Achieving Continual Learning in Deep Neural Networks through Pseudo-Rehearsal

Craig Robert Atkinson

a thesis submitted for the degree of

Doctor of Philosophy

at the University of Otago, Dunedin,

New Zealand.

14 September 2020

Abstract

Neural networks are very powerful computational models, capable of outperforming humans on a variety of tasks. However, unlike humans, these networks tend to catastrophically forget previous information when learning new information. This thesis aims to solve this catastrophic forgetting problem, so that a deep neural network model can sequentially learn a number of complex reinforcement learning tasks. The primary model proposed by this thesis, termed RePR, prevents catastrophic forgetting by introducing a generative model and a dual memory system. The generative model learns to produce data representative of previously seen tasks. This generated data is rehearsed, while learning a new task, through a process called pseudo-rehearsal. This process allows the network to learn the new task, without forgetting previous tasks. The dual memory system is used to split learning into two systems. The short-term system is only responsible for learning the new task through reinforcement learning and the long-term system is responsible for retaining knowledge of previous tasks, while being taught the new task by the short-term system.

The RePR model was shown to learn and retain a short sequence of reinforcement tasks to above human performance levels. Additionally, RePR was found to substantially outcompete state-of-the-art solutions and prevent forgetting similarly to a model which rehearsed real data from previously learnt tasks. RePR achieved this without: increasing in memory size as the number of tasks expands; revisiting previously learnt tasks; or directly storing data from previous tasks. Further results showed that RePR could be improved by informing the generator which image features are most important to retention and that, when challenged by a longer sequence

of tasks, RePR would typically demonstrate gradual forgetting rather than dramatic forgetting. Finally, results also demonstrated RePR can successfully be adapted to other deep reinforcement learning algorithms.

Acknowledgements

First and foremost I would like to thank my supervisors Brendan, Lech and Anthony, whose guidance has been invaluable throughout this endeavour. I would especially like to thank them for their undying commitment and interest over the years, always being there to contribute and giving me the freedom to steer my research in directions that captivate me. I have always appreciated the funny banter before meetings and the intriguing tangents that followed. I would like to extend my thanks to the entire University of Otago Computer Science Department, whose friendly staff have always made themselves available to assist myself and other students in any way they can.

I would also like to express a huge thanks to my partner Steph who has unconditionally supported me through my whole university education. Her enduring support has been vital to my success, whether that be giving countless hours helping me proof read or keeping me fed when I was snowed under with work. I would also like to thank her for trying to understand my nonsensical ramblings about computer science and encouraging me to get some much needed glimpses of sunlight.

A special thanks to my whole family; I would not be who I am, nor where I am, without you. I am exceptionally grateful to my parents who provided me with everything I could ever ask for, encouraged me to be the best person I can be and most importantly served as flawless role models to me. I would also like to thank my friends who have always provided much needed distractions from my work, whether that be in person or online.

Finally, I would like to acknowledge the NVIDIA Corporation for donating the TITAN X GPU used in this research and the University of Otago for

granting me a doctoral scholarship which has allowed me to pursue my research interests.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Continual Learning

The ability to persistently learn over time, integrating new knowledge with previously learnt knowledge, is known as continual learning (or alternatively lifelong learning) (Parisi, Kemker, Part, Kanan, and Wermter, 2019; Thrun and Mitchell, 1995). In particular, new information should be consolidated into long-term memory without forgetting previously learnt knowledge.

One of the common continual learning tests used in machine learning is sequential learning. This requires the model to learn a number of tasks in order, where the model only has access to data from the task currently being learnt. After learning all of the tasks, the model should be able to perform well on all of those tasks. Although this problem seems relatively trivial, neural network models severely struggle as they tend to only remember what they have most recently learnt. This thesis focuses on overcoming this limitation so that neural networks can continuously learn.

Achieving continual learning in neural networks is important as experts believe it is one of the conditions which need to be satisfied for accomplishing machine intelligence (Legg and Hutter, 2007). Not only does human intelligence demonstrate the need for continual learning but so does animal intelligence. For example, when observing how cats learn by trial and error to escape from a puzzle box, Thorndike (1898) found that the cats could recall the appropriate set of actions needed to escape the box even after they had not been exposed to the task for months. Although it appears that biological forms of intelligence avoid forgetting, this is not entirely the case because the human brain still undergoes graceful forgetting of information which is deemed irrelevant. For example, Pallier, Dehaene, Poline, LeBihan, Argenti, Dupoux, and

Mehler (2003) discovered that Korean adults, adopted into French families as children, showed no residual knowledge of the Korean vocabulary. Together, these psychological results suggest that continual learning is an important hurdle to overcome for machine intelligence, although it is acceptable for a continual learning system to demonstrate small amounts of forgetting over long time periods.

Continual learning comes with a number of potential benefits for machine intelligence; complex tasks could be made easier to learn when important building blocks for the task have already been learnt, and information about multiple tasks could be compressed into a single model by allowing it to share similar computations. Therefore, achieving continual learning in neural networks is an instrumental step towards effective machine intelligence.

1.2 Catastrophic Forgetting

Artificial neural networks are a very powerful computational model with the potential to perform comparably to and in some instances better than biological brains. Neural networks comprise a number of distributed units, connected by weights. These units are usually arranged in a layered, feedforward structure where the presence of multiple layers define a deep neural network. Neural networks learn by using backpropagation to iteratively update their weights so that their performance on the current task is improved. Due to the network's distributed structure, knowledge of a task is stored across all of the weights in the network. This means that when these weights are updated while learning a new task, the knowledge of previously learnt tasks is quickly overwritten. The neural network's tendency to forget previous knowledge while learning new knowledge is known as catastrophic forgetting (McCloskey and Cohen, 1989). Carpenter and Grossberg (1988) suggested the analogy of a person moving city. The person has learnt a lot of knowledge specific to the city they came from, such as optimal routes between the city's popular landmarks. When moving to the new city, they must learn similar information specific to the new city. If the person went back to visit their old home city, they should still remember how to navigate it. However, if catastrophic forgetting had occurred, the person would have forgotten all previous knowledge of their old home town and be lost in what should be a familiar city.

Neural networks generally rely on the training data being independent and identically distributed (i.i.d.). When training data is not i.i.d., catastrophic forgetting can occur as the network will prefer recently learnt knowledge. In many training scenarios,

using i.i.d. training data is easy, but there are still numerous instances where this is difficult. For example, an autonomous agent which interacts with its environment is required to learn from recent experiences. Without storing and reusing these experiences, it is extremely unlikely for a stream of experiences to be i.i.d. This thesis primarily focuses on solving catastrophic forgetting in reinforcement learning as it is also difficult to ensure training data is i.i.d in this domain.

Ratcliff (1990) was one of the first to extensively examine the catastrophic forgetting problem in neural networks. He trained small feed-forward networks to reproduce four-element orthogonal vectors. The neural networks were trained on a subset of these vectors to near perfect recall and then trained on a different subset of the vectors to a similar standard. Following this, the neural network demonstrated very poor reproduction of the initial set of vectors, thus catastrophically forgetting. Recall of the initial items was generally a blend of the initial vectors with ones that had been more recently learnt, demonstrating that the recent learning had interfered with the network's ability to reproduce those earlier vectors.

One simple but effective solution to catastrophic forgetting is to train a neural network on all information (new and old) from scratch. However, the aim in continual learning is to persistently learn overtime without guaranteeing that all previously learnt data will be available when learning new knowledge.

1.3 Motivations

This thesis aims to solve the catastrophic forgetting problem in neural networks, allowing the network to successfully learn new tasks without substantially forgetting previously learnt ones. The thesis will predominantly focus on solving this problem in the reinforcement learning domain. A further objective is to achieve this:

- **with a consistent memory size that does not expand with the number of tasks learnt -** otherwise the model will not scale well to increasing task sizes.
- **without revisiting previously learnt tasks -** because this is not possible in many domains. For example, in a real world reinforcement learning situation it is unrealistic to assume the model can always control when it revisits previously seen environments.
- **without directly storing data from the previous tasks -** because there might be reasons (e.g. privacy, space requirements and biological realism) which

do not allow data from previous tasks to be stored.

1.4 Proposed Approach and Contributions

This thesis works toward solving the catastrophic forgetting problem, so that continual learning can be attained by deep neural networks. This research begins in the image classification domain, later shifting into the more difficult reinforcement learning domain. Achieving continual learning in the reinforcement domain is a particular goal of this thesis as it is a generic learning algorithm which could establish machine intelligence by having an agent learn beneficial behaviours through interacting with the world and observing its effects, rather than another entity (e.g. a human) telling it how it should behave, as occurs in supervised learning.

Continual learning is achieved in this thesis by extending earlier work in pseudo-rehearsal (Robins, 1995). Pseudo-rehearsal is where prior knowledge is protected by continuing to rehearse it while learning new information. Instead of rehearsing real data from the previous tasks, pseudo-rehearsal suggests that this data can be usefully approximated by sampling the outputs of the network in response to random inputs. This approximated data is commonly referred to as pseudo-data or pseudo-items. While effective in shallow networks (Robins, 1995), this pseudo-rehearsal method alone does not counteract catastrophic forgetting in deep neural networks because the generated data is not representative of complex, previously learnt tasks and therefore, does not promote their retention.

The main contributions this thesis makes for solving the catastrophic forgetting problem in deep neural networks are:

- Extending pseudo-rehearsal to deep neural networks by introducing a generative network. The training of this generative network is adapted so that it also uses pseudo-rehearsal, resulting in a generator that can produce pseudo-data that is representative of all previously learnt tasks. This method, termed Pseudo-Recursal (Atkinson, McCane, Szymanski, and Robins, 2018a), prevents catastrophic forgetting to a similar standard to rehearsing real data from previously learnt tasks and is designed so that it achieves the objectives specified in Section 1.3 while sequentially learning image classification tasks.

- Extending pseudo-rehearsal and Pseudo-Recursal further so that it can still achieve the objectives specified while sequentially learning reinforcement tasks. This is

attained by incorporating a dual memory model with specific loss functions for promoting the new task to be learnt successfully while still preventing forgetting in this domain. This method, termed RePR (Atkinson, McCane, Szymanski, and Robins, 2018b), is experimentally demonstrated to prevent catastrophic forgetting while sequentially learning reinforcement tasks.

- Extending RePR from learning with Deep Q-Learning to Actor-Critic methods along with experimental results demonstrating similar success in preventing catastrophic forgetting.
- Providing experimental results demonstrating when a continual learner should retain the value function so that it can further learn familiar tasks.
- Providing a method for improving the generative network used in RePR so that it is specialised for continual learning. This is achieved by using information from the continual learner to encourage the generator to produce pseudo-data that promotes effective rehearsal (Atkinson, McCane, Szymanski, and Robins, 2019).
- Providing experimental results demonstrating RePR's shortcomings while learning an extended series of tasks.

