

UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

Brain-Inspired Recurrent Neural Networks for Biomedical Applications

Permalink

<https://escholarship.org/uc/item/0xc25962>

Author

Yin, Yue

Publication Date

2024

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA,
IRVINE

Brain-Inspired Recurrent Neural Networks for Biomedical Applications

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Biomedical Engineering

by

Yue Yin

Dissertation Committee:
Professor Zoran Nenadic, Chair
Adjunct Professor Gregory Brewer
Associate Professor James Brody

2024

DEDICATION

To my family

TABLE OF CONTENTS

	Page
LIST OF FIGURES	v
LIST OF TABLES	ix
ACKNOWLEDGMENTS	x
VITA	xi
ABSTRACT OF THE DISSERTATION	xiii
1 Introduction	1
2 Background	4
2.1 Biological and Artificial Neurons	4
2.2 Feed-Forward Neural Network	7
2.3 Vanilla Recurrent Neural Network	11
2.4 Gradient-Free RNNs	14
2.4.1 Echo State Network	14
2.4.2 FORCE Learning	16
2.4.3 Full-FORCE	18
3 Multi-Layer Full-FORCE with Dynamical Data Coupling	20
3.1 Architecture and Learning Rule	21
3.2 Effect of dynamical data coupling	25
3.2.1 Oscillation Task: Description	25
3.2.2 Generating Oscillation: Results	26
3.2.3 Interval Timing Recognition Task: Description	28
3.2.4 Matching Interval Time: Results	29
3.3 Effect of multi-layer architecture	31
3.3.1 Multi-dimensional Body Joint Movements: Dataset	31
3.3.2 Generating Body Joint Movements: Results	32
3.4 Improvement on processing EEG signals	35
3.4.1 EEG Dataset	35
3.4.2 Classification of Eye State: Results	36

4	Deep Continuous Local Learning with Recurrent Connections	38
4.1	Recurrent Continuous Local Learning	38
4.1.1	Learning Rule Derivation	38
4.1.2	Recurrent Continuous Local Learning vs. Full-FORCE	43
4.2	REDECOLLE	45
4.2.1	Network Architecture	46
4.2.2	REDECOLLE Dynamics	47
4.2.3	Blood Glucose Level Prediction: Description	49
4.2.4	Forecasting Blood Glucose Level: Results	49
5	Conclusion	52
5.1	Summary	52
5.2	Future Work	53
	Bibliography	55
	Appendix A Eigenvalues & Unit Activities for Oscillation Generation	63
	Appendix B Activities of the 1st Hidden Layer for Movement Generation	67

LIST OF FIGURES

	Page
2.1 Biological and artificial neurons. (a) Structure of a biological neuron consisting of dendrites, cell body, and axon [Mhatre, 2020]. (b) Mathematical model of an artificial neuron. This figure is a modification from [Chrislb, 2005]. . .	5
2.2 Illustration of the gradient descent algorithm. Parameter θ is set at a random initial value. Each step parameter is updated based on the calculated gradient $\nabla J(\theta)$ and learning rate. A proper choice of learning rate will ensure the cost function reaches minimum with optimized parameter. This figure is a modification from [Liu, 2022].	6
2.3 Biological and artificial neural networks. (a) Multiple layers in a biological neural network [Swanson et al., 2017]. (b) Artificial feed-forward neural network with two hidden layers. This figure is a modification from [Rathi, 2018].	8
2.4 Overview of four different error transportation mechanisms. Neural networks with two hidden layers are used as example. W_i represents feed-forward weight matrices while B_i in gray denotes fixed random matrices. Error e is computed via cost function J with network prediction $\hat{y}$ and target value y . Red arrows indicate error is involved in the backward pass. (a) Backpropagation (BP) requires transpose of feed-forward weight matrices to compute the gradient. (b) Feedback alignment (FA) uses fixed random matrices and error from the upper layers to generate intermediate targets. (c) Direct feedback alignment (DFA) broadcasts final error to hidden layers via fixed random matrices. (d) Direct random target projection (DRTP) combines target and fixed random matrices to provide hidden layers with local learning objectives. This figure is a modification from [Frenkel et al., 2021, Nøkland, 2016].	9
2.5 Architecture of recurrent neural network with one hidden layer. At time t , input at that time x_t is sent into the hidden layer with feed-forward weight W_{in} . Activities of hidden layer h_t generates the network output at that time as $\hat{y}_t$ using output matrix W_{out} . Different from feed-forward neural network, hidden layer in the RNN has recurrent connection, delineates as weight matrix W . This matrix connects the activity of hidden layer at present time (h_t) to the one at next time step, i.e., h_{t+1}	12
2.6 Recurrent neural network unrolled in time. When in the unrolled form, each time step of the sequence is depicted as a separate frame, with the recurrent weight W connecting the adjacent frames.	12

2.7	Architecture of echo state network (ESN). Here $\mathbf{u}$, $\mathbf{x}$, $\hat{\mathbf{y}}$ represents the input, hidden unit, and output, respectively. Input weight $\mathbf{W}_{\text{in}}$, recurrent weight $\mathbf{W}$, and feedback weight $\mathbf{W}_{\text{fb}}$ are fixed. Only the readout weight $\mathbf{W}_{\text{out}}$ is trainable. This figure is a modification from [Jaeger and Haas, 2004].	15
2.8	Architectures of FORCE learning. Recurrent weight $\mathbf{W}$, in addition to the readout weight $\mathbf{W}_{\text{out}}$, are trainable. This figure is a modification from [Sussillo and Abbott, 2009].	17
2.9	Architecture of full-FORCE (left: target-generating network; right: task-performing network). The target-generating network contains no trainable weights as input weight $\mathbf{W}_{\text{in}}^{\text{D}}$, recurrent weight $\mathbf{W}^{\text{D}}$, and output weight $\mathbf{W}_{\text{out}}^{\text{D}}$ are all fixed. Notice here the target-generating network receives actual target, not network output. Dynamics of this network serve to provide learning targets for the task-performing network, in which recurrent weight $\mathbf{W}$ and output weight $\mathbf{W}_{\text{out}}$ are trainable. This figure is a modification from [DePasquale et al., 2018].	18
3.1	The architecture of a multi-layer full-FORCE with dynamical data coupling (left: Task-Performing Network (TPN); right: Target-Generating Network (TGN)). In the TGN all the connections including input, feed-forward, and recurrent weights are fixed (in gray), while the TPN consists of trainable weights (in red). The couplings between each layer of the TGN and the TPN are indicated by the dashed black arrows.	22
3.2	Oscillation task. The input $u(t)$ for this task is a brief pulse and the the network learns to generate a frequency-modulated oscillation: $y(t) = \sin(\omega(t)t)$ with $\omega(t)$ increasing linearly from 2π to 6π for the first half of the oscillation and doing the opposite for the second half.	25
3.3	MSE at the end of each iteration for the oscillation task. We use full-FORCE network of two different sizes (200 & 400 units). Both are trained with two weight update frequencies (500 & 1000 per epoch). Network with dynamical coupling between the TGN and the TPN achieve lower errors. The MSE values depicted in the figure are the averaged results over 50 different network initializations.	27
3.4	Example of interval timing recognition task. The network is presented with two pulses and the delay between them is determined from an uniform distribution (min: 0.1 s, max: 1.0 s). The goal is for the network to generate a bump after the second pulse with the same delay. A hint signal (a function that ramps linearly in time between the pulses and then decreases linearly at the same rate following the second pulse) can be used to assist the learning.	28
3.5	Performance on interval matching task using the full-FORCE network with 300 units. The dynamical coupling is shown to help full-FORCE generate the delayed signal closer to the target time after both 1 and 5 iterations of training. We test full-FORCE with 10 different initialized weights (indicated by the vertical error bars in all four panels) on 10 cases with delays ranging from 0.1 to 1.0 seconds evenly.	30

3.6	Example of joint movement. Here the 3 coordinates (x , y , and z) of one joint movement is plotted.	31
3.7	Results for the task of generating body joint movements. The top panel shows the MSE during the training stage while the bottom three show the MSE for the testing at the end of each training epoch. Three different network sizes with total number of units: 240 (left 2 plots), 360 (middle 2 plots), and 480 (right 2 plots) are used. For each network size, we distribute the units evenly into one, two and three layers. For the case of two layer networks, we explore the uneven distributions of units between the two layers as well. The numbers in each plot's legend indicate the amount of units in each layer, with early layer comes first. Each curve is the average of 10 runs.	32
3.8	EEG signals and eye state. EEG signals from four selected channels are plotted along with corresponding eye state.	35
3.9	Accuracy (averaged over 10 RNN initializations) for the eye state classification measured at the end of each training epoch on all testing samples. Outcomes from our networks are plotted along with the one produced by full-FORCE (one-layer 400 units without coupling). Both coupling and an increased number of layers in network architecture improve the classification accuracy on this task: coupling shows marginal gains while the two-layer networks offer compelling improvement.	37
4.1	Architecture of recurrent continuous local learning. All the weights (input, recurrent, and output) are trainable in this RNN model.	39
4.2	Comparing full-FORCE w/ dynamical coupling and recurrent continuous local learning on oscillation generation task. Both network contains one hidden layer with 100 units. Both networks are trained over 100 epochs and the MSE at the end of each training iteration is plotted.	43
4.3	Generated oscillations after training for full-FORCE w/ dynamical coupling and recurrent continuous local learning. The black dash trace is the target output. Recurrent continuous local learning is able to generate oscillation closely follows the target while full-FORCE with dynamical coupling struggles in the beginning portion of the task.	44
4.4	Comparing full-FORCE w/ dynamical coupling and recurrent continuous local learning on joint movement generation task. Both network contains one hidden layer with 200 units. Both networks are trained over 50 epochs. . . .	44
4.5	Joint movements during walking generated by full-FORCE w/ dynamical coupling and recurrent continuous local learning. Both networks generate patterns close to target. The shaded frame is one example where recurrent neural network outperforms full-FORCE with dynamical coupling.	45
4.6	Architecture of recurrent deep continuous local learning (REDECOLLE). REDECOLLE features deep architectures with hidden layers connected using feed-forward weights. Unlike traditional RNNs that are trained by back-propagating error signal from the output layer to the hidden ones, REDECOLLE uses additional network (target-generating network) to generate layer specific learning objectives.	47

4.7	Prediction of blood glucose level for Patient #1 using REDECOLLE.	50
4.8	Prediction of blood glucose level for Patient #2 using REDECOLLE.	51
4.9	Prediction of blood glucose level for Patient #3 using REDECOLLE.	51

LIST OF TABLES

	Page
3.1 Summary of testing results for oscillation task	26
3.2 Summary of multi-dimensional pattern generation experiment	33
4.1 Summary of glucose level forecasting results (RMSE values are shown) . . .	50

ACKNOWLEDGMENTS

As I reach the end of my PhD journey, I am deeply thankful to those who have supported and helped me throughout this process.

First of all, I express my sincere gratitude to my research advisor, Professor Emre Neftci. Your constant guidance, unwavering support, and insightful feedback have been instrumental in shaping this research and helping me grow both academically and personally.

Many thanks to my dissertation committee: Professor Zoran Nenadic (chair), Professor Gregory Brewer, and Professor James Brody for your time, expertise, and constructive suggestions. I would also like to thank Professor Jeff Krichmar and Professor Beth Lopour for serving on my PhD qualifying exam committee and providing critical feedback on my research ideas.

Special thanks are due to my colleagues at the Neuromorphic Machine Intelligence (NMI) Lab: Nick Alonso, Dan Barsever, Travis Bartley, Georgios Detorakis, Tim Hin Wai Lui, Takashi Nagata, Kenneth Stewart, and Jinwei Xing. Your camaraderie, discussions, and encouragement have made this journey both enjoyable and inspiring.

Next, I want to acknowledge both former (Clare Cheng) and current (Maggie Mulcare) academic program coordinators of the BME department. You are tremendously helpful in advising and guiding me through the milestones of my degree.

My dissertation is based upon work funded by Korean Institute of Science and Technology (KIST) and the National Science Foundation (NSF) under grant 1652159. Without generous financial support, this work would not have been possible.

Lastly, I am profoundly grateful to my family, especially my parents. Your love, patience, and encouragement have always been my anchor through the challenges of this endeavor.

Note that portions of Chapter 3 is a reprint of the material as it appears in *Improving full-FORCE with Dynamical Data Coupling and Multilayer Architecture*, used with permission from 2021 IEEE Biomedical Circuits and Systems Conference (BioCAS). The co-author listed in this publication directed and supervised research which forms the basis for the dissertation.

VITA

Yue Yin

EDUCATION

Doctor of Philosophy in Biomedical Engineering University of California, Irvine	2024 <i>Irvine, CA</i>
Master of Science in Biomedical Engineering Carnegie Mellon University	2017 <i>Pittsburgh, PA</i>
Bachelor of Science in Biomedical Engineering University of Wisconsin - Madison	2014 <i>Madison, WI</i>

WORK & RESEARCH EXPERIENCE

Graduate Student Researcher University of California, Irvine	2018–2022 <i>Irvine, CA</i>
Machine Learning Intern Infineon Technologies	2022 <i>Irvine, CA</i>
Research Intern, Machine Learning-Enabled Biosensors Palo Alto Research Center (PARC)	2021 <i>Palo Alto, CA</i>
Intern, Biomedical Engineer InvivoSciences	2013–2015 <i>Madison, WI</i>
Summer Scholar Stowers Institute for Medical Research	2012 <i>Kansas City, MO</i>

TEACHING EXPERIENCE

TA for BME 130 Biomedical Signals and Systems University of California, Irvine	Fall 2019, 2020, 2021 <i>Irvine, CA</i>
TA for BME 60C Engineering Analysis/Design: CAD University of California, Irvine	Spring 2019 <i>Irvine, CA</i>
TA for BioSci 94 Organisms to Ecosystems University of California, Irvine	Winter 2019 <i>Irvine, CA</i>

PUBLICATIONS

- Yin, Y., & Neftci, E. O. (2021). Improving full-FORCE with Dynamical Data Coupling and Multilayer Architecture. In IEEE Biomedical Circuits and Systems Conference (BioCAS) (pp. 1-6). IEEE.
- Gao, X., Yin, Y., Daily, N. J., Engel, T., Pang, L., Carlson, B. E., & Wakatsuki, T. (2017). Computational Model for Simulating Drug-Induced Arrhythmia Sensitivity of Human iPSC-derived Cardiomyocytes. *bioRxiv*, 236240.
- Wakatsuki, T., Carlson, B., Daily, N., & Yin, Y. (2017). Analyzing Effects of Cardiac Fibroblasts on Aging Hearts Using 2D/3D Co-culture Systems and Computational Simulation. *Journal of Pharmacological and Toxicological Methods*, 88, 203.
- Daily, N. J., Yin, Y., Kemanli, P., Ip, B., & Wakatsuki, T. (2015). Improving cardiac action potential measurements: 2D and 3D cell culture. *Journal of Bioengineering & Biomedical Science*, 5(3).

ABSTRACT OF THE DISSERTATION

Brain-Inspired Recurrent Neural Networks for Biomedical Applications

By

Yue Yin

Doctor of Philosophy in Biomedical Engineering

University of California, Irvine, 2024

Professor Zoran Nenadic, Chair

Training artificial recurrent neural networks (RNNs) has traditionally relied on an offline gradient-based learning method: backpropagation through time (BPTT). However, this technique is computationally expensive and biologically implausible as the information for the entire time sequence needs to be stored for backward gradient calculation. Reservoir computing, a brain-inspired learning framework, solves this problem by allowing online adaptations in which parameters can be updated at every step. In this dissertation, we first improve the stability of the state-of-the-art reservoir computing technique (full-FORCE) by dynamically coupling the data with the network (teacher forcing). This coupling has the same effect as learning online. We also extend full-FORCE to multiple layers while permitting online local weight updates for each layer. Next, we introduce a new recurrent neural network that implements a continuous local learning rule and is compatible with the modern machine learning library for gradient-based optimization. This RNN inherits synaptic dynamics, features deep architecture, and enables online learning for weight modification. We test these new models on real-world biomedical time-series tasks, including multi-dimensional movement pattern generation, electroencephalogram (EEG) signal classification, and blood glucose level prediction, and show improved performances.

Chapter 1

Introduction

Recurrently connected neural circuits are ubiquitous in the brain, and they have inspired the creation of artificial recurrent neural networks (RNNs) specialized in processing spatiotemporal information [Rumelhart et al., 1988]. Variants of RNN remain unsurpassed in solving complex tasks, including natural language processing, robotic controls, and time series prediction [Goodfellow, 2016]. To train RNNs, gradient-based learning rules are often used. And backpropagation algorithm becomes indispensable when calculating the gradients across different layers of the network [Rumelhart et al., 1986] at each time step [Werbos, 1990]. However, running gradient-based RNN learning rules with backpropagation requires a large amount of device memory and computational power [Shalev-Shwartz and Ben-David, 2014], making it a challenge to implement on energy-efficient hardware at the edge for real-world applications [Chen and Ran, 2019, Deng, 2019].

One way to address this issue is reservoir computing. Specifically, echo state network (ESN) [Jaeger and Haas, 2004] feeds input signal into a fixed, randomly connected, high dimensional dynamical system (also called a reservoir). Desired output can subsequently be extracted from the reservoir via a readout layer. Liquid state machine (LSM) [Maass et al., 2002]

shares a similar network architecture and learning rules with ESN, but the individual unit in the network follows spike-based dynamics. For both ESN and LSM, only the weights in the readout layer are adjustable, and the learning is achieved mainly offline using linear regression. FORCE learning [Sussillo and Abbott, 2009] improves the training by imposing a rank-one modification of the connection matrix with the online recursive least squares (RLS) method. Recently, full-FORCE learning [DePasquale et al., 2018] introduces an additional network that provides learning targets to the hidden units during the training phase. This configuration enables the full-rank update of recurrent weights and provides a larger training capacity than FORCE learning, especially when the network’s number of units is limited. Chapter 2 presents an overview of different RNN architectures and training modalities.

In Chapter 3, we seek to further enhance full-FORCE’s performance and stability through two methods: implementing dynamical coupling between existing networks and allocating units in the single-layer reservoir into a multi-layer architecture. We show that both techniques improve the original model’s performance in pattern generation, interval matching, and EEG signal classification tasks. Incorporating synchronization into the model stabilizes the learning process while adding hidden layers provides more capacity to learn complex representations.

Chapter 4 introduces another brain-inspired RNN by adding recurrent connections to previously developed deep continuous local learning (DECOLLE) [Kaiser et al., 2020]. This new network merges local online learning rules and spike-based models in computational neuroscience with gradient descent techniques in machine learning. It also surpasses dynamically coupled full-FORCE with better performance. We apply this RNN to predict blood glucose levels and show promising results.

Finally, in Chapter 5, we conclude this thesis by summarizing the main contributions of this study and discussing future directions to improve and implement the RNN models. Overall, a brain-like computational model to process sequential information is of great value [Schuman

et al., 2022]. Neuromorphic hardware provides an energy-efficient way of computing [Burr et al., 2017, Schuman et al., 2017, Yang et al., 2020]. Coupling such device with a biologically plausible RNN that allows local on-chip learning has the potential to be applied more broadly [Esser et al., 2015, Roy et al., 2019, Christensen et al., 2022].

Chapter 2

Background

The human brain is a remarkably versatile and robust learning apparatus that can process information at a high speed while consuming little energy. Engineering artificial intelligence systems that mimic the function of the brain has been a long quest. This effort starts by drawing insights from the brain's structure at different levels, including single neurons and biological neural networks.

2.1 Biological and Artificial Neurons

A neuron is the fundamental computational unit in the human brain [Kandel et al., 2000]. Its basic structure is shown in Figure 2.1 (a). Input signal reaches a neuron through dendrites. The neuron's cell body (soma) integrates all the inputs to determine if the combined signal is strong enough to trigger an action potential or spike [Hodgkin and Huxley, 1952]. This electrical impulse is then transmitted toward the axon terminals, where it is converted into a chemical signal (neurotransmitter) to cross the synapse and enter the next neuron. Synaptic plasticity [Citri and Malenka, 2008], the mechanism that underlies learning and memory, is

the adaptation of synaptic strength based on neural activity [Abbott and Nelson, 2000].

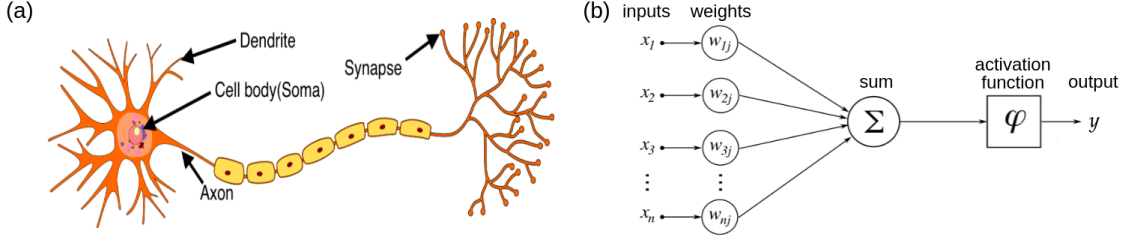


Figure 2.1: Biological and artificial neurons. (a) Structure of a biological neuron consisting of dendrites, cell body, and axon [Mhatre, 2020]. (b) Mathematical model of an artificial neuron. This figure is a modification from [Chrislb, 2005].

The mathematical model of an artificial neuron [McCulloch and Pitts, 1943, Minsky and Papert, 2017] is designed to emulate the information transmission process of the biological one. As delineated in Figure 2.1 (b), each input value (x_i) is associated with a weight (w_{ij}) that represents the strength of the connection. The neuron computes a weighted sum of all input values (often times an extra bias term is included). An activation function [Sharma et al., 2017, Nwankpa et al., 2018] is then applied to the weighted sum to generate the output (y). So the output can be calculated as:

$$y = \varphi \left(\sum_{i=1}^n w_{ij} x_i \right) \quad (2.1)$$

Here the activation function introduces non-linearity into the model, enabling it to express complex patterns [Leshno et al., 1993]. Common activation functions include:

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- ReLU (Rectified Linear Unit): $f(x) = \max(0, x)$
- Tanh (Hyperbolic Tangent): $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Learning in artificial neurons is equivalent to adjusting weights so that the output is closer to the targeted value.

Gradient descent [Bengio et al., 1990, LeCun et al., 1998] is one commonly used algorithm to find the optimal parameters (weights and biases). First, a cost function $J(\theta)$ is defined, which measures the discrepancy between the model output and the actual target. θ represents a parameter of the model. The gradient (partial derivative) of the cost function with respect to the parameter $\nabla J(\theta)$ provides the direction of the steepest increase for the cost function. So, to reduce the cost function, the model parameter needs to be adjusted against the gradient. One can formulate the gradient update rule as:

$$\theta := \theta - \alpha \cdot \nabla J(\theta) \quad (2.2)$$

where α is the learning rate.

