# Mitigating Catastrophic Forgetting in LSTM-RNNs and Biologically Constrained RNNs

by

Amartya Prasad

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Master of Science(M.S.) in Mathematics
Courant Institute of Mathematical Sciences
New York University
August 2020

Professor Zhe S. Chen
Department of Neuroscience and Physiology
Grossman School of Medicine
NYU Langone

To my mother and father

# Acknowledgements

I would like to acknowledge the support of my friends and family as I worked on this thesis.

# Abstract

Catastrophic forgetting is a wide-recognised problem in machine learning refers to performance degradation on previously learned tasks. Taking inspiration from the mechanism for retention and learning in the brain, I employ Deep Generative Replay, an architectural (verify this) technique to mitigate catastrophic forgetting in neural networks, wherein the neural network, while being trained to learn new tasks is also played samples of past data, generated by a generative model trained on said past data. I test the validity and applicability of this technique for Biologically Inspired RNNs, specifically, a Long Short Term Memory (LSTM) Recurrent Neural Network(RNN). We see that Deep Generative Replay can be nearly as effective for Contnual Learning in LSTM RNNs as well, as in as in MultiLayer Perceptrons(MLP) and Convolutional Neural Networks(CNNs). It does not however, perform as well for Biological RNNs beyond two tasks.These results, besides providing possible insights into memory and retention mechanisms of the brain, also encourages future research in the area.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Catastrophic Forgetting is a widespread and well recognised problem plaguing Machine Learning models. When a machine learning model that is trained on some data to be able to learn some task, is re-trained to learn another task, it's performance on the first task degrades considerably. This phenomenon, in which a model's performance on previously learned tasks degrades dramatically when trained for a new task is what is referred to as Catastrophic Forgetting.

There have been several attempts to mitigate 'forgetting' with some techniques working better than others. These techniques can broadly be classified into regularisation approaches and architectural approaches ([10], [11]). Regularisation approaches usually involve adding regularisation terms to the cost function to to be optimised. The regularisation term serves to penalise changes to parameters learnt for previous task(s). Important among these is EWC (Elastic Weight Consolidation) [4]. In this technique, once the network is trained for a task, the already trained network is then retrained, to learn the second task with EWC regularisation. EWC involves modifying the cost function for the training of the second task

by adding a regularisation term. This regularisation term penalises the network for moving away from the learned parameters with greater penalties for weights that were more important for learning the first task acting as a buffer that springs the new solution close to the previous learned solution. This outlines the theoretical argument for combining dropout with EWC. The network is trained to learn the second task by minimising the cost function

$$L(\theta) = L_2(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta_{1,i})^2$$

where, $L_2(\theta)$ is the squared loss for task 2, $\theta_1$ is the set of parameters(weights) obtained after learning the first task, and $F_i$ is the ith diagonal element of the Fisher Information Matrix. $\lambda$ is a hyper parameter that we use to tell the network how important the first task is, relative to the first task, that is, it determines the magnitude of the penalty. The Fisher information matrix, essentially tells us how important each parameter(weight) is to learning of the first task, relative to other parameters. This can easily be extended to N tasks However EWC fails to perform well on incremental class learning, the focus of this paper [11]. Similarly, most other regularisation approaches aim to curb the magnitude of weight changes batch by batch, by modifying the cost function.

Likewise, a well known architectural approach is the one proposed in [6] wherein the network architecture is expanded with each subsequent task such that a sub network of the new expanded network is trained to learn the new task, while the older part of the network, which proposes to block any changes to the network trained on previous knowledge and expand the architecture by allocating novel sub-networks with fixed capacity to be trained with the new information. With this approach, given that parameters learnt after training for a past task are kept fixed, catastrophic forgetting naturally does not occur [6]. However, this approach

has significant drawbacks. The complexity of the network architecture increases as the number of the tasks increases, thereby needing more computational power and memory. Experiments reported good results on a wide variety of reinforcement learning tasks, outperforming common baseline approaches that either pre-train or incrementally fine-tune the models by incorporating prior knowledge only at initialisation.

Regularisation and Architectural strategies can be combined too. In the AR1 (Auto Regressive TIme Series Model), proposed in [12] for single-incremental-task scenarios, instead of simply progressively capping the magnitude of weight changes batch by batch, with most of the changes occurring in the top layers, in AR1 the intermediate layers weights are adapted without negative impact in terms of forgetting. Reported results on CORe50 [13] and iCIFAR-100 [14] show that AR1 allows the training of deep convolutional models with less forgetting, outperforming LwF and EWC.

Table 1 outlines a number of techniques that attempt to achieve continual learning as mentioned in [10] and [11].

| Technique | Description | References |
|---|---|---|
| EWC(Elastic Weight Consolidation) | Regularisation Approach - Modifies the Cost Function to force new parameters to be found in a neighbourhood of the old parameters | [1] |
| Retrain from Scratch: | The network is retrained on all the data. | [2], [3] |
| Learning without Forgetting (LwF): | Regularisation Approach - The network with predictions of the previously learned tasks is enforced to be similar to the network with the current task through the transferring of knowledge from a large, regularised model into a smaller model | [4] |
| Synaptic importance estimation for solving a learned task | Regularization Approach - Allows individual synapses to estimate their importance for solving a learned task in an online fashion | [5] |
| Progressive networks | Architectural Approach - this blocks any changes to the network trained on previous knowledge and expands the network architecture by allocating novel sub networks with fixed capacity to be trained with the new information | [6] |
| Pre-trained CNN with Self-Organizing Incremental Neural Network (SOINN) | This capitalises on the representational power of CNNs while allowing the classification network to grow according to the task requirements. | [7], [8] |
| Short and Long-Term Plasticity | CLS(Comprehensive Learning System)-Inspired Approach - This consolidates new information on the basis of a cause-effect hypothesis testing when learning with delayed reward | [9] |

Table 1.2: Some Techniques for Mitigating Catastrophic Forgetting

Besides the above, there is another approach, a variant of which I employ in this paper, to mitigate catastrophic forgetting, one where is trained to learn the

current task and all previous task(s) again. That is, the network is retrained on all the data. It could also mean instead, that samples from old task(s) are interleaved with samples drawn from the new data. Naturally when the network is retrained to learn the previous tasks along with the new tasks, catastrophic forgetting will be very limited. This has been shown in [15], [16], [6] and [17]. In fact, in Humans too, major catastrophic forgetting of previously learnt information is not commonly observed. They do not typically exhibit strong events of catastrophic forgetting because the kind of experiences we are exposed to are very often interleaved [18]. This however, requires the explicit storage of all old data as well which results in high working memory requirements, besides increased computational costs. This can become very untenable when the number of tasks increases. This is where the idea of Deep Generative Replay comes in [19]. It is shown in [19] that Deep Generative Replay outperforms other methods like Learning without forgetting etc.

In Deep Generative Replay, the tasks are learnt by successive scholars as defined in the paper, with as many scholars in total as the total number of tasks to be learnt. A scholar is composed of a Solver and a Generator, both of which are separate neural networks. The solver is the neural network which actually learns the task, while the Generator is a neural network that represents a generative model. In this paper, a GAN(Generative Adversarial Network) is used as the Generator. The Generator is trained to learn to reconstruct all past data and samples from the trained generator which could simulate old data from the previous tasks. The labels for this generated data are obtained from the Solver of the most recent scholar. These pseudo-samples are interleaved with the new data when the solver is trained to learn the next task. Thus, this prevents catastrophic forgetting and

enables the network to learn new tasks without forgetting previously learnt tasks, without a need for explicit storage of data.

Thus, this approach solves the drawback of having to explicitly store old data which eliminates the need to store large amounts of data, thereby resolving the issue of requirements of large working memory and computational power, which makes it highly scaleable. This is particularly since the same network can be trained to learn more and more tasks, without a need to expand the architecture or change the network configuration. At the same time, since the generator, together with the solver can produce as much labelled pseudo data as needed, it provides one the flexibility of changing or expanding the network architecture or configuration at any point should we choose to. This is due to the fact that the input(generator) - output(solver) pairs from any previous scholar can be used to teach a new generator and a new solver, irrespective of the old and new network architectures. In addition, this setup does not require us to know in advance the number of tasks to learn and hence new resources, be it memory, computational power or neural resources, could be allocated at any time.

As explained in their paper, Deep Generative Replay is inspired by the Comprehensive Learning System(CLS)([20], [21], [22]) Theory for learning and memory retention. Humans and other primates are able to acquire new knowledge while retaining previously learnt information even from limited experiences. This has been attributed to the Comprehensive Learning System, that is a dual memory system consisting of the hippocampus and the neocortex [23]. The hippocampus is responsible for learning and encoding new information, that is recent experiences, in

the context of humans and primates, which is replayed later during a sleep period. The neocortex is where the long term memory is consolidated when the recent encoded experiences are replayed to the neocortex during. It has also been seen that in some cases, there could be certain false memories created in the hippocampus, quite like a generative model, which is akin to a generative model that generates samples that may not be identical to true data but by and large comes from the same distribution as the true data, assuming the generative model is indeed able to estimate the distribution accurately [24] . While an extreme simplification, the generator in a Scholar could be seen as analogous to the Hippocampus while the neocortex where long term memory is consolidated can be considered analogous to the neocortex.

Generative replay has several advantages over other approaches because the network is jointly optimized using an ensemble of generated past data and real current data. Therefore, in theory, performance should be equivalent to joint training on the cumulative real data as long as the generator recovers the input distribution.

While Deep Generative Replay naturally has been employed before to mitigate Catastrophic Forgetting, this paper is novel in multiple ways, Firstly, Deep Generative Replay has not been employed to prevent catastrophic forgetting in RNNs(Recurrent Neural Networks) before. I test whether Deep Generative Replay mitigates Catastrophic Forgetting in RNNs, in this paper.

More importantly, I employ Deep Generative Replay to mitigate catastrophic forgetting in Biologically Inspired Recurrent Neural Networks. Since the very idea of Deep Generative Replay itself was inspired by the Comprehensive Learn-

ing System(CLS) in humans and other animals, it makes sense to test whether such a mechanism can really be valuable for mitigating Catastrophic forgetting in an artificial neural network(ANN) that attempts to closely simulate Human Neural Networks . The Biologically Constrained RNN used in this paper is the one adapted from [25]. Human neurons have firing rates which could be understood to correspond to network weights or synaptic weights between neurons in an Artificial Neural Network. Neurons in the mammalian cortex are said to satisfy Dale's principle [26]. That is each neuron has either purely excitatory or purely inhibitory effects on other neurons [27]. More fundamentally, existing networks do not satisfy Dale's principle, the basic and ubiquitous observation that neurons in the mammalian cortex have purely excitatory or inhibitory effects on other neurons. Equivalently, this means that all connection weights from a particular neuron to other neurons must be either positive or negative but not both. This constraint has been shown to have a significant on network dynamics such as non-normality [28].

Moreover, connections from excitatory and inhibitory neurons exhibit different levels of sparseness and specificity, with non-random features in the distribution of connection patterns among neurons both within local circuits ([29], [30]) and among cortical areas ([27], [31],[32]). Notably, long-range projections between areas are primarily excitatory. Such details must be included in a satisfactory model of local and large-scale cortical computation.

We address this challenge by describing flexible, gradient descent-based training of excitatory-inhibitory RNNs that can incorporate a variety of biological knowledge, particularly of local and large-scale connectivity in the brain.

Another way this is novel is because the above biological RNN has only been

using inputs that can only simulate sensory information. However, I train the network to learn MNIST digits.

Chapter 2 deals with the Experiments and Training Methodology.-It includes sections on each of the Vanilla LSTM - RNN, the Biologically Constrained RNN, the GAN as well as a section dedicated to Deep Generative Replay. Chapter 3 outlines the results of the experiments and the final Chapter, Chapter 4 is the Conclusion.

