UNIVERSITY OF CAMBRIDGE

Department of Engineering

# Biological recurrent neural networks with long short-term memory

Author Name: David He

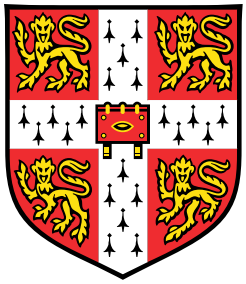
Supervisor: Dr Yashar Ahmadian

Date: 31.05.2023

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

*Signed* _____David He_____ *date* _____31/05/2023___

# Biological recurrent neural networks with long short-term memory



**David He**

Supervisor: Dr Yashar Ahmadian

Department of Engineering
University of Cambridge

This report is submitted for the degree of
*Master of Engineering*

Pembroke College May 2023

# Technical Abstract

Gated recurrent neural networks (RNNs) have been used extensively in fields of machine learning for a variety of complex sequence tasks such as natural language and video processing. They are able to solve these tasks in part due to their ability to hold long short-term memory. These networks are able to remember past inputs at previous time-steps and store key information in a vector known as a *hidden state*. Common gated recurrent neural networks include the GRU (gated recurrent unit) and the LSTM (long short-term memory). These networks are able to process information by using their *gates* to learn which parts of the information are important and should be kept, as well as information that can be forgotten and are not needed to solve a specific task. Beyond the ability to solve tasks by remembering past inputs, gated RNNs have also been shown to be able to *meta-learn*. This phenomenon is when a network is trained on various different structures of the same task and it's still able to learn and adapt to different task structures after training has completed and weights are frozen. Meta-learning enables a network to show *flexible* behaviour, which refers to the ability of a network to react to changes in task structure despite not having been implicitly trained on it.

We studied the effects of meta-learning and flexible behaviour by training a single-layer LSTM of hidden size 48 on the Two-armed bandit task. Results showed that LSTMs can indeed enable flexible behaviour, the agent was able to react to different task structures (different reward probabilities) by altering its behaviour accordingly in an effort to maximise reward over time. GRUs were also found to be able to meta-learn in some form although the performance of GRUs were not as high as LSTMs. From this point, we removed features of the GRU incrementally in an effort to isolate each feature and observe the changes it has on the GRU as a whole.

While gated RNNs have been shown to be very high performing in many different tasks, the gating equations of these artificial networks are not biologically plausible. To investigate this, we proposed a more biologically plausible network inspired from neuroscience theory. Specifically, a network derived from the phenomenological model of synaptic short-term plasticity (STP) proposed by Tsodyks and Markram. Short-term plasticity is theorised to play a part in regulating *working memory* (WM) in the brain. WM refers to a type of memory which is held for a relatively short amount of time but allows information to be held for processing purposes and plays an important role in a variety of cognitive tasks.

The performance of this biologically inspired network, which we called the *Short-term Synaptic Plasticity network* (STPN), was assessed by using it to solve a sequential classification task (permuted sequential MNIST). Different task structures with various levels of difficulty were used to gauge the performance of the STPN on tasks which required more long short-term

memory. The test accuracies from these experiments were plotted and then compared with that of the GRU and vanilla RNN (an RNN without any gates).

Two main types of STPNs were also proposed, one with matrix variables (Synaptic STPN) and one with vector variables (neuronal STPN). We found that despite the large increase in parameters for the synaptic STPN, it only offered slightly better performance over the neuronal STPN. Both networks outperformed the vanilla RNN especially on the tasks requiring more long short-term memory. However, their performance did not reach that of the GRU on the tasks that required more long short-term memory, easier task structures at times resulted in similar test accuracies for the STPNs and GRUs.

We also sought to study the structure of the trained STPNs, by analysing the synaptic properties of STP, we observed that after training, synapses began to become specialised in either facilitation or depression.

The results from using STPNs on permuted sequential MNIST provided evidence of the ability of STP to support working memory in an artificial RNN. The last stage of research focused on whether or not STPNs can meta-learn and show flexible behaviour. While efforts to achieve this on the Two-armed bandit task proved unsuccessful, further research could be carried out on STPNs to improve their performance and potentially get to a point where they are able to be flexible. Changes could be made to the training methods for these biologically inspired networks to improve them and investigate behaviour. Research into STPNs with multiple layers could also prove to be interesting.

# Table of contents

# List of figures

# List of tables

# Nomenclature

**Roman Symbols**

| | |
|---|---|
| $a$ | Action |
| $c_t$ | Cell State |
| $h_t$ | Hidden State |
| $Q^\pi$ | Action-value function |
| $s$ | State |
| $V^\pi$ | State-value function |
| $z_u$ | Inverse of facilitation time constant |
| $z_x$ | Inverse of depression time constant |
| $R_t$ | Return |
| $U$ | Increment of $u$ produced by a spike |
| $u$ | Utilisation parameter for STF |
| $x$ | Normalised variable for STD |

**Greek Symbols**

| | |
|---|---|
| $\gamma$ | Discount rate |
| $\pi$ | Policy |
| $\tau_D$ | Time constant for synaptic depression |
| $\tau_F$ | Time constant for synaptic facilitation |

**Acronyms / Abbreviations**

| | |
|---|---|
| GRU | Gated Recurrent Unit |
| LSTM | Long Short-term Memory |

PCA Principal Component Analysis

PFC Prefrontal Cortex

RNN Recurrent Neural Network

STD Short Term Depression

STF Short Term Facilitation

STPN Short Term Plasticity Network

STP Short Term Plasticity

WM Working Memory

# Chapter 1

# Introduction

Gated recurrent neural networks have been used extensively in machine learning for a variety of complex sequence tasks such as natural language and video processing. They are able to support these models by using recurrence and short-term memory similar to the human brain. A vanilla recurrent neural network (RNN) contains a layer that is fed back into the network and used alongside the external input to produce the output. The advantage RNNs have over other neural networks is that this layer, referred to as the hidden state, essentially acts as "memory" for the network. In this way, future outputs are influenced by both past inputs and outputs.

Currently these gated RNNs feature non-biological elements and we aim to investigate the contribution of both biological and artificial features to the memory timescales and flexibility of RNNs. Some previous research in attempting to unify artificial recurrent networks with their biological counterparts has proved promising however unsatisfactory. Cortical microcircuits as gated-recurrent neural networks (Costa et al 2017)[2] proposed an RNN inspired by LSTMs that gates information through inhibitory cells that are subtractive instead of multiplicative, called a subLSTM. The performance of this RNN used on sequential MNIST and language modelling is similar to that of standard LSTMs, however this work starts from an artificial neural network instead of building up from a biological one. We hope to gain more theoretical understanding of LSTM computations and investigate which features can be compensated for by enlarging the network or adding biological components.

Behavioural flexibility refers to the ability of a network to adapt its behaviour based on the inputs its receiving over long timescales. In the case of an LSTM, the synaptic weights are set during training so that the dynamic variables are able to respond to the inputs and alter the behaviour of the network accordingly. This will be tested using the Two-Armed Bandit problem.[13]

Studying the dynamics of artificial RNNs on this particular task will allow interpretation of the behaviour of the flexible nature of different gated RNNs.

After this, work will shift to focus on developing a new type of artificial RNN derived from short-term synaptic plasticity (STP)[11]. STP has been theorised to be an important process in working memory (WM), a type of memory specialised for computational purposes[8]. This network will involve both short-term facilitation (STF) and short-term depression (STD). Which refer to the increase and decrease respectively of synaptic strength after repetitive stimulation within small timescales. The behaviour and performance of this Short-term Synaptic Plasticity Network (STPN) will be explored by using it to solve sequential MNIST[6]. This will allow us to verify the validity of using biological features on an artificial RNN as well as better understand the dynamics of the time-dependent variables. The behaviour of STPNs will be compared to other gated RNNs.

# Chapter 2

# Background

## 2.1 Biological RNNs

A simple kind of recurrent neural network which is heavily used as a model of biological networks is called the vanilla RNN (it has no gating equations). Vanilla RNNs are heavily used in neuroscience simulations however suffer from issues such as the vanishing gradient problem. They can be derived directly from the time independent form of the biological RNN. From there, we add more features and increase complexity to provide an overview of various kinds of RNNs.

1. The time independent form of the biological RNN is defined as follows:

$$\mathbf{h} = \varphi(\mathbf{W}\mathbf{h} + \mathbf{I})$$

Where $\mathbf{h}$ is the vector of neuronal firing rates or in the case of artificial RNNs, the hidden state vector. $\varphi$ represents a non-linearity, $\mathbf{W}$ is the matrix of recurrent synaptic weights and $\mathbf{I}$ is the vector of external inputs. The equation above is recurrent because the hidden state vector is dependent on itself.

2. $\mathbf{h}$ can be made time dependent by introducing a $\frac{d\mathbf{h}}{dt}$ term. This means that $\mathbf{h}$ is no longer in steady state and now will change throughout time.

$$\tau \odot \frac{d\mathbf{h}}{dt} = -\mathbf{h} + \varphi(\mathbf{W}\mathbf{h} + \mathbf{I})$$

Here $\tau$ is the single-neuronal relaxation time constant.

3. Discretize time using the Euler method:

$$\mathbf{h}_t = (1 - \mathbf{z}) \odot \mathbf{h}_{t-1} + \mathbf{z} \odot \varphi(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{P}\mathbf{x}_t + \mathbf{b})$$

with $\mathbf{z} = \frac{\Delta_t}{\tau}$ which is a constant, however the network can be made more complex allowing more memory flexibility by turning $\mathbf{z}$ into a vector with different components that are constant in time, this means that the value for $\mathbf{z}$ is different for different neurons in the network:

$$\mathbf{z_i} = \frac{\Delta \mathbf{t}}{\tau_i}, \quad \mathbf{i} = 1, ..., N$$

$\tau$ is now a vector which means different neurons can have different relaxation time constants. Note that after discretizing time, the $\mathbf{h}_{t-1}$ term is what allows the network to have memory. $\mathbf{h}_{t-1}$ represents the hidden state at the previous timestep, in this way, information is propagated through time.

4. For even more complexity, $\mathbf{z_i}$ which is constant through time can also be made into a time dependent dynamic variable. The RNN reached at this stage will be referred to as a "simple GRU".

The main issue with these simpler RNNs or "vanilla" RNNs is the vanishing/exploding gradient problem. For long sequences of data, during gradient descent, the gradients become too small or too large leading to either no learning or divergence. This problem can be solved by

using more complex RNNs called gated RNNs, these include Gated recurrent units (GRUs)[1] which uses two gating variables, the Reset gate $\mathbf{r}_t$ and the Update gate $\mathbf{z}_t$.

The GRU update rules are given by:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{h}_{t-1} + \mathbf{P}_z\mathbf{x}_t + \mathbf{b}_z)$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{h}_{t-1} + \mathbf{P}_r\mathbf{x}_t + \mathbf{b}_r)$$

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1-\mathbf{z}_t) \odot \varphi(\mathbf{W}\mathbf{r}_t \odot \mathbf{h}_{t-1} + \mathbf{P}\mathbf{x}_t + \mathbf{b})$$

Both gates use the sigmoid function to control the flow of memory in the network through time. Because the sigmoid function is bounded between 0 and 1, it allows clear interpretation of memory propagation. If the Update gate $\mathbf{z}_t$ outputs a 1, it indicates that all the memory from the previous timestep should be saved. It's obvious from the equations that:

$$\mathbf{z}_t = 1 \quad \Rightarrow \quad \mathbf{h}_t = \mathbf{h}_{t-1}$$

The above result is independent of the value of the Reset gate $\mathbf{r}_t$. Similarly, if the Update gate outputs a 0, it indicates that all the information from previous timesteps should be forgotten. From the equations:

$$\mathbf{z}_t = 0 \quad \Rightarrow \quad \mathbf{h}_t = \varphi(\mathbf{W}\mathbf{r}_t \odot \mathbf{h}_{t-1} + \mathbf{P}\mathbf{x}_t + \mathbf{b})$$

In this case, the Reset gate will decide how much of $\mathbf{h}_{t-1}$ will be retained, if the Update gate is also 0, then $\mathbf{h}_t$ will be updated depending exclusively on the new input.

Another type of gated RNN is the Long short-term memory network (LSTM)[3], which is even more complex than the GRU, having three gates, the Input gate $\mathbf{i}_t$, the Forget gate $\mathbf{f}_t$ and the Output gate $\mathbf{o}_t$. Another key feature of LSTMs is that it includes an extra state compared to GRUs called the Cell state $\mathbf{c}_t$ or Long term memory. This means that compared to GRUs, LSTMs are more able to maintain memory from many timesteps ago.

The LSTM update rules are given by:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \varphi_1(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{P}\mathbf{x}_t + \mathbf{b})$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \varphi_2(\mathbf{c}_t)$$

$$\mathbf{y}_t = \sigma(\mathbf{W}_y\mathbf{h}_{t-1} + \mathbf{P}_y\mathbf{x}_t + \mathbf{b}_y), \ \ \mathbf{y} \in \{\mathbf{i}, \mathbf{o}, \mathbf{f}\}$$

The standard choice of nonlinearities $\varphi_1(u), \varphi_2(u)$ is $\tanh(u)$. This is a common choice for artificial neural networks, however to achieve biological realism, $\varphi_1(u), \varphi_2(u)$ must have a non-negative output, an example of this would be the rectified supralinear power-law function: $\varphi(u) = \max(0,u)^n$ with $n > 1$. The sigmoid function is used in STPNs shown later.

Another important biological detail to note is Dale's Principle, which states that any given neuron can only make either excitatory or inhibitory connections, because the main neurotransmitters in the brain are glutamate (excitatory) and GABA (inhibitory). This means that the synaptic weights $\mathbf{W}$ and $\mathbf{P}$ in the RNN models should satisfy the condition that matrix elements in any column must all have the same sign. An artificial neural network must then satisfy both of these principles in order to be more "biological" however Dale's Principle will be ignored for our research.

## 2.2 Reinforcement Learning

Reinforcement learning is a type of machine learning which focuses on how an agent should act in a specific environment in order to maximize total reward[10]. An agent essentially learns by trying different actions and reacting to the amount of reward received after each ac-

tion. The learner is not told which actions to take, unlike other types of machine learning. In more complex cases, actions at any one time step may not only affect the immediate reward but also future rewards.

Reinforcement learning is used widely in neuroscience due to its similarity to biological learning, as dopamine neurons have the formal characteristics of the teaching signal known as the temporal difference (TD) error. It will be used for the Two-armed bandit task which will be described below in order to simulate the actions of an agent (rhesus monkeys) and to train an RNN.

One of the issues with reinforcement learning is the importance of a number of hyperparameters that are set at the beginning of training.

### 2.2.1 Elements of Reinforcement learning

In reinforcement learning, the agent can be seen as the learner which makes actions and receives rewards, the environment is everything the agent interacts with. A state represents the environment at a given time step.

Besides these there are four main subelements of reinforcement learning: a *policy*, a *reward function*, a *value function* and a *model* of the environment[10].

A policy essentially defines the agent's behaviour, at any time step, the policy is used by the agent to decide what action to take. The policy can exist as a function or a lookup table. In some cases including the experiments below, the policy is represented by an RNN.

A reward function maps a state-action pair of the environment to a reward. Since the goal of reinforcement learning is to maximise the total reward over a period of time, the reward function defines which states and actions are desirable. In biology, reward can be seen as representing pleasure or pain.

The value function is similar to the reward function except that it specifies which actions are good in the long-term. The value of a state represents the total reward an agent can accumulate in the future. However, unlike the reward function which comes from the environment, the value function must be estimated.

The fourth element of reinforcement learning is a model of the environment. A model mimics the environment and predict the next state and next reward for a given state and action. In some reinforcement learning methods, learning comprises of both trial and error and learning a model of the environment.

### 2.2.2 Returns

If a sequence of rewards that are received after time step $t$ is written as

$$r_{t+1}, r_{t+2}, r_{t+3}, \ldots$$

then the goal of reinforcement learning is to maximise the expected return, where the return $R_t$ is defined as a specific function of the reward sequence. In the most basic form the return is just the sum of all the rewards:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T$$

where $T$ is the final time step in the reward sequence. T exists in cases where the task structure has a natural end. For example, in the Two-bandit task which will be described below, each set of trials will end after a constant fixed number of trials. However, there are some situ-

ations where $T = \infty$ and the task has no natural endpoint.

An additional concept that is needed now is *discounting*, in certain cases, future rewards are worth less in the present because it takes more time to actually receive those rewards. The *discount factor* $\gamma$ determines the present value of future rewards.