Gradient descent is an iterative process. The learning rate determines the number of steps taken toward the minimum. A small learning rate ensures stable convergence but may be slow, while a large one accelerates convergence but risks overshooting the minimum [Ruder, 2016]. Figure 2.2 depicts how the parameter is adjusted each step to reach the minimum of the cost function using gradient descent.

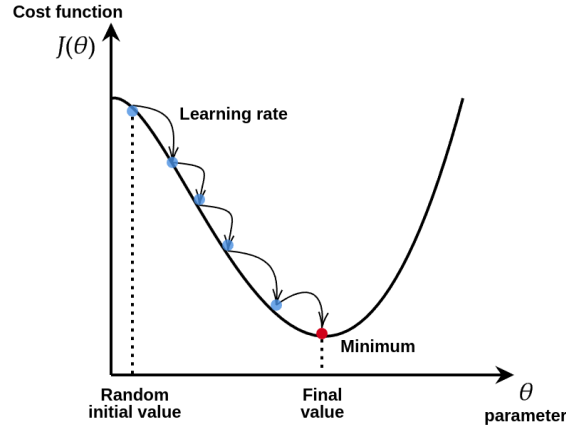


Figure 2.2: Illustration of the gradient descent algorithm. Parameter θ is set at a random initial value. Each step parameter is updated based on the calculated gradient $\nabla J(\theta)$ and learning rate. A proper choice of learning rate will ensure the cost function reaches minimum with optimized parameter. This figure is a modification from [Liu, 2022].

The gradient descent algorithm can be divided into three main categories: batch gradient descent [Rumelhart et al., 1986], stochastic gradient descent [Bottou et al., 1991], and mini-batch gradient descent [LeCun et al., 2002], based on how the training dataset is used during the parameter optimization. Batch gradient descent uses the entire training dataset to compute the gradient and update the parameters in one iteration. In contrast, stochastic gradient descent adjusts the parameter using only one training example at a time before it moves on to the following example. Mini-batch gradient descent combines aspects of the former two methods by using a small, random subset of the training data to compute the gradient for each parameter update step. In practice, the choice of method often comes down to the computational time and cost, the accuracy of gradient calculation, and the convergence rate. An epoch is one complete pass of all the training data through the optimization procedure.

2.2 Feed-Forward Neural Network

Neurons in the brain connect in a complex yet structured manner. One example is laminar organization [Bear et al., 2020], which indicates that neurons are arranged in layers at specific regions [Raizada and Grossberg, 2003]. The size and shape of the neuronal bodies differentiate each layer, which dictates that each layer is involved in different aspects of processing Shipp [2007]. Figure 2.3 (a) displays an early drawing of multiple layers in a biological neural network. These layered structures ultimately support the integration of information at different levels and the performance of complex tasks.

Artificial neural networks share these layered connections with their biological counterparts. A feed-forward neural network [Bebis and Georgiopoulos, 1994, Svozil et al., 1997] is the simplest form of neural network architecture, where multiple layers of neurons (nodes) connect in an unidirectional order. The input layer is the first one to receive input data, and

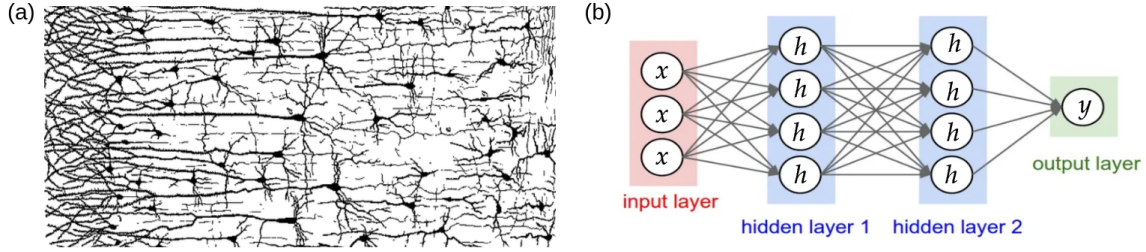


Figure 2.3: Biological and artificial neural networks. (a) Multiple layers in a biological neural network [Swanson et al., 2017]. (b) Artificial feed-forward neural network with two hidden layers. This figure is a modification from [Rathi, 2018].

each node in this layer represents a feature or attribute of the data. The final layer produces the output of the network. Depending on the task, it could generate a classification label, a regression value, or other types of prediction. One or more hidden layers are stacked one after another between the input and output layers. Information flows from the input to the output layer through hidden ones. Figure 2.3 (b) shows a feed-forward neural network with two hidden layers.

Each neuron in a hidden layer connects to the ones in the adjacent layers via feed-forward weights. Mathematically, summation of weighted input and nonlinear transformation occur at each neuron in the same fashion described in the previous section. The results are passed to the next layer using another set of feed-forward weights. As the number of hidden layers increases, so does the amount of nonlinear transformation on the input data. A multi-layer architecture enables the network to approximate more complex functions, capture hierarchical patterns, and increase model capacity [Eldan and Shamir, 2016, Thomas et al., 2017, Uzair and Jamil, 2020].

Like an artificial neuron, a feed-forward neural network can perform parameter updates using gradient-based optimization. Precisely, backpropagation [Rumelhart et al., 1986, Rojas and Rojas, 1996], or backward propagation of errors, is required to compute the gradient in the hidden layers. Forward propagation is the step mentioned above of computing the network’s output using input, feed-forward weights, and nonlinear transformation layer by layer. A

cost function (error) is defined based on the output and the desired target. For the backward propagation, the gradient of the cost function with respect to parameters in each layer is calculated using the chain rule of calculus. Essentially, the error information is passed backward through the network, from the output layer to the input layer, via hidden layers.

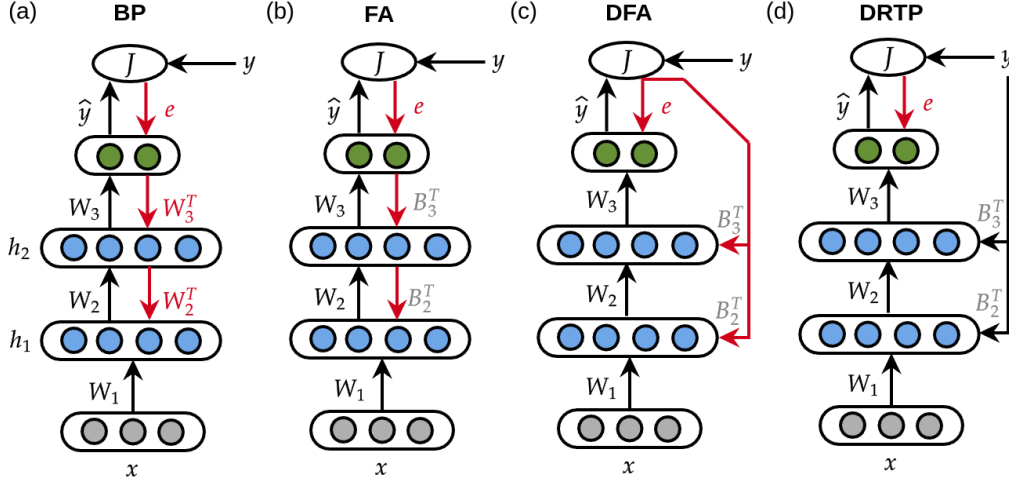


Figure 2.4: Overview of four different error transportation mechanisms. Neural networks with two hidden layers are used as example. W_i represents feed-forward weight matrices while B_i in gray denotes fixed random matrices. Error e is computed via cost function J with network prediction $\hat{y}$ and target value y . Red arrows indicate error is involved in the backward pass. (a) Backpropagation (BP) requires transpose of feed-forward weight matrices to compute the gradient. (b) Feedback alignment (FA) uses fixed random matrices and error from the upper layers to generate intermediate targets. (c) Direct feedback alignment (DFA) broadcasts final error to hidden layers via fixed random matrices. (d) Direct random target projection (DRTP) combines target and fixed random matrices to provide hidden layers with local learning objectives. This figure is a modification from [Frenkel et al., 2021, Nøkland, 2016].

Despite its effectiveness in training artificial neural networks, backpropagation (Figure 2.4 (a)) is considered biologically implausible [Lillicrap et al., 2020] for three main reasons: the weight transport problem [Crick, 1989, Saxe et al., 2013], the non-locality of the error signal [Baldi and Sadowski, 2016], and update locking [Czarnecki et al., 2017, Jaderberg et al., 2017]. When computing the gradient across multiple layers using backpropagation (BP), the transpose of the feed-forward weight matrices is involved. Essentially, the forward and backward passes share the same weights. This weight symmetry requirement is hardly

feasible in neural circuits since it requires post-synaptic neurons to transmit the signal back to the pre-synaptic one with the same level of intensity. Feedback alignment (FA) [Lillicrap et al., 2016], or random backpropagation [Baldi et al., 2018], relaxes this restriction by using fixed random weights instead of feed-forward weights during the backward pass. FA (Figure 2.4 (b)) reduces the need for exact alignment between forward and backward weights and offers a more bio-plausible training approach. However, the learning signal is not local for both BP and FA, as it has to be relayed backward from the output to the hidden ones, layer by layer. This contrasts with local learning mechanisms observed in biological neurons, in which synaptic plasticity occurs with directly accessible information through connected neurons. Direct feedback alignment (DFA) [Nøkland, 2016], shown in Figure 2.4 (c), solves this issue by providing a direct feedback path from the output layer to the intermediate hidden ones with fixed random connections. This error-driven learning method disconnects the forward and feedback paths, introducing more flexibility in delivering error signals. Although DFA offers local learning targets, hidden layers still have to wait till the completion of the forward pass before error signals can be transmitted. This drawback, known as update locking, requires BP, FA, and DFA to reserve extra memory to buffer all the activities during the forward pass. More recently, direct random target projection (DRTP) [Frenkel et al., 2021], depicted in Figure 2.4 (d), breaks update locking by projecting the final target with fixed random weights to each hidden layer as local learning goals. Through DRTP, parameters in hidden layers can be adjusted before passing the signal forward to the upper layer.

In short, FA, DFA, and DRTP are more biologically plausible alternatives. They reduce computational complexity and memory usage during the training. FA and DFA have shown better performance than BP in limited scenarios. DRTP might be the best solution for prioritizing computational efficiency and biological plausibility, especially on neuromorphic hardware deployment. However, BP remains favored over the rest for practical applications as it provides more accurate gradient-based updates and better task performance [Chauvin and Rumelhart, 2013].

2.3 Vanilla Recurrent Neural Network

Biological neural circuits contain an abundance of recurrent connections within and across regions [Douglas and Martin, 2007]. This type of connection plays a vital role in working memory [Baddeley, 1992] as it maintains information over short periods by sustaining neural activity. Artificial recurrent neural networks (RNNs), inspired by their biological counterparts, are a special class of neural network that exhibits temporal dynamics [Hopfield, 1982]. They are designed to handle tasks involving sequential data, such as natural language processing [Chowdhary and Chowdhary, 2020] and time-series forecasting [Lim and Zohren, 2021].

The most frequently used RNNs are derived from feed-forward neural networks, in which multiple layers of hidden units are included between input and output layers. Each hidden layer includes recurrent connections where the output of that hidden layer from a previous time step is fed back into the same layer. This feedback connection allows the network to incorporate information from previous time steps into its current state, thus creating a memory [Gallicchio, 2018]. This hidden state can be updated at each time step based on the current input and the previous hidden state.

A simple RNN architecture with one hidden layer is shown in Figure 2.5. In this RNN, hidden layer at time t receives both input at that time step as well as hidden state from the previous time step, and the current state of hidden layer is then computed via nonlinear activation as:

$$\mathbf{h}_t = \sigma(\mathbf{W}_{\text{in}}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b}) \quad (2.3)$$

where $\mathbf{h}_t$ is the current hidden state, $\mathbf{h}_{t-1}$ is the previous hidden state, $\mathbf{x}_t$ is the input at current time, $\mathbf{b}$ is the bias term, $\mathbf{W}_{\text{in}}$ and $\mathbf{W}$ are input and recurrent weight matrices,

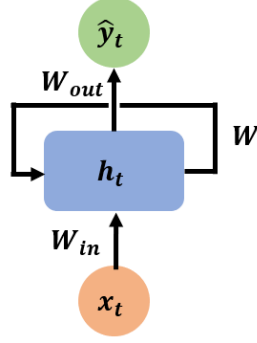


Figure 2.5: Architecture of recurrent neural network with one hidden layer. At time t , input at that time x_t is sent into the hidden layer with feed-forward weight W_{in} . Activities of hidden layer h_t generates the network output at that time as $\hat{y}_t$ using output matrix W_{out} . Different from feed-forward neural network, hidden layer in the RNN has recurrent connection, delineates as weight matrix W . This matrix connects the activity of hidden layer at present time (h_t) to the one at next time step, i.e., h_{t+1} .

respectively.

A linear readout layer is attached on top of hidden layer to generate output of the network:

$$\hat{\mathbf{y}}_t = \mathbf{W}_{out} \mathbf{h}_t \quad (2.4)$$

where $\hat{\mathbf{y}}_t$ is the network output and $\mathbf{W}_{out}$ is the output weight matrix.

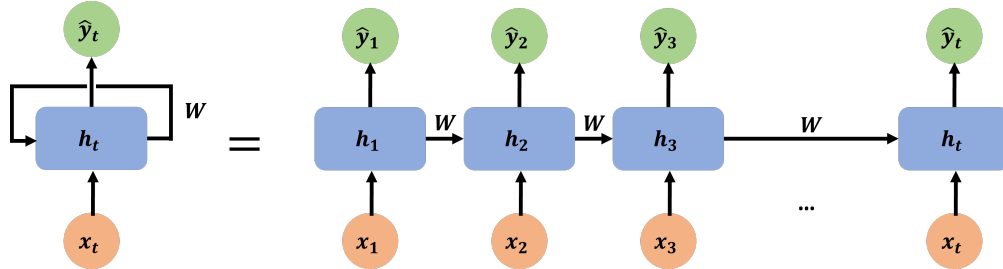


Figure 2.6: Recurrent neural network unrolled in time. When in the unrolled form, each time step of the sequence is depicted as a separate frame, with the recurrent weight W connecting the adjacent frames.

RNNs can be unrolled, in this way the compact representation is transformed into a sequence of frames where each step is explicitly represented at each time point [Sutskever, 2013]. The same recurrent weight connects each frame of the hidden state one by one. This process is

illustrated in Figure 2.6.

The unrolled RNN essentially becomes a feed-forward neural network with the unique property of repeating the same parameters throughout the expanded network. Similar to training feed-forward neural networks in which the chain rule is applied to backpropagate the gradient from output downward through the hidden layers, the gradient in RNNs can be backpropagated through the entire sequence from the final time frame to the initial one [Rumelhart et al., 1986]. This procedure, known as backpropagation through time (BPTT) [Werbos, 1990], is fundamental in training RNN and its variants. During BPTT, gradients at each time step are computed and aggregated before updating. This method becomes problematic when the sequence is long. For instance, the transpose of recurrent weight is multiplied at each time step to compute the gradient [Pascanu et al., 2013b]. A vanishing/exploding gradient occurs when the weight is too small/large [Hochreiter and Schmidhuber, 1997]. This challenge becomes more pronounced when the unrolled network has a long sequence in time [Bengio et al., 1994].

Advanced RNN variants like long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] and gated recurrent unit (GRU) [Cho et al., 2014] are created to alleviate this problem. Specifically, LSTM incorporates gating mechanisms [Yu et al., 2019], which include three gates (input gate, forget gate, and output gate) to regulate the information stored in and retrieved from the memory cell. GRU is a simplified version of LSTM as it combines the input and the forget gate into one and uses a reset gate to control the influence of the previous hidden state. Both networks increase the stability and boost the performance over vanilla RNNs at the expense of computational power [Fu et al., 2016, Shewalkar et al., 2019].

Another issue that hinders the implementation of BPTT, especially on edge devices, is that BPTT achieves parameter updates through an offline scheme. Contrary to how the human brain learns as information arrives, BPTT requires seeing the complete sequence before computing the gradient for the backward pass [Lillicrap and Santoro, 2019]. The

online versions of BPTT [Marschall et al., 2020] and other gradient-based continuous learning rules, such as real-time recurrent learning (RTRL) [Williams and Zipser, 1989], can perform updates in a more incremental manner. However, they are more computationally intensive and memory-demanding, leading to few practical uses.

2.4 Gradient-Free RNNs

Reservoir computing [Lukoševičius and Jaeger, 2009, Schrauwen et al., 2007] is a framework for training recurrent neural networks (RNNs) in a gradient-free fashion. It simplifies the training process by decoupling the training of the recurrent network’s dynamic behavior from the training of the readout layer. This approach leverages the concept of a ”reservoir” - a fixed, randomly connected network that generates rich and diverse dynamical responses to input data.

2.4.1 Echo State Network

Echo state network (ESN) [Jaeger and Haas, 2004] is an early model of reservoir computing in which an input signal is fed into a random dynamical system, and the dynamics of the sparsely interconnected units in the reservoir map the input to a higher dimension. ESN exhibits the echo state property [Yildiz et al., 2012] that the effect of past inputs fades away, and the network’s internal states settle into a regime where they respond consistently to new inputs.

Standard architecture of ESN is shown in Figure 2.7.

The dynamics of units in the reservoir can be described as non-spiking leaky integrator

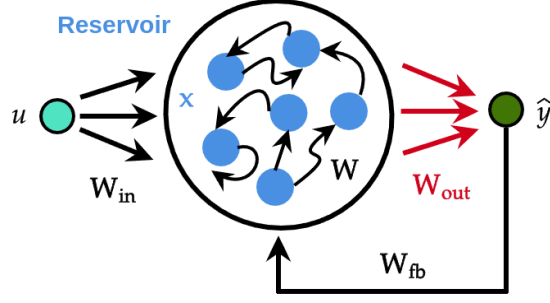


Figure 2.7: Architecture of echo state network (ESN). Here $\mathbf{u}$, $\mathbf{x}$, $\hat{\mathbf{y}}$ represents the input, hidden unit, and output, respectively. Input weight $\mathbf{W}_{\text{in}}$, recurrent weight $\mathbf{W}$, and feedback weight $\mathbf{W}_{\text{fb}}$ are fixed. Only the readout weight $\mathbf{W}_{\text{out}}$ is trainable. This figure is a modification from [Jaeger and Haas, 2004].

neurons using the following continuous time dynamical system equation:

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{W}f(\mathbf{x}) + \mathbf{W}_{\text{in}}\mathbf{u} + \mathbf{W}_{\text{fb}}\hat{\mathbf{y}} \quad (2.5)$$

where

- $\mathbf{u}$ is the input to the network,
- $\mathbf{W}_{\text{in}}$ is the input weight,
- $\mathbf{x}$ is the activities of the units (neurons) in the reservoir,
- $\mathbf{W}$ is the recurrent weight that connects each unit to one another,
- $\hat{\mathbf{y}}$ is the output generated by the network,
- $\mathbf{W}_{\text{fb}}$ is the feedback weight that sends output back into the reservoir,
- τ is the time constant,
- $f()$ is nonlinear activation function,

respectively.

The output of the network is extracted from the reservoir using a linear readout:

$$\hat{\mathbf{y}} = \mathbf{W}_{\text{out}}\mathbf{x} \quad (2.6)$$

in which $\mathbf{W}_{\text{out}}$ is the output weight.

The goal is to learn the readout weight $\mathbf{W}_{\text{out}}$ such that the predicted output $\hat{\mathbf{y}}$ matches the target value $\mathbf{y}$. This is done by formulating a cost function and minimizing its value with respect to the weight. One common cost function is mean squared error (MSE):

$$\text{MSE} = \frac{1}{T} \sum_{t=1}^T \|\hat{\mathbf{y}}(t) - \mathbf{y}(t)\|^2 \quad (2.7)$$

where T is the total number of time steps. Since the recurrent weight $\mathbf{W}$ is fixed, the problem reduces to a linear regression which has analytical solutions.

Liquid state machine (LSM) developed by Maass et al. [2002] is another form of reservoir computing in which the units in the network are spike-based rather than continuous values. Although LSM shares the same architecture as ESN, it is more biologically realistic since it uses the dynamics of spiking neurons to generate complex responses. Both ESN and LSM have fixed weights in the reservoir. Although training of such network is computationally inexpensive, restriction on weight update can limit the model's capacity to capture complex temporal patterns.

2.4.2 FORCE Learning

First-order, reduced and controlled error (FORCE) learning is an improved version of ESN, developed by Sussillo and Abbott [2009]. It harnesses the chaotic activity of recurrent networks to generate a wide range of temporal patterns. More importantly, FORCE allows for different types of architectures in which training is not restricted to the output weight,

thus making it more biologically plausible and powerful than ESN.

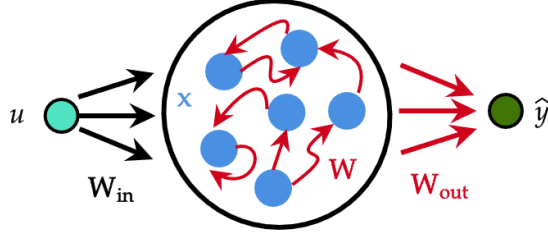


Figure 2.8: Architectures of FORCE learning. Recurrent weight $\mathbf{W}$, in addition to the readout weight $\mathbf{W}_{\text{out}}$, are trainable. This figure is a modification from [Sussillo and Abbott, 2009].

A generic FORCE model is illustrated in Figure 2.8. The dynamics of hidden units in this model is very similar to the aforementioned ESN except no feedback loop is included:

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{W}f(\mathbf{x}) + \mathbf{W}_{\text{in}}\mathbf{u} \quad (2.8)$$

And the output of model is a linear readout as well:

$$\hat{\mathbf{y}} = \mathbf{W}_{\text{out}}\mathbf{x} \quad (2.9)$$

Different from ESN in which network is silent without out input, FORCE uses a network that exhibits chaotic activity prior to training and it is suppressed via strong and rapid weight update in the early stage of training. The recurrent weight $\mathbf{W}$ can be trained by fed back output error (the different between the output $\hat{\mathbf{y}}(\mathbf{t})$ and target $\mathbf{y}(\mathbf{t})$ at each time step), not the output itself, back into the network. In FORCE, this error term is kept small (first-order) to avoid learning disruption. And recursive least square (RLS), an online learning algorithm [Haykin, 2002], is typically used for training.

2.4.3 Full-FORCE

Full-FORCE [DePasquale et al., 2018] introduces additional network during training to provide suitable target dynamics. This second network is able to modify the full connectivity matrix of recurrent weight. Networks trained using this method require fewer neurons and exhibit greater noise robustness compared to the traditional approaches.

