Adaptive and Topological Deep Learning with Applications to Neuroscience

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Edward Mitchell

May 2023

© by Edward Mitchell, 2023
All Rights Reserved.

For Cass, Lottie, and Luna.
Your love means the world to me.

Acknowledgements

First I would like to thank my advisor, Professor Vasileios Maroulas, whose research group I joined at a crossroads in my graduate school career. His leadership and accessibility proved vital to my completion of this degree. My confidence in myself as a researcher and mathematician would not exist if not for his encouragement.

I would like to extend my gratitude to all of my co-authors Professor Andreas Aristotelous, Dr. Dave Boothe, Dr. Piotr Franaszczuk, and Dr. Brittany Story. I am proud of the work we accomplished together. I would also like to thank my committee member, Professor Ioannis Sgouralis. I appreciate his feedback and donation of his time to not just my dissertation defense, but also my oral exam.

Thank you to my colleagues and friends at the University of Tennessee, especially those who shared A109 with me our first year. Their willingness to collaborate saved me from early failure. Specifically, from this bunch, I must thank Dr. Jesse Sautel. I can not think of any other student or fellow worker that has ever lifted those around him like Jesse does.

I would like to thank my family and my friends from back home. To my parents that make life a little too easy, thank you. I feel your love, and I will pay it forward to Lottie in the hopes that she will be as grateful for me as I am for you. To my siblings that will not read past this page, thanks. Your genuine excitement for my academic accomplishments has always pushed me to achieve more. I am grateful for my friends Max Gallagly, Connor Gallagly, and Jake Iurato, whose visits and texts kept me sane through an oftentimes lonely and isolating experience. I would also like to offer a special thanks to my in-laws, the Beaches. Their support of not just me, but also Cass, cannot be overstated.

Finally, I must thank my number one supporter, my wife, Cass, who has stuck by my side during a grueling and thankless graduate school journey. Her commitment to me and our little family inspires me to be my best every day.

My PhD dissertation was partially funded by the US Army Research Lab Contract No W911NF2120186.

Abstract

Deep Learning and neuroscience have developed a two way relationship with each informing the other. Neural networks, the main tools at the heart of Deep Learning, were originally inspired by connectivity in the brain and have now proven to be critical to state-of-the-art computational neuroscience methods. This dissertation explores this relationship, first, by developing an adaptive sampling method for a neural network-based partial different equation solver and then by developing a topological deep learning framework for neural spike decoding. We demonstrate that our adaptive scheme is convergent and more accurate than DGM – as long as the residual mirrors the local error – at the same number of training steps and using the same or less number of training points. We present a multitude of tests applied to selected PDEs discussing the robustness of our scheme.

Next, we further illustrate the partnership between deep learning and neuroscience by decoding neural activity using a novel neural network architecture developed to exploit the underlying connectivity of the data by employing tools from Topological Data Analysis. Neurons encode information like external stimuli or allocentric location by generating firing patterns where specific ensembles of neurons fire simultaneously for one value. Understanding, representing, and decoding these neural structures require models that encompass higher order connectivity than traditional graph-based models may provide. Our framework combines unsupervised simplicial complex discovery with the power of deep learning via a new architecture we develop herein called a simplicial convolutional recurrent neural network (SCRNN). Simplicial complexes, topological spaces that use not only vertices and edges but also higher-dimensional objects, naturally generalize graphs and capture more than just pairwise relationships. The effectiveness and versatility of the SCRNN is demonstrated on head direction data to test its performance and then applied to grid cell datasets with the task to automatically predict trajectories.

Table of Contents

1 Introduction **1**

2 Background **5**

2.1 Neuronal Signaling 5

2.2 (Artificial) Neural Networks 6

3 The Adaptive Deep Learning Galerkin Method **10**

3.1 Deep Galerkin Method 10

3.2 Adaptive Deep Learning Galerkin Method 14

3.2.1 Method Description 15

3.2.2 Hyperparameter Sensitivity Analysis 16

3.2.3 Adaptive Variance 22

3.3 ADLGM vs DGM Comparisons 22

3.3.1 Poisson with Oscillatory Solution 24

3.3.2 Poisson with Squared Nonlinearity 24

3.3.3 Burgers' Equation 32

3.4 Notes on Residual as a Marking Strategy 36

3.4.1 ADLGM on Allen-Cahn Equation 36

3.4.2 ADLGM on Poisson on a Notch Domain 40

3.5 Cable Equation 43

4 A Topological Deep Learning Framework for Neural Spike Decoding **48**

4.1 Neural Decoding 49

4.2 Topological Data Analysis Background 51

4.3 Method 52

4.3.1 Pre-processing 52

4.3.2 SCRNN Input 54
4.3.3 SCRNN 54
4.4 Results 59
4.4.1 Head Direction Cells 59
4.4.2 Grid Cells 63

5 Conclusions **67**

Bibliography **70**

Appendix **80**
A Neural Decoding 80
A.1 Hyperparameter Tuning 80
A.2 Analysis of Different Train/Test Splits 85
A.3 Neural Data 85

Vita **90**

List of Tables

3.1 Error and loss measurements for the Poisson problem (3.6) with $n = 4$. Values reported are averages of 10 realizations, see Figure 3.3. The error and loss both increase with the number of points added per marked mean. 18

3.2 Error and loss measurements for the Poisson problem (3.6) with $n = 4$, and 8. Values reported are averages of 10 realizations. For $n = 4$ and 8, ADLGM improves accuracy and achieves a lower residual. 26

3.3 Error and loss measurements for Burgers' problem (3.9). Values reported are averages of 10 realizations. ADLGM improves accuracy and achieves a lower residual. 34

3.4 Error and loss measurements for the Cable equation (3.13). 46

4.1 The AAE and MAE on the training and testing data, as well as the total numbers of catastrophic errors. We observe the SCRNN produces the least amount of catastrophic errors. 61

4.2 The AED on the training data and the AED on the testing data. While the SCRNN achieves the lowest train AED, the RNN shows the best generalizability with its low test AAE. 65

1 A table comparing the different networks based on their trial with the lowest catastrophic error. All trials were executed with 100 epochs, threshold 30, learning rate 0.001, and dropout rate 0.2. For the NN Width, the three values denote the size of each NN Layer. 82

2 Search values used for hyperparameter tuning. All hyperparameter search tests were conducted with 15 minutes of training data and 5 minutes of testing data. 83

3 A table of hyperparameter values used during hyperparameter tuning for the grid cell decoding task. Following the same procedure as was done for the HD application, we used a manual trial-and-error search method to identify optimal hyperparameters that minimized AED. 84

List of Figures

2.1 (left) diagram of a FFNN with a two dimensional input, two hidden layers with five nodes, and a one dimensional output. (right) an Elman RNN with a two dimensional input and four dimensional hidden layer and hidden state, and four dimensional output. The hidden state is denoted by orange nodes. . 7

3.1 A schematic of the DGM network architecture with three DGM layers. . . . 12
3.2 Diagram displaying the four sub-layers and flow within a single DGM layer. . 13
3.3 Error and loss per episode measurements for the Poisson problem (3.6) with $n = 4$. The solid points and shaded region correspond to the mean and two standard deviations, respectively. The square root of the residual and error was measured on a uniform grid for 10 realizations of each points per mean value. The magnification of the last 2000 episodes reveals that adding only one point per mean performs best. 19
3.4 Absolute residual (left column) and absolute error (right column) plots for the Poisson problem (3.6) with $n = 4$ at episode 500 with sample points superimposed. Each row corresponds to the number of points added per marked mean. From top to bottom, we show 1, 3, 5, and 10 points points per mean. The residual and error, though of different orders of magnitude, mirror each other locally. 20
3.6 Loss per episode measurements for the Poisson problem (3.6) with $n = 4$ comparing plain adaptive sampling to adaptive variance sampling (ADLGM) for the test cases $C_b = 100, 800$. (top row): $C_b = 100$. (bottom row): $C_b = 800$. The solid points and shaded region correspond to the mean and two standard deviations, respectively. The right plots are a magnification of the last 2000 episodes. The plots show that ADLGM trains faster and the shaded regions reveal that the adaptive variance improves stability. 23

3.7 Solutions of the Oscillatory Poisson problem (3.6) generated by ADLGM. (left) $n = 4$, and (right) $n = 8$. 25
3.8 Error and loss per episode measurements on a logarithmic scale for the Poisson problem (3.6) with $n = 4$. The solid points and shaded region correspond to the mean and two standard deviations, respectively. (left): residual (right): error. The plots reveal that ADLGM again achieves better accuracy with improved stability. 27
3.9 Error and loss per episode measurements on a logarithmic scale for the Poisson problem (3.6) with $n = 8$. The solid points and shaded region correspond to the mean and two standard deviations, respectively. (left): residual (right): error. Similar to the $n = 4$ case, ADLGM achieves better accuracy with improved stability. 28
3.10 Solutions of the Poisson problem with square nonlinearity as in equation (3.8) generated by ADLGM. (left) $\alpha = 8$, and (right) $\alpha = 16$. 29
3.11 Residual and absolute error per episode on a logarithmic scale for comparison of ADLGM to DGM for the Poisson problem with square nonlinearity as in equation (3.8) with $\alpha = 8, 16$. The solid points and shaded region correspond to the mean and two standard deviations, respectively. (top row): $\alpha = 8$ (bottom row): $\alpha = 16$. ADLGM achieves better accuracy than DGM throughout training. 30
3.12 Residual and absolute error plots for the Poisson problem with square nonlinearity as in equation (3.8) with $\alpha = 16$ at episode 500. 31
3.13 Solution to Burgers' Equation (3.9) generated by ADLGM. 33
3.14 Absolute residual (left column) and absolute error (right column) plots for Burgers' problem (3.9) with sample points superimposed. From top to bottom, each row corresponds to episode 250 and 1000. Note the regions of highest magnitude for the residual and error converge to the same parts of the domain as training occurs. 35
3.15 Solution to Allen-Cahn Equation (3.10) generated by ADLGM. 37
3.16 Residual and error plots for (top row): ADLGM and (bottom row): DGM. We see that ADLGM achieves similar results to DGM. 38

3.17 Absolute residual (left column) and absolute error (right column) plots for the Allen-Cahn PDE (3.10) with sample points superimposed. From top to bottom, each row corresponds to episode 200 and 1000. Note the regions of highest magnitude for the residual and error do not mirror each other throughout all of training. 39
3.18 Solution to the Poisson problem on the notch domain (3.11) generated by ADLGM using the inverse inequality marking strategy. 41
3.19 Absolute residual (left column) and absolute error (right column) plots for the notch domain Poisson problem (3.11) with sample points superimposed. From top to bottom, each row corresponds to episode 250 and 1000. Note the regions of highest magnitude for the residual and error do not mirror each other throughout all of training. 42
3.20 Value of (3.12) (left column) and absolute error (right column) plots for the notch domain Poisson problem (3.11) with sample points superimposed. From top to bottom, each row corresponds to episode 250 and 1000. The inverse inequality marking adds points to the region of high error, but this does not lead to a more accurate model. 44
3.21 (left) Exact solution to the Cable equation (3.13) given by equation (3.14). (right) Solution obtained using ADLGM. 47

4.1 An overview of our framework. Neural data in the form of spike trains, represented by a raster plot, is defined on a simplicial complex via a pre-processing procedure, see Figure 4.2. The raster shown includes the activity of only five neurons for clarity. The simplicial complex gets input into a SCRNN, see Figure 4.3. Finally, the SCRNN decodes the desired variable(s). Here, we depict 2D location being decoded with the start and end locations of the trajectory depicted by a circle and star, respectively. 50

4.2 An example of the pre-processing procedure. Neural spiking data is represented as a raster plot on the left. The data is binned and converted to a spike count matrix. A row-wise thresholding procedure, given in Equation (4.2), binarizes the matrix. The binary matrix is displayed here with elements equal to 1 being colored in and the white elements conveying a 0. The colored regions within each column are then connected via the appropriate dimensional simplex to create the simplicial complex. For example, we see that the second column of the binarized matrix has three active neurons (green, orange, and blue). This generates a 2-simplex on the corresponding nodes. Note this allows for a multi-way description of these three nodes' relationship as opposed to the clique of 1-simplices that can only describe these nodes by their pairwise relationships. 53

4.3 A diagram of $L = 2$ simplicial convolutional layers each equipped with two filters, $H_k^1(1)$ and $H_k^2(1)$, for each simplicial dimension $k = 0, 1, 2$. Filters are color coded with filters that get concatenated bearing the same color as the resultant simplicial complex. In the first layer, we see three orange filters indicating the three dimensions of the input simplicial complex. The features extracted using these filters result in a new, orange simplicial complex. The second filter, depicted in light blue, extracts a separate simplicial complex. In the second simplicial convolutional layer, the process is repeated with two new filters, depicted by yellow and dark blue. In order to prevent exponential growth, features extracted from the same input from the previous layer are summed. Finally, all extracted features are summed and flattened to create one feature vector. 58

4.4 Plots depicting the true head angle and the predicted head angle for the first two minutes (left) and the catastrophic error for each time bin for the full twenty minutes (right) for four different networks, a) SCNN, b) FFNN, c) SCRNN, and d) RNN. 62

4.5 a)-c) Plots showing results from two minutes of the grid cell decoding task. a) Comparison of decoded versus ground truth x−coordinate. b) Comparison of decoded versus ground truth y−coordinate. c) Error for each time bin measured by equation (4.19). d) In grey is the ground truth position of the rat in the environment for all time bins used. Though we decode the entire trajectory shown in grey, for visual purposes, we include colorized paths showing a 5 second comparison of decoded versus ground truth position. . . 66

1 Plots depicting the true head angle and the predicted head angle for the first two minutes with the catastrophic error for each time bin for four different networks, a) SCNN, b) FFNN, c) SCRNN, and d) RNN. 86

2 Plots depicting the true head angle and the predicted head angle for the first two minutes with the catastrophic error for each time bin for four different networks, a) SCNN, b) FFNN, c) SCRNN, and d) RNN. 87

3 A raster plot showing the HD neural activity from 'Mouse28-140313' in the source data [1]. The HD system includes 22 neurons recorded over a 38 minute period while the mouse foraged in an open environment. 88

4 A raster plot showing the neural activity from a single module of 166 grid cells recorded over a 19 minute period while the rat foraged in an open 1.5×1.5 meter environment. The data is labeled R1 day 1 in the source data [2]. . . 89

Chapter 1

Introduction

Deep learning has seen a surge in popularity as accessibility to modern computational resources continues to grow and more complex models have been deployed. The main tool of deep learning is called an *(artificial) neural network* (NN), and since the implementation of NNs containing more parameters has become computationally feasible, larger datasets can be used for training, thus, leading to state-of-the-art results across a variety of tasks in a number of domains. Neural networks get their name from the similarities between their weights and activation functions and the axon-synapse-dendrite neuronal signaling in the mammalian brain. Some NN architectures take the neuroscience inspiration one step further in an attempt to mimic certain brain functions; for example, recurrent and gated connections can provide neural networks with a type of working memory helpful for dealing with time-series data [3]. The relationship between deep learning and neuroscience extends beyond just architecture. Common training techniques like dropout have drawn inspiration from neuronal activity, specifically the stochastic firing present in population activity. Deep learning informs areas of neuroscience either with direct applications, e.g., neural decoding [4, 5, 6] or EEG-based epilepsy detection, or by biologically restricting neural networks to experimentally observed connections and analyzing the networks dynamics as it performs a specific task [7, 8]. Thus, the inspiration drawn from neuroscience by deep learning gets reciprocated proving the pair of research areas are truly intertwined.

The symbiotic relationship is evident in the application of NNs to solving partial differential equations (PDEs), where NNs can be applied to PDEs relevant in computational neuroscience. In these applications, the NNs model the solution to the PDE. Some popular methods are called the Deep Ritz Method (DRM) [9], Physics-Informed Neural Networks

(PINN) [10], and the Deep Galerkin Method (DGM) [11]. Each method follows a similar algorithm of sampling the domain in a mesh-free manner and updating weights using a PDE-based loss function. However, the methods differ in sampling strategy, NN architectures, and the specific loss functions employed. In this work, we choose to focus on DGM due to its resampling for training iterations and its incorporation of the strong formulation of the PDE into the loss function. The use of meshless resampling helps avoid the curse of dimensionality and provides better coverage of the domain than methods that sample only once at the beginning of training. Other numerical methods for solving PDEs, like the finite difference method (FDM) or the finite element method (FEM), have seen improved results when certain adaptive sampling strategies have been employed, and, thus, it appears worthwhile to explore adaptive sampling strategies for DGM. These classical adaptive methods use what is called a marking strategy driven by an error indicator to identify regions of the domain that could benefit from additional sampling points. There exist some previous works that focus on adaptive sampling for NN-based PDE solvers. In [12, 13], additional points are sampled, but only a subset are marked and kept for training. Additionally, in [12], the domain is segmented across the temporal dimension and several networks are pieced together to get a solution for the entire domain. For the sake of efficiency, we choose to develop an adaptive sampling strategy that utilizes computations already necessary for training as our error indicators. Using computational efficiency and classical adaptive marking strategies as our motivation, we develop the *Adaptive Deep Learning Galerkin Method* (ADLGM). In this method, no extra points are wasted by not being used for training. ADLGM and its application to a number of PDEs are presented in Chapter 3 (published in the *Journal of Computational Physics* [14]). The final application of ADLGM featured in this work is to the Cable equation which bares significance in neuroscience as it helps model the membrane potential in axons and dendrites.

In Chapter 4, we take a task oriented approach to developing a topological deep learning framework for neural decoding. Decoding methods typically employ statistical or deep learning based models since one may view them as a regression problem where we learn the relationship between the dependent variable being decoded and the independent spike trains. Statistical methods like, but not limited to, linear regression, Bayesian reconstruction, and Kalman filtering are utilized for their interpretability and relatively low computational cost [4, 15, 1]. On the other hand, deep learning for neural decoding is a rapidly growing field due to neural networks' observed success at time-series tasks like sequence prediction as well

as neural networks' ability to generalize beyond training data [4, 16, 17, 18]. Neural networks have outperformed statistical methods at decoding head direction and two-dimensional, environment-based position from neural recordings of head direction (HD) cells and place cells, respectively [4, 5, 6]. Deep learning's superior decoding performance has been observed for a variety of network architectures including recurrent (RNNs) [19, 20], fully-connected feed forward (FFNNs), and convolutional neural networks (CNNs) [17, 21]. The smaller network sizes required for success in decoding compared to visual tasks allows for state-of-the-art performance on limited amounts of data [15].

Neurons in the brain form dense connections that lead to heavily correlated activity. Beyond these structural connections, higher-dimensional functional connectivity has been observed within groups of neurons exhibiting similar firing properties; for example, grid cells within a module [22]. Topological data analysis is comprised of techniques that heavily utilize objects called simplicial complexes and has been applied to real-world problems in different fields including, but not limited to, signal processing [23, 24, 25, 26], materials science [27, 28], biology [29, 30, 31], and chemistry [32, 33]. Simplicial complexes, topological spaces with the ability to describe multi-way relationships, naturally lend themselves to defining and encapsulating the hierarchical properties of neuronal data [22, 34], making them an increasingly popular tool for representing neural activity [35, 2, 36, 37, 38, 39, 40].

We propose a topological deep learning framework for neural spike train decoding by defining population activity on a simplicial complex that is then embedded in our new neural network architecture called a *simplicial convolutional recurrent neural network* (SCRNN). The neural activity is defined on a simplicial complex via a pre-processing procedure, which consists of binning the spikes to generate a spike count matrix that is then binarized, and active cells within a time bin are connected by a simplex. The construction of the simplicial complex makes no assumptions about the spike train's encoding, and the higher dimensional connectivity of the simplicial complex ameliorates feature representation. Note the pre-processing procedure does not require prior knowledge of the neural activity beyond spike counts, thus avoiding the high computational cost of computing properties such as wavelet representations, coactivity, or other similarity measurements. The SCRNN employs simplicial convolutional layers [41, 42, 43, 44] to extract features from the simplicial complex that are then fed to recurrent layers. We validate the framework on HD data by comparing results to those of simpler NN architectures that ignore the topology of the input data. Next, we apply the framework to the more difficult task of decoding two-dimensional position in an

environment from a population of grid cells. The higher dimension of the encoded variable requires more cells for encoding; hence, the task requires larger networks and is more difficult to perform.

The chapters are organized as follows. Chapter 2 provides a foundational understanding of the relevant deep learning tools used in this work. Chapter 3 describes ADLGM, an adaptive marking strategy for DGM. We show ADLGM can provide more accurate solutions in fewer training steps than baseline DGM. In Chapter 4, we develop a topological deep learning framework for neural decoding that combines tools from topological data analysis with the power of deep learning. We compare the method to common NN architectures on HD data. Finally, we apply the method to grid cell data, which, to our knowledge, is the first deep learning application to decoding recorded grid cell activity.

Chapter 2

Background

This chapter serves to detail aspects of, and elucidate parallels between, the electrical signaling in the mammalian brain and NNs. In section 2.1, we provide a brief overview of the different parts of a neuron and their roles in the electric message-passing system in the mammalian brain. Section 2.2 contains a review of the deep learning machinery employed throughout this dissertation.

2.1 Neuronal Signaling

Neurons are the main units of information-processing in the brain. A neuron consists of dendrites, a soma (cell body), and an axon; each part has a specific role in the transmission of signals throughout networks of neurons. *Dendrites* receive signals from many other neurons and pass these signals on to the *soma*. The soma sends these signals to the *axon hillock*, which sums them. If the accumulated signal strength exceeds a certain threshold, it triggers an action potential, or *spike*, and the axon hillock fires a signal down the axon. The signal is then transmitted to other neurons via *synapses*; this is where the axon terminals of one neuron communicate with the dendrites of another.

Biological neural networks are densely connected with some neurons receiving and sending signals to thousands of other neurons. How well two neurons communicate with one another is referred to as synaptic strength and is controlled by what is called *synaptic plasticity*. Synaptic plasticity is the weakening or strengthening of synapses based on their level of activity. Two neurons frequently communicating with one another see an increase in synaptic strength whereas two neurons rarely sharing signals learn to have a weaker connection.

2.2 (Artificial) Neural Networks

To avoid any confusion, let us remark here that, for the rest of this work, we will strictly refer to the population of neurons in the brain as a "biological neural network."

The main units of a NN are *perceptrons*. Perceptrons consists of weights and a bias term used to compute a weighted sum of its inputs that are then fed to an activation function, see Defintion 2.1 for a formal description.

Definition 2.1. *Let $W \in \mathbb{R}^m$, $b \in \mathbb{R}$, and $\sigma : \mathbb{R} \to \mathbb{R}$ be a nonlinear function. Then a perceptron receiving input $\boldsymbol{x} \in \mathbb{R}^m$ computes*

$$\sigma\left(W^T\boldsymbol{x} + b\right).$$

Note here the similarities between a neuron and perceptron: weights analogous to synaptic strength transform inputs received from dendritic structures that are then summed and only high enough values trigger a meaningful output from the nonlinear function.