Now the return can be written as:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where the goal is to maximise the expected discounted return. Note that $0 \leq \gamma \leq 1$, if $\gamma = 0$, then future rewards have no present value whereas $\gamma = 1$ means that any future rewards are worth the same in the present.

### 2.2.3 Value Functions

As stated above, many reinforcement learning algorithms involve estimating the value function of states or state-action pairs. Because any future rewards are dependent on any future actions taken, value functions are defined for specific policies.

A policy, denoted by $\pi$ maps each state, $s \in \mathbb{S}$, and action, $a \in \mathbb{A}(s)$ to the probability $\pi(s,a)$ of taking action $a$ when in state $s$[10].

The value of a state $s$ with a policy of $\pi$ is denoted as $V^\pi(s)$ and can be seen as the expected return when starting in $s$ and following $\pi$. $V^\pi(s)$ can be defined as

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t | s_t = s\} \\ &= E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right\} \end{aligned}$$

where $E_\pi\{\}$ is the expected value for an agent that follows a policy $\pi$. The function $V^\pi$ is the *state-value function* for $\pi$.

Another version of a value function is the *action-value function* for a policy $\pi$, denoted $Q^\pi$. In this case the value is defined by taking action $a$ in state $s$ under $\pi$

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\{R_t | s_t = s, a_t = a\} \\ &= E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a \right\} \end{aligned}$$

The value functions $V\pi$ and $Q^\pi$ can be estimated from experience. This means that if an agent follows a policy $\pi$, the average of the returns following every state encountered will converge to the state's value $V^\pi(s)$. This convergence will occur as the number of times that state is encountered approaches infinity.

If also separate averages are kept for each action taken for each state encountered, then these averages will converge to the action values, $Q^\pi(s,a)$. These methods of estimation where averaging over random samples of actual returns are known as Monte Carlo methods.

In reality, if there are a large number of states, then it isn't practical to calculate averages for each state encountered. Instead, the value functions are approximated using parameterised functions with parameters that are updated to better match the observed returns.

### 2.2.4 Temporal-Difference Learning

Temporal-Difference Learning (TD Learning) is a method within reinforcement learning which updates estimates based on other learned estimates, instead of waiting for a final outcome. This idea is known as bootstrapping[10].

Given some experience following a policy $\pi$, TD learning updates the estimate $V$ of $V^\pi$, the estimate $V(s_t)$ is updated based on what

happens after the visit to $s_t$. This update also applies to the Monte Carlo method, which uses the return following a visit.

A basic every-visit Monte Carlo method suitable for nonstationary environments is

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$$

where $R_t$ is the actual return after time $t$ and $\alpha$ is a constant step-size (learning rate).

The difference between Monte Carlo methods and TD learning is that Monte Carlo methods must wait until the end of the episode to use the return for the update equation. Whereas, TD methods only wait until the next time step.

The simplest TD method, known as TD(0), is

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

Essentially, the target for the Monte Carlo update is $R_t$, and the target for TD learning is $r_{t+1} + \gamma V_t(s_{t+1})$.

This works because of the below conversion:

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t | s_t = s\} \\ &= E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right\} \\ &= E_\pi\left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s \right\} \\ &= E_\pi\{r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s\} \end{aligned}$$

Note that TD learning uses the current estimate $V_t$ because the true $V^\pi$ is not known.

### 2.2.5 Actor-Critic Methods

Actor-critic methods are a type of TD learning that have a separate memory structure to explicitly represent the policy independent of the value function[10].

The policy structure is referred to as the *actor* and is used to select actions, the *critic* is the estimated value function and is used to criticise the actions made by the actor. Learning is always on-policy, which means that the critic must learn from experiences that comes from actions made by the actor.

Typically, the critic is a state-value function, the critic evaluates if a new state is better or worse after each action. This evaluation is the TD error:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t),$$

where $V$ is the current value function estimation.

The TD error is used to evaluate action selection from the actor. If the TD error is positive then it suggests that the action that was just taken, $a_t$ is good and should be selected more in the future. The direct opposite is true if the TD error is negative.

In the methods section of this report, a specific type of actor-critic methods called Advantage Actor-Critic (A2C) is used[7].

## 2.3 Two-armed bandit problem and meta-learning

In order to assess the performance of gated RNNs and their flexibility, different types of RNNs with different hidden sizes (representing different numbers of neurons) will be used to solve the two-armed bandit task.

The two-armed bandit task consists of two 'arms', each of which has a particular probability of providing a reward when chosen. The probabilities of each arm is unknown to the

subject and the task is for the subject to try to maximise their total reward over a fixed number of trials. In order to solve this problem, the subject has to estimate the probabilities of the arms and try to choose the arm with the higher probability more often than the other.

This leads to the exploration-exploitation tradeoff, the subject must explore by picking each arm multiple times in an attempt to figure out which arm is more likely to provide a reward. Once this is done, the arm with a higher reward probability should be picked every time for the remaining trials. Too much exploration or exploitation will lead to suboptimal rewards.


Fig. 2.1 *Illustration of the Two-armed bandit task, subject has to choose between two arms, actions taken can be denoted as $a_L$ for the left arm and $a_R$ for the right arm. The probability of $a_L$ is $p_0$ and that of $a_R$ is $p_1$, two types of the task can be formed, one where $p_0 = 1 - p_1$ and the other where $p_0 \neq 1 - p_1$.*

Wang et al seeks to solve the two-armed bandit problem using reinforcement learning specifically the Actor-Critic (A2C) method. At each trial, an LSTM takes as input the previous action (which arm was chosen) and the previous reward. In this model, the action is one-hot encoded and the reward is represented as 0 for no reward and 1 for reward given. At each trial, the LSTM outputs the next action to be taken.

Using an RNN to represent the policy function of the reinforcement learning algorithm allows an easy solution to a problem with fixed reward probabilities for both arms.

However, in order to test the flexible nature of RNNs, the experiment is set up in a way which has the reward probabilities change throughout trials. Wang et al define an *episode* to contain a set number of trials, the task then consists of multiple episodes all with a constant number of trials per episode. At the beginning of each episode, the reward probabilities are randomly drawn from a distribution and then held fixed for the entirety of that episode. In this way, the RNN must adapt to different episodes and learn to respond to changes in reward probabilities.

Wang et al found that after training an LSTM using the above model and A2C that the dynamic variables in the network can change over trials so that the LSTM can change its behaviour for all reward probabilities not just a single set. The adaptability is due to the flexible nature that some gated RNNs have.

Since the synaptic weights are fixed after training, any learning the network does in order to adapt to a new set of reward probabilities are not related to the initial reinforcement learning algorithm. This is known as 'meta-learning', which is when one learning algorithm produces a second learning algorithm (essentially learning to learn).

This also leads to the idea of a hierarchy of learning, the initial process of training the synaptic weights using error backpropagation can be seen as high level learning. This process takes a long time and enables another type of

learning. One which is faster and more low level.

## 2.4 Short-Term Synaptic Plasticity and working memory

*Working memory (WM)* allows information to be held for a relatively short amount of time compared to long-term memory. It allows information to be held for processing purposes and plays an important role in a variety of cognitive tasks [8]. Working memory is required to remember information that varies unpredictably in time. Whereas, another form of memory called *reference memory* is used to retain information that remains constant over time.

In the delayed-response task, both working memory and reference memory are required by a subject to solve the task. In these tasks, some information is revealed to a subject, the information is then withdrawn and a waiting period takes place (recall delay). After this delay, the same initial information is presented to the subject as well as some different information, the goal for the subject is to choose which information was presented at the beginning of the trial. Working memory is clearly needed to maintain memory of the information during the recall delay, reference memory is also needed to remember the task structure across multiple trials.

Enhanced, stimulus-specific spiking activity has been observed during the recall delay, this is considered to be a neuronal correlate of WM. Delay activity has been theorized to either emerge from intrinsic cell properties or persistent reverberations in selective neural populations that code for different memories. However, studies have shown that the delay activity increase is sometimes very modest and in certain scenarios may disappear completely during part of the delay period. This suggests that perhaps WM does not reside completely in the spiking activity. Another point to note is that preserving information in a spiking form is energetically expensive due to the high metabolic cost of action potentials.

Another theory of WM is based on properties of excitatory synaptic transmission in the prefrontal cortex (PFC). Mongillo et al [8] theorised that an item is maintained in the WM state by *short-term synaptic facilitation* mediated by increased residual calcium levels at the presynaptic terminals of the neurons that code for this particular item of information.

The removal of residual calcium from presynaptic terminals is a relatively slow process, this means that memory can be held for around 1s without any enhanced spiking activity.

Short-term plasticity (STP) refers to a phenomenon where synaptic efficacy changes over time in a way which reflects the history and timing of presynaptic activity. Synaptic efficacy refers to the strength of the connection between neurons.

There are two types of STP, one is Short-Term Facilitation (STF) and the other Short-Term Depression (STD). These have opposite effects on synaptic efficacy.

STF is caused by an influx of calcium into the axon terminal after spiking activity, which increases the release probability of neurotransmitters.

STD on the other hand is caused by the depletion of neurotransmitters consumed during the synaptic signaling process at the axon terminal of a presynaptic neuron.

It's found that different synapses in different cortical areas have different forms of STP, some are more facilitation dominant and others more depression dominant.

In contrast to long-term plasticity, STP has a shorter time scale, typically on the order of hundreds to thousands of milliseconds. The changes to synaptic efficacy made by STF and STD are temporary and disappear completely without continued presynaptic activity. The speed at which these changes disappear can be modelled using time constants which can differ between facilitation and depression.

The biophysical processes underlying STP are very complex, consequently, studies in the computational behaviour of STP have relied on the creation of phenomenological models.

### 2.4.1 Phenomenological model of STP

In the model proposed by Tsodyks and Markram (Tsodyks 98), STD is modelled by a normalised variable $x$ where $0 \leq x \leq 1$, denoting the fraction of neurotransmitters that remain available after some have been depleted. STF is modelled by the parameter $u$, which represents the fraction of available resources ready for use (release probability).

The dynamics of STP under this model are given as

$$\frac{du}{dt} = -\frac{u}{\tau_F} + U(1-u^-)\delta(t-t_{sp}),$$

$$\frac{dx}{dt} = \frac{1-x}{\tau_D} - u^+x^-\delta(t-t_{sp}),$$

$$\frac{dI}{dt} = -\frac{I}{\tau_s} + Au^+x^-\delta(t-t_{sp}),$$

$$\Delta I(t_{sp}) = Au^+x^-$$

where $t_{sp}$ denotes the spike time and $U$ is the increment of $u$ produced by a spike. $u^-$ and $x^-$ are the values of $u$ and $x$ just before the arrival of the spike and $u^+$ denotes $u$ just after the spike.

$\Delta I(t_{sp})$ is the synaptic current generated at the synapse by the spike arriving at $t_{sp}$. $A$ is known as the absolute synaptic efficacy of the connections.

Upon a spike, an amount $u^+x^-$ of the available resources is used to produce the postsynaptic current. This results in $x$ being reduced to form $x^+$. This process mimics neurotransmitter depletion. Following a spike, $u$ also increases, mimicking calcium influx into the presynaptic terminal.

As described above, the changes to synaptic efficacy caused by changes to $u$ and $x$ are not permanent. Between spikes, $u$ and $x$ recover to their baseline levels ($x = 1$ and $u = 0$) with time constants $\tau_D$ and $\tau_F$ for depression and facilitation respectively. The phenomenological model reproduces the behaviour of cortical synapses, a synapse is facilitating if $\tau_F > \tau_D$, meaning that it takes more time for $u$ to return to its baseline level. If the opposite is true and $\tau_D > \tau_F$, then the synapse is depressing.

The Tsodyks and Markram model can be slightly altered so that the baseline of $u$ is $U$ instead of 0,

$$\frac{du}{dt} = -\frac{U-u}{\tau_F} + U(1-u^-)\delta(t-t_{sp}).$$

# Chapter 3

# Results

## 3.1 Two-armed bandit task

The simulations we carried out on the Two-armed bandit task were influenced from Wang et al [13]. After training on various reward probabilities, the weights of the network were fixed and the network was then used on multiple episodes in order to test for meta-learning.

In the case of the first simulation involving an LSTM of a hidden size of 48, the only properties of the LSTM that can change are the hidden state $h_t$ and cell state $c_t$.

Episodes used for testing used reward probabilities that were manually set instead of randomly sampled.

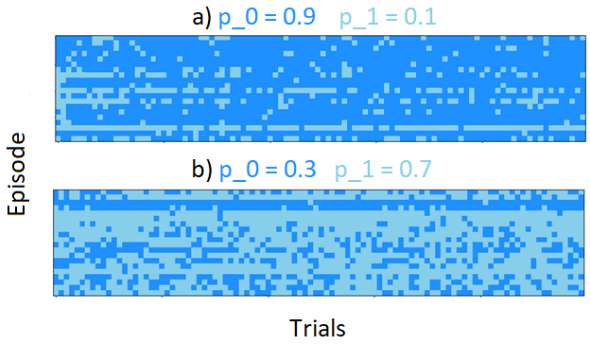



Fig. 3.1 *Colourmap showing actions decided by the LSTM model after training, dark blue corresponds to the first arm with probability $p_0$ and light blue corresponds to the right arm, probability $p_1$.*

Fig.3.1 shows the actions performed by the agent during 20 episodes for two different sets of reward probabilities (episodes were limited to 100 trials which is the same as training).

From looking at the two colourmaps from afar, it is clear that some form of meta-learning has taken place. The agent is able to identify which arm provides a higher probability of reward multiple times (since hidden state and cell state are reinitialised to 0 at the start of each episode). In some cases, the network is unable to make the correct decision which leads to entire episodes with many instances of the "wrong" arm being chosen. This issue occurs for more episodes in the experiment with $p_0 = 0.3, p_1 = 0.7$, which is to be expected because the reward probabilities are more similar so harder to differentiate by the network. This behaviour is not present in Wang et al and our attempts to perfectly replicate their results were unsuccessful. This also means that while the LSTM in our experiments is able to learn the reward probabilities, it doesn't show clear, separate regions of exploration and exploitation. However, our results are good enough to conclude that LSTMs are able to support meta-learning.