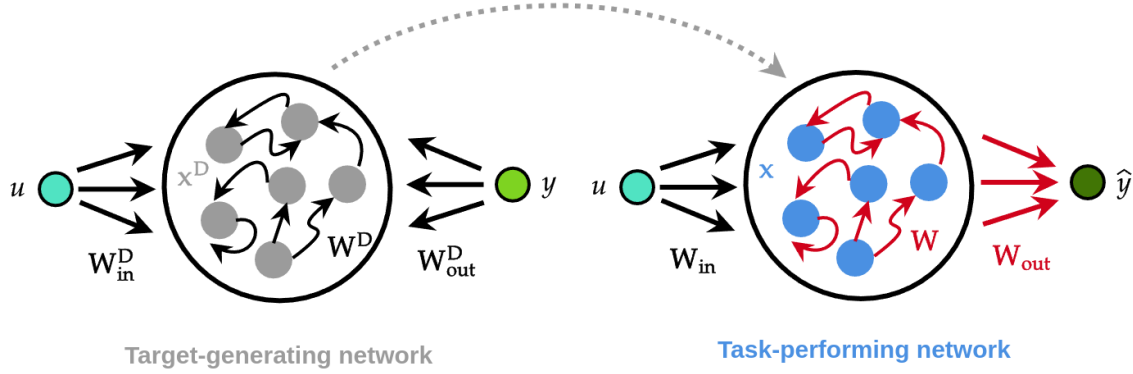


Figure 2.9: Architecture of full-FORCE (left: target-generating network; right: task-performing network). The target-generating network contains no trainable weights as input weight $\mathbf{W}_{in}^D$, recurrent weight $\mathbf{W}^D$, and output weight $\mathbf{W}_{out}^D$ are all fixed. Notice here the target-generating network receives actual target, not network output. Dynamics of this network serve to provide learning targets for the task-performing network, in which recurrent weight $\mathbf{W}$ and output weight $\mathbf{W}_{out}$ are trainable. This figure is a modification from [DePasquale et al., 2018].

As shown in Figure 2.9, full-FORCE has two networks: target-generating network and task performing network.

The dynamics of target-generating network is described as follows:

$$\tau \frac{d\mathbf{x}^D}{dt} = -\mathbf{x}^D + \mathbf{W}^D f(\mathbf{x}^D) + \mathbf{W}_{in}^D \mathbf{u} + \mathbf{W}_{out}^D \mathbf{y} \quad (2.10)$$

- $\mathbf{x}^D$ is the activities of the units in the reservoir,
- $\mathbf{u}$ is the input to the network,

- $\mathbf{y}$ is the target,
- $\mathbf{W}^{\mathbf{D}}$ is the recurrent weight that connects each unit to one another,
- $\mathbf{W}_{\text{in}}^{\mathbf{D}}$ is the input weight,
- $\mathbf{W}_{\text{out}}^{\mathbf{D}}$ is the weight that sends target into the reservoir,
- $f()$ is nonlinear activation function,
- τ is the time constant,

respectively.

Target is fed into the network along with the input and all the weights are fixed after initialization, hence the superscript $\mathbf{D}$. Task-performing network, on the other hand, receives input only and its dynamics is the same as the one in FORCE:

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x} + \mathbf{W}f(\mathbf{x}) + \mathbf{W}_{\text{in}}\mathbf{u} \quad (2.11)$$

Same as ESN and FORCE, task-performing network in full-FORCE generates output using a readout weight $\mathbf{W}_{\text{out}}$.

The underlying assumption of full-FORCE is that a linear readout can extract target output from the mixture of internal and external signals within the activities of the target-generating network once they are transferred to the task-performing network. Therefore, the recurrent weight in the task-performing network are adjusted to minimize the activity difference between the two networks' reservoir. Recurrent weight training can be done using the RLS in the same fashion as FORCE learning. Full-FORCE can be implemented on a spike-based model, allows online training, and provides a local learning target for hidden units. It is considered as the state-of-the-art reservoir computing technique that shares more brain-inspired features than gradient-based RNNs.

Chapter 3

Multi-Layer Full-FORCE with Dynamical Data Coupling

As mentioned in the previous chapter, full-FORCE enables the full-rank update of recurrent weights and provides a larger training capacity than FORCE learning, especially when the reservoir has limited units. Here, we describe an improved model by strengthening the link between the existing networks in full-FORCE and increasing the depth of the hidden layer.

Two different nonlinear dynamical systems: Target-Generating Network (TGN) and Task-Performing Network (TPN), are used during full-FORCE training. The TGN has fixed, recurrent, random weights and receives input and task-relevant targets. Its activities serve as a learning objective for the TPN to update its recurrent weights in real time. In other words, the TPN and the TGN are trained to exhibit similar activities, *i.e.* to synchronize. Full-FORCE learning achieves this synchronization through online weight adaptation using the difference between the two network dynamics as the cost function. However, this synchronization is not guaranteed to occur depending on the choice of hyperparameters (*e.g.* learning rates), which may lead to slow convergence. One way to improve this synchro-

nization is to use a technique inspired by data assimilation, namely Dynamical State and Parameter Estimation (DSPE) [Abarbanel et al., 2009]. DSPE introduces a coupling akin to a control term between the experimental data and the mathematical model to enhance synchronization, eventually improving the optimization of the model states and parameters. Mathematically, DSPE removes positive conditional Lyapunov exponents responsible for instabilities during data assimilation [Abarbanel et al., 2009]. This advantage comes at the cost of requiring experimental data for every coupled variable. Interestingly, the TGN in full-FORCE generates a target for each hidden unit, fulfilling the key requirement for DSPE. We explore the idea of incorporating this coupling into full-FORCE dynamics to facilitate learning in conjunction with RLS and show that it offers stabilization.

Modern RNN architectures also feature multi-level connections [Pascanu et al., 2013a] and use BPTT to compute gradients across multiple layers and time steps. The added depth of the network enables a higher level of abstraction for the input and is essential to ensure success in modeling real-world sequential data [Graves et al., 2013, Hermans and Schrauwen, 2013]. Lately, hierarchical reservoir computing frameworks such as Deep-ESN [Gallicchio and Micheli, 2017, Ma et al., 2017] have also shown promising results without training any hidden weights. In this vein, we extend the single reservoir in full-FORCE to multiple layers while still permitting weight to be updated in the hidden layers. Unlike BPTT, which is offline and non-local, each layer in the TPN of the proposed model receives a layer-specific target from the TGN at every time step. The flexibility of this online training paradigm is appealing for implementation on edge devices for mobile, low-power learning applications.

3.1 Architecture and Learning Rule

The proposed multi-layer full-FORCE with dynamical data coupling has the architecture shown in Figure 3.1. As in the original full-FORCE, we use a pre-initialized TGN (in

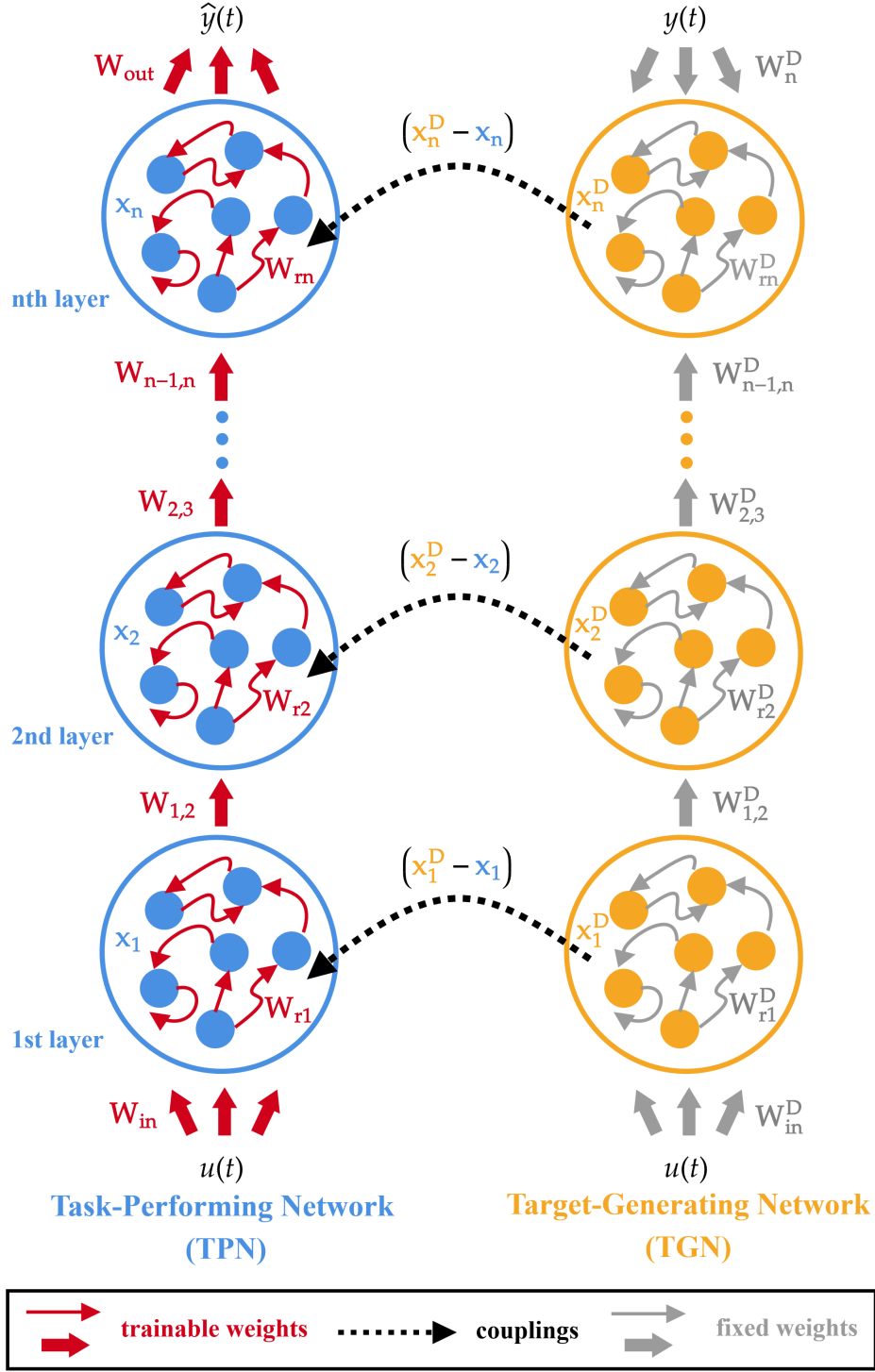


Figure 3.1: The architecture of a multi-layer full-FORCE with dynamical data coupling (left: Task-Performing Network (TPN); right: Target-Generating Network (TGN)). In the TGN all the connections including input, feed-forward, and recurrent weights are fixed (in gray), while the TPN consists of trainable weights (in red). The couplings between each layer of the TGN and the TPN are indicated by the dashed black arrows.

orange) to generate training targets for the TPN (in blue).

We extend the TGN to multiple layers and the activities of units in the i th layer (denoted as $\mathbf{x}_i^{\mathbf{D}}$) as follows:

$$\tau \frac{d\mathbf{x}_i^{\mathbf{D}}}{dt} = -\mathbf{x}_i^{\mathbf{D}} + \mathbf{W}_{\mathbf{ri}}^{\mathbf{D}} f(\mathbf{x}_i^{\mathbf{D}}) + \mathbf{W}_{i-1,i}^{\mathbf{D}} f(\mathbf{x}_{i-1}^{\mathbf{D}}) \quad (3.1)$$

where τ is the time constant, $f()$ is the nonlinear activation function (*tanh* unless stated otherwise), and $\mathbf{x}_{i-1}^{\mathbf{D}}$ is the activities from previous layer. $\mathbf{W}_{\mathbf{ri}}^{\mathbf{D}}$ and $\mathbf{W}_{i-1,i}^{\mathbf{D}}$ are the recurrent weight matrix and feed-forward weight matrix connecting the adjacent layers, respectively. The superscript $\mathbf{D}$ is used here in the TGN as the network “drives” the learning; all the weights in the TGN are fixed and initialized in a similar manner as the original full-FORCE. For the 1st layer, $\mathbf{W}_{i-1,i}^{\mathbf{D}} f(\mathbf{x}_{i-1}^{\mathbf{D}})$ is replaced by $\mathbf{W}_{\mathbf{in}} u(t)$ as it receives the input. For the last (n th) layer, additional term $\mathbf{W}_{\mathbf{n}}^{\mathbf{D}} y(t)$ is included in the TGN to send target into the network.

The TPN in our network also has a multi-layer architecture and activities of units in the i th layer (denoted as $\mathbf{x}_i$) have the following dynamics:

$$\tau \frac{d\mathbf{x}_i}{dt} = -\mathbf{x}_i + \mathbf{W}_{\mathbf{ri}} f(\mathbf{x}_i) + \mathbf{W}_{i-1,i} f(\mathbf{x}_{i-1}). \quad (3.2)$$

The network output $\hat{y}(t)$ is extracted from the final (n th) layer of the TPN via readout weight matrix $\mathbf{W}_{\mathbf{out}}$, *i.e.*

$$\hat{y}(t) = \mathbf{W}_{\mathbf{out}} f(\mathbf{x}_{\mathbf{n}}). \quad (3.3)$$

To adjust $\mathbf{W}_{\mathbf{out}}$, we use the RLS method with error: $\hat{y}(t) - y(t)$, the same way as described in Sussillo and Abbott [2009], DePasquale et al. [2018].

The TGN is used only during the training to facilitate the learning. It receives both input $u(t)$ and task-relevant target $y(t)$. To ensure the TPN dynamics contains information related

to the desired target, the dynamics of the TPN and the TGN can be matched to each other DePasquale et al. [2018]. Mathematically, this can be obtained by minimizing the difference between (3.1) and (3.2) through adjusting $\mathbf{W}_{\mathbf{r}\mathbf{i}}$ and $\mathbf{W}_{\mathbf{i}-1,\mathbf{i}}$. To achieve this, we use RLS with the following error:

$$[\mathbf{W}_{\mathbf{r}\mathbf{i}}^{\mathbf{D}}f(\mathbf{x}_{\mathbf{i}}^{\mathbf{D}}) + \mathbf{W}_{\mathbf{i}-1,\mathbf{i}}^{\mathbf{D}}f(\mathbf{x}_{\mathbf{i}-1}^{\mathbf{D}})] - [\mathbf{W}_{\mathbf{r}\mathbf{i}}f(\mathbf{x}_{\mathbf{i}}) + \mathbf{W}_{\mathbf{i}-1,\mathbf{i}}f(\mathbf{x}_{\mathbf{i}-1})]. \quad (3.4)$$

Coupling the target with the model dynamics, specifically Dynamical State and Parameter Estimation (DSPE), is a technique that is developed for data assimilation in dynamical systems Abarbanel et al. [2009]. DSPE introduces a coupling term to synchronize the data and the model during the state and parameter estimation, reducing the conditional Lyapunov exponents of the dynamics which then stabilizes data assimilation.

Here, we want to synchronize the activity between the two networks, *i.e.* minimize $(\mathbf{x}_{\mathbf{i}}^{\mathbf{D}} - \mathbf{x}_{\mathbf{i}})$, which has the same effect as reducing error in (3.4). Following the central idea of DSPE, we update the dynamics of the TPN in (3.2) with couplings (shown as the dashed black arrows in Figure 3.1):

$$\tau \frac{d\mathbf{x}_{\mathbf{i}}}{dt} = -\mathbf{x}_{\mathbf{i}} + \mathbf{W}_{\mathbf{r}\mathbf{i}}f(\mathbf{x}_{\mathbf{i}}) + \mathbf{W}_{\mathbf{i}-1,\mathbf{i}}f(\mathbf{x}_{\mathbf{i}-1}) + k_i(t) (\mathbf{x}_{\mathbf{i}}^{\mathbf{D}} - \mathbf{x}_{\mathbf{i}}) \quad (3.5)$$

where $k_i(t)$ is the layer-specific time-varying coupling coefficient that regulates the synchronization manifold. Since the goal is to train the TPN so that it can generate correct output in the *absence* of targets, we need to choose a $k_i(t)$ that vanishes at the end of training [Abarbanel et al., 2009]. For simplicity, here we use

$$k_i(t) = g_i(\hat{y}(t) - y(t))^2 \quad (3.6)$$

where g_i is a gain factor and $(\hat{y}(t) - y(t))^2$ is the squared error between the TPN output and

desired target. As training proceeds, $k_i(t)$ tends to zero. The gain is selected manually by examining the initial magnitude of the cost function and kept fixed afterwards or it can be treated as an additional trainable parameter during the optimization as in DSPE. We use the coupling term to facilitate the learning process and it is removed after training when the TGN is no longer used.

The time evolution of the TGN and the TPN formulated in (3.1) and (3.5) are discretized using the forward Euler method. For computational efficiency, during training, activities in the TGN are computed for each input-target pair before being used as training targets for the TPN. When training concludes, the TPN with optimized weights is the sole product to test on different time-series tasks.

3.2 Effect of dynamical data coupling

We investigate the effect of incorporating dynamical data coupling into the TPN through two different tasks.

3.2.1 Oscillation Task: Description

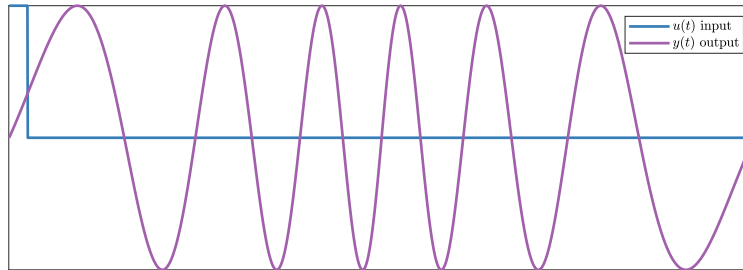


Figure 3.2: Oscillation task. The input $u(t)$ for this task is a brief pulse and the the network learns to generate a frequency-modulated oscillation: $y(t) = \sin(\omega(t)t)$ with $\omega(t)$ increasing linearly from 2π to 6π for the first half of the oscillation and doing the opposite for the second half.

RNNs are well-suited to generate desired sequential patterns when the weights in the network are correctly adjusted during the training. We test our model on the synthesized oscillation task described in DePasquale et al. [2018]. The goal is to train the network to automatically generate a one-dimensional frequency-varying sinusoidal signal (also known as chirp signal) when a brief pulse is presented as the input, as shown in Figure 3.2. A single layer full-FORCE has been shown to successfully produce this pattern with fewer units (200) than what FORCE learning needs (400). For this task, Mean Squared Error (MSE) between the desired target $y(t)$ and the network output $\hat{y}(t)$:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y(i) - \hat{y}(i))^2 \quad (3.7)$$

where n is the the total number of points in a given sequence of pattern is computed during training and testing stages to measure network’s performance.

3.2.2 Generating Oscillation: Results

We present the effect of including the coupling into the dynamics of the TPN on the task of generating one-dimensional oscillation. The results are shown in Table 3.1 and Fig. 3.3 (over 50 initializations for network).

Table 3.1: Summary of testing results for oscillation task

# of units	# of updates per training	Use of coupling	MSE of testing $\mu \pm \sigma$
200	500	No	0.22±0.29
		Yes	0.06±0.13
	1000	No	0.15±0.29
		Yes	0.03±0.13
400	500	No	8e-4±9e-4
		Yes	6e-4±9e-4
	1000	No	4e-4±1e-3
		Yes	3e-4±4e-3

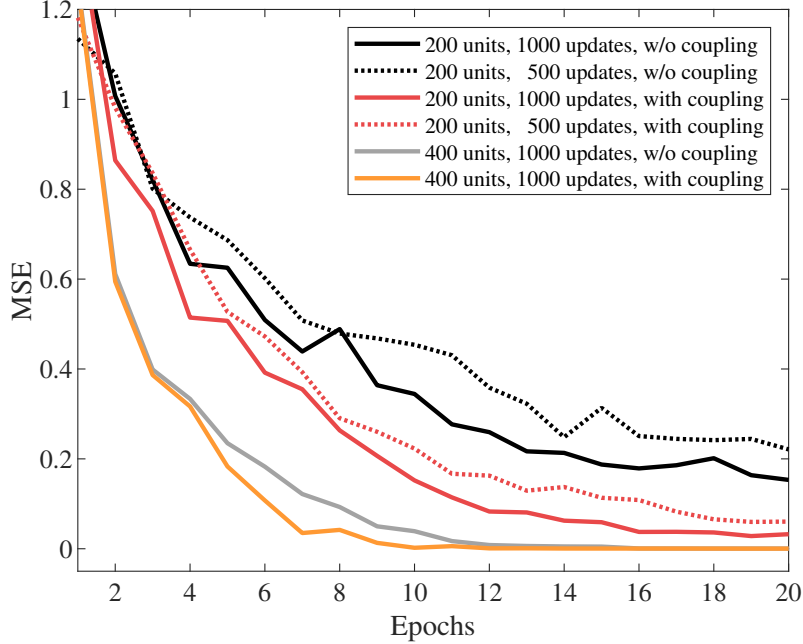


Figure 3.3: MSE at the end of each iteration for the oscillation task. We use full-FORCE network of two different sizes (200 & 400 units). Both are trained with two weight update frequencies (500 & 1000 per epoch). Network with dynamical coupling between the TGN and the TPN achieve lower errors. The MSE values depicted in the figure are the averaged results over 50 different network initializations.

Training output weights and recurrent weights connecting 200 units in one reservoir with 1000 RLS updates per iteration (solid black line) results in 0.15 for MSE on testing at the end of 20 epochs. When incorporating the coupling term during the training, full-FORCE with the same setup shows consistently better learning outcome (solid red line) on the same task, with an ultimate 80% decrease in error (0.03). Reducing the weight update frequency to half (500) causes higher errors in both cases (black and red dotted lines). However, when including the dynamical coupling, full-FORCE with 500 updates per training iteration still exceeds that with 1000 updates considerably. Furthermore, full-FORCE with dynamical coupling (solid and dotted red lines) only need 10 iterations of training to reach comparably the same or even better MSE than those without the coupling (solid and dotted black lines) but with 20 iterations of training. These results demonstrate that for a small network, dynamical data coupling has the same effect as learning online and can compensate for the decreased number of weight updates while improving the training outcome in the meantime.

Doubling the number of units to 400 (solid gray line) leads to a dramatically lower error on testing for a full-FORCE network, leaving little to no room for improvement. Nevertheless, adding the coupling (solid orange line) produces an MSE curve that converges slightly faster than the full-FORCE. In all cases shown in Table 3.1, training full-FORCE with coupling between the TPN and the TGN not only gives significantly lower errors it also provides a more stable learning process, manifested by minor standard deviations over all of the cases.

Eigenvalues of learned networks listed in 3.1 along with activities of sample hidden units are presented in Appendix A.

3.2.3 Interval Timing Recognition Task: Description

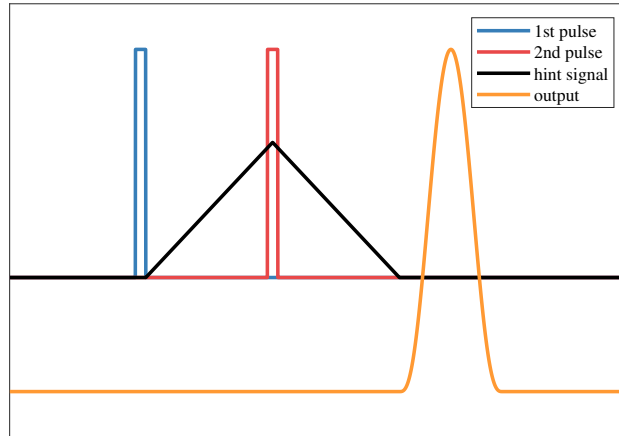


Figure 3.4: Example of interval timing recognition task. The network is presented with two pulses and the delay between them is determined from an uniform distribution (min: 0.1 s, max: 1.0 s). The goal is for the network to generate a bump after the second pulse with the same delay. A hint signal (a function that ramps linearly in time between the pulses and then decreases linearly at the same rate following the second pulse) can be used to assist the learning.