A git repository containing code which can be used to recreate the majority of this thesis' experimental findings can be found at `https://bitbucket.org/catk1ns0n/repr_public/src/master/`.

1.5 Layout

The remainder of this thesis consists of the following five chapters:

- **Chapter 2** introduces neural networks and how they are trained for continual learning in both image classification and reinforcement learning. Then existing methods are presented for preventing catastrophic forgetting in these two learning domains. This research includes the basic pseudo-rehearsal method as well as other competing methods. These methods are then compared to theories about how continual learning is achieved in the human brain.
- **Chapter 3** specifies how pseudo-rehearsal can be adapted for image classification with deep neural networks. The importance of these improvements are experimentally shown along with results demonstrating how this method compares to other common approaches.

- **Chapter 4** outlines how this thesis' pseudo-rehearsal method can be further adapted into the reinforcement learning domain, along with results demonstrating the method solving the catastrophic forgetting problem in reinforcement learning and comparisons to other state-of-the-art approaches.

- **Chapter 5** further investigates pseudo-rehearsal's capabilities in the reinforcement domain and further extends the model to improve its performance. This chapter contains research on; how effective pseudo-rehearsal is for another popular type of deep reinforcement learning known as Actor-Critic methods; how pseudo-rehearsal can be used to continue learning a task that has been partially learnt; how the generative model can be designed specially for improving pseudo-rehearsal; and how pseudo-rehearsal performs on longer task sequences.

- **Chapter 6** concludes the thesis, giving some final remarks and suggesting future work that could be done in this field.

Chapter 2

Background

2.1 Neural Networks

2.1.1 Architecture

Artificial neural networks are a computing model loosely based upon the neuronal system in the human brain. They are powerful models capable of achieving extraordinary performance in a wide variety of challenging tasks including large scale image recognition (He, Zhang, Ren, and Sun, 2016) and strategy board games like Go (Silver, Huang, Maddison, Guez, Sifre, Van Den Driessche, Schrittwieser, Antonoglou, Panneershelvam, Lanctot, *et al.*, 2016). Neural networks are a distributed model made from artificial neurons (Rosenblatt, 1957). Each artificial neuron takes a vector of input values and returns a single output value determined by the weights, bias and activation function of the unit. Mathematically, the output of an artificial neuron is defined as:

$$o = \sigma(w \cdot x + b), \tag{2.1}$$

where x is a vector of input values, w a vector of trainable weights with the same length as x, b a single trainable bias weight and σ the activation function of the neuron. There are many activation functions that may be used for these units, one of the most common of these being the ReLU (Glorot, Bordes, and Bengio, 2011) function where $\sigma(a) = max(a, 0)$.

Separately these neurons are relatively weak. However, when organised into a single model they become very powerful. Generally they are organised into fully-connected layers, where each layer is made up of a number of artificial neurons and the input to

each neuron is the output values from all of the previous layer's neurons (or in the case of the first layer, the model's input values). All layers, besides the final output layer, is known as a hidden layer, because it is hidden between the input and output of the model. Input is fed forward through the layers of the network to the final output layer. Even a two layer architecture (i.e. with one hidden layer), is powerful enough to solve any non-linearly separable problem (Cybenko, 1989). This architecture is commonly known as an artificial neural network (or neural network for short) and a neural network with more than two layers becomes a deep neural network.

A special class of these networks which will be used extensively in this thesis is the Convolutional Neural Network (CNN). This model was developed by LeCun, Bottou, Bengio, and Haffner (1998) to resemble the visual system in biological brains, where the brain processes the visual field by detecting small simple features from which larger more complicated features are detected. These CNNs take spatial information, such as an image, as input and process it through a number of layers. At first, the layers are generally convolutional and max-pooling layers and then simple fully-connected layers are usually used to map onto the output units. A diagram demonstrating the layers in a CNN can be found in Figure 2.1.

Each neuron in a convolutional layer has trainable weights (also known as a filter) connecting it to a localised area in the previous layer (or the input). Each neuron shares the same trainable weights with a group of other neurons with the only difference being the area of the previous layer the neuron is locally connected to. This means that the groups of neurons that share weights are essentially looking for the same feature over different localised areas of the input. There are multiple groups of neurons in each layer, each group detecting a different feature and the output from each of these groups make up a feature map. The feature maps are fed to the next layer as input so that more complex features can be detected upon them. In practice, convolutional layers also use activation functions and trainable bias weights which are shared within feature maps and simply added to each of the feature map's values.

Max-pooling layers often come after blocks of convolutional layers. These layers simply downsample the feature maps outputted by the previous convolutional layer by applying a window across each of the feature maps, outputting the largest value found in each application of the window. Max-pooling layers downsample the feature maps when their stride parameter (how far the windows move after each application) is greater than 1 unit. Downsampling can similarly be achieved by a convolutional layer by using a filter stride greater than 1 unit.

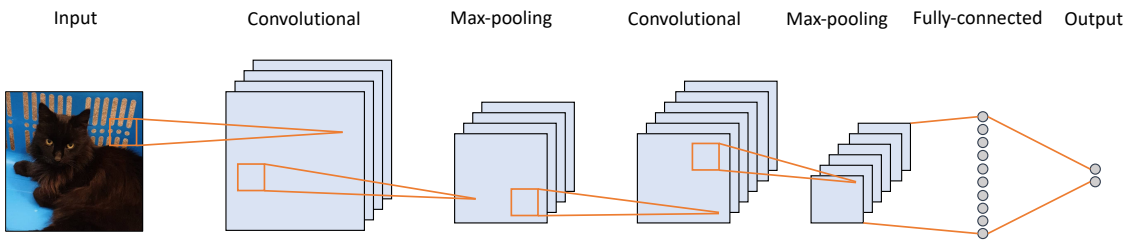

Figure 2.1: Diagram displaying the typical layout of a Convolutional Neural Network. Each square in the convolutional layers represents a feature map and the values of this feature map are calculated by applying the same filter across different spatial locations in the previous layer. Each square in the max-pooling layers represents a downsampled version of the corresponding feature map in the previous convolutional layer and this is calculated by using the maximum value in a window applied to different spatial locations in the previous layer. The fully-connected and output layers are made of a number of units which have weighted connections to every value outputted by the previous layer.

The first fully-connected layer has a number of units each connected by trainable weights to every output value in the previous layer's feature map. Later layers use the typical fully-connected structure where each unit is connected to all units in the previous layer.

2.1.2 Optimisation

The trainable weights in a neural network are initialised to random values. To solve a problem, these weights need to be changed so that the network learns to output desirable values. This is achieved by changing the weights to minimise a loss function. More specifically, back-propagation is used to calculate the gradients for each weight in the network with respect to the network's loss on the training data (Rumelhart, Hinton, and Williams, 1986). These gradients inform the network in which direction the weights should be changed to minimise the loss function and thus, they are changed very slightly in that direction. This process, termed stochastic gradient descent, is repeated until the loss function has converged at either a global or a local minimum.

Supervised Learning

In supervised learning, the goal of the neural network is to learn a training dataset which contains input examples along with each of the desired outputs for these examples. A common example of supervised learning is image classification. Image classification has a neural network model learn to categorise which class a data example belongs to. Let $\{X, Y\}$ denote a dataset of input images and target output pairings. The classification neural network is a decision function $h(x; \theta)$ and the weights of the network θ are learnt by minimising a loss function, usually cross-entropy:

$$CE(\hat{y}, y) = -\sum_{i}^{C} y_i \log \hat{y}_i, \tag{2.2}$$

where the target output y is a vector of C values, with each value representing the probability of being in its corresponding class. $\hat{y} = h(x; \theta)$ for some input image x.

There are various continual learning scenarios which cause catastrophic forgetting to occur in image classification; expanding a model by learning data from one or more new classes or expanding the model's knowledge of existing classes by learning different data within the classes. The experiments in this thesis explore how to prevent catastrophic forgetting when expanding the model by learning a number of new classes at once. The new classes are from a different task which can be either similar to or dissimilar from previously learnt tasks. This scenario will be referred to as sequential task learning and is more specifically defined as learning a sequence of datasets $D_1, D_2, ..., D_T$ of length T, where, during each learning session, the network can only directly train on the current dataset. Each dataset $D_t = \{X_t^{C_t}, Y_t^{C_t}\}$ contains only data from a set of classes C_t not included in previous datasets, this is $(C_1 \cup C_2 \cup ...C_{t-1}) \cap C_t = \emptyset$.

Reinforcement Learning

A reinforcement learning task is generally framed as a Markov Decision Process where there is an agent learning to maximise its reward by interacting with an environment. At each time step t, the agent receives the current state of the environment $s_t \in S$ and then selects an action to take from the set of all possible actions $a_t \in A(s_t)$[1]. This action is selected subject to the agent's policy, which is a mapping between the environment's states and the action the agent has learnt to take when observing the state. Based on this action and previously taken actions, the agent receives a reward r_t from the environment (either positive, negative or neutral) and is informed whether

[1]When the set of possible actions is consistent across all states this can be simplified to $a_t \in A$.

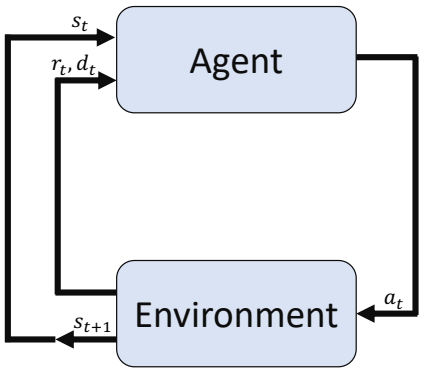

Figure 2.2: Summary of the interaction between the agent and the environment in a Markov Decision Process.

or not this action caused the environment to be terminal (end) d_t. If the environment was not terminal, it then transitions to a new state s_{t+1}. A diagram summarising the interaction between the agent and the environment in a Markov Decision Process can be found in Figure 2.2.

In this thesis, reinforcement learning is extended into a continual learning problem by requiring the agent to learn a sequence of environments $E_1, E_2, ..., E_T$ of length T, where, during each learning session, the agent's network can only directly train on the current environment. The difference between these environments can be simple, for example, flipping the rules for receiving positive and negative rewards, or the difference can be major, such as having completely different observable states and different rules for receiving rewards.

Q-learning is a reinforcement learning algorithm which maps each possible state-action pair to a Q-value which represents the expectation of the discounted reward of taking that action. This is,

$$Q : S \times A \to \mathbb{R}, \tag{2.3}$$

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}, \tag{2.4}$$

where S is the collection of possible states and A is the collection of possible actions. R_t is the discounted reward for time t, r_t is the reward received at that time, T is the

time step that the episode terminates at and γ is a discount factor where $0 \leq \gamma \leq 1$.