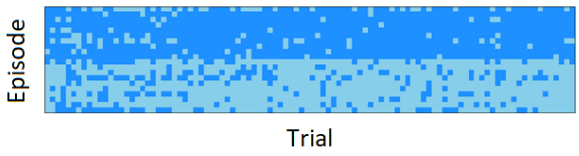



Fig. 3.2 *LSTM subject to reward probabilities of $p_0 = 0.9, p_1 = 0.1$ for 10 episodes and then $p_0 = 0.1, p_1 = 0.9$ for 10 more episodes. Hidden states are not reinitialised between episodes.*

Another experiment that we ran in order to test for meta-learning was to put the network through 20 trials without reinitialising between

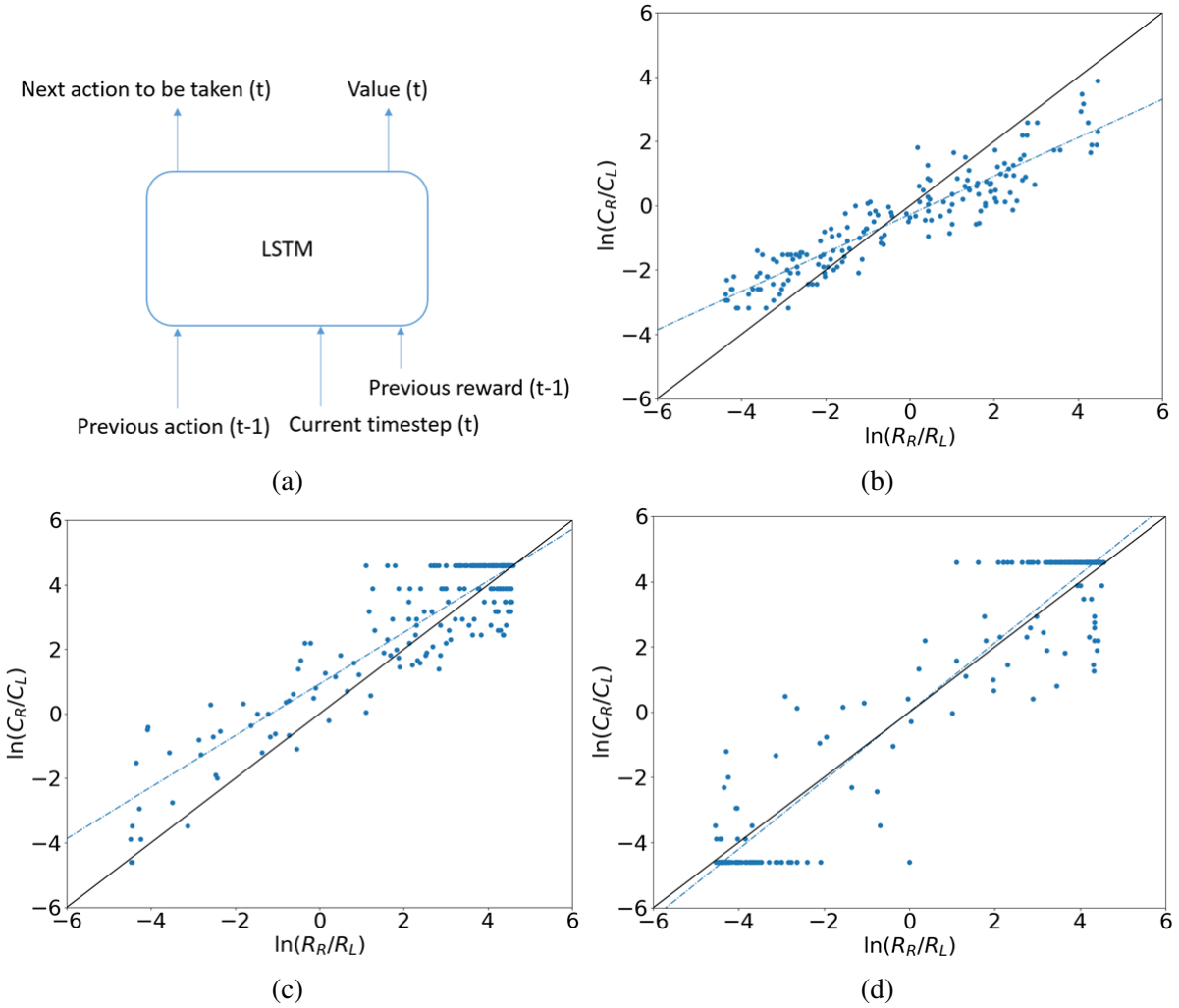



Fig. 3.3 *Model and results after RNN training. a) Agent architecture, the LSTM takes as input: the previous action (one-hot encoded), the current timestep and the previous reward (0 or 1), the LSTM layers are connected all-to-all with a softmax layer which outputs the next action (0 or 1) and a single linear unit which outputs a value. b) Probability matching from Lau and Glimcher[5]. $C_R/C_L$ is the ratio of the number of trials the right arm was selected to the left arm, $R_R/R_L$ is the number of times choosing the right arm provided a reward compared to choosing the left arm. This shows the results produced by the LSTM after running on 200 episodes with incremental reward probabilities (weights fixed). c) Probability matching for GRU trained with same hyperparameters as LSTM. d) Same as c but for a GRU with twice the hidden size (96).*

episodes. After the 10th trial, the reward probabilities are flipped.

The shift in Fig.3.2 from dark blue to light blue shows the LSTM reacting to the sudden change in reward probabilities. The results are promising because they show that the network is able to recognise changes in reward probabilities not just learn them from the start of the experiment. This could explain the performance and behaviour of the network, it doesn't show a clear exploitation region because the network is always exploring which enables it to react quickly to sudden changes in reward probabilities.

Despite the network not displaying a clear exploitation region, Fig 3.3b does show a linear correlation between choice and reward received which is similar to the probability matching

from Lau and Glimcher (from monkeys)[5]. The best fit line however does not perfectly match that of $y = x$ which points to inefficiencies in the model.

The next experiments swapped out the LSTM for a GRU (hidden size also at 48) while all other training and testing parameters remained the same. The probability matching of this GRU after training is shown in Fig.3.3c, unlike the graph of the LSTM of the same hidden size, the points are not evenly distributed alone a straight line. Instead there seems to be a preference for picking one arm over the other. We find that in this case, the GRU will in most cases pick the right arm even though it might not be the more rewarding one.

Fig.3.3d shows the probability matching of a GRU with a hidden size of 96, this network displays a similar type of behaviour to the GRU with hidden size 48. However, the main difference is that the network here no longer shows preference for a certain arm. Both GRUs will in most cases reach a decision very quickly and exploit for most of the episode even if the decision it made was wrong.

From the figures it's clear that the performance of the GRUs are not very good when it comes to maximising reward, however the points that lie between either end of the graphs are still roughly distributed along a diagonal. We find that GRUs (at a size similar to that of an LSTM) are not able to replicate the exploration regimes of an LSTM and tend to maximise exploitation. However, the graphs still show that GRUs are able to meta-learn in some form because the behaviour of the network is not independent of the reward probabilities.

The performance of the three networks described above can be assessed by comparing network performance against a 'dumb' agent which has a 50% chance of choosing either arm and an 'oracle' agent which always picks the arm with the higher reward probability.

| Agent | LSTM(48) | GRU(48) | GRU(96) |
|---|---|---|---|
| Dumb | 9940 | 9902 | 9954 |
| Network | 12194 | 10677 | 11359 |
| Oracle | 14571 | 14579 | 14526 |

Table 3.1 *Total reward obtained after evaluating 200 episodes of incremental reward probabilities.*

It's somewhat interesting that the LSTM network doesn't reach the performance of the oracle because it displays too much exploration whereas the GRUs exploit too much without exploring. These differences could be attributed to the differences between the network architectures, i.e. the GRU lacks long term memory because it doesn't have a cell state $c_t$. However, these differences could also come from the reinforcement learning algorithm used during training.

### 3.1.1 Hidden state trajectories

To further understand the performance of the trained LSTM of hidden size 48, the trajectories of the hidden states of the network can be plotted and visualised in two dimensions.

At the beginning of every episode, the hidden state is reinitialised to 0, after 100 trials, the hidden state at each time-step is recorded. PCA is performed on the sequence of hidden states [9] and the first two principal components are used to produce a 2D plot of the trajectory.

The trajectories in Fig3.4a are inspired by Jordan et al[4], which views GRUs as continuous time dynamical systems, and inputs are seen as noise, which interrupts the trajectory of the GRU. To explore the system of

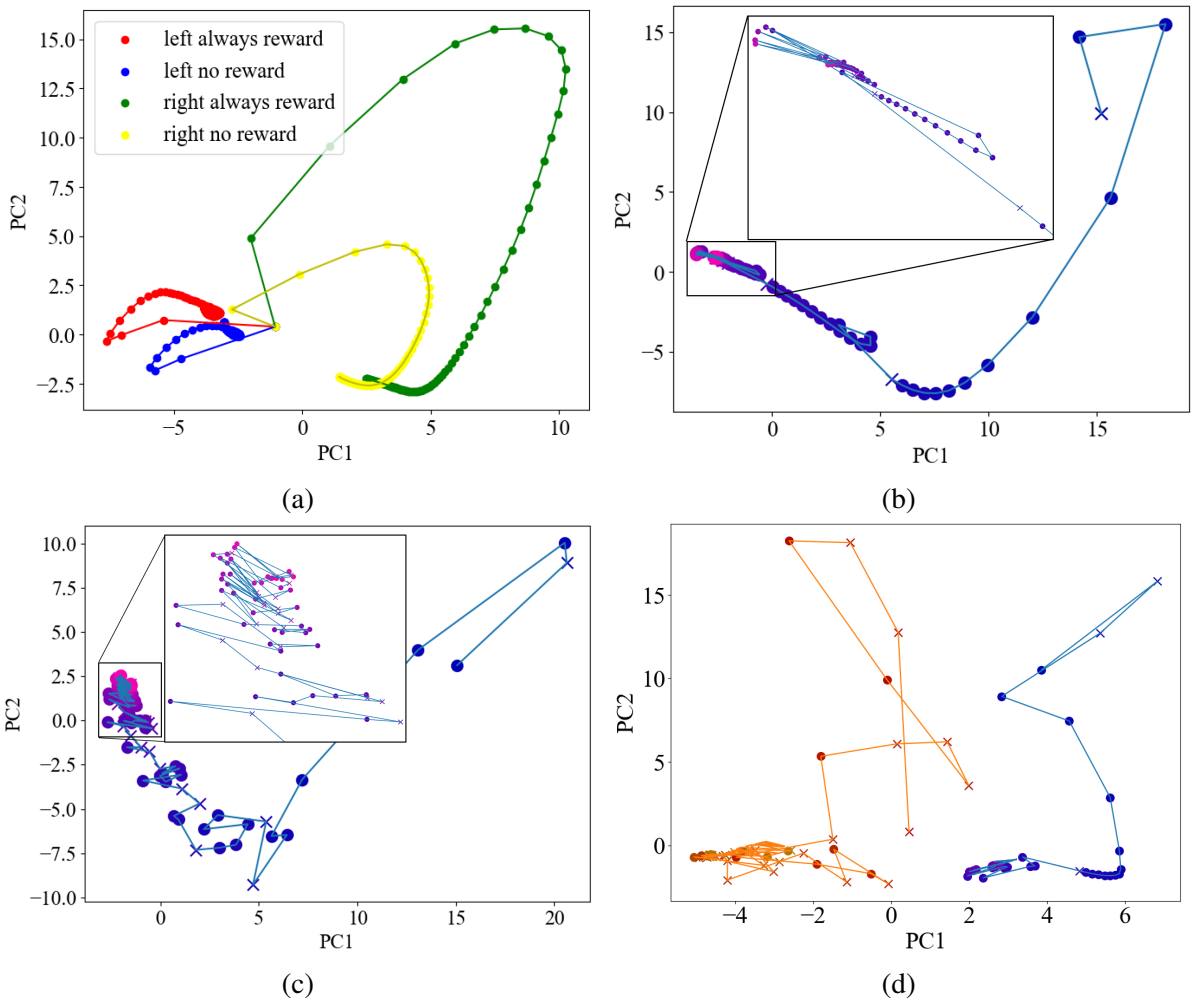



Fig. 3.4 *Hidden state trajectories of the LSTM, dots and crosses indicate which arm was chosen. a)Trajectories of fixed inputs(i.e. the network always chooses a particular arm and either always receives reward or never receives reward). b)Trajectory based on reward probabilities of* $p_0 = 0.99, p_1 = 0.01$. *c)Trajectory based on reward probabilities of* $p_0 = 0.45, p_1 = 0.55$. *d)Two trajectories plotted together of reward probabilities* $p_0 = 0.9, p_1 = 0.1$ *and* $p_0 = 0.1, p_1 = 0.9$.

the LSTM, Fig3.4a shows the LSTM receiving a constant input throughout the episode (four different inputs were used based on possible combinations of choice and reward), smooth trajectories were produced which all tended toward different fixed points. Plotting a 3D plot and using PC3 further differentiated these four points but not by much more. The four trajectories are labelled in the figure, "left always reward" for example, corresponds to a case where the network is receiving the input $[1,1,0,\text{current time-step}]$[12] at every time-step for 100 trials. This input corresponds to a case where the agent sees that it chose the left arm on the last trial and always receives a reward.

We can see from the trajectories that 'Left always reward is in a very different area to'Right always reward', this is to be expected since the two areas represent different conclusions reached by the agent. If the network finds that the left arm always provides a reward then it is found to be the optimum arm, which leads the hidden state to follow a smooth trajectory to the point the blue line tends toward. Similarly, if the network reaches the opposite con-

clusion, then the hidden state will follow the green line. The fact that these two points are far apart means that the hidden states are very different in the cases where the network makes these deductions.

It's interesting that the trajectories that are based on the same actions are grouped closer together. 'Left always reward' is very close to 'Left no reward', and this is also the same case for 'Right always reward' and 'Right no reward'. This could explain the explorative nature of the LSTM network, because these pairs of regions are very close together, the network is able to quickly pivot to another regime if it realises that the arm it decided was optimal is suddenly no longer providing any reward. This case can happen during exploitation, which means that if the network receives no reward at a trial, it will attempt to try the other arm.

It's also worth noting that the region for 'Left always reward' isn't close to 'Right no reward', this could point to the network being unable to infer a relationship between the two arms. The network could recognise the fact that a certain arm never provides a reward but it doesn't subsequently always pick the other arm. This could change if the LSTM was trained using episodes with dependent reward probabilities (detail in methods section).

Hidden state trajectories were also plotted for actual decision making shown in Fig3.4b and Fig3.4c, results show that as the network gets through more trials and decisions the distance between the hidden states of consecutive trials decrease, slowly tending towards a fixed point. Any 'wrong' decisions made by the network led the trajectories off its intended path, these distances are much larger than the distances travelled by 'correct' decisions. This is more evident in Fig3.4c where the reward probabilities are more similar. Towards the end of the episode, the network is still unsure which arm it should exploit and sticks to exploration. In this case, the two regions of contrasting conclusions (left arm being better than the right or vice versa), are very close which means that hidden state is able to change in only a few time-steps if the network starts to exploit.

Fig3.4d further shows that there is no one region where the network decides one arm is better than the other.

## 3.2 Short-term Synaptic Plasticity Network

The next experiments that were carried out involved using various RNNs (both gated RNNs and biologically inspired ones) on the sequential MNIST classification task. This allows us to gauge the performance of our own Short-term synaptic plasticity network (STPN) and learn about its ability to hold long short-term memory, before testing its flexibility on the two armed bandit task.

Details of the methodology of the sequential MNIST and the permuted (spiral) sequential MNIST tasks can be found in the methods section.

Fig3.5 shows the spiral sequential MNIST task with varying time gaps and input sizes. Essentially allowing us to control the complexity of the classification task.

| Hidden size | Input size | Time gap | Stride |
|---|---|---|---|
| 24 | 4,8,16 | 1,4,8 | 1 |
| 48 | 4,8,16 | 1,4,8 | 1 |

Table 3.2 *Various parameters that were tested for in Fig3.5.*

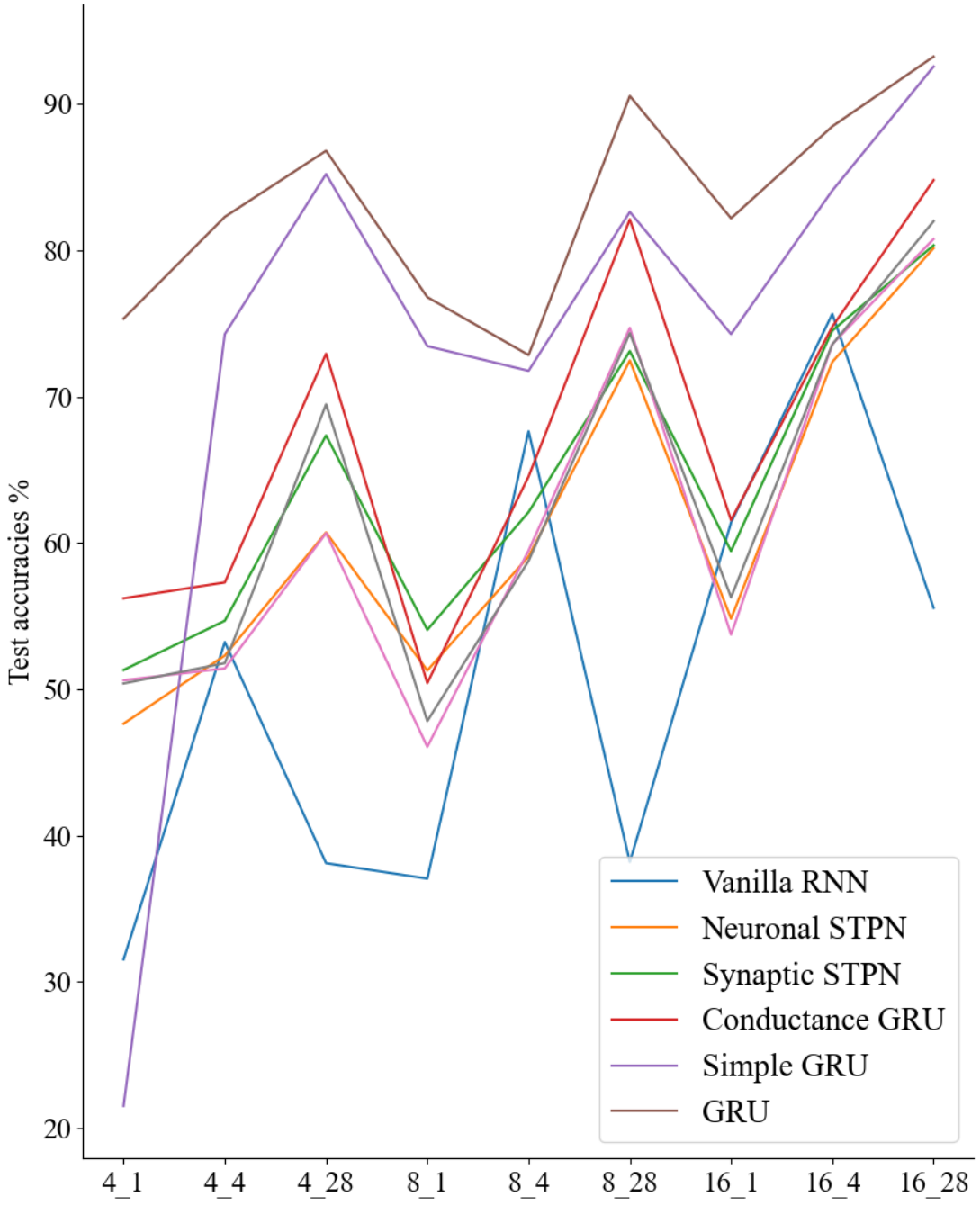



Fig. 3.5 *Spiral sequential MNIST test accuracies for RNNs with hidden size 24. The x-axis is labelled with the form 'input size_time gap'.*

