UNIVERSITY OF CAMBRIDGE
Department of Engineering

Computation in Biological Neural Networks

Author Name: Muhammad Hamza Chhapra
Supervisor: Dr. Yashar Ahmadian Tehrani
Date: 26/05/2024

I hereby declare that, except where specifically indicated, the work submitted herin is my own original work.

Signed _____ *date* _____26/05/2024_____

Computation in Biological Neural Networks

Muhammad Hamza Chhapra

Supervisor: Dr. Yashar Ahmadian

Department of Engineering

University of Cambridge

This report is submitted for the degree of

Master of Engineering

Trinity College

May 2024

Computation in Biological Neural Networks

Muhammad Hamza Chhapra

Trinity College

Supervisor: Dr. Yashar Ahmadian

May 2024

Technical Abstract

Traditional recurrent neural networks are primarily trained through the use of backpropagation through time (BPTT). This is a powerful algorithm, that allows for the training of networks with long-term dependencies. However it is also a biologically implausible learning method. There are two properties used by BPTT that are not found in biological systems. These are the use of non-local signals and signals from earlier timesteps. These implausibilities arise from the fact that biological neurons do not have access to the states of cells they are not directly connected to and large biological networks are not fully connected, thus any given neuron would only receive signals from a local sub-set of neurons. Furthermore, biological neurons do not retain precise information of past activity due to the transient nature of chemical and electrical signals in these networks. A mathematically equivalent algorithm, real-time recurrent learning (RTRL) overcomes the need to store past activity, but still relies on non-local signals, and is very expensive computationally. It also still makes use of non-local signals. A third algorithm, the Random-Feedback Local Online (RFLO) learning rule, is a biologically plausible approximation to RTRL that only modifies synapses based on local signals. However, RFLO is unable to learn complex tasks. It still remains a questions as to how biological neural networks can learn complex tasks using synaptic plasticity rules that only rely on local signals. One feature of biological synapses that is missing in artificial neural nets is their internal dynamics over multiple timescales. This notion of multiple timescales arises from various biological processes that alter synaptic efficacy at different speeds. For example, phosphorylation of AMPA receptors occurs after increased neural activity and increases synaptic efficacy on the order of seconds to minutes, while CREB is a transcription factor that over the course of hours or days alters

long term neuronal plasticity via regulation of gene expression. Abstracting these processes and modelling them as dynamical variables, here we investigate whether imbuing an RNN's connections with complex synapses can allow it to learn effectively in a self-organized way and based on information that is local to each synapse. To optimise the parameters governing these synaptic dynamics we perform meta-learning on a battery of tasks and compare this performance to that of a standard RNN trained via BPTT and RFLO. Model training fell under two broad approaches, gradient based meta-learning and non-gradient optimisation methods. The latter methods were unable to perform well with the noisiness of training while the former can be subdivided further. The first approach was an unsupervised meta-learning approach in which the network and synapses were run in parallel, the loss evaluated and backpropagation carried out. This method was computationally prohibitive so ultimately a supervised approach was selected. Here learning signals and gradients were generated by training an RNN using BPTT, these formed the inputs and outputs for our synapses to learn. Once trained we evaluated the performance of this new architecture on synthetic continuous tasks. The outcome was a stark under-performance compared to an RFLO trained RNN which itself was worse than a BPTT trained RNN. The three networks performance on a battery of 5 tasks can be summarised by their Pearson correlation coefficients, $-0.1, 0.21,$ and 0.98 respectively for 20 neuron networks. We suspect the learning signals were responsible for this under-performance. In order to verify such a claim, we added an arbitrary number of layers for both the input and output layers of the complex synapses to ascertain whether the network can over-fit to its training data. This was unsuccessful as we achieved prediction $r^2 \simeq 0.4$ which is too inaccurate for the gradient prediction required of the complex synapses. This serves as evidence that the learning signals do not possess sufficient information for the complex synapses to learn the mapping for the gradients. To assess the viability of complex synapses in future work, it would be worth providing the local signals as they are and letting the network implicitly learning functions from them. Additionally providing non-local signals to assess if complex synapses in the best case are competitive with BPTT would help determine if this is a viable approach for a biologically plausible learning algorithm.

Table of contents

List of figures **vi**

1 Introduction **1**

2 Background **3**

2.1 Backpropagation through Time 3

2.2 Real-time Recurrent Learning 5

2.3 Local online learning in recurrent networks with random feedback, Murray . . 7

2.4 Computational principles of synaptic memory consolidation, Benna and Fusi . 9

2.5 Meta-learning biologically plausible plasticity rules, Shervani-Tabar and Rosenbaum 11

3 Methods **15**

3.1 Model Architecture 15

3.2 Meta-Learning Approaches 18

3.2.1 Supervised Meta-Learning of BPTT Gradients 18

3.2.2 Unsupervised Meta-Learning using BPTT 20

3.3 Optimisation Approaches 21

3.3.1 NGOpt by Meta 22

3.3.2 Evolutionary Strategies 22

3.4 Discussion on Rationale and Biological Plausibility of Model 24

3.5 Sequential Task Creation 25

3.5.1 EMNIST Dataset 26

3.5.2 Gaussian Process Tasks 28

4 Results **31**

4.1 Performance on Continuous Tasks 31

4.2 Performance on Classification Tasks 35

4.3 Limitations and Suggestions for Improvement 35

4.3.1 Information Content of Learning Signals 35

4.3.2 Addressing Large Weight Values 38

5 Conclusion **39**

References **40**

Appendix A Risk Assessment Retrospective **41**

A.1 Acknowledgment 41

List of figures

2.1 *Local online learning in recurrent networks with random feedback*, Murray [7]. Sub-figure (a) demonstrates the loss of BPTT vs RFLO during training on a task to output a one dimensional periodic output. The right panel shows the fit of the produced signal versus the desired one. Sub-figure (b) shows the loss for the two learning algorithms as the period of the desired output increases compared to the RNN time constant τ. Sub-figure (d) shows the loss for the intermediate modifications of the RFLO algorithm, where only random feedback or locality is enforced. 8

2.2 *Computational principles of synaptic memory consolidation*, Benna and Fusi [2]. Beaker Representation of the dynamical variables in the model. Sub-figure (a) has equally sized beakers with equal cross-section between connections. Sub-figure (b) has merged beakers such that we receive progressively larger beakers that is, $C_k = 2C_{k-1}$. Sub-figure (c) is the same as (b) but now the cross-sections of the connections are inversely scaled with beaker size that is, $g_{k,k+1} = \frac{1}{2}g_{k-1,k}$. 9

2.3 Effect of long term potentiation (LTP) and depression (LTD), and spacing of inputs on synaptic efficacy. [2] 10

2.4 *Meta-learning biologically plausible plasticity rules with random feedback pathways*, Shervani-Tabar and Rosenbaum [8]. Performance of a classifier network trained with a pool of plasticity rules, backpropagation and feedback alignment. The top figure plots the meta-loss and the bottom figure shows the convergence of the plasticity coefficients. 13

2.5 Candidate plasticity rules from Shervani-Tabar and Rosenbaum [8]. 14

3.1 *Versatile Black-Box Optimization*, Liu et al. [6]. Features of the optimisation problem considered in determining which algorithm to use. 22
3.2 Paths illustrating the two possible sequentalisation strategies considered. Figure courtesy of Smith et al. [10]. 26
3.3 Nine sample images contained in the EMNIST dataset. Figure courtesy of TensorFlow [11]. 27
3.4 Plots of 3 different Fourier series illustrating the ordering of values in x(t). . . . 29
3.5 Plots of two tasks train and test sets. Each task consists of 2 sequences sampled at the coloured dots. 30

4.1 Loss of a 20 neuron RNN trained using BPTT and RFLO on a separate validation set over the course of training over 2 different tasks. 32
4.2 Plots of predictions on specific sequences of different tasks by a 20 neuron RNN trained using BPTT and RFLO. The ground truth is included as well. These plots are sub-sampled from a set of 2 tasks with 5 validation sequences. 33
4.3 Plots of predictions on specific sequences of different tasks by a 20 neuron RNN trained using BPTT and complex synapses. The ground truth is included as well. These plots are sub-sampled from a set of 2 tasks with 5 validation sequences. . 34
4.4 Plots of predictions on specific sequences of different tasks by a 20 neuron RNN trained using BPTT, RFLO and complex synapses. The ground truth is included as well. These plots are sub-sampled from a set of 5 tasks with 10 validation sequences. 34
4.5 Loss for various tasks on the validation set. The two sub-figures illustrate the different local minima the network parameters converged to. 36

Chapter 1

Introduction

Recurrent neural networks are trained using backpropagation through time (BPTT), a very powerful learning algorithm which propagates the error signal backwards through the connections of the network. This enables the reliable training of large networks for a wide range of problems. However, BPTT poses some issues when considering biological plausibility. Firstly, biological neurons are unable to store past activities, that is they are local in time while BPTT requires this storage. Secondly, the propagation of gradient information to earlier time steps assumes symmetric connectivity, a phenomenon not observed in biological neurons. Lastly, the weight updates in BPTT make use of global information, that is a neuron can impact the weight of every other neuron in the network which contrasts with biological neurons that only receive local information to update their states.

Real time Recurrent learning (RTRL) [12] is an online learning algorithm that is able to solve for the need to store past activity. It does so by calculating and maintaining gradient information in the forward pass and so is able to update its weights upon receiving an error signal. The drawback of this approach is that it scales poorly with the size and depth of the network rendering it impractical for use in large networks. For reference BPTTs time complexity scales as $O(N^2)$ while RTRL scales as $O(N^4)$ where N is the number of neurons in the network. Additionally, while RTRL is local in time, it is not local in space as it still makes use of global information. Thus it is not a biologically plausible alternative to BPTT.

A third learning algorithm, Random Feedback Local Online (RFLO) [7] was proposed to solve this issue of spatial locality by approximating gradients using pre- and post-synaptic information

as well as a random feedback of the error signal. While this algorithm overcomes the biological implausibility of BPTT and can be used to train RNNs on a variety of tasks, it has 2 drawbacks. It is not able to learn as well as BPTT, and it tends to struggle with longer input sequences.

Motivated by these algorithms, we will explore an alternative method of achieving a biologically plausible learning algorithm. Neural network weights are represented by a scalar value which is fixed once the network is fine-tuned to its training data. While this works well it does not adequately represent the complexity present within biological synapses. Thus we propose a complex model of the synapses inspired by that of Benna and Fusi [2], where the internal synaptic dynamics will receive local information and govern the learning of the network. These complex synapses receive some set of candidate learning signals inspired by Shervani-Tabar and Rosenbaum [8] and Murray [7] from which they will output an update to the RNN's weights. Included in these learning signals will be a global error signal which contradicts our requirement of local signals, the biological plausibility of this is argued by Lillicrap et al. [4] who suggest that feedback connections may induce neural activity that can be used to locally approximate these signals. Lastly, not all synaptic dynamics will be suitable and so we will optimise this architecture by meta-learning on a battery of tasks.

Chapter 2

Background

In this section, we will first examine the backpropagation through time and real-time recurrent learning algorithms to isolate which features make them unsuitable followed by an exploration of work carried out in current literature and the results achieved to better contextualise the modifications made over the course of this project.