Neural networks are formed by connecting perceptrons, using the output of one or several perceptrons as the input to others. One of the most basic NN architectures is a fully-connected feedforward neural network (FFNN), see Definition 2.2 and Figure 2.1. In a FFNN, perceptrons are stacked in layers where the output of each perceptron from the previous layer is fed to each perceptron in the following layer.

Definition 2.2. *For $\ell = 1, \ldots, L$, let $W_\ell \in \mathbb{R}^{n(\ell-1)\times n(\ell)}$, $\boldsymbol{b}_\ell \in \mathbb{R}^{n(\ell)}$, and let $\sigma : \mathbb{R} \to \mathbb{R}$ be a nonlinear function. Then a fully-connected feedforward neural network with $L-1$ hidden layers receiving input $\boldsymbol{x} \in \mathbb{R}^m$ computes*

$$\begin{aligned}
\boldsymbol{x}_0 &= \sigma\left(W_0^T\boldsymbol{x} + b_0\right), \\
\boldsymbol{x}_\ell &= \sigma\left(W_\ell^T\boldsymbol{x}_{\ell-1} + b_\ell\right), \quad \ell = 1, \ldots, L, \\
\boldsymbol{x}_{out} &= W_{L+1}^T\boldsymbol{x}_L + b_{L+1},
\end{aligned}$$

where σ is applied element-wise, $n(\ell)$ is the width of layer ℓ for $\ell = 1, \ldots, L$, $n(-1) = m$, and we take $n(L+1)$ to be the desired dimension of the output.

Two other NN architectures relevant to this dissertation; the Elman RNN [19], see Definition 2.3 and Figure 2.1, and the long short-term memory network (LSTM) [45], see

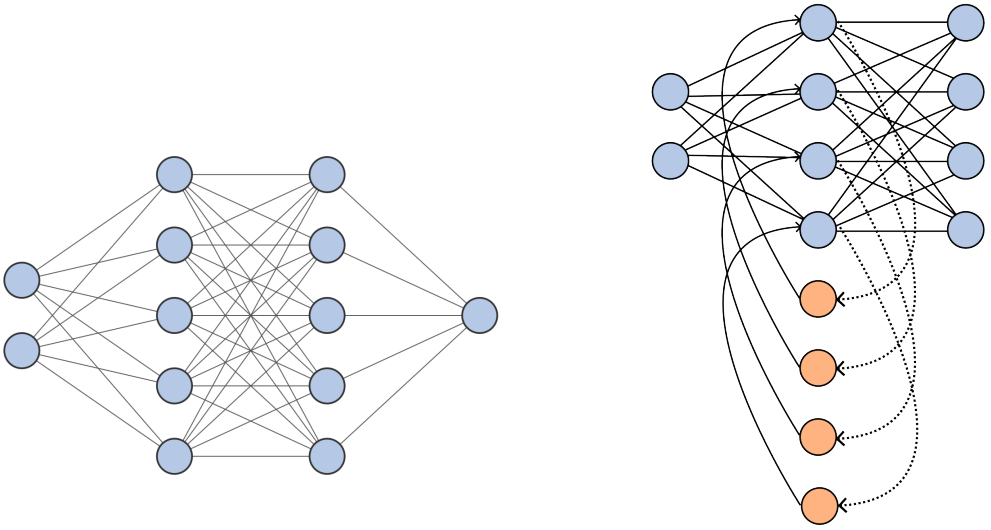


Figure 2.1: (left) diagram of a FFNN with a two dimensional input, two hidden layers with five nodes, and a one dimensional output. (right) an Elman RNN with a two dimensional input and four dimensional hidden layer and hidden state, and four dimensional output. The hidden state is denoted by orange nodes.

Definition 2.4; feature connections that help NNs handle time-series data by using recurrent connections.

Definition 2.3. *Let $W_h \in \mathbb{R}^{m\times n}$, $W_y, V_c \in \mathbb{R}^{n\times n}$, $b_h, b_c \in \mathbb{R}^n$, and let $\sigma : \mathbb{R} \to \mathbb{R}$ be a nonlinear function. Then an Elman RNN receiving input $\boldsymbol{x}_{\tilde{t}}$ from sequence $\{\boldsymbol{x}_t\}_{t=1}^T \subset \mathbb{R}^m$, for some $1 \leq \tilde{t} \leq T$ computes*

$$\begin{aligned} h_{\tilde{t}} &= \sigma\left(W_h^T \mathbf{x}_{\tilde{t}} + b_h + V_c h_{\tilde{t}-1} + b_c\right), \\ y_{\tilde{t}} &= \sigma\left(W_y h_{\tilde{t}} + b\right). \end{aligned}$$

where σ is applied element-wise, $h_0 = \mathbf{0} \in \mathbb{R}^n$, and $h_{\tilde{t}}$ is called a hidden state.

A multi-layer RNN is created by stacking multiple RNNs, feeding the outputs, $\{\boldsymbol{x}_t\}_{t=1}^T$, of one as the inputs to another. The hidden state gives the network a type of working memory similar to the mammalian brain.

An LSTM uses gated connections in addition to recurrent connections to mimic working memory. The gated connections are incorporated in order to give the network a sense of what information from the input sequence is relevant to the current time step.

Definition 2.4. *Let $W_f, W_i, W_o, W_c \in \mathbb{R}^{m\times n}$, $V_f, V_i, V_o, V_c \in \mathbb{R}^{n\times n}$, and $b_h, b_c \in \mathbb{R}^n$. Let $\sigma_s : \mathbb{R} \to \mathbb{R}$ be a sigmoid function and $\sigma_h : \mathbb{R} \to \mathbb{R}$ be the hyperbolic tangent function, i.e. $\sigma_h(x) = tanh(x)$. Then an LSTM cell receiving input $\boldsymbol{x}_{\tilde{t}}$ from sequence $\{\boldsymbol{x}_t\}_{t=1}^T \subset \mathbb{R}^m$, for some $1 \leq \tilde{t} \leq T$ computes*

$$\begin{aligned} f_{\tilde{t}} &= \sigma_s\left(W_f^T \mathbf{x}_{\tilde{t}} + V_f h_{\tilde{t}-1} + b_f\right), \\ i_{\tilde{t}} &= \sigma_s\left(W_i^T \mathbf{x}_{\tilde{t}} + V_i h_{\tilde{t}-1} + b_i\right), \\ o_{\tilde{t}} &= \sigma_s\left(W_o^T \mathbf{x}_{\tilde{t}} + V_o h_{\tilde{t}-1} + b_o\right), \\ \hat{c}_{\tilde{t}} &= \sigma_h\left(W_c^T \mathbf{x}_{\tilde{t}} + V_c h_{\tilde{t}-1} + b_c\right), \\ c_{\tilde{t}} &= f_{\tilde{t}} \odot c_{\tilde{t}-1} + i_{\tilde{t}} \odot \hat{c}_{\tilde{t}}, \\ h_{\tilde{t}} &= o_{\tilde{t}} \odot \sigma_h\left(c_{\tilde{t}}\right). \end{aligned}$$

where σ_s, σ_h are applied element-wise, $\odot$ denotes Hadamard (element-wise) multiplication, and $h_0 = \mathbf{0} \in \mathbb{R}^n$.

Network weights are computed using a training process with the ultimate goal of minimizing some chosen loss function. Unlike biological neural networks' employment of

synaptic plasticity, neural network parameters are updated via stochastic gradient descent or one of its variations. Despite this departure from biological relevance, there exist training techniques, outside of updating weights, that have drawn inspiration from observed behavior of the mammalian brain. One such technique, utilized in both Chapter 3 and 4, is *dropout*. When employing dropout, some desired percentage, p_{drop}, of network nodes, including the nodes' incoming and outgoing connections, are temporarily and randomly not included in the forward pass through the network. This mirrors the stochastic firing of neurons in the brains and is aimed to prevent overfitting and improve the generalization power of the network. Here, overfitting refers to a network that performs well on training data, but does not generalize to data not seen during training. By only including a subset of the network in different training steps, nodes of the network will not learn to correct the errors of other nodes, which can lead to overfitting.

Chapter 3

The Adaptive Deep Learning Galerkin Method

In this chapter, we develop ADLGM and test it against its baseline counterpart on a variety of PDEs. In section 3.1, we describe DGM. Section 3.2 details ADLGM and its features. Section 3.3 contains comparisons of ADLGM and DGM on several PDEs over different domains. In section 3.4, we remark on the effectiveness of using the residual as an error indicator for a marking strategy. In section 3.5, we directly apply our ADLGM algorithm to the Cable equation, a popular PDE in neuroscience used to model membrane potential throughout dendritic trees.

3.1 Deep Galerkin Method

Consider a PDE of the form,

$$\begin{aligned} \mathcal{N}(u(x,t)) &= f(x,t), && x \in \Omega, t \in (t_0, T] \\ \mathcal{B}(u(x,t)) &= 0, && x \in \partial\Omega, t \in (t_0, T] \\ \mathcal{I}(u(x,t_0)) &= 0, && x \in \Omega \end{aligned} \tag{3.1}$$

where $\mathcal{N}$ is a general differential operator and $\Omega \in \mathbb{R}^d$ is open and bounded. In DGM, the solution to the PDE (3.1) is approximated by a neural network $U(x,t;\theta)$ with parameters θ, which are learned through training. The training process of DGM involves sampling from the spatiotemporal domain, evaluating a loss function, and updating the network parameters

according to a preferred optimization scheme, typically some form of gradient descent. The goal of training is to minimize the loss function, which for DGM, incorporates the strong form of the PDE (3.1) and is defined by

$$LF_{DGM} = LF_r + LF_b + LF_0, \tag{3.2}$$

where, for $\{\mathbf{x}_n\} = \{(x_n, t_n)\} \subset \Omega \times (t_0, T]$, $\{\mathbf{x}_{b,n}\} = \{(x_{b,n}, t_n)\} \subset \partial\Omega \times (t_0, T]$, and $\{\mathbf{x}_{0,n}\} = \{(x_n, t_0)\} \subset \Omega \times \{t_0\} \subset \mathbb{R}^{d+1}$,

$$\begin{aligned} LF_r &:= MSE\left[\mathcal{N}(U(\{\mathbf{x_n}\};\theta)) - f(\{\mathbf{x}_n\})\right], \\ LF_b &:= MSE\left[\mathcal{B}(U(\{\mathbf{x_{b,n}}\};\theta))\right], \\ LF_0 &:= MSE\left[\mathcal{I}(U(\{\mathbf{x_{0,n}}\};\theta))\right], \end{aligned} \tag{3.3}$$

and MSE denotes the mean square error.

DGM employs a specific network architecture, shown in Figure 3.1, to approximate the solution. The hidden portion of a DGM network consists of what we refer to as DGM layers followed by one fully connected layer. Each DGM layer contains four sub-layers, depicted in Figure 3.2.

For DGM, forward propagation for a given input $\mathbf{x} = (x, t) \in \mathbb{R}^{d+1}$ through a network $U(\mathbf{x};\theta)$ with $L+1$ hidden layers is as follows [11]:

$$\begin{aligned} S^1 &= \sigma(W^1\mathbf{x} + b^1), \\ Z^l &= \sigma(V^{z,l}\mathbf{x} + W^{z,l}S^l + b^{z,l}), \quad l = 1, \ldots, L, \\ G^l &= \sigma(V^{g,l}\mathbf{x} + W^{g,l}S^l + b^{g,l}), \quad l = 1, \ldots, L, \\ R^l &= \sigma(V^{r,l}\mathbf{x} + W^{r,l}S^l + b^{r,l}), \quad l = 1, \ldots, L, \\ H^l &= \sigma(V^{h,l}\mathbf{x} + W^{h,l}(S^l \odot R^l) + b^{h,l}), \quad l = 1, \ldots, L, \\ S^{l+1} &= (1 - G^l) \odot H^l + Z^l \odot S^l, \quad l = 1, \ldots, L, \\ U(\mathbf{x};\theta) &= WS^{L+1} + b, \end{aligned} \tag{3.4}$$

where $\odot$ denotes Hadamard (element-wise) multiplication, $\sigma : \mathbb{R}^M \to \mathbb{R}^M$ is the element-wise activation function, M is the number of units per sub-layer, and $\theta = \{V, W, b\}$ with the differing superscripts are the model parameters.

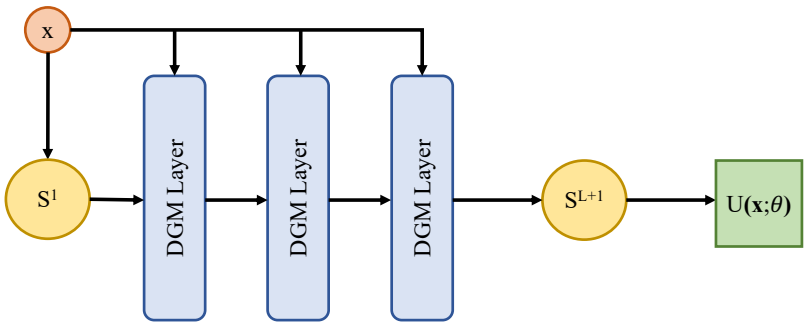


Figure 3.1: A schematic of the DGM network architecture with three DGM layers.

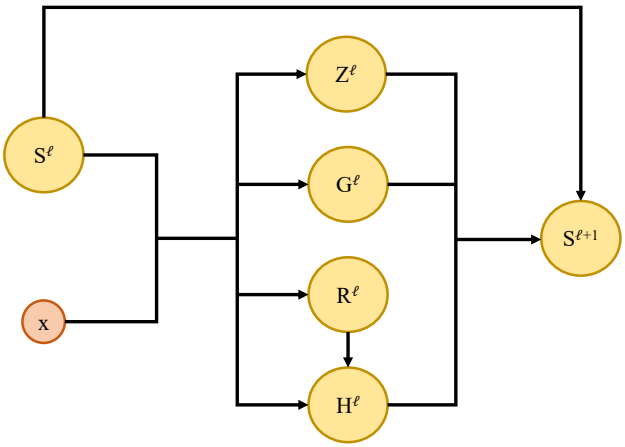


Figure 3.2: Diagram displaying the four sub-layers and flow within a single DGM layer.

As can be observed in Figure 3.2 and the equations listed in (3.4), a DGM layer consists of four sub-layers, denoted $Z, G, R,$ and H, each with its own two weight matrices and a single bias vector, totaling eight weight matrices and four bias vectors per DGM layer. These extra parameters help balance the effort between avoiding vanishing gradients and capturing the complex behavior of certain solutions, something not explicitly addressed in the architecture of a simple feed-forward network.

Algorithm 1: DGM [11]

initialize the network $U(\cdot;\theta)$;
while $i \leq i_{max}$ **do**
  Sample $\{\mathbf{x}_j\} \subset \overline{\Omega} \times [t_0, T]$ according to chosen probability densities;
  Calculate loss at the randomly sampled points $LF_{DGM}(\mathbf{x}_j)$;
  Perform one step of gradient descent at the randomly sampled points;
end

3.2 Adaptive Deep Learning Galerkin Method

In this section, we develop an adaptive sampling framework which optimizes the training points for that method. We are inspired by the idea of adaptive mesh refinement [46] techniques, mostly used in traditional numerical PDEs and specifically the finite element method and its derivatives [47]. This technique improves the accuracy and efficiency of a base numerical method by better resolving the computational solution in parts of the spatiotemporal domain where the PDE solution has "difficult" features that require more information. The method of identifying and marking those regions that require more information, is called a marking strategy, and this marking strategy uses an error indicator function to drive it. The marked regions are then enriched with more information via adaptive mesh refinement. The adaptive sampling proposed is efficiently applied to the DGM algorithm, which constantly needs resampling of the training points set. The method incurs no additional significant overhead since it does not require the resampling of a second set of points just for adaptivity purposes, where only few are eventually chosen, although the error indicator is calculated on all. The error indicator driving the adaptive procedures is the loss function used in DGM already, which means no additional calculations are required. The marking strategy uses an efficient standard sorting routine, and the "additional" points are introduced following the same spirit of the refinement idea used in traditional finite element

methods when local refinement is performed but without the extra overhead. The batch size does not grow larger than a certain max size, which can be chosen *a priori*, because of the way we are implementing the marking and the resampling; hence, the computational cost is even less than the DGM with the same max number of training points which we use to compare it with. So our method really improves DGM by making it more efficient and more accurate.

3.2.1 Method Description

Taking the square root of a loss function is a common machine learning technique that can help with updating weights during training and was shown to improve the process of solving PDEs using neural networks [48]. Another machine learning and optimization technique is to employ penalty parameters to emphasize specific terms of interest within the loss function [12, 13]. For these reasons, we define our loss function, LF, to be,

$$LF := \sqrt{C_r LF_r + C_b LF_b + C_0 LF_0}, \tag{3.5}$$

where $C_r, C_b, C_0 > 0$ are the penalty parameters for the PDE residual term, the boundary and initial conditions terms as defined in equation (3.3), respectively.

In DGM, one simply samples points according to preferred probability densities and uses those points for one training iteration. For our method, we use the loss function of DGM, equation (3.2), as an error indicator to identify regions in need of refinement and generate extra points in these parts of the domain suffering from high loss values. We first sample a fixed amount of uniformly distributed points, and we calculate the loss function value on the existing uniformly sampled spatiotemporal training points $\{\mathbf{x}_\ell\}_{\ell=1}^N \subset \overline{\Omega} \times [t_0, T]$. We note here that this is already done as a part of the base DGM algorithm, so no extra computational cost is incurred for this step. We then insert the contribution of each training point to the loss function, $LF_\ell := LF(\mathbf{x}_\ell)$, where

$$LF(\mathbf{x}_\ell) = \begin{cases} LF_r & , \ \mathbf{x}_\ell \in \Omega \times (t_0, T] \\ LF_b & , \ \mathbf{x}_\ell \in \partial\Omega \times (t_0, T] \\ LF_0 & , \ \mathbf{x}_\ell \in \Omega \times \{t_0\} \end{cases},$$

in a vector and sort it from high to low. We choose the points with the higher loss function values by following the classical idea in [49], i.e. by choosing only the loss function contributions from the first $k < N$ points, where the sum of those k is a certain prescribed percentage p of the global loss function value LF. That is,

$$\sum_{\ell=1}^{k} LF_\ell \leq p\ LF.$$

We use each one of the original, marked training points $\{\mathbf{x}_\ell\}$ as the mean of a multivariate normal distribution with covariance matrix $\Sigma = v_0 \mathrm{Id}$, where $\mathrm{Id} \in \mathbb{R}^{(d+1)\times(d+1)}$ is the identity matrix, to increase the resolution of the data set around that point. In this work, an episode refers to a single sampling trained for a fixed number of ADAM optimization steps. We call each of these ADAM training steps an iteration. At every episode, we repeat the adaptive sampling process until training is complete (see Algorithm 2). That is, we resample the fixed uniform points and also sample the extra points adaptively where is needed. In this way, we are naturally defining our adaptive framework in an efficient manner following the spirit of classical adaptive methods.

Algorithm 2: Adaptive Sampling DGM

initialize the network $U(\cdot, \theta)$;
while $LF > tol$ *and* $i < i_{max}$ **do**
  Sample $\{\mathbf{x}_j\}$ in $\bar{\Omega} \times [t_0, T]$ uniformly;
  Calculate loss function contribution $LF(\mathbf{x}_j)$ at each point;
  Insert loss at each point in vector $\{LF_\ell\}_{\ell=1}^{N}$ and sort from high to low;
  if $\sum_{\ell=1}^{k} LF_\ell \leq p\ LF$ **then**
    for each uniform point $\mathbf{x}_\ell$, $\ell = 1 \dots k$;
    generate m_ℓ additional points s.t. $\mu_\ell \sim \mathcal{N}(\mathbf{x}_\ell, v_0 \mathrm{Id})$;
  end
  Use the enriched training set to find $U(\mathbf{x}, \theta)$;
  using one episode of DGM[11] by doing a certain fixed number of training steps;
end

3.2.2 Hyperparameter Sensitivity Analysis

We perform hyperparameter analysis to understand the sensitivity of our method to the number of additional points sampled per marked mean. For this, we select the following

Poisson problem

$$\begin{aligned} -\Delta u(x,y) &= 2n^2\pi^2 \sin(n\pi x)\sin(n\pi y), && \text{in } \Omega \\ u(x,y) &= 0, && \text{on } \partial\Omega \end{aligned} \tag{3.6}$$

where $\Omega = (0,1)^2$, which has known oscillatory solution

$$u(x,y) = \sin(n\pi x)\sin(n\pi y).$$

Specifically, we solve the problem for $n = 4$ by employing a network with the DGM architecture containing $L = 2$ layers (3 hidden layers total) and $M = 16$ units per sub-layer. In this work, although ADLGM may be applied to initial and boundary points, we only perform adaptive sampling on the interior of the domain, effectively replacing LF with LF_r in Algorithm 2. We resample 500 interior points with 2,000 boundary points at every episode. Each episode consists of 10 iterations. At each resampling the interior points accounting for $p = 20\%$ of the interior loss are used as the means of normal distributions for additional points with a fixed value $v_0 = 0.001$ for each chosen mean. For penalty terms, we use $C_r = 1$ and $C_b = 800$. The high penalty term and oversampling on the boundary helps compensate for the adaptivity on the interior. Here we test solving eq. (3.6) with 1, 3, 5, and 10 additional points per mean. Absolute and relative error measurements of the approximate solution are computed on uniform grids of points $\{\mathbf{x}_i\}_{i=1}^{N_{grid}}$ evenly spaced in each spatiotemporal direction covering the entire domain using the following equations,

$$\begin{aligned} MSE_{abs} &:= \frac{1}{N_{grid}} \sum_{i=1}^{N_{grid}} \left(U(\mathbf{x}_i;\theta) - u(\mathbf{x}_i)\right)^2, \\ MSE_{rel} &:= \left[MSE_{abs}\right] / \left[\frac{1}{N_{grid}} \sum_{i=1}^{N_{grid}} u(\mathbf{x}_i)^2\right]. \end{aligned}$$

As shown in Table 3.1 and Figure 3.3, generating 1 extra point per marked mean achieved the best accuracy and proved to be the most stable. This suggests adding too many points to each marked mean degrades the enhancement from employing the adaptive method. Intuitively, this makes sense as the resultant clustering, evident in Figure 3.4, from the superfluous adaptive points can lead to an imbalanced training data set. For efficiency and to eliminate a hyperparameter, in the subsequent tests we only keep one extra point per mean.