Fig3.5 first shows the performance of various RNNs all with a hidden size of 24. For all following experiments involving sequential MNIST we use the vanilla RNN of hidden size 24 and the GRU with hidden size 24 as the lower and upper baselines of network performance. We can see that the vanilla RNN's general performance is lower than all the other more complex networks. Both the neuronal and synaptic STPN seem to lie in between the GRUs and the vanilla RNN in test accuracy. The shape of the lines are also interesting. By changing the input size and time gap of the task, we are able to restructure the task so that the RNNs have to maintain memory differently for each different version. Note that the shape of

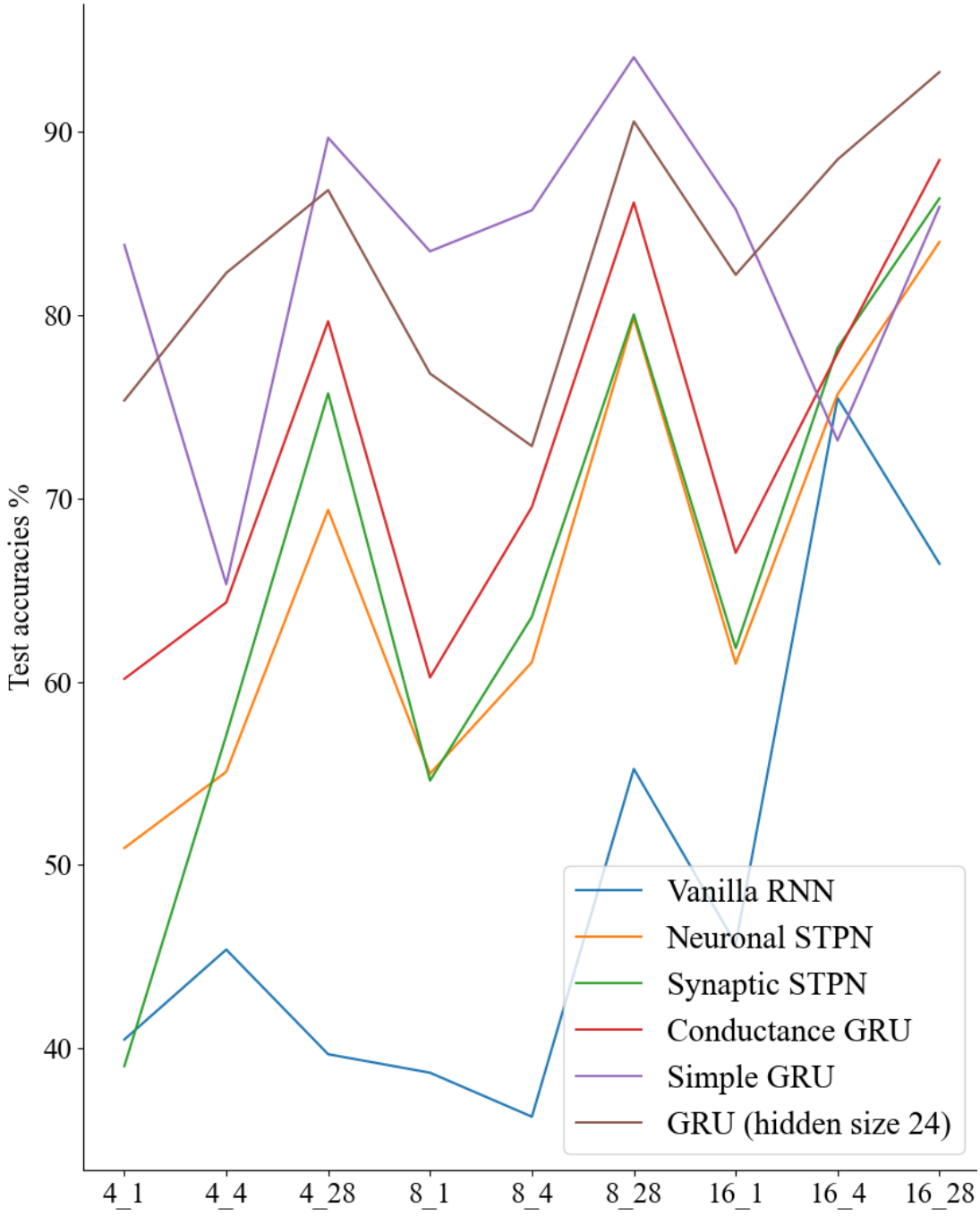



Fig. 3.6 *Spiral sequential MNIST test accuracies for RNNs with hidden size 48 (and a GRU of hidden size 24). The x-axis is labelled with the form 'input size_time gap'.*

the STPNs lines are very similar to each other, the synaptic STPN is as expected slightly better than the neuronal STPN however this small increase in performance comes at the expense of a lot more parameters.

The STPNs are also very similar in shape to the Conductance GRU (see method section for details), whereas the simple GRU and GRU are both a bit different (less steep lines in the graph shows that these two networks offer more consistent performace across different task structures). We also expect some anomalies in performance such as the simple GRU (24) at 4_1.

Fig.3.6 shows the same results except for RNNs with a hidden size of 48. The performance of the networks improved at expected,

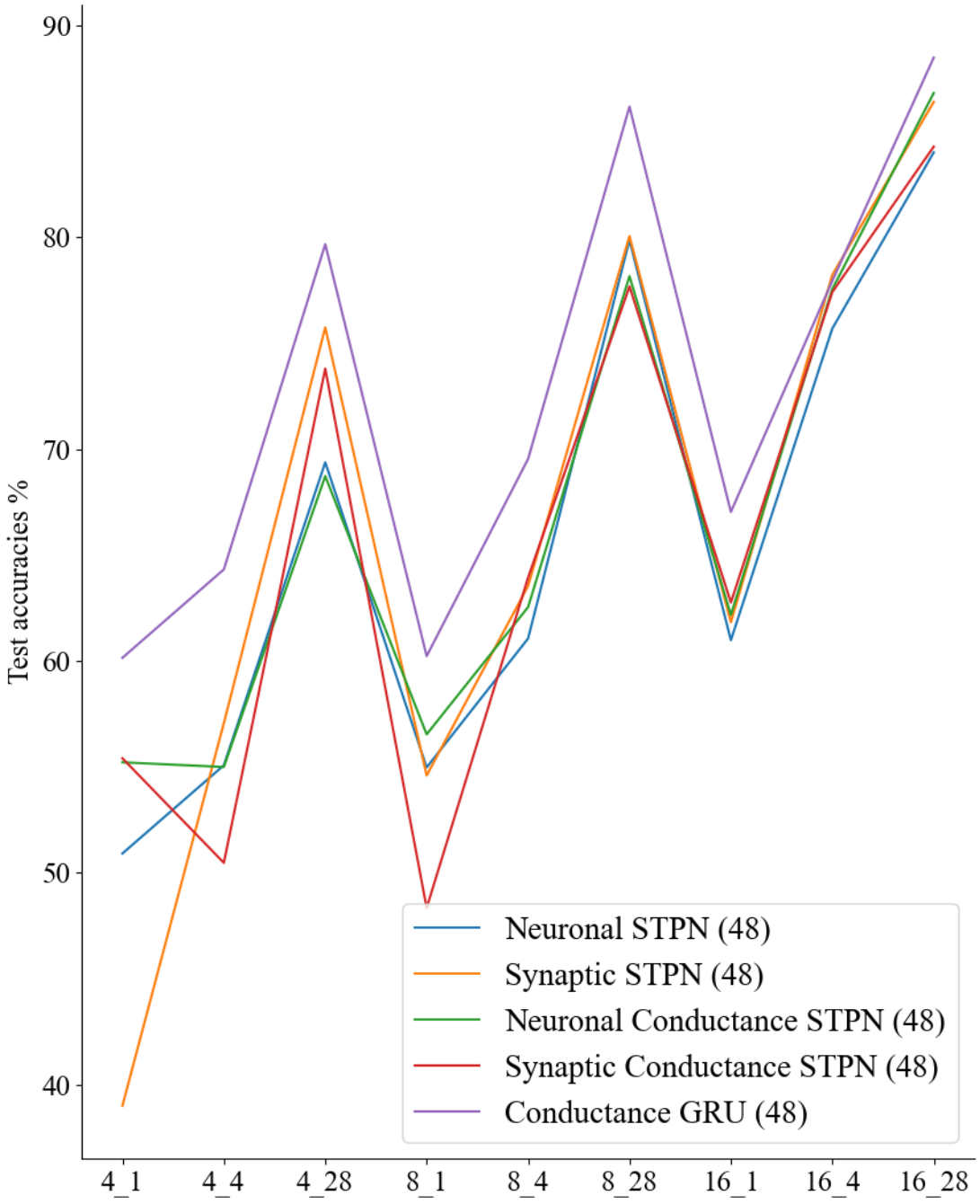



Fig. 3.7 *Spiral sequential MNIST test accuracies for STPNs with hidden size 48. All four variations of the STPN is shown for comparison. The x-axis is labelled with the form 'input size_time gap'.*

with the simple GRU even outperforming the GRU(24) at certain task structures. What stands out is the stochastic nature of the performance of the simple GRU and the vanilla RNN, the plots for these networks do not seem predictable between the two different hidden sizes. Evaluating on 10000 test images led to these networks outputting differently shaped plots between the two hidden sizes. However, this is not the case for the biologically inspired networks, as mentioned above, the shape of the neuronal STPN is very similar to that of the synaptic STPN. They are also very similar to the Conductance GRU which is surprising since

the equations for the Conductance GRU is quite different from STPNs (details in methods section). It's encouraging that this similar shape for these networks are carried through to the plots for hidden size 48. Larger networks than 48 were not tested due to computational costs.

The Conductance STPN essentially combines the equations of the Conductance GRU and the STPNs. The equation for $U$ in the Conductance STPN used was:

$$U = 0.99\sigma(c_U)$$

In this case, the hyperparameters of the Conductance STPN are adjusted slightly. $c_U$ is changed from zero initialisation to 2 and $U_{\max}$ is initialised to 0.99 instead of 0.9. This means that the network is initialised in a form where there is very little facilitation because $u$ can only move between 0.99 and 1. $c_U$ being initialised at 2 means that it starts at a fairly flat point on the sigmoid.

During training, the network should update the STP parameters so that it either gets rid of facilitation and depression entirely so that the network behaves like a Conductance GRU or optimise the STP capabilities of the network so that performance increases above that of the Conductance GRU.

Despite these changes made to the initialisation, the Conductance STPN didn't prove to be much of an improvement and instead remained quite similar to its standard STPN counterpart. Disappointingly, the Conductance STPNs also performed slightly worse than the Conductance GRU, which is unexpected because the Conductance GRU is technically a subset of the Condunctance SPTN. This could be because the Conductance STPN is difficult to train due to its high number of variables.

### 3.2.1 STP variables

Once training and evaluation were completed, the STP variables of the networks were plotted. By plotting the time constants of both facilitation and depression we are able to see the properties of the neurons in the network and whether or not they have biological implications. Comparison plots were also made between various synaptic and neuronal variables.

Fig.3.8 and Fig.3.9 show the STP variables and comparison graphs for 2 different networks of varying sizes.

By plotting the heat maps of $\tau_F$ and $\tau_D$, we are able to see the range of time constants in the synapses of the network. The larger $\tau_F$ becomes, the more facilitating the synapse is, this corresponds to a synapse having a longer time constant which means $u$ takes more time-steps to decay to $U$. On the other hand, the larger $\tau_D$ becomes, the more depressive a synapse gets.

$U$ also plays a role in the behaviour of the network, the larger $U$ becomes the more depressing a synapse gets. As U gets larger and closer to 1, the range of $u$ decreases. If $U = 1$, a synapse ceases to display any facilitation.

We can see from the heat maps that the range for $\tau_F$ is from 0 to 500 which is much larger than that of $\tau_D$, which is only from 0 to 150. A synapse can be defined as a facilitating synapse if $\tau_D$ is much larger than $\tau_F$. From part 'h' of the figures we can see that some synapses have become specialised. No synapse has a high value (relative to the other synapses) of both $\tau_F$ and $\tau_D$, the data in h seems to be close to either axes.

There also seems to be a relationship between $\tau_F$ and $U$ shown in 'g' of the figures. The synapses with very high values of $\tau_F$ also have the highest values of U. This implies that facilitating synapses have less range for $u$ to

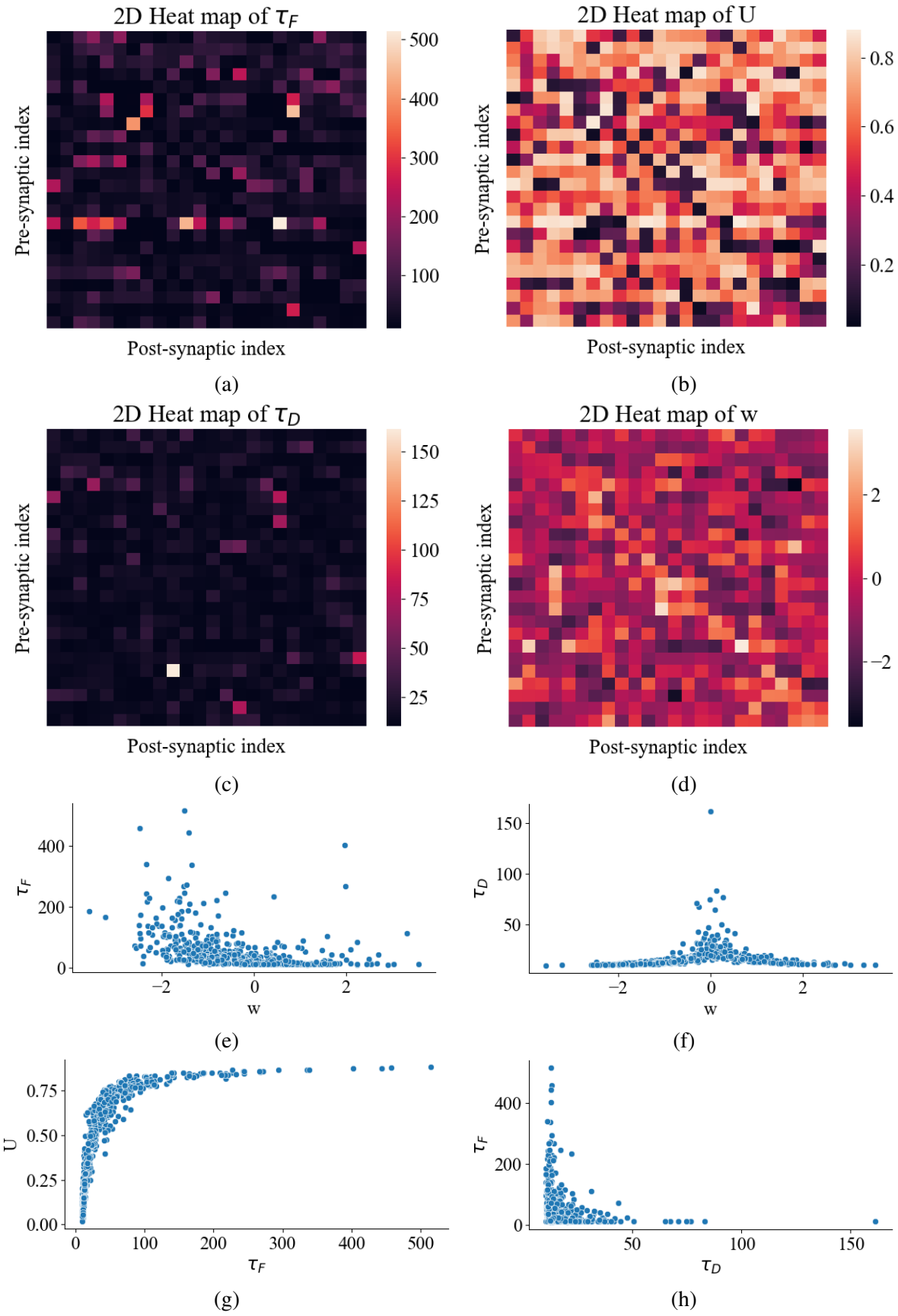



Fig. 3.8 *Plots of the parameters of a synaptic STPN (hidden size 24) trained on input size 8 with a time gap of 28 (test accuracy of* 73.12%*.* $\tau_F$ *and* $\tau_D$ *are the inverses of* $z_u$ *and* $z_x$ *used in the network equations and represent the time constants of short-term facilitation and depression respectively.*