2.1 Backpropagation through Time

The scope of this project pertains to a recurrent neural network (RNN), so the following analysis will be restricted to that architecture. An RNN processes sequences by maintaining a state vector $\mathbf{h}_t$ at each time step t, updated based on the previous state $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$. The equations governing the dynamics are:

$$\mathbf{h}_t = (1 - \frac{1}{\tau})\mathbf{h}_{t-1} + \frac{1}{\tau}\phi_1(\mathbf{W}_{rec}\mathbf{h}_{t-1} + \mathbf{W}_{in}\mathbf{x}_t + \mathbf{b}),$$
$$\hat{\mathbf{y}}_t = \phi_2(\mathbf{W}_{out}\mathbf{h}_t + \mathbf{b}_y)$$

where $\mathbf{h}_t$ is the hidden state or neuron state at time t, $\mathbf{x}_t$ is the input at time t, $\hat{\mathbf{y}}_t$ is the output at time t, $\mathbf{W}_{rec}, \mathbf{W}_{in}, \mathbf{W}_{out}$ are the recurrent, input and output weight matrices respectively, $\mathbf{b}, \mathbf{b}_y$ are the bias vectors, $\phi_1(\cdot)$ and $\phi_2(\cdot)$ are non-linear activation functions (commonly tanh or sigmoid), and τ is the time constant moderating the influence of the previous state on the current state.

Loss Function

Assuming a loss function L defined over a sequence, the total loss for a sequence of length T is given by:

$$L = \sum_{t=1}^{T} L_t(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

where L_t is the loss at time t and $\mathbf{y}_t$ is the target output at time t. For discrete classification tasks, a categorical cross-entropy loss function is generally used while for continuous problems, a mean-squared error function is preferred.

Gradients of the Loss Function

To train the RNN, we compute gradients of the loss function with respect to the parameters by applying the chain rule and proceeding backwards through time:

Gradient of the Output Layer

$$\frac{\partial L}{\partial \mathbf{W}_{out}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{W}_{out}}$$

Using $\frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{W}_{out}} = \mathbf{h}_t$ (and similar for biases), the gradients are accumulated over all time steps.

Gradient of the Hidden State

$$\frac{\partial L}{\partial \mathbf{h}_t} = \frac{\partial L_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} + \frac{\partial L}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}$$

The first term relates to the impact of the hidden state on the output at time t, and the second term propagates the gradient information from future time steps.

Gradient with Respect to Weights

$$\frac{\partial L}{\partial \mathbf{W}_{rec}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{rec}}$$

where $\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{rec}} = \phi_1'(\mathbf{W}_{rec}\mathbf{h}_{t-1} + \mathbf{W}_{in}\mathbf{x}_t + \mathbf{b}) \cdot (\mathbf{h}_{t-1} + \frac{1}{\tau})$, including the derivative of the activation function.

Biological Implausibility

The following aspects of BPTT are biologically implausible:

- **Non-locality**: The gradient equation above includes the entire W_rec matrix to update any given synapse and thus is non-local.
- **Symmetry of weight matrices**: The same weights are used in the forward and backward pass (gradient calculation), which implies a symmetry that doesn't necessarily exist in biological neural networks.
- **Storage of past activations**: The algorithm needs to store all past activations to compute gradients as seen by the summations over t, which is not evident in biological systems.

2.2 Real-time Recurrent Learning

The following is analysis presented by Murray [7]. Noting the implausibility of the storage of past activity, we can rewrite BPTTs update rules to work in an online manner. Importantly, the two rules are mathematically equivalent if done over the same number of time-steps, except that with RTRL we have the ability to update at every time-step.

Derivation of Gradient Equations

The loss function can be expressed as follows when taking implicit dependencies into account:

$$L = L(h^0, \ldots, h^t(W_{\text{rec}} h^{t-1}(W_{\text{rec}} h^{t-2}(\ldots))))$$

We consider the updated formulas for the gradient calculations with respect to the recurrent weights W_{rec}. The gradient of the loss can be written as:

$$\frac{\partial L}{\partial W_{ab}} = \begin{cases} \sum_t \frac{\partial L(\ldots, h^t, \ldots)}{\partial h^t} \cdot \frac{\partial h^t(W_{\text{rec}} h^{t-1}(W_{\text{rec}} h^{t-2}(\ldots)))}{\partial W_{rec}^{ab}} & \text{(RTRL)} \\ \sum_t \frac{\partial L(\ldots, h^t(W_{\text{rec}} h^{t-1}(\ldots)), h^{t+1}(W_{\text{rec}} h^t(\ldots)), \ldots)}{\partial h^t} \cdot \frac{\partial h^t(W_{\text{rec}} h^{t-1})}{\partial W_{rec}^{ab}} & \text{(BPTT)} \end{cases}$$

In RTRL, the first derivative can easily be calculated because we treat the loss as an explicit function of h^t. The dependencies of h^t are then accounted for by the second derivative which must be computed recursively due to the nested W terms. Note how the gradient depends on earlier states of h and so can be calculated in an online manner. In BPTT, the implicit dependencies are in the first derivative which must be calculated recursively in a backward manner. This can be more clearly seen with complete expressions. Assuming our loss function is of the mean squared error form, we can write the above gradients as:

$$\frac{\partial L}{\partial W_{ab}} = \begin{cases} -1T\sum_t \sum_i (\mathbf{W}_{\text{out}}^T e(t))_i p_{ab}^i(t), & \text{(RTRL)} \\ -1\tau T \sum_t z_a(t)(\phi'(u_a(t))h_b^{t-1}, & \text{(BPTT)} \end{cases}$$

where we have defined,

$$p_{ab}^i(t) = \frac{\partial h^t(W_{\text{rec}} h^{t-1}(W_{\text{rec}} h^{t-2}(\ldots)))}{\partial W_{ab}}$$

$$z_i(t) = -\frac{\partial L(\ldots, h^t(W_{\text{rec}} h^{t-1}(W_{\text{rec}} h^{t-2}(\ldots))), \ldots)}{\partial h_i^t}$$

Recursive Relationships for RTRL and BPTT

The updates for the above terms which can be thought of as accumulation of gradients are given as follows by applying the chain rule.

$$p_{ab}^i(t) = (1-\frac{1}{\tau})p_{ab}^i(t-1) + \frac{1}{\tau}\sum_j \phi'(u_i(t-1))W_{\text{rec}}^{ij} p_{ab}^j(t-1) + \frac{1}{\tau}\delta_{ia}\phi'(u_a(t))h_{t-1}^b \tag{2.1}$$

$$z_i(t) = (1-\frac{1}{\tau})z_i(t+1) + \frac{1}{\tau}\sum_j \phi'(u_j(t+1))W_{\text{rec}}^{ji} z_j(t+1) + \frac{1}{\tau}\sum_j W_{\text{out}}^{ji} e_j(t) \tag{2.2}$$

Note that $p_{ab}^i(t)$ is a rank 3 tensor that is computed forward in time hence the practicality of RTRL for online learning. $z_i(t)$ on the other hand is computed backward in time and since no approximations have been made, the two methods are equivalent. For this reason RTRL makes use of non-local information the same as BPTT and the following section provides an algorithm that approximates these algorithms in a more biologically plausible manner.

2.3 Local online learning in recurrent networks with random feedback, Murray

We now consider the Random Feedback Local Online (RFLO) algorithm proposed by Murray [7]. The aim of this paper is similar to that of this project, to propose biologically plausible learning rules. To that end, the author of this paper derives an approximation to gradient based learning that is both local in time and space. This is achieved through two steps. Firstly, locality is enforced by dropping non-local parts of the loss function gradient hence approximating the gradient. Then the feedback weights required by gradient based learning are replaced with fixed random feedback weights. This is achieved by dropping the second term in equation 2.1. This results in the following accumulation term:

$$p_{ab}(t) = (1 - \frac{1}{\tau})p_{ab}(t-1) + \frac{1}{\tau}\phi'(u_a(t))h^b_{t-1} \tag{2.3}$$

Note that $p_{ab}(t)$ is now a rank 2 tensor. A similar treatment can be performed for the accumulation term used for the updates for W_{in}. The result of these approximations can be seen in 2.4-2.7 which provide the gradient updates. The two extra terms p_{ab} and q_{ab} are the accumulated products of pre- and post-synaptic activity at the recurrent and input synapses respectively.

$$\frac{\delta W^{ab}_{\text{out}}(t)}{\delta t} = \eta_1 \varepsilon_a(t) h_b(t) \tag{2.4}$$

$$\frac{\delta W_{ab}(t)}{\delta t} = \eta_2 [B\varepsilon(t)]_a p_{ab}(t) \tag{2.5}$$

$$\frac{\delta W^{ab}_{\text{in}}(t)}{\delta t} = \eta_3 [B\varepsilon(t)]_a q_{ab}(t) \tag{2.6}$$

where η_α are learning rates, B is a random matrix of feedback weights, $p_{ab}(t)$ is as defined in equation 2.3, and $q_{ab}(t)$ are defined as,

$$q_{ab}(t) = -\frac{\phi'(u_a(t))}{\tau} x_b(t-1) + \left(1 - \frac{1}{\tau}\right) q_{ab}(t-1) \tag{2.7}$$

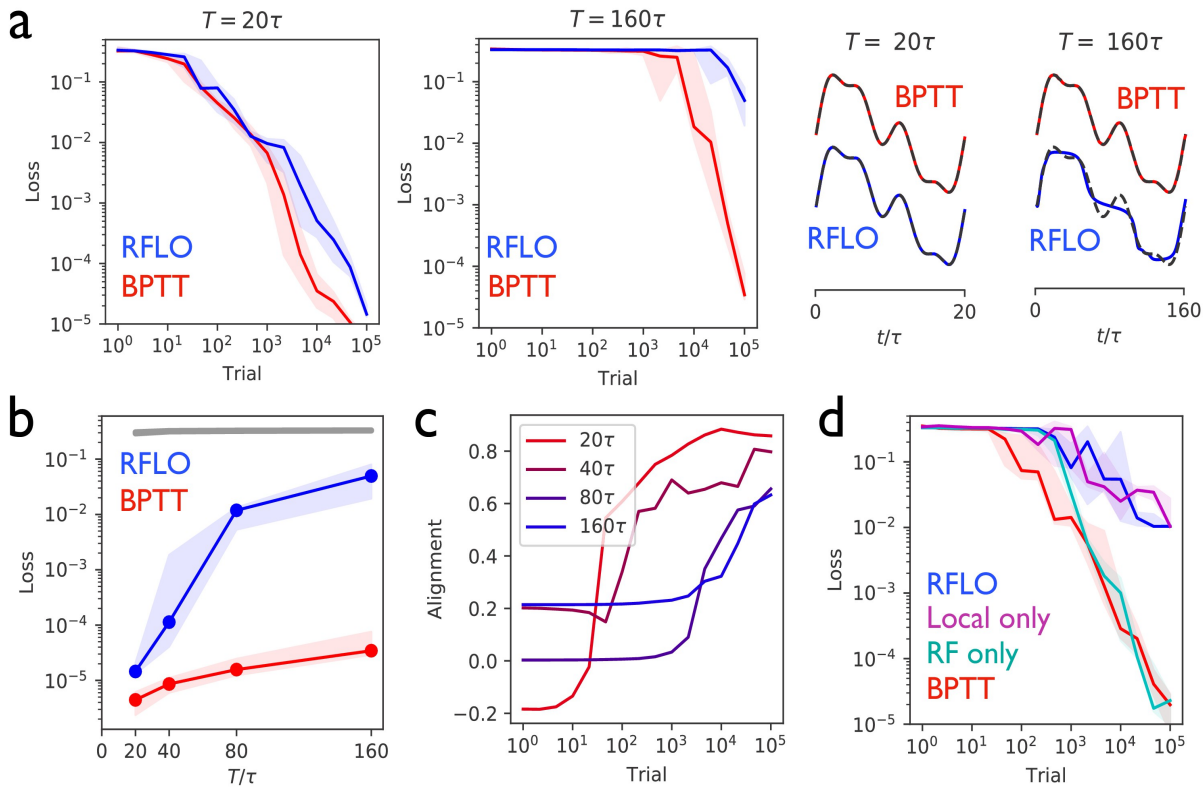


Fig. 2.1 *Local online learning in recurrent networks with random feedback*, Murray [7]. Sub-figure (a) demonstrates the loss of BPTT vs RFLO during training on a task to output a one dimensional periodic output. The right panel shows the fit of the produced signal versus the desired one. Sub-figure (b) shows the loss for the two learning algorithms as the period of the desired output increases compared to the RNN time constant τ. Sub-figure (d) shows the loss for the intermediate modifications of the RFLO algorithm, where only random feedback or locality is enforced.