Neuromorphic devices often receive spikes or pulses as the input. Thus potential neuromorphic computing algorithms must be capable of processing information from pulse trains. Here we adopt the interval matching task from DePasquale et al. [2018] to test our model’s ability to learn the time interval between the adjacent pulses. For each trial of this task,

the network first receives one pulse (duration: 50 ms), and its onset time is randomly drawn from an exponential distribution (mean: 1.0 s). A second pulse with the same duration is then sent after a delay. The time interval between the two pulses are determined from a uniform distribution (min: 0.1 s, max: 1.0 s) for each trial. The goal is for the network to track the time interval and always generate a bump function after the second pulse with the same amount of delay shown between the two input pulses. One example of such task is presented in Figure 3.4. We train the network with 40 different trials and test it on 10 cases with delays sampled evenly from 0.1 to 1.0 s. The normalized error between the output delay from the network and the target delay is measured to determine how accurate the RNN model can recognize the interval timing.

3.2.4 Matching Interval Time: Results

The dynamical data coupling also helps solve the interval matching task for full-FORCE. As reported by DePasquale et al. [2018], a similar task demands a network with 1000 units and an additional signal sent into the TGN as the hint during the training is necessary. Here we show that the coupling will improve the accuracy of recognizing the delays for a smaller hint network with 300 units. After training on 40 samples, the original full-FORCE produces a mean error of 58% over 10 test cases (upper left panel in Figure 3.5). The error is reduced to 52% with the same network that has coupling included (upper right panel in Figure 3.5). One way to boost network’s learning on this task is to repeat the training process multiple times. After 5 training iterations, full-FORCE produces a lower error (45%) over the same testing cases (lower left panel in Figure 3.5). At the same time, the coupled full-FORCE performs much better on the task with a minor error (23%), as shown in the lower right panel in Figure 3.5.

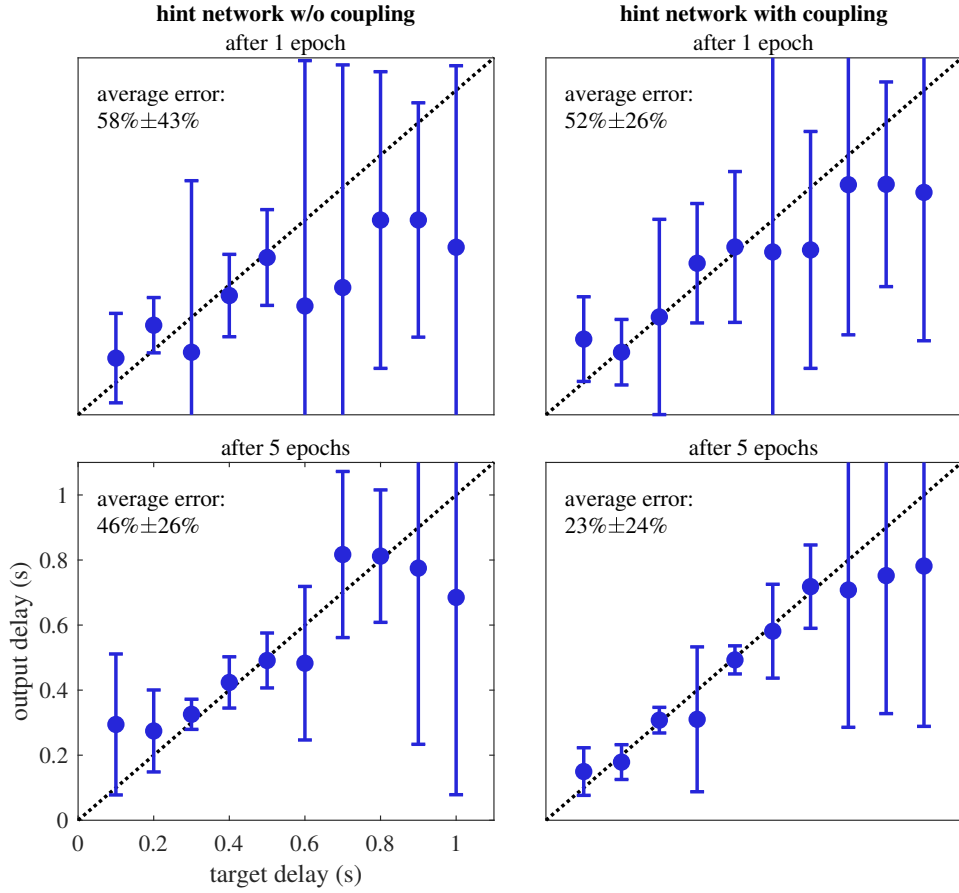


Figure 3.5: Performance on interval matching task using the full-FORCE network with 300 units. The dynamical coupling is shown to help full-FORCE generate the delayed signal closer to the target time after both 1 and 5 iterations of training. We test full-FORCE with 10 different initialized weights (indicated by the vertical error bars in all four panels) on 10 cases with delays ranging from 0.1 to 1.0 seconds evenly.

3.3 Effect of multi-layer architecture

We then study how increasing the depth of full-FORCE can affect network’s capacity to learn high-dimensional real-world patterns.

3.3.1 Multi-dimensional Body Joint Movements: Dataset

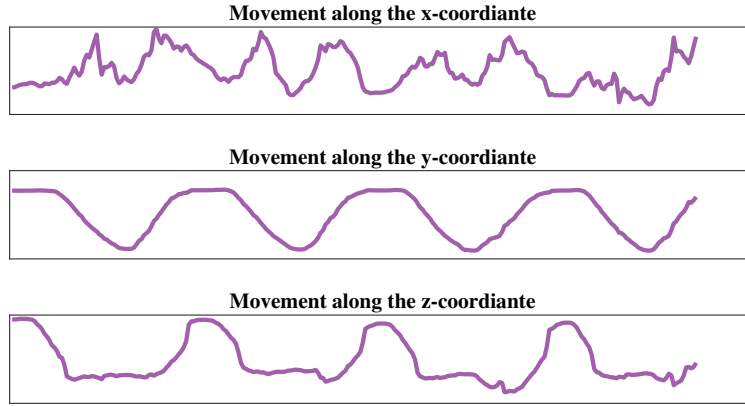


Figure 3.6: Example of joint movement. Here the 3 coordinates (x , y , and z) of one joint movement is plotted.

For this task, we use the University of Idaho - Physical Rehabilitation Movements Data Set (UI-PRMD) collected by Vakanski et al. [2018]. This dataset comprises positions and angles of 22 body joints during ten different rehabilitation movements. The network is trained to recognize different inputs (10-dimensional one-hot representation) and generate the corresponding high-dimensional ($22 \text{ joints} \times 3 \text{ coordinates}$) cyclical joint movements. Figure 3.6 displays one joint movement traces. For this task, Mean Squared Error (MSE) between the desired target $\mathbf{y}(t)$ and the network output $\hat{\mathbf{y}}(t)$:

$$\text{MSE} = \frac{1}{m} \frac{1}{n} \sum_{i=1}^m \sum_{j=1}^n (\mathbf{y}_i(j) - \hat{\mathbf{y}}_i(j))^2 \quad (3.8)$$

where m is the total number of joints and n is the the total number of points in the each movement trace is computed during the training and the testing stages.

3.3.2 Generating Body Joint Movements: Results

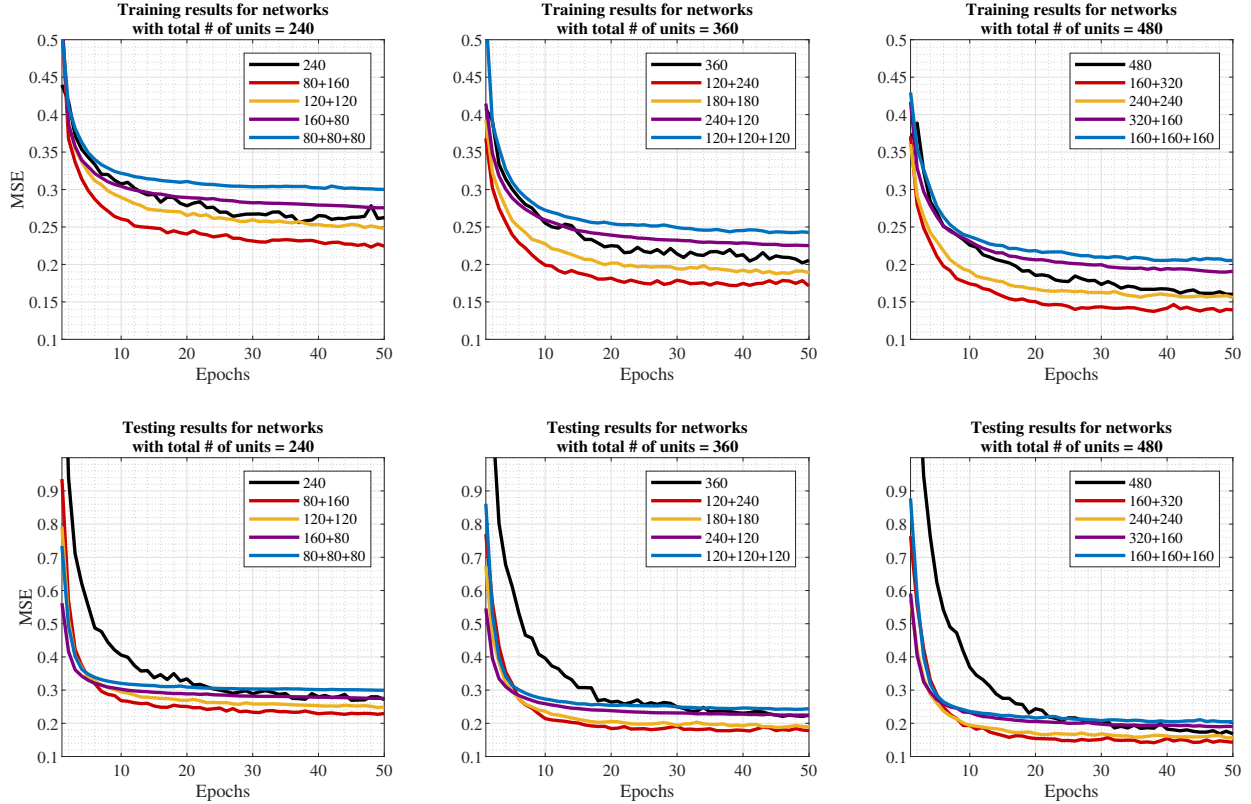


Figure 3.7: Results for the task of generating body joint movements. The top panel shows the MSE during the training stage while the bottom three show the MSE for the testing at the end of each training epoch. Three different network sizes with total number of units: 240 (left 2 plots), 360 (middle 2 plots), and 480 (right 2 plots) are used. For each network size, we distribute the units evenly into one, two and three layers. For the case of two layer networks, we explore the uneven distributions of units between the two layers as well. The numbers in each plot’s legend indicate the amount of units in each layer, with early layer comes first. Each curve is the average of 10 runs.

We show the impact of converting the original single-layer full-FORCE into multi-layer ones on generating patterns resembling joint movements with different input signals, a more difficult task in which both input and desired target are multi-dimensional. Figure 3.7 plots all the training and testing results. Table 3.2 (averaged results on producing all 10 joint patterns over 10 different network initializations) shows that we construct RNNs with three total numbers of units (240, 360, & 480). For each case, we distribute the units evenly into one, two, and three layers. In one-layer full-FORCE, recurrent weights are trained in

addition to the output weights. In our proposed two-layer and three-layer RNN models, we also update the feed-forward weights connecting adjacent layers as described previously. For all models, the input weights remain fixed after initialization to illustrate how the number of layers and trainable hidden (recurrent and feed-forward) weights can influence the learning results on this task.

Table 3.2: Summary of multi-dimensional pattern generation experiment

Total # of units	Total # of layers	# of units in each layer	# of trainable hidden weights	MSE of testing $\mu \pm \sigma$
240	1	240	57600	$0.27 \pm 2.6\text{e-}2$
	2	80+160	44800	$0.23 \pm 1.1\text{e-}2$
		120+120	43200	$0.25 \pm 5.8\text{e-}3$
		160+80	44800	$0.28 \pm 6.4\text{e-}3$
	3	80+80+80	32000	$0.30 \pm 5.1\text{e-}3$
360	1	360	129600	$0.22 \pm 1.8\text{e-}2$
	2	120+240	100800	$0.18 \pm 1.5\text{e-}2$
		180+180	97200	$0.19 \pm 1\text{e-}2$
		240+120	100800	$0.22 \pm 7.4\text{e-}3$
	3	120+120+120	72000	$0.24 \pm 9\text{e-}3$
480	1	480	230400	$0.17 \pm 1.3\text{e-}2$
	2	160+320	179200	$0.15 \pm 1.1\text{e-}2$
		240+240	172800	$0.16 \pm 5.8\text{e-}3$
		320+160	179200	$0.19 \pm 4.6\text{e-}3$
	3	160+160+160	128000	$0.21 \pm 8.7\text{e-}3$

We show that two-layer full-FORCE networks can generate different joint movements with lower errors than their one-layer counterparts. For instance, splitting one-layer of 240 units into two layers of 120 units reduces the MSE from 0.27 to 0.25. Increasing the number of units for one-layer full-FORCE naturally leads to better results on generating desired patterns due to the greater learning capacity, indicated by lower MSEs for the network with 360 (0.22) and 480 units (0.17). Still, the two-layer networks each with 180 units and 240 units have lower MSEs (0.19 and 0.16, respectively). This is of great significance as dividing one layer into two and connecting them with feed-forward weights reduces the number of trainable hidden weights (a one-layer full-FORCE of $2N$ units has $4N^2$ recurrent weights,

a two-layer network each with N units has $3N^2$ hidden weights in total: $2N^2$ recurrent weights and N^2 feed-forward weights) and cuts down the number of flops required for each RLS computation. Both factors can ultimately contribute to an accelerated weight update process.

When trying an unbalanced distribution of units in the two-layer networks, we observe interesting results. Networks favoring more units for the 2nd layer than the 1st one (*i.e.* 80+160, 120+240, and 160+320) are the best. One explanation is this joint movement pattern generation task, in particular, has higher dimensions for output than input, therefore more units are entailed for the layer closer to the output. Although the three-layer RNNs have MSE values with lower standard deviations for the testing, they show the highest errors. In this case, a significantly lowered number of hidden weights available for training may have limited the network’s overall performance, or the last layer with fewer units does not have enough capacity to generate the high dimensional patterns.

The added layers in the architecture provide more flexibility in changing the overall dynamics of full-FORCE, but it also introduces more complexity when configuring the network. For example, the intermediate layers ($\mathbf{x}_i$) in the TPN architecture we illustrate in Fig. 3.1 may not receive proper target-relevant information to train from the corresponding TGN layers ($\mathbf{x}_i^D$), as only the first ($\mathbf{x}_1^D$) and the last ($\mathbf{x}_n^D$) layer in the TGN receives task input ($u(t)$) or target ($y(t)$) directly. This could also have caused the higher errors for the three-layer networks on pattern generation task. One can choose to feed both input and target into each layer of the TGN, which is the same as stacking multiple full-FORCE networks. However, this can potentially overwhelm the learning for the TPN and cause instability. One solution is to adjust the network’s hyperparameters (such as the initialization of the RLS algorithm) so the learning starts properly. Other than that, the feed-forward connections ($\mathbf{W}_{i-1,i}^D$) between adjacent layers of the TGN can be removed to reduce the complexity of the network dynamics. We experiment with all these factors and report the best results in Table

3.2. A detailed or theoretical investigation on this topic is still absent to explain further the effect of an increased number of layers on the learning dynamics of full-FORCE. Appendix B provides a quick look at how units in the first hidden layer of the best performing network responds to different inputs.

3.4 Improvement on processing EEG signals

Finally, we put both dynamical data coupling and multi-layer architecture into use by improving the existing full-FORCE to predict eye state using EEG signals.

3.4.1 EEG Dataset

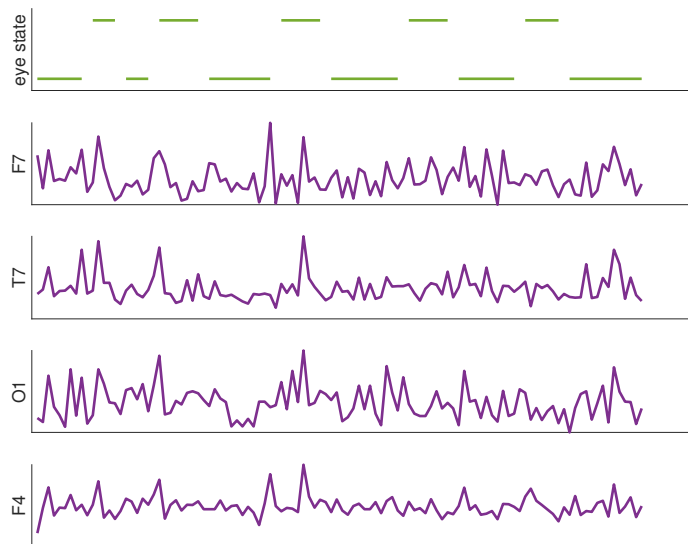


Figure 3.8: EEG signals and eye state. EEG signals from four selected channels are plotted along with corresponding eye state.

One practical application of RNNs in the biomedical field is processing spatial temporal electroencephalogram (EEG) signals on brain interfacing devices. We therefore train our model on EEG recordings from Emotiv EPOC headsets to classify eye states (open and closed). Figure 3.8 shows sample recordings out of selected channels and the corresponding

eye state (0:eyes closed; 1: eyes open). We curate a dataset constituting 180 training and 43 testing samples by extracting half-second continuous segments ($14 \text{ channels} \times 64 \text{ data points}$ each) from the original measurement (117 seconds in duration with a sampling frequency of 128 Hz) [Rösler and Suendermann, 2013]. The network receives a randomized sequence of training samples during each epoch to learn the mapping between EEG signal and eye state. It generates two outputs (one-hot encoding for each state) at every step, and we evaluate the network’s prediction at the end of each EEG segment. The accuracy of classification (number of correct classifications over the total number of test samples) on eye state is calculated for comparison.

3.4.2 Classification of Eye State: Results

All the RNNs are trained on 180 half-second measurements and tested on 43 samples. Using the original full-FORCE of 400 units (dotted black line in Fig. 3.9) results in an averaged classification accuracy of 0.772 ± 0.052 through 20 epochs. Our new models, two-layer full-FORCE with coupling, can predict eye state with noticeably higher accuracies. The proposed network that has an unbalanced distribution of units between the two layers with 300 units in the lower layer and 100 in the second one (solid blue line) ultimately labels the eye state better (0.886 ± 0.030) than the other two networks (red and orange solid lines, 0.847 ± 0.029 and 0.802 ± 0.052 , respectively). For this EEG-based classification task, the input dimension (14 channels) is higher than that of the target (2 states), which explains why more units are required for the first layer to handle the input.

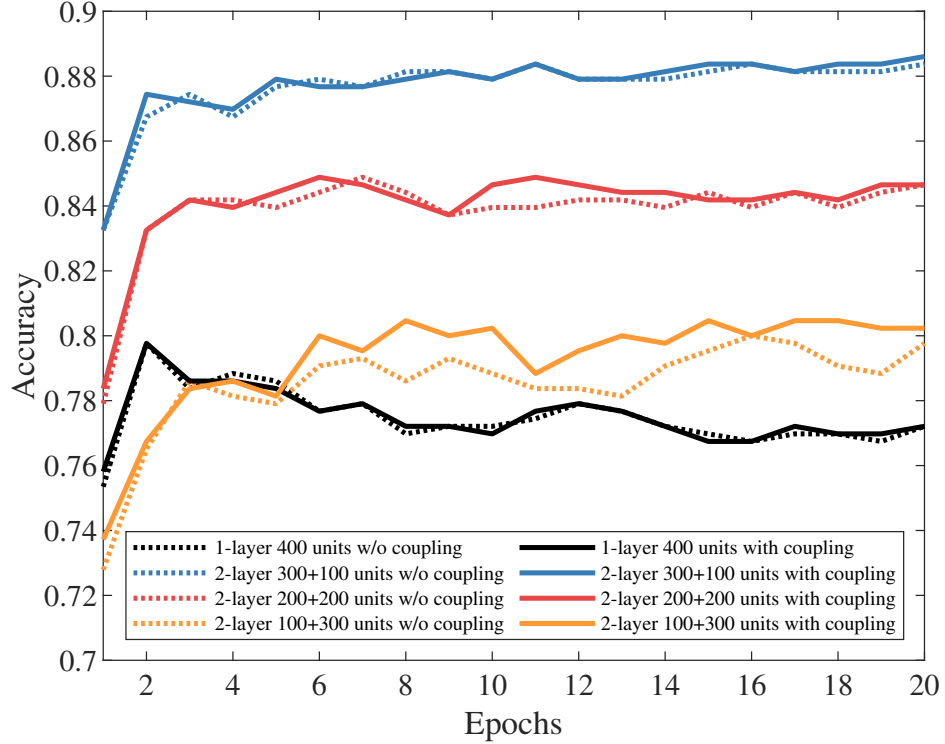


Figure 3.9: Accuracy (averaged over 10 RNN initializations) for the eye state classification measured at the end of each training epoch on all testing samples. Outcomes from our networks are plotted along with the one produced by full-FORCE (one-layer 400 units without coupling). Both coupling and an increased number of layers in network architecture improve the classification accuracy on this task: coupling shows marginal gains while the two-layer networks offer compelling improvement.

Chapter 4

Deep Continuous Local Learning with Recurrent Connections

4.1 Recurrent Continuous Local Learning

In this section we introduce a novel recurrent network that not only includes neural dynamics but also exploits the gradient descent optimization technique. This network follows the same learning rule as one described in deep continuous local learning (DECOLLE) [Kaiser et al., 2020] and showcases recurrent connections.

4.1.1 Learning Rule Derivation

We first introduce the recurrent continuous local learning network with one hidden layer and show its performance on multiple tasks. The architecture of the network is shown in 4.1: input, recurrent, and output weights are all adjustable during the training. Each neuron (unit) in the network follows the leaky integrate-and-fire (LIF) model which can be described

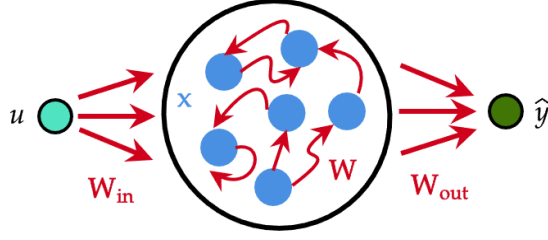


Figure 4.1: Architecture of recurrent continuous local learning. All the weights (input, recurrent, and output) are trainable in this RNN model.

using multiple continuous time dynamical systems.