## 1.1 Motivation

The Motivations to investigating techniques to mitigate catastrophic forgetting in Biological Neural Networks are manifold. Unlike machines, humans or animals are able to assimilate and learn new information continuously over the course of their lifetimes without forgetting previously acquired knowledge. This has been attributed to the Comprehensive Learning System (CLS), where-in the Hippocampus is responsible for encoding new information(short term memory) which is then replayed during sleep together with previously learned information from the Cortex and the memory is consolidated in the Cortex(long term) through the activation synchronized with multiple replays of the encoded experience. Using Deep Generative Replay, a technique inspired by knowledge acquisition and memory retention and consolidation mechanisms in the brain, to mitigate catastrophic forgetting in a Recurrent Neural Network(Biologically Constrained) which satisfies Dale's principle, that is, which consists of populations of purely excitatory and inhibitory neurons, besides fixed ratios of plastic and fixed neurons, could provide significant insights into internal mechanisms and processes in the Cortex and the

HIppocampus, Thus, from a neurobiological perspective, It could provide insights into how neural synaptic connections are optimized in the brain that allow for efficient retention and learning of new information.

From a Machine Learning Perspective, it has to do with computational costs and access to data. As described above, it has been shown that replaying actual past data while training the network and jointly optimise parameters can achieve performance as high training separate networks independently [24], such an exercise would require massive amounts of computational power and working memory. However, besides the challenge of meeting such high resource requirements, in any real lifelong learning scenario, since the number of tasks in future cannot be known, it would be a non trivial task to apportion an appropriate amount of resources be it memory or computational power. Furthermore, access to past data may not always be available. The Deep Generative Replay approach addresses all of these concerns.

# Chapter 2

# Data and Methods

The First Section of this Chapter describes the Dataset. The Next two sections of this Chapter describe the Solvers, that is, the Vanilla LSTM RNN and the Biologically Constrained RNN which learn the tasks(the MNIST digits) and explain their network structures and internal connections respectively. The Fourth Section describes the GAN, that is, the neural network which is trained to generate samples that resemble past data. The Final Section explains Deep Generative Replay as well as details the actual training and learning methodology.

## 2.1 The Dataset

The MNIST Dataset is a database of 70000 handwritten digits where each digit is a $28 \times 28$ pixel image. Each of the 784 values is an integer between 0 and 255 denoting the colour for that pixel. The dataset is split into training and test sets of 60000 and 10000 images respectively It is a dataset widely used for training and testing in the field of Machine Learning. Specifically it is among the most commonly used datasets for mage Classification. There are 10 classes, representing the

digits 0-9. There are two factors about this dataset that make it highly suitable for evaluating Catastrophic Forgetting Techniques. First, the classes are roughly uniformly distributed in the dataset, with an adequate number of training and test samples. Second, the dataset has 10 classes, which makes it ideal for incremental class learning. The large number of classes also allows the assigning of more than one distinct class to each task.

Further, it is a well established benchmark used to develop and evaluate techniques and frameworks that mitigate Catastrophic Forgetting. Using the same dataset allows for a better comparison of performance using such techniques.

Finally, a neural network learning handwritten digits could be seen as being analogous to humans learning numbers. This analogy with a human example of learning and continual learning aligns well with the technique being evaluated in this paper, that is, one involving using a framework inspired by the brain, to mitigate catastrophic forgetting in a biologically constrained neural network that seeks to simulate neural networks in the Brain.

Figure 2.1 and Figure 2.2 show examples of MNIST samples and Figure 2.3 shows the explicit pixels of the image with grid lines:

Figure 2.1: A Collection of MNIST Digits

## 2.2 The Vanilla LSTM-RNN

RNNs or Recurrent Neural networks a family of Artificial Neural Networks Recurrent neural networks (RNN) are a class of artificial neural networks where connections between neurons form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Although derived from feedforward neural networks, they are distinct from feedforward networks in that, the output of the network is not fed back into the network in feedforward networks. RNNs can use their internal state (memory) to process variable length sequences of inputs. This allows RNNs to learn/capture time dependencies in data which makes them especially suitable for tasks that involve a time component, such as such as unsegmented, connected handwriting recognition or speech recognition or

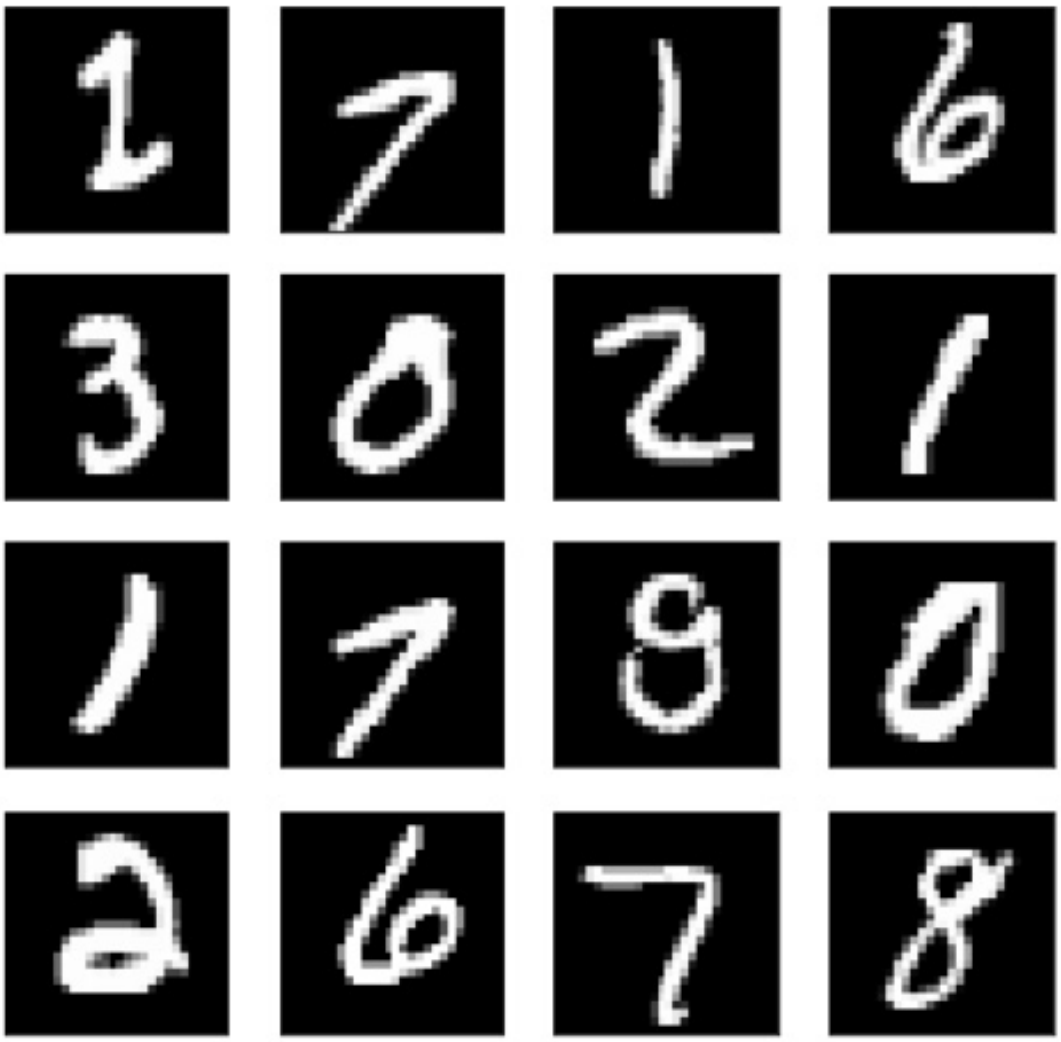

Figure 2.2: Some MNIST Digits

other time series data.

RNNs are best suited to deal with sequences of data. The sequence length defines the number of time points. Given sequence length $T$, at each time point $t$, the RNN receives an input $u(t)$ which is generally an $N_{in}$ - dimensionnal vector, where $N_{in}$ represents the dimension of the input at each time point. The network, produces an outputs $z(t)$ at each time point $t$ : $z(t)$, an $N_{out}$ - dimensional vector,

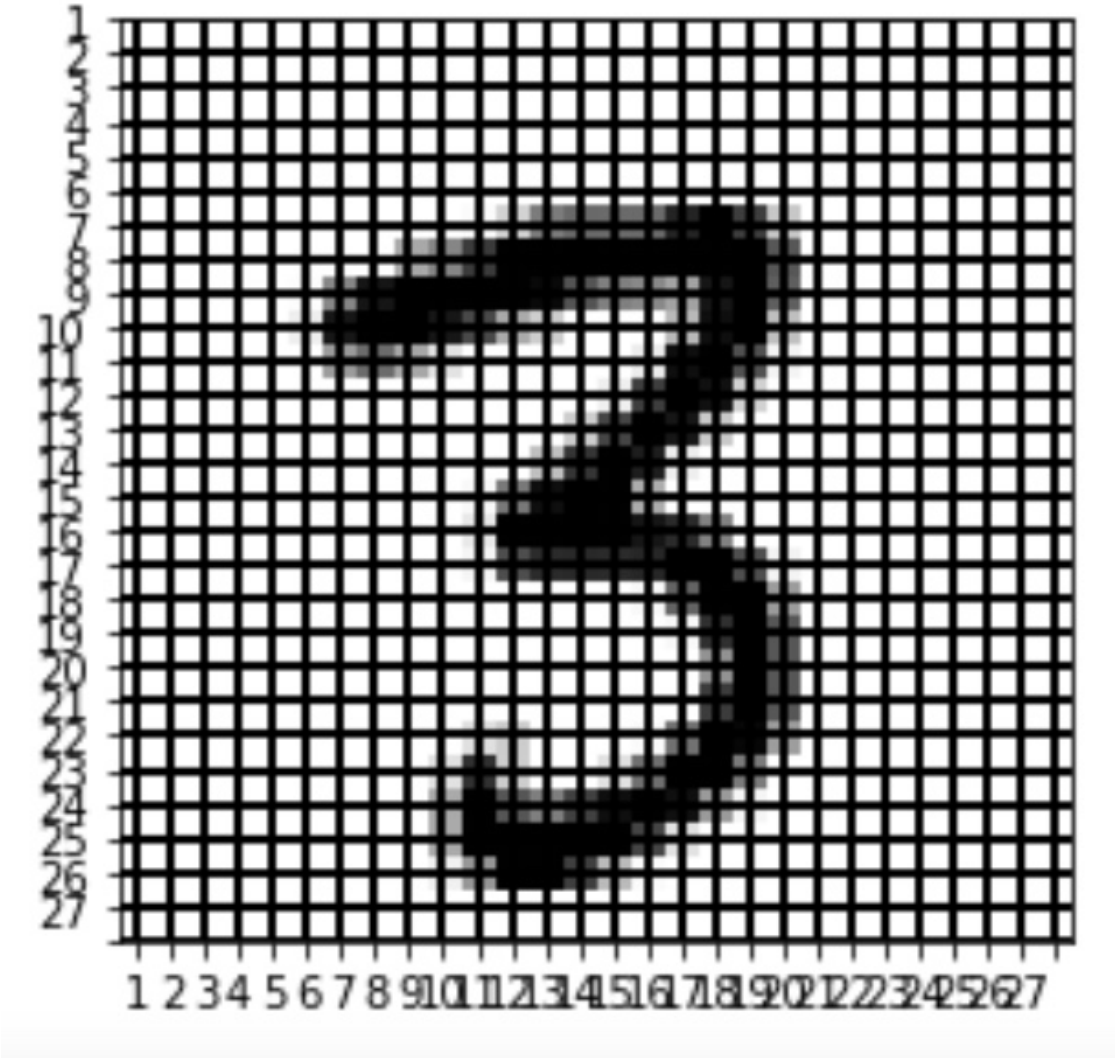

Figure 2.3: The 28 x 28 Pixels of the Image

where $N_{out}$ represents the size of the output at each time point. The current output depends not only on the current input but also on the state of the system. The inputs encode task-relevant information and outputs typically represent the categorical classification or probability distribution. As shown in Figure 2.4 below, the RNN :

- first receives input $x(t)$ hat varies across time

- which is passed to an Encoder say Enc($x(t)$) which generates a representation of the input, $h(t)$
- $g$ is function that can be a complicated neural network; one of the inputs of which is $h(t)$.
- $w$ represents trainable parameters of $g$, or the recurrent network weights, also referred to as synaptic weights
- $z(t)$: represents the last hidden state of the system, which is the output of the previous time step and $z(t)$ the current hidden state
- Dec($z(t)$): decoder that generates the final output $y(t)$ for time $t$.