Having established the RFLO update rules, we can examine the results in Figure 2.1. From the upper-left and -center panels it can be seen that for tasks of small multiples of the RNN time constant, RFLO is competitive with BPTT. However once the task moves to longer time scales, it is unable to learn sufficiently well likely due to the accumulation of errors in the gradient approximations, a problem not faced by BPTT. The lower left panel shows the extent of discrepancy over a range of time constant multiples further highlighting the difference between the two learning algorithms. Lastly, the lower left panel illustrates that the local only enforcement is responsible for the performance difference between RFLO and BPTT. The random feedback modification is still able to perform nearly as well as BPTT as mentioned by Lillicrap et al. [3]

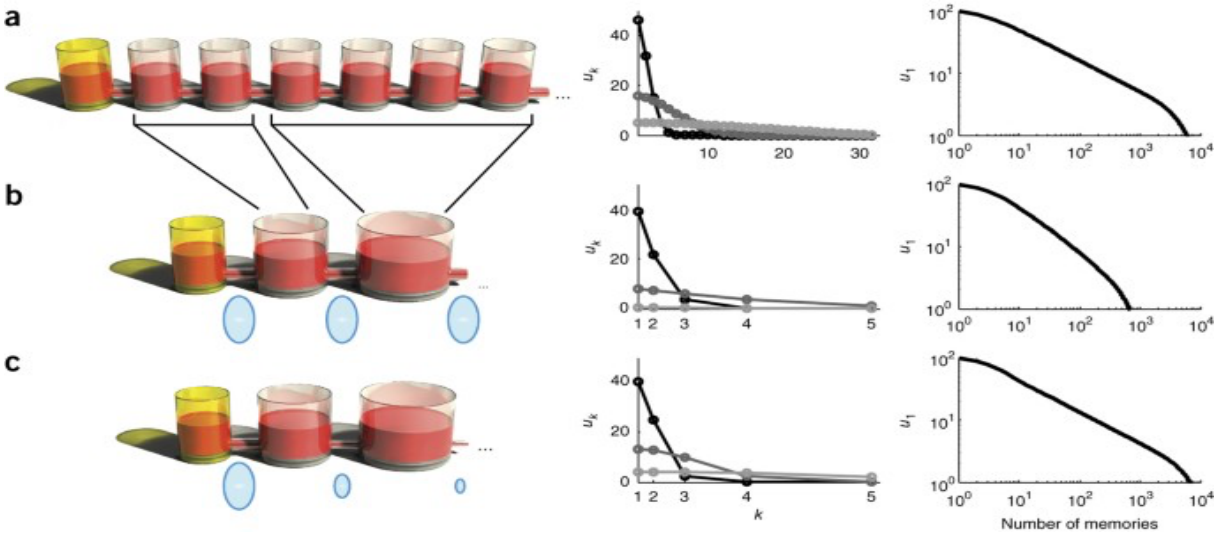
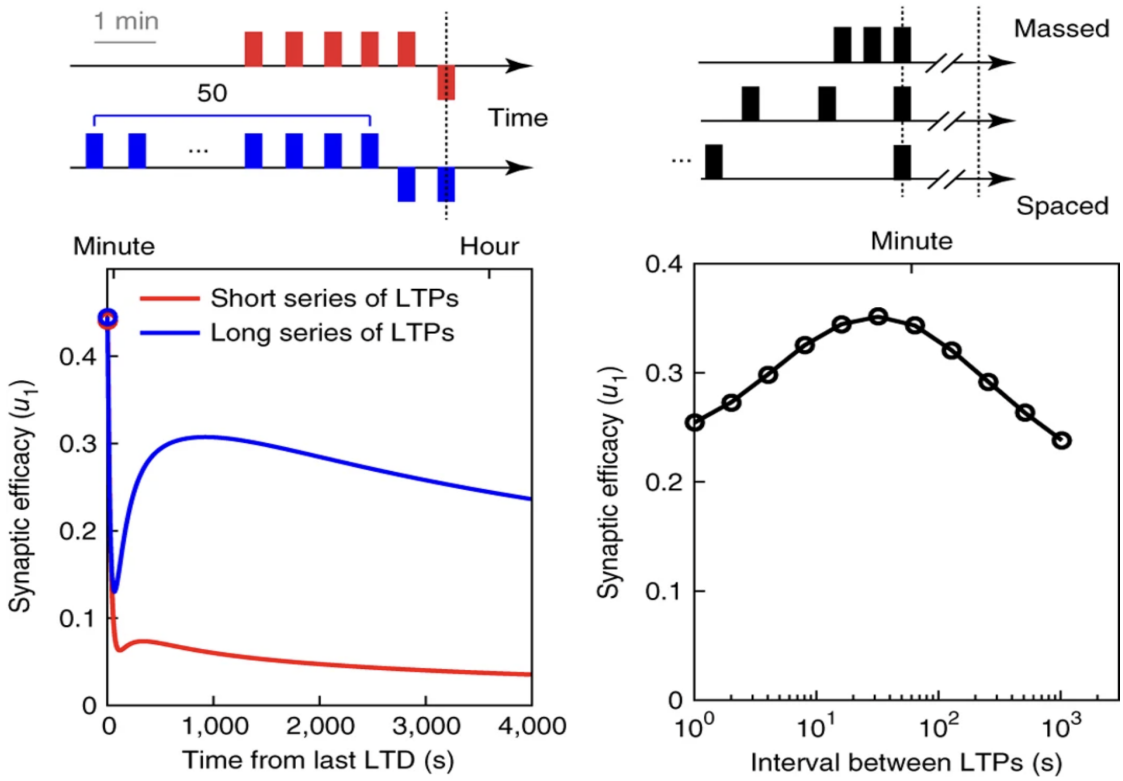


Fig. 2.2 *Computational principles of synaptic memory consolidation*, Benna and Fusi [2]. Beaker Representation of the dynamical variables in the model. Sub-figure (a) has equally sized beakers with equal cross-section between connections. Sub-figure (b) has merged beakers such that we receive progressively larger beakers that is, $C_k = 2C_{k-1}$. Sub-figure (c) is the same as (b) but now the cross-sections of the connections are inversely scaled with beaker size that is, $g_{k,k+1} = \frac{1}{2} g_{k-1,k}$.

2.4 Computational principles of synaptic memory consolidation, Benna and Fusi

The authors of [2] were inspired by the problem of memory consolidation in biological systems. In order to understand the computational principles they created a synaptic model that was able to preserve memories by preventing them from being overwritten. This was achieved by using multiple dynamical processes operating on different timescales. Initially the memories would be stored in the fast variables and then gradually transferred to slower variables. The connections between these dynamical variables are bidirectional and properties of biological memory such as delayed expression of synaptic modification, meta-plasticity and spacing effects emerge. In its simplest form, the model contains a chain of dynamical variables u_k which represent the different biochemical processes responsible for memory consolidation. Each variable only interacts with its neighbours according to the following differential equation:

$$C_k \frac{du_k}{dt} = g_{k-1,k}(u_{k-1} - u_k) + g_{k,k+1}(u_{k+1} - u_k) \tag{2.8}$$



(a) Plot of synaptic efficacy for network with 5 LTP events followed by an LTD event, and 50 LTP events followed by 2 LTD events.

(b) Synaptic efficacy plotted 100s after after the end of 3 LTP events plotted against the interval between LTPs.

Fig. 2.3 Effect of long term potentiation (LTP) and depression (LTD), and spacing of inputs on synaptic efficacy. [2]

C_k and $g_{k,k+1}$ are parameters that determine the time-scale on which these variables operate. Figure 2.2 shows 3 different versions of this model using a beaker analogy. C_k represents the area of the beaker and $g_{k,k+1}$ represents the cross-section of the connections between the beakers. The middle plots of the figure are the levels of u_k at 3 different times indicating how fast these variables converge to equilibrium and is thus a measure of the persistence of information in these variables. It can be seen that merging beakers results in faster convergence which is undesirable, but by reducing the cross-section of connections, we can retain the persistence of model (a) and thus the memory storage capacity all while using much fewer dynamical variables. This is an important result that we use later for our own model.

Once this chain model is generalised so arbitrary connections of the variables can be constructed, it is able to reproduce various experimental observations of biological memory systems. This can be seen from Figure 2.3 which shows the occurrence of meta-plasticity and spacing effects.

The plot in Figure 2.3a shows that the depression in synaptic efficacy of the model experiencing 50 LTP events is more transient than that of the model with 5 LTP events despite the former also experiencing 2 LTD events. This is indicative of the model's ability to model long term relationships while being resilient to short term noise, another useful property for the goal of this project. In Figure 2.3b, we can observe the emergence of spacing effects under this model. The synaptic efficacy is measured an arbitrary time after 3 LTP events with a constant interval between the events. This is plotted against varying lengths of the interval and an inverted U-shape is observed due to the fact that shorter intervals likely lead to the saturation of dynamical variables and longer intervals allow the variables to return to equilibrium. Having understood the merits of a complex model of synapses, we now turn to the topic of plasticity rules that we could provide to our learning algorithm.

2.5 Meta-learning biologically plausible plasticity rules, Shervani-Tabar and Rosenbaum

Noting the biological implausibility of backpropagation, the authors of this paper propose a meta-learning approach to develop an interpretable, biologically plausible set of learning rules. A major limitation of backpropagation is the need for symmetric connectivity for propagation of gradient to earlier time-steps or in the case of this paper, earlier layers as they work with a multi-layer artificial neural network. This is known as the weight transport problem and Lillicrap et al. [3] shows that random backward connections can sufficiently transmit teaching signals. However, this comes with the drawback that dropping symmetry in this manner makes it more difficult to train deeper networks or ones with online data streams as shown by Amit [1]. In the context of RNNs this issue would manifest as struggling to learn for long sequences.

To solve this issue, the authors employ a meta-learning approach optimise the parameters over some candidate local plasticity rules. The goal is to develop general rules that are the same for all weights in the network rather than each weight having its own unique plasticity rules. Algorithm 1 shows the pseudo-code to accomplish this task. There is a two level learning scheme employed in this pseudo-code. The inner loop uses the plasticity rule $F(\Theta)$ that learns the parameters $\mathbf{W}$ of

Algorithm 1 Meta-learning algorithm.

Require: Meta-training set $\{T_\varepsilon\}_{0\leq\varepsilon\leq E} = \{(X_\varepsilon^{\text{train}}, Y_\varepsilon^{\text{train}}), (X_\varepsilon^{\text{query}}, Y_\varepsilon^{\text{query}})\}$, plasticity rule F, number of episodes E, meta-learning rate η, and regularization coefficient λ.