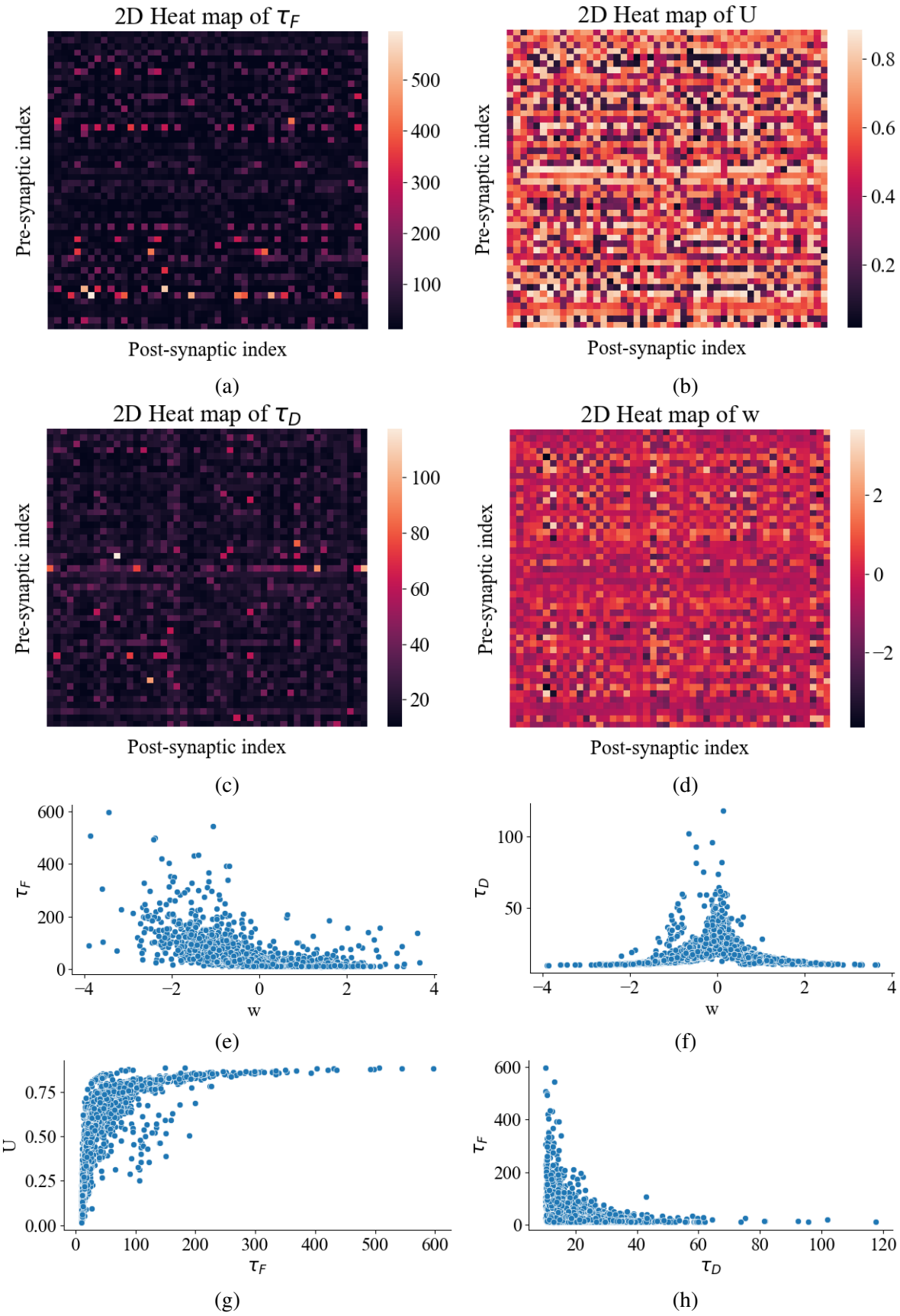



Fig. 3.9 *Plots of the parameters of a synaptic STPN (hidden size 48) trained on input size 4 with a time gap of 4 (test accuracy of 57.07%. $\tau_F$ and $\tau_D$ are the inverses of $z_u$ and $z_x$ used in the network equations and represent the time constants of short-term facilitation and depression respectively.*

move between but decays much more slowly so can hold onto information for longer.

We also compared STP properties to the weights $w$. Synapses with large $\tau_F$ all correspond to a negative weight in $w$. Additionally, synapses with large $\tau_D$ correspond to elements in $w$ with a small magnitude.

It is also possible to consider neuronal variables such as $p$. Taking the weights $p$ and calculating the Euclidean norm along the neurons returns the magnitude of input weights. These points can be plotted against magnitude of output weights of each neuron which is similarly obtained from the weights of the linear layer connected to the STPN in the network.

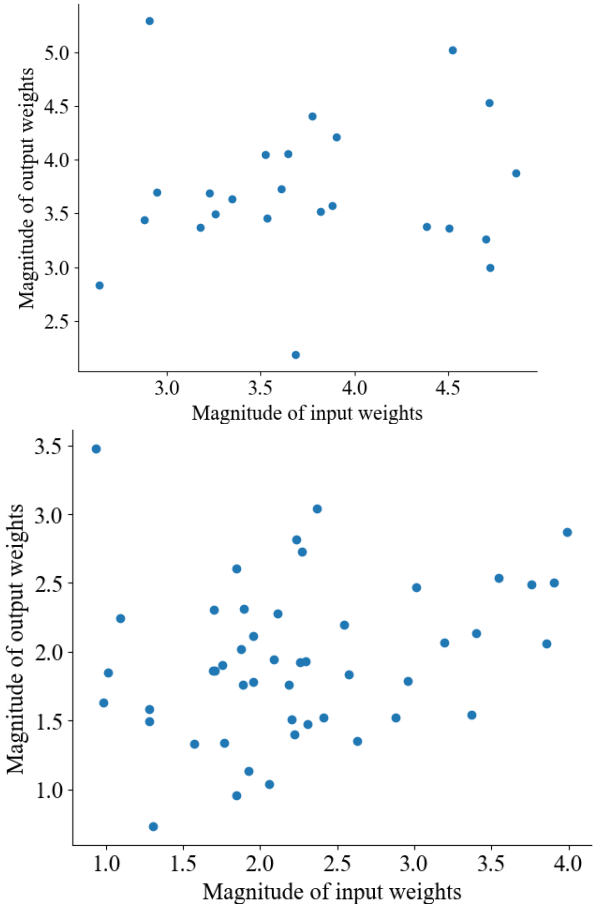



Fig. 3.10 *Magnitude of input weights against output weights in an STPN network used for the spiral MNIST task. Two plots for different tasks structures (8_28 and 4_4) and different hidden sizes (24 and 48).*

From the figures, there doesn't appear to be a noticeable distinction between the different neurons in the network. There are no neurons with a much stronger output weight than input weight and vice versa.

### 3.2.2 Dynamic plots

Besides observing the time constants and weights of the trained STPNs, dynamic plots of $u$ and $x$ were made to show the behaviour of STP on an image from MNIST.

This involved using a specific image, passing it to a trained network and tracking the dynamics of $u$, $x$ and the hidden state $h$ at each time-step.

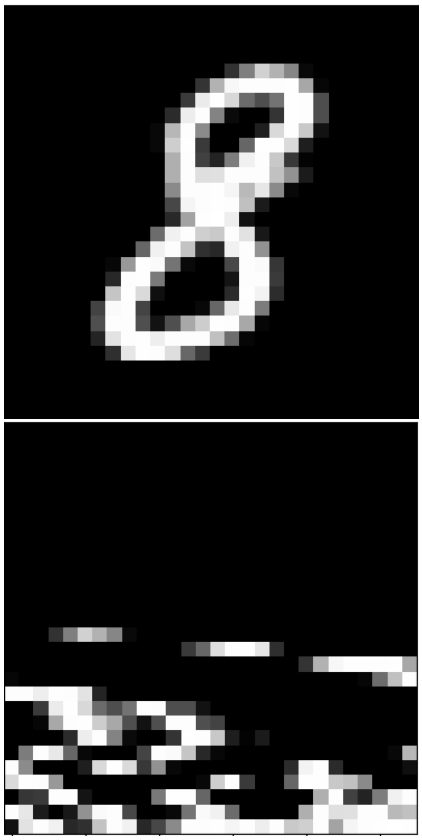
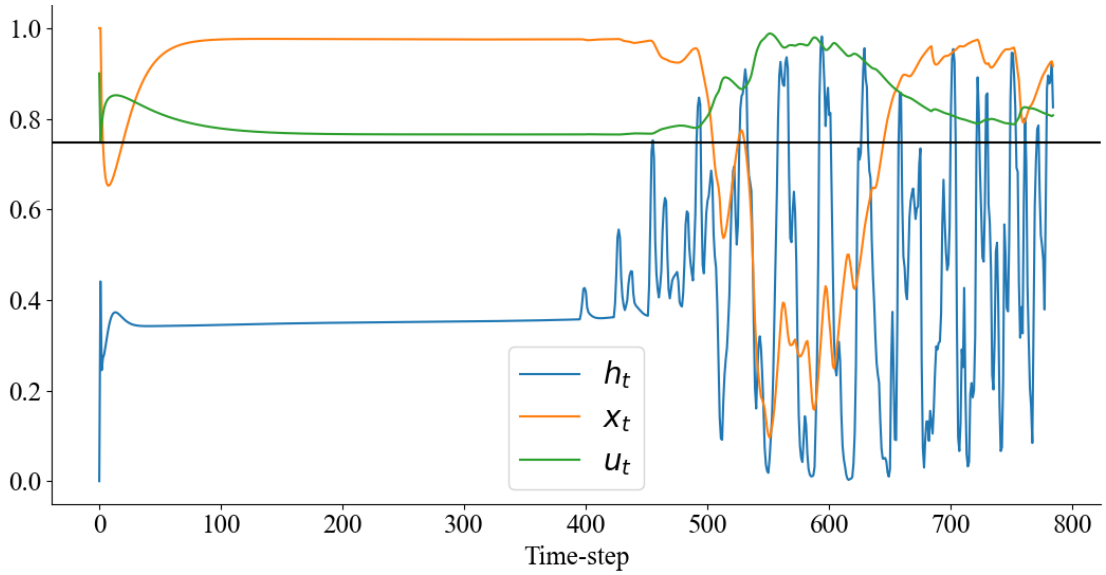
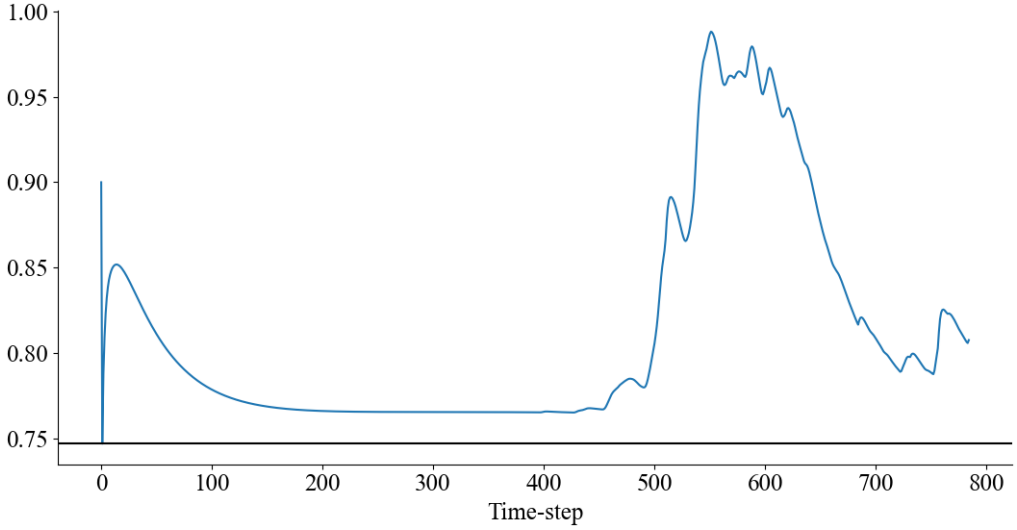


Fig. 3.11 *MNIST image used for the dynamic plots and spiral form of the image.*

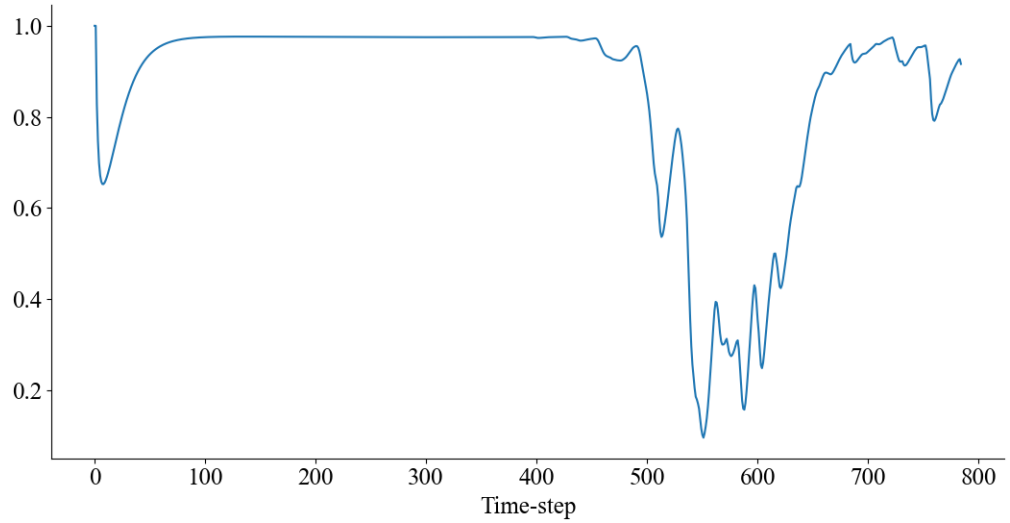
The spiral form of the image used provides a relatively easy task for classification because the bright pixels holding the information are re-




(a)

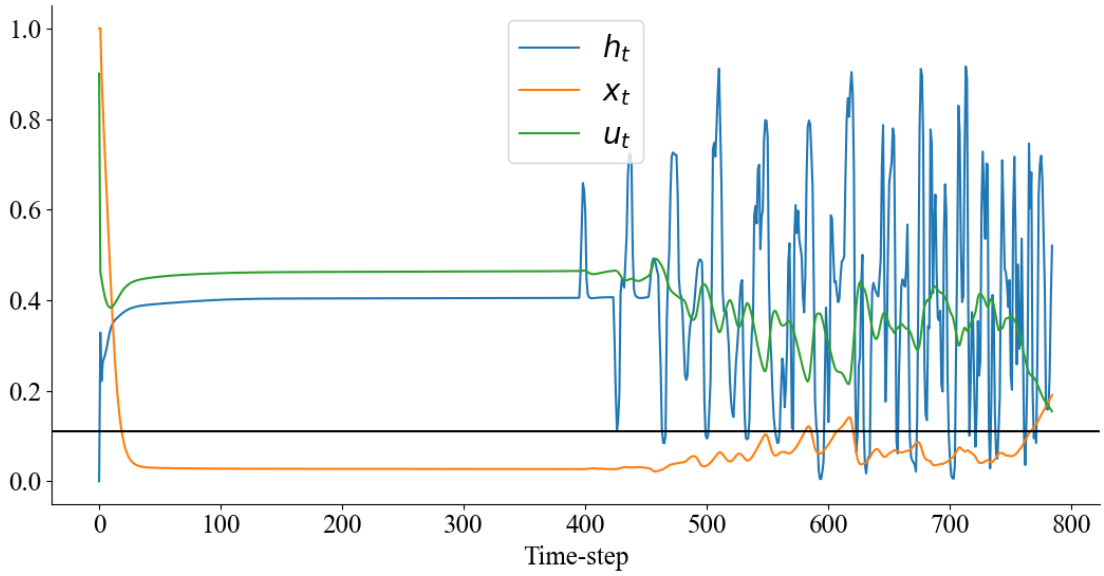



(b)