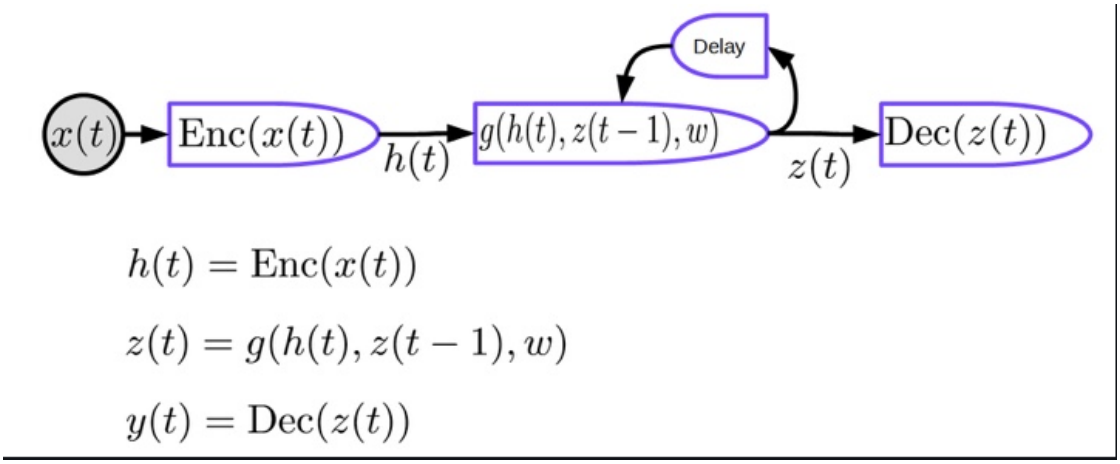


Figure 2.4: RNN Architecture

However, RNNs suffer from two major problems :

- Vanishing gradients: For backpropogation through time, the gradients get multiplied by the weight matrix (transpose) at every time step. If there are

small values in the weight matrix, the norm of gradients get smaller and smaller exponentially which could lead the gradient to become arbitrarilly small.

Consider initial state $\boldsymbol{h}_0$ and input sequence $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$. The RNN state $\boldsymbol{h}_t$ is a function of the previous hidden state and current input, usually

$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}_1 \boldsymbol{h}_{t-1} + \boldsymbol{W}_2 \boldsymbol{x}_t)$$

Usually, loss $\mathcal{L}_t$ at time $t$ is a function of the state $\boldsymbol{h}_t$. We minimise total loss $\mathcal{L} = \sum_t \mathcal{L}_t$. The gradient $\partial\mathcal{L}/\partial\boldsymbol{W} = \sum_t \partial\mathcal{L}_t/\partial\boldsymbol{W}$ and $\partial\mathcal{L}_\tau/\partial\boldsymbol{W}$ depends on $\partial\boldsymbol{h}_\tau/\partial\boldsymbol{h}_t$ for all $\tau > t$, where $\partial\boldsymbol{h}_\tau/\partial\boldsymbol{h}_t$ is what vanishes or explodes due to repeated application of matrix $\boldsymbol{W}_1$. Therefore, RNNs can't learn long dependencies when gradients vanish or explode from $\mathcal{L}_\tau$ back to $\boldsymbol{h}_t$ (and therefore to $\boldsymbol{W}_1$) for $\tau \gg t$, due to vanishing or exploding in $\partial\boldsymbol{h}_\tau/\partial\boldsymbol{h}_t$.

- Exploding gradients: Similarly, If we have a large weight matrix $\boldsymbol{W}$ and the non-linearity in the recurrent layer is not saturating, the gradients will explode. The weights will diverge at the update step. We may have to use a tiny learning rate for the gradient descent to work.

One of the principal reasons RNNs are used are for their ability to retain past information. However, a Vanilla RNN could fail at remembering information from several time steps past, without tricks.

This makes learning very difficult with Vanilla RNNs and as a result, Vanilla RNNs aren't widely used in practise. To get over this problem, we frequently

use LSTMs instead. LSTMs are a kind of specialised RNNs where their modified architectural setup allows the network to better 'store' older information.
The LSTM maintains an auxiliary cell state $\boldsymbol{c}$ that helps copy the hidden state $\boldsymbol{h}$ forward in time as $\boldsymbol{x}$ is processed. Letting $\boldsymbol{f}_t, \boldsymbol{i}_t, \tilde{\boldsymbol{c}}_t$ and $\boldsymbol{o}_t$ be nonlinear functions of $\boldsymbol{h_{t-1}}$ and $\boldsymbol{x}_t$ and letting $\odot$ denote element-wise product, the LSTM hidden state computation for $\boldsymbol{c}$ and $\boldsymbol{h}$ is defined by

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t; \tag{2.1}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t). \tag{2.2}$$

In the update of the cell, the '*don't forget*' gate $\boldsymbol{f}_t$ determines the influence of previous cell states on the next. The input gate $\boldsymbol{i}_t$ determines the influence of new information on the cell. $\boldsymbol{f}_t$, $\boldsymbol{i}_t$, $\boldsymbol{o}_t$ and $\tilde{\boldsymbol{c}}_t$ are defined via usual RNN recurrences:

$$\begin{aligned}
\tilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_{gh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{gx}\boldsymbol{x}_t + \boldsymbol{b}_g);\\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{fh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t + \boldsymbol{b}_f);\\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ih}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t + \boldsymbol{b}_i);\\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_{oh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ox}\boldsymbol{x}_t + \boldsymbol{b}_o).
\end{aligned}$$

Consider the gradient $\partial\mathcal{L}_t/\partial\boldsymbol{W}$ for all eight matrices $\boldsymbol{W}$. Because there is a path $\boldsymbol{W} \to \boldsymbol{c} \to \boldsymbol{h} \to \mathcal{L}_t$, the cell update (eq. 2.1) affects the gradients $\partial\mathcal{L}_t/\partial\boldsymbol{W}$. The gradient for $\boldsymbol{W}$ is accumulated through both terms in the sum in (eq. 2.1)

The $\boldsymbol{f}_t \odot \boldsymbol{c}_{t-1}$ term in (eq. 2.1) is credited with solving the vanishing gradient

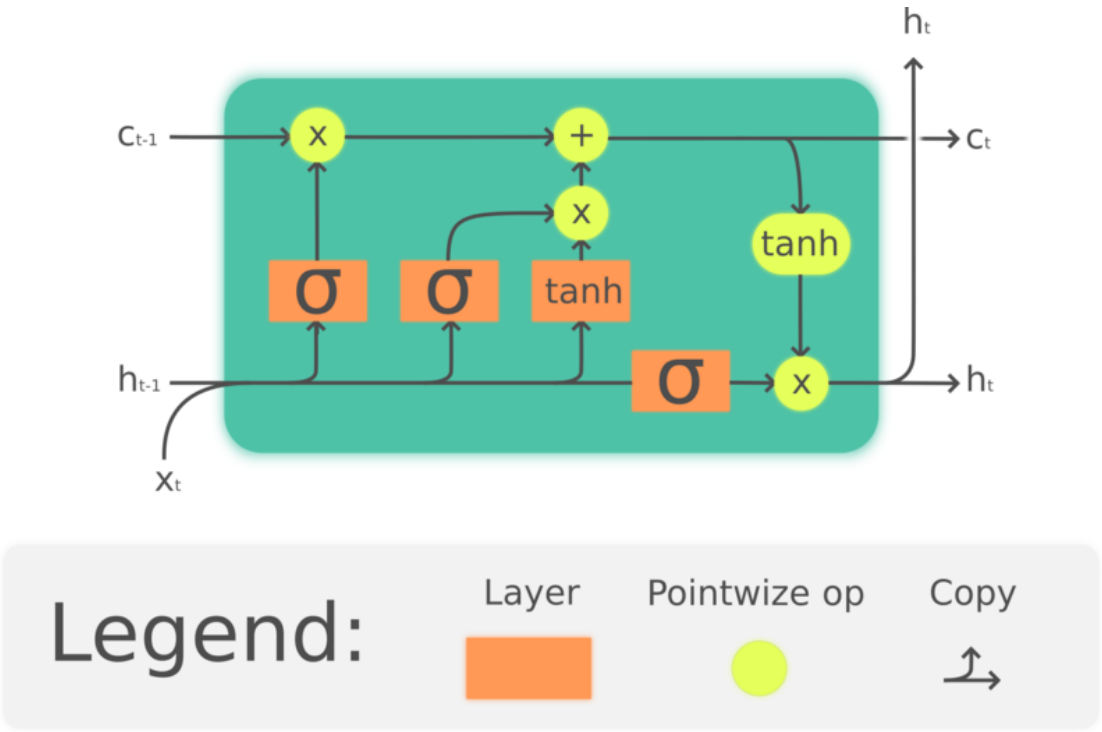


Figure 2.5: LSTM Architecture

problem. An LSTM unit uses a cell state $\boldsymbol{c}_t$ to convey the information through the unit. It regulates how information is preserved or removed from the cell state through structures called gates. The forget gate $\boldsymbol{f}_t$ decides how much information we want to keep from the previous cell state $\boldsymbol{c}_{t-1}$ by looking at the current input and previous hidden state, and produces a number between 0 and 1 as the coefficient of $\boldsymbol{c}_{t-1}$.

Since each $\boldsymbol{h}_{t-1}$ is a function of $\boldsymbol{c}_{t-1}$, the gradient that could lead to a vanishing gradient is the intermediate gradient(when computing $\partial \mathcal{L}_t / \partial \boldsymbol{W}$) $\partial\ \boldsymbol{c}_t / \partial \boldsymbol{c}_{t-1}$ since it gets multiplied over and over until the first time step. For LSTMs, this intermediate gradient, $\partial \boldsymbol{c}_t / \partial \boldsymbol{c}_{t-1}$ becomes :

$$\partial \boldsymbol{c}_t / \partial \boldsymbol{c}_{t-1} = \boldsymbol{f}_t + K_t$$

where $K_t$ is the sum of the remaining gradient terms :

$$K_t = \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{f}_t}\frac{\partial \boldsymbol{f}_t}{\partial \boldsymbol{h}_{t-1}}\frac{\partial \boldsymbol{h}_{t-1}}{\partial \boldsymbol{c}_{t-1}} + \frac{\partial \boldsymbol{c}_t}{\partial \boldsymbol{i}_t}\frac{\partial \boldsymbol{i}_t}{\partial \boldsymbol{h}_{t-1}}\frac{\partial \boldsymbol{h}_{t-1}}{\partial \boldsymbol{c}_{t-1}} + \frac{\partial \boldsymbol{c}_t}{\partial \bar{\boldsymbol{c}}_t}\frac{\partial \bar{\boldsymbol{c}}_t}{\partial \boldsymbol{h}_{t-1}}\frac{\partial \boldsymbol{h}_{t-1}}{\partial \boldsymbol{c}_{t-1}}$$

Since this gradient contains $\boldsymbol{f}_t$, that is, the forget gate activations, if the gradient starts to vanish, the network can learn to set $\boldsymbol{f}_t$'s (as well as the other gates) high enough(higher than 1) to prevent the gradient from vanishing. The network can do this as $\boldsymbol{f}_t$ is a learnable parameter and the network can choose different values for $\boldsymbol{f}_t$ for different $t$(different time steps) and thus since the gradient is additive, it lends to more balanced gradient values. Therefore, the network can choose what it wants to forget or not forget by choosing $\boldsymbol{f}$ .

The MNIST Dataset contains 60000 images which are handwritten digits from 0-9. Each image is $28 \times 28$ pixels. To train, each row of the image is considered to be a time step, such that the sequence length is 28 and the size of the input at each time point is also 28. That is, $N_{in}$ is 28. $N_{out}$ is 10, ie the targets are 10-dimensional vectors where the vectors are one-hot encoded representation of the classified digit. That is to say, the vector has zeros everywhere except the row which corresponds to the true digit where there is a 1. In a practical Continual Learning Scenario, the number of tasks, or the number of classes cannot be known apriori. Therefore, $N_{out}$ has been chosen as 10 to avoid overlapping representation. It is unrealistic to be able to know apriori the number of distinct classes and choose a one hot representation of an appropriate size. But $N_{out}$ is set as 10 for convenience. The optimization algorithm used for training was ADAM [33] with learning rate 0.01, $\beta_1$ 0.9 and $\beta_2$ 0.999.