Ensure: Initialize learning parameters $\Theta^{(0)}$.

for $\varepsilon = 0, \ldots, E$ **do**
  Initialize network parameters $W^{(0)}$ and B.
  for $(x^{(i)}, y^{(i)}) \in (X_\varepsilon^{\text{train}}, Y_\varepsilon^{\text{train}})$ **do**
    Set $Y_0 = x_{\text{train}}^{(i)}$.
    for $\ell = 1, \ldots, L$ **do**
      Compute Z_ℓ
      Compute Y_ℓ
    end for
    Compute $C(Y_L, y_{\text{train}}^{(i)})$.
    Compute $e_L = \frac{\partial C}{\partial Z_L}$.
    for $\ell = L, \ldots, 1$ **do**
      Compute $e_{\ell-1} = B_{\ell,e} e_\ell \sigma'(Z_{\ell-1})$
      Update $W_{\ell-1,\varepsilon} = W_{\ell-1,\varepsilon} + F(e_{\ell-1}, Y_{\ell-1}, e_\ell, Y_\ell, W_{\ell-1,\varepsilon}, \Theta_\varepsilon)$.
    end for
  end for
  Update meta-parameters $\Theta^{(\varepsilon+1)} = \Theta^{(\varepsilon)} - \eta \nabla_\Theta \left[C\left((X_\varepsilon^{\text{query}}, Y_\varepsilon^{\text{query}}); W^{(\varepsilon+1)} \right) + \lambda \|\Theta^{(\varepsilon)}\|_1 \right]$.
end for

a model $f_{\mathbf{W}}$. Then an outer loop runs over all tasks in the set $\{T_\varepsilon\}$ where the loss over a query or test set is evaluated from which the meta-parameters are updated using gradient descent. L1 regularisation is also employed so that the final learning rule has fewer terms simplifying its interpretability.

In Figure 2.4 we can observe the result of this training optimisation. This network is trained on a 5-way classification task from the EMNIST dataset using a 3-layer fully connected neural network. The top plot clearly demonstrates that feedback alignment struggles at minimising the loss over this task as stated by Amit [1]. Meanwhile the pool of plasticity rules demonstrates performance similar to backpropagation albeit with slower convergence.

For reference the candidate learning terms in in F^{pool} are presented in equations 2.10-2.19 in Figure 2.5. The plasticity update is then calculated as a linear combination of these learning rules,

$$F(\Theta) = \sum_{i=0}^{R-1} \theta_i F^i \tag{2.9}$$

where θ are the strengths of these signals which we are optimising over with L1 regularisation. We will later adapt these to work with an RNN instead. It is worth noting that in their analy-

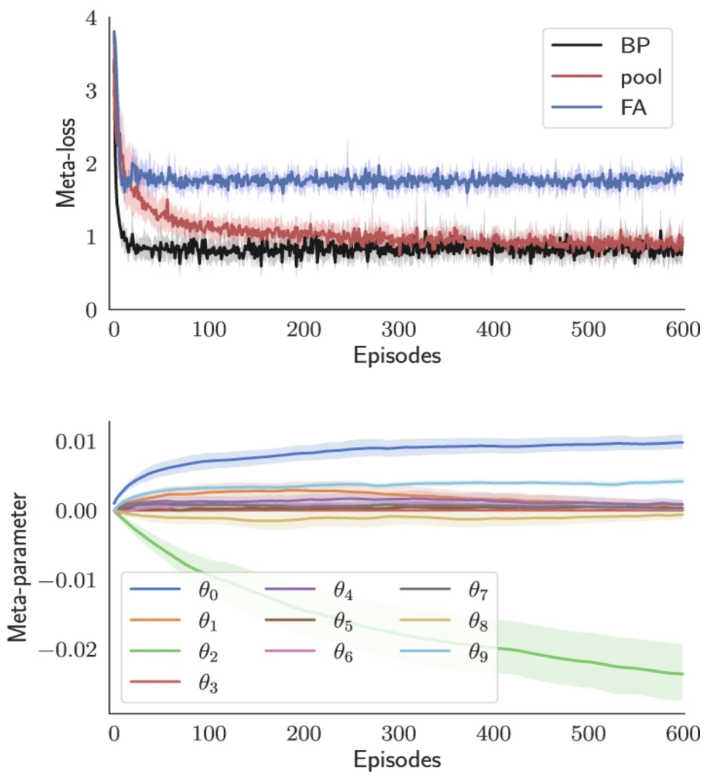


Fig. 2.4 *Meta-learning biologically plausible plasticity rules with random feedback pathways*, Shervani-Tabar and Rosenbaum [8]. Performance of a classifier network trained with a pool of plasticity rules, backpropagation and feedback alignment. The top figure plots the meta-loss and the bottom figure shows the convergence of the plasticity coefficients.

sis, Shervani-Tabar and Rosenbaum [8] found that all but F^0, F^2, and F^9 converged to zero. These remaining terms are a pseudo-gradient rule, Hebbian-like plasticity rule, and Oja's rule respectively. However, for our model we will begin with all of these terms and optimise for their weights ourselves.

$$F^0 = -e_\ell Y_{\ell-1} \tag{2.10}$$
$$F^1 = -Y_\ell e_{\ell-1}^T \tag{2.11}$$
$$F^2 = -e_\ell e_{\ell-1}^T \tag{2.12}$$
$$F^3 = -W_{\ell-1,\ell} \tag{2.13}$$
$$F^4 = -1_\ell e_{\ell-1}^T \tag{2.14}$$
$$F^5 = -e_\ell 1_\ell Y_\ell Y_{\ell-1}^T \tag{2.15}$$
$$F^6 = -Y_\ell Y_\ell^T W_{\ell-1,\ell} e_{\ell-1} e_{\ell-1}^T \tag{2.16}$$
$$F^7 = -e_\ell Y_\ell^T W_{\ell-1,\ell} e_{\ell-1} Y_{\ell-1}^T \tag{2.17}$$
$$F^8 = -Y_\ell Y_{\ell-1}^T W_{\ell-1,\ell}^T e_\ell e_{\ell-1}^T \tag{2.18}$$
$$F^9 = (Y_\ell Y_{\ell-1}^T - (Y_\ell Y_\ell^T) W_{\ell-1,\ell}) \tag{2.19}$$

Fig. 2.5 Candidate plasticity rules from Shervani-Tabar and Rosenbaum [8].

Chapter 3

Methods

Having explored relevant work in literature, we can now begin describing the various approaches taken over the course of this project. The following model is the final architecture that was evaluated however earlier modifications to it are presented in later subsections. The tasks that will be used to train the model are detailed later, the following is a brief explanation of the idea. We aim to train parameters associated with our complex synapses. These parameters will be shared across all synapses with independent states. To train them we generate a battery of tasks and evaluate the loss over all predictions in the test set of a single task as the loss for one task. Then we can obtain the loss over all tasks and minimise the average of this loss.

3.1 Model Architecture

We begin with a slight modification of the standard RNN equations.

$$r_{s,0} = \mathbf{0} \tag{3.1}$$

$$r_{s,t+1} = r_{s,t} + \frac{1}{\tau}\left(-r_{s,t} + \phi_1\left(W_s^{rec} r_{s,t} + W_s^{in} x_t\right)\right) \tag{3.2}$$

$$\hat{y_{s,t}} = \phi_2(W_s^{out} r_t) \tag{3.3}$$

where: $r_t \in \mathbb{R}^n$ is the hidden state with n neurons, x_t is the input at time t. $W_s^{in} \in \mathbb{R}^{1\times n}$, $W_s^{rec} \in \mathbb{R}^{n\times n}$, and $W_s^{out} \in \mathbb{R}^{n\times m}$, where m is the number of classes for prediction or features in the case of a continuous problem, are the weight matrices for the input, recurrent, and output connections.

τ is the length scale parameter and ϕ_2 is an activation function (softmax for classification and tanh for continuous problems) applied to the hidden state to produce the output $\hat{y}_t \in \mathbb{R}^m$. Observe that there is an extra index s which corresponds to the sequence index, our weight tensors are dependent on previous sequences through the relationship presented in the following equations. We now introduce another smaller synaptic RNN which for the sake of clarity will be referred to as an RCN. There will be a unique occurrence of this RCN present in each synapse with its own unique state, however, the parameters for these RCNs will be shared. The RCNs will govern synaptic dynamics via the following equations.

$$h_0 = \mathbf{0} \tag{3.4}$$

$$h_{sT+1} = h_{sT} \tag{3.5}$$

$$h_{sT+t+1} = (1-z)h_{sT+t} + z\tanh(F_t(\Theta)_t P_{\text{in}} + h_{sT+t}P_{\text{rec}} + b), 1 \leq t < T \tag{3.6}$$

$$W_{s+1} = W_s + \eta \tanh(h_{s(T+1)}P_{out} + b_{out}) \tag{3.7}$$

Here P_{in}, P_{rec}, and P_{out} represent the weights of the RCN, b, b_{out} are the bias vectors, η is the learning rate, and h_{sT+t} is the vector representing the states of the dynamical variables during the tth time step of the $(s+1)^{th}$ sequence. Note the weights matrices are now updated at the end of every image by the synapses and also that unlike the RNN states the RCNs retain their state information from sequence to sequence. It is in effect this feature that is intended to enable the RCNs to have memory of past information thus enabling learning as they are effectively being tasked with calculating the RNN gradients. Furthermore z is similar to the time constant τ from equation 3.2 except it is a vector of the same length as h_{sT+t} and is evaluated as an element-wise product with h_t. It is constructed as follows:

$$z_0 = k \tag{3.8}$$

$$z_i = \alpha^i z_0 \tag{3.9}$$

This then allows the notion of different time scales to be imposed on the dynamical variables as introduced by Benna and Fusi [2]. In the final model these were fixed such that $z_P^{-}1 \simeq ST$ that is

python notation? :-))
z[-1] = ST

the slowest dynamical variables changes on the time scale of the length of a full task.

Now we define the learning signals that provide the input to the RCNs. These were inspired by Shervani-Tabar and Rosenbaum [8] and Murray [7]. They are computed at each time-step and are a function of pre- and post-synaptic activity as well as a global error signal.

$$F_t(\Theta) = \sum_{i=0}^{10} \theta_i \cdot F_t^i \tag{3.10}$$

where $F(\Theta)$ is the input for the RCNs and F_i are defined as:

$$F^0 = -e_t r_{pre}^t \tag{3.11}$$

$$F^1 = -r_{post}^t e_t^T \tag{3.12}$$

$$F^2 = -e_t e_t^T \tag{3.13}$$

$$F^3 = -W_{t-1} e_t \tag{3.14}$$

$$F^4 = -e_t^T \tag{3.15}$$

$$F^5 = -e_t r_{post} r_{pre}^T \tag{3.16}$$

$$F^6 = -r_{post} r_{post}^T W_{t-1} e_t e_t^T \tag{3.17}$$

$$F^7 = -e_t r_{post}^T W_{t-1} e_t r_{pre}^T \tag{3.18}$$

$$F^8 = -r_{post} r_{pre}^T W_{t-1}^T e_t e_t^T \tag{3.19}$$

$$F^9 = (r_{post} r_{pre}^T - (r_{post} Y_\ell^T) W_{t-1}) \tag{3.20}$$

$$F_{in}^{10} = \eta_0 (\mathbf{B}\varepsilon^T)^T \cdot q_t \tag{3.21}$$

$$F_{rec}^{10} = \eta_1 (\mathbf{B}\varepsilon^T)^T \cdot p_t \tag{3.22}$$

$$F_{out}^{10} = \eta_2 \varepsilon \cdot r_{\text{post}}^{(t)} \tag{3.23}$$

The trainable parameters for the network are: P_{in}, P_{rec}, P_{out}, and θ. With these modifications our network has a runtime complexity of $O(N^2P^2)$ where N is the number of neurons and P is the number of dynamical variables. This contrasts with RTRL which has a complexity of $O(N^4)$. The smaller value of P means that our version of the network will be a lot less computationally demanding.