In Q-learning, the Q function is represented with a lookup table. However, Q-learning does not scale effectively when the number of actions or possible states is very large. Deep Q-learning (Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, Ostrovski, *et al.*, 2015) avoids this limitation by replacing the lookup table with a non-linear function approximator. This non-linear function approximator is a deep neural network and is referred to as a Deep Q-Network (DQN). The loss function used in deep Q-learning is:

$$L_{DQN} = \mathbb{E}_{(s_t,a_t,r_t,d_t,s_{t+1})\sim U(B)}\left[\left(y_t - Q(s_t, a_t; \theta_t)\right)^2\right], \tag{2.5}$$

$$y_t = \begin{cases} r_t, & \text{if } d_t \\ r_t + \gamma \max\limits_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_t^-), & \text{otherwise} \end{cases} \tag{2.6}$$

where there exist two Q functions, a deep predictor network and a deep target network with the weights θ_t and θ_t^- respectively. The weights of the predictor are continuously updated by stochastic gradient descent and the weights of the target network are infrequently updated with the values of θ_t. $(s_t, a_t, r_t, d_t, s_{t+1}) \sim U(B)$ is the state, action, reward, terminal and next state that is drawn uniformly from a large record of previous experiences, known as an experience replay. Drawing samples from this experience replay allows the network to be learning from i.i.d. data and thus, prevents it from forgetting how to respond in less recent states.

Off-policy learning is where a policy is improved with data that is generated from a different policy; in this case, the next action in the loss function is selected greedily from the target network, whereas the agent selects actions (generating the data) using an ε-greedy policy (Sutton and Barto, 2017). In deep Q-learning, an ε-greedy policy selects a random action with the probability of ε, and otherwise selects the action by passing the current state through the predictor network and selecting the action with the highest Q-value associated with it. Bootstrapping is where an update is made using an estimated value; in this case, the estimated Q-value of the resulting state (Sutton and Barto, 2017).

A reinforcement algorithm that uses function approximation, bootstrapping and off-policy learning together (known as the deadly triad (Sutton and Barto, 2017)) is in danger of instability and divergence. To counter the effects of this, deep Q-learning uses the addition of experience replay, infrequently updating the target network and clipping

error gradients between $[-1, 1]$ to achieve Q-learning in a deep network. The original DQN paper (Mnih *et al.*, 2015) demonstrated the model individually learning a large number of Atari 2600 games, many of them to a standard above human expertise. There have also been many variations of DQNs such as Double DQN (Van Hasselt, Guez, and Silver, 2016), Prioritised Experience Replay (Schaul, Quan, Antonoglou, and Silver, 2016) and the Dueling Architecture (Wang, Schaul, Hessel, van Hasselt, Lanctot, and de Freitas, 2016). However, these improvements are for learning a single task more effectively and thus, they all suffer from catastrophic forgetting.

2.2 Catastrophic Forgetting

This section will focus on existing methods for preventing catastrophic forgetting in neural networks. Many of these methods rely on keeping a copy of the neural network before it learns the new task (often referred to as the previous network), as this copy should be a reasonable example of how to retain previously learnt tasks. There are two main strategies for avoiding catastrophic forgetting. The first strategy is to restrict how the network is optimised. Generally this involves either constraining the network's weights to yield similar values compared to when the network had learnt previous tasks (i.e. methods that use weight constraints) or introducing units trained only on specific tasks (i.e. methods that use task specific weights). The second strategy is to amend the training data to be more representative of previous tasks. This thesis builds upon pseudo-rehearsal which falls into the latter category of amending the training dataset.

Lopez-Paz and Ranzato (2017) proposed that a measure called forward transfer and backward transfer should be taken into account in continual learning experiments. Backward transfer measures what effect learning the current task has on previously learnt tasks. When backward transfer is positive, performance in previous tasks is increased due to learning the new task. However, when it is negative, performance on previously learnt tasks is decreased and this decrease is otherwise known as catastrophic forgetting. Forward transfer measures what effect previously leant tasks have on learning the current task. Positive forward transfer increases performance in the current task and negative forward transfer decreases performance in the current task. Forward transfer is also important when attempting to overcome catastrophic forgetting as it is possible to retain complete knowledge of previously learnt tasks by simply not learning anything from later tasks. However, this is not an acceptable solution as a continual learner must be able to consolidate new knowledge into its current knowledge

and thus, it is important to be able to learn new tasks while also retaining previous ones.

2.2.1 Preventing Catastrophic Forgetting by Restricting the Optimisation of the Network

Knowledge is retained in a neural network across all of the network's weights so that the network's output is the contribution of all weights rather than a select few. Dropout (Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov, 2014) was proposed to increase the capability of a neural network to generalise to unseen input examples by compensating for noise in the network's activation patterns. This was achieved by multiplying intermediate layers' activations by a binary mask so that a proportion of the units' outputs were set to zero regardless of the input. This binary mask was set randomly for each input and therefore, it forced the neural network to compensate for noise, relying less heavily on specific neurons in the network providing useful information about the task. Goodfellow, Mirza, Xiao, Courville, and Bengio (2014) have used dropout to prevent catastrophic forgetting by introducing redundancy into a neural network. They found that, regardless of the task, it was always beneficial to use dropout to limit catastrophic forgetting. Goodfellow *et al.* (2014) also investigated how the choice of activation function resulted in catastrophic forgetting but did not identify a single activation function which best prevented catastrophic forgetting across multiple tasks.

Learning without Forgetting (Li and Hoiem, 2018) is another method for preventing catastrophic forgetting which utilises dropout and knowledge distillation. Knowledge distillation teaches a student network how to solve a task by training it to produce the same output patterns as a teacher network does on input examples from the task (Hinton, Vinyals, and Dean, 2015). The teacher network has already been trained on the task and thus, it is a method for passing its knowledge to another neural network. This method was originally used for compressing knowledge of a task from a large teacher network to a smaller student network.

In Learning without Forgetting, a continual learning network learns the new task while retaining knowledge from previous tasks with knowledge distillation. Before learning the new task, the continual learning network is copied and this copied network is used as the teacher. Distillation is used to teach the continual learner (i.e. the student) to produce similar output patterns as the teacher on certain input examples,

while the network also learns the new task. In sequential learning, the network should not have access to the previous dataset and thus, the input examples used for knowledge distillation are actually examples from the new task. More specifically, the continual learner is taught to produce the real target outputs for the training examples, along with additional output patterns (corresponding to previous tasks) given by the teacher. Effectively, this constrains the network's weights to learn the new task while trying to retain its computations on previous tasks. However, this is only effective when the new task's dataset is similar to the previous tasks, so that the output patterns used in knowledge distillation are meaningful. In Learning without Forgetting, there is a set of output units which learns the current task, along with a set of separate output units for each previously learnt task. This is disadvantageous because it means that the model must increase in size as the number of tasks increases[2]. However, Kim, Kim, and Lee (2018) addressed this by manipulating the model so that it only has two sets of output units, one for all previous tasks and another for the task currently being learnt.

Similar to Learning without Forgetting, Jung, Ju, Jung, and Kim (2018) have attempted to overcome catastrophic forgetting by restricting the network to have similar hidden layer activation patterns to the previous network. More specifically, the final linear layer of the neural network is frozen after learning the first task and then subsequent tasks are taught by constraining the neural network's final hidden layer activations to be similar to the previous network's activations for the new task's input examples. This constrains the network to learn the new task while reusing features that the previous network has already learnt to detect. The main disadvantage with this method is that it assumes that the new task can be effectively learnt while reusing the last linear layer in the network along with similar features detected by the last hidden layer of the previous network.

The research presented so far has been predominantly tested on image classification problems, but Elastic Weight Consolidation (EWC) (Kirkpatrick, Pascanu, Rabinowitz, Veness, Desjardins, Rusu, Milan, Quan, Ramalho, Grabska-Barwinska, *et al.*, 2017) is arguably the most popular method in this category that has been tested in supervised learning (notably image classification) and reinforcement learning. EWC is a weight constraint method that evaluates the importance of each weight in a neural

[2]This is acceptable when sequentially learning tasks whose outputs represent different concepts (e.g. new classes) across the tasks. However, when sequentially learning tasks whose outputs represent the same concepts across the tasks, a model should ideally be capable of sharing its output units across all tasks, remaining a constant size.

network for determining its output. If the weight can change the output of the network drastically, it is considered important to the task. Important weights should be changed as little as possible when learning the new task, so that previous tasks can be retained. Less important weights can be changed more significantly so that the new task can be learnt.

EWC implements its weight constraint by penalising the loss function for changing the network's weights from the values learnt in the previous task. This weight change is multiplied by the weight's importance value so that important weights are penalised more. More specifically, the final loss function is:

$$L = L_N + \frac{\lambda}{2} L_{EWC} \tag{2.7}$$

$$L_{EWC} = \sum_j F_j(\theta_j - \theta_j^*)^2, \tag{2.8}$$

where L_N is the loss for learning the new task (e.g. cross-entropy), λ is a scaling factor determining how important the constraint is, the current network's weights are θ, the previous network's weights after learning the previous task are θ^* and j iterates over each of the weights in the network. F_j is an approximation of the importance of each weight in the network to the network's output; these are the diagonal elements in a Fisher information matrix.

For a network with a linear output layer, such as a DQN, the Fisher information matrix is calculated by approximating the posterior as a Gaussian distribution using the optimal weights after learning a previous task θ_j^* as the mean and $\beta = 1$ as the standard deviation. The precise calculation of this matrix is taken from Pascanu and Bengio (2014):

$$F = \beta^2 \mathbb{E}_{x\sim\tilde{q}}[\boldsymbol{J}_y^T \boldsymbol{J}_y], \tag{2.9}$$

where an expectation is calculated by drawing inputs x from some distribution $\tilde{q}$. $\boldsymbol{J}_y$ is the Jacobian matrix $\frac{\partial y}{\partial \theta}$ for the output activation y.

When the standard EWC implementation is extended to more than two tasks, a separate penalty is added for every task. This means the current network weights are constrained to be similar to the weights after learning the first task and the weights after sequentially learning the second task and so on.

MNIST (LeCun, Cortes, and Burges, 1998) is an image classification dataset containing images of small handwritten digits from 0 to 9. Each of the images are labelled

with a class representing which digit is written. In continual learning, a variation of this dataset known as permuted-MNIST is used. This is where a model is iteratively taught to classify the images into the 10 classes. At each iteration, the pixels in the images are scrambled into a random order, consistent between all images, and the model is required to remember how to classify the task into the same 10 output units for all previously learnt permutations. EWC was found to perform very well on this task, even on 10 different permutations of MNIST, whereas dropout's performance deteriorated drastically. However, Kemker, McClure, Abitino, Hayes, and Kanan (2018) found that EWC did not perform as well when tasks were very similar or in an incremental class learning setup where new tasks mapped onto different output units than previous tasks.