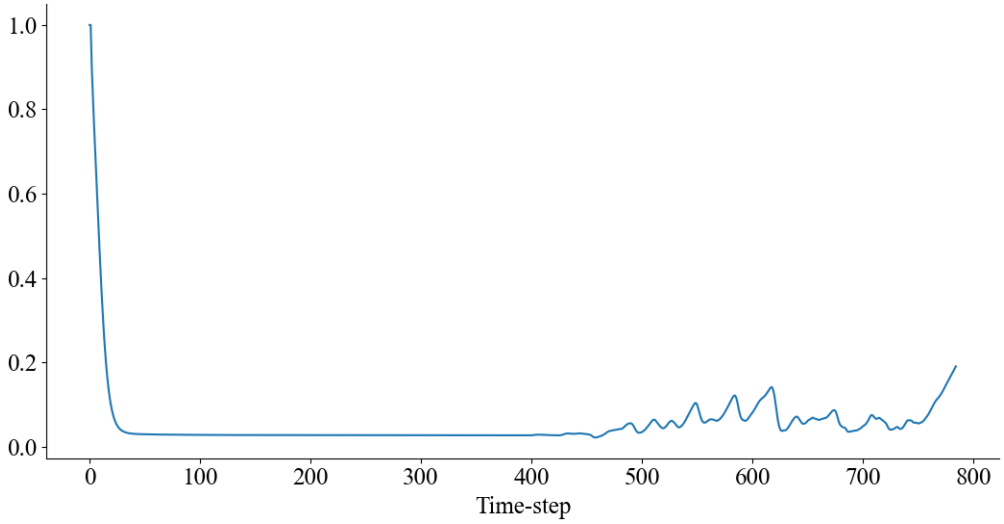



(c)

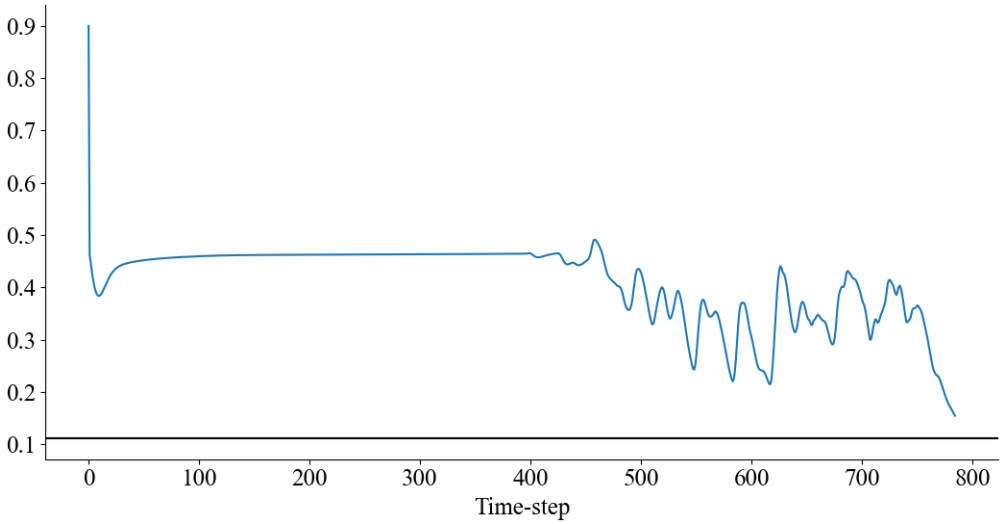
Fig. 3.12 *Dynamic plots of STPN of hidden size 24 (input size 8, time gap 28). a)* $x_t$ *and* $u_t$ *of a random synapse and the corresponding neuron's* $h_t$ *against time-step. Black horizontal line shows* $U$ *of the synapse. b)* $x_t$ *(*$\tau_D = 14.40$*). c)* $u_t$ *(*$\tau_F = 45.32$*).*




(a)

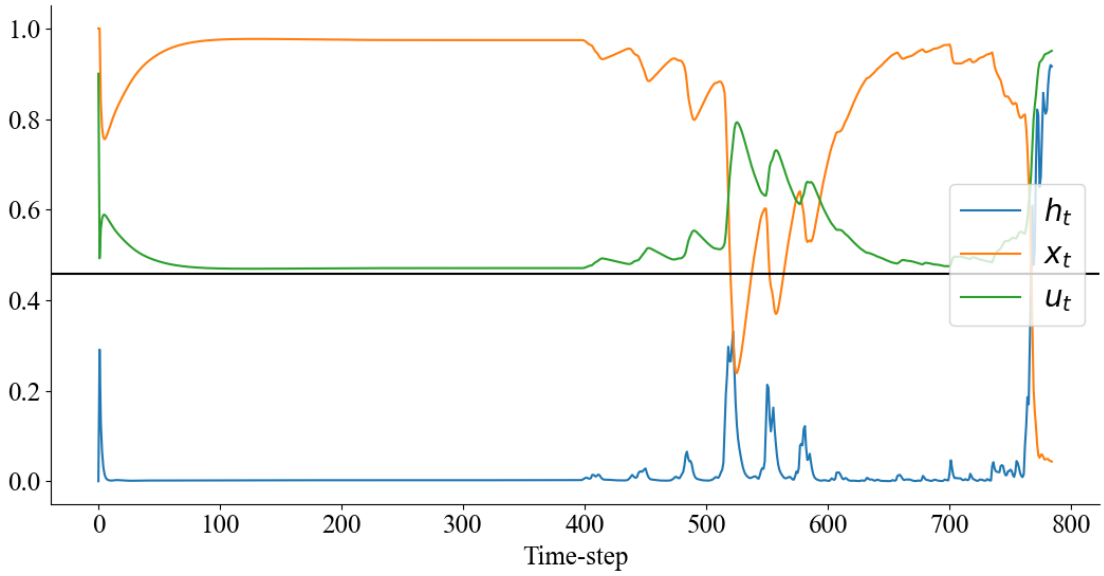



(b)

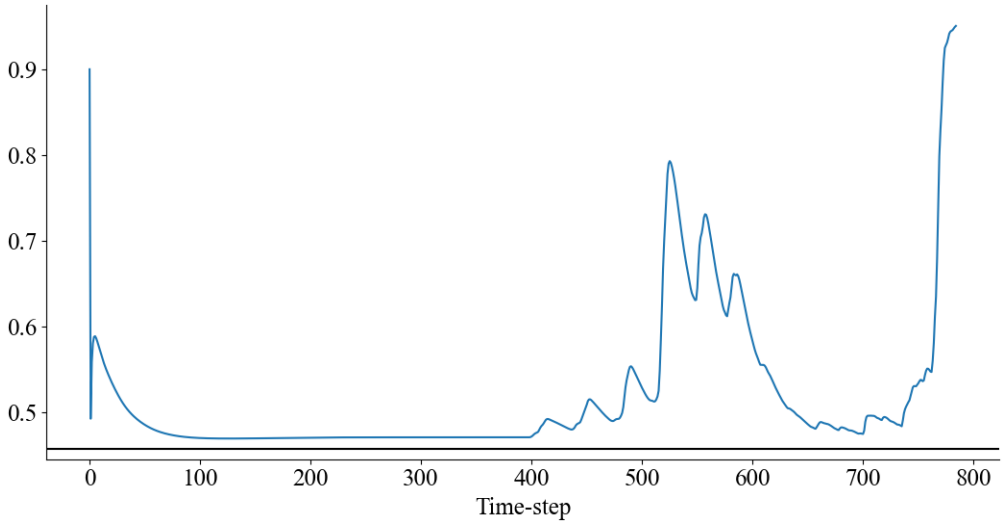



(c)

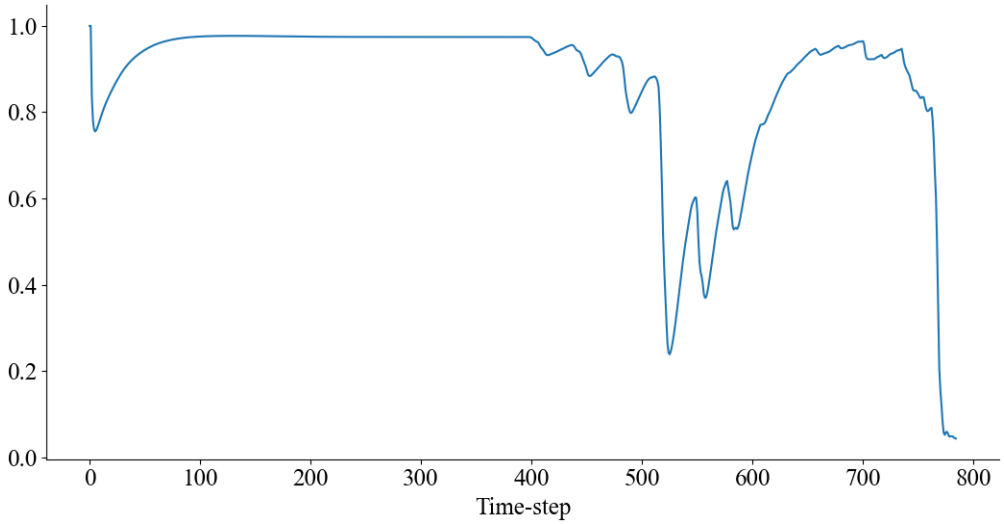
Fig. 3.13 *Dynamic plots of STPN of hidden size 24 (input size 8, time gap 28). a)* $x_t$ *and* $u_t$ *of a random synapse and the corresponding neuron's* $h_t$ *against time-step. Black horizontal line shows* $U$ *of the synapse. b)* $x_t$ *(*$\tau_D = 161.45$*). c)* $u_t$ *(*$\tau_F = 12.34$*).*




(a)




(b)




(c)

Fig. 3.14 *Dynamic plots of STPN of hidden size 48 (input size 4, time gap 4). a) $x_t$ and $u_t$ of a random synapse and the corresponding neuron's $h_t$ against time-step. Black horizontal line shows $U$ of the synapse. b) $x_t$ ($\tau_D = 25.19$). c) $u_t$ ($\tau_F = 24.53$).*

arranged to be closer to the end. Thus decreasing the amount of memory a network needs to hold.

Because the spiral form of the image is all black (inputs are 0) for the first half, we see that the dynamic variables decayed and settled at a steady state at the beginning of both sets of plots. Once the brighter pixels were reached, the variables start to move more dynamically for the second half of the time-steps.

We can see that $u_t$ and $x_t$ seem to move inversely to each other, as one increases the other decreases. This is more obvious in Fig.3.14 since the time constants for both facilitation and depression are very similar ($\tau_D = 25.19$ and $\tau_F = 24.3$). Fig.3.12 features a more facilitating neuron with the facilitation time constant being larger than the depression time constant by around a factor of 3. ($\tau_D = 14.40$ and $\tau_F = 45.32$. We can see from the figures that $u$ is much more stable than $x$ because it decays slower.

### 3.2.3 Two-armed Bandit task with STPNs

While the performance of the GRU trained on the Two-armed Bandit task (using the same reinforcement learning algorithm as the training of the LSTM) wasn't very impressive, the network still shows meta-learning by adapting differently to each episode. This was also true for the simple GRU, during training, actions taken per episode were tracked and the simple GRU showed different behaviour between different episodes.

At the start of training, network behaviour would be very similar regardless of reward probability. This meant that for the early stages of training, networks would always pick a certain arm some number of times and this stayed constant between consecutive episodes. As weights were updated the number of times a network picked a certain arm changed but the behaviour between consecutive episodes were still very similar.

However, for the networks capable of meta-learning, after around 500 gradient updates, the behaviour between consecutive episodes became very different as the network started to adapt to the changing reward probabilities.

This observed phenomenon during training did not however occur in experiments using STPNs. All variations of the STPN were used for the Two-armed bandit task and the results showed that these networks reached a point where they would exclusively pick one arm over the other regardless of reward probabilities for a certain episode.

Even the Conductance STPN which uses a dynamic gate outside of the STP variables was not able to display flexible behaviour.

## 3.3 Conclusion

The two main aims of the research project was to assess the performance and behaviour of RNNs on their flexibility and long short-term memory as well as making artificial RNNs more biologically plausible and checking that these networks can still perform as expected.

While efforts to replicate the results from Wang et al [13] proved unsuccessful, using a similar method on a network focused around an LSTM of hidden size 48 did result in meta-learning from the agent. Flexible behaviour such as this was also observed in the GRU and simple GRU. However the performance of the LSTM was much better than these two networks for the Two-armed bandit task, most

likely due to the LSTM having an extra cell state $c_t$ which specifically codes for long-term memory. This allowed the LSTM to explore both arms and even enabled it to adapt its behaviour to react to sudden changes in reward probabilities.

The biologically plausible recurrent neural networks inspired by synaptic short-term plasticity were also not able to display any form of meta-learning similar to the vanilla RNN. This result is disappointing due to the synaptic STPN actually having more parameters than the GRU. Flexible behaviour is perhaps difficult to achieve for STPNs due to a lack of holding onto long-term memory. Nevertheless, evaluating all variations of the STPN on the (permuted sequential) MNIST dataset showed that these networks do perform fairly well in tasks requiring long short-term memory. It's encouraging that STPNs, both the neuronal and synaptic variations performed significantly better than the vanilla GRU in these tasks.

These biologically inspired networks also displayed very similar behaviour to each other for all combinations of input size and time gap for the spiral sequential MNIST task. For all experiments carried out, different STPNs showed the same pattern for all 9 task structures, which is not observed in either the vanilla RNN nor the GRU.

Moreover, analysing the trained variables as well the dynamics of STPNs displayed interesting relationships between the facilitation time constants $\tau_F$, depression time constants $\tau_D$ and $U$ of different synapses in the network. Results showed that during training, some synapses began to become more specialised, either turning more facilitating or more depressing.

To futher explore the performance of STPNs, future work could introduce variations in training such as a different optimizer, truncated backpropagation or simply a longer training time. This is especially true for Conductance STPNs since they weren't able to exceed the test accuracies of Conductance GRUs on the MNIST dataset using our training methods. Even though Conductance GRUs are technically a subset of all possible Conductance STPNs.

Furthermore, all the experiments that were ran in our research used single layer RNNs, it could be worth investigating how STPNs behave with multiple layers in future.

## 3.4 Risk Assessment

Due to this project being mostly computational, there were no significant risks that needed to be prevented or avoided.

# Chapter 4

# Methods

## 4.1 Two-armed bandit task

The simulations of the two-armed bandit task employed a common set of methods for multiple different networks. However, the initial agent architecture centred around a fully connected, LSTM. In all experiments, the input is a vector of length 4 consisting of the reward received on the preceding timestep, a one-hot representation of the action taken on the preceding timestep and the current timestep [12]. Recall that timesteps are limited to the maximum number of trials in each episode. The output consisted of a scalar baseline (value function) and a real vector with length 2 (number of available actions). Actions were then sampled from the softmax distribution defined by this vector.

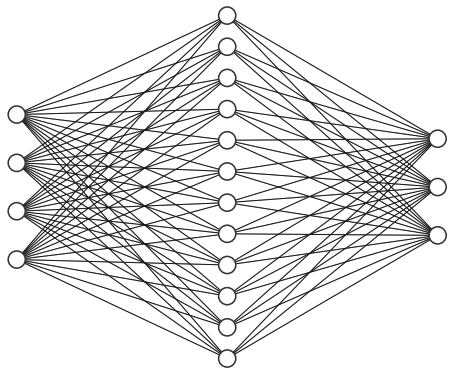



Fig. 4.1 *General structure of the network with a hidden size of 12.*

In Fig.4.1, in between the input layer and hidden layer is an RNN, the hidden layer $h_t$ off the RNN is used at each time-step to form the output layer.

Training of the network used reinforcement learning and was implemented by Advantage Actor-Critic (A2C). Details of training including the use of entropy regularization and a combined policy and value estimate loss, are described by Mnih et al.