Algorithm 2 Generate Training Data

```
Require: Network initialized with tasks data
function GENERATE_TRAINING_DATA(self)
    Initialize empty tensors to store weights, learning signals and targets
    for each task τ_i in task set 𝒯 do
        Initialize RNN_BPTT and its optimiser
        for each epoch in num_epochs do
            Initialize RFLO accumulataion terms, p, q
            for each x_train, y_train in τ_{i,train} do
                Forward pass through x_train
                Collect outputs and hidden states at each timestep
                for each timestep in x_train do
                    Calculate input and hidden learning signals
                end for
                Calculate loss and backpropagate
                Collect weights and gradients (targets)
            end for
            Accumulate weights, learning signals, and targets
        end for
    end for
    return weights, learning signals and targets
end function
```

3.2 Meta-Learning Approaches

3.2.1 Supervised Meta-Learning of BPTT Gradients

With the forward pass logic established, we now turn our attention towards the meta-learning process. We considered the possibility of generating training data for our synapses via the use of BPTT effectively making this a supervised meta-learning approach. Since our RCNs were intended to model the learning that would have taken place under BPTT, we could then train an RNN using gradients generated by BPTT and using state information from the RNN to record the learning signals our synapses would have received as well as the gradients that BPTT evaluates which forms the target tensor for our problem. This approach yielded some advantages in particular due to implementation as separating the training of these RCNs meant that we can effectively ignore running the network RNN and thus storing all values of the tensors associated with it. This led to much lower memory requirements and also more flexibility in the batching of synapse training.

Algorithm 2 shows the procedure for creating our dataset for supervised meta-learning. The method involves training an RNN using BPTT and accumulating the learning signals and gradients generated from backpropagation after each sequence. There are a few meaningful changes that have been made here compared to our earlier approach. The most significant is the frequency of updates. The previous models were all capable of making completely online updates that is at each time step of a sequence. Due to the need to generate gradient information using BPTT, we have modified this update frequency to be at the end of each sequence. While this is not desirable it was necessary from a practicality perspective. Furthermore, we now also provide information about the weight associated with the current synapse. The intention of this is so that the RCNs would be able to learn some negative feedback behavior to prevent weights from becoming too large.

The following is an explanation of the shapes of the gradient training data as well as the batching strategy employed. Our tasks data contains τ tasks, each task contains M training sequences which are each of length T. Each of these points generates a learning signal at every timestep and a gradient target every T timesteps. Furthermore, every synapse in the RNN, quantified by N, generates an input, output pair. Thus at each epoch we generate an input tensor of shape (τ, N, M, T) and target tensor of shape (τ, N, M). However, the RNN is trained via BPTT for many epochs denoted by k. Thus we end up with tensors of shapes (τ, N, Mk, T), and (τ, N, Mk). We can then sub-sample tasks and synapses that is select tensor of shape (τ_B, N_B, Mk, T) randomly, where $\tau_B \leq \tau$ and $N_B \leq N$. It is important to note that Mk should not be sub-sampled randomly since we want to preserve the ordering of sequences in epochs as the RCNs may learn dependencies that evolve over epochs. Algorithm 3 shows the method of determining the loss over one task. The training process is similar to algorithm 5 except the loss over all tasks is calculated for each gradient step that is we use the full batch.

Algorithm 3 Function to obtain loss over a task.

1: **for** each synapse $n \in \mathscr{N}$ **do**
2: Run RCN dynamics on learning signals over all sequences and all epochs.
3: Predict $\nabla \hat{W}_{s+1}$ from $h_{(s+1)T}, \forall s \in \mathscr{S}$.
4: $L_\tau \leftarrow \frac{1}{N} MSE(\nabla \hat{\mathbf{W}}, \nabla \mathbf{W})$
5: **end for**

Algorithm 4 Function calculate_loss(τ) for obtaining loss over a task.

1: Reset RNN weights W_t
2: Reset synapse states R
3: **for** each (x, y) sampled from training set for τ **do**
4: Reset neuron states H
5: **for** each time step t in length of input x **do**
6: Update RNN for time t using x_t
7: Update synapse states R based on $f(h^t_{pre}, h^t_{post}, e_t, x_t)$
8: Update network weights: $W_{t+1} \leftarrow W_t + tanh(R_{t+1}P_{out})$
9: **end for**
10: **end for**
11: **for** each (x, y) sampled from test set for τ **do**
12: Reset synapse states R
13: **for** each time step t in length of input x **do**
14: Update RNN for time t using x_t
15: Predict $\hat{y}_t$ using softmax
16: $L_x = L_x + \frac{1}{T}$ Loss$(\hat{y}_t, y)$
17: **end for**
18: $L_{task} = L_{task} + \frac{1}{K} L_x$
19: **end for**

3.2.2 Unsupervised Meta-Learning using BPTT

The previous approach was developed due to the an inability to achieve a meaningfully optimised model via our earlier methods. For the sake of completeness, the following subsections describe those approaches. The first attempt involved using an unsupervised version of the problem above. The RCNs are tasked with implicitly determining the gradients for their corresponding RNN weights. Algorithm 4 describes the procedure to determine the loss over one task. Note that in earlier versions of the network lines 1 and 8 were not present, W_{t+1} was simply the readout of the synaptic states followed by an activation function. We later established that the method presented in algorithm 4 was more stable with better performance. The addition of line 1 was so that the learning algorithm would be agnostic to starting states. Determining the optimal initialisation of weights was not the aim of this project and this method enables the learning process to be more general.

Note the presence of two loops, the first is where the network is allowed to run on training examples to align its RNN weights. No loss is computed in this first loop, it is analogous to the fine-tuning of weights using BPTT. After this, the network is run on a separate 'test' set where we accumulate the loss of the predictions at each time step for every sequence. In earlier versions of

Algorithm 5 Training Procedure for Synaptic dynamics.

1: **for** each task τ_i in task set $\mathscr{T}$ **do**
2: L = calculate_loss(τ_i)
3: Calculate gradients $\nabla_\theta L$ using BPTT
4: Perform gradient descent on parameters associated with B: $\theta \leftarrow \theta + \nabla_\theta L$
5: **end for**
6: Return trained parameter matrix θ.

this network, the weights would be updated while running on the test set, but we determined that by freezing the weights during this process, the network was more stable. Having determined the method to calculate the loss over one task, algorithm 5 describes the method for updating our trainable parameters.

It is also worth noting that we train this network via BPTT and due to the need to store past activity over one full task, this process becomes very memory intensive. Given that this is a meta-learning problem, the loss over one task is technically a single training data-point for our synaptic parameters. Thus even with a batch size of one data-point, we encountered difficulty with speed and storage due to the size of the sequences, this is further discussed in a later section on task design. Another issue encountered with this training method is that we would end up in local optima for parameters, the network would predict a uniform distribution for all classes in the case of discrete classification and for continuous function prediction tasks it would saturate the tanh function or predict a straight line. We briefly considered using forward mode differentiation to work around the issue of storing long memory sequences however this was quite impractical and slow so it was abandoned shortly.

3.3 Optimisation Approaches

Given the memory and time complexity of training the network via gradient based methods and noting the small quantity of trainable parameters, ≈ 200, we explored the possibility of optimising with non-gradient methods. Intuitively, this approach is reasonable due to the local optimum issue faced by the previous network, the non gradient methods would be able to explore a larger space of this function and it was simple to have multiple processes running independently exploring many solutions in parallel.

3.3.1 NGOpt by Meta

Two methods were evaluated, the first was a python library called nevergrad which was a toolbox containing multiple optimisers. We chose to employ a meta-optimiser called NGOpt which can dynamically select a variety of optimisation strategies based on the problem details using a heuristic that was determined by benchmarking optimisers across a variety of problems. Unfortunately this algorithm is constantly being modified so an updated version of design is unavailable, Figure 3.1 shows the design from the original paper that this meta-optimiser was based on.

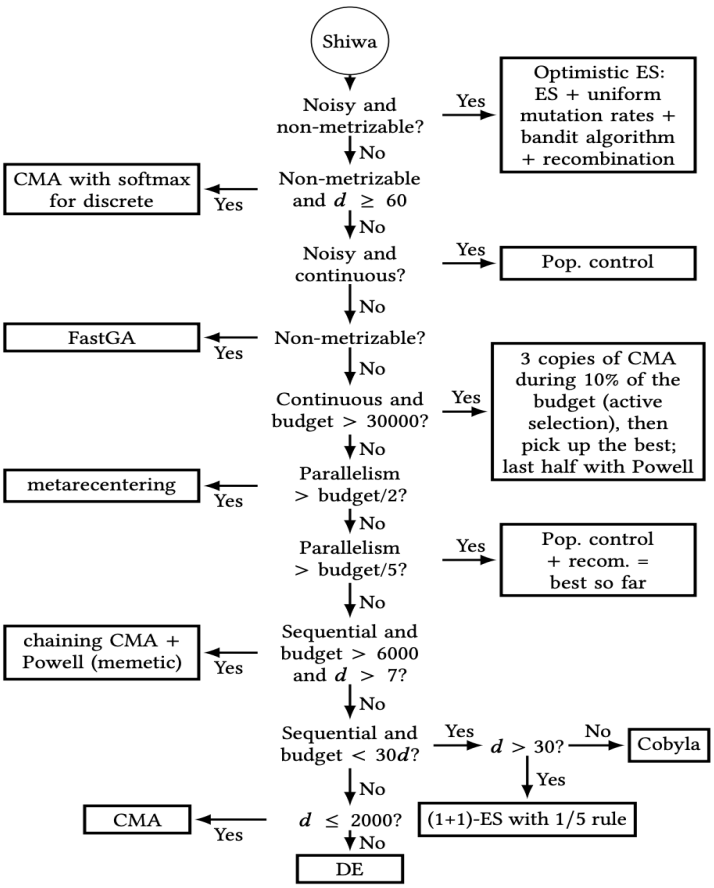


Fig. 3.1 *Versatile Black-Box Optimization*, Liu et al. [6]. Features of the optimisation problem considered in determining which algorithm to use.

3.3.2 Evolutionary Strategies

Lack of success with the NGOpt method motivated the use of evolutionary strategies. The idea behind this algorithm is to initialise a population of 'individuals' which correspond to a set

of parameters that are a proposed solution for the problem. The fitness of the population is evaluated corresponding to determining their loss on some test set. From this population, parents are selected based on which the children will be created. These parents will have a random probability to undergo mutation and crossover. Mutation in this case results in Gaussian noise being added to the parameter thereby encouraging the local exploration of parameter space. Crossover involves swapping parameters between two randomly selected individuals. Note that for our parameter matrices P, this involves element-wise swapping of individual entries. This operation enables the algorithm to escape local minima by performing larger jumps in the parameter space. Algorithm 6 demonstrates the method used for this optimisation algorithm. Due to the low memory requirements and highly parallelisable nature of this algorithm, we were able to explore the parameter space significantly but the random nature of it induced inconsistent results where the optimisation would be promising on occasion which also made it difficult to select appropriate hyper-parameters for it.

Algorithm 6 Pseudo-code for Mutate, Crossover and Evolve functions.

```
function MUTATE(individual)
    for each attribute in individual do
        if random < mutation probability then
            Add Gaussian noise to attribute
        end if
    end for
    return mutated individual
end function

function CROSSOVER(ind1, ind2)
    Initialize children from ind1 and ind2
    for each attribute in ind1 do
        Perform element-wise crossover probabilistically
    end for
    for non-crossover attributes do
        Swap entire attribute probabilistically
    end for
    return children
end function

procedure OPTIMIZEMODEL(model, num_generations)
    Initialize population using Xavier uniform.
    for generation = 1 to num_generations do
        Select parents from population
        Generate offspring via crossover and mutation
        Evaluate fitness of each individual
        Update population based on fitness
    end for
end procedure
```

3.4 Discussion on Rationale and Biological Plausibility of Model

The approach of incorporating complex synapses is motivated by the positive results seen in the work of Benna and Fusi [2]. To our knowledge, this method has not been adopted for the purpose of finding a biologically plausible learning algorithm. This approach appears promising due to the fact that unlike BPTT we would not need to store history of activity, it would in theory keep track of local activity through each synapses state information. Ideally the reduced view of the activity of the entire RNN seen through the 'window' of pre- and post-synaptic activity would

provide sufficient information to in principle recover the global dynamical state of the RNN. Regarding the biological plausibility of these RCNs, we do not possess full knowledge of the structure of molecular networks inside synapses but we do know that different molecular concentrations change on different time-scales. An example of this is the phosphorylation of AMPA receptors which is a process that occurs on the order of seconds to minutes [5]. This process is responsible for modulating the synaptic strength in response to activity on a fast time scale. A particular occurrence of this is during high frequency stimulation, AMPA receptors are phosphorylated by CaMKII (Ca^{2+}/calmodulin-dependent protein kinase II) thereby increasing the probability of these channels opening thus raising synaptic efficacy [5]. On the opposite end is CREB (cAMP response element-binding protein) which is a transcription factor that operates on time scales of hours to days [9]. It is responsible for long term neuronal plasticity and memory maintenance via the regulation of gene expression in response to neuronal activity [9]. Thus through these two examples we can observe the role of multiple time-scales in synaptic plasticity which provides the basis for the time scales we impose on the synapses in our computational model. However, directly mapping our dynamical variables to specific bio-molecular processes would be difficult given our incomplete knowledge of these processes [2]. As a result we have elected to adopt a toy model view here and employ complex synapses as a generic dynamical system which can be approximated by a modified RNN model.