EWC has also been investigated in the reinforcement learning domain. Here, the model is sequentially taught to play a number of old arcade style Atari games where an increase or decrease in the arcade game's score translates to the reward or punishment received in reinforcement learning. EWC has been tested in this domain by training a single DQN (Mnih *et al.*, 2015) to play a total of 10 games, only ever training the network on a single game at a time but allowing the network to revisit previously seen games and further train on them. In these experiments, the network used by EWC was further restricted by giving the network an additional two weights per unit which were specific to the task being learnt/tested, such that the training of the network was restricted to only optimising the weights specific to the current task along with weights shared across all tasks. Although EWC prevented some catastrophic forgetting, it still clearly suffered from it, as many games had sharp dips in performance which were only recovered by further training on the game.

EWC is a very popular method which many researchers have suggested improvements for. For example, Chaudhry, Dokania, Ajanthan, and Torr (2018) found that, at a local minimum, gradients are very small and thus, so are the Fisher values and therefore, a very large λ hyper-parameter is necessary to mitigate catastrophic forgetting. Therefore, authors suggested that EWC could be combined with Path Integral (Zenke, Poole, and Ganguli, 2017) so that the algorithm was less dependant on this hyper-parameter selection. Furthermore, Hong, Li, and Shin (2019) found that if the network learnt only images from the new task which were being classified the most incorrectly, this had the potential to reduce training time and further minimise the catastrophic forgetting problem.

One major disadvantage with EWC is that the weights after learning each of the previous tasks must be stored to calculate how much the current set of weights differ

from these values. Therefore, a better and less complex solution to this problem would be to have a separate set of weights for every single task. Furthermore, either data from previously learnt tasks needs to be stored so that the Fisher information matrix can be calculated for each of the previous tasks or a precomputed matrix needs to be stored for each previously learnt task. However, the Progress and Compress algorithm (Schwarz, Luketina, Czarnecki, Grabska-Barwinska, Teh, Pascanu, and Hadsell, 2018) suggests that EWC can be modified so that the aforementioned space requirements do not scale with each new task. This modification, known as online-EWC, stores only the most recently learnt network's weights along with a discounted sum of previously calculated Fisher information matrices. Inherently, this prefers the retention of more recently learnt tasks, encouraging the gradual forgetting found in biologic neuronal systems. Online-EWC replaces the L_{EWC} constraint in Equation 2.7 with:

$$L_{OEWC} = \sum_j F_j^*(\theta_j - \theta_{j,i-1}^*)^2, \tag{2.10}$$

where the optimal weights after learning the most recent previous task are θ_{i-1}^* and the single Fisher information matrix F^* is updated by:

$$F^* = \gamma F_{i-1}^* + F_i, \tag{2.11}$$

where γ is a discount parameter ($0 \leq \gamma \leq 1$) and i represents the index of the current task. Min-max normalisation is used on the Fisher information matrices before addition so that the importance of the weights are not affected by the differing reward scales of the games.

The Progress and Compress algorithm was also improved by introducing a dual memory system where two neural networks exist; one which focuses solely on learning the new task and the second which learns the new task while also retaining knowledge of the previous tasks. The second network is taught to perform the new task by the first network using distillation, while also being constrained to remember previous tasks through online-EWC. When the new task is being learnt by the first network, there are layer-wise connections from it to the second network which aim to encourage it to utilise features which have already been learnt in previous games.

Progress and Compress has been taught how to play Atari 2600 games similar to the previously mentioned EWC. It was taught to play 6 Atari games with a schedule that also restricted it to learning from one game at a time but allowing previously learnt games to be revisited for further learning. No task specific weights were added

to the network, but the network was only required to remember the policy function. A policy function describes only how the agent should act in a game, which is less complex to learn than the Q-values learnt in the EWC paper. Progress and Compress showed promising retention of previous tasks, suggesting it could be used to overcome some of the limitations of EWC.

Other weight constraint methods (Kaplanis, Shanahan, and Clopath, 2018; Kobayashi, 2018) have been proposed and tested in reinforcement learning. However, these methods have only been tested on relatively simple reinforcement learning tasks compared to the set of Atari 2600 games and therefore, do not demonstrate as promising results.

The previously mentioned methods have primarily focused on constraining weights to remain similar to their previous values. Solutions to catastrophic forgetting have also focused on optimising selective weights depending on the current task. This can be done by applying hard restrictions, like using task specific weights, but can also be achieved through weaker restrictions. For example, Coop, Mishtal, and Arel (2013) suggested adding a Fixed Expansion Layer between the hidden layer and the output layer of a neural network. This layer was designed so that only a select number of its units would be activated for a particular learning example. This restricted the input to the hidden layer and reduced how drastically the hidden layer weights updated in response to new information, as only a portion of the error signal was passed backward through back-propagation. When combined with ensemble learning, this method was found to counteract catastrophic forgetting in incremental image classification.

Wen and Itti (2019) present a method which uses hard restrictions, this is the addition of task specific weights, so that selective weights are only optimised for a particular task. More specifically, the network includes neurons called memory units which have weights specific to each image classification task taught. These weights are trained with a method inspired by adversarial attacks[3] so that they bias the network toward classifying the examples within that task correctly. Furthermore, authors use EWC in this method to constrain non-task specific weights to retain some knowledge of previous tasks.

Progressive Neural Networks (Rusu, Rabinowitz, Desjardins, Soyer, Kirkpatrick, Kavukcuoglu, Pascanu, and Hadsell, 2016) are a well-known method for overcoming

[3]An adversarial attack is where a small amount of noise is added to an input example (e.g. image). Although this noise is not detectable by a human, it can fool a neural network into being very certain that the example is from an incorrect class and, without the noise, the network is very certain of its correct class.

catastrophic forgetting in complex reinforcement learning tasks including Atari games. This model begins with a typical neural network which learns the first task. When a new task appears, the current network is frozen and a new network is initialised. Each layer in this new network has weights connecting its units to those in the previous layer along with units in the respective layer of all previously learnt neural networks. Essentially, this creates a structure whereby a new network can learn the new task, while also making use of processing done by the previous tasks' networks. Because optimisation is restricted to weights in the current network, the previous network's weights do not change and thus, cannot catastrophically forget the previous tasks. The main disadvantage of this network is that its memory requirements dramatically grow as further networks are required with the addition of each new task.

Although it is well-known that in infants the brain rapidly creates new neurons, research suggests that in adulthood the production of neurons in areas related to memory slows to undetectable levels (Sorrells, Paredes, Cebrian-Silla, Sandoval, Qi, Kelley, James, Mayer, Chang, Auguste, *et al.*, 2018). Therefore, human intelligence appears to be capable of avoiding catastrophic forgetting while learning new knowledge without an abundance of new neurons needing to be incorporated into its network. This further supports that machine intelligence should be capable of overcoming catastrophic forgetting without requiring these task specific neurons.

2.2.2 Preventing Catastrophic Forgetting by Amending the Training Data

Perhaps one of the simplest solutions to catastrophic forgetting is what is known as rehearsal (Ratcliff, 1990). This is where the training datasets of previously learnt tasks are either fully or partially kept so that their previous examples can be interleaved with examples from the new task. Continuing to learn previous examples forces the network to retain its previously learnt knowledge when integrating the new task. There are many variations of rehearsal methods which have been used in both image classification and reinforcement learning. One of these variations is the pseudo-rehearsal approach which is used in this thesis. This section will begin by introducing a variety of rehearsal approaches in image classification and then reinforcement learning. Following this, several pseudo-rehearsal approaches will be described in both image classification and reinforcement learning.

It is also popular to use Knowledge Distillation in rehearsal methods, although it is

used to remember the previous task rather than teach the new task. This is achieved by storing input examples from previous tasks and then passing them through the previous network to attain the desired output values. The new network is then taught to produce the desired outputs from the input examples. Typically, a variation of the cross-entropy loss function is used with Knowledge Distillation. However, Kim, Bae, Jo, and Choi (2019) suggested that it was beneficial to use the Maximal Entropy Regulariser. This loss function does not overfit as severely to the data and, because the previous model's output on the examples can be incorrect, the network should avoid over-optimising on incorrectly classified data. Authors further extend their method by excluding examples from the new task's dataset from being learnt. This begins in a random manner but is later done by excluding high certainty samples so that uncertain samples can be focused on during later training.

The major disadvantage with rehearsal methods is that they require data from the previous tasks to be stored. Zhang, Zhang, Ghosh, Li, Tasci, Heck, Zhang, and Kuo (2020) argued that this was not necessary if generic, diverse and related data could be collected for consolidation, for example, by trawling the internet. In this model, a new neural network is initialised and taught the new task. After training, distillation is used to teach the new task to another newly initialised network, while also transferring knowledge from another teacher neural network which contains knowledge of all previously learnt tasks. This is done without using any real data but rather, a large number of unlabelled examples classified by the teacher networks. This results in the student network containing knowledge of the current task along with previously learnt tasks. Surprisingly, authors found distillation with the L2 loss function to be more successful than typical cross-entropy variations.

Another way to minimise the main disadvantage of rehearsal methods is to only store a subset of the data from previously learnt tasks. For example, Lopez-Paz and Ranzato (2017) exploit the gradients from previously learnt samples while picking a subset of examples that best represents the learnt datasets. This idea has also been utilised in reinforcement learning. Isele and Cosgun (2018) explored different strategies for selecting samples to later use in rehearsal. The strategies investigated were: favouring surprise, favouring reward, matching the global training distribution and maximising the coverage of the state space. In general, catastrophic forgetting was most consistently prevented by selecting samples which matched the global training distribution, which was achieved by simply selecting samples randomly from the training data/stream.

Another popular rehearsal algorithm in reinforcement learning is PLAID (Berseth, Xie, Cernek, and Van de Panne, 2018). In PLAID, two neural networks are used; one for learning the new task and the other for remembering all previously learnt tasks. When a new task is presented, the first network is initialised with the weights from the latter network and then taught the new task. It does not matter that this network will forget previously learnt tasks, as long as it can learn the new task while preferably reusing some of the knowledge from past tasks that it was initialised with. After this, the policy learnt by the two networks is merged via distillation so that the new network contains the policy from each task seen so far. This distillation step requires data from previously learnt tasks and is therefore, preventing catastrophic forgetting with rehearsal. Rehearsal methods have also been combined with optimisation based meta-learning so that transfer is maximised and interference is minimised, reducing catastrophic forgetting (Riemer, Cases, Ajemian, Liu, Rish, Tu, and Tesauro, 2019).

Similar distillation methods have also been applied to a set of Atari games in multi-task learning. In multi-task learning, the goal is to teach a neural network multiple reinforcement learning tasks using a number of pre-trained neural networks each specialised in an individual task (Rusu, Colmenarejo, Gulcehre, Desjardins, Kirkpatrick, Pascanu, Mnih, Kavukcuoglu, and Hadsell, 2016; Parisotto, Ba, and Salakhutdinov, 2016). However, in multi-task learning the network is being taught the games simultaneously and therefore, catastrophic forgetting is not an issue.