The recurrent layer used has a hidden dimension of 64. The first layer of the network is a fully connected linear layer that increases the dimension of the input from 28 to 64. Following this the input is passed to the recurrent layer. The output of the recurrent layer at the last time step is then passed through a final fully connected linear layer which takes the dimensionality to 10 from 64, to be able to get a one hot encoded prediction. It is standard practise when using RNNs for Image Classification to take into account the output from only the last time step of the recurrent layer for computing loss. This is since, if losses were computed and summed for each time step, as is the case for RNNs in general, we would essentially be expecting the network to be able to recognise the digit upon seeing just a single pixel which would be both extremely implausible and intractable. This output from the last time step is then passed through a ReLU(Rectified Linear) Activation function. The Final Layer is another fully connected layer such that final output has dimension 10. A softmax can be applied to this output to convert this output into a probability distribution over the digits and the digit with the highest probability is taken to be the prediction. The Loss function used is Cross Entropy Loss. The Network was written and trained in Pytorch.

## 2.3 The Biologically Constrained RNN

This ANN(Artificial Neural Network) used is adapted from the RNN used in [25]. The RNN receives a set of $N_{in} = 128$ scalar time-varying inputs $u(t)$ and produces $N_{out}$ time varying outputs $z(t)(N_{out} \times 1$ vector), where inputs encode task-relevant sensory information and outputs typically represent a decision variable or probability distribution(Figure 2.6. Each unit of an RNN can be interpreted as

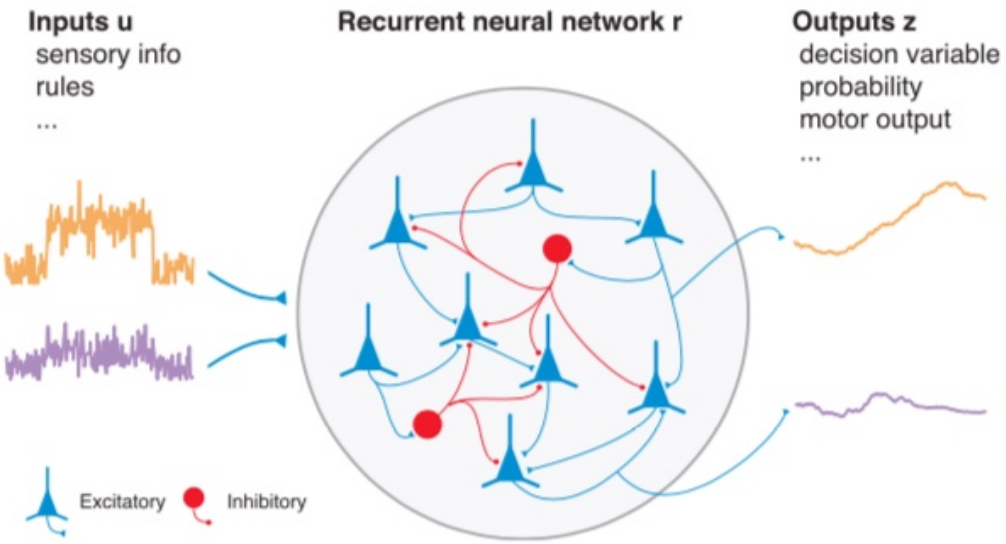


Figure 2.6: RNN with Excitatory and Inhibitory Neurons

the temporally smoothed firing rate of a single neuron or the spatial average of a group of similarly tuned neurons. RNNs considered are whose N firing rates $r(t)$ ($N \times 1$ vector) are related to their corresponding currents $x(t)$ ($N \times 1$ vector) by the rectified linear f-I curve $[x]+ = \max(x, 0)$, which maps arbitrary input currents to positive firing rates: $x$ if $x > 0$ and 0 otherwise.

From [25], we have equations modelled on recurrent neural networks (RNNs) that simulate the dynamical and computational mechanisms seen in large neural populations. The units of an RNN can thus be seen as the temporal or ensemble average of one or more co-tuned spiking neurons. Therefore, complete access to the activity and connectivity of the circuit, and the ability to manipulate them in arbitrary ways, make such neural networks a suitable proxy for biological circuits and serve as a valuable tool for research. The actual dynamical equation that is simulated by the continuous-time RNNs is :

$$\tau \dot{\boldsymbol{x}} = x + W^{rec}\boldsymbol{r} + W^{in}u + \sqrt{2\tau\sigma^2_{rec}\xi}$$

where $\boldsymbol{x}(t)$ is the current in the neurons, $\boldsymbol{r}(t)$ is the firing rate given by, $r = [x]_+ = \max(\boldsymbol{x}, 0)$, $\tau$ is the time constant associated with the network and the output $z$ is given by:

$z = W^{out}\boldsymbol{r}$

or more explicitly the current components $x_i$'s dynamics and the firing rate components $r_i$ are given by

$$\tau \frac{dx_i}{dt} = x_i + \sum_{j=1}^{N} W_{ij}^{rec} r_j + W_i^{in} u + \sqrt{2\tau\sigma_{rec}^2}\xi$$
$$r_i = [x_i]_+ = \max(x_i, 0)$$

and the output components $z_l$ are given by:

$$\sum_{l=1}^{N} W_{li}^{out} r_i$$

for $i = 1, \ldots, N$ and $l = 1, \ldots, N_{out}$.$W^{in}$ is an $N \times N_{in}$ (here $N_{in} = 1$ and $N = 100$) matrix of connection weights from the inputs to network units, $W^{rec}$ is an $N \times N$ matrix of recurrent connection weights between network units, $W^{out}$ is an $N_{out} \times N$ matrix of connection weights from the network units to the outputs, and $\xi$ are $N$ independent Gaussian white noise processes with zero mean and unit variance that represent noise intrinsic to the network.

In these equations,

- $N = N_{rec} = N_{Excitatory} + N_{Inhibitory}$ : and Dale's principle is given by:

  $\frac{N_{Excitatory}}{N_{Inhibitory}} = \frac{51}{13}$ ,because $N = 64$

- $W^{in} = \{w^{in}\}_{N \times N_{in}}$ has only positive elements. Here, $N_{in} = 28$

- $W^{rec} = \{w^{rec}\}_{N \times N}$ has EE, EI, IE and II connections as follows $\begin{bmatrix} EE & IE \\ EI & II \end{bmatrix}$

These EE, EI, IE and II connections are represented in $W^{rec}$ matrix as $51 \times 51, 13 \times 51, 51 \times 13$ and $13 \times 13$ sub-matrices or blocks respectively. Both EE and EI connections are from excitatory neurons and therefore the weights are positive, whereas IE and II connections are from inhibitory neurons and therefore has negative weights.

- $W^{out} = \{w^{out}\}_{N_{out} \times N}$ is a partial identity matrix. Here, $N_{out} = 10$ and $W^{out}$ is a partial identity matrix, that is $w^{out}_{i,j} = 0 \forall i \neq j$.

However, machines can not operate in continuous time. So to simulate continuous processes in computer programs, we use infinitesimally small but discrete time steps. That is, for this network, In practice, the continuous-time dynamics as explained in the equations 2.1, 2.2. and 2.3, are discretized to Euler form (which is denoted by the subscript $t$ which represents the time step, such as $x_t = x(t.\Delta t)$ for a time-dependent variable $x$.

$$x_t = (1 - \alpha)\boldsymbol{x}_{t-1} + \alpha(W^{rec}\boldsymbol{r}_{t-1} + W^{in}u_t) + \sqrt{2\alpha\sigma^2_{rec}}\boldsymbol{N}(0, 1)$$
$$\boldsymbol{r}_t = [\boldsymbol{x}_t]_+,$$
$$\boldsymbol{z}_t = W^{out}\boldsymbol{r}_t$$

where $\alpha = \Delta t/\tau$ and $\boldsymbol{N}(0, 1)$ are normally distributed random numbers with zero mean and unit variance, sampled independently at every time step.

Dale's principle [21], which is based on observations in the mammalian cortex, states that cortical neurons have purely excitatory or purely inhibitory effects on postsynaptic neurons. Moreover, excitatory neurons outnumber inhibitory neurons by approximately 4 times. In a rate model with positive firing rates such as the one given by Eqs 2.1-2.3, a connection from unit $j$ to unit $i$ is excitatory if

$W_{ij}^{rec} > 0$ and inhibitory if $W_{ij}^{rec} < 0$. A unit $j$ is excitatory if all of its projections on other units are zero or excitatory, i.e., if $W_{ij}^{rec} > 0$ for all $i$; similarly, unit $j$ is inhibitory if $W_{ij}^{rec} < 0$ for all $i$.

Besides assigning units separately as excitatory and inhibitory populations, we can also constrain their pattern of connectivity. This can range from simple constraints such as the absence of self-connections to more complex structures derived from biology. Local cortical circuits have distance, layer ([36], [37], [38], [39]) , and cell-type ([40], [41], [42], [43])( dependent patterns of connectivity and different over- all levels of sparseness for excitatory to excitatory, inhibitory to excitatory, excitatory to inhibitory, and inhibitory to inhibitory connections ([44], [45]). Here we fix the density of connections in the trained network to impose some known biological structure on trained networks. We also fix a subset( 2%) of the connection weights to predefined values which are set to 0 during training with the remaining weights being plastic, that is, weights that can change during training. The other weights don't change during training(Appendix A)

The inputs and outputs are the same as for the Unconstrained LSTM from the previous section.

Additional Training Details along with Hyper-parameters are given in Appendix A.

## 2.4 The GAN

GANs are a framework for teaching a Machine Learning model to capture the training datas distribution so we can generate new data from that same distribu-

tion. GANs were developed by Ian Goodfellow in 2014 and first described in the paper Generative Adversarial Nets [34]. They are made of two distinct models, two separate neural networks that compete with each other in a game, specifically a zero sum game, a Generator and a Discriminator. The generator is trained to produce fake images that look like the training images. On the other hand, the discriminator is trained to look at an image and output whether or not it is a real training image or a fake image from the generator. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics.

During training, the generator is constantly trying to outsmart the discriminator by generating better and better fakes, while the discriminator is working to become a better detective and correctly classify the real and fake images. The equilibrium of this game is when the generator is able to produce near perfect pseudo samples that look as if they came directly from the training data, and the discriminator is left to always guess with at most 50% confidence that the generator output is real or fake.

The flowchart in Figure 2.7 outlines how GANs learn.

Let $x$ denote a data sample, that is, an image, Let $D(x)$ be the Discriminator network which outputs the probability that $x$ came from training data as opposed to the generator. Naturally $D(x)$ should be high when $x$ comes from training data and low when $x$ comes from the generator. $D(x)$ is thus trained to be a de facto binary classifier.

Let $z$ be a vector in the latent space vector, sampled from a standard normal distribution, which is passed as input to the Generator network $G$. $G(z)$ represents

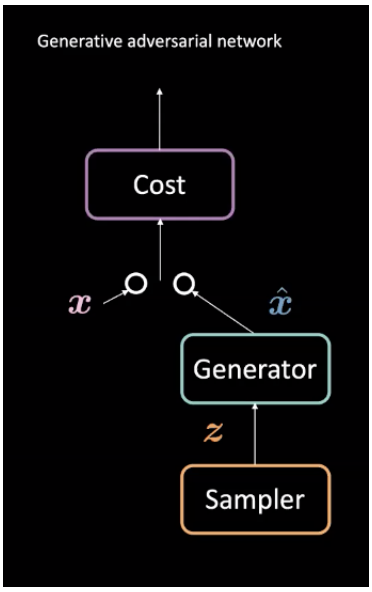


Figure 2.7: Generative Adversarial Network Architecture, where $x$ represents Real Samples while $\hat{x}$ represents Generated Samples

the generator function which maps the latent vector $z$ to the data-space. The goal is to train $G$ to approximate the distribution that the training data comes from ($P_{data}$) so it can generate fake samples from that estimated distribution.Thus, $D(G(z))$ then is the probability that the output of the generator $G$ is a real image. $D$ and $G$ play a minimax game in which $D$ tries to maximize the probability it correctly classifies reals and fakes, ($\log D(x)$), and $G$ tries to minimize the probability that $D$ will predict its outputs are fake ($\log(D(G(x)))$. As in the paper, the Loss function that is optimized for training the GAN is given by :

$$\min_G \max_D V(D,G) = E_{x\sim P_{data}(x)}[\log D(x)] + E_{z\sim P_z(z)}[\log(1 - D(G(z)))]$$

In the GAN used for the experiments:

- The Generator $G$ : A vector $z$ of size 100 is sampled from a multivariate standard normal distribution. This input is sent to a fully connected linear layer which takes its dimensionality to 32, followed by the Leaky ReLU function with a negative slope of 0.2, and then a dropout layer with a dropout rate of 0.3. This is the output from the LeakyReLU function is then passed to another fully connected layer followed by the Leaky ReLU function with a slope of - 0.2 again, which takes its dimensionality to 64. This is repeated twice more, at the end of which the dimensionality changes to 128. Finally, another fully connected linear layer takes the dimensionality to 784 followed by the Hyperbolic Tan function which gives us the pseudo sample. This 784 dimensional vector is reshaped to $28 \times 28$ to have the same dimensions as the MNIST Images.