Table 3.1: Error and loss measurements for the Poisson problem (3.6) with $n = 4$. Values reported are averages of 10 realizations, see Figure 3.3. The error and loss both increase with the number of points added per marked mean.

number of points	$\sqrt{LF_r}$	$\sqrt{MSE_{abs}}$	$\sqrt{MSE_{rel}}$
1	0.4067	1.960×10^{-3}	3.940×10^{-3}
3	0.4917	3.141×10^{-3}	6.314×10^{-3}
5	0.5970	4.137×10^{-3}	8.316×10^{-3}
10	0.8184	6.057×10^{-3}	1.217×10^{-2}

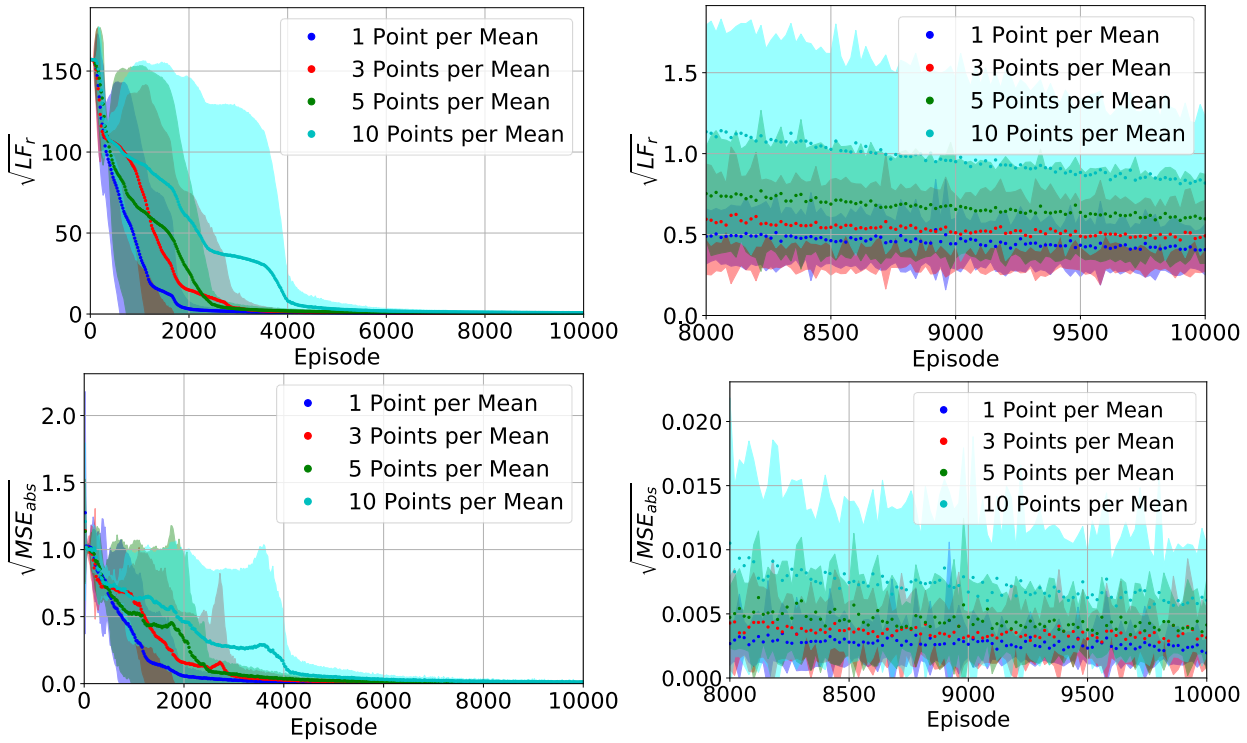
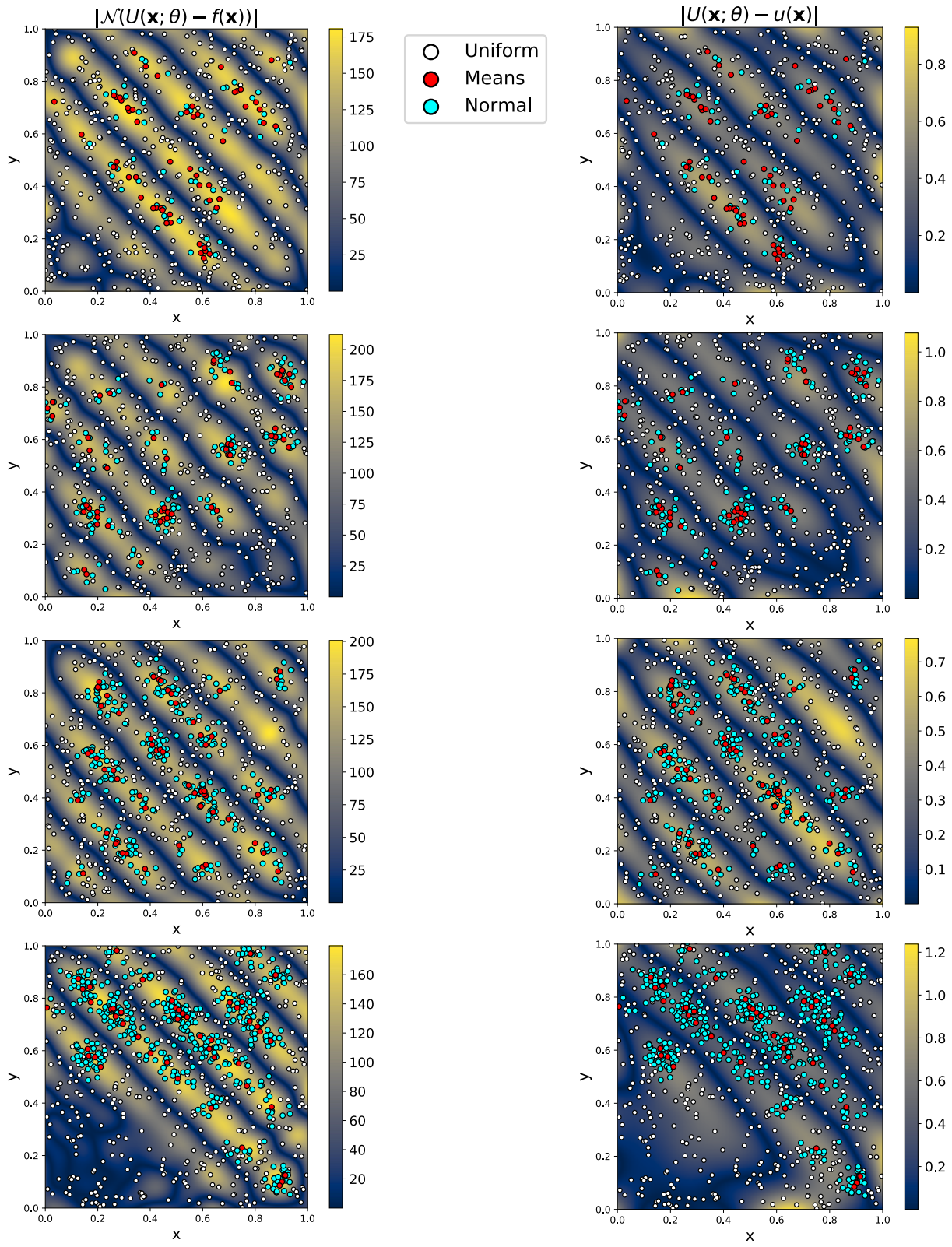


Figure 3.3: Error and loss per episode measurements for the Poisson problem (3.6) with $n = 4$. The solid points and shaded region correspond to the mean and two standard deviations, respectively. The square root of the residual and error was measured on a uniform grid for 10 realizations of each points per mean value. The magnification of the last 2000 episodes reveals that adding only one point per mean performs best.

Figure 3.4: Absolute residual (left column) and absolute error (right column) plots for the Poisson problem (3.6) with $n = 4$ at episode 500 with sample points superimposed. Each row corresponds to the number of points added per marked mean. From top to bottom, we show 1, 3, 5, and 10 points points per mean. The residual and error, though of different orders of magnitude, mirror each other locally.


$|\mathcal{N}(U(\mathbf{x};\theta) - f(\mathbf{x}))|$
$|U(\mathbf{x};\theta) - u(\mathbf{x})|$
Uniform
Means
Normal
x
y

3.2.3 Adaptive Variance

Here we show that adapting the variance increases the stability of the plain adaptive sampling DGM and makes the method less sensitive to penalty terms. The motivating intuition behind the adaptive variance is to further localize the extra adaptively added points in the marked regions with the highest residual. We employ the adaptive method mentioned in Algorithm 2, but within an episode, the hyperparameter controlling the covariance matrix, v_0, is not fixed for each marked mean. The diagonal elements of the covariance matrix are now dependent on the value of the residual at that point. To be more precise, given a marked mean, $\mathbf{x}_\ell$, and chosen initial variance, v_0, we sample extra points according to a multivariate normal distribution centered at $\mathbf{x}_\ell$ with covariance matrix $\Sigma_\ell = v_\ell \mathrm{Id}$ such that

$$v_\ell = v_0 \left| \frac{LF_r(\mathbf{x}_{min})}{LF_r(\mathbf{x}_\ell)} \right|, \tag{3.7}$$

with $\mathbf{x}_{min}$ denoting the marked mean with the lowest residual of the original uniformly sampled points at the start of that particular episode.

We note that we have performed various tests that showed that the adaptive variance DGM (ADLGM), exhibits similar behavior to plain adaptive with regards to points per mean. Thus, for each uniformly sampled point that has been marked for refinement, we only adaptively sample one additional point when we employ ADLGM. A similar adaptive technique was tested on the points per mean hyperparameter where, guided by the ratio featured in equation (3.7), more points were adaptively distributed around marked points with higher residuals but showed no improvement.

We demonstrate the results for the Poisson problem (3.6) with $n = 4$ for the penalty terms $C_b = 100, 800$ using the same network architecture and sampling hyperparameters as the previous section. Figure 3.6 exhibits the additional stability resulting from the adaptive variance and shows that ADLGM converges in fewer episodes than its plain adaptive counterpart.

3.3 ADLGM vs DGM Comparisons

In this section, we demonstrate the effectiveness of our algorithm on the oscillatory solution Poisson problem the Poisson problem with a square nonlinearity, and Burgers' equation, two benchmark problems often used to test adaptive methods. We also solve more examples

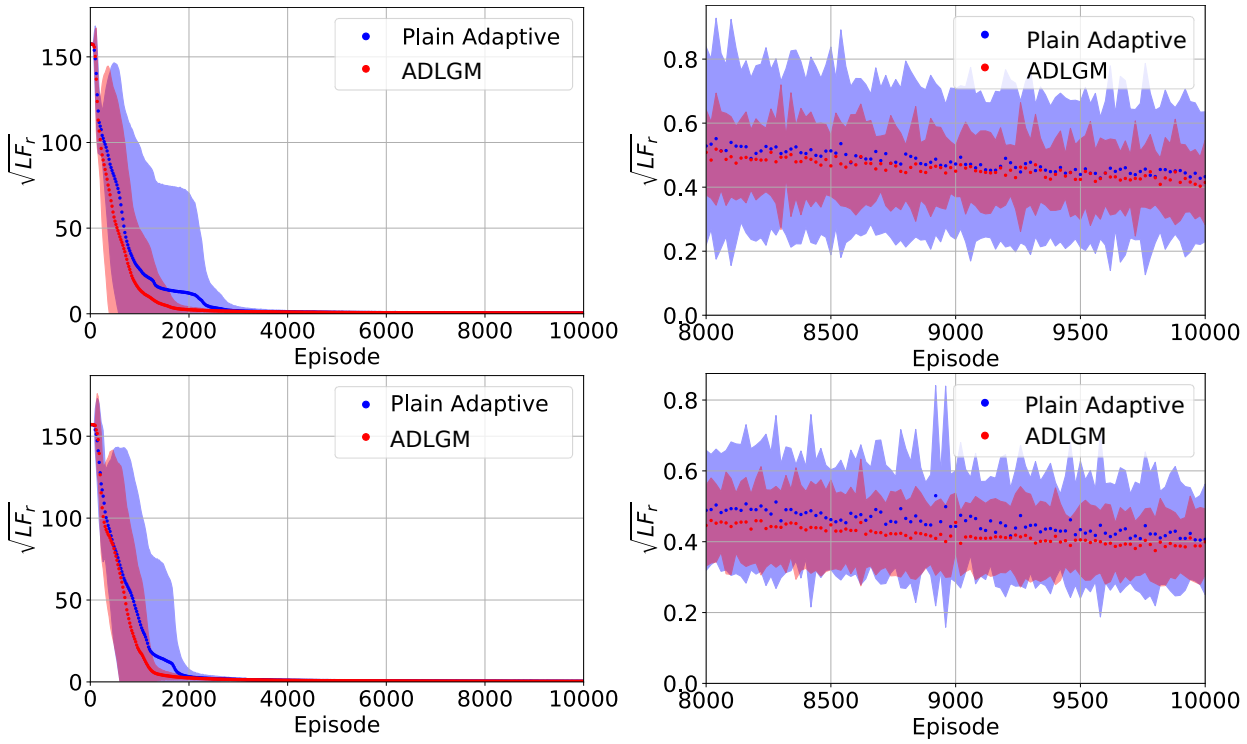


Figure 3.6: Loss per episode measurements for the Poisson problem (3.6) with $n = 4$ comparing plain adaptive sampling to adaptive variance sampling (ADLGM) for the test cases $C_b = 100, 800$. (top row): $C_b = 100$. (bottom row): $C_b = 800$. The solid points and shaded region correspond to the mean and two standard deviations, respectively. The right plots are a magnification of the last 2000 episodes. The plots show that ADLGM trains faster and the shaded regions reveal that the adaptive variance improves stability.

discussing the suitability of the PDE residual as a good error indicator for our adaptive procedures. All DGM realizations use the same number of episodes and iterations as their ADLGM counterpart as well as the same loss function (3.5).

3.3.1 Poisson with Oscillatory Solution

Now we show that our ADLGM learns better and faster than the DGM, the solution to the oscillatory solution PDE in (3.6). We provide results for both DGM and ADLGM for the cases $n = 4$ and 8, see Figure 3.7 for approximate solutions.

For all comparisons, the batch size used in the DGM tests is slightly higher than the maximum batch size observed in all episodes of ADLGM runs. We remark that ADLGM maximum batch size was used in only few episodes.

In Table 3.2, we display the loss, error, and relative error, and in figures 3.8 and 3.9, we see that ADLGM converges faster to a more accurate solution in a significantly more stable manner. We note that for $n = 4$ we use the same network parameters as in subsection 3.2.2 for 5,000 episodes of training. For $n = 8$ since it produces a more oscillatory solution, we increased the width of our DGM sub-layers to $M = 64$ units from the 16 used for $n = 4$.

3.3.2 Poisson with Squared Nonlinearity

Now we proceed to test a different elliptic PDE that has a square nonlinearity equipped with non-constant Dirichlet boundary conditions,

$$\begin{aligned} -\Delta u + u^2 &= f(x,y), && (x,y) \in \Omega, \\ u(x,y) &= g(x,y), && (x,y) \in \partial\Omega \ . \end{aligned} \tag{3.8}$$

We use the method of manufactured solutions and choose f and g such that $u(x,y) = \sin(\alpha\pi(x-0.5)(y-0.5))$ is the exact solution to the boundary value problem (3.8) in $\Omega = (0,1)^2$. The chosen two values of the coefficient $\alpha = 8, 16$ correspond to progressively more complicated PDE problems, see Figure 3.10. We plot the results in Figure 3.11 and observe that the improvement of ADLGM over the DGM in row 1 Figure 3.11 grows as the PDE becomes more complicated using the same size network (see, row 2 Figure 3.11). In Figure 3.12 we plot also the 2D exact error and residual landscape plots, showing that in this case the residual properly tracks the error.

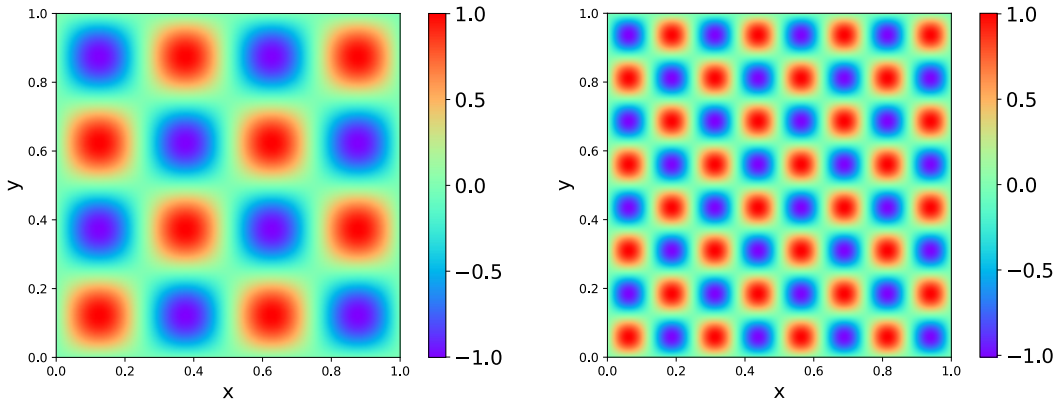


Figure 3.7: Solutions of the Oscillatory Poisson problem (3.6) generated by ADLGM. (left) $n = 4$, and (right) $n = 8$.

Table 3.2: Error and loss measurements for the Poisson problem (3.6) with $n = 4$, and 8. Values reported are averages of 10 realizations. For $n = 4$ and 8, ADLGM improves accuracy and achieves a lower residual.

n	Method	$\sqrt{LF_r}$	$\sqrt{MSE_{abs}}$	$\sqrt{MSE_{rel}}$
4	DGM	1.088	8.266×10^{-3}	1.259×10^{-2}
	ADLGM	0.6644	4.173×10^{-3}	8.389×10^{-3}
8	DGM	4.451	1.390×10^{-2}	2.794×10^{-2}
	ADLGM	3.053	8.519×10^{-3}	1.712×10^{-2}

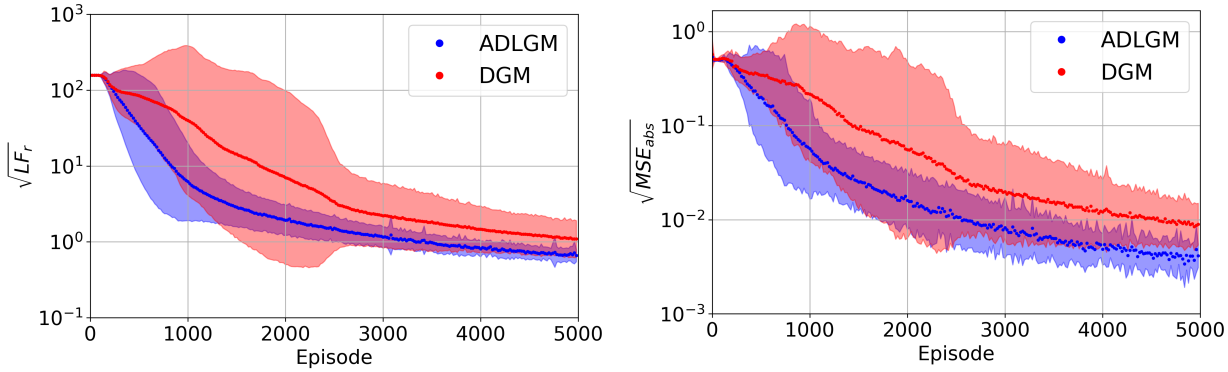


Figure 3.8: Error and loss per episode measurements on a logarithmic scale for the Poisson problem (3.6) with $n = 4$. The solid points and shaded region correspond to the mean and two standard deviations, respectively. (left): residual (right): error. The plots reveal that ADLGM again achieves better accuracy with improved stability.

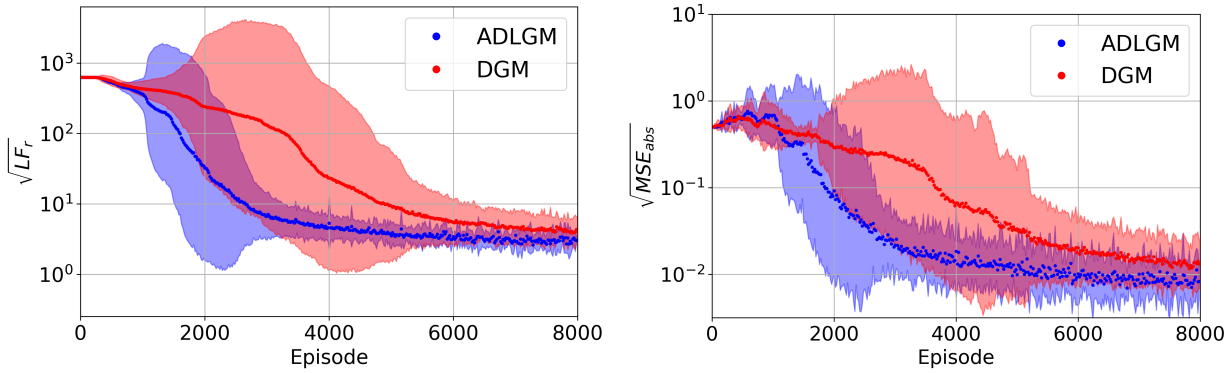


Figure 3.9: Error and loss per episode measurements on a logarithmic scale for the Poisson problem (3.6) with $n = 8$. The solid points and shaded region correspond to the mean and two standard deviations, respectively. (left): residual (right): error. Similar to the $n = 4$ case, ADLGM achieves better accuracy with improved stability.

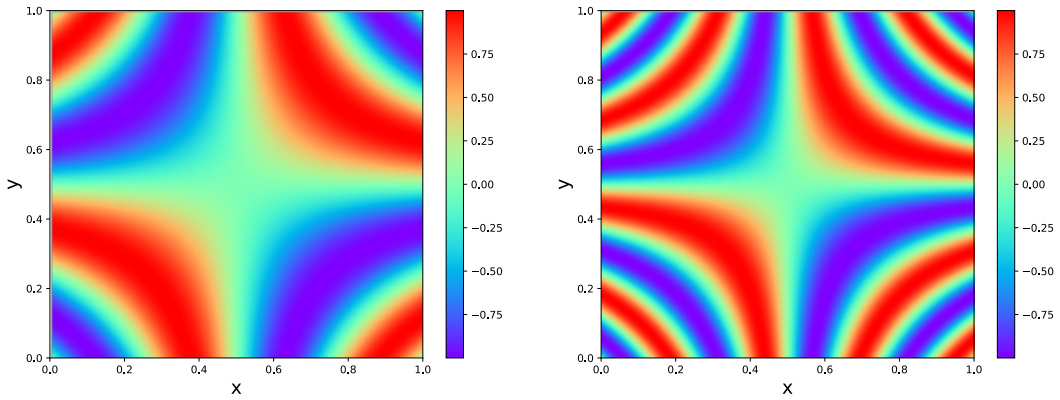


Figure 3.10: Solutions of the Poisson problem with square nonlinearity as in equation (3.8) generated by ADLGM. (left) $\alpha = 8$, and (right) $\alpha = 16$.