CLEAR (Rolnick, Ahuja, Schwarz, Lillicrap, and Wayne, 2019) is a rehearsal method which could overcome catastrophic forgetting in sequential reinforcement learning where the task boundaries are not known. This algorithm was tested on the same set of Atari games used by the Progress and Compress algorithm. The method used a data store, containing an even distribution of previously seen samples. A reinforcement loss function was used so that the network learnt how to maximise its reward on new samples from the current task as well as stored samples from the current and previous tasks. To further prevent catastrophic forgetting, two more loss functions were introduced. The first loss function used KL divergence to constrain the network to produce similar policy values for a stored sample that it had historically been given by the continual learner. The second used L2 loss to constrain the network to produce similar state values for a stored sample that it had historically been given by the continual learner. These state values were the cumulative reward expected to be gained by the current policy for a given input state. The authors' CLEAR method was found to perform comparably against the more complicated Progress and Compress algorithm on the set

of Atari games.

Robins (1995) proposed pseudo-rehearsal, a method which did not store data from previous tasks[4] and instead rehearsed pseudo-items which were generated when required. Similar to examples in a training dataset, pseudo-items contain an input pattern to be passed to the network and a desired output pattern which the network should learn to return from the given input. A pseudo-item's input pattern is generated with a simple random number generator so that it represents the whole distribution of real input examples. The pseudo-item's output pattern is calculated by passing the pseudo-input pattern through the network before it begins learning the new task. A collection of these pseudo-items can then be rehearsed while learning the new task so that the network is encouraged to learn the new task without severely changing its output on other inputs.

More specifically, pseudo-rehearsal can be formularised as:

$$L = L_N + L_{PR} \tag{2.12}$$

$$L_N = \mathcal{L}(h\left(x; \theta_i\right), y)\,, \tag{2.13}$$

$$L_{PR} = \mathcal{L}\left(h(\breve{x}; \theta_i), \breve{y}\right), \tag{2.14}$$

where the final loss function L is the sum of the loss function used for learning the new task L_N and the loss function used for retaining previous tasks with pseudo-rehearsal L_{PR}. Furthermore, $\mathcal{L}$ is a loss function, such as cross-entropy, which is used to learn and retain each task. h is a neural network with weights θ_i while learning task i. x, y is the input-output pair for an item from the current task, whereas $\breve{x}$ is a pseudo-item generated by a random number generator (e.g. $U(0, 1)$) and its target output is calculated by $\breve{y} = h(\breve{x}; \theta_{i-1})$.

Pseudo-items are generated so that they cover the space of possible inputs for previous tasks. Therefore, pseudo-items represent the network's function across the whole input space. When the network is changed to accommodate a new task, these changes are made as local as possible to the input space of the new task so that it is learnt without disrupting how the network responds to the remaining areas of the input space, including areas belonging to previously learnt tasks (Robins and Frean,

[4]Early work focused on remembering/reproducing input patterns (or small sets of patterns) taught iteratively to the network, but for simplicity these will be referred to as separate tasks.

1998). In linear networks, this has been proven (under fairly general conditions) to be achieved by pseudo-items 'orthogonalising' weight changes so that updating weights for a new training item does not tend to change the outputs of the network for other inputs (Frean and Robins, 1999).

Early research around pseudo-rehearsal has proposed dual memory approaches to the problem, similar to the aforementioned Progress and Compress method. In French (1997), two neural networks exist. The first network learns examples from the new task along with learning pseudo-items produced by passing random inputs through the second network during training. After learning the new task and pseudo-items to a criteria, the weights of the network are then copied across to the second network so that the first network is available for learning future tasks. Authors also showed that this copy and paste of weights could be replaced by generating pseudo-items from the first network and using these to teach the new information to the second long-term network, a technique similarly used in Ans and Rousset (1997).

Silver, Mason, and Eljabu (2015) developed a context-sensitive Multiple Task Learning (csMTL) network which extended previous works (Fowler and Silver, 2011; Silver and Poirier, 2004; Silver and Mercer, 2002) by aiming to more efficiently retain previously learnt classification tasks through pseudo-rehearsal. The csMTL neural network is a feedforward network which shares its output with all tasks being learnt and has its input split into two parts. One part is the standard input variables from the task. The second part is a set of context variables, which are used to identify the task being learnt. When learning a new task, the model uses pseudo-rehearsal to retain knowledge from previous tasks. Pseudo-inputs are generated using a random number generator and statistics on the probability distribution of each of the previous task's input variables. Importantly, these statistics are used to improve the quality of the pseudo-items being rehearsed. Silver *et al.* (2015) improved the efficiency of the csMTL network by applying the sweep rehearsal method, originally developed by Robins (1995). Sweep rehearsal suggests that when using large batches of training data to learn a new task, it is only necessary to rehearse a very small batch of items from previous tasks to retain them. To achieve this, the small batch of previous items needs to be dynamic, such that over the course of training, the network has still rehearsed a large number of unique items from the previous tasks. Consequently, Silver *et al.* (2015) found their csMTL model could more efficiently learn various task sequences, including a synthetic dataset consisting of 20 tasks.

In Section 3.3 we demonstrate that using simple random number generators does not

result in effective pseudo-items when the input space is large and complex[5]. This is because the curse of dimensionality (Bellman, 1966) means there is a negligible probability of generating a pseudo-input example within the subspace of previously learnt tasks. Therefore, random number generators cannot realistically be used to collect enough data representative of previous tasks to achieve retention with pseudo-rehearsal. This becomes a major problem in sequential image classification and reinforcement learning because the tasks learnt often have large, complex input spaces. Improving the quality of pseudo-items is one of the problems overcome in this thesis.

Draelos, Miner, Lamb, Cox, Vineyard, Carlson, Severa, James, and Aimone (2017) overcame a similar problem while incrementally learning to reconstruct digits from the MNIST dataset using an auto-encoder. When a new class was detected, their model could expand by adding neurons into the architecture to facilitate learning the new class. They also generated pseudo-items which could be rehearsed alongside the new items to prevent forgetting. This was achieved through a process they called intrinsic replay, which randomly generates input for the auto-encoder based on prerecorded statistics about the encoder's output for previous classes. Passing this through the decoder produced an image representative of a previously learnt class.

Intrinsic replay has been used in FearNet (Kemker and Kanan, 2018) to achieve continual learning in image classification and audio classification. Their model splits learning into a dual memory system. The short-term component stores data from new tasks which are later consolidated into the long-term component alongside previously learnt information. The long-term component contains an auto-encoder which is part of the classification network but is also used to generate pseudo-items representative of previous tasks through intrinsic replay. These pseudo-items are learnt alongside new items to prevent forgetting. The long-term component is trained by minimising the classification error along with the reconstruction loss between each layer in the encoder and its corresponding layer in the decoder. When the model is queried about a particular input example, a third component is used to decide whether to use the short-term or long-term component for classification.

Mellado, Saavedra, Chabert, and Salas (2017) used pseudo-rehearsal to prevent catastrophic forgetting while a convolutional neural network learnt the first 5 digits

[5]Recently, Silver and Mahfuz (2020) showed that using pseudo-items produced by simple random number generators can also impair pseudo-rehearsal in tasks with as few as 3 input variables. However, the resulting CF shown in this small input space is substantially less than the CF this thesis finds in a large input space.

from the MNIST dataset, followed by the remaining 5 digits. A recurrent auto-encoder was used to randomly generate pseudo-items which were representative of the first 5 digits so that they could be rehearsed while learning the later digits. For each of the first 5 digits, the mean and standard deviation of each pixel location is stored. When generating a pseudo-item from a class, the class's mean and standard deviation for each pixel location is used to construct an image from a normal distribution. This image is inputted to the recurrent model, which then refines the input into an image representative of that class. A similar idea had been employed on a much simpler task by Ans and Rousset (1997), who had connections so that they could pass back and forward random inputs within their model to improve their representability. One disadvantage with Mellado *et al.* (2017)'s model is that it grows in size when a new class is learnt because a new mean and standard deviation must be stored for each pixel location in the new class's images. The variation between a class's images in MNIST is relatively small compared to real world tasks and many other popular image classification datasets and therefore, using the mean and standard deviation for each pixel location is unlikely to be effective in more complicated problems. Also, the authors' model spends time checking and discarding any pseudo-image generated which the network is less than 95% confident belongs to a learnt class.

Prior to the work presented in this thesis, pseudo-rehearsal had been scarcely applied to reinforcement learning, especially in sequential learning. Outside of sequential learning, however, a form of catastrophic forgetting usually occurs when a network learns a single reinforcement task. This is because the states the network are presented with are typically not i.i.d. as the state returned from the environment is usually more closely related to the state from the previous time step than much earlier states. Therefore, a reinforcement learning algorithm will often optimise to recently seen states, catastrophically forgetting what it has learnt about less recent states. One common solution to this problem is to store the states in a large buffer and randomly sample it. However, some researchers have overcome this problem with pseudo-rehearsal, where the neural network learns the new state while rehearsing pseudo-items generated from some simple distribution (e.g. uniform distribution) (Marochko, Johard, and Mazzara, 2017; Baddeley, 2008; Marochko, Johard, Mazzara, and Longo, 2018). However, these methods have only been applied to simple reinforcement tasks where states could be represented by simple methods of random sampling.

Given the previously stated motivations, this thesis accepts the Progress and Compress method to be the state-of-the-art solution to catastrophic forgetting, particularly

in the reinforcement domain. Therefore, this thesis will compare Progress and Compress with pseudo-rehearsal. Other methods which restrict the optimisation of the network either underperform compared to the Progress and Compress method or overcome catastrophic forgetting by adding units to the architecture, scaling its memory requirements as the number of tasks increases. Therefore, these methods are not compared in this thesis, except for EWC because of its similarity to the Progress and Compress method.

Before this thesis' work, there was no available research using pseudo-rehearsal to solve complex reinforcement learning problems to compare to. Instead, there was only research amending the dataset with real data from previously learnt tasks for rehearsal. While these methods will presumably outperform pseudo-rehearsal, they require the storage of real data and therefore, this thesis will not compare pseudo-rehearsal to a range of such methods. Instead, this thesis will only compare pseudo-rehearsal to the generic rehearsal strategy to provide a baseline demonstrating how well pseudo-rehearsal could perform if its generator could produce pseudo-items which are indistinguishable from real data.