- The Discriminator $D$ : The input image of size $28 \times 28$ is first flattened into a vector of size 784. First, a fully connected linear layer reduces the dimensionality of this input from 784 to 128 followed by a Leaky ReLU activation function and a dropout layer with a dropout rate of 0.3. Like with the generator, this repeats twice more after which the dimensionality to 32. The final fully connected linear layer then takes this 32-dimensional vector and returns a scalar value. A sigmoid applied to this scalar value converts this into a probability, that is the probability that the input came from true training dataset.

## 2.5 Deep Generative Replay

This section describes the actual training framework that mitigates Catastrophic Forgetting during incremental class learning. For the context of incremental class learning with MNIST digits define a task $T_i$ as optimizing a model towards an objective on data distribution $D_i$, from which training samples $(x_i, y_i)$ come, with all samples only coming from the classes tt is desired for the model to learn. Within this framework we define the sequence of tasks to be solved as a task sequence $T = (T_1, T_2, T_3, T_4, T_5)$ such that task $T_1$ requires learning the digits 0 and 1, task $T_2$ .digits 2 and 3, $T_3$ digits 4 and 5, $T_4$ digits 6 and 7 and $T_5$, digits 8 and 9. The paper describes the algorithm for N tasks.

As in [19] , the Deep Generative Replay framework involves scholars. Scholars are capable of learning new tasks as well as teaching other scholars what they learn. As defined in the paper a scholar $H$ is a tuple $(G, S)$ where $G$ is a generative model that produces real-like samples and the solver $S$ is a task solving model parameterized by $\theta$.

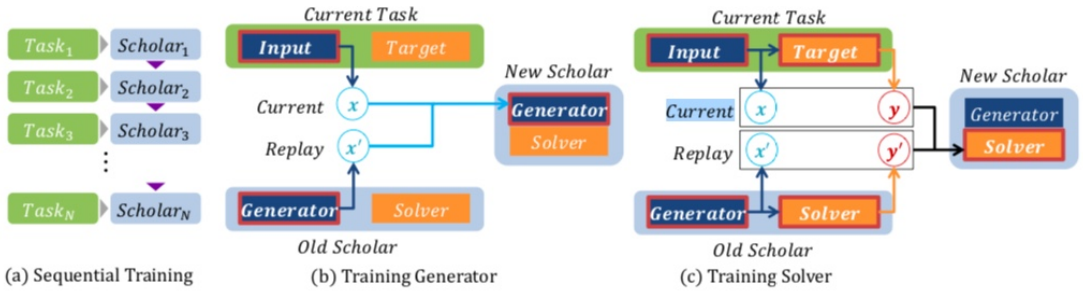


Figure 2.8: Deep Generative Replay : Sequential Training of Scholar Models [19]

The solver has to perform all tasks in the task sequence $T$. The final objective

therefore is to minimize the unbiased sum of losses among all tasks in the task sequence $E_{(x,y)\sim D}[L(S(x;\theta),y)]$, where $D$ is the entire data distribution and $L$ is the loss function. We consider sequential training on our scholar model, training a sequence of scholar models $(H_i)_{i=1}^{5}$ where the $n-th$ scholar $H_n$, $(n>1)$ learns the current task $T_n$ and the knowledge of previous scholar $H_{n-1}$. Training the scholar model from another scholar involves two independent procedures of training the generator and the solver. First, the new generator receives current task input $x$ and replayed inputs $x$ from previous tasks. Real and replayed samples are mixed together at a ratio of $1:9$. That is, for each new sample from the current task, 9 pseudo samples from the last generator are mixed with the data. The generator learns to reconstruct cumulative input space, It is assumed that past data is completely discarded so the generator is trained to mimic both current inputs and the generated samples from the previous generator. The generator thus reproduces cumulative input distribution of all encountered examples so far. The new solver is trained to couple the inputs and targets drawn from the same mix of real and replayed data. Here, the replayed target is past solvers response to replayed input.

The loss function of the $i$-th solver thus becomes :

$$L_{train}(\theta_i) = rE_{(x,y)\sim D_i}[L(S(x;\theta_i),y)] + (1-r)E_{x''\sim G_{i-1}}[L(S(x';\theta_i),S(x''\theta_{i-1}))]$$

where $\theta_i$ represents the network parameters of the $i$-th scholar while $r$ represents the fraction of real(new) data.

.

# Chapter 3

# Results

This chapter describes the main results of the experiments.

## 3.1 Prelliminary Results

### 3.1.1 Training Using Input Size 1 ($N_{in}$) with 784 Time Steps

While first trying to establish baselines, the LSTM as well as the Biological RNN were both trained to learn all digits, by flattening the image and using 784 Time Steps with an Input Size ($N_{in}$) as 1. However, neither of the networks were able to learn well, with test accuracy not exceeding 80% for either. That both of the networks failed to learn was possibly due to the long sequence length, and consequently the Vanishing Gradient Problem in RNNs, which makes it hard for the network for recall information from too far in the past, inspite of LSTM memory cells.

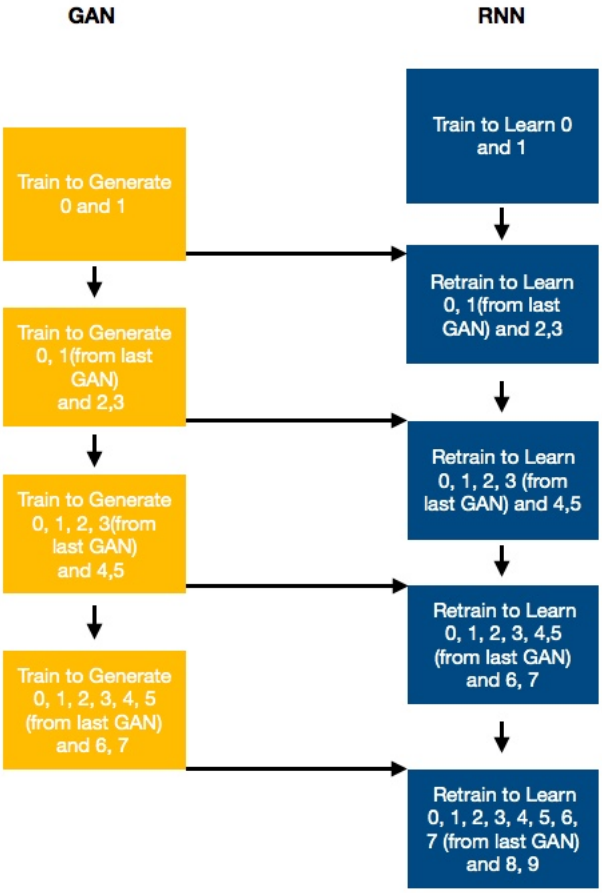


Figure 3.1: Sequential Training Algorithm

### 3.1.2 Benchmarks

The performance of both networks on the testset after being trained to learn all digits are used the baselines.

Test Accuracy of the LSTM on MNIST Digits - 99.99%

Test Accuracy of the Biological RNN on MNIST Digits - 99%

It is clear that both of these networks are capable of learning to classify MNIST digits.

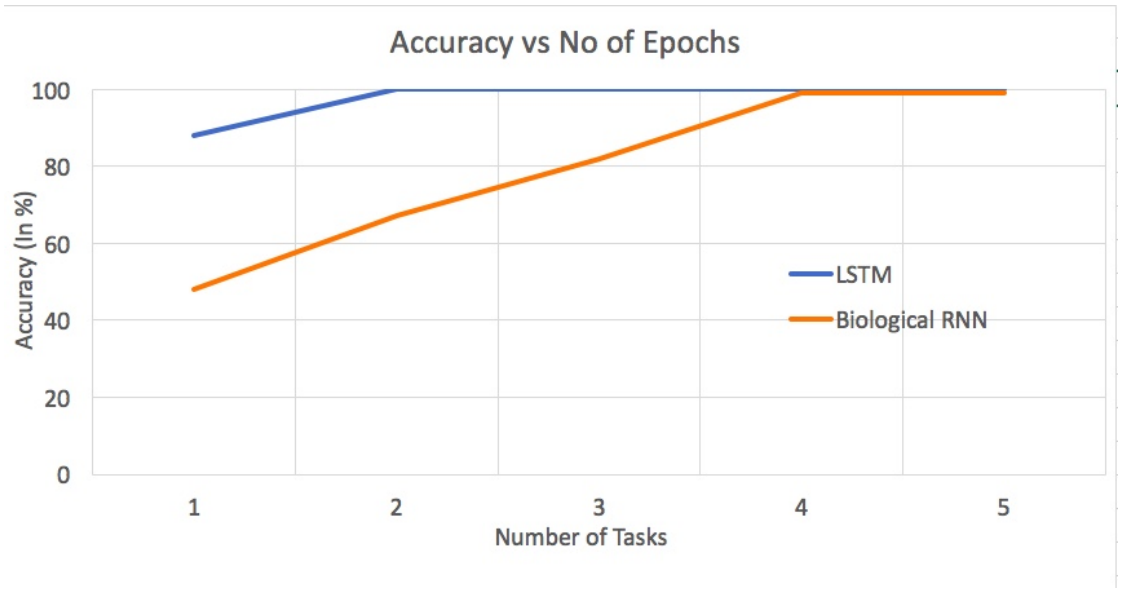


Figure 3.2: Test Performance of the LSTM and the Biological RNN when trained to learn all digits at once

From the above figure, it can be observed that while both networks manage to learn to classify very well, it takes the Biologically Constrained RNN 4 epochs to be able to classify with near perfect accuracy, while the LSTM is able to perform as well after just 2 epochs of training. This is likely to be a result of the Biological Constraints themselves which makes learning of atypical data more difficult.

## 3.2 Main Results - Catastrophic Forgetting

In Figure 3.3, one can clearly see, that Deep Generative Replay mitigates Catastrophic Forgetting to a very large extent in the LSTM. Even after learning all 5 tasks, the network is able to classify digits with an accuracy of roughly $\sim 80\%$. However, it does not perform as well as the Model in the Deep Generative Replay as can be seen in the perfomance of the model in Figure 3.6 on the same incremental learning task and the same dataset, which manages to maintain an

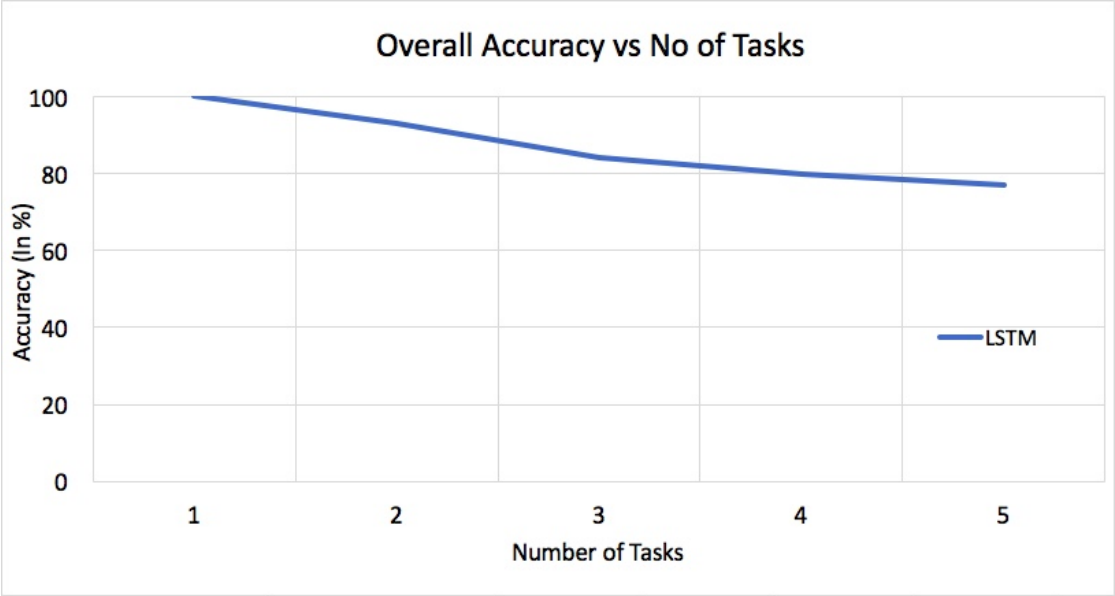


Figure 3.3: Test Performance of the LSTM as the Number of Tasks(Classes) Increases