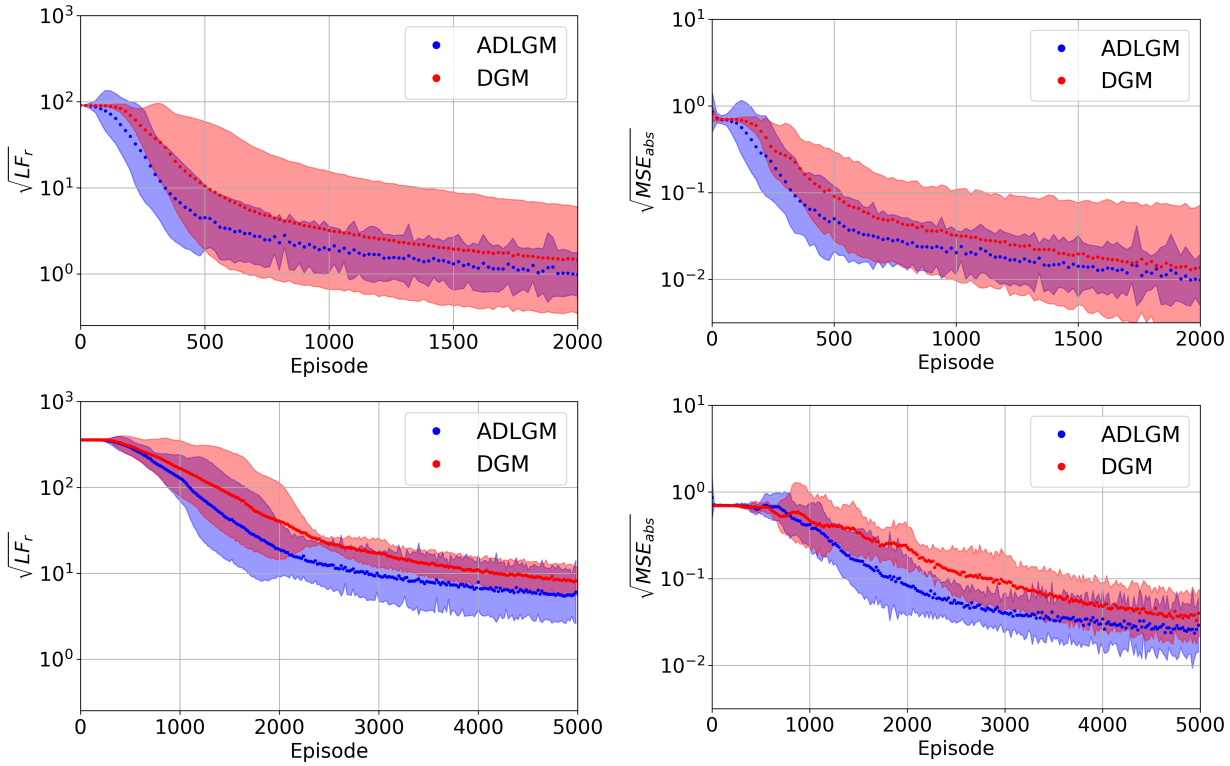


Figure 3.11: Residual and absolute error per episode on a logarithmic scale for comparison of ADLGM to DGM for the Poisson problem with square nonlinearity as in equation (3.8) with $\alpha = 8, 16$. The solid points and shaded region correspond to the mean and two standard deviations, respectively. (top row): $\alpha = 8$ (bottom row): $\alpha = 16$. ADLGM achieves better accuracy than DGM throughout training.

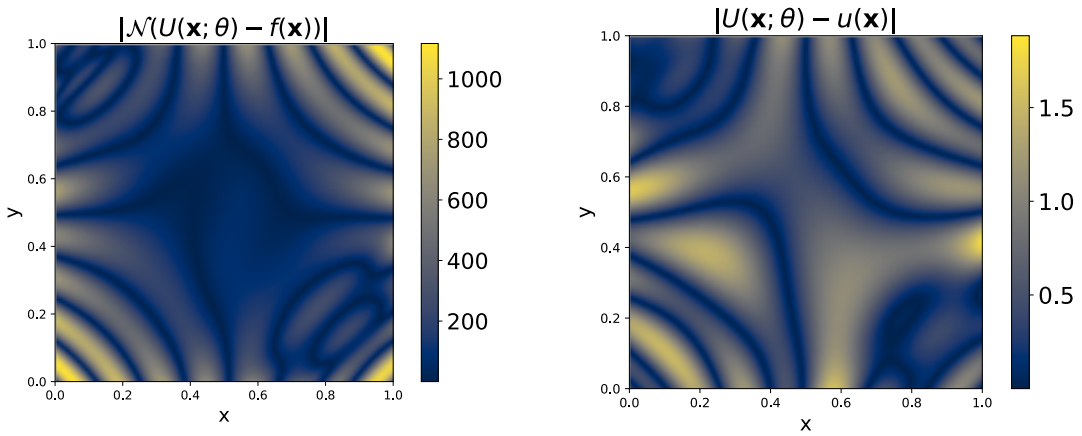


Figure 3.12: Residual and absolute error plots for the Poisson problem with square nonlinearity as in equation (3.8) with $\alpha = 16$ at episode 500.

3.3.3 Burgers' Equation

In this section, we provide results on solving Burgers' equation using ADLGM. We tested our algorithm on the following Burgers' equation with one spatial dimension and Dirichlet boundary conditions:

$$\begin{aligned} &u_t + uu_x - \frac{0.01}{\pi} u_{xx} = 0, && x \in (-1,1), \quad t \in (0,1] \\ &u(x,0) = -\sin(\pi x), && x \in (-1,1) \\ &u(-1,t) = u(1,t) = 0, && t \in (0,1] \end{aligned} \tag{3.9}$$

For comparison purposes, we compute a high fidelity numerical solution using FEniCS [50, 51], see Figure 3.13. We solve the problem with a network with the DGM architecture containing $L = 2$ layers (3 hidden layers total) and $M = 5$ units per sub-layer. We resample 10,000 interior points with 200 boundary points and 100 initial points at every episode. Each episode consists of 10 iterations. At each re-sampling the interior points accounting for $p = 10$ percent of the interior loss are used as the means of normal distributions for additional points with an initial variance value $v_0 = 0.0001$, smaller than the initial variance used for the oscillatory Poisson problem (3.6) to account for the thin shock in the Burgers solution.

In the rest of the work, we do 1000 episodes of ADAM training, and using the sample points from the last adaptive episode, we perform 10,000 iterations of L-BFGS. This allows for comparison to various adaptive methods that have been applied to PINN [13]. As shown in Table 3.3, ADLGM achieves better accuracy and lower residual than DGM. These results, Table 3.3, are comparable to those observed for an adaptive version of the PINN method in figure 8 in [13].

We remark here that in the Burgers' tests, as shown in Figure 3.14, we observe that earlier in the training process, the residual does not mirror the error locally like we observed in figure 3.8 for the oscillatory problem. Later in the training process they match better.

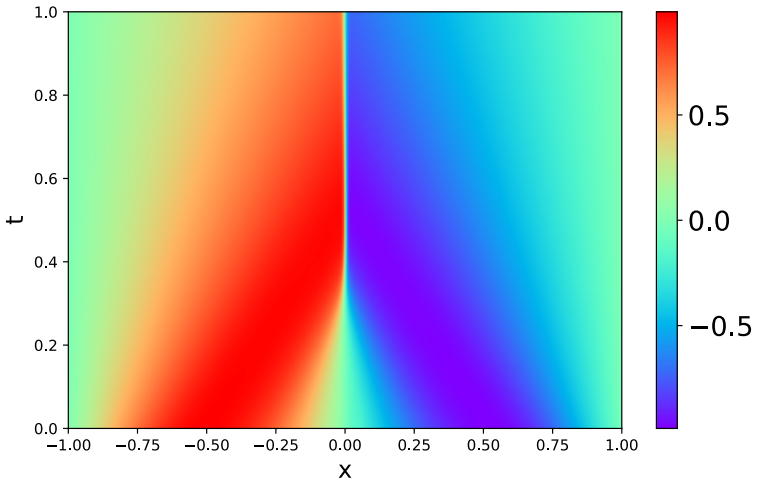


Figure 3.13: Solution to Burgers' Equation (3.9) generated by ADLGM.

Table 3.3: Error and loss measurements for Burgers' problem (3.9). Values reported are averages of 10 realizations. ADLGM improves accuracy and achieves a lower residual.

Method	$\sqrt{LF_r}$	$\sqrt{MSE_{abs}}$	$\sqrt{MSE_{rel}}$
DGM	1.394×10^{-2}	1.329×10^{-2}	2.166×10^{-2}
ADLGM	9.450×10^{-3}	5.763×10^{-3}	9.394×10^{-3}

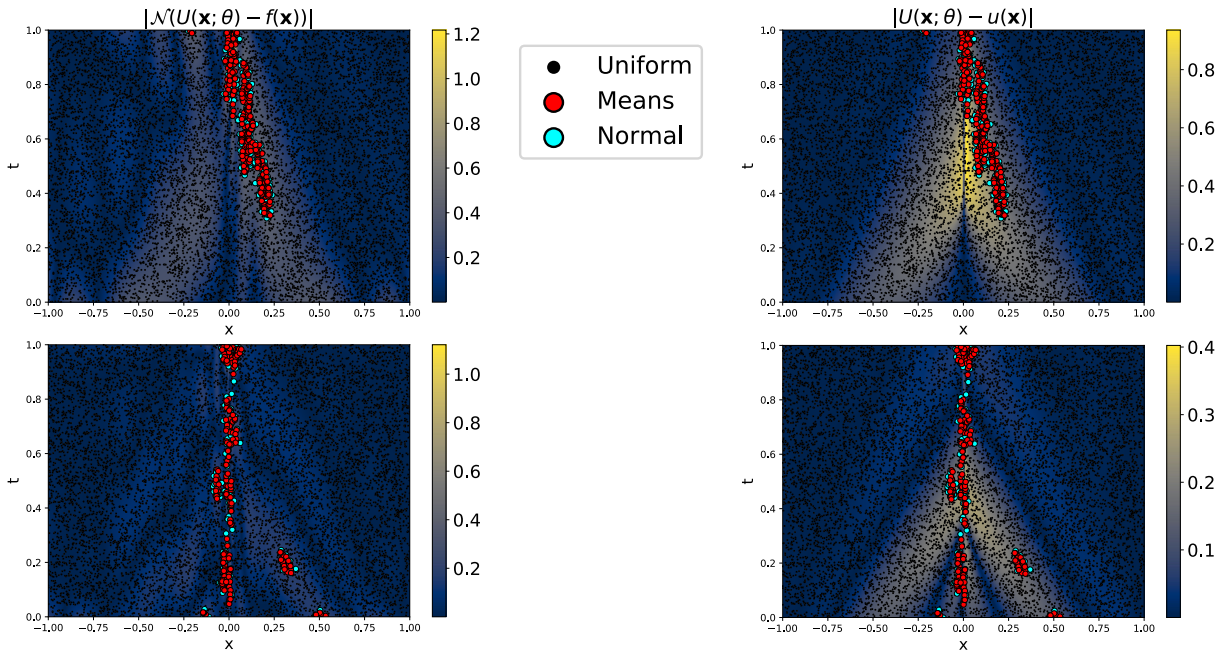


Figure 3.14: Absolute residual (left column) and absolute error (right column) plots for Burgers' problem (3.9) with sample points superimposed. From top to bottom, each row corresponds to episode 250 and 1000. Note the regions of highest magnitude for the residual and error converge to the same parts of the domain as training occurs.

3.4 Notes on Residual as a Marking Strategy

3.4.1 ADLGM on Allen-Cahn Equation

Here we show the results from testing ADLGM on the following Allen-Cahn PDE:

$$\begin{aligned} &u_t - 0.0001u_{xx} + 5u^3 - 5u = 0, && x \in (-1,1),\ t \in (0,1] \\ &u(x,0) = x^2\cos(\pi x), && x \in (-1,1) \\ &u(t,-1) = u(t,1), \quad u_x(t,-1) = u_x(t,1), && t \in (0,1] \end{aligned} \tag{3.10}$$

Here, we again use FEniCS [50, 51] to compute a high fidelity solution for the purpose of calculating errors. The network we use to solve the Allen-Cahn problem consists of $L = 3$ DGM layers with $M = 40$ units per sub-layer. At each ADAM episode we sample 20,000 interior, 200 boundary, and 100 initial points. We use $p = 5$ percent for marking and set the initial variance to $v_0 = 0.001$ for adaptively adding points. For a penalty term, we use $C_0 = 100$ because DGM and ADLGM, and PINN, see [12], were not able to converge to a solution without it.

Figure 3.16 shows ADLGM is able to drive down the absolute residual to slightly lower values than DGM; however, this does not correspond to a lower maximum error. Looking at the residual plots in Figure 3.17, we see that throughout training the regions of high residual do not coincide with those of highest error leading the residual-based marking strategy to add points to region of relatively low error. This might explain why the effect of ADLGM on the error is negligible.

Note that the regions marked by ADLGM in the earlier episodes are the same regions identified for the highest weights in [52], indicating the agreement of our adaptive method and marking strategy with their results, even if they use a radically different approach to adaptivity. This fact, emboldens us to say that the residual is not a sharp local error indicator, like you would expect to have, in order to correctly drive the marking strategy and adaptivity.

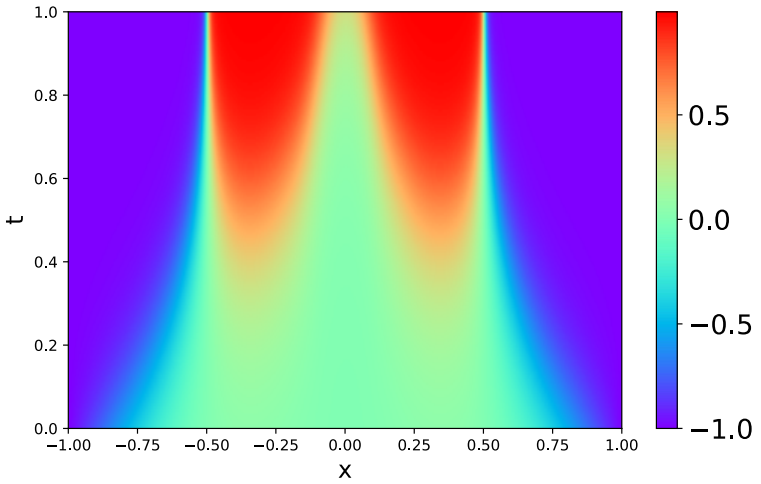


Figure 3.15: Solution to Allen-Cahn Equation (3.10) generated by ADLGM.

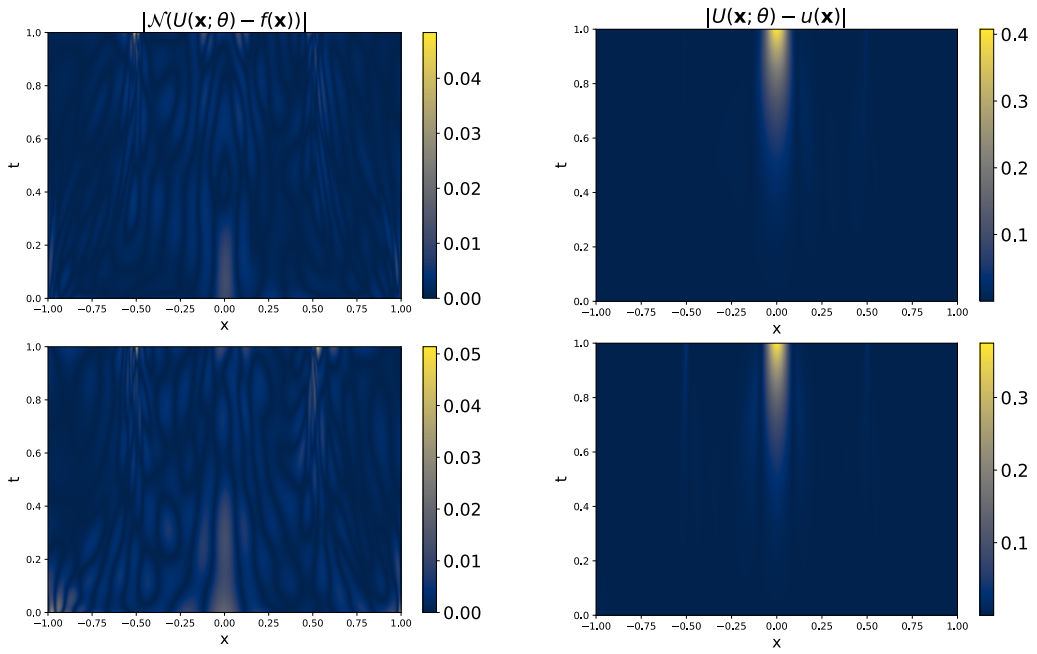


Figure 3.16: Residual and error plots for (top row): ADLGM and (bottom row): DGM. We see that ADLGM achieves similar results to DGM.

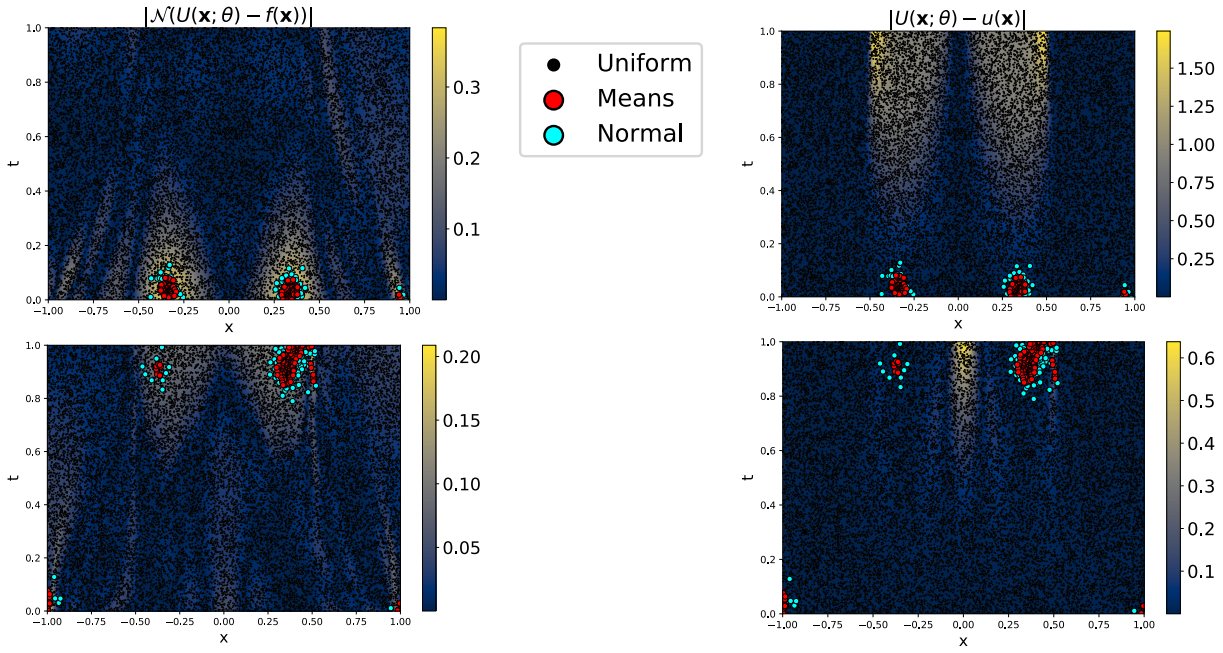


Figure 3.17: Absolute residual (left column) and absolute error (right column) plots for the Allen-Cahn PDE (3.10) with sample points superimposed. From top to bottom, each row corresponds to episode 200 and 1000. Note the regions of highest magnitude for the residual and error do not mirror each other throughout all of training.

3.4.2 ADLGM on Poisson on a Notch Domain

To further test our algorithm and the assertion regarding error and residual discrepancy we consider the following Poisson problem on a notch domain:

$$\begin{aligned} \Delta u &= 0, && \text{in } \Omega \\ u(r,\phi) &= r^{\frac{2}{3}} \sin\left(\frac{2}{3}\phi\right), && \text{on } \partial\Omega \end{aligned} \tag{3.11}$$

where $\Omega = [-0.5, 0.5]^2 \cap \{(x,y) : \frac{-3\pi}{4} \leq \phi \leq \frac{3\pi}{4}\}$ and ϕ denotes the polar angular coordinate of the pair (x,y). This problem has an exact solution

$$u(r,\phi) = r^{\frac{2}{3}} \sin\left(\frac{2}{3}\phi\right),$$

shown in Figure 3.18, with a point singularity in the first derivative at the origin. To tackle this PDE, we use the same network and sampling hyperparameters as we did for the $n = 4$ case of the oscillatory Poisson problem (3.6).

We observe that, similarly to Allen-Cahn, the residual does not mark the regions with the highest error during training, see Figure 3.19. The highest residual can be found in the sharp corners, while the highest error lies close to the boundary near the origin where the derivative is infinite. We like to note that our algorithm performs as intended, which is to minimize the residual, but the residual is failing to mirror the error locally. The consequence of this is that ADLGM shows negligible improvement in accuracy over DGM, when comparing the error plots.

In order to drive our marking strategy to the regions suffering the worst accuracy, we now test a different ad-hoc error indicator [53] adapted from the inverse inequality used in finite element methods [54]. Specifically, instead of marking the means with the highest residual, we use the value

$$\frac{\|\nabla_x U(\mathbf{x})\|}{\|U(\mathbf{x}) + \epsilon\|}, \tag{3.12}$$

for some small $\epsilon > 0$, to drive the "refinement".

We test this new strategy using the same network parameters as before, but now we mark the points where

$$\frac{\|\nabla_x U(\mathbf{x})\|}{\|U(\mathbf{x}) + \epsilon\|} > p \cdot c,$$

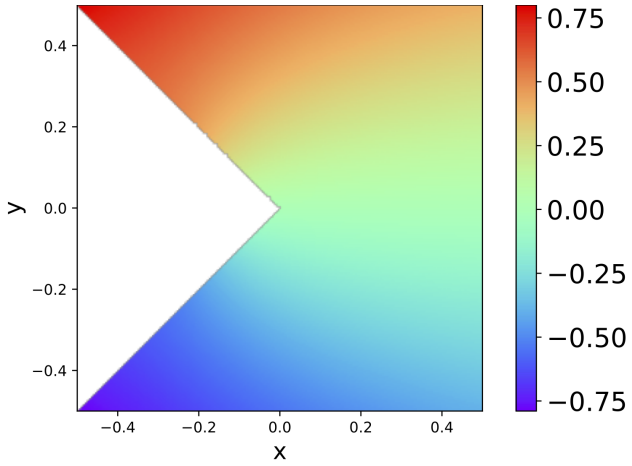


Figure 3.18: Solution to the Poisson problem on the notch domain (3.11) generated by ADLGM using the inverse inequality marking strategy.