3.5 Sequential Task Creation

Now that we have established the various model architectures and optimisation techniques used over the course of the project, we shall describe the datasets used to train the models. Given that we are meta-learning our parameters, we require a dataset consisting of multiple tasks. Two sets of tasks were considered. The first was a discrete image classification problem inspired by that of Shervani-Tabar and Rosenbaum [8]. The second dataset that was ultimately used for later models consisted of generating synthetic continuous data from a Gaussian Process.

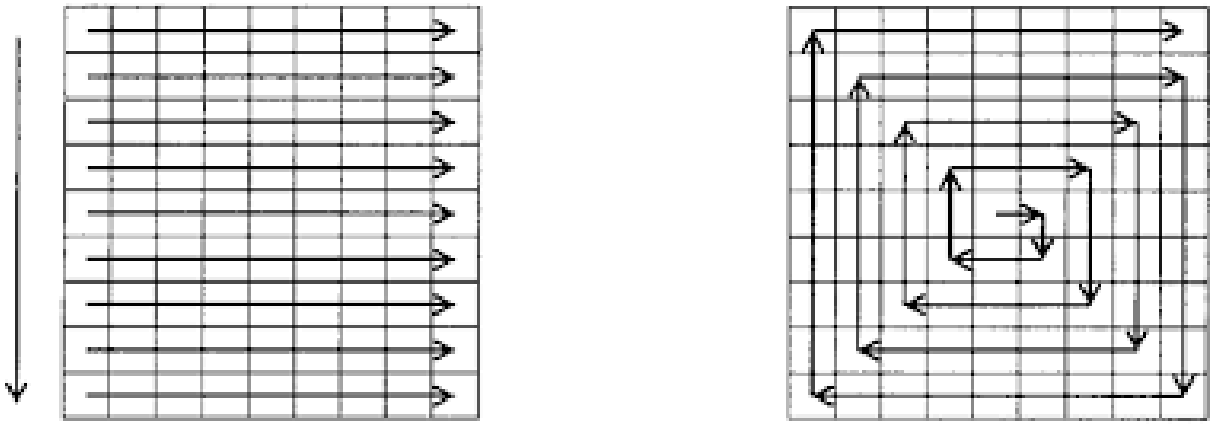


Fig. 3.2 Paths illustrating the two possible sequentalisation strategies considered. Figure courtesy of Smith et al. [10].

3.5.1 EMNIST Dataset

The first iterations of the model were intended to be trained on tasks created via sub-sampling the EMNIST dataset which contains 47 labelled classes of images. We would create a task as follows, L classes would randomly be selected from the 47 available in the EMNIST dataset. Then we create a balanced dataset by finding image-label pairs that are not present in any other task since multiple tasks can have some overlapping classes. This task is then split into train, test, and validation sub-sets. Using this we can easily create many different tasks, for our model we kept the quantity of tasks in the range of 2-10 tasks.

Having created the tasks there was still the issue of how to provide the images as an input to our network. The images were of shape 28x28 and so we elected to unroll them into a vector resulting a sequence of length $T = 784$. The method of unrolling required some consideration as that would impact the difficulty of the task. The simplest approach would simply be to flatten the image which would append rows to the end of the previous one. This however would create discontinuity in the data as we jump across the image. Another approach considered was a spiral unfolding pattern, this would unroll the image from outside towards the center in a spiral fashion and was ultimately the approach selected for sequentalising the image. However, this method also had a drawback, given that the informative data is in the center of the images, most of the predictive data would be received by the network near the end of the sequence while we were penalising it for incorrect predictions from the start of the sequence. A possible solution to this

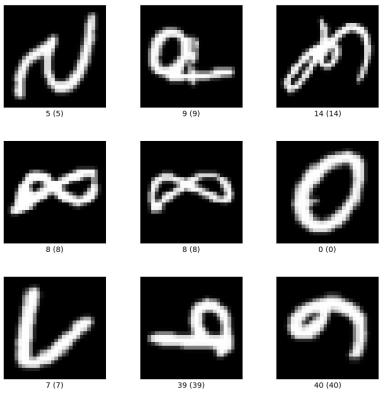


Fig. 3.3 Nine sample images contained in the EMNIST dataset. Figure courtesy of TensorFlow [11].

is to simply reverse the direction of the spiral but this was not explored due to problems with this dataset as detailed in the following section.

Drawbacks

As mentioned in the model architecture section, the meta-learning approach meant that the smallest possible gradient step required running the network on one full task. For BPTT, this meant storing history for MT timesteps where $M \approx 20$ is the number of images and $T = 784$ is the length of the sequence. Furthermore at each step we needed to store data for each synapse which scales as $O(N^2P)$ where N is the number of neurons and P is the number of dynamical variables in the RCNs. As a result training the network even with stochastic gradient descent was very computationally expensive and rendered this approach impractical. Another issue with discrete classification using our network was discovered when training this network on smaller synthetic tasks. Those tasks were created by uniformly sampling in the range $[10(i-1), 10i$ for task i and dividing this interval into L sub-intervals corresponding to the L classes. The label of the data would be the lth sub-interval if the mean of the input data belonged in the lth sub interval. Training on these tasks showed that the network would preferentially predict a uniform distribution, a very common local optimum likely due to the fact the network was being penalised for incorrect predictions early on despite not having sufficient information for correct classification. The aim of predicting at every time-step was to encourage predicting correct

labels earlier however it resulted in an undesirable local optimum. This issue would have been prevalent even in the full EMNIST task set. As a result, we chose to re-purpose the network and tasks to a continuous function approximation problem.

3.5.2 Gaussian Process Tasks

The need for multiple continuous function approximation tasks led to the use of Gaussian processes where we sampling from the prior to generate a sequence. Algorithm 7 shows the procedure to generate this data. We first generate a Fourier series by sampling its parameters from a uniform distribution. Using this Fourier series we then sample values on the x-axis followed by sampling the corresponding function values from the Gaussian Process prior. This sampling from Fourier series is needed so that we can generate unique sequences or 'images' corresponding to the same Gaussian Process. It also adds some difficulty to the tasks since the network will see values in an unsorted but still deterministic order. It also provided some control over the sampling statistics since we are using a truncated Fourier series we can select both the number of Fourier components and sampling frequency. The following is an explanation of how the Fourier series is used to generated our sequence $x(t)$. Some examples of what the ordering of $x(t)$ looks like can be seen in Figure 3.4. The result of this whole process can be observed in Figure 3.5 in which the sampled points as well as the functions are plotted. It can be seen how each sequence only provides some of the function, ideally the network would learn the sequence while the synapses would retain memory of previously seen sequences to guide learning.

1. Define the angular frequency parameterized by time constant, τ_c:

$$\omega_0 = \frac{2\pi}{\tau_c}$$

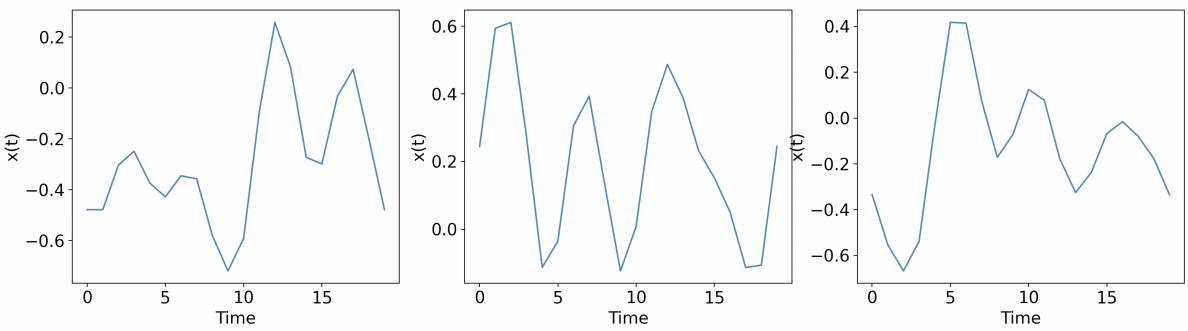


Fig. 3.4 Plots of 3 different Fourier series illustrating the ordering of values in x(t).

2. Accumulate Fourier components for each i from 1 to C where C is the number of Fourier components for our truncated Fourier series.:

$$
\begin{aligned}
d_i &= \frac{\mathrm{U}(-1,1)}{C}, \\
a_i &= \frac{\mathrm{U}(-1,1)}{C}, \\
b_i &= \frac{\mathrm{U}(-1,1)}{C}, \\
x(t) &= \sum_{i=0}^{M} d_i + a_i \cdot \cos(i \cdot \omega_0 \cdot t) + b_i \cdot \sin(i \cdot \omega_0 \cdot t).
\end{aligned}
$$

3. Scale the series to [-1,1] using the hyperbolic tangent function:

$$x(t) = \tanh(x(t))$$

Algorithm 7 Pseudo-code illustrating data generation using Gaussian Processes.

```
procedure CREATE_GP_TASKS(τ)
    for each task from 1 to τ do
        Initialise a Gaussian Process prior
        Sample function values from the GP
        Apply tanh to normalize values
        for i from 1 to data_points do
            Generate Fourier series to sample the x-axis
            Obtain function values at sampled points
            Store (x,y) pairs in task_data
        end for
    end for
    return task_data
end procedure
```

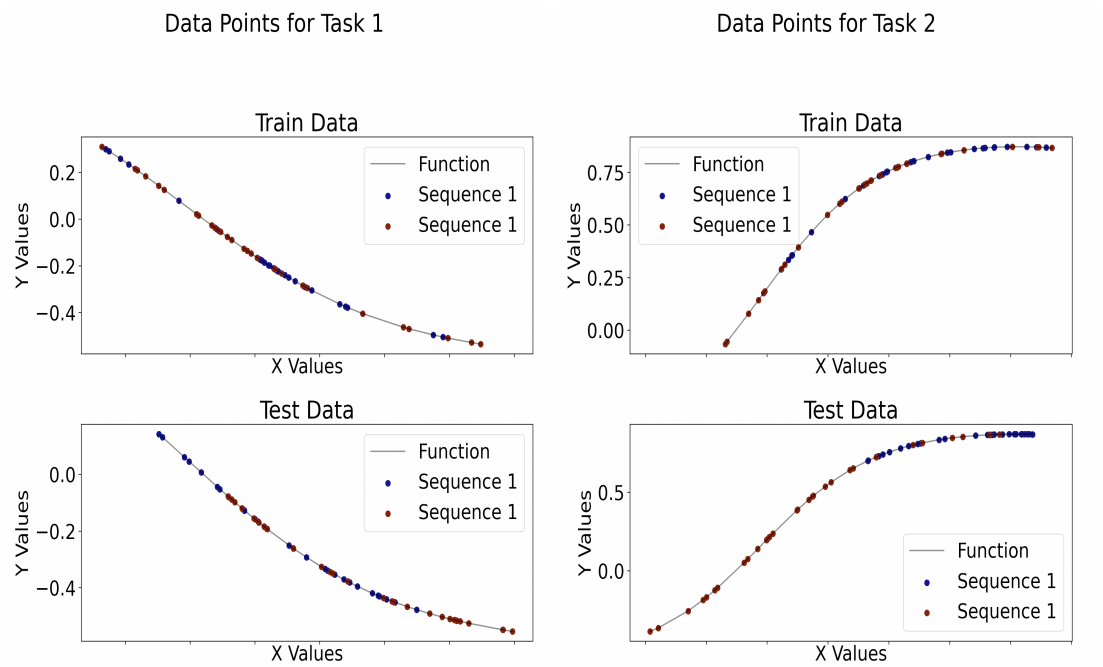


Fig. 3.5 Plots of two tasks train and test sets. Each task consists of 2 sequences sampled at the coloured dots.