Details of training an LSTM for the Two-armed bandit task is heavily influenced by Wang et al.To summarise, the gradient of the full objective function is the weighted sum of the policy gradient, the gradient with respect to the state-value function loss, and an entropy regularization term, these are described by the equations below:

$$\begin{aligned}\nabla\mathscr{L} &= \nabla\mathscr{L}_\pi + \nabla\mathscr{L}_v + \nabla\mathscr{L}_{\text{ent}} \\ &= \frac{\partial \log \pi(a_t|s_t;\theta)}{\partial\theta}\delta_t(s_t;\theta_v) \\ &\quad + \beta_v\delta_t(s_t;\theta_v)\frac{\partial V}{\partial\theta_v} \\ &\quad + \beta_e\left[\frac{\partial H(\pi(a_t|s_t;\theta))}{\partial\theta}\right]\end{aligned}$$

$$\delta_t(s_t;\theta_v) = [R_t - V(s_t;\theta_v)]$$

$$R_t = \sum_{i=0}^{k-1}[\gamma^i r_{t+i}] + \gamma^k V(s_{t+k};\theta_v)$$

where

- $a_t$ is action
- $s_t$ is state
- $R_t$ is the discounted $n$-step bootstrapped return at time $t$
- $k$ is the number of steps until the next terminal state and is upper bounded by the maximum unroll length $t_{\max}$ In the simulations carried out, the maximum unroll

length is set to the maximum number of trials per episode.

- $\pi$ is the policy parametrized by the parameters $\theta$
- $V$ is the value function which is parametrized by $\theta_v$. This means that instead of being defined by a function, the policy and value function estimation are defined by the synaptic weights and biases of the neural network. During training these parameters are optimised and updated.
- $H(\pi)$ is the entropy of the policy
- $\beta_v$ is a hyperparameter which controls the relative contribution of the value estimate loss
- $\beta_e$ is similar to $\beta_v$ except it controls the relative contribution of the entropy regularization term.
- $\delta_(s_t;\theta_v)$ is the $n$-step return temporal-difference error that provides an estimate of the advantage function.

The parameters $(\theta, \theta_v)$ are updated using gradient descent and backpropagation through time. The optimiser used was RMSprop. Weight updates occurred at the end of every episode.

The RNNs used have various hidden sizes, the baseline LSTM used had a hidden size of 48, and different RNN architectures such as GRUs had larger hidden sizes in order to assess whether a larger hidden size can compensate for the higher number of gates and the cell state in LSTMs.

### 4.1.1 Task structure

The task structure of the simulations were all episodic. At the beginning of each experiment, the total number of episodes and the total number of trials per episode were set. (E.g. 20000 episodes where each episode has 100 trials).

Then, at the beginning of each episode, the arm probabilities are set and held constant for that episode. Arm probabilities were sampled independently from a uniform distribution. This means that tasks within certain episodes will be more difficult than others. Episodes where the arm probabilities differ more (e.g. $p_0 = 0.9, p_1 = 0.1$) will be more difficult than episodes where the arm probabilities are similar (e.g. $p_0 = 0.4, p_1 = 0.5$). When the arm probabilities are similar, it takes more exploration to find the arm with the higher probability.

Independence means that $p_0$ and $p_1$ do not necessarily have to sum to 1. This is more difficult to solve because exploring one arm does not provide any information about the other arm.

| Parameter | Values |
|---|---|
| No. LSTMs | 1 |
| No. hiddens | 48 |
| Steps unrolled | 100 |
| $\beta_e$ | 0.5 |
| $\beta_v$ | 0.5 |
| Learning rate | 7e-4 |
| Discount factor | 0.75 |
| No. trials/episode | 100 |
| Total episodes | 20000 |

Table 4.1 *List of hyperparameters used during training of LSTM.*

At the beginning of each episode, the hidden state and the cell state if the RNN is an LSTM are set to 0. Essentially representing the

fact that the agent has 0 knowledge of the task structure.

For the initial experiment the parameters and hyperparamters are specified as shown in table 1. Hyperparameters used in the simulation are taken from Wang et al.

Once training has completed, the network weights are fixed which means that the only properties of the LSTM that can change are the dynamic states (hidden state and cell state). Because the weights of the network are now fixed, it can no longer learn using the reinforcement learning algorithm that was used during gradient updates.

The next stage of the experiment was then to use the network on a fixed number of episodes with the same number of trials as the training process. The reward probabilities for these "meta trials" were sampled the same way as they were during training (independent uniform distribution). Hidden states are again initialised to 0 at the beginning of every episode.

## 4.2 Short-term Synaptic Plasticity Network

The Short-term Synaptic Plasticity network we've created in order to assess the validity of using biological features on artificial recurrent neural networks is directly derived from the Tsodyks and Markram STP model.

In order to convert the dynamics of the model into an artificial network, the continuous equations must be converted to discrete versions of themselves. This is carried out using the Euler method.

We start from the version of the Tsodyks and Markram model with $u$ decaying to $U$.

$$\frac{du}{dt} = -\frac{U-u}{\tau_F} + U(1-u)\delta(t-t_{sp}),$$

$$\frac{dx}{dt} = \frac{1-x}{\tau_D} - ux\delta(t-t_{sp}),$$

In an artificial neural network the rate is expressed by the hidden state ($h_t$), substituting this for $\delta(t-t_{sp})$ and then discretising produces the equations:

$$x_t = z_x + (1-z_x)x_{t-1} - (\Delta t \odot u_t \odot x_{t-1})h_{t-1}$$

$$u_t = Uz_u + (1-z_u)u_{t-1} + (\Delta t \odot U \odot (1-u_{t-1}))h_{t-1}$$

where $\odot$ denotes the Hadamard product which is element-wise multiplication between matrices. The parameters $z$ are correlated with the reciprocals of the STP time constants so that $z_x = \frac{\Delta t}{\tau_D}$ and $z_u = \frac{\Delta t}{\tau_F}$.

The resulting discrete equations are used in an RNN instead of gates in networks such as GRUs and LSTMs.

As specified in the Tsodyks and Markram model, the resulting synaptic efficacy is obtained by $Aux$. Therefore $u$ and $x$ will be used in the network to alter the synaptic weight $W$ of the RNN since $W$ mimics synaptic strength. This produces the final rate equation of the STPN: (* Note that since $x$ is used to represent STD in the STPN, capital $X$ will be used for the input.)

$$h_t = (1-z_h)h_{t-1} + z_h \odot \varphi((u_t \odot x_t \odot W)h_{t-1} + PX_t + b)$$

Where $\varphi()$ is the activation function and in the STPN is set to be the sigmoid() function. This is done to make the network more biological because $h_t$ represents the rate and therefore must

be $\geq 0$. Recall that in gated RNNs, the activation function is set to tanh which isn't biological because rates can be negative.

- $\Delta t = 1$ can be set if we define a single unit of time to be 5ms by convention. This then means that the maximum rate which is 1 ($\max(h_t) = \max(\sigma()) = 1$), can be interpreted as $\frac{1}{5\text{ms}} = 200\text{Hz}$. This fits in well with biology.

- The three parameters corresponding to the reciprocal of recovery time $z_h, z_x$ and $z_u$, are different for all three processes because different processes will have different recovery times. Recall that $z$ was also used in the GRU as the Update gate, the main difference is that the update gate $z_t$ is dynamic and changes over time depending on the hidden state and the input. Whereas in the STPN, $z$ is a fixed trained parameter.

- For biological accuracy, the $z$ parameters used in STPN also all have to be $\geq 0$ because they represent time. In order to achieve this, $z$'s are constrained by using sigmoid. Different hyperparamers are set to define the maximum and minimum values for $z$'s.

| Parameter | Values |
|---|---|
| $z_h^{\min}$ | 0.01 |
| $z_h^{max}$ | 0.9 |
| $z_{ux}^{min}$ | 0.001 |
| $z_{ux}^{max}$ | 0.1 |

  During training, constraints are defined by the equations:

$$z_h = z_h^{min} + (z_h^{max} - z_h^{min})\sigma(c_h)$$

$$z_x = z_{ux}^{min} + (z_{ux}^{max} - z_{ux}^{min})\sigma(c_x)$$

$$z_u = z_{ux}^{min} + (z_{ux}^{max} - z_{ux}^{min})\sigma(c_u)$$

  where $c_h, c_x$ and $c_u$ are parameters that are updated during training.

- Since $z$'s are the reciprocals of $\tau$'s, as $z$ approaches 1, this indicates that $\tau$ approaches 1s. The corresponding $\tau$'s are shown in the table below.

| Parameter | Values |
|---|---|
| $\tau_h^{\max}$ | 100s |
| $\tau_h^{\min}$ | 1.11s |
| $\tau_{ux}^{\max}$ | 1000s |
| $\tau_{ux}^{\min}$ | 10s |

- The dynamics of the continuous forms of the STP equations ensure that the values of $x$ and $u$ are constrained to be $0 \leq x \leq 1$ and $U \leq u \leq 1$. This is because the sign of the rate adjusts as each variable approaches either limit. However, in the discrete form, at a particular time-step, if the rate is high in magnitude then $u$ and $x$ can update outside the limits. To rectify this issue, the values of $u$ and $x$ are clipped in the model to fit between the limits.

- $U$ is also a trainable parameter in the model. Since it represents the minimum value of $u$, it must be constrained between 0 and 1. This can be done by parameterising U as:

$$U = 0.9\sigma(...)$$

  where 0.9 is decided as the maximum U can take (0.9 is the maximum of the lower limit of $u$).

- $x_t$ is initialised to be $\mathbf{1}$ and $u_t$ is initialised to be $U$ which itself is initialised to $\mathbf{0.9}$. $c$'s are all initialised to 0 so that the output of the sigmoid lies in the middle of the sigmoid function.

### 4.2.1 Synaptic and Neuronal STPN

In the simulations completed with the STPN, there were two main types of the network that were used. In the more complex version, a "rich" model containing more parameters defines $x$ and $u$ as matrices, these matrices are the same size as the synaptic weight of the network, $W$. In the simpler version, a "poor" model is less computationally expensive by only defining $x$ and $u$ as vectors of length *hidden size*.

In the rich model, it can be interpreted that each synapse has its own values of $x$ and $u$, hence *synaptic* STPN. Conversely, the poor model interprets that each neuron has its own values of $x$ and $u$, therefore it is known as the *neuronal* STPN.

Due to these two different models using different dimensions for its key parameters, the equations are modified slightly for the neuronal model. The dimensions of the dynamic equations for both versions of the STPN are expressed in tables 4.2 and 4.3.

For the equations of synaptic STPNs, $x_t^{ij}$ can be used to show that $x_t$ is a matrix. $i$ being the rows of the matrix and $j$ being the columns. If the hidden size is $d$:

$$x_t^{ij} = \underbrace{\begin{bmatrix} x_t^{11} & x_t^{12} & x_t^{13} & x_t^{14} & \cdots & x_t^{1d} \\ x_t^{21} & x_t^{22} & x_t^{23} & x_t^{24} & \cdots & x_t^{2d} \\ x_t^{31} & x_t^{32} & x_t^{33} & x_t^{34} & \cdots & x_t^{3d} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_t^{d1} & x_t^{d2} & x_t^{d3} & x_t^{d4} & \cdots & x_t^{dd} \end{bmatrix}}_{j} \Bigg\} i$$

Conventions:
i: postsynaptic
j: presynaptic

Similarly for the neuronal STPN, $x_t^i$ can be used, taking $x_t^i$ to be a neuronal vector means that all other $j$ columns are removed. We interpret this as $j$'s being the post-synaptic index meaning in the neuronal model, the STP dynamic variables are only dependent on the pre-synaptic activity of a neuron. This fits with STP theory.

$$x_t^i = \underbrace{\begin{bmatrix} x_t^1 \\ x_t^2 \\ x_t^3 \\ \vdots \\ x_t^d \end{bmatrix}}_{1} \Bigg\} i$$

This notation is very similar for $u_t$.

Note that by convention, each row of a matrix should correspond to a fixed post-synaptic neuron and elements of that row correspond to the incoming synapses. Columns correspond to synapses going out of the neuron. Under this convention, the neuronal STPN should use $x_t^j$ which is a horizontal vector.

However, because the hidden state used in gated RNNs is typically a vertical vector, the notation described here is used instead of neuroscience convention to keep all vectors in the network equations vertical.

2D colour maps of variables of STPNs from the experiments carried out are however plotted according to regular convention. A note of clarification has been included in such cases.

### 4.2.2 Conductance STPN

The standard STPN network appears very different to gated RNNs such as GRUs. In the GRU, the Reset gate and the Update gate are both dynamic since they are dependent on the hidden state $h_{t-1}$ as well as the input. $x_t$ and $u_t$ in the STPN are both also dynamic in time

Table 4.2 Dimensions of synaptic STPN computations

$$\begin{bmatrix} \\ x_t^{d\times d} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ z_x^{d\times d} \\ \\ \end{bmatrix} + \left( \mathbf{1} - \begin{bmatrix} \\ z_x^{d\times d} \\ \\ \end{bmatrix} \right) \odot \begin{bmatrix} \\ x_{t-1}^{d\times d} \\ \\ \end{bmatrix} - \left( \left( \begin{bmatrix} \\ u_t^{d\times d} \\ \\ \end{bmatrix} \odot \begin{bmatrix} \\ x_{t-1}^{d\times d} \\ \\ \end{bmatrix} \right) \odot \begin{bmatrix} \\ h_{t-1}^{d\times 1} \\ \\ \end{bmatrix} \right)$$

---

$$\begin{bmatrix} \\ u_t^{d\times d} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ U^{d\times d} \\ \\ \end{bmatrix} \odot \begin{bmatrix} \\ z_u^{d\times d} \\ \\ \end{bmatrix} + \left( \mathbf{1} - \begin{bmatrix} \\ z_u^{d\times d} \\ \\ \end{bmatrix} \right) \odot \begin{bmatrix} \\ u_{t-1}^{d\times d} \\ \\ \end{bmatrix} + \begin{bmatrix} \\ U^{d\times d} \\ \\ \end{bmatrix} \odot \left( \mathbf{1} - \begin{bmatrix} \\ u_{t-1}^{d\times d} \\ \\ \end{bmatrix} \right) \odot \begin{bmatrix} \\ h_{t-1}^{d\times 1} \\ \\ \end{bmatrix}$$

---

$$\begin{bmatrix} \\ h_t^{d\times 1} \\ \\ \end{bmatrix} = \left( \mathbf{1} - \begin{bmatrix} \\ z_h^{d\times 1} \\ \\ \end{bmatrix} \right) \begin{bmatrix} \\ h_{t-1}^{d\times 1} \\ \\ \end{bmatrix} + \begin{bmatrix} \\ z_h^{d\times 1} \\ \\ \end{bmatrix} \odot \sigma \left( \left( \left( \begin{bmatrix} \\ u_t^{d\times d} \\ \\ \end{bmatrix} \odot \begin{bmatrix} \\ x_t^{d\times d} \\ \\ \end{bmatrix} \odot \begin{bmatrix} \\ W^{d\times d} \\ \\ \end{bmatrix} \right) \begin{bmatrix} \\ h_{t-1}^{d\times 1} \\ \\ \end{bmatrix} \right) + \begin{bmatrix} \\ P^{d\times i} \\ \\ \end{bmatrix} \begin{bmatrix} \\ X^{i\times 1} \\ \\ \end{bmatrix} + \begin{bmatrix} \\ b^{d\times 1} \\ \\ \end{bmatrix} \right)$$

---

Where $b$ is the hidden size of the network and $i$ is the input size. Note that in the cases where a matrix $A$ of size $d \times d$ and a vector $b$ of length $d$ are multiplied together by the Hadamard product, broadcasting is implied. This means that the result of the product is a matrix of size $d \times d$ and is obtained through element-wise multiplication of all the columns in the matrix $A$ individually with the vector $b$.