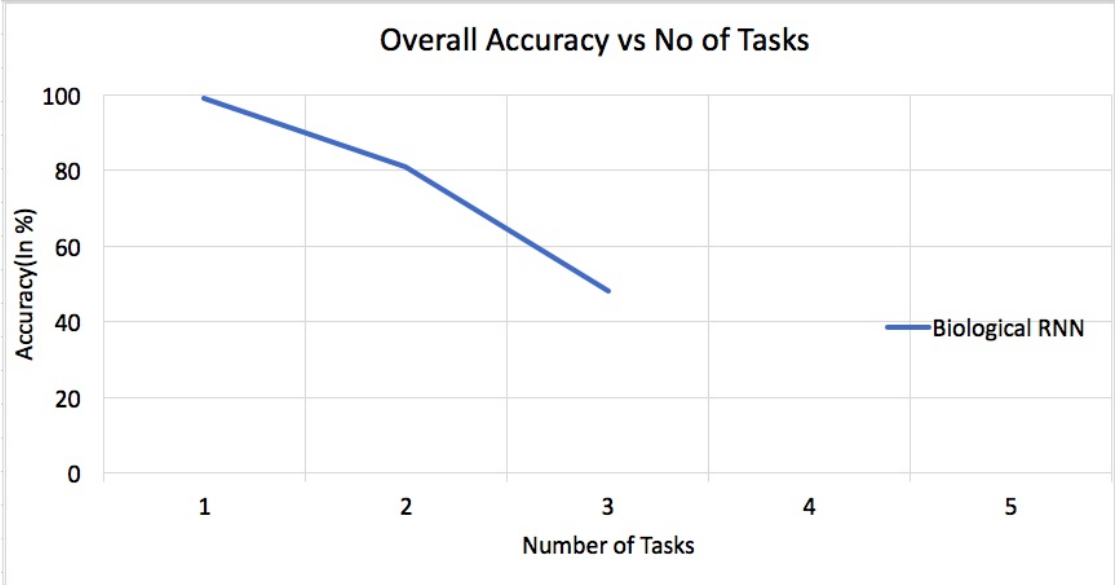


Figure 3.4: Test Performance of the Biologically Constrained RNN as the Number of Tasks(Classes) Increase

overall of over 95% even after learning all 5 tasks. This is likely due to the fact that the solver used here is an LSTM, while that used in the Deep Generative

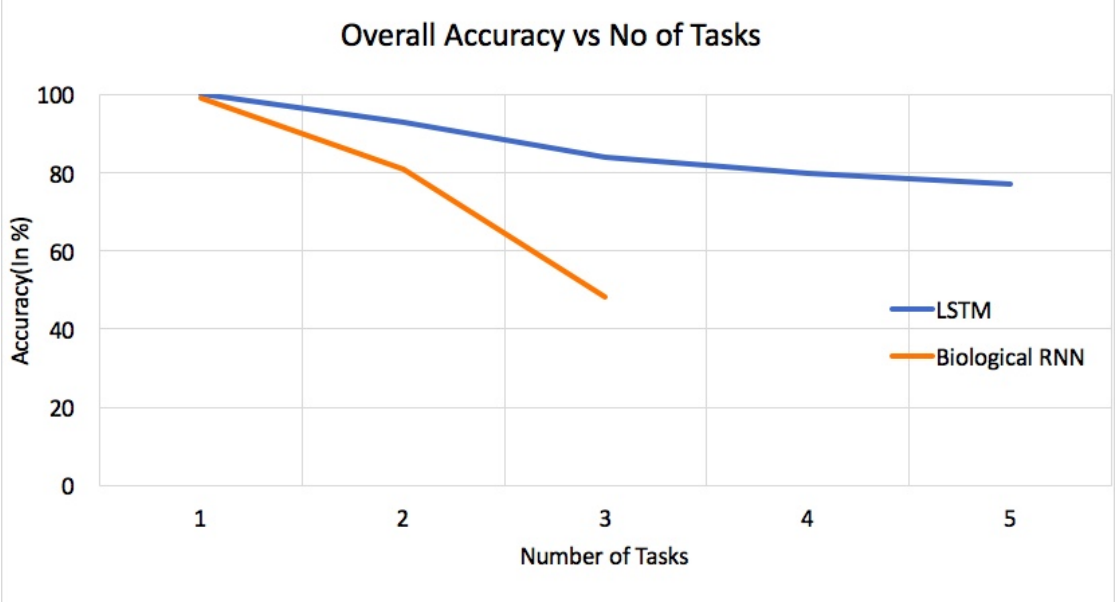


Figure 3.5: Test Performance of the LSTM and the Biological RNN compared

Replay paper is a CNN(Convolutional Neural Network. It is possible that this is due to the quality of the generator and generated pseudo samples. Training for longer or a different architecture for the GAN or a different generative model altogether might help bring performance up to par with that achieved in the paper..

In the case of the Biological RNN, the overall accuracy on the test set,while around the same as the LSTM for the first task, drops to 82 % after training for 2 tasks(4 classes), that is learning the digits 0, 1, 2 and 3 vs 92 % in the case of the LSTM. This significantly worse performance of the Biological RNN could be attributed to the Biological Constraints. Further, after learning 3 tasks, the test performance of the Biological RNN degrades dramatically. The test accuracy drops to 48.2% which effectively means that the network is not able to learn at all as even randomly initialised untrained networks achieve accuracy rates of around 50%. This degradation in performance can be attributed to the biological

constraints, which makes learning progressively difficult as the number of tasks increase until learning becomes very hard. In this case too, while it is plausible that a better trained generative model could lead to improved accuracy Thus, catastrophic forgetting occurs. At this point, training naturally had to be stopped since learning more tasks likely would lead to even further decline in performance.

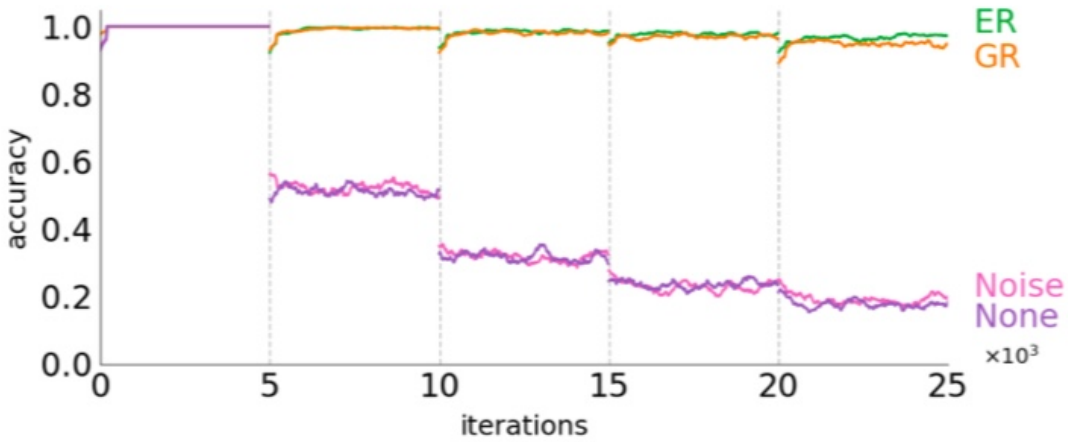


Figure 3.6: Test Performance of the Solver from the Deep Generative Replay Paper [19] on Incremental Class Learning - MNIST Digits where GR refers to Generative Replay

Figure 3.7 shows how Overall Performance as well as performance on old data as well as new data changes, as the number of Tasks Increase.. While performance on the new task does also decline over time, it is performance on old tasks that degrades significantly, and accounts for much of the decline in overall accuracy. This is to be expected, since the model can only be trained on pseudo samples of past data, while it can train on actual data for the current task.

Another important observation was made while analyzing the performance of

the networks. For both networks, besides the fact that the older tasks account for the majority of errors, it was also observed that out of the errors on the older tasks, there were more errors made on digits that came from older tasks than from more recent tasks. For example, after training to learn Task 3, that is learning digits 4 and 5, it was observed during testing that the network incorrectly classified the digits 0, 1, 2 and 3 more often than 4 and 5. However, even out of the digits 0, 1, 2 and 3, the network incorrectly classified 0 and 1 more often than 2 and 3. This is since the 0 and 1 samples that the solver is trained on are samples generated by a Generative Model, in particular a generative model for which the training samples for 0 and 1 are generated by an older generative model that was trained on actual samples of 0 and 1. It is obvious that the more times the distribution a approximated from a previous approximation, the further the new approximation will be from the true distribution.

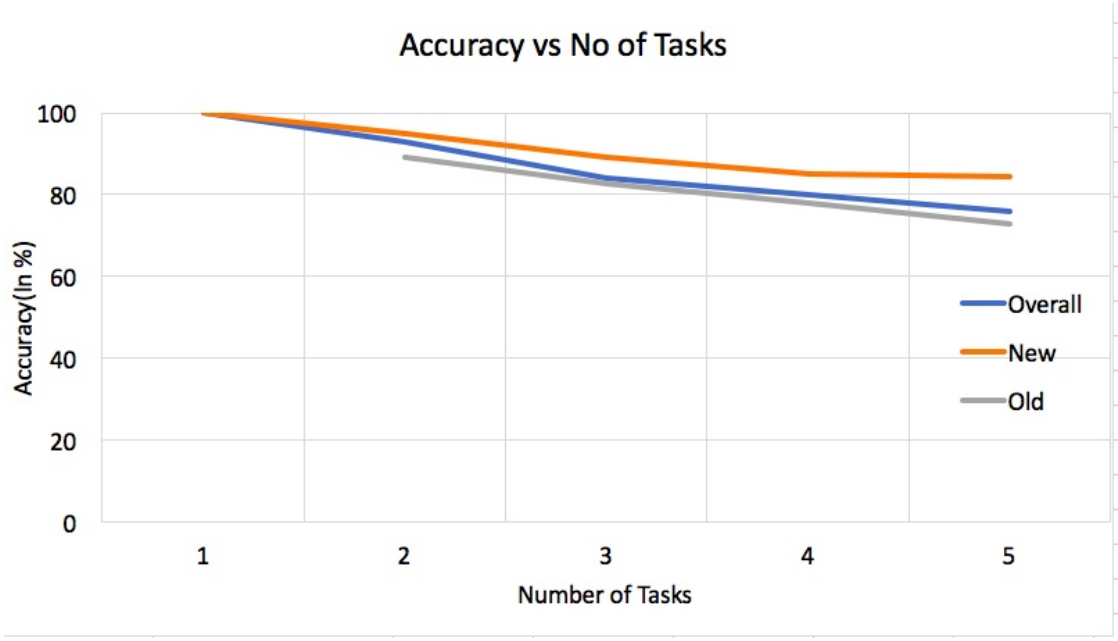


Figure 3.7: Test Performance on Current Tasks, Old Tasks and Cumulatively as the Number of Tasks Increases

## 3.3 Intermediate Observations

This section contains important Intermediate Results and observations. The graph in Figure 3.8 shows the number of epochs it took for the GAN to learn well enough to be able to produce samples of reasonable quality. As expected, it takes more and more epochs for the GAN to learn as the number of classes increases.