2.2.3 Synaptic Stability vs. Synaptic Plasticity

In real neuronal circuits it is unknown whether memory is retained through synaptic stability or synaptic plasticity (Abraham and Robins, 2005; Gallistel and Matzel, 2013; Abraham, Jones, and Glanzman, 2019). The synaptic stability hypothesis states that memory is encoded by the weights between units and therefore, these weights should be fixed to retain knowledge. This implies that it is also important to retain the same activation patterns being passed throughout the whole neural network to its output layer. The synaptic plasticity hypothesis states that it does not matter how the memory is encoded, as long as the correct output pattern is still produced after changing the weights between units. Therefore, the synaptic plasticity hypothesis assumes that activation patterns between input and output units do not have to remain similar to retain knowledge. Methods restricting the optimisation of the network (including weight constraint methods) align with the synaptic stability hypothesis, whereas methods amending the training dataset with more representative samples (including pseudo-rehearsal methods) align with the synaptic plasticity hypothesis.

For example, EWC pressures the important weights in a neural network to remain similar. This aligns with the synaptic stability hypothesis as this pressure forces connections in the network to remain the same so that they pass similar activation patterns

through the layers, eventually reaching the output layer. In contrast, pseudo-rehearsal aligns with the synaptic plasticity hypothesis because it only constrains the output of the neural network to remain similar for inputs representative of the previous tasks. This results in the network having the freedom to change the network's weights and thus, activations of intermediate layers, to whatever best encodes the new information alongside previous information.

Methods that align with the synaptic stability hypothesis are very rigid and can be thought of as splitting the network into separate pathways, where different subsets of weights are important for performing different tasks. Although some sharing may exist, there is arguably little difference with training separate networks on each task. The major advantage of methods aligning with the synaptic plasticity hypothesis is that they give the network the flexibility to restructure its weights and compress previous representations to make room for new ones. This is significant because neural networks are inherently a distributed model and therefore, knowledge is, to a degree, encoded across all the network's weights and so a drastically different encoding needs to be found to truly incorporate new knowledge without forgetting. Effective sharing of weights could also importantly lead to more generalisable features, improving the network's ability to learn new tasks quickly and to a higher standard.

In other works (Silver *et al.*, 2015; Silver, Yang, and Li, 2013; Fowler and Silver, 2011), the idea of synaptic stability vs. plasticity has been similarly referred to as functional stability and representational plasticity. Functional stability describes one of the goals of a model in continual learning as having its function remain stable for previously learnt tasks. Representational plasticity describes the other goal as allowing its internal weights (or representation) to change so that new tasks can also be learnt. This definition stresses the importance of a continual learner retaining knowledge, while also being able to change its internal representation to learn new knowledge.

Chapter 3

Pseudo-Rehearsal in Deep Neural Networks for Continual Image Classification

3.1 Extending Pseudo-Rehearsal to Deep Neural Networks

This chapter extends pseudo-rehearsal to deep neural networks to solve catastrophic forgetting in image classification. As previously stated, standard pseudo-rehearsal cannot prevent catastrophic forgetting in complex tasks because its randomly generated pseudo-items are not representative of previously learnt tasks. Therefore, this chapter proposes Pseudo-Recursal which overcomes this limitation by introducing a generative network which is trained to produce images which are representative of all previously learnt tasks. This procedure recursively applies pseudo-rehearsal to both the classifier network and the generator, so that the model's memory requirements do not scale with each new task, and thus, the process is termed Pseudo-Recursal

Firstly, let's formalise the loss functions used in Pseudo-Recursal. Let $x \in \mathbb{R}^d$ and a neural network of a chosen architecture with its function given as $h : \mathbb{R}^d \mapsto \mathbb{R}^k$ and weights θ_i, while learning task i. Let y be a one-hot encoded vector indicating the assignment of one of the task's classes from k possible classes, with zero elements for all other classes.

Given the set of weights θ_{i-1}, which minimise some loss function $\mathcal{L}\left(h(x_{i-1}, \theta_{i-1}), y_{i-1}\right)$ for the previously learnt task $i-1$, we next want to train the

network on task i and find the optimal values for the weights θ_i that minimise the new task's loss $\mathcal{L}\left(h(x_i;\theta_i), y_i\right)$ while still performing well on the previous task, that is $h(x_{i-1};\theta_i) \approx h(x_{i-1};\theta_{i-1})$. Now, if we were to rehearse all previous tasks while learning task i we would need to minimise the following:

$$L = \sum_{j=1}^{i} \mathcal{L}\left(h(x_j;\theta_i), y_j\right) \tag{3.1}$$

As mentioned in Section 2.2.2, this is problematic because it requires data from all previous tasks.

In Pseudo-Recursal, the same can be achieved by minimising:

$$L = L_N + L_{PRec} \tag{3.2}$$

$$L_{PRec} = \mathcal{L}\left(h(\widetilde{x};\theta_i), \widetilde{y}\right), \tag{3.3}$$

where L_N is the network's loss while learning the current task, as specified in Equation 2.13. $\widetilde{x}$ is a randomly generated pseudo-input pattern representative of a previously learnt task (i.e. task 1 to $i-1$) and $\widetilde{y} = h(\widetilde{x};\theta_{i-1})$ is the previous network's output on this pattern[1], before learning the current task i. These randomly generated pseudo-items are supplied by a generative model. The generative model has been trained on the same sequence of tasks as the classifier, such that it can generate items which are representative of all previously learnt tasks. Details of the generative model and its training procedure will be outlined in the following subsections. The actual loss function $\mathcal{L}$ used in this chapter's experiments is cross-entropy (CE) from Equation 2.2.

3.1.1 Generative Adversarial Network

A Generative Adversarial Network (GAN) (Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville, and Bengio, 2014) is a neural network model which uses unsupervised learning to generate random images which are representative of the input dataset. This is achieved by creating two network models; a discriminative model and a generative model. The goal of the discriminative model is to identify whether an input image is a real image or a generated/fake image, whereas the goal of the

[1]In practice this output pattern was one-hot encoded because it was assumed the generator successfully produces images which predominantly represent a single class.

generative model is to create images which fool the discriminator. This results in the generator learning to create images that represent the training images.

A Variational Auto-Encoder (Kingma and Welling, 2014) was also considered as the generative model in Pseudo-Recursal. However, a Variational Auto-Encoder learns to generate images with a reconstruction loss function that struggles to produce fine details. Pilot tests confirmed that GANs generated images with superior detail compared to the Variational Auto-Encoder, particularly on datasets, like CIFAR10, that have high variation between images. Therefore, a GAN was used throughout this thesis as the generator, but could easily be replaced by other unsupervised generative models.

The GAN used in this section is based upon the Deep Convolutional Generative Adversarial Network (DCGAN) (Radford, Metz, and Chintala, 2016) which improves the training stability of GANs through various architectural changes. The loss functions for the discriminator ($L_{DiscPRec}$) and generator ($L_{GenPRec}$) are defined as:

$$L_{DiscPRec} = CE(D(\widetilde{x};\phi), 0) + CE(D(x;\phi), 1), \tag{3.4}$$

$$L_{GenPRec} = CE(D(\widetilde{x};\phi), 1), \tag{3.5}$$

where x is an input item from the training data. $\widetilde{x}$ is an item produced by the current generative model ($\widetilde{x} = G(z;\varphi)$). z is an array of latent variables, where $z = U(-1, 1)$. D and G are the discriminator and generator networks with the weights ϕ and φ respectively. The final layer of the discriminator uses a sigmoid activation function so that a probability is returned relating to how certain the network is that the input item is real. During a training iteration, the discriminator's weights are updated once and then the generator's weights are updated twice using their corresponding loss functions. A diagram summarising the layout of a GAN can be found in Figure 3.1.

3.1.2 Overcoming Catastrophic Forgetting in a Generative Adversarial Network

In the first stage of training, only data from the first task is available. This means that there are no previous tasks for the classification network to retain and therefore, it can simply be trained on the first task's data without any pseudo-items, this is using only the new task's loss L_N from Equation 2.13. While data from the first task is available, it is also important for the GAN to learn to generate images representative of the first task. This is done by simply training the GAN with images from the task so that it can later be used to generate pseudo-images which are representative of that task, without

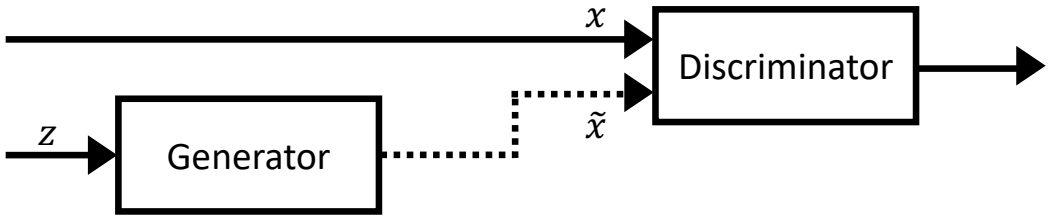

Figure 3.1: Diagram displaying the typical layout and information flow in a GAN. Solid lines represent the information flow of real data and dashed lines represent the information flow of generated data.

storing the actual dataset. When the first task is no longer available, but the second task is, the classification network can be trained on images from the second task along with pseudo-images generated by the GAN (which are representative of the first task), using the loss defined in Equation 3.2.

The main complication arises when a third task is introduced. To retain knowledge of both the first and second tasks, pseudo-items must now be representative of both tasks. A simple solution is to train a second GAN to generate images from the second task. This is effective, but requires a new allocation of memory for each task. A more elegant solution is to do pseudo-rehearsal on the GAN as well. This allows the GAN to produce pseudo-items from both the first and second tasks, without requiring extra memory per task. Pseudo-rehearsal on the GAN model can easily be achieved by generating pseudo-images from the GAN before training it on the new task and mixing them with the new task's images. All of these images can then be taught to a randomly initialised GAN, so that it learns to generate images representative of the new task and all previously learnt tasks. This procedure can be repeated recursively, every time a new task is presented. More specifically, the images used to train the GAN are defined by:

$$x = \begin{cases} x_i, & \text{if } r < \frac{1}{T} \\ \widetilde{x}_j, & \text{otherwise} \end{cases} \tag{3.6}$$

where r is a random number uniformly drawn from $[0, 1)$ and x_i is a randomly selected item from the dataset of the new task i. T is the number of tasks learnt (including the

new task) and $\widetilde{x}_j$ is a randomly generated item from the GAN before learning the new task. This item could represent an image from task 1 to $i-1$.

Typically, pseudo-rehearsal methods are used on a network initialised with the weights it has learnt to solve previous tasks. This is beneficial because the network then only needs to learn how to map the new inputs to their corresponding outputs, while retaining mappings for previously learnt inputs. However, when using pseudo-rehearsal on a GAN, its generator is learning to map the same inputs it has previously been taught to a different/extended set of output images. Therefore, the mapping of previously learnt inputs must be changed and thus, it is not beneficial to initialise the GAN with its previous weights[2].

In summary, continuous learning is achieved by applying pseudo-rehearsal to both a classification model and a GAN model. This allows the classifier to overcome the catastrophic forgetting problem without requiring extra memory when a new task is presented. The connectivity between components within the Pseudo-Recursal model while training the classifier and the GAN is outlined in Figure 3.2 and Figure 3.3 respectively. Pseudo-code demonstrating the basic training procedure for Pseudo-Recursal can also be found in Appendix A as Algorithm 1.