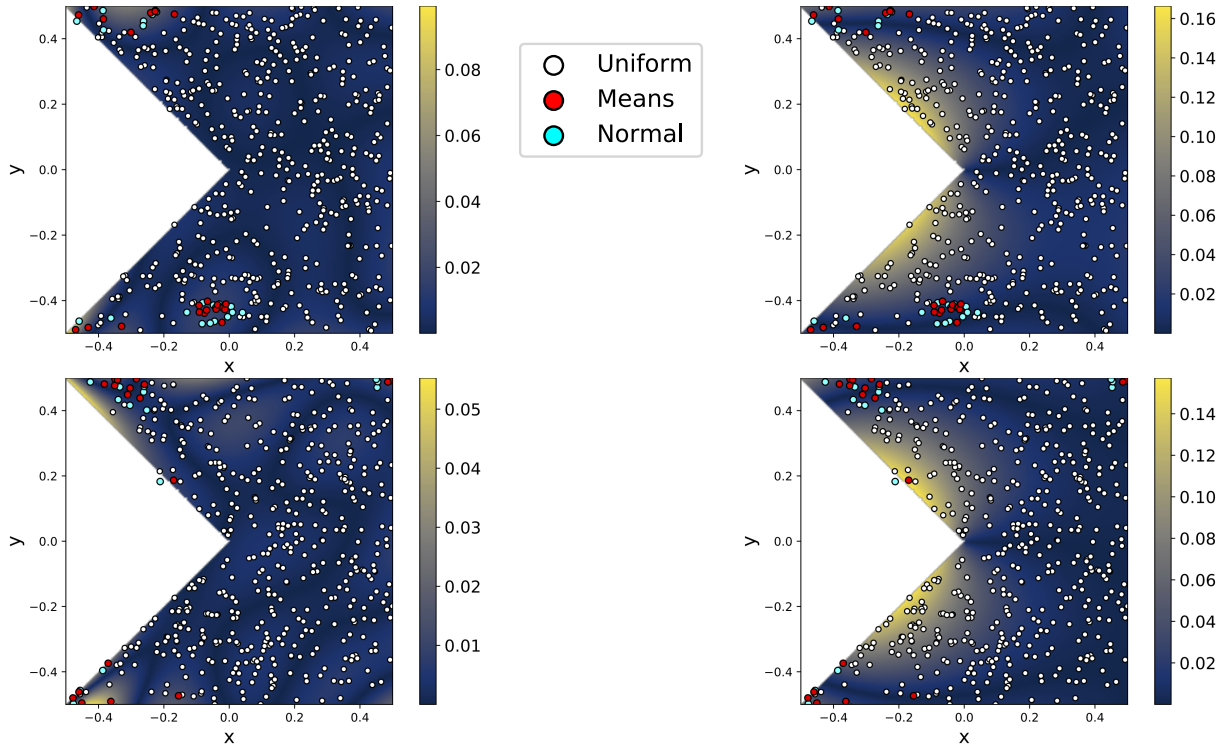


Figure 3.19: Absolute residual (left column) and absolute error (right column) plots for the notch domain Poisson problem (3.11) with sample points superimposed. From top to bottom, each row corresponds to episode 250 and 1000. Note the regions of highest magnitude for the residual and error do not mirror each other throughout all of training.

with $p = 60$ percent and $c = 2.83$, which was observed to be a suitable upper bound to (3.12) as the model converged to the solution of problem (3.11). Note the value for c is problem dependent and, hence, would need to be recalculated if the strategy was employed to solve a different PDE. The results, see Figure 3.20, show that while the inverse inequality marking strategy successfully adds points to the parts of the domain with the highest error, especially later in training, it does not lead to a significant improvement in approximating the solution. We conjecture this is because the DGM setup, and other deep learning PDE solving techniques, minimize a loss function that is based on the residual for the interior of the domain. Thus, any adaptive strategy can only aim to reduce the residual even when it proves to be a poor error indicator for certain PDEs. This realization might point to the need for modifying appropriately the loss term specifically the residual in order to become more sharp error indicator, analogously with the classical mesh adaptivity setup where a sharp residual type *a posteriori* error estimator is not just the norm of the residual.

3.5 Cable Equation

Now we apply ALDGM directly to a PDE relevant in neuroscience, the Cable equation. The cable equation has been used to model membrane potential in axons and dendrites. One of the most famous of these models is the Rall model, which represents a dendritic tree as a cylinder of finite length, L_c, with uniform passive membrane [55, 56]. Assuming "sealed end" boundary conditions; that is, no flow out of either end of the cylinder; this model takes the form

$$\begin{aligned} &\frac{\partial^2 V}{\partial X^2} = V + \frac{\partial V}{\partial T}, && x \in [0, L_c],\ T > 0 \\ &\frac{\partial V(0,T)}{\partial X} = 0 = \frac{\partial V(L_c,T)}{\partial X}, && T > 0 \\ &V(X,0) = h(X), && x \in [0, L_c]. \end{aligned} \tag{3.13}$$

A general solution to (3.13) has been obtained using separation of variables [57] and is given

$$V(X,T) = \sum_{n=0}^{\infty} B_n cos\left(\frac{n\pi X}{L_c}\right) e^{-\left[1+\left(\frac{n\pi}{L_c}\right)^2\right]T}, \tag{3.14}$$

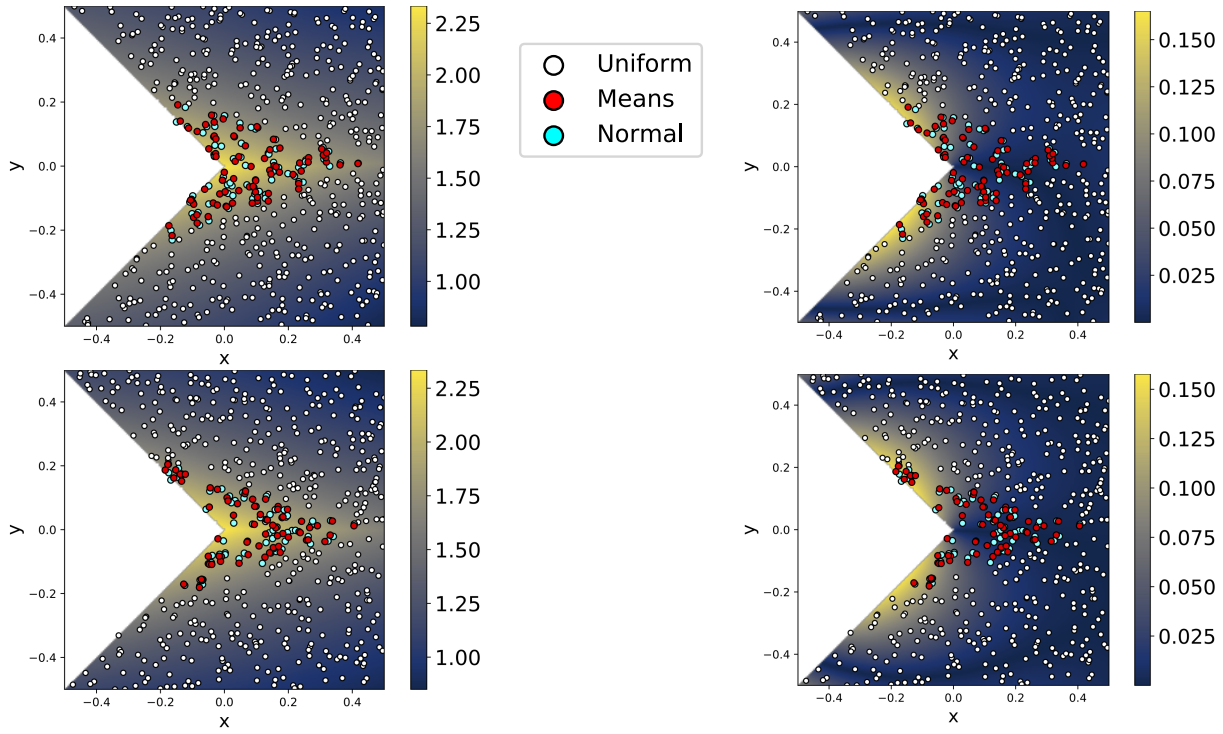


Figure 3.20: Value of (3.12) (left column) and absolute error (right column) plots for the notch domain Poisson problem (3.11) with sample points superimposed. From top to bottom, each row corresponds to episode 250 and 1000. The inverse inequality marking adds points to the region of high error, but this does not lead to a more accurate model.

where

$$B_0 = \frac{1}{L_c}\int_0^{L_c} h(X)dX$$

$$B_n = \frac{2}{L_c}\int_0^{L_c} h(X)cos\left(\frac{n\pi X}{L_c}\right)dX.$$

Here, we apply ADLGM to (3.13) with $h(X) = 50cos(\pi X)$ and $L_c = T = 1$, which, by (3.14), has exact solution

$$V(X,T) = 50cos\,(\pi X)\,e^{-\left(1+\pi^2\right)T}.$$

We solve the problem with a network consisting of $L = 4$ layers (5 hidden layers total) and $M = 32$ units per sub-layer. We resample 5,000 interior points with 400 boundary points and 200 initial points at every episode. At each re-sampling the interior points accounting for $p = 1$ percent of the interior loss are marked and used as means to the multivariate normal distributions used to generate adaptive points. Similar to solving the Allen-Cahn, we use $C_0 = 100$ in the loss function to make sure we properly capture the initial activity, but here we set $C_b = 50$ to improve the solution along the boundary. After 4,000 episodes of ADAM optimization and 10,000 iterations of L-BFGS, we obtain the results presented in Table 3.4 and Figure 3.21.

Table 3.4: Error and loss measurements for the Cable equation (3.13).

$\sqrt{LF_r}$	$\sqrt{MSE_{abs}}$	$\sqrt{MSE_{rel}}$
8.125×10^{-2}	0.5368	6.890×10^{-2}

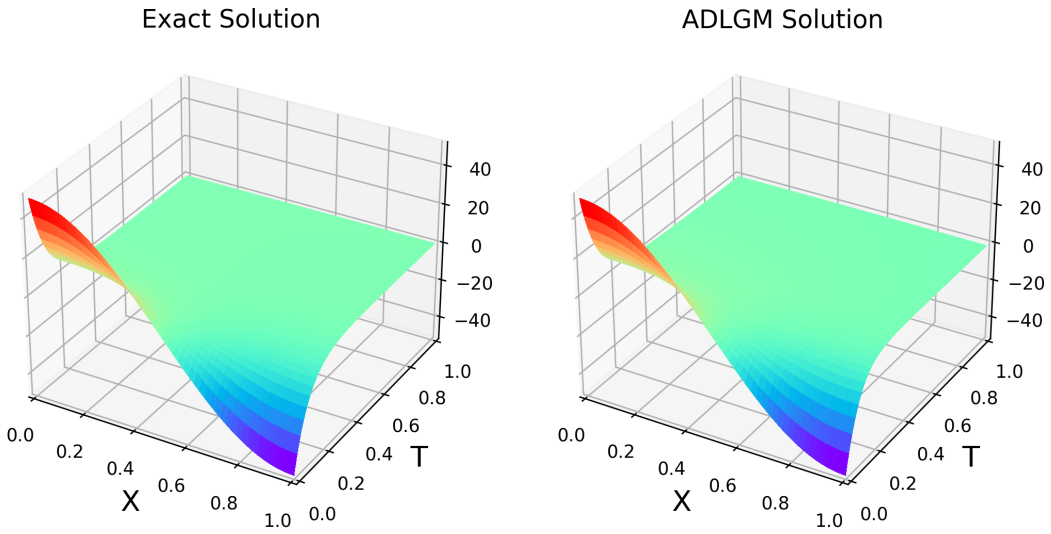


Figure 3.21: (left) Exact solution to the Cable equation (3.13) given by equation (3.14). (right) Solution obtained using ADLGM.

Chapter 4

A Topological Deep Learning Framework for Neural Spike Decoding

In this chapter, we take a task-oriented approach to develop a topological deep learning framework for decoding grid cell spike trains. In section 4.1, we provide background on neural decoding and grid cells. Section 4.2 provides details on tools from TDA that we use to represent neural activity. In section 4.3, we present the overall framework; specifically, we describe how we define neural activity on a simplicial complex and the dynamics of the novel SCRNN architecture that we employ to handle these simplicial complexes. We conclude the chapter with section 4.4 in which we apply our framework to two separate datasets of neural population activity. We first validate the method by decoding head direction from a population of HD cells, and compare the results to those produced by other NN architectures. Next, we demonstrate the effectiveness of the method by decoding two-dimensional location from a population of grid cells. Notably, to the best of our knowledge, our grid cell decoding task marks one of the first deep learning applications to decoding experimental grid cell data. Most difficulties accompanying decoding experimental, rather than simulated, neuronal data can be attributed to the inherent noise in the data itself as well as the number of cells required to encode certain variables. This noise may be typically generated from recording devices and the fact that some cells are responsible for encoding more than one piece of information [58]. Several steps of the pre-processing procedure were designed with such noise in mind.

4.1 Neural Decoding

Neurophysiological recording techniques have produced simultaneous recordings from increased numbers of neurons, both in vitro and in vivo, allowing for access to the activity of the hundreds of neurons required to encode certain variables [2, 59, 60, 61]. This makes efficient algorithms for decoding the information content from neural spike trains of increasing interest. Neural decoding can help provide insight into the function and significance of individual neurons or even entire regions of the brain [62]. Further, recent work in the area of brain-machine interfaces has integrated neural decoding of brain cells of certain utility into models used for prosthetic device control, perceptual readout, and communication with disabled patients [63, 64, 65]. One particular type of brain cell recently recorded in a quantity that allows for the analysis of its functional connectivity and structure of its population activity is the grid cell [2].

Grid cells are believed to be pivotal to mammalian navigation by playing a role in path integration [22, 58, 66], which is the integrating of one's velocities, and vector-based navigation, the planning of trajectories to target locations [67, 68, 69]. Grid cells encode two-dimensional allocentric location by forming hexagonal, periodic firing fields within an environment [22, 58, 67]. Grid cells with firing fields exhibiting the same spacing and orientation form what are referred to as modules [58, 70]. Due to the critical role grid cells play in the mammalian navigation system, we are interested in decoding grid cell activity.

Existing grid cell decoding efforts simulate grid cell activity within a heavily-tuned deep learning-based model trained on a spatial orientation or navigational task, and the simulated activity is then decoded using fully-connected feed forward or recurrent layers [71, 72]. However, these deep learning applications to neural decoding utilize architectures that ignore the underlying structure of the input neural activity. To appeal to this existing blindspot, we propose a topological deep learning framework for neural spike train decoding that takes into account the functional connectivity of neural data by first defining it on a simplicial complex. We then exploit the power of deep learning by employing our new neural network architecture that is designed to input and extract data from simplicial complexes called a *simplicial convolutional recurrent neural network* (SCRNN), see Figure 4.1.

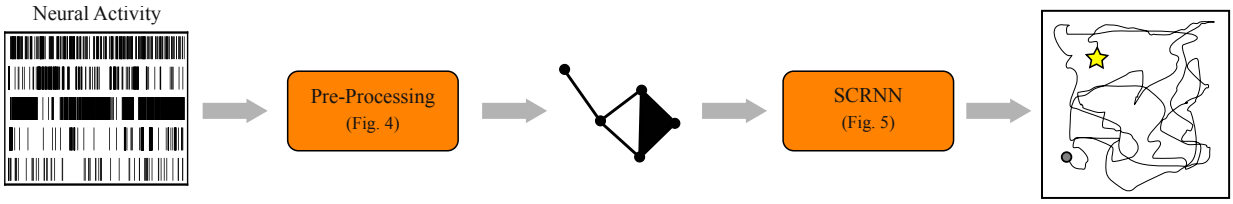


Figure 4.1: An overview of our framework. Neural data in the form of spike trains, represented by a raster plot, is defined on a simplicial complex via a pre-processing procedure, see Figure 4.2. The raster shown includes the activity of only five neurons for clarity. The simplicial complex gets input into a SCRNN, see Figure 4.3. Finally, the SCRNN decodes the desired variable(s). Here, we depict 2D location being decoded with the start and end locations of the trajectory depicted by a circle and star, respectively.

4.2 Topological Data Analysis Background

We introduce simplices and simplicial complexes, the topological structures we exploit for feature representation.

To define a $k-$simplex, we first must define what it means to be geometrically independent.

Definition 4.1. *A collection* $\{v_0, v_1, \ldots, v_n\} \subset \mathbb{R}^d \setminus \{0\}$ *is geometrically independent if and only if for any* $\{t_0, t_1, \ldots, t_n\} \subset \mathbb{R}$ *with* $\sum_{i=0}^{n} t_i = 0$*, the condition* $\sum_{i=0}^{n} t_i v_i = 0$ *implies* $t_i = 0$ *for all* $i \in \{0, 1, \ldots, n\}$*.*

Definition 4.2. *A* $k-$*simplex,* s^k*, is the convex hull of* $k+1$ *geometrically independent points* $\{v_0, v_1, \ldots, v_k\}$*, denoted by* $[v_0, v_1, \ldots, v_k]$*.*

Definition 4.3. *The faces of a* $k-$*simplex* $[v_0, v_1, \ldots, v_k]$ *are the* $(k-1)-$*simplices given by* $[v_0, \ldots, v_{j-1}, v_{j+1}, \ldots, v_k]$ *for some* $j \in \{0, 1, \ldots, k\}$ *and are denoted* $s_j^{k-1} \subset s^k$*.*

Definition 4.4. *A simplicial complex* S *is a collection of simplices satisfying*

1. *if* $s \in S$*, then every face of* s *is in* S *and*
2. *if* $s_1, s_2 \in S$*, then* $s_1 \cap s_2 = \emptyset$ *or* $s_1 \cap s_2 \in S$.

To ease understanding, one may consider a 0-simplex as a vertex, a 1-simplex as an edge, a 2-simplex as a triangle, a 3-simplex as a tetrahedron, and so on. Orientation can be assigned to $k-$simplices forming what is called an *ordered* $k-$*simplex*. For a face $s^{k-1} \subset s^k$, if the orientation of s^{k-1} coincides with that of s^k, we write $s^{k-1} \prec s^k$. Additionally, features, typically vectors or scalars, can also be assigned to the simplices. The features of the $k-$simplices are represented by a vector, or matrix depending on the feature size, called the $k-$*cochain*, and it is denoted by c_k.

Definition 4.5. *Let* $\{s_i^k\}_{i=1}^{N_k}$ *be the ordered* $k-$*simplices of a simplicial complex. Then for each* $s_i^k \in \{s_i^k\}_{i=1}^{N_k}$*, assign a feature* $c^i \in \mathbb{R}^{N_{feat}}$*. The* $k-$cochain*,* $c_k \in \mathbb{R}^{N_k \times N_{feat}}$*, is then given,*

$$[c_k]_{ij} = \left[c^i\right]_j \ . \tag{4.1}$$

4.3 Method

4.3.1 Pre-processing

To define the population activity of a group of cells on a simplicial complex, we partition time into non-intersecting bins and compute the spike counts for each bin. After a thresholding procedure, active cells within a time bin are connected via the appropriate-dimensional simplex. The result is a simplicial complex where connectivity is generated by functional activity. That is, simplices do not represent anatomically supported structural connections between neurons, but rather a temporally linked firing of neurons. We call this complex the *functional simplicial complex*. Note that the functional simplicial complex is discovered in an unsupervised manner without imposing any prior structure during any step.

The experimental HD data and grid cell data consist of neurons and their corresponding spike times. Given the spike times of N simultaneously recorded neurons, we first construct a spike count matrix A by creating N_{time} non-intersecting bins of width t_{bin} and counting each individual neuron's number of spikes within each bin, shown in Figure 4.2. The element A_{ij} is then set equal to the spike count of neuron i within bin j. The next step is to binarize A via a row-wise thresholding procedure. For a fixed row, consider the elements $\{a_\ell\}_{\ell=1}^{N_{time}}$ ordered from highest to lowest. Then for some value $p \in (0, 1]$, we select $\{a_\ell\}_{\ell=1}^{m^\star}$ for $m^\star$ given by,

$$m^\star = \underset{1\leq m\leq N_{time}}{\arg\min} \left\{ \sum_{\ell=1}^{m} a_\ell \quad \text{s.t.} \quad \sum_{\ell=1}^{m} a_\ell \geq p \cdot \sum_{\ell=1}^{N_{time}} a_\ell \right\} \ . \tag{4.2}$$

The $m^\star$ selected row elements are then set to 1 while the remaining $N_{time} - m^\star$ elements are set to 0. This is repeated for every row of A using the same value for p as before. Note that thresholding row-wise accounts for the variability in total spikes among neurons by comparing each neuron's activity against itself. We then proceed column-wise through the binarized matrix, connecting each active neuron within a time bin by the appropriate-dimensional simplex, see Figure 4.2. Specifically, if there are $0 \leq n_{act} \leq N$ active neurons in a column, an $(n_{act} - 1)-$simplex is constructed on the nodes corresponding to those n_{act} rows.

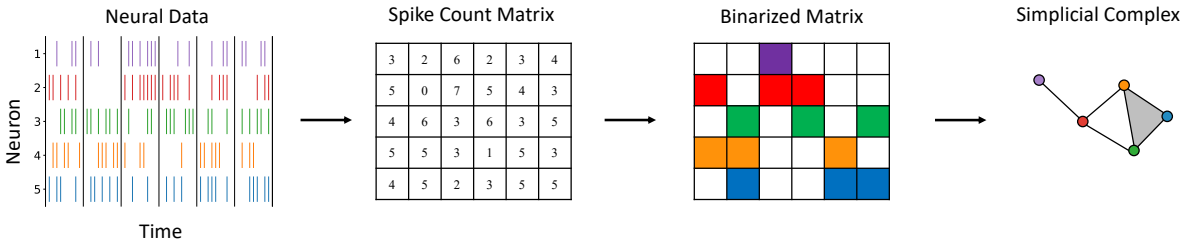


Figure 4.2: An example of the pre-processing procedure. Neural spiking data is represented as a raster plot on the left. The data is binned and converted to a spike count matrix. A row-wise thresholding procedure, given in Equation (4.2), binarizes the matrix. The binary matrix is displayed here with elements equal to 1 being colored in and the white elements conveying a 0. The colored regions within each column are then connected via the appropriate dimensional simplex to create the simplicial complex. For example, we see that the second column of the binarized matrix has three active neurons (green, orange, and blue). This generates a 2-simplex on the corresponding nodes. Note this allows for a multi-way description of these three nodes' relationship as opposed to the clique of 1-simplices that can only describe these nodes by their pairwise relationships.