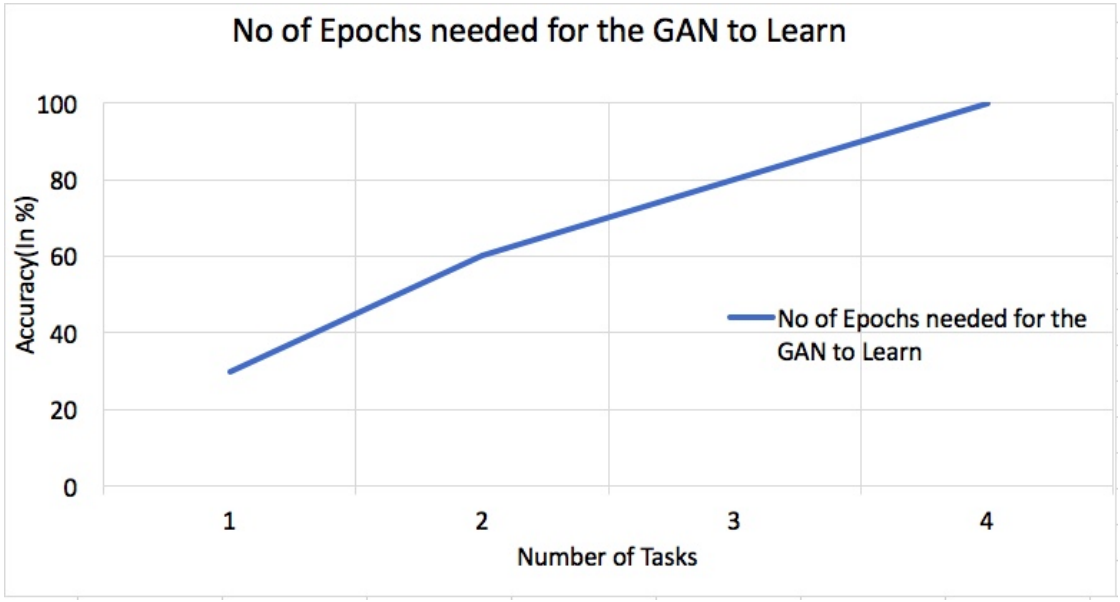


Figure 3.8: Number of Epochs Required for the GAN to Learn

Figure 3.9 shows examples generated by the first generator and Figure 3.10 shows some examples generated by the second generator.

Figures 3.11 and 3.12 show t-SNE[35] visualisations of true samples against samples generated by the GAN. t-SNE(t-distributed Stochastic Neighbor Embedding) is a dimensionality reduction technique that makes visualisation of high dimensional data convenient, by embedding the data in 2-D space. It is a ma-

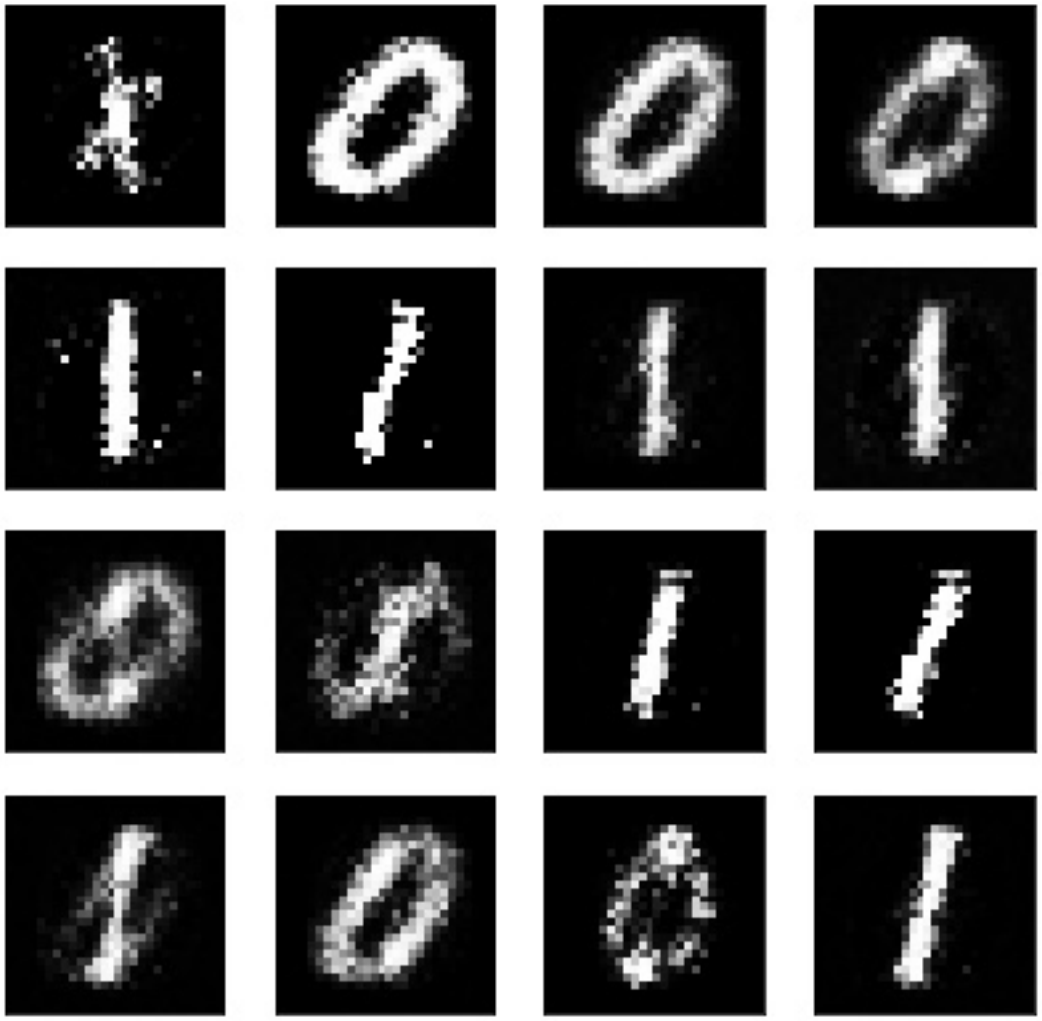

Figure 3.9: Pseudo Samples from the First Generator

chine learning algorithm that converts pairwise distance between data points to joint probabilities and minimizes the Kullback-Leibler divergence between the joint distributions of the low-dimensional embedding, usually 2-D and the original high-dimensional data [35]. In the figures, the color bar on the right represent the 10 digits(classes). Each dot represents the embedded representation of the corresponding image sample in 2D space while the color of the dot represents the class

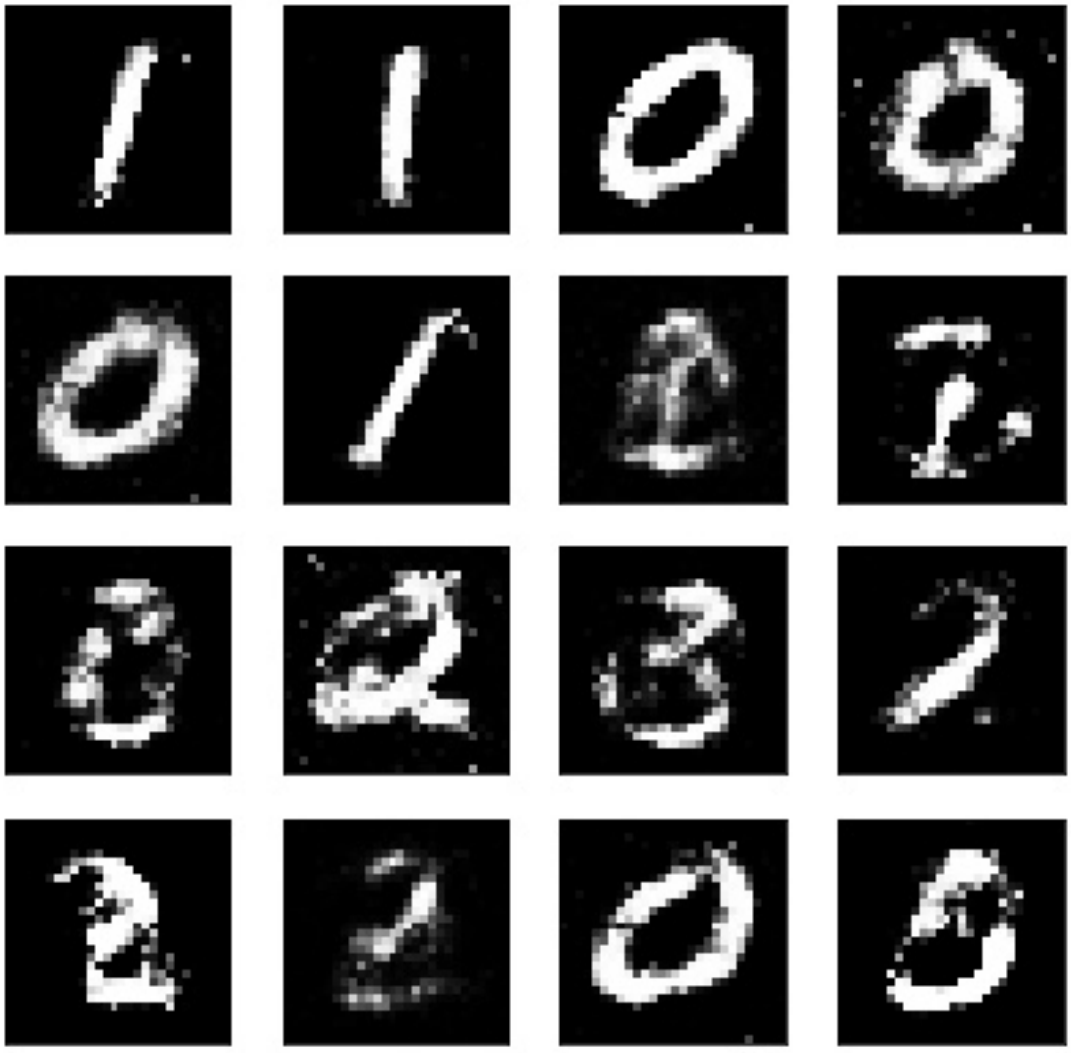

Figure 3.10: Pseudo Samples from the Second Generator

it belongs to. From Figure 3.11, it can be observed that samples of 0 and 1 are clearly separate from each other while GAN samples of 0 and 1 aren't as clearly disparate from each other. That samples are muddled together is what makes it harder for the subsequent solver to discern and learn. Similarly, in Figure 3.12, one can see that True Samples of 0,1,2 and 3 are clearly distinguishable from each other while in the case of samples that are generated by the GAN, samples from

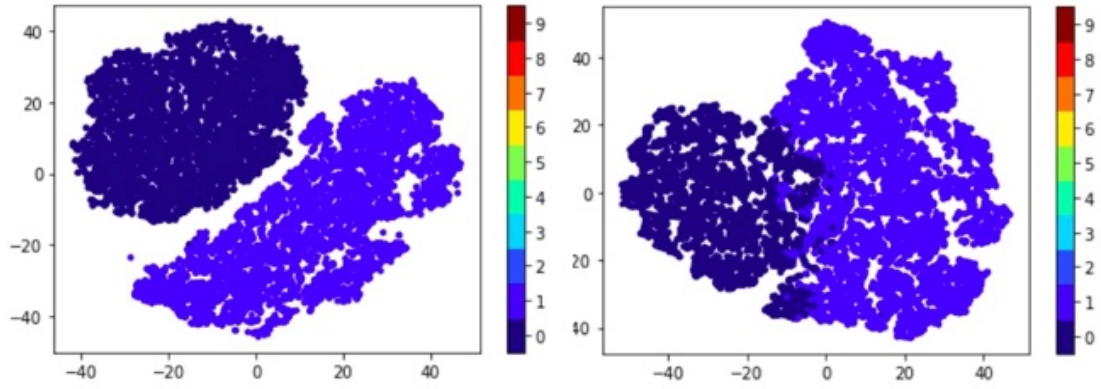

Figure 3.11: Left: True Samples of 0 and 1; Right : 0 and 1 Samples from the First Generator

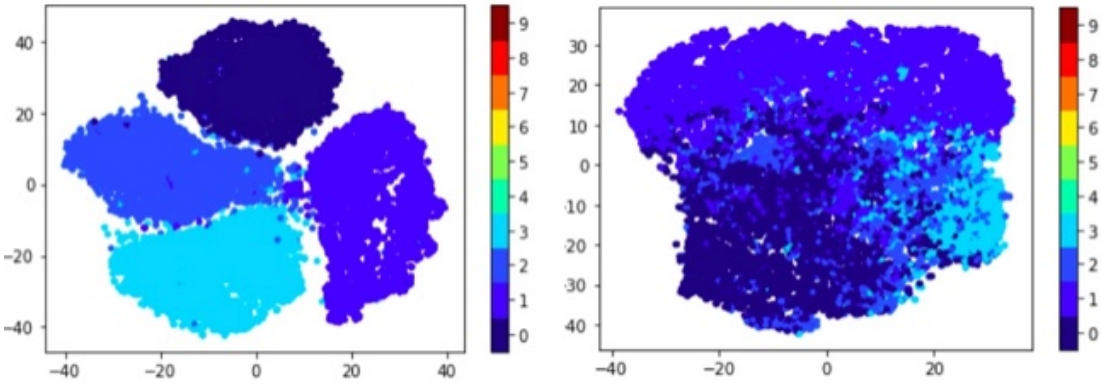

Figure 3.12: Left: True Samples of 0,1,2 and 3; Right : 0, 1, 2, and 3 Samples from the Second Generator

different classes are muddled together, as can be seen in the Visualisations.