Pseudo-Recursal was developed at approximately the same time as Deep Generative Replay (Shin, Lee, Kim, and Kim, 2017). However, Deep Generative Replay is an analogous method, published first, which also uses a GAN to produce pseudo-items for rehearsal. Although these two methods are comparable, this thesis extends Shin *et al.* (2017)'s research by investigating the method's ability to learn dissimilar tasks without forgetting, as well as conducting experimental comparisons against methods predominantly used to overcome catastrophic forgetting in the reinforcement learning domain - namely, EWC and Progress and Compress.

3.1.3 Biological Similarities

Psychological research suggests that mammal brains could be preventing catastrophic forgetting by consolidating memories through a method analogous to pseudo-rehearsal. Two important areas of the brain involved in memory consolidation are the hippocampus and the cortex (McClelland, McNaughton, and O'Reilly, 1995). The hippocampus is responsible for short-term knowledge, whereas the cortex is responsible for long-term knowledge. Research indicates that the hippocampus and sleep are both important

[2]This was confirmed by preliminary tests.

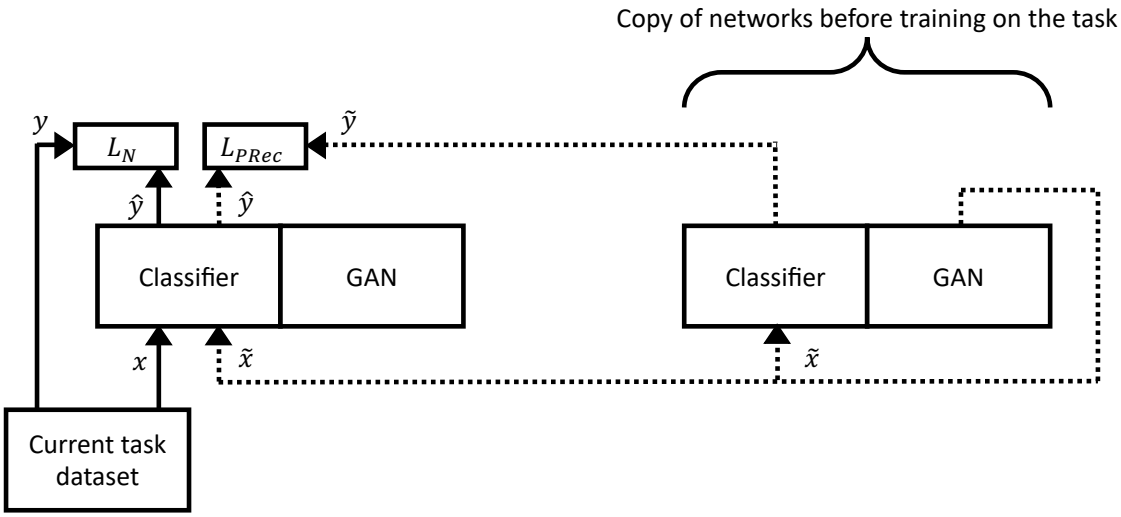

Figure 3.2: Summary of the connectivity between Pseudo-Recursal's components while training the classifier. Solid lines represent the information flow of real data (for learning the new task) and dashed lines represent the information flow of generated data (for retaining previous tasks with pseudo-rehearsal). Mathematical notation relates to the previously specified loss functions L_N and L_{PRec}.

components for retaining previously learnt information (Gais, Albouy, Boly, Dang-Vu, Darsaud, Desseilles, Rauchs, Schabus, Sterpenich, Vandewalle, *et al.*, 2007). While sleeping, the hippocampus can be seen to replay activation patterns that occurred over the day (Louie and Wilson, 2001). This demonstrates that the brain can replay previously learnt experiences similar to the way in which Pseudo-Recursal uses a GAN to generate items representative of previous experiences. Due to the similarity between artificial neural networks and the biological brain, the above suggests that pseudo-rehearsal methods could be an effective solution to the catastrophic forgetting problem.

Unfortunately, there are still key differences between the brain's memory consolidation process and Pseudo-Recursal, some of which are addressed by the RePR model introduced in the next chapter. For example, the memory consolidation process involves transferring knowledge from a short-term system to a long-term system. However, Pseudo-Recursal does not contain a system which is dedicated to short-term knowledge and thus, new information is incorporated directly into its "long-term" network.

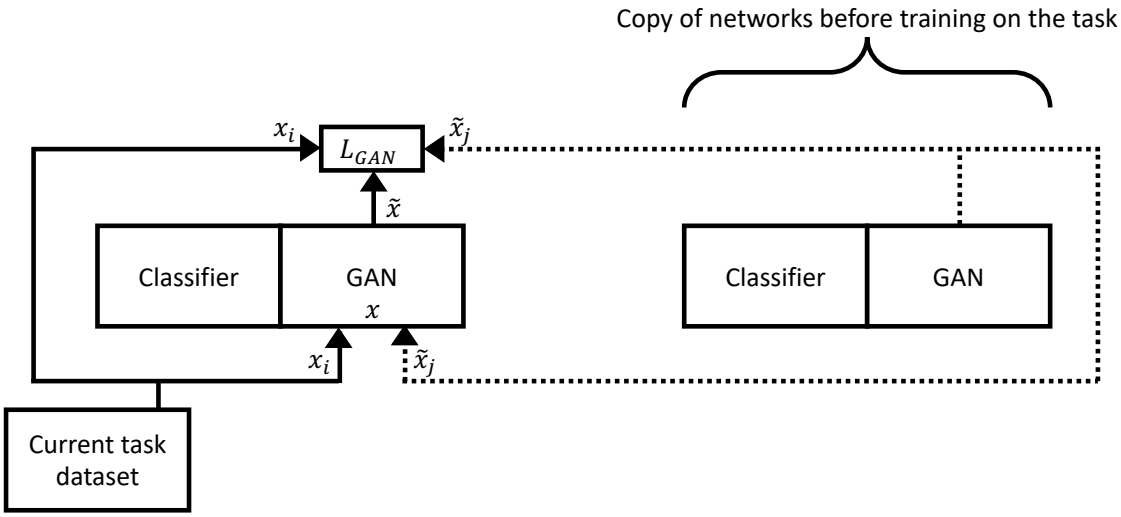

Figure 3.3: Summary of the connectivity between Pseudo-Recursal's components while training the GAN. Solid lines represent the information flow of real data (for learning the new task) and dashed lines represent the information flow of generated data (for retaining previous tasks with pseudo-rehearsal). Mathematical notation relates to the previously specified loss functions $L_{DiscPRec}$ and $L_{GenPRec}$ (i.e. L_{GAN}).

This is resolved in RePR by introducing a short-term system which is trained solely on the current task. Knowledge is later transferred from the short-term system to the long-term system through distillation. This distillation process incorporates the knowledge into the long-term system without copying it directly through a biologically implausible mechanism.

3.2 Methodology

The remainder of this chapter experimentally investigates the effectiveness of Pseudo-Recursal as a solution to the catastrophic forgetting problem. The first experiment aims to convey the shortcomings of standard pseudo-rehearsal methods with a deep neural network on complex tasks and demonstrate how a generative network can overcome these shortcomings. The second experiment compares Pseudo-Recursal to some baseline methods, these being a simple rehearsal strategy and where no effort to pre-

vent catastrophic forgetting is made. The final experiment compares Pseudo-Recursal to EWC, online-EWC and rehearsal (where rehearsal has a limited number of previous input examples stored, equal to the memory size of the generative model used in Pseudo-Recursal).

3.2.1 Datasets

Throughout this chapter's experiments, a classifier model is sequentially trained on the CIFAR-10, SVHN and MNIST datasets[3]. These datasets have been chosen because they all comprise similar sized images, the same number of classes and a range of similarities and differences between the datasets' tasks. CIFAR-10 contains images of animals and types of transport which are dissimilar to images found in SVHN and MNIST which both contain images of the digits 0-9. The SVHN dataset contains coloured images of numbers from houses as seen from the street, whereas the MNIST dataset contains greyscale images of handwritten digits. MNIST images are converted to colour (RGB) and zero-padded so that they are the same size as the other datasets' images (32×32). All datasets are divided up so that there are 37,500 training, 12,500 validation and 10,000 testing items.

For the classification network, all the tasks' and pseudo-datasets' validation and test images are center cropped to 24×24 and then standardised. For the training images, distortions are applied every epoch by randomly cropping the 32×32 images down to 24×24, flipping images left or right (only for CIFAR-10), adjusting brightness between -63 and 63, adjusting contrast between 0.2 and 1.8 and then standardising the images.

3.2.2 Network Architectures

The classification network used is based on Krizhevsky, Sutskever, and Hinton (2012) and its architecture can be found in Table 3.1. Each of the 3 task's datasets contains 10 classes and therefore, the classification network has a total of 30 output units. The datasets are one-hot encoded so that only 1 of the 30 output units is trained to be active for any given input example. The GAN used in this thesis is based on DCGAN (Radford *et al.*, 2016). The only differences are the use of pseudo-rehearsal with a pseudo-dataset size of 50,000 items and a mini-batch discrimination layer (Salimans, Goodfellow, Zaremba, Cheung, Radford, and Chen, 2016). The mini-batch discrimi-

[3]Other variations of this order were also tested and similar results were found.

Table 3.1: Classification network architecture for Pseudo-Recursal, where CONV is a convolutional layer, MAXPOOL is a max-pooling layer and FC is a fully connected layer.

Classifier				
Input: $24 \times 24 \times 3$				
layer	# units/filters	filter/window shape	filter/window stride	activation
CONV	128	3×3	1×1	ReLU
CONV	128	3×3	1×1	ReLU
MAXPOOL		3×3	2×2	
CONV	256	3×3	1×1	ReLU
CONV	256	3×3	1×1	ReLU
MAXPOOL		3×3	2×2	
FC	512			ReLU
FC	384			ReLU
FC	30			Softmax

nation layer provides the discriminator with information about the whole mini-batch of real or fake samples to further inform its decision. This reduces the training time needed for the generator to produce visually appealing images and helps stop the network from converging at a point where it only outputs the same image. The GAN's discriminator and generator architecture can be found in Table 3.2 and Table 3.3 respectively.

3.2.3 Training and Evaluation

The classifier is trained using the hyper-parameters specified in Table 3.4. When the first task is being trained, all of the mini-batch's training examples come from the task's dataset. However, for later tasks, half of the examples come from the current task's dataset and the remaining are from the pseudo-dataset.

The validation error is recorded after each epoch on both the current task's dataset and the pseudo-dataset (if one exists). After training is completed, the network's weights at the epoch with the lowest validation loss are reloaded into the network and it is evaluated on real test items from the current task and all previously learnt tasks.