4.3.2 SCRNN Input

Sub-complexes corresponding to a desired number, n_{col}, of consecutive time bins (columns of the spike count matrix A) are used as inputs to the SC layers. Suppose we aim to find the value of the decoded variable in time bin n_t, where $n_{col} \leq n_t \leq N_{time}$. Then the elements of the 0-cochain, $c_0(n_t) \in \mathbb{R}^{N_0 \times n_{col}}$, are given

$$[c_0(n_t)]_{ij} = [A]_{i(n_t - n_{col} + j)}\,, \tag{4.3}$$

for $i = 1, 2, ..., N$ and $j = 1, 2, ..., n_{col}$. The 1-cochains are the Pearson product-moment correlation coefficients of the row-vectors of A corresponding to the 1-simplices' faces. Let N_1 denote the number of 1-simplices present in the functional simplicial complex. After arbitrarily indexing the 1-simplices, let $q(i)$ and $r(i)$ denote the row index corresponding to the faces of the ith 1-simplex. We define the elements of the 1-cochain, $c_1(n_t) \in \mathbb{R}^{N_1}$, as follows: for any $i \in \{1, 2, ..., N_1\}$ not present in the n_t time bin, we set $[c_1(n_t)]_i = 0$; otherwise,

$$[c_1(n_t)]_i = R_{q(i)r(i)}, \tag{4.4}$$

where $R_{q(i)r(i)}$ is the Pearson product-moment correlation coefficients of the $q(i)$th and $r(i)$th row-vectors of A. Similarly, for the 2-cochains, let $q(i), r(i)$, and $u(i)$ denote the row index corresponding to the 0-simplices contained in the ith 2-simplex. We define the elements of the 2-cochain, $c_2(n_t) \in \mathbb{R}^{N_2}$, as follows: for any $i \in \{1, 2, ..., N_2\}$ not present in the n_t time bin, we set $[c_2(n_t)]_i = 0$; otherwise,

$$[c_2(n_t)]_i = \min\{R_{q(i),r(i)u(i)},\ R_{r(i),q(i)u(i)},\ R_{u(i),q(i)r(i)}\}, \tag{4.5}$$

where $R_{q(i),r(i)u(i)}$ denotes the multiple correlation coefficient of the $q(i), r(i)$, and $u(i)$th row-vectors of A with $r(i)$ and $u(i)$ considered dependent on $q(i)$. Higher dimensional cochains can be defined in a similar manner, but with a metric other than the correlation of the simplices' faces that can be chosen specific to the cell population being decoded.

4.3.3 SCRNN

It is common practice for neural activity to be converted to a matrix where rows represent individual neurons and columns correspond to time bins. The most widely used deep learning

approach to handling matrices as inputs is to employ a convolutional neural network (CNN). In a CNN, convolutional layers extract features from the input by aggregating weighted information from neighboring elements in the input matrix. This localization of information-sharing assumes regular connectivity where only neighboring rows, or columns, possess significance to each other. Thus, in tasks where rows of a matrix neighboring each other bares no significance, CNNs do not intuitively extract features.

Simplicial convolutions generalize convolutions to account for data with irregular connectivity. For these layers, input data is defined on a simplicial complex, and information-sharing is generated by the *Hodge-Laplacian*. To define the Hodge-Laplacian, we must first introduce the $k-$*dimensional incidence matrix*, $B_k \in \mathbb{R}^{N_{k-1} \times N_k}$, where the ijth element is given by,

$$[B_k]_{ij} = \begin{cases} 0, & \text{if } s_i^{k-1} \not\subset s_j^k, \\ -1, & \text{if } s_i^{k-1} \subset s_j^k \text{ and } s_i^{k-1} \not\prec s_j^k \\ 1, & \text{if } s_i^{k-1} \subset s_j^k \text{ and } s_i^{k-1} \prec s_j^k \end{cases} , \tag{4.6}$$

where N_{k-1} and N_k are the number of $(k-1)-$simplices and $k-$simplices, respectively. Note, we consider $B_0 = 0 \in \mathbb{R}^{N_0 \times N_0}$. Then, finally, the $k-$Hodge-Laplacian, $L_k \in \mathbb{R}^{N_k \times N_k}$, is defined as,

$$L_k = B_k^T B_k + B_{k+1} B_{k+1}^T \ . \tag{4.7}$$

In simplicial convolutions, the terms of the Hodge-Laplacian in equation (4.7) act as shift-operators defining which simplices of the same dimension share information. The terms $B_k^T B_k$ and $B_{k+1} B_{k+1}^T$ are called the *lower* and *upper Laplacian*, and they capture connectivity by lower and higher dimensional simplices, respectively. A *degree D simplicial filter* consisting of weights $W = \{W_i\}_{i=0}^{2D}$ is an operator, $H_k \in \mathbb{R}^{N_k \times N_k}$, given by,

$$H_k = W_0 I + \sum_{i=1}^{D} W_i (B_k^T B_k)^i + \sum_{i=1}^{D} W_{i+D} (B_{k+1} B_{k+1}^T)^i \ , \tag{4.8}$$

where k is the dimension of simplices and $(\cdot)^i$ denotes the $i-$th power of a matrix. Note, each power of the lower and upper Laplacians localizes information-sharing to within the i nearest $k-$simplices, similar to increasing the filter size in a traditional convolutional layer.

In the case where we restrict $W_i = W_{i+D}$ in Equation (4.8), i.e.

$$H_k = W_0 I + \sum_{i=1}^{D} W_i L_k^i \ , \tag{4.9}$$

it has been shown [43] that the filters are low-degree polynomials in the frequency domain.

We now discuss the dynamics of the *simplicial convolutional layers* of an SCRNN.

Proposition 4.6. *Consider an SCRNN consisting of L simplicial convolutional layers, each equipped with F filters, $\{H_k^f(\ell)\}_{f=1}^F$, for each dimension k of the functional simplicial complex with maximum simplicial dimension K, where $\ell \in \{1, 2, \ldots, L\}$ denotes the simplicial convolutional layer. In such a network, the number of parameters used in the simplicial convolutional layers is $F[2(D+1) + (K-1)(2D+1)]L$.*

Proof. Let $f \in \{1, 2, \ldots, F\}$ and $\ell \in \{1, 2, \ldots, L\}$ be arbitrary. Fix $k = 0$. Then because $B_0 = 0 \in \mathbb{R}^{N_0 \times N_0}$, we have

$$H_0^f(\ell) = W_0^{f,0}(\ell) I + \sum_{i=1}^{D} W_i^{f,0}(\ell)(B_1 B_1^T)^i \ , \tag{4.10}$$

where $\{W_i^{f,0}(\ell)\}_{i=0}^D$ are filter parameters. Thus, the 0-dimensional component of an arbitrary filter in an arbitrary simplicial convolutional layer contains $D+1$ parameters. Similarly, for fixed $k = K$, we have

$$H_K^f(\ell) = W_0^{f,K}(\ell) I + \sum_{i=1}^{D} W_i^{f,K}(\ell)(B_K^T B_K)^i \ . \tag{4.11}$$

Therefore, the K-dimensional component of an arbitrary filter in an arbitrary simplicial convolutional layer also contains $D+1$ parameters. Now, for an intermediate dimension $k \in \{1, 2, \ldots, K-1\}$, a filter is defined

$$H_k^f(\ell) = W_0^{f,k}(\ell) I + \sum_{i=1}^{D} W_i^{f,k}(\ell)(B_k^T B_k)^i + \sum_{i=1}^{D} W_{i+D}^{f,k}(\ell)(B_{k+1} B_{k+1}^T)^i \ , \tag{4.12}$$

which contains $2D+1$ parameters $\{W_i^{f,k}(\ell)\}_{i=0}^{2D}$. For an arbitrary filter, there are $K-1$ such components (one for each dimension $k \in \{1, 2, \ldots, K-1\}$). Hence, for a single filter, the total number of parameters for all intermediate k-dimensional components combined is

$(K-1)(2D+1)$. Adding this sum to the number of parameters for $k=0$ and $k=K$, we see that one filter contains $2(D+1)+(K-1)(2D+1)$ parameters. Finally, because this holds for any filter in any layer, we multiply by the number of filters and the number of layers giving us $F[2(D+1)+(K-1)(2D+1)]L$ total simplicial convolutional parameters. □

Note that the dynamics of the simplicial convolutional layers prevent exponential growth of parameters with respect to filters and number of layers.

For the first layer $\ell=1$, features $\{\mathbf{x}_k^f(1)\}_{f=1}^F$ are extracted from the input, $\mathbf{x}_k(0)$, via the nonlinear transformations,

$$\mathbf{x}_k^f(1) = \sigma\left(H_k^f(1)\ \mathbf{x}_k(0)\right)\ , \tag{4.13}$$

for each $f=1,2,\ldots,F$ and $k=1,2,\ldots,K$. Note $\mathbf{x}_0(0)\in\mathbb{R}^{N_0\times n_{col}}$ for some hyperparameter $1\leq n_{col}\leq N_{time}$, and $\mathbf{x}_k(0)\in\mathbb{R}^{N_k}$ for $1\leq k\leq K$. For the intermediate simplicial convolutional layers $\ell=2,3,\ldots,L-1$ and fixed k, each of the filters $\{H_k^f(\ell)\}$ is applied to each of the extracted features from the previous layer. To prevent the exponential growth of the number of filters, the outputs extracted from the same feature from the previous layer are summed together to create one single output feature. That is, for each feature $\{\mathbf{x}_k^g(\ell-1)\}_{g=1}^F$ from the previous layer, we extract,

$$\mathbf{x}_k^g(\ell) = \sigma\left(\sum_{f=1}^F H_k^f(\ell)\ \mathbf{x}_k^g(\ell-1)\right)\ , \tag{4.14}$$

for $g\in\{1,2,\ldots,F\}$. In the final simplicial convolutional layer, $\ell=L$, features are extracted following the same procedure as the intermediate layers, but additionally, all extracted features are summed:

$$\mathbf{x}_k(L) = \sum_{g=1}^F \mathbf{x}_k^g(L) = \sum_{g=1}^F \sigma\left(\sum_{f=1}^F H_k^f(\ell)\ \mathbf{x}_k^g(\ell-1)\right)\ , \tag{4.15}$$

where $x_k^g(L)$ as in Equation (4.14) for $\ell=L$. If $1<n_{col}$, then $\mathbf{x}_0(L)$ is summed across columns, which gives us $\mathbf{x}_0(L)\in\mathbb{R}^{N_0}$. Finally, the outputs for each dimension of the simplicial complex, $\{\mathbf{x}_k(L)\}_{k=1}^K$, are stacked to create one output feature vector, $\mathbf{x}(L)\in\mathbb{R}^{\sum_{k=0}^K N_k}$. For illustrative purposes, Figure 4.3, depicts $L=2$ simplicial convolutional layers each consisting of $F=3$ filters for each dimension of the input simplicial complex.

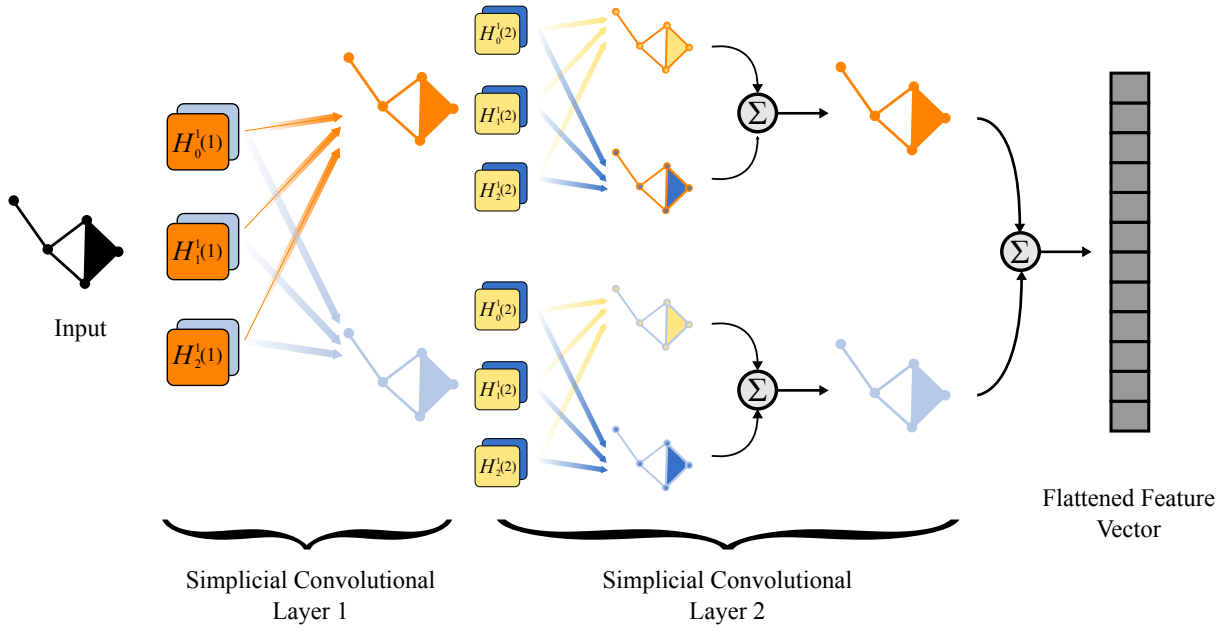


Figure 4.3: A diagram of $L = 2$ simplicial convolutional layers each equipped with two filters, $H_k^1(1)$ and $H_k^2(1)$, for each simplicial dimension $k = 0, 1, 2$. Filters are color coded with filters that get concatenated bearing the same color as the resultant simplicial complex. In the first layer, we see three orange filters indicating the three dimensions of the input simplicial complex. The features extracted using these filters result in a new, orange simplicial complex. The second filter, depicted in light blue, extracts a separate simplicial complex. In the second simplicial convolutional layer, the process is repeated with two new filters, depicted by yellow and dark blue. In order to prevent exponential growth, features extracted from the same input from the previous layer are summed. Finally, all extracted features are summed and flattened to create one feature vector.

To form an input sequence to the RNN component of the SCRNN, we consider the outputs of the simplicial convolutional layers corresponding to a desired number of consecutive time bins. Given the sequential nature of the decoding task, we append the simplicial convolutional layers with a multi-layer RNN, see Definition 2.3.

4.4 Results

For comparison of the SCRNN to existing NN architectures, we decode experimental HD data using an FFNN and an RNN. We also choose to compare our method to using only a graph neural network (GNN), which corresponds to removing the $2-$simplices. To highlight the role of the recurrent layers of the SCRNN, we also employ an architecture consisting of simplicial convolutional layers attached to fully connected feed-forward layers; we refer to this network as the *simplicial convolutional neural network* (SCNN).

Finally, after testing all five methods on the HD data and demonstrating that the SCRNN has the best performance, we implement our SCRNN framework on the more complex task of decoding three modules of grid cells that includes conjunctive grid/HD cells. For the grid cell task, we not only include comparisons to the other NN architectures, but we also test the performance of generating functional simplicial complexes for each grid cell module individually. We describe this method further below. Below, we outline the results for both the head direction and grid cell tasks.

4.4.1 Head Direction Cells

The neurons making up the head direction (HD) system in the brain encode the direction the head is facing at any given time. This encoding is done by identifying different ensembles of certain neurons, called HD cells, which fire simultaneously, where each grouping of cells represents a different direction [73]. To demonstrate the effectiveness of our method, we analyze HD data recorded in [1]. The spike times of HD cells in the anterodorsal thalamic nucleus (ADn) along with the corresponding ground truth head angles of seven mice were recorded using multi-site silicon probes and an alignment of LED lights on the mice's headstage, respectively. The sessions recorded comprised of two hours of sleep followed by 30-45 minutes of foraging in an open rectangular environment followed by 2 more hours of sleep. For the results shown in this work, we used the foraging portion of session 'Mouse28-140313'

in [1]. This dataset contains the spike data of 22 neurons, and to reduce computational cost, we looked at the first 20 minutes of a 38 minute foraging session for a single mouse.

Decoding accuracy was measured in three different ways. First, we considered the median absolute error (MAE), which is defined,

$$MAE = \underset{n=1,\ 2,\ \ldots,\ N_{time}}{median} \Big| \ rescale\big[\theta_{dec}(n) - \theta_{true}(n)\big] \ \Big| \ , \tag{4.16}$$

where N_{time} is the number of time bins, and $\theta_{dec},\ \theta_{true} \in [0°, 360°)$ are the decoded and the ground truth directions, respectively. The mapping *rescale* accounts for the ring structure of HD. For example, 310° and 20° should be recorded as a difference of 70° instead of 290°.

Similarly, we compute the average absolute error (AAE), which considers the average instead of the median discrepancy as defined below,

$$AAE = \frac{1}{N_{time}} \sum_{n=1}^{N_{time}} \Big| \ rescale\big[\theta_{dec}(n) - \theta_{true}(n)\big] \ \Big| \ . \tag{4.17}$$

The final recorded error is the catastrophic error, denoted as CAT, which counts the number of predictions off by 90° or more from the real HD, since this would imply that our prediction is completely wrong. We focus on CAT error based on the importance of bounding the cost of a single error as opposed to just bounding the overall error [74]. That is, considering the median or average of errors places less significance on the larger outliers that can have devastating consequences in applications of models employing neural decoding.

We ran 10 trials with 75% of the data used for training and 25% of the data used for testing for each method using the hyperparamters listed in Appendix Table 1. All around the SCRNN performed the best across all non-geometric architectures, see Table 4.1 and Figure 4.4. While the GNN achieved lower MAE and AAE, these are not the measurements for which the networks were optimized. Note the GNN produced 19 catastrophic errors compared to the 6 observed for the SCRNN. Similarly, if we instead use 50% for training and 50% for testing or 25% minutes for training and 75% for testing, the SCRNN still outperforms the non-geometric networks. See the Appendix 5 for additional tests and hyperparameter tuning information.

Though we include the MAE and AAE measurements, it is important to note that the NN hyperparameters were only tuned to minimize CAT. We observe the SCRNN produces the least amount of catastrophic errors of all the networks included in the comparison. Thus,

Table 4.1: The AAE and MAE on the training and testing data, as well as the total numbers of catastrophic errors. We observe the SCRNN produces the least amount of catastrophic errors.

method	train AAE	test AAE	train MAE	test MAE	CAT
FFNN	12.03	16.85	8.87	12.75	70
SCNN	14.67	15.61	11.00	11.70	88
GNN	12.96	14.44	10.11	10.79	19
RNN	7.92	14.49	5.92	11.26	9
SCRNN	8.20	13.99	6.13	10.57	6

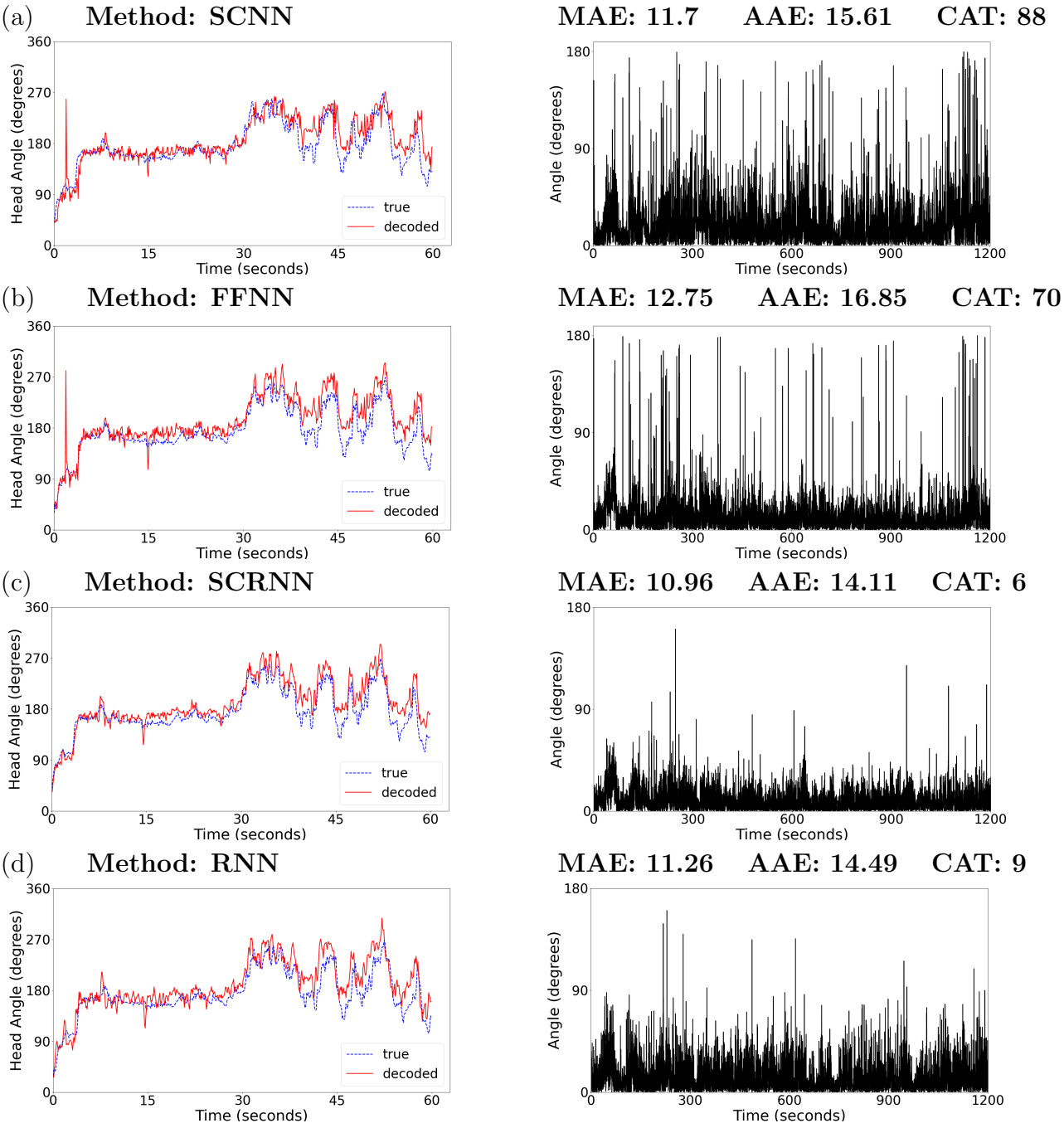


Figure 4.4: Plots depicting the true head angle and the predicted head angle for the first two minutes (left) and the catastrophic error for each time bin for the full twenty minutes (right) for four different networks, a) SCNN, b) FFNN, c) SCRNN, and d) RNN.

we conclude that inputting the underlying simplicial complex structure into the network and using its connectivity to generate information-sharing prevents less outlier errors that could become problematic in application. With this in mind, we apply the SCRNN to grid cells; a different type of cell that encodes environment-based location.

4.4.2 Grid Cells

Layers in the Medial Entorhinal Cortex (MEC) have been shown to include both pure grid cells which are believed to fire independent of head direction and conjunctive grid/HD cells which display firing patterns tuned to a single head direction [2, 75]. We decode the position (as in xy−coordinates) from the activity of a population of cells recorded in a moving rat, which contains pure grid cells and conjunctive grid/HD cells [2].

Due to the fact firing fields for cells within a module are the same, except for a shift in space, it takes more than one module to encode position [70, 76]. Cells firing at the same time within a module generate a spatial grid over the environment. A multi-scale representation for location is then created by layering the grids generated by different modules.

What follows is a brief description of the grid cell data reported in [2]. Neural activity was recorded in layers II and III of the medial entorhinal cortex using high-site-count Neuropixels silicon probes [60, 61] while rats foraged alone in an open square 1.5m×1.5m arena. Three-dimensional motion capture was utilized to track the rats' head directions and two-dimensional positions in the environment. The results presented in this paper use modules labeled R1-R3 day one in the source data [2]. In the following analysis, we look at a population of 482 cells that contains three grid modules consisting of 166, 167, and 149 cells total with 93, 149, and 145 of them being pure grid cells, respectively; for a look at the spike trains from the first module, see Appendix Figure 4. The rest of the population is made up of conjunctive grid/HD cells. The difficulties of decoding such a population stem from not only the large amount of cells, but also the fact that some cells are not solely responsible for encoding position, the target variable we aim to decode [58]. The larger population of cells and, consequently, the larger size of the functional simplicial complex compared to the HD decoding task, means a more heavily parameterized SCRNN is required to decode position.