First,

$$U_i(t) = V_i(t) - \rho R_i(t) + b_i \quad (4.1)$$

Here U and V are both membrane potentials and can be seen as a two-compartmental model where U and V represent membrane potentials of the somatic and the dendritic compartments, respectively [Gerstner et al., 2014]. R captures the dynamics of refractory mechanism, ρ is the constant weight, and b is the bias term.

The dynamics of V can be delineated as follows:

$$\tau_{mem} \frac{d}{dt} V_i(t) = -V_i(t) + I_i(t) \quad (4.2)$$

where τ_{mem} is the time constant of the membrane and I is the total synaptic current received by the neuron.

The dynamics of R is governed by the following differential equation:

$$\tau_{ref} \frac{d}{dt} R_i(t) = -R_i(t) + S_i(t) \quad (4.3)$$

where τ_{ref} is the time constant of the refractory and S is the threshold of firing.

A spike is emitted when the membrane potential surpasses the threshold:

$$S_i(t) = \Theta(U_i(t)) \quad (4.4)$$

where Θ is the unit step function.

The synaptic current dynamics for the neuron is the following:

$$\tau_{syn} \frac{d}{dt} I_i(t) = -I_i(t) + \sum_j W_{ij} S_j(t) \quad (4.5)$$

where τ_{syn} is the time constant of the synapse and W_{ij} is the connection weight.

The dynamics of V can be rewritten as the following since V_i and I_i are linear with respect to W_{ij} :

$$V_i(t) = \sum_j W_{ij} P_{ij}(t) \quad (4.6)$$

where P describes the traces of the membrane-based synapse.

P has the following dynamics:

$$\tau_{mem} \frac{d}{dt} P_{ij}(t) = -P_{ij}(t) + Q_{ij}(t) \quad (4.7)$$

where Q represent the traces of current-based synapse.

Q has the following dynamics:

$$\tau_{syn} \frac{d}{dt} Q_{ij}(t) = -Q_{ij}(t) + S_j(t) \quad (4.8)$$

The discrete form of the network dynamics for computer simulation is shown as follows:

$$\begin{aligned}
U_i[t] &= \sum_j W_{ij} P_{rj}[t] - \rho R_i[t] + b_i \\
S_i[t] &= \Theta(U_i[t]) \\
P_j[t + \Delta t] &= \alpha P_j[t] + (1 - \alpha) Q_j[t] \\
Q_j[t + \Delta t] &= \beta Q_j[t] + (1 - \beta) S_j[t] \\
R_i[t + \Delta t] &= \gamma R_i[t] + (1 - \gamma) S_i[t]
\end{aligned} \tag{4.9}$$

where $\alpha = \exp(-\frac{\Delta t}{\tau_{\text{mem}}})$, $\beta = \exp(-\frac{\Delta t}{\tau_{\text{syn}}})$, and $\gamma = \exp(-\frac{\Delta t}{\tau_{\text{ref}}})$ are the time constants for the membrane potential U , the synaptic state Q , and the refractory (resetting) state R , respectively. And this is equivalent to discrete-time version of Spike Response Model with linear filter [Gerstner et al., 2014].

Gradient descent technique is used in continuous local learning network and the gradient of loss function $\mathcal{L}$ with respect to weight can be computed using the chain rule:

$$\Delta W_{ij} = -\eta \frac{\partial L}{\partial W_{ij}} = -\eta \frac{\partial L}{\partial S_i} \frac{\partial S_i}{\partial W_{ij}} = -\eta \frac{\partial L}{\partial S_i} \frac{\partial \Theta(U_i)}{\partial U_i} \frac{\partial U_i}{\partial W_{ij}} \tag{4.10}$$

The non-differentiability of step function Θ poses a challenge in calculating the partial derivative with respect to membrane potential U .

Surrogate gradient [Nefci et al., 2019] is used in this case to overcome the problem. Essentially, a smooth sigmoid function σ which has a similar shape as step function Θ is adopted when computing the gradient. This technique approximates the true gradient of original function and enables weight update towards the correct direction at the same time [Zenke et al., 2017].

With the surrogate gradient:

$$\frac{\partial \Theta(U_i)}{\partial U_i} = \sigma'(U_i) \quad (4.11)$$

The gradient of weight W becomes:

$$\Delta W_{ij} = -\eta \frac{\partial L}{\partial S_i} \sigma'(U_i) \frac{\partial U_i}{\partial W_{ij}} \quad (4.12)$$

The last term can be computed according to the discrete dynamics of membrane potential:

$$\frac{\partial U_i}{\partial W_{ij}} = P_j - \rho \frac{\partial R_i}{\partial W_{ij}} \quad (4.13)$$

We drop the $\frac{\partial R_i}{\partial W_{ij}}$ as it requires the spike history of the neuron. Regularization can be added to enforce a low firing rate of neurons.

Therefore, the final synaptic weight update becomes the following:

$$\Delta W_{ij} = -\eta \frac{\partial L}{\partial S_i} \sigma'(U_i) P_j \quad (4.14)$$

This weight update follows the three-factor synaptic plasticity rules [Guerguiev et al., 2017, Neftci et al., 2017] in the brain, where neurons receives only local pre- (P_j) an post- ($\sigma'(U_i)$) synaptic spike-based activity along with a global modulation ($\frac{\partial L}{\partial S_i}$) [Urbanczik and Senn, 2014]. The recurrent weight update rule can also be derived in the same fashion described above.

Common auto-differentiation software library such as PyTorch [Paszke et al., 2019] can be used to compute the gradient and update the weight at each step.

4.1.2 Recurrent Continuous Local Learning vs. Full-FORCE

We implement recurrent continuous local learning on two pattern generations tasks previously described in Chapter 3 and compare its performance with dynamically coupled full-FORCE. First, recurrent continuous local learning and dynamically coupled full-FORCE with 100 units are trained to generate time-varying oscillation. As shown in Figure 4.2, full-FORCE with dynamical coupling converges faster than its counterpart. However, gradient-based recurrent continuous local learning reaches lower MSE at the end of training. The low MSE is also reflected on the generated oscillation plotted in Figure 4.3. Recurrent continuous local learning generates nearly perfect time varying trace while full-FORCE with dynamical coupling shows deviations in the beginning.

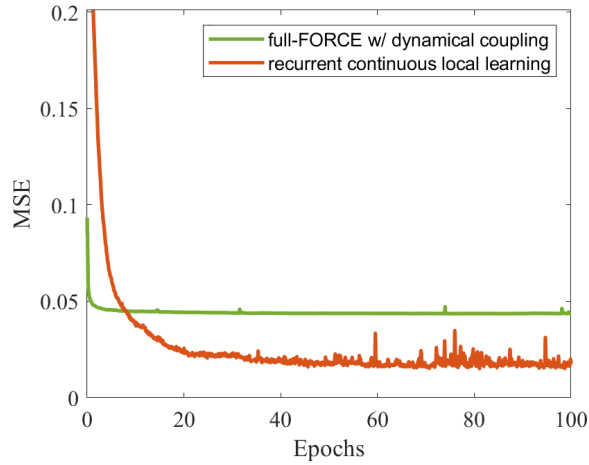


Figure 4.2: Comparing full-FORCE w/ dynamical coupling and recurrent continuous local learning on oscillation generation task. Both network contains one hidden layer with 100 units. Both networks are trained over 100 epochs and the MSE at the end of each training iteration is plotted.

We then test recurrent continuous local learning and dynamically coupled full-FORCE on generating high-dimensional joint movement patterns. This time both networks have 200 hidden units. Similar to the results in oscillation generation task, dynamically coupled full-FORCE converges faster while recurrent continuous local learning generates patterns with lower error, as illustrated in Figure 4.4 and 4.5.

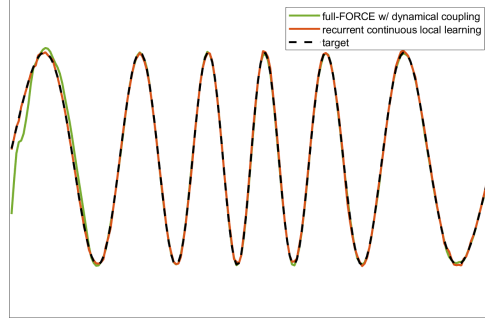


Figure 4.3: Generated oscillations after training for full-FORCE w/ dynamical coupling and recurrent continuous local learning. The black dash trace is the target output. Recurrent continuous local learning is able to generate oscillation closely follows the target while full-FORCE with dynamical coupling struggles in the beginning portion of the task.

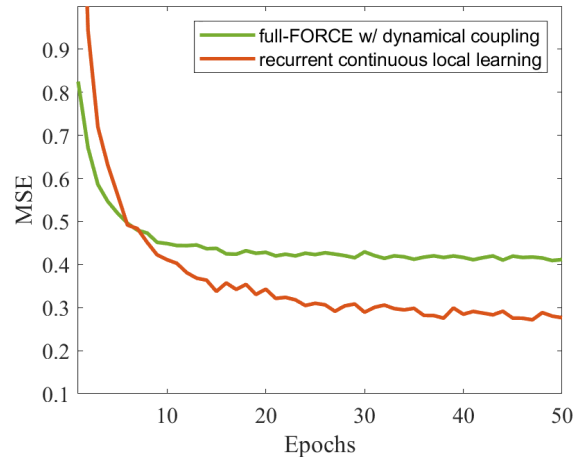


Figure 4.4: Comparing full-FORCE w/ dynamical coupling and recurrent continuous local learning on joint movement generation task. Both network contains one hidden layer with 200 units. Both networks are trained over 50 epochs.

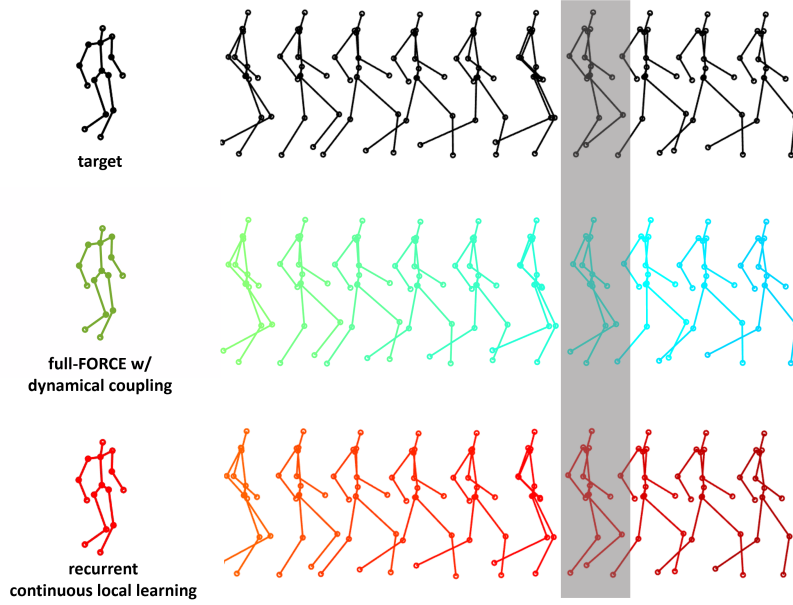


Figure 4.5: Joint movements during walking generated by full-FORCE w/ dynamical coupling and recurrent continuous local learning. Both networks generate patterns close to target. The shaded frame is one example where recurrent neural network outperforms full-FORCE with dynamical coupling.

Overall, training full-FORCE using RLS provides faster convergence than gradient-based method as the algorithm intends to quickly reduce the error between the output and the target in the beginning of learning stage to avoid instability. Updating weight in recurrent continuous local learning, on the other hand, is a more gradual process since a small learning rate is often used in gradient descent technique to escape potential local minimum.

4.2 REDECOLLE

Recurrent neural network (RNN) with one hidden layer that directly generates the output is the basic structure to learn spatiotemporal information. Much like the fact that a feed-forward network with only one hidden layer has limited computational capacity, single layered RNN is not capable of performing difficult tasks, especially when temporal hierar-

chy is presented in the data. Deep recurrent neural network architecture, which consists of multiple hidden layers stacked on one another, has thus been developed and showcased improved computational power. Training of such network, however, typically requires back-propagation through time (BPTT), a non-local learning rule as error signal computed at the output stage is transmitted backward across layers. Build upon the recurrent continuous local learning introduced in previous section, we construct a recurrent deep continuous local learning (REDECOLLE) that incorporates only locally available targets for training weight matrices in each layer.

4.2.1 Network Architecture

Figure 4.6 shows the architecture of REDECOLLE in which feed-forward and recurrent weights in each hidden layer are updated based on local targets. Inspired by full-FORCE [DePasquale et al., 2018], local learning objectives are provided using a target-generating network (TGN). Global target y is fed into the TGN in which all the connections (feed-forward and recurrent ones) are fixed random weights. The dynamics of TGN provide local target at each time step, permitting online learning. REDECOLLE aims to train a RNN in which activities in the hidden layers closely match that in the TGN. The training of hidden layers can be strengthened by adding the synchronization (inspired by dynamical state and parameter estimation technique [Abarbanel et al., 2009] into the dynamics, as demonstrated previously for dynamically coupled full-FORCE. The output layer has direct access to the global target, so it can be adjusted using the cost function of the learning task. This configuration of REDECOLLE can also be viewed as an extension of direct random target projection (DRTP) [Frenkel et al., 2021] to RNNs. DRTP projects target into hidden layers using fixed random weights to address both weight transport and update locking problems. In REDECOLLE, the TGN exists to continuously project global target directly to each hidden layer during training.

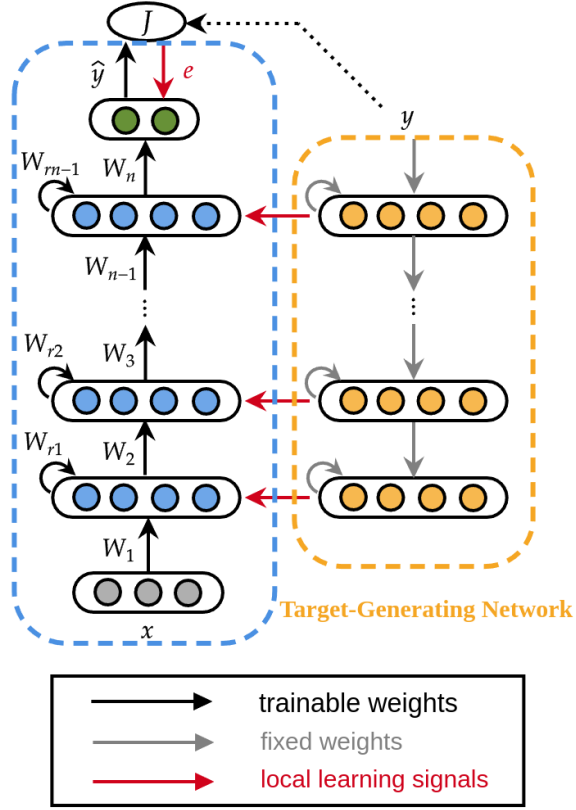


Figure 4.6: Architecture of recurrent deep continuous local learning (REDECOLLE). REDECOLLE features deep architectures with hidden layers connected using feed-forward weights. Unlike traditional RNNs that are trained by backpropagating error signal from the output layer to the hidden ones, REDECOLLE uses additional network (target-generating network) to generate layer specific learning objectives.

4.2.2 REDECOLLE Dynamics

With a multi-layer architecture, the discrete dynamical equations of REDECOLLE can be updated as the following:

$$U_i^l[t] = \sum W_{f_{ij}}^l P_{f_j}^l[t] + \sum W_{r_{ij}}^l P_{r_j}^l[t] - \rho R_i^l[t] + b_i^l \quad (4.15)$$

where $U_i^l[n]$ is the membrane potential of neuron i at layer l at time step n .

The firing threshold is:

$$S_i^l[t] = \Theta(U_i^l[t]) \quad (4.16)$$

P , Q , R can be described using:

$$\begin{aligned} P_{f_j}^l[t + \Delta t] &= \alpha P_{f_j}^l[t] + (1 - \alpha) Q_{f_j}^l[t] \\ Q_{f_j}^l[t + \Delta t] &= \beta Q_{f_j}^l[t] + (1 - \beta) S_j^{l-1}[t] \\ P_{r_j}^l[t + \Delta t] &= \alpha P_{r_j}^l[t] + (1 - \alpha) Q_{r_j}^l[t] \\ Q_{r_j}^l[t + \Delta t] &= \beta Q_{r_j}^l[t] + (1 - \beta) S_j^l[t] \\ R_i^l[t + \Delta t] &= \gamma R_i^l[t] + (1 - \gamma) S_i^l[t] \end{aligned} \quad (4.17)$$

where the constants $\alpha = \exp(-\frac{\Delta t}{\tau_{\text{mem}}})$, $\beta = \exp(-\frac{\Delta t}{\tau_{\text{syn}}})$, and $\gamma = \exp(-\frac{\Delta t}{\tau_{\text{ref}}})$ capture the decay dynamics of the membrane potential U , the synaptic state Q , and the refractory (resetting) state R during a Δt time step (the τ are the respective times constants). Two sets of P and Q are included, one (P_f and Q_f) reflects feed-forward connections while the other (P_r and Q_r) represents the recurrent connections.

Same as the recurrent continuous local learning and DECOLLE, update of feed-forward and recurrent weights follows a biologically inspired learning rule and gradient-based method can be implemented to train REDECOLLE.

We then test REDECOLLE on a real-world biomedical application: blood glucose level forecasting.

4.2.3 Blood Glucose Level Prediction: Description

We seek to apply REDECOLLE to predict blood glucose levels in patients with type 1 diabetes. Existing complex models often demand many inputs, such as carbs and insulin doses. Collecting those inputs is challenging and prone to mislabeling. A powerful RNN has the potential to receive only the blood glucose values from a continuous glucose monitor (CGM - a sensor that shows blood glucose values for the patient) and make accurate predictions for ensuing trends, preventing the patient from having to deal with unexpected hypo or hyperglycemia.

Datasets used in this experiment is from Xie and Wang [2020]. Blood glucose levels from four adult patient are available. Here we use 20 data points in the past as network input and predict 6 points in the future. The time gap between each CGM signal is 5 minutes, so essentially the model is asked to predict 30 minutes ahead with history in the previous 100 minutes. Long carbs usually take around 90 minutes to be ingested and insulin reaches peak at 60-90 minutes.

We train our model on one patient and test it on the rest three. To evaluate the performance of model, we use the root mean square error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^N (y(i) - \hat{y}(i))^2} \quad (4.18)$$

where N is the number of predicted values.

4.2.4 Forecasting Blood Glucose Level: Results

Table 4.1 summarizes the results of using recurrent neural networks to predict blood glucose level. Three 2-layer models (each layer contains 128 units) are compared: full-FORCE, RE-

DECOLLE, and vanilla RNN (trained with BPTT). Models are tested on three patients with different levels of diabetic conditions. When incorporating dynamical coupling, full-FORCE forecasts more accurately, with lower RMSE values for all testing cases. REDECOLLE makes better prediction than both versions of full-FORCE networks with reduced errors. It also generates the lowest RMSE (3.2 mg/dL) for Patient #1. RNN trained using BPTT forecasts the best on Patient #2 & #3, with RMSE values of 4.9 mg/dL and 5.2 mg/dL, respectively.

Table 4.1: Summary of glucose level forecasting results (RMSE values are shown)

	Patient #1	Patient #2	Patient #3
full-FORCE w/o coupling	9.3	12.4	16.9
full-FORCE w/ coupling	6.8	9.7	11.8
REDECOLLE	3.2	5.4	6.5
RNN w/ BPTT	3.7	4.9	5.2

Predicted blood glucose level using REDECOLLE on all patients are plotted in Figure 4.7, 4.8, and 4.9. Overall, REDECOLLE is able to predict general trends but struggle to generate high frequency changes.

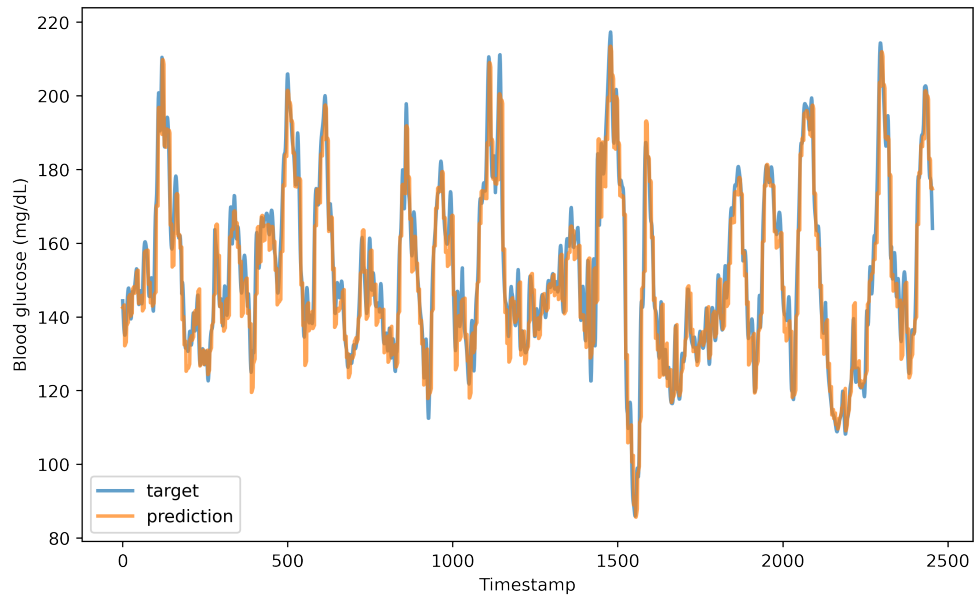


Figure 4.7: Prediction of blood glucose level for Patient #1 using REDECOLLE.

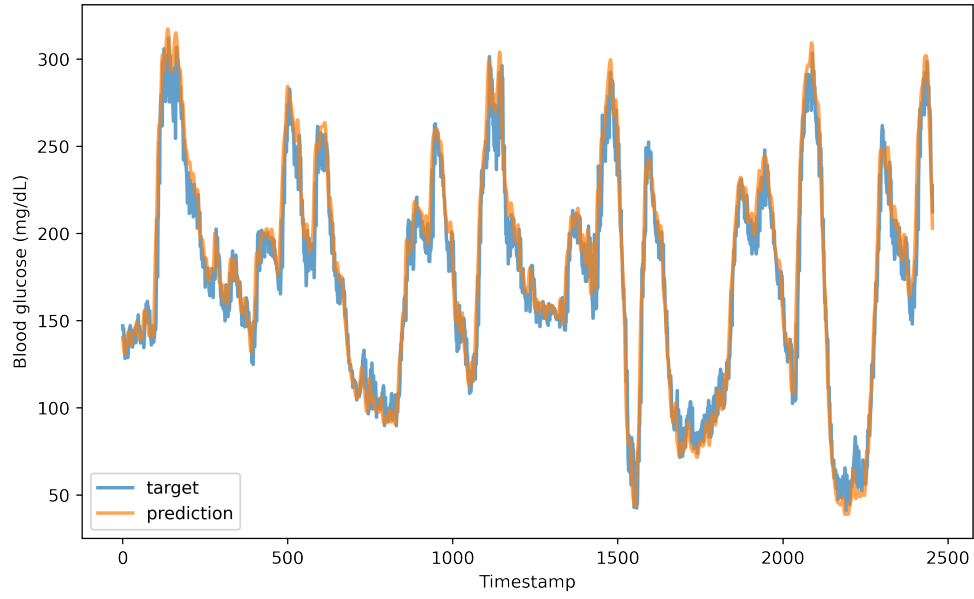


Figure 4.8: Prediction of blood glucose level for Patient #2 using REDECOLLE.