Chapter 4

Results

4.1 Performance on Continuous Tasks

Before assessing the performance of the Complex Synapses on the task, it is worth examining how a BPTT and RFLO trained RNN performs as a benchmark. Figure 4.1 shows the average loss over some holdout data over the course of training the RNN. It can be clearly seen that RFLO performs noticeably worse than BPTT with both a slower convergence time as well as a higher loss asymptote. Note that the RNN trained via BPTT employed stochastic gradient descent to make the comparison as fair as possible, that is each gradient update occurred after a forward pass on a single sequence. A further comparison can be seen in Figure 4.2 which shows plots of the predictions made by the two networks overlaid on the true function. The complex synapse predictions were omitted from this figure for clarity. The plots represent two different sequences from the validation set of two different tasks. Note that the BPTT RNN matched the ground truth much better but even then it is not a perfect match. Comparatively the RFLO RNN tended to struggle significantly but it does appear to capture the trend present in the data. We will shortly present the Pearson correlation coefficient to quantify this measure.

In figure 4.3 we can see similar predictions for the complex synapse RNN compared with the BPTT RNN. Clearly, the network performance is significantly worse than even the RFLO RNN. This particular run was performed with 200 dynamical variables to determine if it was a lack of complexity on the part of the network however, increasing dynamical variables/ size of the RCNs did not yield much improvement in predictive power. Given the large jumps in predicted

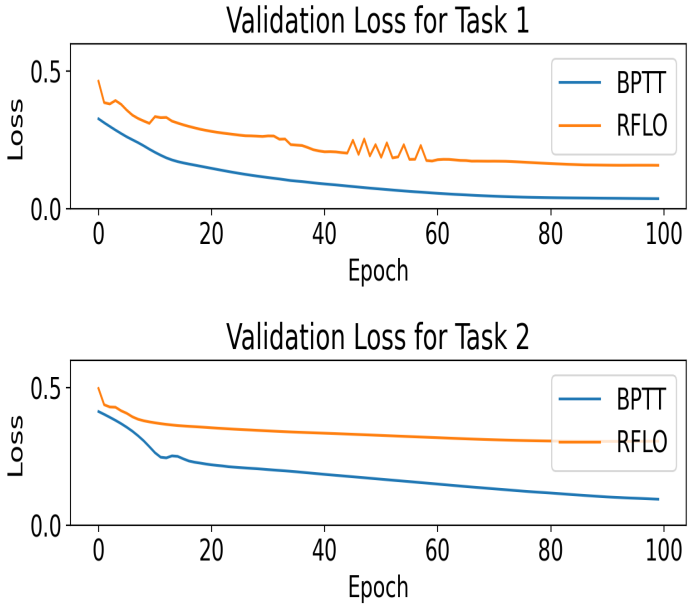


Fig. 4.1 Loss of a 20 neuron RNN trained using BPTT and RFLO on a separate validation set over the course of training over 2 different tasks.

values, we believe there are two possibilities. The first is that the gradient updates result in the network ending up with large quantities in the W tensors. The solution to this would likely be trying weight decay where subsequent weight updates are multiplied by a geometrically shrinking factor. The second possibility is that the sampling method made it difficult to tune the network appropriately. The network is provided values of x(t) in a periodic manner and there is a likelihood that subsequent values receive similar predictions as the network cannot adapt to the changes fast enough. Testing this would require providing sorted inputs to the network. However, that detracts from the fact that the RFLO RNN was still able to learn some representation that our network was not capable of despite much added complexity. Figure 4.4 is provided for reference to illustrate the performance of the 3 training methods on a larger set consisting of 5 tasks. The outcomes are not much different that of the 2 task case. Table 4.1 provides a more clear understanding of the performance of the training methods. We evaluate the average Pearson correlation coefficient over the set of 5 tasks. This effectively measures how closely the data varies with each other, that is are the variations in the sequences correlated or not. Clearly the BPTT RNN performs very well as seen in the right-most column where its correlation is compared with the ground truth. RFLO on the other hand struggles quite a lot

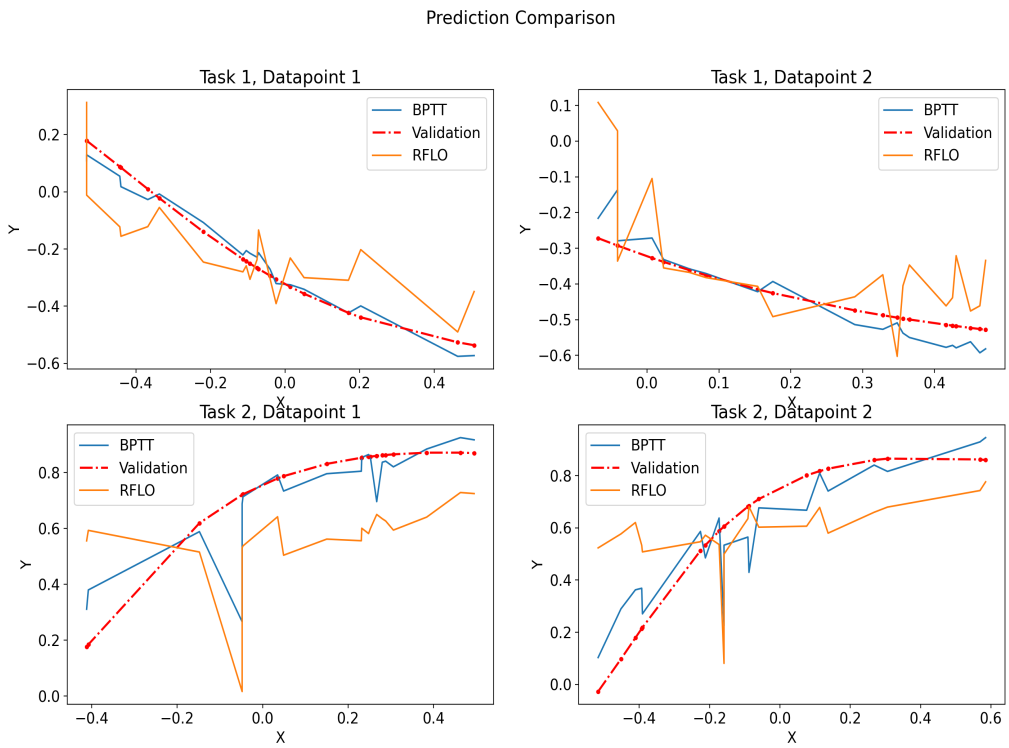


Fig. 4.2 Plots of predictions on specific sequences of different tasks by a 20 neuron RNN trained using BPTT and RFLO. The ground truth is included as well. These plots are sub-sampled from a set of 2 tasks with 5 validation sequences.

with a Pearson correlation of 0.21. This however is still better than the complex synapse RNN which has a negative Pearson's correlation indicating it has not learned the mapping from input to output.

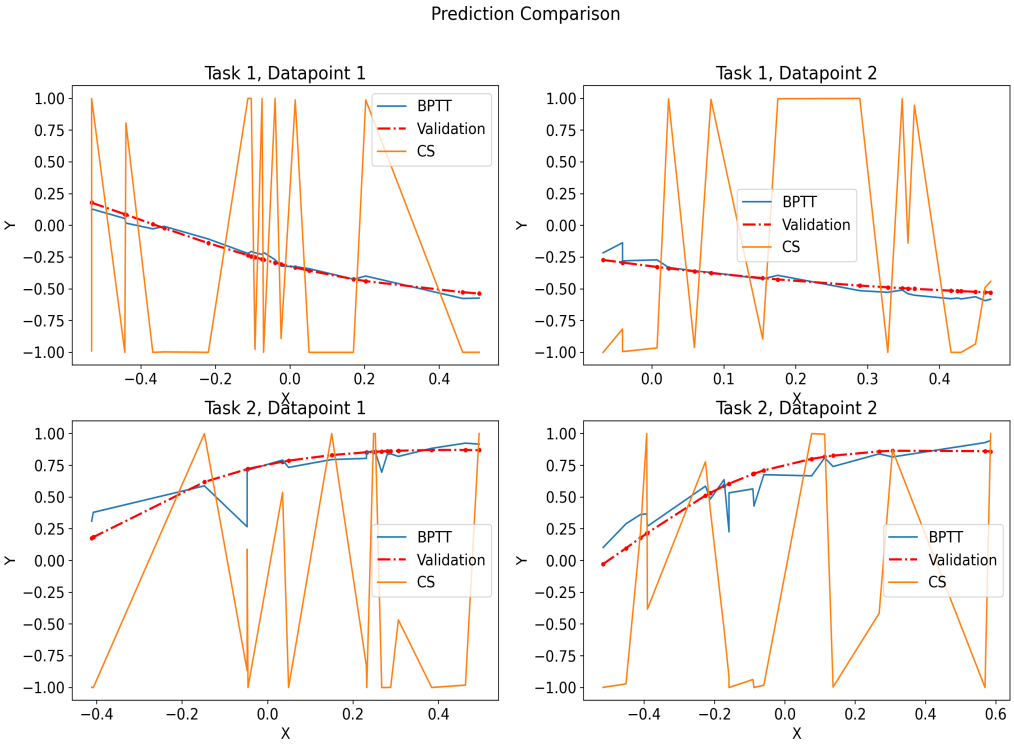


Fig. 4.3 Plots of predictions on specific sequences of different tasks by a 20 neuron RNN trained using BPTT and complex synapses. The ground truth is included as well. These plots are sub-sampled from a set of 2 tasks with 5 validation sequences.

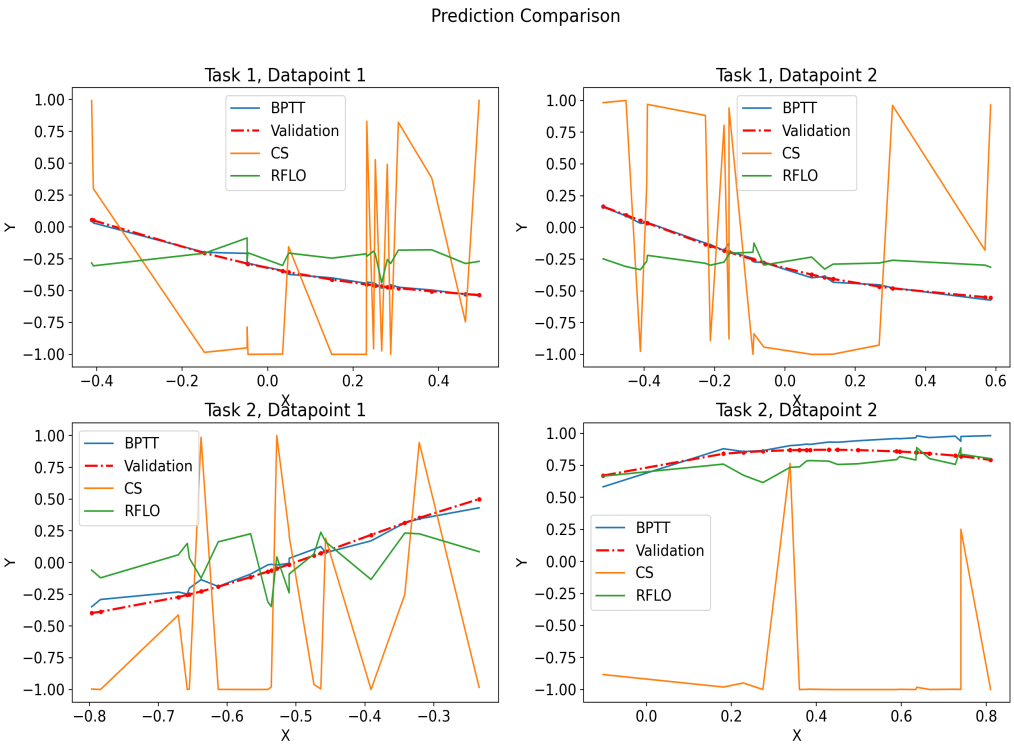


Fig. 4.4 Plots of predictions on specific sequences of different tasks by a 20 neuron RNN trained using BPTT, RFLO and complex synapses. The ground truth is included as well. These plots are sub-sampled from a set of 5 tasks with 10 validation sequences.