We include comparisons to a FFNN, SCNN, RNN, and to an altered version of our method, abbreviated SCRNN$_{mod}$ where we create a functional simplicial complex for each individual grid cell module. That is, we follow the same pre-processing procedure as above

treating each module as its own population. This generates incidence matrices $B_k(r)$ for each simplicial dimension k, where $r = 1, 2, 3$ denotes the grid cell module. We then create a block matrix $\tilde{B}_k$ for each simplicial dimension k:

$$\tilde{B}_k = \begin{bmatrix} B_k(1) & 0 & 0 \\ 0 & B_k(2) & 0 \\ 0 & 0 & B_k(3) \end{bmatrix}.$$

The simplicial convolutional filter then takes the form

$$H_k = W_0 I + \sum_{i=1}^{D} W_i (\tilde{B}_k^T \tilde{B}_k)^i + \sum_{i=1}^{D} W_{i+D} (\tilde{B}_{k+1} \tilde{B}_{k+1}^T)^i \ . \tag{4.18}$$

To measure the success of the models, we compute the Average Euclidean Distance (AED) across all time-bins:

$$AED = \frac{1}{N_{time}} \sum_{n=1}^{N_{time}} \sqrt{(x_{dec}(n) - x_{true}(n))^2 + (y_{dec}(n) - y_{true}(n))^2} \ , \tag{4.19}$$

where N_{time} is the number of time bins, (x_{dec}, y_{dec}) are the decoded xy−coordinates, and (x_{true}, y_{true}) are the ground-truth xy−coordinates. We once again measure catastrophic errors; in this context, we consider a catastrophic error to be when the euclidean distance between the model output and ground truth is greater than 0.6m.

We tuned hyperparameters to minimize AED. With the hyperparameters detailed in Supplementary Material, we calculate the AED from Equation (4.19) on the training and test data with an 80% training and 20% testing split.

The SCRNN is clearly able to learn the pattern between grid cell activity and position in the environment, and it achieves the lowest AED on the training set. However, the lowest testing AED and least amount of catastrophic errors were produced by the RNN. We remark that in this case, we optimized the networks with respect to AED, so the CAT listed is simply a byproduct of those networks; when optimized to produce the lowest CAT, results could differ. Note that a discrepancy between training and test results is expected given the fact grid cells may not encode the exact location, so training could bias the network to map neural codes for general locations to the specific labelled locations included in the training data.

Table 4.2: The AED on the training data and the AED on the testing data. While the SCRNN achieves the lowest train AED, the RNN shows the best generalizability with its low test AAE.

method	train AED	test AED	CAT
FFNN	0.0521	0.1551	47
SCNN	0.0541	0.1779	84
RNN	0.0298	0.0958	16
SCRNN	0.0286	0.1473	57
SCRNN$_{mod}$	0.0297	0.1629	75

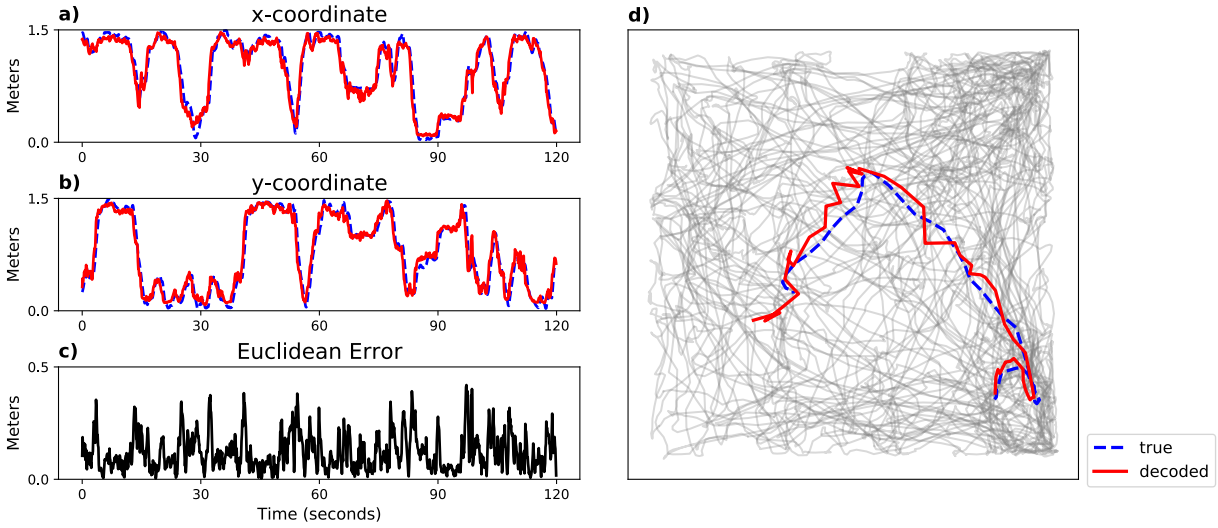


Figure 4.5: a)-c) Plots showing results from two minutes of the grid cell decoding task. a) Comparison of decoded versus ground truth x−coordinate. b) Comparison of decoded versus ground truth y−coordinate. c) Error for each time bin measured by equation (4.19). d) In grey is the ground truth position of the rat in the environment for all time bins used. Though we decode the entire trajectory shown in grey, for visual purposes, we include colorized paths showing a 5 second comparison of decoded versus ground truth position.

Chapter 5

Conclusions

In this work, we discussed and explored the relationship between deep learning and neuroscience through applications of NN-based models to PDE solving and neural decoding.

In chapter 3, we developed an adaptive sampling marking strategy for the DGM algorithm. To the best of our knowledge, this is the first adaptive scheme for DGM. Although adaptive sampling methods have been applied on PINN, they are not transferable in an efficient way to the DGM setup. Our algorithm takes advantage of the DGM algorithm and its continuous resampling of training points in order to minimize extra computational cost when selecting additional training points adaptively. We have demonstrated that our adaptive algorithm performed well on PDEs that are used as classic benchmark problems to evaluate adaptive procedures, like the oscillatory solution Poisson (3.6) and the Burger's equation (3.9). We found that the residual for those PDEs was mirroring adequately the local error landscape during the training process in the interior of the spatiotemporal domain. Hence, for those problems, we concluded that the residual is a good error indicator to drive our adaptive algorithm. In examples where the residual poorly mirrored the local error, the results were affected. For example, in the case of the Allen-Cahn (3.10) and the classical benchmark problem of the notch domain (3.11), the residual was not following the error's local behavior. For these, we observed that our adaptive procedures did not have an improvement over the DGM algorithm. Even when we used a different marking criterion more closely following the error, the improvement we observed was negligible. This indicated to us that is not enough to mark areas where the actual error is large, as long as the residual, which the training process is based upon its minimization, is small at those areas. The result

of this, is the network remaining unaware of those high error areas that the residual does not mark, hence no error improvement over DGM, is observed there.

For future work, we theorize that a modified residual term could be used in the loss function and as an error indicator helping possibly to better follow the local error in the spirit of *a posteriori* error estimators used in classical adaptive numerical methods. This might improve the performance of the ADLGM algorithm over DGM in those cases. Another potential future endeavor is to build upon the application of ADLGM to the cable equation by using ADLGM to aid in neuroscience models involving complex domains. Certain neural processes occur over irregular geometries that can proved difficult for traditional mesh-based methods like FEM and FDM. Such domains would not cause issues, however, for meshless methods like DGM or ADLGM.

In chapter 4, the simplicial convolutional framework was able to successfully decode a population containing pure grid and conjunctive grid/HD cells. Decoding position from the activity of a population containing more than just pure grid cells requires a framework robust to noisy input data, which our pre-processing is heuristically designed to be by binning spike counts and thresholding out low-activity time windows for each neuron. The results also showed defining neural activity on a simplicial complex and extracting features via simplicial convolutions that are then fed to recurrent layers improves the decoding of HD cell spike train data. It is not surprising that the SCRNN provided better results than the FFNN and SCNN, which lack recurrent connections, considering the time-series nature of the data. Further, the recurrent layers in the back-end RNN are more biologically relevant than those of an FFNN: the hidden states in the recurrent layers act as memory buffers similar to the working memory maintained within the prefrontal cortex in human brains [3].

This work features a low-complexity version of the framework to assist with network comparison and reducing computation time. For comparison purposes, the number of deep learning techniques applied to any NN architecture was kept to a minimum: only dropout, which itself was inspired by the stochastic Poisson-like firing of neurons [3, 77], was employed. For computational purposes, the maximal simplicial dimension was capped at two. Future work could involve higher complexity, for example, a biologically relevant functional simplicial complex of higher dimensions where simplices imply a connection that is experimentally supported. In both the low-complexity and biologically inspired cases, it would be worthwhile to analyze the learned filter parameters and better understand the dynamics of the simplicial convolutional layers. Similar analysis to existing NN architectures,

such as CNNs, has shown an understanding of the learned parameter space can be exploited to improve network performance [78]. Further analysis of the simplicial convolutional layers of a successfully trained decoding model could help with understanding the dynamics of the decoded population. Specific to the grid cell application, layers simulating grid cell activity within a path navigation model could employ a simplicial encoder [41] and simplicial convolutional layers. The encoder would generate simplicial complex representations of space that are then decoded using the simplicial convolutions. This would differ from existing deep learning-based navigational models [71, 72] in that it would take into account the functional connectivity in both the encoding and decoding of spatial location. Though this work focused on decoding population activity from single cell recordings, the simplicial convolutional framework has wider applicability. With only slight modifications, the framework can be adapted to other computational neuroscience tasks like brain-machine interfaces [79] or epileptic seizure detection [80], essentially any task where connectivity and higher dimensional relationships have traditionally been ignored. Beyond the scope of neuroscience, the SCRNN can be used on any dataset where shape can be characterized by a simplicial complex. A recent increase in works employing tools from Topological Data Analysis (TDA) has revealed that the underlying shape of data can be exploited to improve performance in tasks across a number of domains [24, 81, 27].

Bibliography

[1] Adrien Peyrache, Marie M Lacroix, Peter C Petersen, and György Buzsáki. Internally organized mechanisms of the head direction sense. *Nature Neuroscience*, 18(4):569–575, 2015. xiv, 2, 59, 60, 88

[2] Richard J. Gardner, Erik Hermansen, Marius Pachitariu, Yoram Burak, Nils A. Baas, Benjamin A. Dunn, May-Britt Moser, and Edvard I. Moser. Toroidal topology of population activity in grid cells. *Nature*, 602(7895):123–128, 2022. xiv, 3, 49, 63, 89

[3] Demis Hassabis, Dharshan Kumaran, Christopher Summerfield, and Matthew Botvinick. Neuroscience-inspired artificial intelligence. *Neuron*, 95:245–258, 07 2017. 1, 68

[4] Zishen Xu, Wei Wu, Shawn S. Winter, Max L. Mehlman, William N. Butler, Christine M. Simmons, Ryan E. Harvey, Laura E. Berkowitz, Yang Chen, Jeffrey S. Taube, Aaron A. Wilber, and Benjamin J. Clark. A comparison of neural decoding methods and population coding across thalamo-cortical head direction cells. *Frontiers in Neural Circuits*, 13, 2019. 1, 2, 3

[5] Markus Frey, Sander Tanni, Catherine Perrodin, Alice O'Leary, Matthias Nau, Jack Kelly, Andrea Banino, Christian F. Doeller, and Caswell Barry. Deepinsight: a general framework for interpreting wide-band neural activity. *bioRxiv*, 2019. 1, 3

[6] Ardi Tampuu, Tambet Matiisen, H. Freyja lafsdttir, Caswell Barry, and Raul Vicente. Efficient neural decoding of self-location with a deep recurrent network. *PLOS Computational Biology*, 15:1–22, 02 2019. 1, 3

[7] Samuel A. Neymotin, George L. Chadderdon, Cliff C. Kerr, Joseph T. Francis, and William W. Lytton. Reinforcement Learning of Two-Joint Virtual Arm Reaching in a

Computer Model of Sensorimotor Cortex. *Neural Computation*, 25(12):3263–3293, 12 2013. 1

[8] Nuttida Rungratsameetaweemana, Robert Kim, John T. Serences, and Terrence J. Sejnowski. Contributed session ii: Probabilistic visual processing in humans and recurrent neural networks. *Journal of Vision*, 22, 2 2022. 1

[9] Weinan E and Bing Yu. The Deep Ritz Method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, Mar 2018. 1

[10] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. 2

[11] Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018. 2, 11, 14, 16

[12] Colby L. Wight and Jia Zhao. Solving allen-cahn and cahn-hilliard equations using the adaptive physics informed neural networks. *Communications in Computational Physics*, 29(3):930–954, 2021. 2, 15, 36

[13] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021. 2, 15, 32

[14] Andreas C. Aristotelous, Edward C. Mitchell, and Vasileios Maroulas. ADLGM: An efficient adaptive sampling deep learning Galerkin method. *Journal of Computational Physics*, 477:111944, March 2023. 2

[15] Joshua I. Glaser, Ari S. Benjamin, Raeed H. Chowdhury, Matthew G. Perich, Lee E. Miller, and Konrad P. Kording. Machine learning for neural decoding. *eNeuro*, 7(4), 2020. 2, 3

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q.

Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 3

[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. 3

[18] Péter Szabó and Péter Barthó. Decoding neurobiological spike trains using recurrent neural networks: a case study with electrophysiological auditory cortex recordings. *Neural Computing and Applications*, 34(4):3213–3221, 2022. 3

[19] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. 3, 6

[20] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 3

[21] Yann Lecun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. 3

[22] Torkel Hafting, Marianne Fyhn, Sturla Molden, May-Britt Moser, and Edvard I. Moser. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801–806, 2005. 3, 49

[23] Cssio M.M. Pereira and Rodrigo F. de Mello. Persistent homology for time series and spatial data clustering. *Expert Systems with Applications*, 42(15):6026–6038, 2015. 3

[24] Andrew Marchese and Vasileios Maroulas. Topological learning for acoustic signal identification. In *2016 19th International Conference on Information Fusion (FUSION)*, pages 1377–1381, 2016. 3, 69

[25] Andrew Marchese and Vasileios Maroulas. Signal classification with a point process distance on the space of persistence diagrams. *Advances in Data Analysis and Classification*, 12(3):657–682, 2018. 3

[26] Christopher Oballe, Alan Cherne, Dave Boothe, Scott Kerick, Piotr Franaszczuk, and Vasileios Maroulas. Bayesian topological signal processing. *Discrete & Continuous Dynamical Systems - S*, 15, 01 2021. 3

[27] Vasileios Maroulas, Farzana Nasrin, and Christopher Oballe. A bayesian framework for persistent homology. *SIAM Journal on Mathematics of Data Science*, 2(1):48–74, 2020. 3, 69

[28] Adam Spannaus, Kody J.H. Law, Piotr Luszczek, Farzana Nasrin, Cassie Putman Micucci, Peter K. Liaw, Louis J. Santodonato, David J. Keffer, and Vasileios Maroulas. Materials fingerprinting classification. *Computer Physics Communications*, 266:108019, 2021. 3

[29] Monica Nicolau, Arnold J. Levine, and Gunnar Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proceedings of the National Academy of Sciences*, 108(17):7265–7270, 2011. 3

[30] Ioannis Sgouralis, Andreas Nebenführ, and Vasileios Maroulas. A bayesian topological framework for the identification and reconstruction of subcellular motion. *SIAM Journal on Imaging Sciences*, 10(2):871–899, 2017. 3

[31] Vasileios Maroulas, Cassie Putman Micucci, and Farzana Nasrin. Bayesian Topological Learning for Classifying the Structure of Biological Networks. *Bayesian Analysis*, 17(3):711 – 736, 2022. 3

[32] Jacob Townsend, Cassie Putman Micucci, John H. Hymel, Vasileios Maroulas, and Konstantinos D. Vogiatzis. Representation of molecular structures with persistent homology for machine learning applications in chemistry. *Nature Communications*, 11(1):3230, 2020. 3

[33] Kelin Xia, Xin Feng, Yiying Tong, and Guo-Wei Wei. Persistent homology for the quantitative prediction of fullerene stability. *J. Comput. Chem.*, 36(6):408–422, 2015. 3

[34] John O'Keefe. Place units in the hippocampus of the freely moving rat. *Experimental Neurology*, 51(1):78–109, 1976. 3

[35] Farzana Nasrin, Christopher Oballe, David Boothe, and Vasileios Maroulas. Bayesian topological learning for brain state classification. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1247–1252, 2019. 3

[36] Carina Curto and Vladimir Itskov. Cell groups reveal structure of stimulus space. *PLOS Computational Biology*, 4:1–13, 10 2008. 3

[37] Chad Giusti, Eva Pastalkova, Carina Curto, and Vladimir Itskov. Clique topology reveals intrinsic geometric structure in neural correlations. *Proceedings of the National Academy of Sciences*, 112(44):13455–13460, 2015. 3

[38] Miroslav Andjelković, Bosiljka Tadić, and Roderick Melnik. The topology of higher-order complexes associated with brain hubs in human connectomes. *Scientific Reports*, 10(1):17320, 2020. 3

[39] Rishidev Chaudhuri, Berk Gerçek, Biraj Pandey, Adrien Peyrache, and Ila Fiete. The intrinsic attractor manifold and population dynamics of a canonical cognitive circuit across waking and sleep. *Nature Neuroscience*, 22(9):1512–1520, 2019. 3

[40] Jacob Billings, Manish Saggar, Jaroslav Hlinka, Shella Keilholz, and Giovanni Petri. Simplicial and topological descriptions of human brain dynamics. *Network Neuroscience*, 5(2):549–568, 06 2021. 3

[41] Mustafa Hajij, Ghada Zamzmi, Theodore Papamarkou, Vasileios Maroulas, and Xuanting Cai. Simplicial complex representation learning. *Machine Learning on Graphs (MLoG) Workshop at 15th ACM InternationalWSD Conference*, 2022. 3, 69

[42] Maosheng Yang, Elvin Isufi, and Geert Leus. Simplicial convolutional neural networks. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8847–8851, 2022. 3

[43] Stefania Ebli, Michaël Defferrard, and Gard Spreemann. Simplicial neural networks, 2020. 3, 56

[44] Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lió, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1026–1037. PMLR, 18–24 Jul 2021. 3

[45] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. 6

[46] P. L. George, H. Borouchaki, P. J. Frey, P. Laug, and E. Saltel. *Mesh Generation and Mesh Adaptivity: Theory and Techniques*, chapter 17. American Cancer Society, 2007. 14

[47] Ricardo H. Nochetto, Kunibert G. Siebert, and Andreas Veeser. Theory of adaptive finite element methods: An introduction. In Ronald DeVore and Angela Kunoth, editors, *Multiscale, Nonlinear and Adaptive Approximation*, pages 409–542, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 14

[48] Tim Dockhorn. A discussion on solving partial differential equations using neural networks. *ArXiv*, abs/1904.07200, 2019. 15

[49] Willy Drfler. A convergent adaptive algorithm for poissons equation. *SIAM Journal on Numerical Analysis*, 33(3):1106–1124, 1996. 16

[50] Anders Logg, Kent-Andre Mardal, Garth N. Wells, et al. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012. 32, 36

[51] Martin S. Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E. Rognes, and Garth N. Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015. 32, 36

[52] Levi D. McClenny and U. Braga-Neto. Self-adaptive physics-informed neural networks using a soft attention mechanism. In *Proceedings of the AAAI 2021 Spring Symposium on Combining Artificial Intelligence and Machine Learning with Physical Sciences (AAAI-MLPS, Stanford, CA, USA)*, March 2021. 36

[53] Andreas C. Aristotelous, Ohannes A. Karakashian, and Steven M. Wise. Adaptive, second-order in time, primitive-variable discontinuous Galerkin schemes for a CahnHilliard equation with a mass source. *IMA Journal of Numerical Analysis*, 35(3):1167–1198, 07 2014. 40

[54] S. Brenner and R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer-Verlag, New York, NY, 1994. 40

[55] Wilfrid Rall. Theory of physiological properties of dendrites. *Annals of the New York Academy of Sciences*, 96(4):1071–1092, 1962. 43

[56] Wilfrid Rall. Electrophysiology of a dendritic neuron model. *Biophysical journal*, 2 2 Pt 2:145–67, 1962. 43

[57] Wilfrid Rall. Time constants and electrotonic length of membrane cylinders and neurons. *Biophysical Journal*, 9(12):1483–1508, 1969. 43

[58] Bruce L. McNaughton, Francesco P. Battaglia, Ole Jensen, Edvard I Moser, and May-Britt Moser. Path integration and the neural basis of the 'cognitive map'. *Nature Reviews Neuroscience*, 7(8):663–678, 2006. 48, 49, 63

[59] Takashi Yoshida and Kenichi Ohki. Natural images are reliably represented by sparse and variable populations of neurons in visual cortex. *Nature Communications*, 11(1):872, 2020. 49

[60] James J. Jun, Nicholas A. Steinmetz, Joshua H. Siegle, Daniel J. Denman, Marius Bauza, Brian Barbarits, Albert K. Lee, Costas A. Anastassiou, Alexandru Andrei, Çağatay Aydn, Mladen Barbic, Timothy J. Blanche, Vincent Bonin, João Couto, Barundeb Dutta, Sergey L. Gratiy, Diego A. Gutnisky, Michael Häusser, Bill Karsh, Peter Ledochowitsch, Carolina Mora Lopez, Catalin Mitelut, Silke Musa, Michael Okun, Marius Pachitariu, Jan Putzeys, P. Dylan Rich, Cyrille Rossant, Wei-lung Sun, Karel Svoboda, Matteo Carandini, Kenneth D. Harris, Christof Koch, John O'Keefe, and Timothy D. Harris. Fully integrated silicon probes for high-density recording of neural activity. *Nature*, 551(7679):232–236, 2017. 49, 63

[61] Nicholas A. Steinmetz, Cagatay Aydin, Anna Lebedeva, Michael Okun, Marius Pachitariu, Marius Bauza, Maxime Beau, Jai Bhagat, Claudia Bhm, Martijn Broux, Susu Chen, Jennifer Colonell, Richard J. Gardner, Bill Karsh, Fabian Kloosterman, Dimitar Kostadinov, Carolina Mora-Lopez, John OCallaghan, Junchol Park, Jan Putzeys, Britton Sauerbrei, Rik J. J. van Daal, Abraham Z. Vollan, Shiwei Wang, Marleen Welkenhuysen, Zhiwen Ye, Joshua T. Dudman, Barundeb Dutta, Adam W. Hantman, Kenneth D. Harris, Albert K. Lee, Edvard I. Moser, John OKeefe, Alfonso Renart, Karel Svoboda, Michael Husser, Sebastian Haesler, Matteo Carandini, and Timothy D. Harris. Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings. *Science*, 372(6539):eabf4588, 2021. 49, 63

[62] Joshua I. Glaser, Ari S. Benjamin, Roozbeh Farhoodi, and Konrad P. Kording. The roles of supervised machine learning in systems neuroscience. *Prog Neurobiol*, 175:126–137, 04 2019. 49