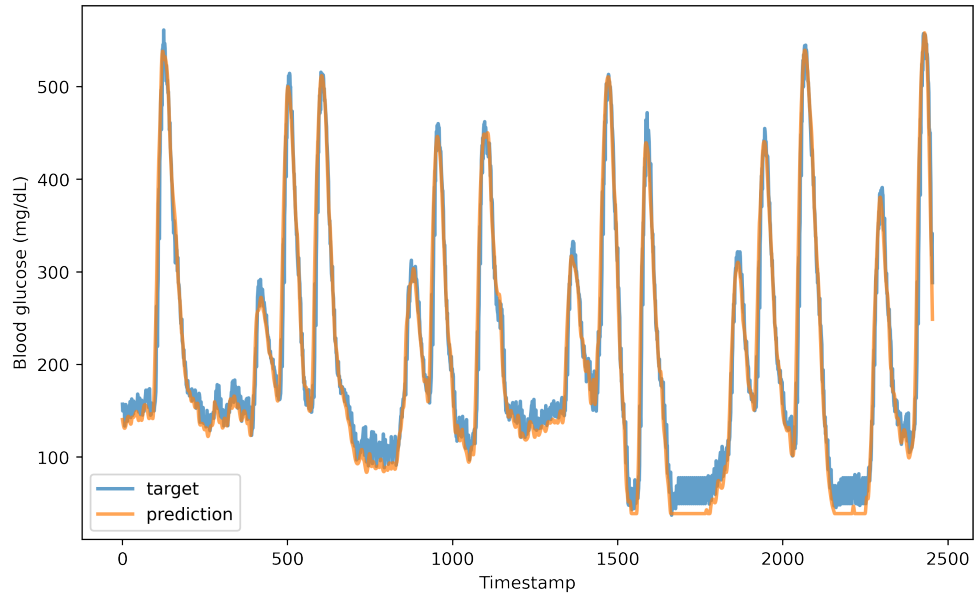


Figure 4.9: Prediction of blood glucose level for Patient #3 using REDECOLLE.

Chapter 5

Conclusion

5.1 Summary

This study introduces two brain-inspired recurrent neural networks (RNNs) and tests them on multiple benchmark experiments and real-world biomedical applications.

We first improve the stability of the state-of-the-art reservoir computing technique (full-FORCE) by coupling the target-generating network with the task-performing network through the dynamical state and parameter estimation (DSPE) method. We also increase the depth of the hidden layer in full-FORCE. This multi-layer full-FORCE with dynamical coupling includes online weight adaptation, layer-specific training target, and backpropagation-free error computation. It yields better results with either lower error or higher accuracy in learning to generate patterns, recognize time intervals, and classify EEG signals. On top of that, it achieves such improvement by exploiting the existing connection in the full-FORCE models and redistributing the units in the network more efficiently. In this way, either the iteration of weight update or the number of weights involved during the training is reduced, directly contributing to a faster learning process, making our model well-suited for real-time

signal processing. Additionally, dynamical data coupling and one added layer in architecture enhance performance when the network’s size is restricted. These are all desirable features for promising applications in neuromorphic computing, where computation expense and power consumption are paramount factors to consider.

We also present a new RNN model that implements continuous local learning rules and is compatible with the modern machine learning library for gradient-based optimization. This recurrent deep continuous local learning (REDECOLLE) inherits synaptic dynamics and enables online learning for weight modification. It uses a target-generating network to provide local learning objectives for the hidden layers, thus demanding less computational resources than the traditional BPTT. We test REDECOLLE on several tasks and conclude that it shows superior results to full-FORCE with better training outcomes.

5.2 Future Work

Our work can be further improved in multiple directions:

- In Chapter 3, we demonstrate that a two-layer network with fewer trainable weights surpasses full-FORCE, yet a three-layer network does not replicate the same trend. This discrepancy suggests that having a suitable learning target for the middle layer of a three-layer network is crucial. Further work on assigning local training information for hidden units in a deep recurrent neural network during online learning is recommended to tackle this problem.
- For all the tasks tested in this thesis, units in our RNNs exhibit real-valued continuous activity. However, implementing them as a spike-based model is possible, and a similar conversion has been delineated on FORCE learning by Nicola and Clopath [2017]. We also derive the dynamics of REDECOLLE, which can be seen as a spiking neural

network. A multi-layer recurrent spiking neural network can then be applied to an event-based neuromorphic system for biomedical signal processing [Donati et al., 2018, Bauer et al., 2019, Corradi et al., 2019, Azghadi et al., 2020, Vitale et al., 2022].

- RNN models described in this dissertation are more biologically plausible with online local learning rules and spike-based implementation capacity. They can be used as the fundamental building block to construct a small brain-like system. One example is modeling the ventral stream. Studies [Koivisto et al., 2011, Kar et al., 2019, Kietzmann et al., 2019] have discovered that recurrent connections in the ventral stream play a vital role in object recognition. It is believed that recurrent connection behaves functionally like additional nonlinear transformation, thus improving the object recognition performance. A convolutional neural network with recurrent connections can be created based on REDECOLLE. This model can also be evaluated using Brain-Score [Schrimpf et al., 2018], a metric that measures the similarity between artificial and biological neural networks for visual processing.

Bibliography

- Henry DI Abarbanel, Daniel R Creveling, Reza Farsian, and Mark Kostuk. Dynamical state and parameter estimation. *SIAM Journal on Applied Dynamical Systems*, 8(4):1341–1381, 2009.
- Larry F Abbott and Sacha B Nelson. Synaptic plasticity: taming the beast. *Nature neuroscience*, 3(11):1178–1183, 2000.
- Mostafa Rahimi Azghadi, Corey Lammie, Jason K Eshraghian, Melika Payvand, Elisa Donati, Bernabe Linares-Barranco, and Giacomo Indiveri. Hardware implementation of deep network accelerators towards healthcare and biomedical applications. *IEEE Transactions on Biomedical Circuits and Systems*, 14(6):1138–1159, 2020.
- Alan Baddeley. Working memory. *Science*, 255(5044):556–559, 1992.
- Pierre Baldi and Peter Sadowski. A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks*, 83:51–74, 2016.
- Pierre Baldi, Peter Sadowski, and Zhiqin Lu. Learning in the machine: Random backpropagation and the deep learning channel. *Artificial intelligence*, 260:1–35, 2018.
- Felix Christian Bauer, Dylan Richard Muir, and Giacomo Indiveri. Real-time ultra-low power ecg anomaly detection using an event-driven neuromorphic processor. *IEEE transactions on biomedical circuits and systems*, 13(6):1575–1582, 2019.
- Mark Bear, Barry Connors, and Michael A Paradiso. *Neuroscience: exploring the brain, enhanced edition: exploring the brain*. Jones & Bartlett Learning, 2020.
- George Bebis and Michael Georgiopoulos. Feed-forward neural networks. *Ieee Potentials*, 13(4):27–31, 1994.
- Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning a synaptic learning rule*. Citeseer, 1990.
- Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.
- Léon Bottou et al. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nimes*, 91(8):12, 1991.

- Geoffrey W Burr, Robert M Shelby, Abu Sebastian, Sangbum Kim, Seyoung Kim, Severin Sidler, Kumar Virwani, Masatoshi Ishii, Pritish Narayanan, Alessandro Fumarola, et al. Neuromorphic computing using non-volatile memory. *Advances in Physics: X*, 2(1):89–124, 2017.
- Yves Chauvin and David E Rumelhart. *Backpropagation: theory, architectures, and applications*. Psychology press, 2013.
- Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.
- Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- KR1442 Chowdhary and KR Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.
- Chrislb. Diagram of an artificial neuron. https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png, 2005.
- Dennis V Christensen, Regina Dittmann, Bernabe Linares-Barranco, Abu Sebastian, Manuel Le Gallo, Andrea Redaelli, Stefan Slesazeck, Thomas Mikolajick, Sabina Spiga, Stephan Menzel, et al. 2022 roadmap on neuromorphic computing and engineering. *Neuromorphic Computing and Engineering*, 2(2):022501, 2022.
- Ami Citri and Robert C Malenka. Synaptic plasticity: multiple forms, functions, and mechanisms. *Neuropsychopharmacology*, 33(1):18–41, 2008.
- Federico Corradi, Sandeep Pande, Jan Stuijt, Ning Qiao, Siebren Schaafsma, Giacomo Indiveri, and Francky Catthoor. Ecg-based heartbeat classification in neuromorphic hardware. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.
- Francis Crick. The recent excitement about neural networks. *Nature*, 337(6203):129–132, 1989.
- Wojciech Marian Czarnecki, Grzegorz Świrszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. Understanding synthetic gradients and decoupled neural interfaces. In *International Conference on Machine Learning*, pages 904–912. PMLR, 2017.
- Yunbin Deng. Deep learning on mobile devices: a review. *Mobile Multimedia/Image Processing, Security, and Applications 2019*, 10993:52–66, 2019.
- Brian DePasquale, Christopher J Cueva, Kanaka Rajan, G Sean Escola, and LF Abbott. full-force: A target-based method for training recurrent networks. *PloS one*, 13(2):e0191527, 2018.

- Elisa Donati, Melika Payvand, Nicoletta Risi, Renate Krause, Karla Burelo, Giacomo Indiveri, Thomas Dalgaty, and Elisa Vianello. Processing emg signals using reservoir computing on an event-based neuromorphic system. In *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4. IEEE, 2018.
- Rodney J Douglas and Kevan AC Martin. Recurrent neuronal circuits in the neocortex. *Current biology*, 17(13):R496–R500, 2007.
- Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940. PMLR, 2016.
- Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. *Advances in neural information processing systems*, 28, 2015.
- Charlotte Frenkel, Martin Lefebvre, and David Bol. Learning without feedback: Fixed random learning signals allow for feedforward training of deep neural networks. *Frontiers in neuroscience*, 15:629892, 2021.
- Rui Fu, Zuo Zhang, and Li Li. Using lstm and gru neural network methods for traffic flow prediction. In *2016 31st Youth academic annual conference of Chinese association of automation (YAC)*, pages 324–328. IEEE, 2016.
- Claudio Gallicchio. Short-term memory of deep rnn. *arXiv preprint arXiv:1802.00748*, 2018.
- Claudio Gallicchio and Alessio Micheli. Deep echo state network (deepesn): A brief survey. *arXiv preprint arXiv:1712.04323*, 2017.
- Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.
- Ian Goodfellow. *Deep learning*, volume 196. MIT press, 2016.
- Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.
- Jordan Guerguiev, Timothy P Lillicrap, and Blake A Richards. Towards deep learning with segregated dendrites. *Elife*, 6:e22901, 2017.
- Simon Haykin. Adaptive filter theory. *Prentice Hall google schola*, 2:333–346, 2002.
- Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. *Advances in neural information processing systems*, 26:190–198, 2013.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

- Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.
- John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.
- Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International conference on machine learning*, pages 1627–1635. PMLR, 2017.
- Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.
- Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic plasticity dynamics for deep continuous local learning (decolle). *Frontiers in Neuroscience*, 14, 2020.
- Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven Siegelbaum, A James Hudspeth, Sarah Mack, et al. *Principles of neural science*, volume 4. McGraw-hill New York, 2000.
- Kohitij Kar, Jonas Kubilius, Kailyn Schmidt, Elias B Issa, and James J DiCarlo. Evidence that recurrent circuits are critical to the ventral stream’s execution of core object recognition behavior. *Nature neuroscience*, 22(6):974–983, 2019.
- Tim C Kietzmann, Courtney J Spoerer, Lynn KA Sörensen, Radoslaw M Cichy, Olaf Hauk, and Nikolaus Kriegeskorte. Recurrence is required to capture the representational dynamics of the human visual system. *Proceedings of the National Academy of Sciences*, 116(43):21854–21863, 2019.
- Mika Koivisto, Henry Railo, Antti Revonsuo, Simo Vanni, and Niina Salminen-Vaparanta. Recurrent processing in v1/v2 contributes to categorization of natural scenes. *Journal of Neuroscience*, 31(7):2488–2492, 2011.
- Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.
- Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.
- Timothy P Lillicrap and Adam Santoro. Backpropagation through time and the brain. *Current opinion in neurobiology*, 55:82–89, 2019.
- Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7(1):13276, 2016.

- Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.
- Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021.
- Clare Liu. <https://www.kdnuggets.com/2020/05/5-concepts-gradient-descent-cost-function.html>. <https://www.kdnuggets.com/2020/05/5-concepts-gradient-descent-cost-function.html>, 2022.
- Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer science review*, 3(3):127–149, 2009.
- Qianli Ma, Lifeng Shen, and Garrison W Cottrell. Deep-esn: A multiple projection-encoding hierarchical reservoir computing framework. *arXiv preprint arXiv:1711.05255*, 2017.
- Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.
- Owen Marschall, Kyunghyun Cho, and Cristina Savin. A unified framework of online learning algorithms for training recurrent neural networks. *Journal of machine learning research*, 21(135):1–34, 2020.
- Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.
- Saurabh Mhatre. What is the relation between artificial and biological neuron? <https://saurabhnativeblog.medium.com/what-is-the-relation-between-artificial-and-biological-neuron-18b05831036>, 2020.
- Marvin Minsky and Seymour A Papert. *Perceptrons, reissue of the 1988 expanded edition with a new foreword by Léon Bottou: an introduction to computational geometry*. MIT press, 2017.
- Emre O Neftci, Charles Augustine, Somnath Paul, and Georgios Detorakis. Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in neuroscience*, 11:324, 2017.
- Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.
- Wilten Nicola and Claudia Clopath. Supervised learning in spiking neural networks with force training. *Nature communications*, 8(1):1–15, 2017.
- Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

- Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.
- Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013a.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013b.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- Rajeev DS Raizada and Stephen Grossberg. Towards a theory of the laminar architecture of cerebral cortex: Computational clues from the visual system. *Cerebral cortex*, 13(1): 100–113, 2003.
- Mukul Rathi. Feedforward neural network. <https://mukulrathi.com/demystifying-deep-learning/feed-forward-neural-network/>, 2018.
- Raul Rojas and Raúl Rojas. The backpropagation algorithm. *Neural networks: a systematic introduction*, pages 149–182, 1996.
- Oliver Rösler and David Suendermann. A first step towards eye state prediction using eeg. *Proc. of the AIHLS*, 2013.
- Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.
- Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- David E Rumelhart, James L McClelland, PDP Research Group, et al. *Parallel distributed processing*, volume 1. IEEE Massachusetts, 1988.
- Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.
- Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007*, pages 471–482, 2007.

- Martin Schrimpf, Jonas Kubilius, Ha Hong, Najib J Majaj, Rishi Rajalingham, Elias B Issa, Kohitij Kar, Pouya Bashivan, Jonathan Prescott-Roy, Kailyn Schmidt, et al. Brain-score: Which artificial neural network for object recognition is most brain-like? *BioRxiv*, page 407007, 2018.
- Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*, 2017.
- Catherine D Schuman, Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, Bill Kay, et al. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, 2(1):10–19, 2022.
- Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.
- Apeksha Shewalkar, Deepika Nyavanandi, and Simone A Ludwig. Performance evaluation of deep neural networks applied to speech recognition: Rnn, lstm and gru. *Journal of Artificial Intelligence and Soft Computing Research*, 9(4):235–245, 2019.
- Stewart Shipp. Structure and function of the cerebral cortex. *Current Biology*, 17(12):R443–R449, 2007.
- David Sussillo and Larry F Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009.
- Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, ON, Canada, 2013.
- Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.
- Larry W Swanson, Eric Newman, Alfonso Araque, and Janet M Dubinsky. *The beautiful brain: the drawings of Santiago Ramón y Cajal*. Abrams, 2017.
- Alan J Thomas, Miltos Petridis, Simon D Walters, Saeed Malekshahi Gheytsi, and Robert E Morgan. Two hidden layers are usually better than one. In *Engineering Applications of Neural Networks: 18th International Conference, EANN 2017, Athens, Greece, August 25–27, 2017, Proceedings*, pages 279–290. Springer, 2017.
- Robert Urbanczik and Walter Senn. Learning by the dendritic prediction of somatic spiking. *Neuron*, 81(3):521–528, 2014.
- Muhammad Uzair and Noreen Jamil. Effects of hidden layers on the efficiency of neural networks. In *2020 IEEE 23rd international multitopic conference (INMIC)*, pages 1–6. IEEE, 2020.

- Aleksandar Vakanski, Hyung-pil Jun, David Paul, and Russell Baker. A data set of human body movements for physical rehabilitation exercises. *Data*, 3(1):2, 2018.
- Antonio Vitale, Elisa Donati, Roger Germann, and Michele Magno. Neuromorphic edge computing for biomedical applications: Gesture classification using emg signals. *IEEE Sensors Journal*, 22(20):19490–19499, 2022.
- Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
- Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.
- Jinyu Xie and Qian Wang. Benchmarking machine learning algorithms on blood glucose prediction for type i diabetes in comparison with classical time-series models. *IEEE Transactions on Biomedical Engineering*, 67(11):3101–3124, 2020.
- Jia-Qin Yang, Ruopeng Wang, Yi Ren, Jing-Yu Mao, Zhan-Peng Wang, Ye Zhou, and Su-Ting Han. Neuromorphic engineering: from biological to spike-based hardware nervous systems. *Advanced Materials*, 32(52):2003610, 2020.
- Izzet B Yildiz, Herbert Jaeger, and Stefan J Kiebel. Re-visiting the echo state property. *Neural networks*, 35:1–9, 2012.
- Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.
- Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pages 3987–3995. PMLR, 2017.

Appendix A

Eigenvalues & Unit Activities for Oscillation Generation

For all 8 full-FORCE networks trained in Table 3.1, we compute the eigenvalues of the recurrent weight in the task-performing network. Figure A.1 plots all the eigenvalues of learned connectivity. For the networks that show better results in generating oscillation, the eigenvalues are perturbed more, indicating by the shrunken range compared to the initial connectivity. Figure A.2 also shows that the norm of most eigenvalues are significantly smaller than that before training.

We also plot activities from selected hidden units in the task-performing network and compare that with the ones in the target-generating network. The results are presented in Figure A.3 and A.4. Activities from networks with 400 units in general follow closely with that from the target network, which explains the better performance compared to the networks with only 200 units.

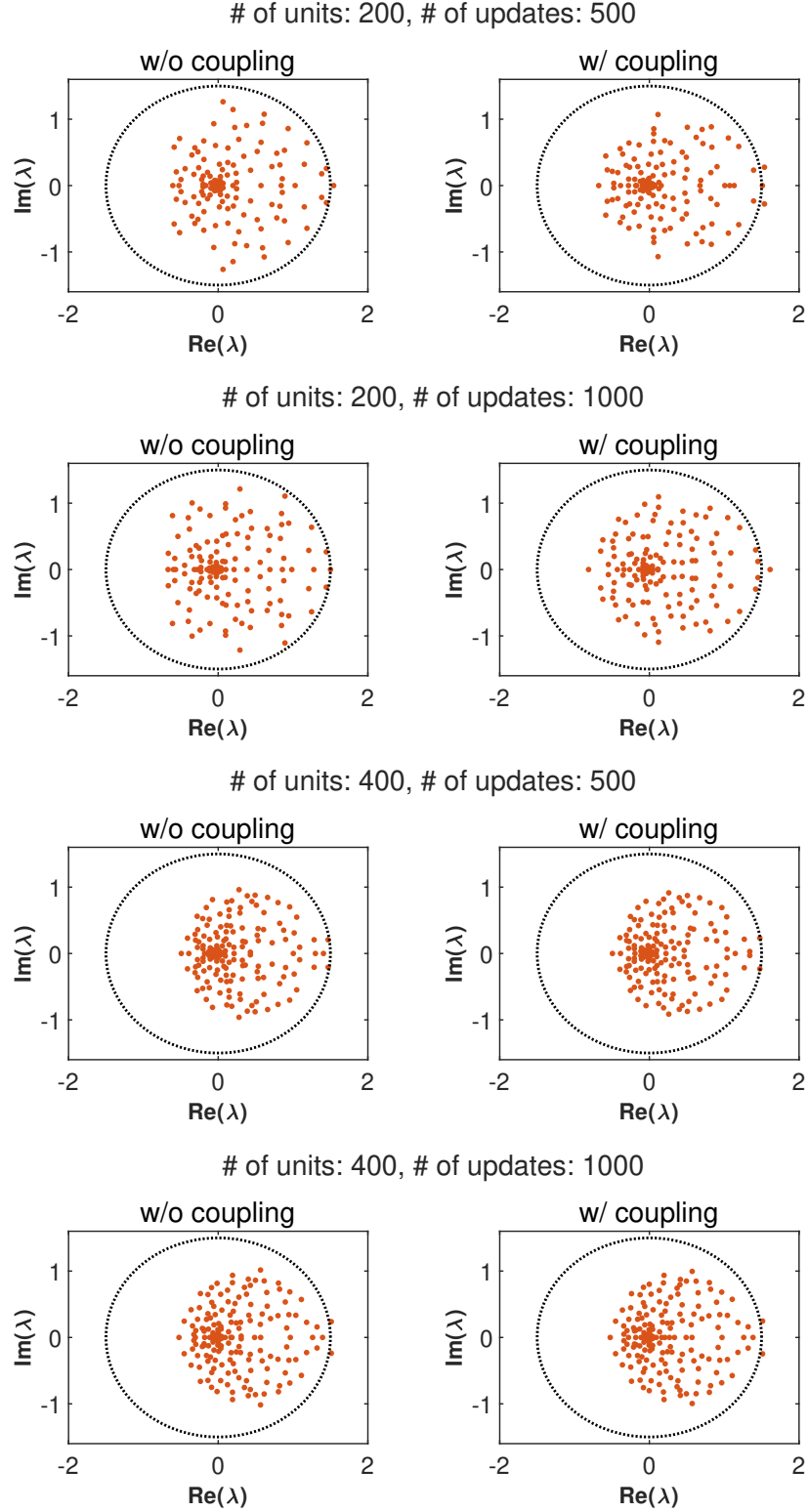


Figure A.1: Eigenvalues of the recurrent weight after training. The real and imaginary of eigenvalues are plotted (orange dots) and the dotted circle in each plot shows the range of eigenvalues during the initialization.