# Chapter 4

# Conclusion

In this thesis, the Deep Generative Replay framework is used to Mitigate Catastrophic Forgetting in LSTM - RNNs as well as in Biologically Constrained RNNs. In the case of the LSTM, test performance for incremental class learning was only slightly below that of the performance in the Deep Generative Replay and thus he results show that Deep Generative Replay is highly effective and catastrophic does not occur to any significant extent. This is a novel result which confirms the applicability and effectiveness of Deep Generative Replay for LSTMs as well. It is likely that performance could be improved further with more hyperparameter tuning or a different choice of architecture for the Generator and Discriminator or perhaps a different generative model altogether,

In the case of the Biologically Constrained RNN, that is the RNN that adheres to Dale's Principle with Constrained Synaptic Plasticity and Connectivity patterns, although, not to the same degree, Deep Generative Replay did curtail Catastrophic Forgetting of the first task after learning the second task. However, this performance could not be replicated beyond the second task. It becomes very

difficult for the network to learn at all, beyond the second task. There could be multiple possible reasons for this. The imposition of Dale's Law at the beginning of training could be what makes learning difficult. Alternatively, it is possible that a plasticity rate of 98% as against 100% for unconstrained networks could be the cause for the sudden plunge in performance. Both of those constraints reduce the number of degrees of freedom and thus are likely to hinder learning to some extent. Finally, like with the unconstrained case, a different Generative Model or different architectures for the Generator and Discriminator might lead to better performance. There is also the possibility that Deep Generative Replay itself may not be an appropriate framework for Continual Learning in such Biologically Constrained RNNs It is hard to conclude why performance drops. Thus, this result encourages further research and exploration for the future. It would be interesting to investigate how varying rates of plasticity, different GAN Architectures, and other training paradigms could make Continual Learning possible. Being able to mitigate Catastrophic Forgetting in Biologically Inspired Neural Networks in general, would provide highly valuable insights into Memory and Retention Mechanisms in the Brain.

While Deep Generative Replay itself might not have proved effective for Continual Learning in such Neural Networks, the experiments conducted, provide other significant insights. The most significant of these observations is that the Biological RNN was able to learn to classify handwritten digits nearly as well as the LSTM, when being trained on all of the data. That this biologically constrained network which was originally written to take only simulated sensory data as inputs is able to learn to classify real handwritten digits, is quite significant in itself. It is perhaps an indication that the network architecture of this Neural Network might

simulate parts of the Brain accurately and be a valuable tool for investigation.

# Appendix A

# Training Hyperparameters for the Biological RNN

| **Parameter** | **Setting/Value** |
|---|---|
| GPU: | NO |
| INIT SEED: | 100 |
| DISTRIBUTION (WIN): | UNIFORM |
| DISTRIBUTION (WREC): | GAMMA |
| DISTRIBUTION (WOUT): | UNIFORM |
| NIN/N/NOUT: | 28/64/10 |
| DALE?S LAW: | E/I = 51/13 |
| INITIAL SPECTRAL RADIUS: | 1.50 |
| TRAIN RECURRENT BIAS: | NO |
| TRAIN OUTPUT BIAS: | NO |
| TRAIN INITIAL CONDITIONS: | YES |
| SPARSENESS (WREC): | P = 0.98, P PLASTIC = 0.98 |

| | |
|---|---|
| E/I POSITIVITY FUNCTION: | RECTIFY |
| HIDDEN ACTIVATION: | RECTIFY |
| OUTPUT ACTIVATION/LOSS: | SOFTMAX/CATEGORICAL CROSS ENTROPY |
| MODE: | BATCH |
| OUTPUT MASK: | NO |
| SIGMA IN: | 0.01 |
| SIGMA REC: | 0.01 |
| RECTIFY INPUTS: | TRUE |
| GRADIENT MINIBATCH SIZE: | 64 |
| VALIDATION MINIBATCH SIZE: | 100 |
| DT: | 1 |
| TAU: | 50 |
| TAU IN: | 100 |
| LEARNING RATE: | 0.001 |
| LAMBDA OMEGA: | 2 |
| MAX GRADIENT NORM: | 1 |
| (THEANO) FLOATX: | FLOAT32 |
| (THEANO) ALLOW GC: | FALSE |

# Bibliography


[1] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521?3526, 2017.

[2] R. Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285?308, 1990

[3] A. Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123?146, 1995

[4] Z. Li and D. Hoiem. Learning without forgetting. In *European Conference on Computer Vision*, pages 614?629. Springer, 2016

[5] Zenke, F., Poole, B. and Ganguli, S. , Continual learning through synaptic intelligence, *ICML?17*, Sydney, Australia, 2017

[6] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pas- canu, R. and Hadsell, R. 2016, Progressive neural networks, arXiv:1606.04671

[7] Part, J. L. and Lemon, O., Incremental on-line learning of object classes using a combination of self-organizing incremental neural networks and deep convolutional neural networks, IROS?16, Workshop on Bio-inspired Social Robot Learning in Home Scenarios, Daejeon, South Korea, 2016

[8] Part, J. L. and Lemon, Incremental online learning of objects for robots operating in real environments, ICDL-EPIROB?17, Lisbon, Portugal, 2017

[9] Soltoggio, A. , ?Short-term plasticity as cause-effect hypothesis testing is distal reward learn- ing?, *Biological Cybernetics* **109**, 75?94, 2015

[10] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019

[11] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Thirty-second AAAI conference on artificial intelligence*, 2018

[12] Maltoni, D. and Lomonaco, V. , Continuous learning in single-incremental-task scenarios., arXiv:1806.08568, 2018

[13] Lomonaco, V. and Maltoni, D. , CORe50: A new dataset and benchmark for continuous object recognition, CoRL, Mountain View, CA., 2017

[14] Krizhevsky, A., Learning multiple layers of features from tiny images, Master?s thesis, University of Toronto, 2009

[15] Parisi, G. I., Tani, J., Weber, C. and Wermter, S. , ?Lifelong learning of humans actions with deep neural network self-organization?, *Neural Networks* **96**, 137?149., 2017

[16] Parisi, G., Tani, J., Weber, C. and Wermter, S. , Lifelong learning of spatiotemporal representations with dual-memory recurrent self-organization, arXiv:1805.10966, 2018

[17] Hertz, J., Krogh, A. and Palmer, R. G. ,Introduction to the Theory of Neural Computation, Redwood City CA: Addison-Wesley, 1991

[18] Seidenberg, M. and Zevin, J., *Connectionist models in developmental cognitive neuroscience: Critical periods and the paradox of success*, In Y. Munakata and M.H. Johnson (Eds.) Processes of change in brain and cognitive development: Attention and Performance XXI. Oxford: Oxford University Press, 2006.

[19] Hanul Shin, Jung Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NeurIPS*, 2017

[20] McClelland, J. L., McNaughton, B. L. and O?Reilly, R. C., ?Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory?, *Psychological Review* **102**, 419?457, 1995

[21] Kumaran, D., Hassabis, D. and McClelland, J. L. , ?What learning systems do intelligent agents need? complementary learning systems theory updated?, *Trends in Cognitive Sciences* **20**(7), 512? 534, 2016

[22] Kumaran, D. and McClelland, J. L. (2012), ?Generalization through the recurrent interaction of episodic memories: A model of the hippocampal system?, *Psychological Review* 119, 573?616

[23] R. C. O?Reilly and K. A. Norman. Hippocampal and neocortical contributions to memory: Advances in the complementary learning systems framework. Trends in cognitive sciences, 6(12):505?510, 2002

[24] S. Ramirez, X. Liu, P.-A. Lin, J. Suh, M. Pignatelli, R. L. Redondo, T. J. Ryan, and S. Tonegawa. Creating a false memory in the hippocampus. Science, 341(6144):387?391, 2013

[25] Song, H.F., Yang, G.R., and Wang, X.-J. (2016). Training Excitatory-Inhibitory Recurrent Neural Networks for Cognitive Tasks: A Simple and FlexibleFramework. PLoS Comput. Biol.12, e1004792.

[26] Eccles JC, Fatt P, Koketsu K. Cholinergic and inhibitory synapses in a pathway from motor-axon collaterals to motoneurones. 1954 J Physiol; 126:524?562

[27] Ercsey-Ravasz M, Markov NT, Lamy C, Van Essen DC, Knoblauch K, Toroczkai Z, et al. A predictive network model of cerebral cortical connectivity based on a distance rule. 2013 Neuron; 80:184?197

[28] Murphy BK, Miller KD. Balanced amplification: A new mechanism of selective amplification of neural activity patterns. 2009 Neuron; 61:635?4

[29] Markram H, Toledo-Rodriguez M, Wang Y, Gupta A, Silberberg G, Wu C. Interneurons of the neocorti- cal inhibitory system. 2004 Nat Rev Neurosci; 5:793?807

[30] Jiang X, Shen S, Cadwell CR, Berens P, Sinz F, Ecker AS, et al. Principles of connectivity among mor- phologically defined cell types in adult neocortex. 2015 Science; 350:aac9462

[31] Markov NT, Ercsey-Ravasz MM, Ribeiro Gomes aR, Lamy C, Magrou L, Vezoli J, et al. A weighted and directed interareal connectivity matrix for macaque cerebral cortex. 2014 Cereb Cortex; 24:17?36

[32] Song HF, Kennedy H, Wang XJ. Spatial embedding of structural similarity in the cerebral cortex. 2014 Proc Natl Acad Sci; 111:16580?16585

[33] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015

[34] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks, 2014

[35] van der Maaten, L. & Hinton, G. E. Visualizing data using t-SNE. J. Mach. Learn. Research 9, 2579?2605, 2008

[36] Levy RB, Reyes AD. Spatial profile of excitatory and inhibitory synaptic connectivity in mouse primary auditory cortex. 2012 J Neurosci; 32:5609?5619.

[37] Potjans TC, Diesmann M. The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model. 2014 Cereb Cortex; 24:785?806

[38] Thomson AM, West DC, Wang Y, Bannister AP. Synaptic connections and small circuits involving excitatory and inhibitory neurons in layers 2?5 of adult rat and cat neocortex: triple intracellular recordings and biocytin labelling in vitro. 2002 Cereb Cortex; 12:936?953

[39] Binzegger T, Douglas RJ, Martin KAC. A quantitative map of the cir- cuit of cat primary visual cortex. 2004 J Neurosci; 24:8441?53

[40] Markram H, Toledo-Rodriguez M, Wang Y, Gupta A, Silberberg G, Wu C. Interneurons of the neocortical inhibitory system. 2004 Nat Rev Neurosci; 5:793?807.

[41] Pfeffer CK, Xue M, He M, Huang ZJ, Scanziani M. Inhibition of inhibition in visual cortex: the logic of connections between molecularly distinct interneurons. 2013 Nat Neurosci; 16:1068?76.

[42] Jiang X, Shen S, Cadwell CR, Berens P, Sinz F, Ecker AS, et al. Principles of connectivity among morphologically defined cell types in adult neocortex. 2015 Science; 350:aac9462.

[43] Wang XJ, Tegn?er J, Constantinidis C, Goldman-Rakic PS. Division of labor among distinct subtypes of inhibitory neurons in a cortical microcircuit of working memory. 2004 Proc Natl Acad Sci U S A; 101:1368?1373.

[44] Fino E, Packer AM, Yuste R. The logic of inhibitory connectiv- ity in the neocortex. 2012 Neurosci; 19:228?237.

[45] Karnani MM, Agetsuma M, Yuste R. A blanket of inhibition: Functional inferences from dense inhibitory connectivity. 2014 Curr Opin Neurobiol; 26:96?102.