[63] Jonathan R. Wolpaw, Niels Birbaumer, Dennis J. McFarland, Gert Pfurtscheller, and Theresa M. Vaughan. Brain computer interfaces for communication and control. *Clinical Neurophysiology*, 113(6):767–791, 2002. 49

[64] Xiang Zhang, Lina Yao, Xianzhi Wang, Jessica Monaghan, David Mcalpine, and Yu Zhang. A survey on deep learning-based non-invasive brain signals: recent advances and new frontiers. *J Neural Eng*, 18(3), 03 2021. 49

[65] Michael A. Schwemmer, Nicholas D. Skomrock, Per B. Sederberg, Jordyn E. Ting, Gaurav Sharma, Marcia A. Bockbrader, and David A. Friedenberg. Meeting brain–computer interface user performance expectations using a deep neural network decoding framework. *Nature Medicine*, 24(11):1669–1676, 2018. 49

[66] Caitlin S. Mallory and Lisa M. Giocomo. From entorhinal neural codes to navigation. *Nature Neuroscience*, 21(1):7–8, 2018. 49

[67] Ila R. Fiete, Yoram Burak, and Ted Brookings. What grid cells convey about rat location. *Journal of Neuroscience*, 28(27):6858–6871, 2008. 49

[68] Daniel Bush, Caswell Barry, Daniel Manson, and Neil Burgess. Using Grid Cells for Navigation. *Neuron*, 87(3):507–520, Aug 2015. 49

[69] Nils Nyberg, Éléonore Duvelle, Caswell Barry, and Hugo J. Spiers. Spatial goal coding in the hippocampal formation. *Neuron*, 110(3):394–422, 2022. 49

[70] Alexander Mathis, Andreas V. M. Herz, and Martin Stemmler. Optimal Population Codes for Space: Grid Cells Outperform Place Cells. *Neural Computation*, 24(9):2280–2317, 09 2012. 49, 63

[71] Andrea Banino, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J. Chadwick, Thomas Degris, Joseph Modayil, Greg Wayne, Hubert Soyer, Fabio Viola, Brian Zhang, Ross Goroshin, Neil Rabinowitz, Razvan Pascanu, Charlie Beattie, Stig Petersen, Amir Sadik, Stephen Gaffney, Helen

King, Koray Kavukcuoglu, Demis Hassabis, Raia Hadsell, and Dharshan Kumaran. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429–433, 2018. 49, 69

[72] Christopher J. Cueva and Xue-Xin Wei. Emergence of grid-like representations by training recurrent neural networks to perform spatial localization. In *International Conference on Learning Representations*, 2018. 49, 69

[73] Jeffrey S. Taube. Head direction cells and the neurophysiological basis for a sense of direction. *Progress in Neurobiology*, 55(3):225–256, 1998. 59

[74] Matjaž Kukar and Igor Kononenko. Cost-sensitive learning with neural networks. In *ECAI*, 1998. 60

[75] Klara Gerlei, Jessica Passlack, Ian Hawes, Brianna Vandrey, Holly Stevens, Ioannis Papastathopoulos, and Matthew F. Nolan. Grid cells are modulated by local head direction. *Nature Communications*, 11(1):4228, 2020. 63

[76] Martin Stemmler, Alexander Mathis, and Andreas V. M. Herz. Connecting multiple spatial scales to decode the population activity of grid cells. *Science Advances*, 1(11):e1500816, 2015. 63

[77] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. 68

[78] Ephy R. Love, Benjamin Filippenko, Vasileios Maroulas, and Gunnar Carlsson. Topological convolutional layers for deep learning, 2023. 69

[79] Vernon J Lawhern, Amelia J Solon, Nicholas R Waytowich, Stephen M Gordon, Chou P Hung, and Brent J Lance. EEGNet: a compact convolutional neural network for EEG-based brain–computer interfaces. *Journal of Neural Engineering*, 15(5):056013, jul 2018. 69

[80] Artur Gramacki and Jarosław Gramacki. A deep learning framework for epileptic seizure detection based on neonatal eeg signals. *Scientific Reports*, 12(1):13010, 2022. 69

[81] Theodore Papamarkou, Farzana Nasrin, Austin Lawson, Na Gong, Orlando Rios, and Vasileios Maroulas. A random persistence diagram generator. *Statistics and Computing*, 32(5), oct 2022. 69

Appendix

A Neural Decoding

A.1 Hyperparameter Tuning

Below, we outline the different hyperparameters used throughout tuning for the results included in the main paper. For all hyperparameter tuning, we used a manual trial-and-error search method. Once the trial-and-error search method identified a hyperparameter value that produced the best results, we then conducted remaining trials with that value.

HD decoding hyperparameters. The training and test data was constructed from 20 minutes of a 38 minute session of open foraging using $t_{bin} = 100$ms. The first 25% of the data was used for testing data and the last 75% of the data was used for training. Ground truth labels were computed by taking the circular mean of recorded head directions within each time bin. During construction of the functional simplicial complex, the maximum dimension of simplices was bounded at $k = 2$. This bound was chosen due to the computational cost associated to including higher dimensional complexes. Framework hyperparameters were manually tuned within a pre-selected range to minimize CAT.

Once the best hyperparameters were identified for each of the four networks, we ran ten different trials with the ascribed hyperparameters, which can be found in Table 1, and identified the trial that resulted in the lowest CAT. When confronted with a tie, we chose the trial that had the least MAE.

Grid cell decoding hyperparameters. We use 10 total minutes of recorded neural activity and ground truth position with bin sizes of $t_{bin} = 100$ms. The first 20% of the data was used for testing data, and the last 80% of the data was used for training. Ground truth position is computed as an average of observed positions within a time bin. We employ 2

simplicial convolutional layers of degree 2, each consisting of 3 filters and using ReLU as the nonlinear activation function. The features extracted from the simplicial convolutional layers are then fed to a RNN with 3 blocks using a hidden dimension of size 50. The network trained for 50 epochs on a batch size of 16 with learning rate 0.001. Similar to the HD decoding task, framework hyperparameters were manually tuned within a pre-selected range to minimize AED.

Table 1: A table comparing the different networks based on their trial with the lowest catastrophic error. All trials were executed with 100 epochs, threshold 30, learning rate 0.001, and dropout rate 0.2. For the NN Width, the three values denote the size of each NN Layer.

	FFNN	SCNN	RNN	SCRNN
Batch Size	16	8	8	32
Max Conv. Dim.	N/A	1	N/A	1
SC Layers	N/A	1	N/A	2
N_filters	N/A	3	N/A	3
Degree	N/A	2	N/A	2
NN Layers	3	3	2	2
NN Width	128, 128, 64	128, 128, 64	N/A	N/A
Hidden Size	N/A	N/A	100	50
Test MAE	12.75	11.56	11.26	10.96
Test AAE	16.85	15.43	14.49	14.11
CAT	70	94	9	6

Table 2: Search values used for hyperparameter tuning. All hyperparameter search tests were conducted with 15 minutes of training data and 5 minutes of testing data.

Decoding Method	Hyperparameter	Range
FFNN	Epochs	100
	Batch Size	4, 8, 16, 32
	Learning Rate	0.001
	NN Layers	1, 2, 3
	Layer Width	64, 128, 256
	Activation Function	ReLU
SCNN	Intervals per Sample	1, 2, 3, 4
	Epochs	100
	Batch Size	8, 16, 32
	Learning Rate	0.001
	Dropout	0.2, 0.3
	SC Layers	1, 2
	Number of Filters	3
	NN Layers	1, 2, 3
	Layer Width	64, 128
	Activation Function	ReLU
RNN	Epochs	100
	Batch Size	8, 16, 32
	Learning Rate	0.001
	Dropout	0.2, 0.3
	NN Layers	1, 2
	Hidden Size	25, 30, 40, 50, 75, 100
	Activation Function	ReLU
SCRNN	Epochs	50, 100
	Sequence Length	3, 5, 8
	Threshold	5, 10, 30
	Batch Size	8, 16, 32
	Learning Rate	0.001
	Dropout	0.2, 0.3
	SC Layers	1, 2
	Number of Filters	3
	NN Layers	1, 2
	Hidden Size	50
	Activation Function	ReLU

Table 3: A table of hyperparameter values used during hyperparameter tuning for the grid cell decoding task. Following the same procedure as was done for the HD application, we used a manual trial-and-error search method to identify optimal hyperparameters that minimized AED.

Decoding Method	Hyperparameter	Range
SCRNN	Test % of Data	20%
	Epochs	50
	Sequence Length	3, 5, 8
	Threshold	1, 10, 20
	Batch Size	8, 16, 32, 64
	Learning Rate	0.0001, 0.001
	Dropout	0.2, 0.3
	SC Layers	1, 2, 3, 5
	Number of Filters	2, 3, 5
	NN Layers	1, 3, 5
	Hidden Size	25, 40, 50, 75, 100, 200
	Activation Function	ReLU

A.2 Analysis of Different Train/Test Splits

In this section, we show results for the HD decoding task with two different training and testing data splits: one with 50% training, 50% testing (Figure 1) and another with 25% training, 75% testing (Figure 2). Following the same manual hyperparameter tuning procedure mentioned in Appendix A.1, we tuned the FFNN, SCNN, SCRNN, and RNN to minimize CAT. For both splits, we observe the SCRNN provides the best results.

Training: 10 minutes, Testing: 10 minutes The results shown in Figure 1 correspond to 10 minutes of training data and 10 minutes of testing data. Note, the SCRNN has the lowest MAE, AAE, and CAT across all architectures.

Training: 5 minutes, Testing: 15 minutes Next, the results in Figure 2 correspond to 5 minutes of training data and 15 minutes of testing data. Again, the SCRNN has the lowest MAE, AAE, and CAT across all architectures.

A.3 Neural Data

In this section, we include visualizations of the neural activity in the form of raster plots. The spike trains of all 22 HD cells used in the HD decoding task are shown in Figure 3. Note how cells can differ in firing pattern. The pre-processing procedure of our framework accounts for this in the binarization step by thresholding row-wise, meaning a neuron's activity is compared to itself. Otherwise, neurons that fire more frequently would lead to the complete zeroing out of activity of neurons that, by comparison, rarely fired.

Figure 4 shows the spike trains from a single module used in the grid cell decoding task. This module consists of 166 neurons. The other two modules (not shown) used in the grid cell decoding task consist of 167 and 149 grid cells. The increased number of neurons compared to the HD system is necessary to encode a two-dimensional variable and makes the grid cell decoding task more complex.

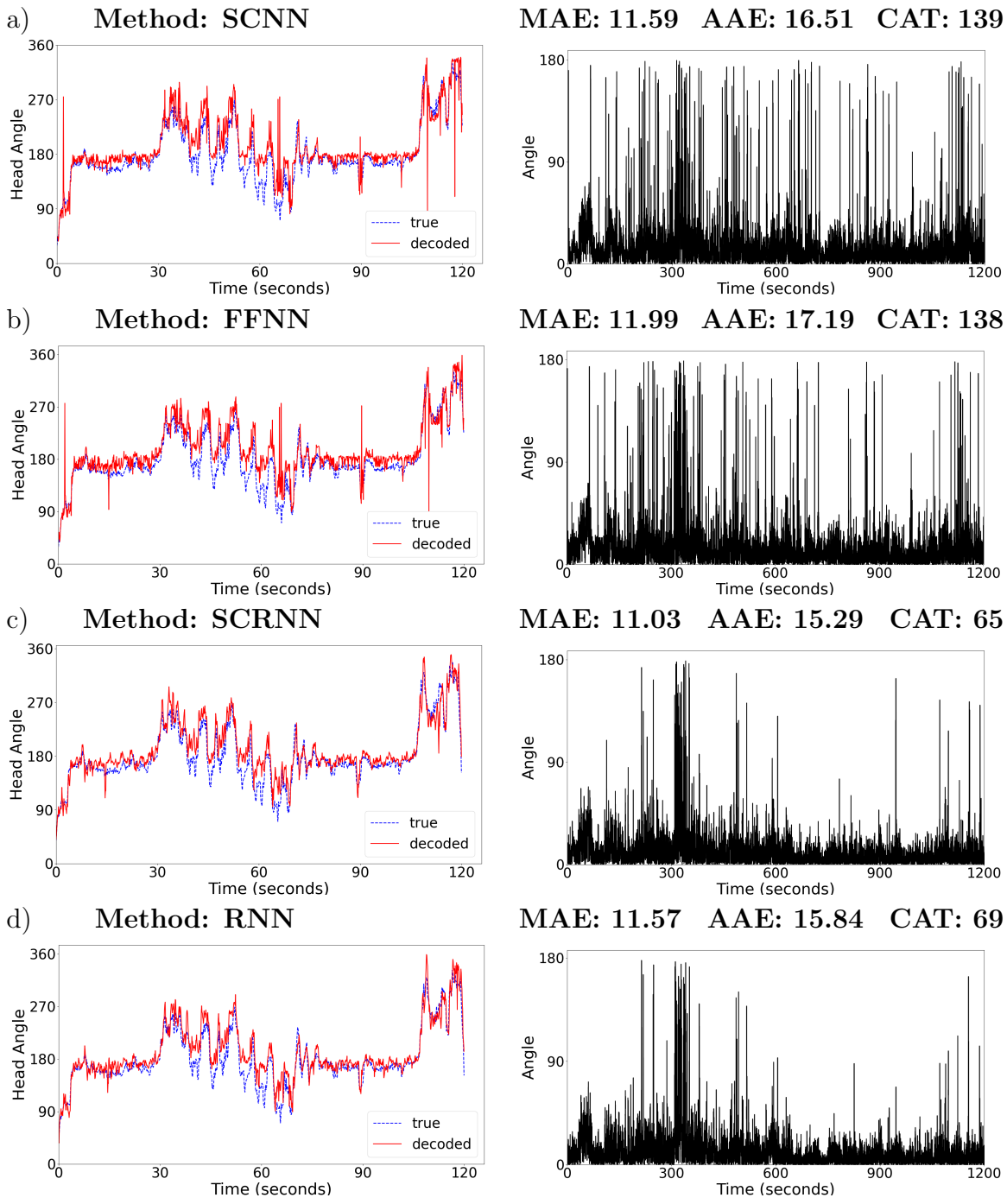


Figure 1: Plots depicting the true head angle and the predicted head angle for the first two minutes with the catastrophic error for each time bin for four different networks, a) SCNN, b) FFNN, c) SCRNN, and d) RNN.

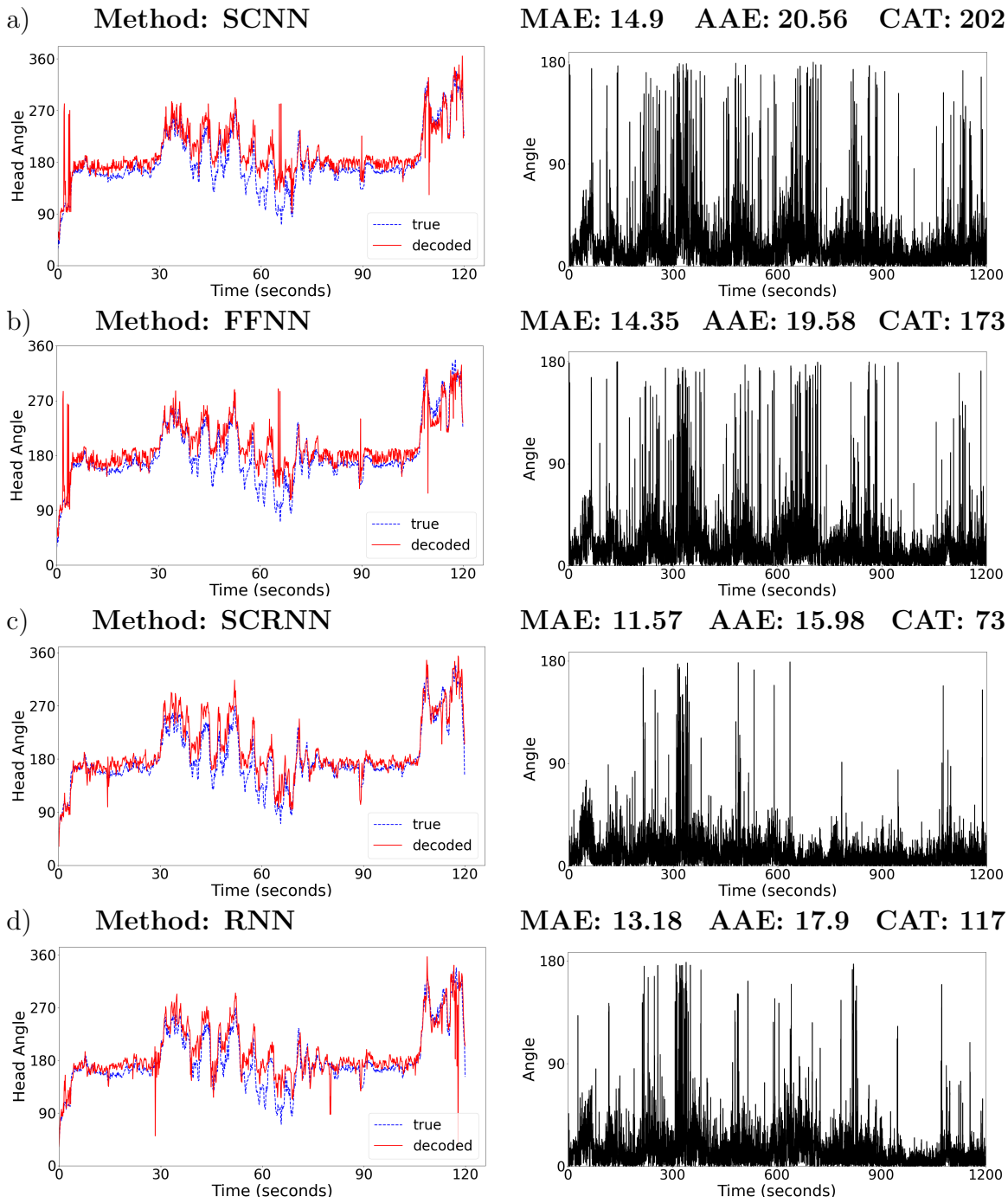


Figure 2: Plots depicting the true head angle and the predicted head angle for the first two minutes with the catastrophic error for each time bin for four different networks, a) SCNN, b) FFNN, c) SCRNN, and d) RNN.

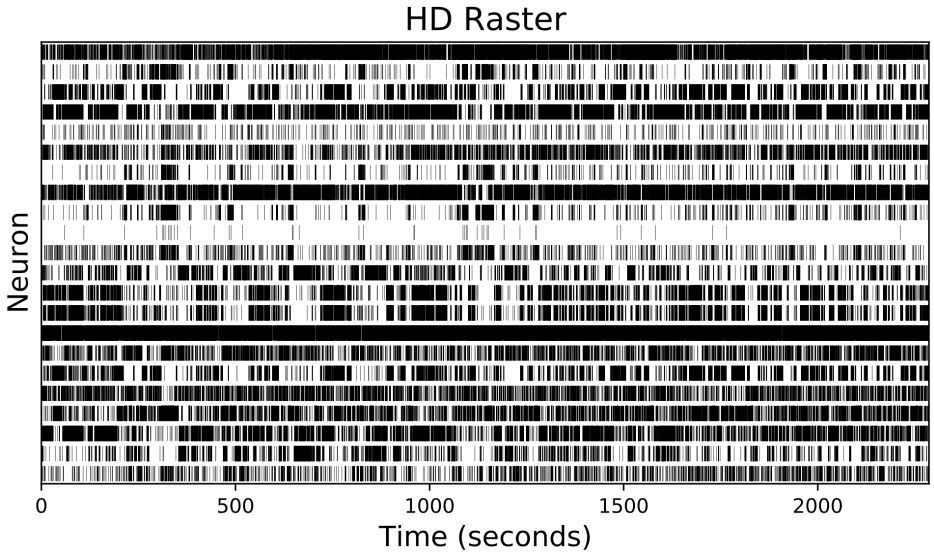


Figure 3: A raster plot showing the HD neural activity from 'Mouse28-140313' in the source data [1]. The HD system includes 22 neurons recorded over a 38 minute period while the mouse foraged in an open environment.

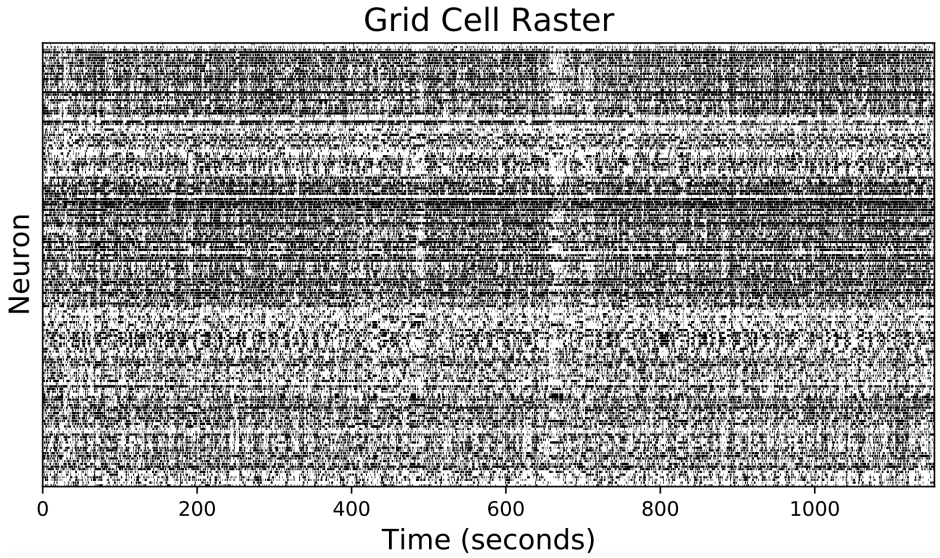


Figure 4: A raster plot showing the neural activity from a single module of 166 grid cells recorded over a 19 minute period while the rat foraged in an open 1.5×1.5 meter environment. The data is labeled R1 day 1 in the source data [2].

Vita

Edward Mitchell was born in Tampa, Florida, where he first became interested in math while attending Carrollwood Elementary School. Originally driven by a love for teaching, he attended the University of South Florida to pursue a Bachelors of Science in mathematics with the hopes to one day become a professor.

Upon graduation from the USF Honors College in Spring 2016, he moved to Knoxville, Tennessee to begin an internship at Oak Ridge National Labs before eventually starting his pursuit of a PhD in mathematics at the University of Tennessee-Knoxville. Later, while attending UTK, he worked closely with the Computational and Applied Mathematics Group at ORNL; it is this work that first exposed him to deep learning. In Summer 2020, he joined the Maroulas Research Group, through which he began a collaboration with the US Army Research Lab. Working under his advisor Professor Vasileios Maroulas with ARL, he learned tools from topological data analysis to combine with those he already knew from deep learning. This research-intensive time persuaded Eddie to pursue a non-teaching career focused solely on research.

As of Spring 2023, Eddie has accepted a position as Head/Principal Machine Learning Research Scientist at Joe Gibbs Human Performance, LLC.