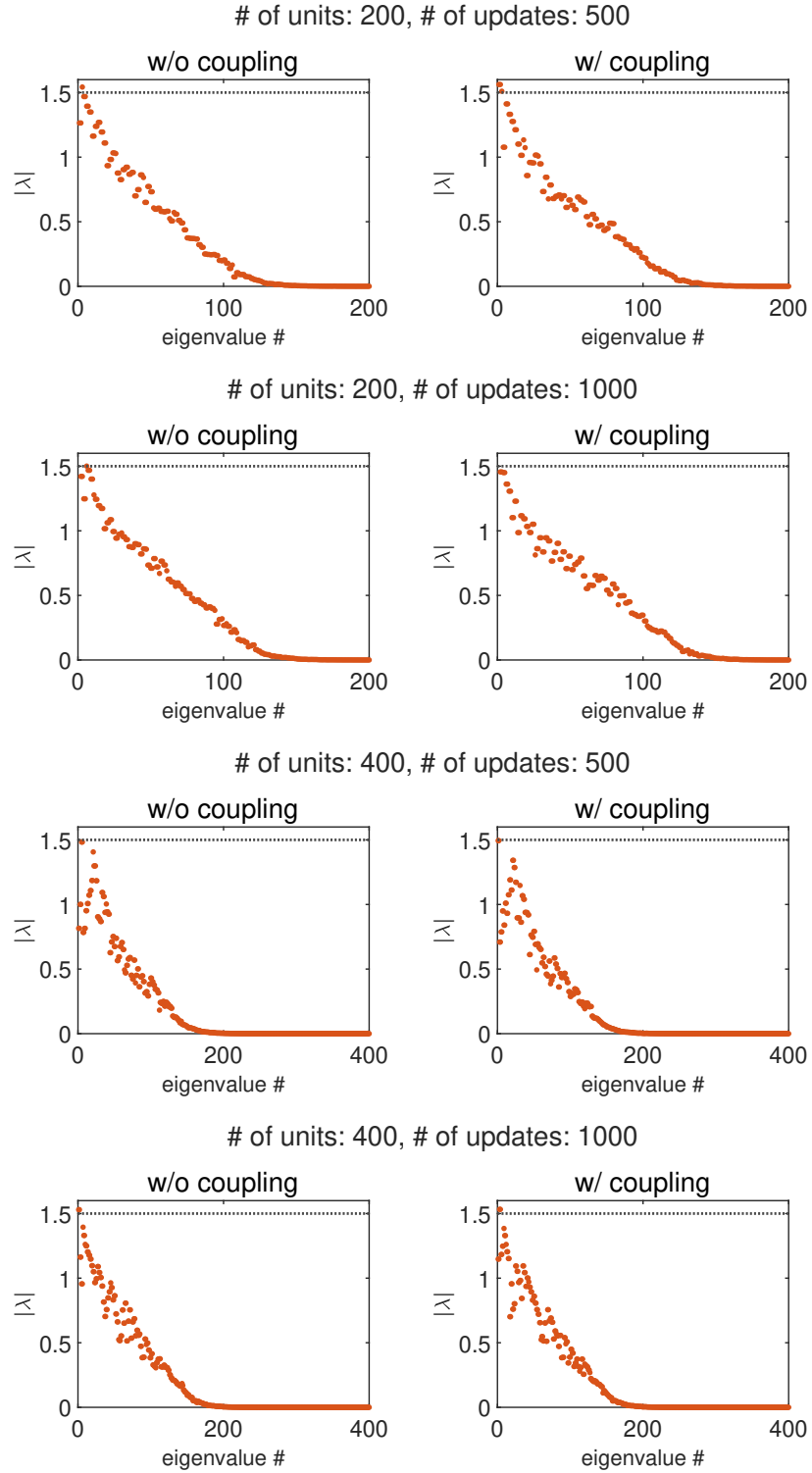


Figure A.2: Complex norm of eigenvalues after training. The dotted line represents the range of eigenvalues before training.

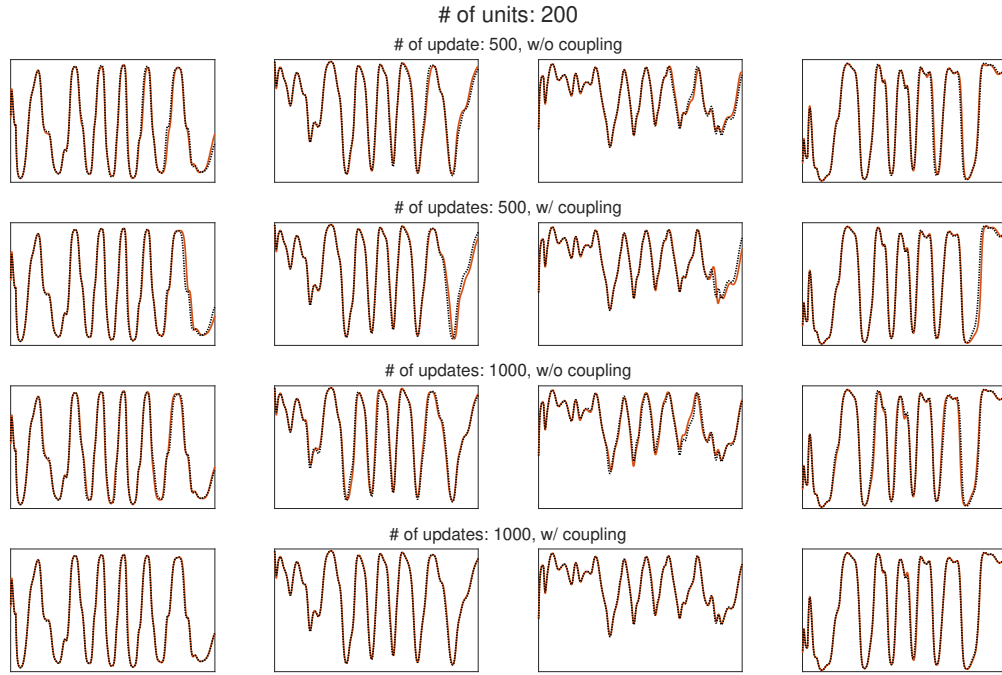


Figure A.3: Sample activities from the trained network (orange solid) with 200 units compared to the ones provided by the target-generating network (black dotted).

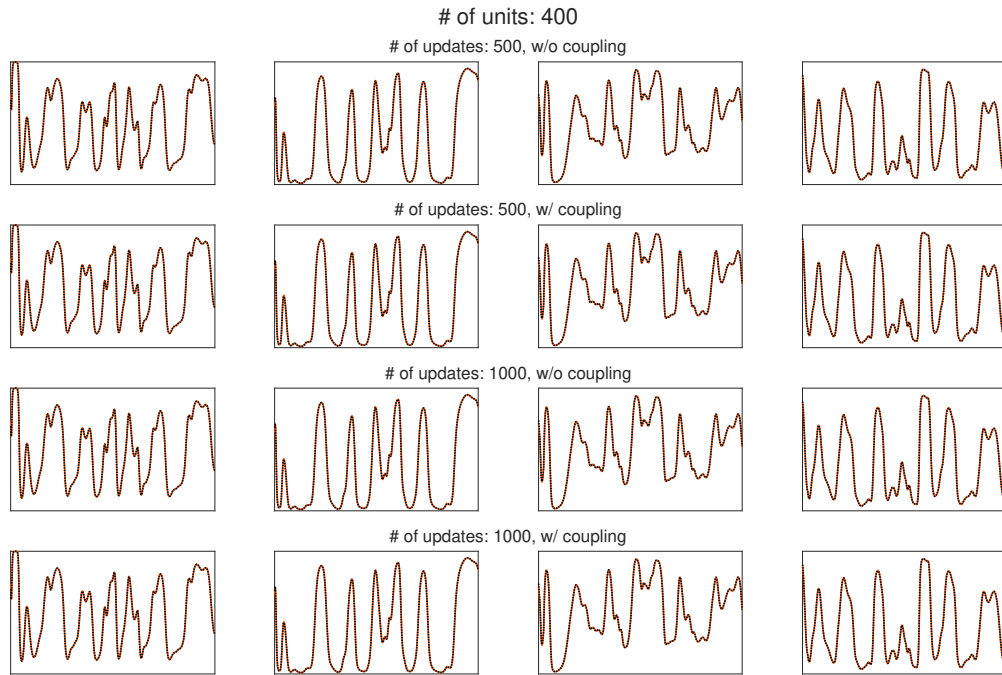


Figure A.4: Sample activities from the trained network (orange solid) with 400 units compared to the ones provided by the target-generating network (black dotted).

Appendix B

Activities of the 1st Hidden Layer for Movement Generation

We show how units in the first hidden layer of a multi-layer full-FORCE respond to 10 different input signals. Here the best performing network presented in Table 3.2 is used. For that network, the first hidden layer contains 160 units and receives input signal directly. 10 different input signals are used in the movement generation task.

Figure B.1 to B.10 plots activities of all 160 hidden units when different input signals are presented. The intensity of those activities can be evaluated by computing the total energy of each trace. These intensities can then be normalized within each unit based on how it responds to all the inputs. Figure B.11 summarizes the results. For each unit, the response (evaluated using normalized intensity) to each input is plotted. This serves as a starting point to understand the role of each hidden layer in a deep neural network architecture.

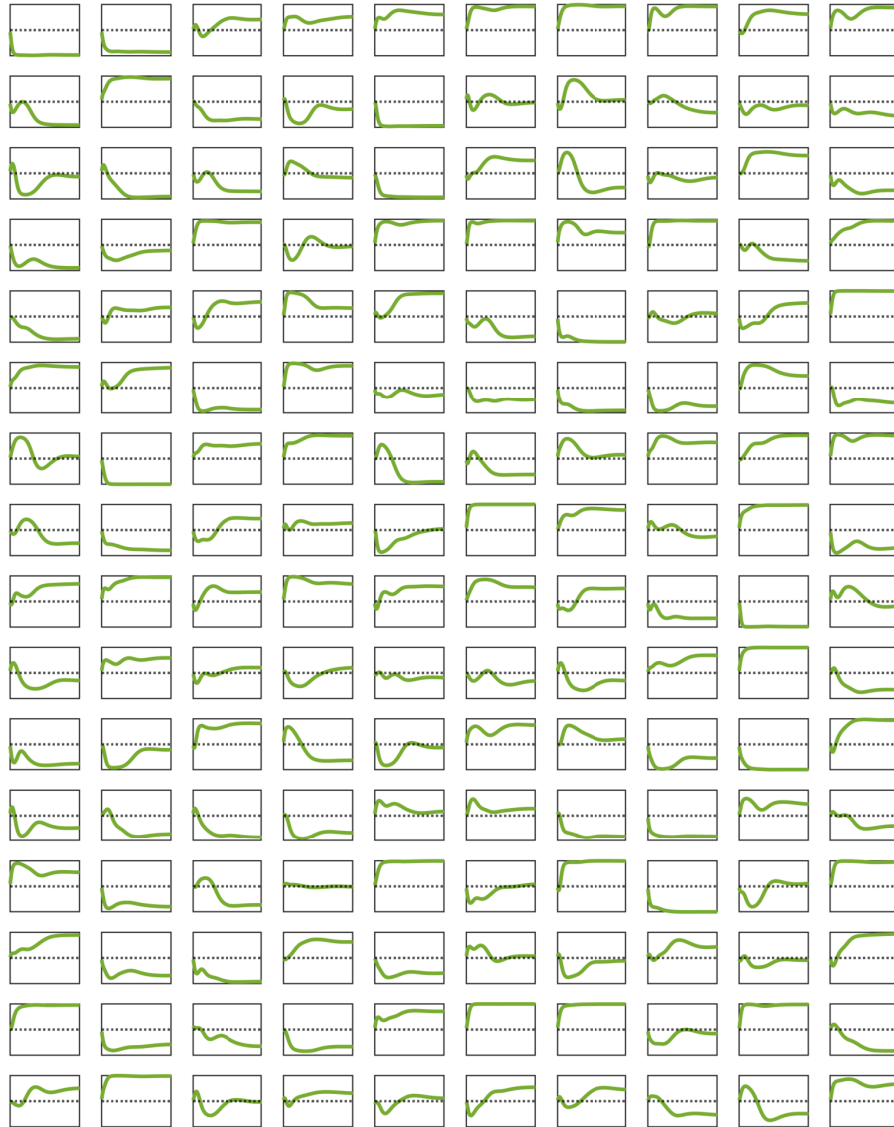


Figure B.1: Activity of each hidden unit when the network is asked to generate movement #1. 160 plots correspond to 160 hidden units.

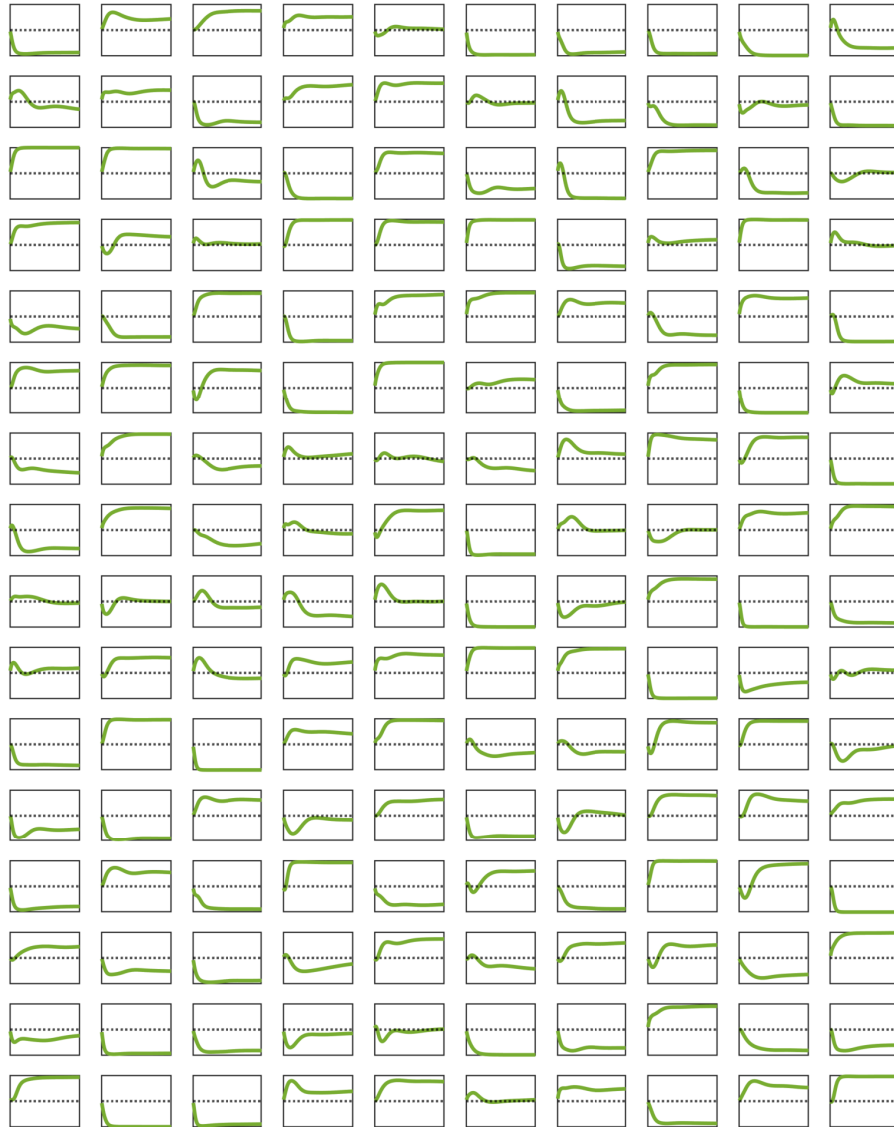


Figure B.2: Activity of each hidden unit when the network is asked to generate movement #2. 160 plots correspond to 160 hidden units.

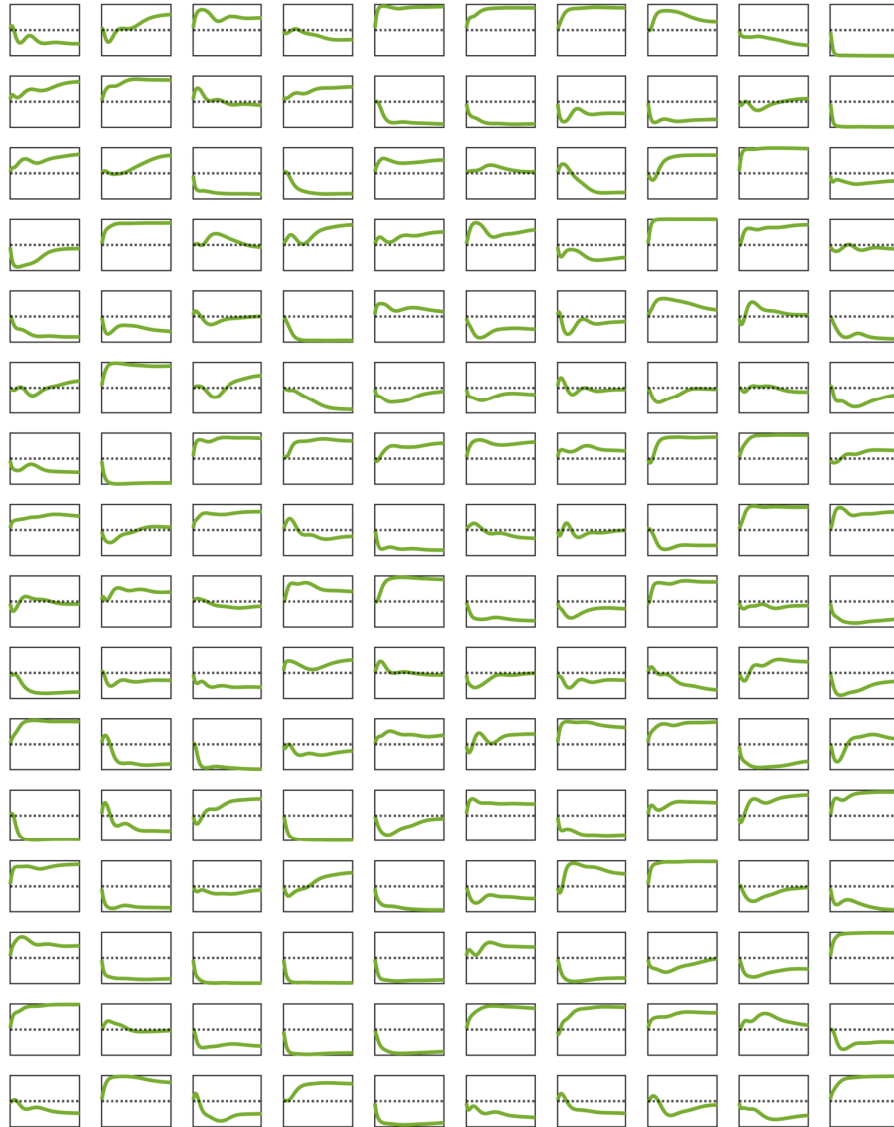


Figure B.3: Activity of each hidden unit when the network is asked to generate movement #3. 160 plots correspond to 160 hidden units.

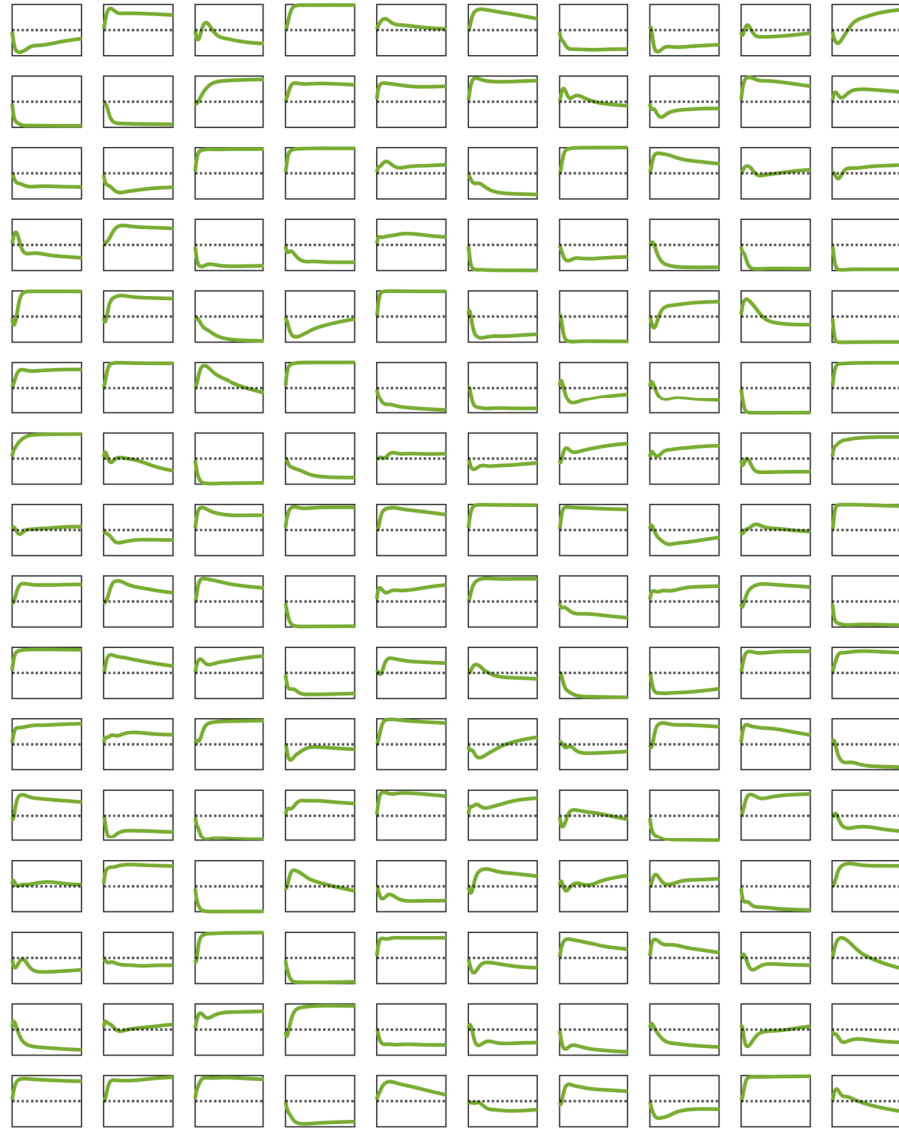


Figure B.4: Activity of each hidden unit when the network is asked to generate movement #4. 160 plots correspond to 160 hidden units.

