2 (p=0.0), R2=-0.22 for group 3(p=0.41).

Monkey

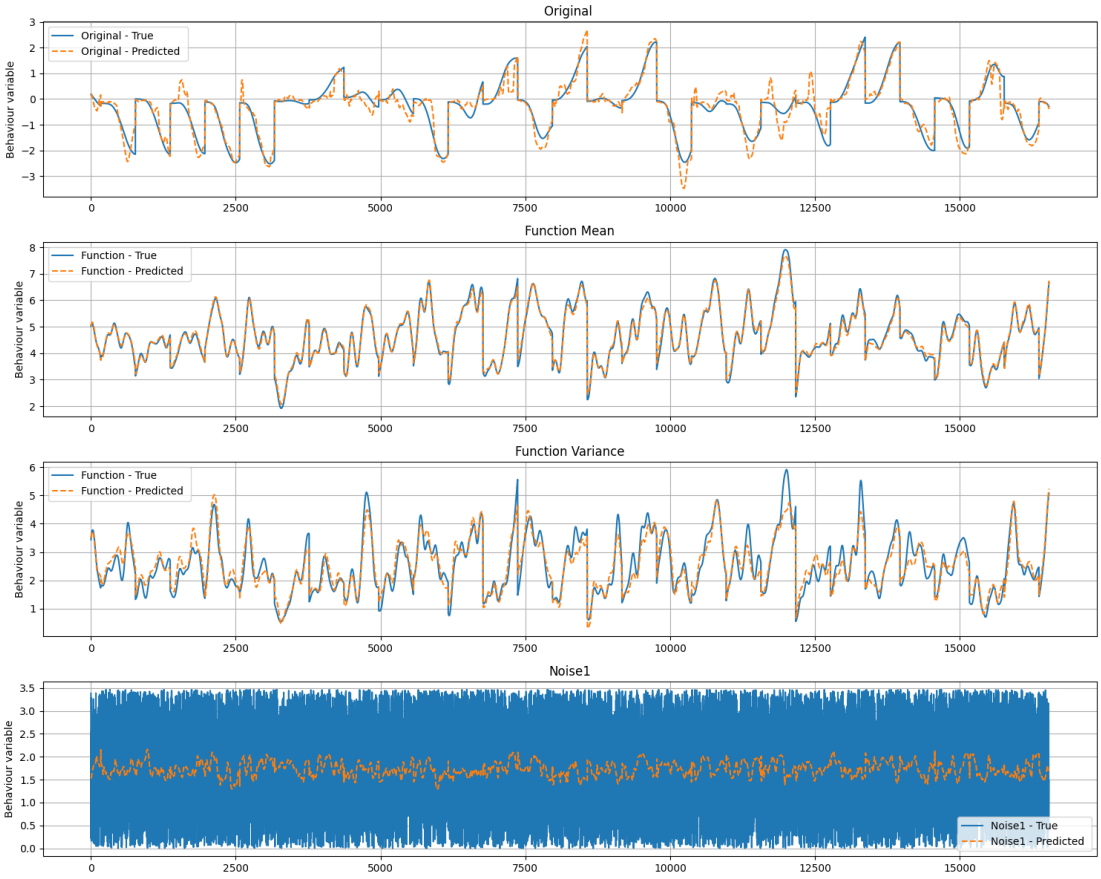

Figure A.13.: True and predicted behavioural variables across groups for monkey dataset

The R2 values were R2 = 0.83 (p=0.0) for group 1, R2 = 0.93 group 2 (p=0.0), R2=-0.02 for group 3(p=0.7).