Table 4.3 Dimensions of neuronal STPN computations

$$\begin{bmatrix} x_t^{d\times 1} \end{bmatrix} = \begin{bmatrix} z_x^{d\times 1} \end{bmatrix} + \left( \mathbf{1} - \begin{bmatrix} z_x^{d\times 1} \end{bmatrix} \right) \begin{bmatrix} x_{t-1}^{d\times 1} \end{bmatrix} - \left( \begin{bmatrix} u_t^{d\times 1} \end{bmatrix} \odot \begin{bmatrix} x_{t-1}^{d\times 1} \end{bmatrix} \right) \odot \begin{bmatrix} h_{t-1}^{d\times 1} \end{bmatrix}$$

---

$$\begin{bmatrix} u_t^{d\times 1} \end{bmatrix} = \begin{bmatrix} U^{d\times 1} \end{bmatrix} \odot \begin{bmatrix} z_u^{d\times 1} \end{bmatrix} + \left( \mathbf{1} - \begin{bmatrix} z_u^{d\times 1} \end{bmatrix} \right) \odot \begin{bmatrix} u_{t-1}^{d\times 1} \end{bmatrix} + \begin{bmatrix} U^{d\times 1} \end{bmatrix} \odot \left( \mathbf{1} - \begin{bmatrix} u_{t-1}^{d\times 1} \end{bmatrix} \right) \odot \begin{bmatrix} h_{t-1}^{d\times 1} \end{bmatrix}$$

---

$$\begin{bmatrix} h_t^{d\times 1} \end{bmatrix} = \left( \mathbf{1} - \begin{bmatrix} z_h^{d\times 1} \end{bmatrix} \right) \odot \begin{bmatrix} h_{t-1}^{d\times 1} \end{bmatrix} + \begin{bmatrix} z_h^{d\times 1} \end{bmatrix} \odot \sigma \left( \begin{bmatrix} W^{d\times d} \end{bmatrix} \left( \begin{bmatrix} u_t^{d\times 1} \end{bmatrix} \odot \begin{bmatrix} x_t^{d\times 1} \end{bmatrix} \odot \begin{bmatrix} h_{t-1}^{d\times 1} \end{bmatrix} \right) + \begin{bmatrix} P^{d\times i} \end{bmatrix} \begin{bmatrix} X^{i\times 1} \end{bmatrix} + \begin{bmatrix} b^{d\times 1} \end{bmatrix} \right)$$

---

Where $b$ is the hidden size of the network and $i$ is the input size.

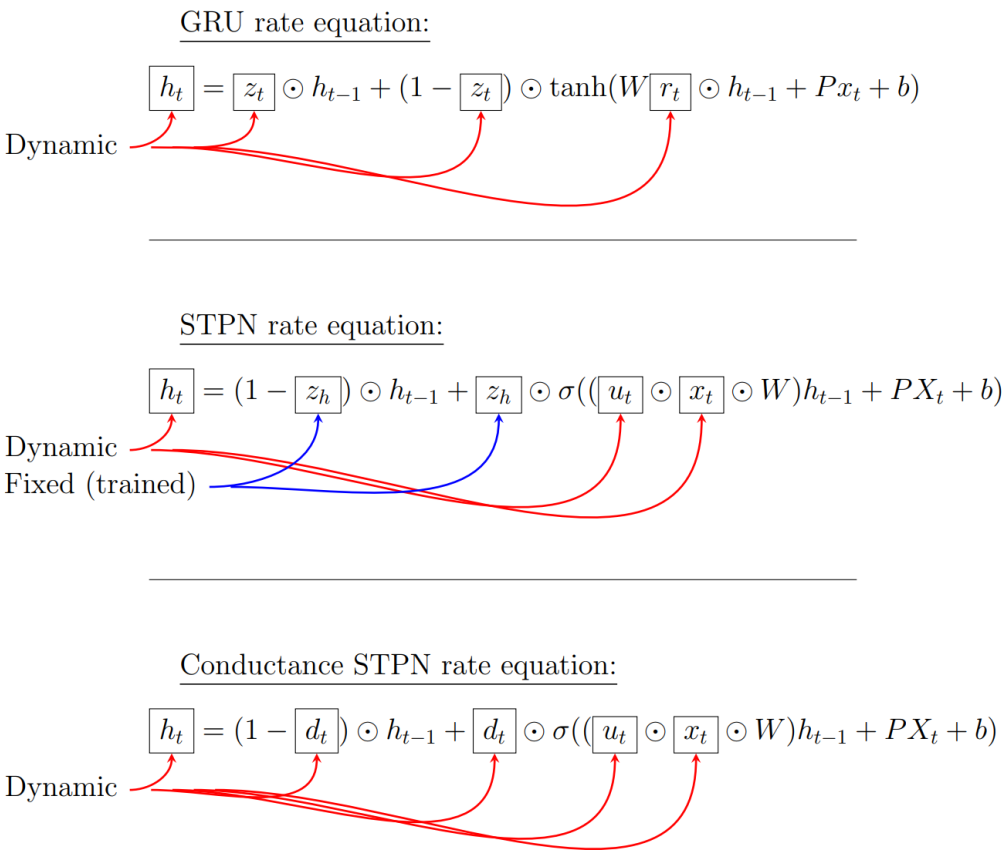



Fig. 4.2 *Rate equations with dynamic and fixed variables labelled.*

however they both only depend on the hidden state and do not take into account the input.

The main difference between STPNs and other gated RNNs is that $z_h$ (and other $z$'s) are all fixed parameters that are trained via gradient descent. The dynamic variables are only used to alter the synaptic weights $W$ and do not directly interact with the rest of the equation outside of the sigmoid function. Whereas in an RNN such as a GRU, the parts of the equation outside of the non-linearity are all dynamic. The rate equations for the GRU and the STPN are written out and labelled in Fig 4.2.

Since this is the main difference between the STPN and the GRU, we theorise that it is the main factor behind the difference in performance between the two networks.

Therefore, we propose a change to the standard STPN such that $z_h$ becomes $d_t$ which is a dynamic variable. We define $d_t$ using the weights already present in the STPN network along with three other new trained variables.

$$d_t = \sigma(\alpha\ |w|\ h_{t-1} + \beta\ |P|\ X_t + b_z),$$

$\alpha$ and $\beta$ are both scalars and $b_z$ is a separate bias vector to $b$ in the rate equation. Making this change to the STPN can be interpreted as $\tau$ of the neurons in the network changing over time. $|.|$ denotes the absolute value.

As charge flows into a neuron, conductance changes as holes in the membrane open which changes $\tau$ ($\tau$ is proportional to conductance). Excitatory and inhibitory channels produce positive or negative changes in current respectively however both make positive changes to conductance. Which is why the absolute value of $W$ and $P$ are used.

Note that this is also where gated RNNs such as the GRU are not biological since in the Update gate $z_t$, the weights $W_z$ and $P_z$ can have negative elements.

The version of the GRU which is modified to alter this issue and used for experiments is called the Conductance GRU to reflect the increase in biological accuracy.

## 4.3 RNN hierarchy

In the completed experiments, several versions of RNNs were used. Starting from the LSTM, key features were removed incrementally until the vanilla RNN was reached.

Decreasing Complexity

| | |
|---|---|
| LSTM | Standard LSTM equations with three gates (Input gate $i_t$, Forget gate $f_t$ and Output gate $o_t$ and two states (hidden state $h_t$ and cell state $c_t$). |
| GRU | Standard GRU equations. Cell state has been removed, two gatets (Update gate $z_t$ and Reset gate $r_t$) instead of three. |
| Simple GRU | Reset gate removed (set to 1 in GRU equations). The Update gate is still present so $z_t$ is both time and neuron dependent. |
| Conduc -tance GRU | Introduces a more biologically accurate Update gate $d_t$ which replaces $z_t$. This also also removes the weights $W_z$ and $P_z$ that were present in the Update gate equation. $d_t = \sigma(\alpha \lvert W \rvert h_{t-1} + \beta \lvert P \rvert X_t + b_z)$ |
| Vanilla RNN | Standard equation with no gates. $h_t = \tanh(Wh_{t-1} + Px_t + b)$ |

Table 4.4 *Incremental changes in network architecture from LSTMs to vanilla RNNs.*

## 4.4 Sequential MNIST

The Two-armed bandit task tests for the flexible nature of RNNs. As a method of assessing a more general form of network performance instead of flexibility, the MNIST dataset was used.

The sequential MNIST task essentially treats each image in the dataset as a long sequence of numbers which are fed into the network in smaller sections over multiple timesteps. This tests the long short-term memory capabilities of an RNN because in order to classify an image, the network must remember the inputs it received from the beginning of the image.

In our experiments, the network reads the pixels with various input sizes in scanline order (i.e. starting at the top left and moving to the right, shift to the next row once the end of a row is reached).

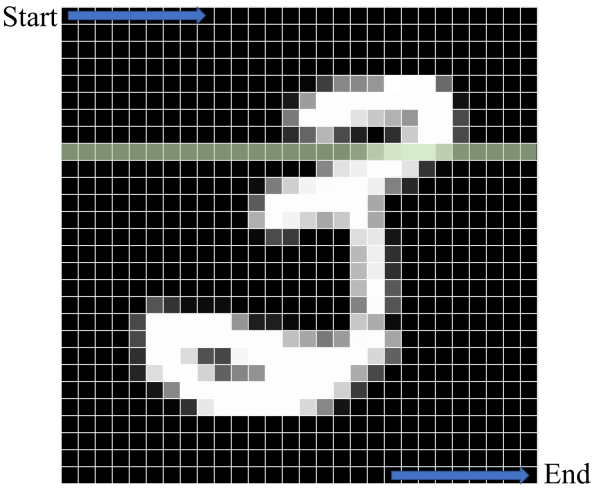



Fig. 4.3 *Illustration of sequential MNIST with scanline order.*

Each image in the MNIST database is consisted of $28 \times 28 = 784$ pixels. To break down the entire sequence of 784 numbers, a smaller input size can be set so that total time-steps $= \frac{784}{\text{input size}}$. The green bar in Fig.4.3 shows an in-

put size of 28. Because of this, the difficulty of the sequential MNIST task can be altered by changing the input size. A smaller input size means that more time-steps will be required for the network to read the entire image, this will require the network to maintain memory of the earlier inputs for longer. In its most difficult form, each pixel is fed into a network separately which means the network has to keep memory for 784 time-steps.

If the input size is not a factor of 784, zero-padding can be used on either end of the sequence.

In many of the experiments using sequential MNIST, we introduced 2 more variables, time gap and stride. These can be used to further alter the task to change its level of complexity.

Time gap refers to the gap between each pixel in an input, for example if each row in the image was fed into the network at each time-step, this would be an input size of 28 and a time gap of 1. This is because each pixel in the input follows the pixel it is next to in the image. For a time gap of 4, the first input into the network should contain pixels 1,4,7,10...

Stride is defined as the gap between the first pixels of every input. In the example described above where an entire row is fed at each time-step, the stride would be 28 because the first input into the network was pixels $1,2,3,4...$ and the second input was pixels $29,30,31,32....$ $(29-1 = \text{a stride of } 28)$.

Training of the networks used a batch size of 100 and Cross entropy loss. Test accuracy of the models were obtained using 10000 test images.

### 4.4.1 Permuted sequential MNIST

The permuted sequential MNIST task was also experimented on. The performance of more simple RNNs on normal sequential MNIST tended to be a lot lower than gated RNNs such as the GRU, especially for particularly low input sizes. This was due to most images in the dataset having multiple rows of black pixels at the bottom, which meant that networks were essentially receiving 0 input for quite a few time-steps.

The spiral sequential MNIST task that was used for majority of the experiments simplified the task by permuting the images in the dataset in a spiral order such that most of the bright pixels are at the bottom of the image.

Fig. 4.4 *Spiral permutation.*

Fig. 4.5 *Example of a 2 in spiral form.*

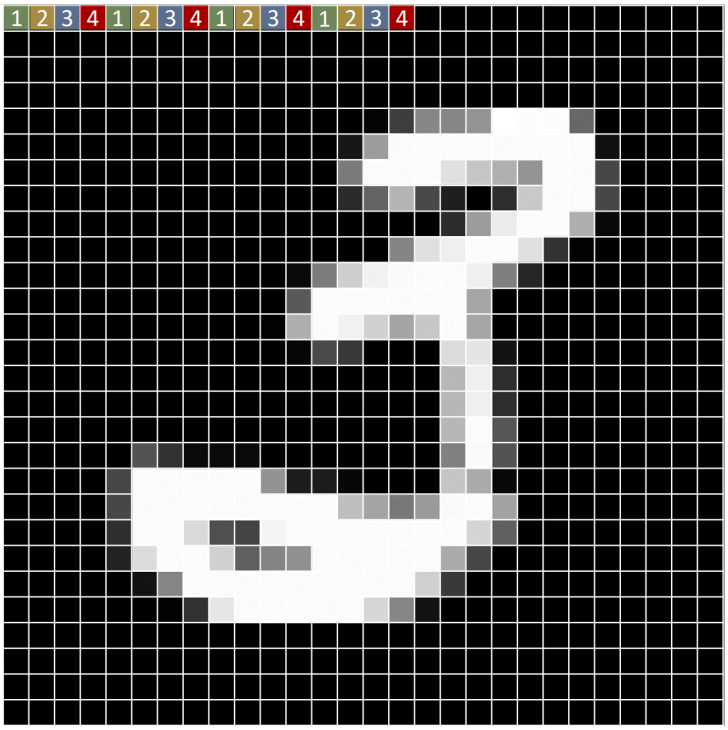



Fig. 4.6 *1 represents the first input, 2 represents the second input and so on. This shows an input size of 4 and a time gap of 4. Input 5 will include pixels that have already been inputted into the network.*

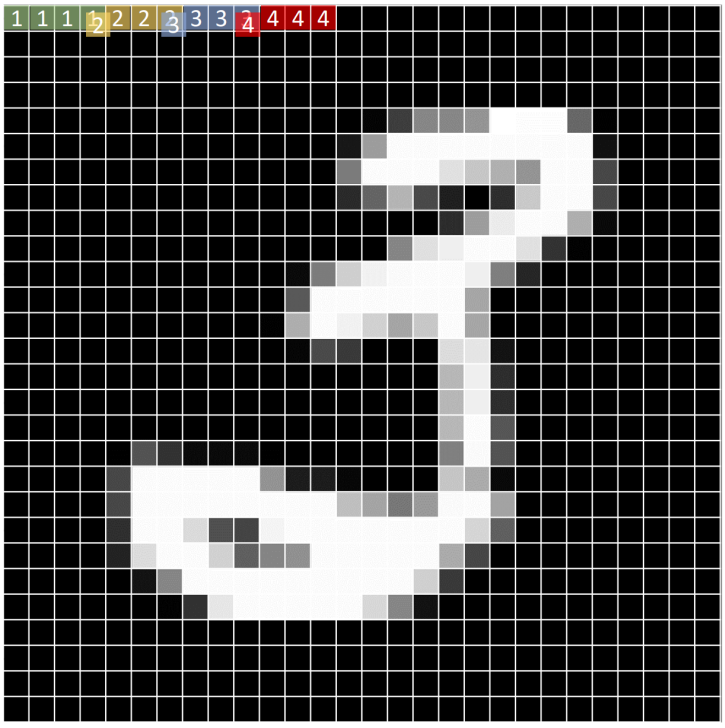



Fig. 4.7 *Input size 4, time gap 1 and a stride of 3. This means that the first pixel of every input will be the same as the last pixel of the previous input.*

# References


[1] Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

[2] Costa, R. P., Assael, Y. M., Shillingford, B., de Freitas, N., and Vogels, T. P. (2017). Cortical microcircuits as gated-recurrent neural networks. https://doi.org/10.48550/arxiv.1711.02448.

[3] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[4] Jordan, I., Sokół, P., and Park, I. (2021). Gated recurrent units viewed through the lens of continuous time dynamical systems. *Front. Comput. Neurosci. 15:678158.* doi:10.3389/fncom.2021.678158.

[5] Lau, B. and Glimcher, P. (2005). Dynamic response-by-response models of matching behavior in rhesus monkeys. *J. Exp. Anal. Behav. 84, 555–579.*

[6] Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941.

[7] Mnih, V. et al. (2016). Asynchronous methods for deep reinforcement learning. *arXiv*. https://doi.org/10.48550/arxiv.1602.01783.

[8] Mongillo, G., Barak, O., and Tsodyks, M. (2008). Synaptic theory of working memory. *Science*, 319(5869):1543–1546.

[9] Sussillo, D. and Barak, O. (2013). Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks. *Neural Computation*, 25(3):626–649.

[10] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

[11] Tsodyks, M. and Wu, S. (2013). Short-term synaptic plasticity. *Scholarpedia*.

[12] Wang, J. et al. (2016). Learning to reinforcement learn. https://arxiv.org/abs/1611.05763.

[13] Wang, J., Kurth-Nelson, Z., Kumaran, D., et al. (2018). Prefrontal cortex as a meta-reinforcement learning system. *Nat Neurosci 21, 860–868.* https://doi.org/10.1038/s41593-018-0147-8.