Table 4.1 Pearson Correlation of Training Methods of a 20 neuron RNN on a battery of 5 tasks.

	BPTT	CS	RFLO	Validation
BPTT	-	-0.102	0.236	0.98
CS	-	-	-0.064	-0.10
RFLO	-	-	-	0.21
Validation	-	-	-	-

4.2 Performance on Classification Tasks

For reference, the results of the network on classification tasks is also presented to highlight the issues faced with it. As mentioned earlier the length of the sequences obtained from the EMNIST images rendered training this network impractical given our resources. The results in Figure 4.5 are obtained by training this network on simpler synthetic classification tasks as detailed in the previous section. It can be seen that there are clearly two different optima the network converges upon. The left panel illustrates the losses of 7 different tasks converging to the same value indicating a uniform prediction. This comes from the fact that the ideal parameters we desire are likely very sharp optima and so more often than not gradient descent converges to the local minima. The other optimum is illustrated by the right panel which shows the network making confident predictions however, it has not learned the correct mapping and so results in a much higher loss. Thus it is easy to see why the uniform distribution predicting minima is a much more common occurrence. To rectify this issue, changes to the loss function were applied such as placing more mass on later time-step predictions and penalising uniform predictions. Unfortunately neither of these solutions helped and so different approaches were taken.

4.3 Limitations and Suggestions for Improvement

4.3.1 Information Content of Learning Signals

It is worth considering why the network was unable to learn the task representations. For the unsupervised meta-learning approach, a big issue was computational complexity where the GPUs would run out of memory on any reasonably sized task. This limited the complexity our

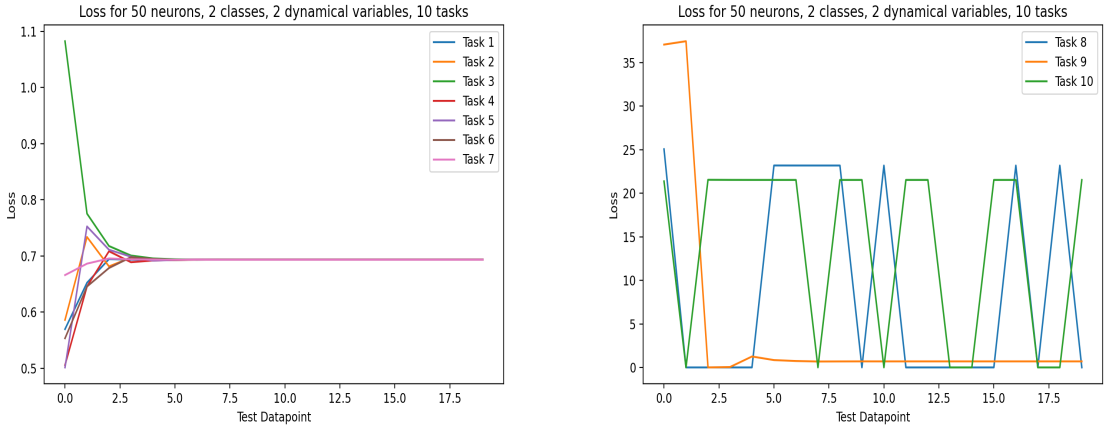


Fig. 4.5 Loss for various tasks on the validation set. The two sub-figures illustrate the different local minima the network parameters converged to.

network could learn since we had to employ stochastic gradient descent on very small datasets which is not ideal when we are attempting to meta-learn a generalisable mapping with very few parameters. However, it is possible that even with more compute, the network would not have been capable of learning sufficiently. This is motivated by the results from the supervised learning approach. When training our synapses on predicting their own gradients from the learning signal inputs, we found that even on training data that the synapses had seen, they were unable to achieve good performance such as an $r^2 \leq 0.2$. In order to test if there was sufficient information in these learning signals, we implemented an overly-complicated network to see if it was possible to over-fit on this training data. This involved adding more linear layers in the following two ways. First instead of using the last synapse state of a sequence for gradient prediction, we use a linear layer layer to accumulate the weighted sum of the synapse states across the last sequence. We replace the $h_{s(T+1)}P_{out}$ term in equation 3.7 with:

$$I_{s+1} = ReLu(A\mathscr{R}[(h_{sT+t})_{t=1}^{t=T}] + b_I) \tag{4.1}$$

where A $\in \mathbb{R}^{1\times P}$ is the parameter vector for the linear layer, $\mathscr{R}[(h_{sT+t})_{t=1}^{t=T}] \in \mathbb{R}^{P\times 11}$ resulting in $I_s \in \mathbb{R}^{1\times 11}$ for each synapse, P is the number of neurons/dynamical variables in the RCNs. This is then followed by another linear layer in the following way:

$$J_{s+1} = ReLu(I_{s+1}B + b_J) \tag{4.2}$$

where B $\in \mathbb{R}^{11\times C_1}$ is another parameter matrix and C_1 is a hyper-parameter determining the size of the linear layer. Values until $10P$ were tried. Then we get a final readout:

$$W_{s+1} = W_s + \eta \tanh(J_{s+1}C + b_W) \tag{4.3}$$

where C $\in \mathbb{R}^{C_1\times 1}$.

After using this expanded model architecture, we were only able to achieve a modest improvement with an $r^2 \simeq 0.3$. This likely indicates that the learning signals do not contain sufficient information for the network to learn an appropriate mapping. It must also be remembered that these learning signals are a linear combination of some candidate learning terms which are themselves a function of only local activity. The goal was that the synapses would implicitly learn the state of the larger system through these signals. This however, is a very difficult task since only a single scalar value is supposed to encode this information for each synapse. A more plausible approach would involves adding a linear layer that expands the dimensionality of these signals and provides that as an input to the network. The following equations represent this modification.

$$F(\Theta) = ReLu(\Theta D + b_\theta) \tag{4.4}$$

$$h_{sT+t+1} = (1-z)h_{sT+t} + z\tanh(F_t(\Theta)_t P_{\text{in}} + h_{sT+t}P_{\text{rec}} + b), 1 \leq t < T \tag{4.5}$$

where $\Theta \in \mathbb{R}^{1\times 11}$ and D $\in \mathbb{R}^{11\times C_2}$. C_2 represents the hyper-parameter specifying the hidden size of this layer. The hidden state update for the RCNs is the same as before except that $P_{\text{in}} \in \mathbb{R}^{C_2\times P}$. The aim here is that the added complexity should do much better at retaining information from the local signals. Unfortunately, putting this into practise did not yield much benefit as we achieved an $r^2 \simeq 0.4$. While this is the best of all approaches, it is still insufficient for these synapses to be practically useful as the error in the gradients is far too large. Thus given all the added complexity was unable to over-fit on the gradients generated by BPTT, it is very likely the learning signals do not contain sufficient information for their intended purpose.

4.3.2 Addressing Large Weight Values

When observing the weight tensor during the course of training with Complex Synapses, it can be seen that there is no bound to them and it is possible for them to result in very large values. These large values could be responsible for the significant bouncing between predicted function values. Thus a possible solution to this is implementing weight decay as follows:

why does alpha^s make this weight decay?

$$W_{s+1} = W_s + \alpha^s \eta \tanh(J_{s+1}C + b_W) \tag{4.6}$$

where $\alpha \leq 1$. Thus over the course of the task the updates would become smaller enabling the weights to converge. While this approach would add to the stability of the network, it is still not a solution for fixing the lacking accuracy in gradient prediction by the synapses.

Chapter 5

Conclusion

We proposed a complex model of synapses as a biologically plausible alternative to BPTT for learning. We examined the model on a variety of tasks both discrete and continuous. Due to a lack of computational resources the original EMNIST dataset could not be used so we created a synthetic battery of tasks involving approximating Gaussian processes. A failure to learn a model with good predictive power led to an exploration of the flaws in the model. Two main issues were observed. The lack of convergence in the weights impacted the stability of the network and a proposed solution for this involves employing weight decay. The more major short-coming of the approach was the candidate learning terms that likely did not encode sufficient information for the problem of gradient prediction for the synapses. This conclusion was tested by adding extra layers to the network to see if we can over-fit on the training data, however this was not successful. For future work, it would be best to provide the pre- and post-synaptic activity and error signal as is. This would allow the network freedom to create functions of these in its implicit mapping. This method could then indicate whether local information can be used in a practical setting for learning in contrast to BPTT's use of non-local signals. Going a step further, it might be worth assessing if complex synapses are practical by providing the RCNs non-local information and assessing if this approach is viable before making the problem harder by reducing it to local information.

References

[1] Amit, Y. (2019). Deep learning with asymmetric connections and hebbian updates. *Frontiers in Computational Neuroscience*, 13.

[2] Benna, M. K. and Fusi, S. (2016). Computational principles of synaptic memory consolidation. *Nature Neuroscience*, 19(12):1697–1706.

[3] Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1):13276.

[4] Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., and Hinton, G. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346.

[5] Lisman, J., Yasuda, R., and Raghavachari, S. (2012). Mechanisms of camkii action in long-term potentiation. *Nature Reviews Neuroscience*, 13(3):169–182.

[6] Liu, J., Moreau, A., Preuss, M., Rapin, J., Roziere, B., Teytaud, F., and Teytaud, O. (2020). Versatile black-box optimization. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, pages 620–628, New York, NY, USA. Association for Computing Machinery.

[7] Murray, J. M. (2019). Local online learning in recurrent networks with random feedback. *eLife*, 8:e43299.

[8] Shervani-Tabar, N. and Rosenbaum, R. (2023). Meta-learning biologically plausible plasticity rules with random feedback pathways. *Nature Communications*, 14(1):1805.

[9] Silva, A., Kogan, J., Frankland, P., and Kida, S. (1998). Creb and memory. *Annual review of neuroscience*, 21:127–48.

[10] Smith, C., Brandt, S., and Papanikolopoulos, N. (1997). Eye-in-hand robotic tasks in uncalibrated environments. *Robotics and Automation, IEEE Transactions on*, 13:903–914.

[11] TensorFlow (2023). Emnist dataset - tensorflow datasets. https://www.tensorflow.org/datasets/catalog/emnist. Accessed: 2023-05-19.

[12] Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

Appendix A

Risk Assessment Retrospective

The project was entirely computational and thus no risks were encountered. Precautions from the risk assessment submitted at the start of the year were sufficient.

A.1 Acknowledgment

Many thanks to Dr. Yashar Ahmadian for his guidance throughout the project. It has been very enlightening and enjoyable working under you. Also a thank you to Puria Radmard for his suggestions on practical matters as well as his encouragement over the course of the project.