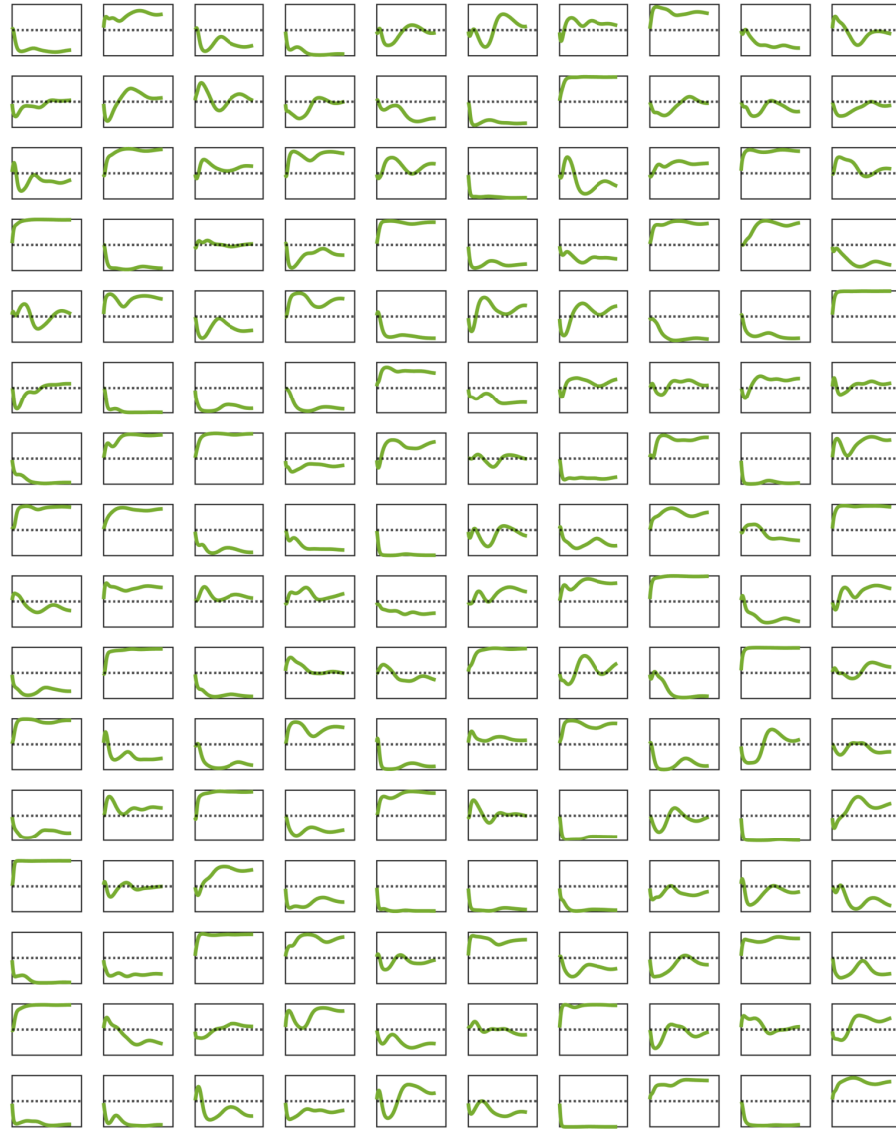


Figure B.5: Activity of each hidden unit when the network is asked to generate movement #5. 160 plots correspond to 160 hidden units.

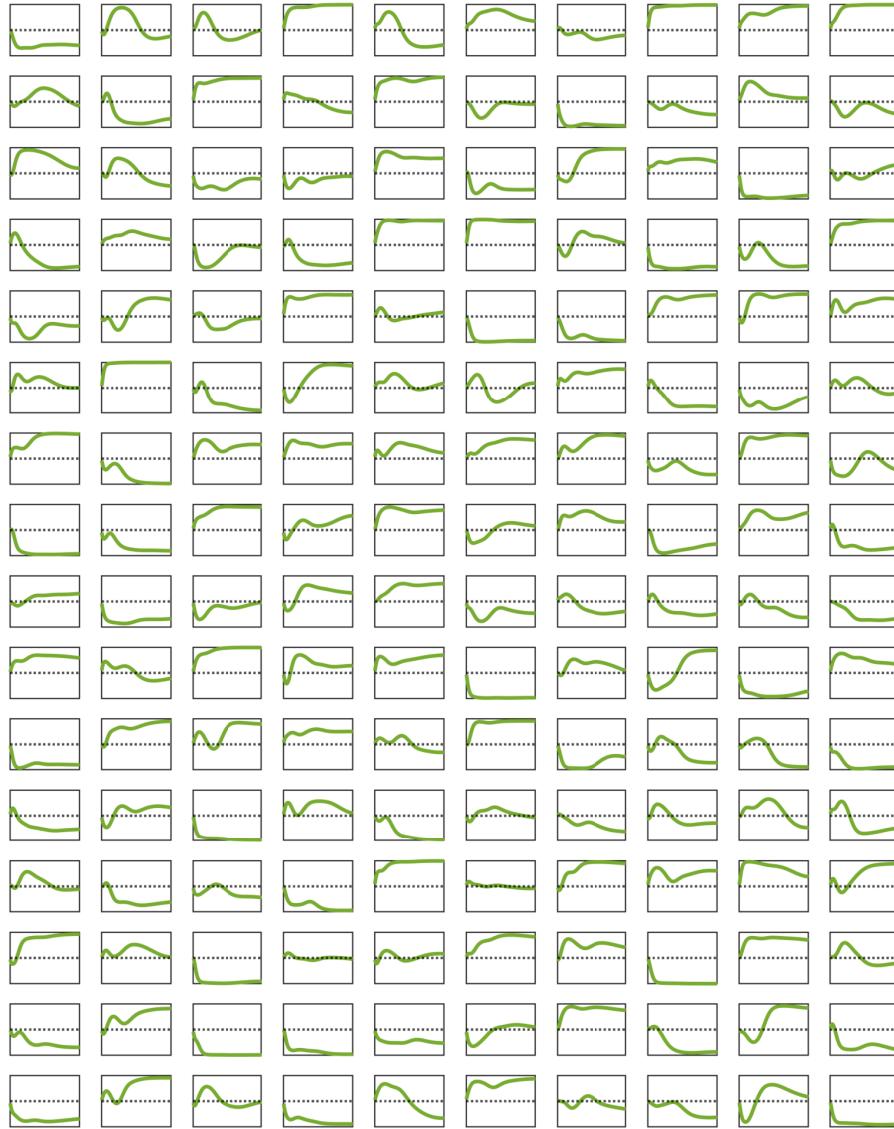


Figure B.6: Activity of each hidden unit when the network is asked to generate movement #6. 160 plots correspond to 160 hidden units.

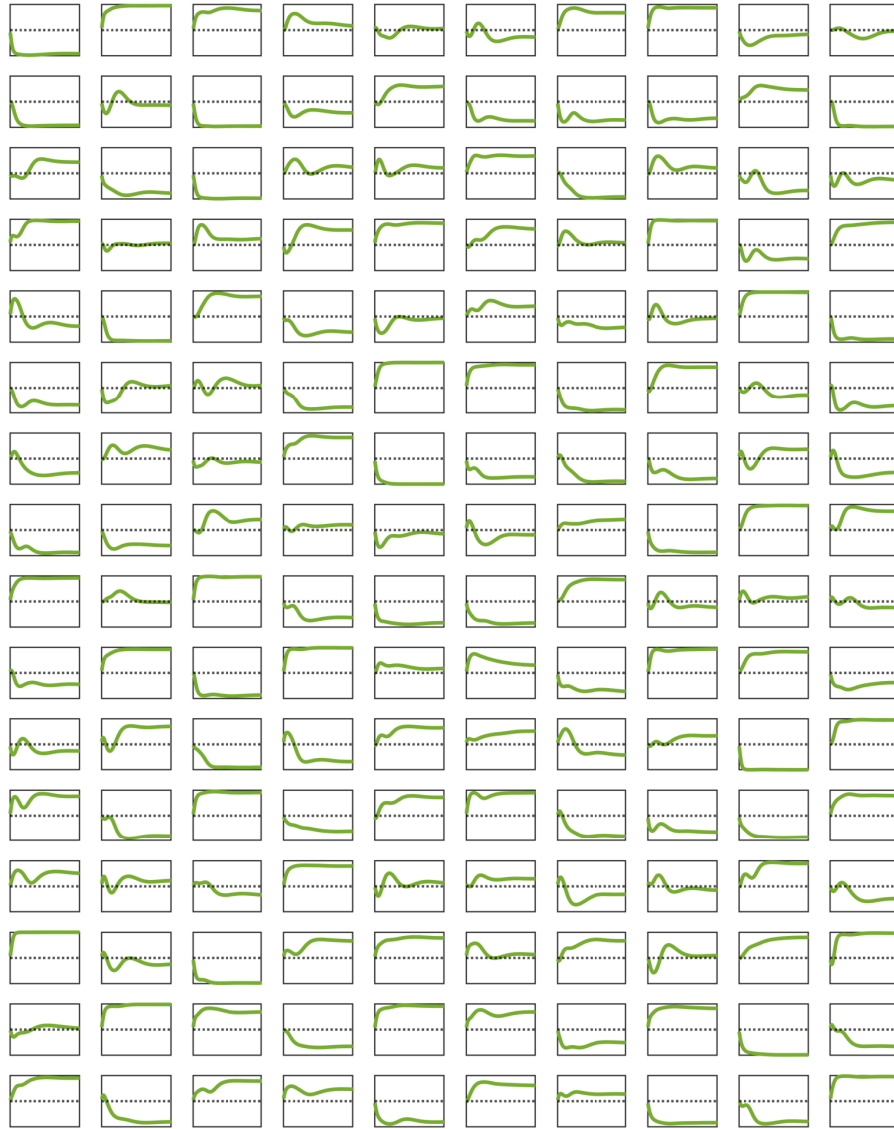


Figure B.7: Activity of each hidden unit when the network is asked to generate movement #7. 160 plots correspond to 160 hidden units.

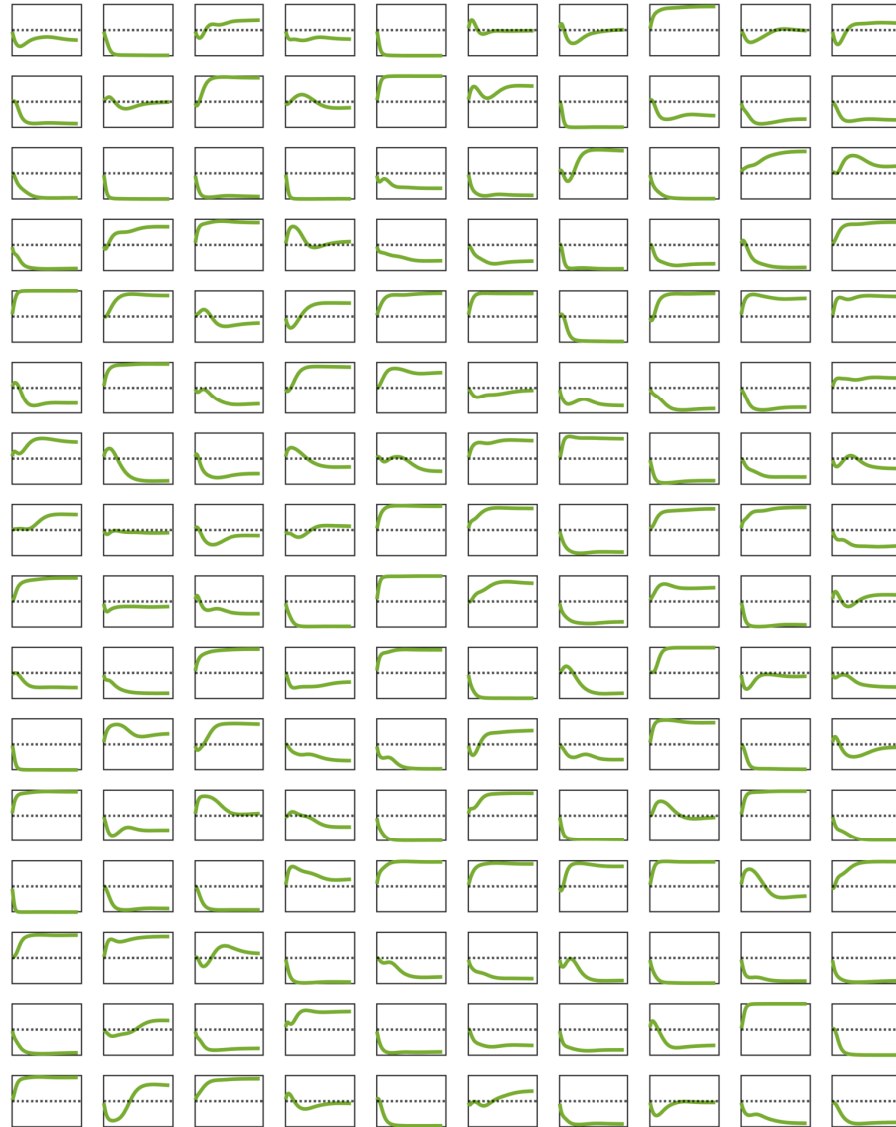


Figure B.8: Activity of each hidden unit when the network is asked to generate movement #8. 160 plots correspond to 160 hidden units.

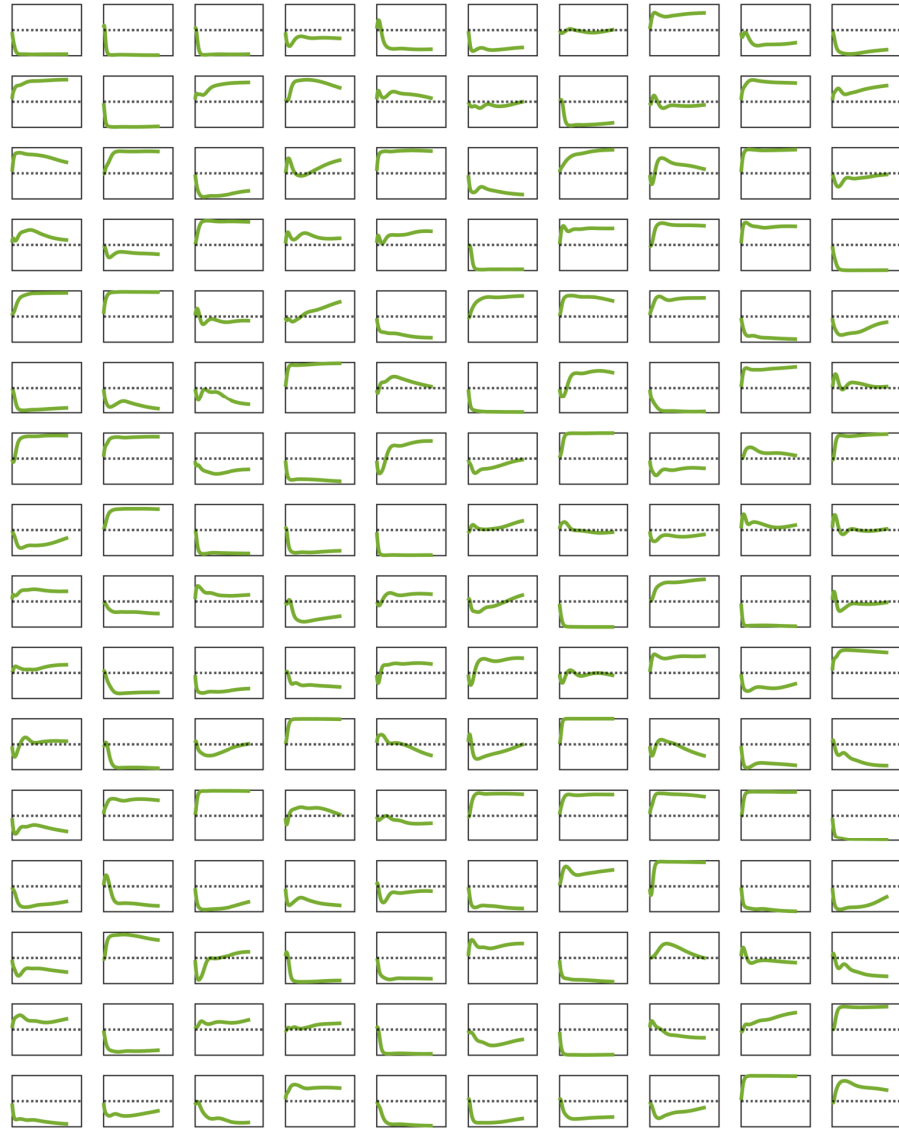


Figure B.9: Activity of each hidden unit when the network is asked to generate movement #9. 160 plots correspond to 160 hidden units.

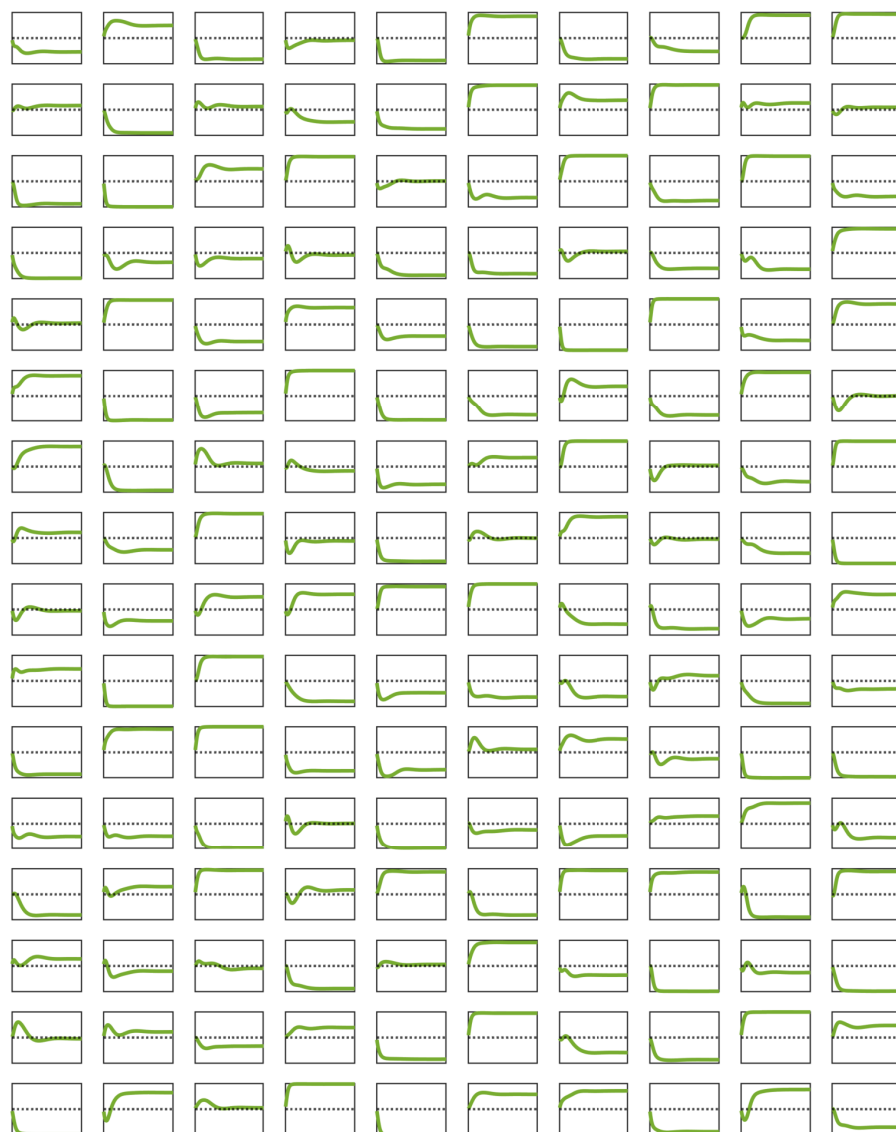


Figure B.10: Activity of each hidden unit when the network is asked to generate movement #10. 160 plots correspond to 160 hidden units.

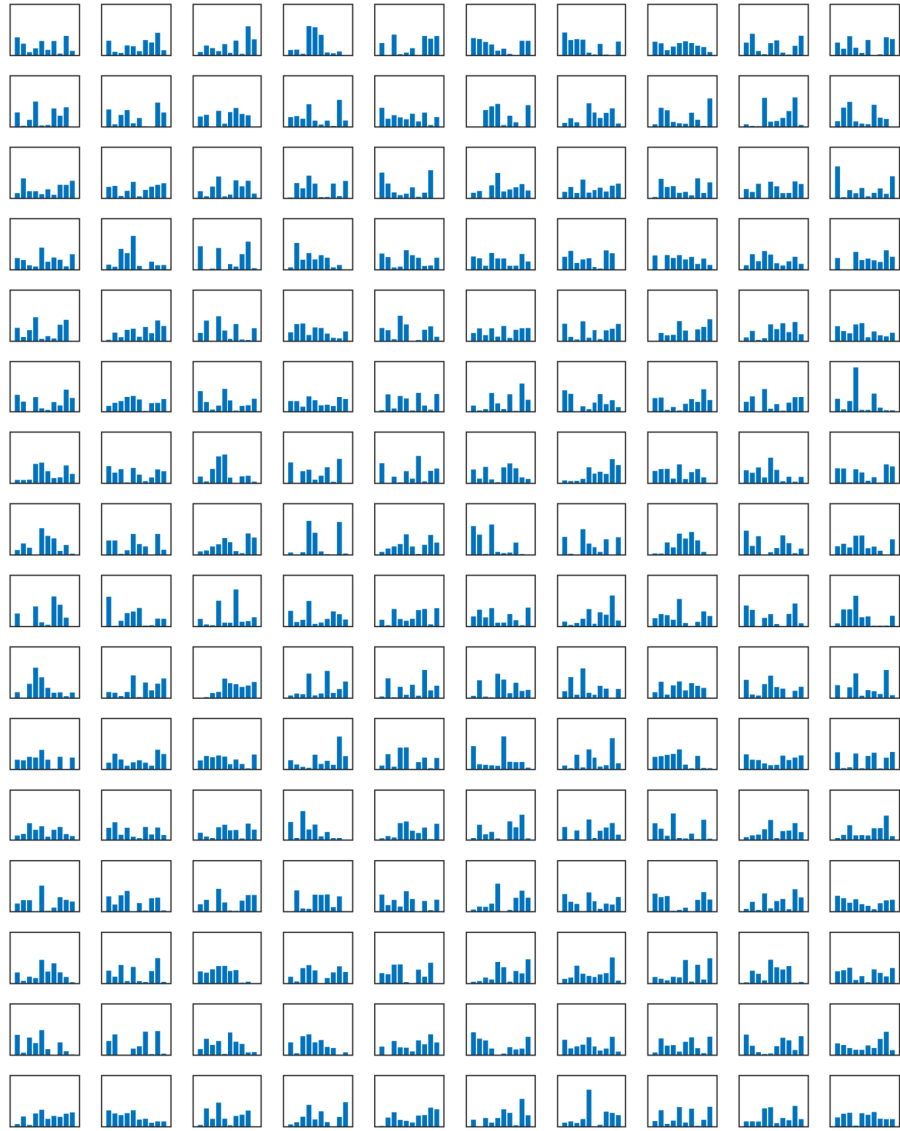


Figure B.11: Summary of hidden unit's response to different inputs. Each panel represents 1 out of 160 units. For each unit, the normalized activity intensity is plotted. There are 10 different input signals, hence 10 bars in each plot.