Table 3.2: The GAN's discriminator network architecture for Pseudo-Recursal, where CONV is a convolutional layer, mBATCH is a mini-batch discrimination layer and FC is a fully connected layer. Layers which use batch normalisation (Ioffe and Szegedy, 2015) before the activation function are marked with an 'x'.

Discriminator					
Input: $32 \times 32 \times 3$					
layer	# units/filters	filter shape	filter stride	batch-norm	activation
CONV	64	5×5	2×2		Leaky ReLU
CONV	128	5×5	2×2	x	Leaky ReLU
CONV	256	5×5	2×2	x	Leaky ReLU
CONV	512	5×5	2×2	x	Leaky ReLU
mBATCH					
FC	1				Sigmoid

The GAN is trained using the hyper-parameters specified in Table 3.5. When the first task is being trained, all of the mini-batch's training examples come from the task's training and validation portion of its dataset. However, for later tasks, examples are taken from both the current task's dataset and the pseudo-dataset (using Equation 3.6) so that each task is fairly represented. Images passed to the discriminator are rescaled between -1 and 1 by applying $f(x) = 2(\frac{x}{255} - 0.5)$ to each raw pixel value so that they are in the same output space as the Tanh activation function used by the generator. The GAN is trained for 25 epochs and then the final generator weights are used for generating pseudo-items. Pseudo-images are rescaled back to the pixel space before being used for pseudo-rehearsal.

3.2.4 Experimental Conditions

Each experimental condition underwent 3 trials and results were averaged. The main experimental conditions are as follows:

- *std*: Learns the datasets sequentially, without using methods to prevent catastrophic forgetting. This is the lower bound on performance.
- *reh*: Learns the datasets sequentially, while still rehearsing all of the real items

Table 3.3: The GAN's generator network architecture for Pseudo-Recursal, where DECONV is a deconvolutional layer and FC is a fully connected layer. Layers which use batch normalisation (Ioffe and Szegedy, 2015) before the activation function are marked with an 'x'.

Generator					
Input: 100 latent variables					
layer	# units/filters	filter shape	filter stride	batch-norm	activation
FC	$512 \times 2 \times 2$			x	ReLU
DECONV	256	5×5	2×2	x	ReLU
DECONV	128	5×5	2×2	x	ReLU
DECONV	64	5×5	2×2	x	ReLU
DECONV	3	5×5	2×2		Tanh

from previously learnt datasets. This is the upper bound on performance.

- *pseudo-rec*: Learns the datasets sequentially with the Pseudo-Recursal method, rehearsing pseudo-items representative of the previously learnt datasets.

- *ewc*: Learns the datasets sequentially, while retaining past knowledge with EWC and using task specific weights[4]. EWC uses a λ parameter which is set to 81,000 (rounded to the nearest thousand) after doing a random search between $[0, 100000)$ with 10 trials.

- *online-ewc*: Learns the datasets sequentially, while retaining past knowledge with online-EWC. Schwarz *et al.* (2018) found that the Progress and Compress method performed competitively against its online-EWC component alone and therefore, only the online-EWC component is used in this condition so that the network's architecture is kept consistent with the other conditions in this section. Online-EWC uses a λ parameter which is set to 40,000 (rounded to the nearest thousand) and a γ parameter which is set to 0.99 after doing a random search for λ between $[0, 100000)$ with 10 trials for each of the γ values $[0.9, 0.95, 0.99]$.

[4]The network is correctly told which task it is classifying so that the correct task specific weights are always applied. This gives EWC the best possible chance at outperforming Pseudo-Recursal.

Table 3.4: Hyper-parameters for Pseudo-Recursal's classification model.

Parameter	Value	Description
initial learning rate	1×10^{-3}	Learning rate used when the network is training only on the first task's dataset.
later learning rate	1×10^{-4}	Learning rate used when the network is being trained on any later task.
mini-batch size	512	The number of items trained from during each mini-batch.
patience	10	Training is stopped when the network has not improved in its validation error for this number of epochs.
β_1	0.9	First moment decay rate for the Adam optimiser.
β_2	0.999	Second moment decay rate for the Adam optimiser.
ϵ	1×10^{-8}	Epsilon value for the Adam optimiser.
pseudo train size	37,500	Number of pseudo-items in the training portion of the pseudo-dataset.
pseudo valid size	12,500	Number of pseudo-items in the validation portion of the pseudo-dataset.

- *ewc-c*10: In some sequential classification problems it makes sense for the output units learning the new task to be shared with the output units used to learn previous tasks (i.e. when the classes they represent are the same). However, in this sequential classification problem it does not make sense, as if the output units were shared between tasks, a single output unit would represent multiple classes (e.g. a digit and an animal). EWC will likely be less effective when the output units are not shared because the output units for previous tasks will be trained to never activate on the current data, potentially resulting in forgetting of the previous tasks. Therefore, this condition investigates whether EWC is more effective when the same output units are shared between tasks and as such this condition has only 10 shared output units. EWC's λ parameter is set to 81,000.

- *online-ewc-c*10: This condition investigates whether online-EWC is more effec-

Table 3.5: Hyper-parameters for Pseudo-Recursal's GAN model.

Parameter	Value	Description
mini-batch size	100	The number of items trained from during each mini-batch.
learning rate	2×10^{-4}	Learning rate for the Adam optimiser.
β_1	0.5	First moment decay rate for the Adam optimiser.
β_2	0.999	Second moment decay rate for the Adam optimiser.
ϵ	1×10^{-8}	Epsilon value for the Adam optimiser.
decay_{bn}	0.9	Decay value for batch normalisation.
ϵ_{bn}	1×10^{-5}	Epsilon value for batch normalisation.
leak	0.2	Leakage value for Leaky ReLU.
mbatch nkernels	100	Number of kernels used in the mini-batch discrimination layer.
mbatch kernel dim	5	Size of the kernels used in the mini-batch discrimination layer.

tive when the same 10 output units are shared between tasks. Online-EWC's λ parameter is set to 40,000 and the γ parameter to 0.99.

- *reh-limit*: The generator in Pseudo-Recursal is capable of generating a wide variety of images (pseudo-items) without having a very large memory footprint. Therefore, it is likely that using the generator to produce pseudo-items for rehearsal is more effective than using the same sized allocation of memory to store a limited subset of items from previous tasks for rehearsal. This condition tests this hypothesis by rehearsing a subset of real items limited by the memory footprint of the generator. The number of free parameters/weights in the generative model is approximately 4.5m and thus, 1,500 images[5] (and their true labels) were randomly selected to be stored from past tasks. This condition learns the datasets sequentially, while still rehearsing the stored items. The images are split between the training and validation sets in the same 3:1 ratio as all other datasets and distortions are also applied to the training images at each epoch.

[5]These images are stored as 32×32. This is because, each epoch, training images are randomly cropped down to a different 24×24 image.

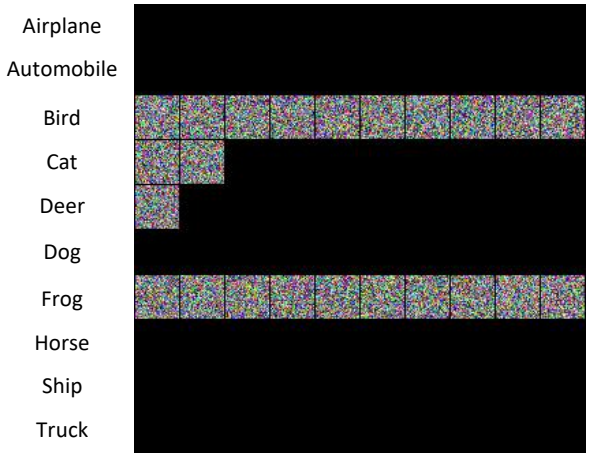

Figure 3.4: Pseudo-images generated by a uniform distribution $[0, 255]$. These images are labelled by the classification network. Images are black when no instances of that class occurred after 2,048 randomly generated images.

3.3 Results and Discussion

The first experiment aims to demonstrate the shortcomings of using the standard pseudo-rehearsal method (Robins, 1995) in deep learning. Typically, this method will use a simple random distribution to generate pseudo-items for rehearsal. However, in deep learning, the problems are much harder and thus, pseudo-vectors generated purely at random are not likely to be good representations of the training data. This is particularly obvious for images because generating pseudo-images with a uniform distribution produces static images which do not represent natural images (see Figure 3.4). Furthermore, these static images poorly represent the distribution of classes. For example, after 2,048 randomly generated images, the network believed almost all static images were either birds or frogs and when these static images are used in pseudo-rehearsal, the network retains little knowledge of its previously learnt tasks (see Figure 3.5).

To confirm that a GAN can be used to generate pseudo-images which look similar to real images, the GAN was applied to the CIFAR-10 dataset. Figure 3.6 (B) illustrates that the generated images look similar to real CIFAR-10 images from a distance, although differences emerge on close inspection. Nevertheless, the generated images still contain class specific features which the network can learn to retain.

To confirm that pseudo-rehearsal can be used on the GAN to still generate pseudo-

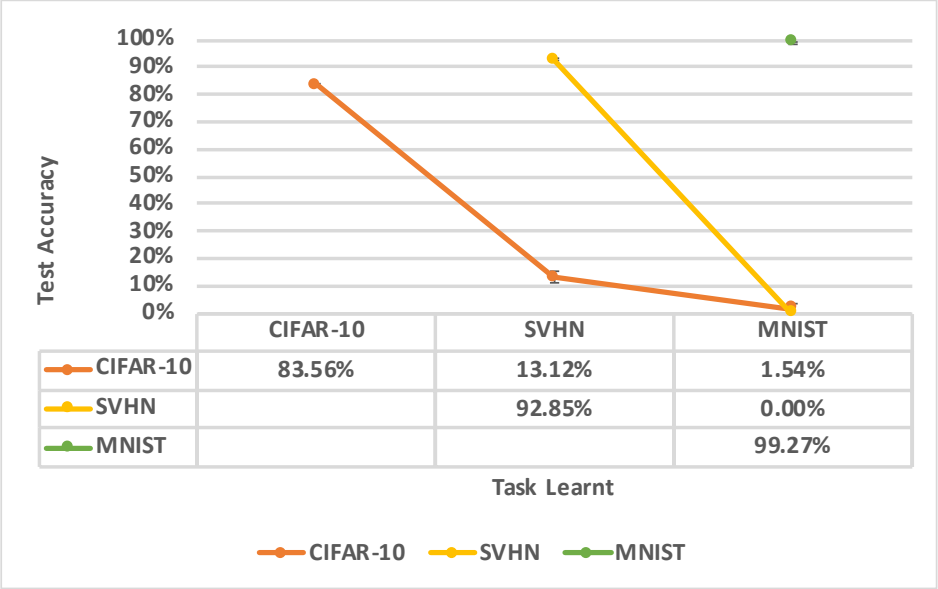

Figure 3.5: Average accuracy of a classification network when using pseudo-rehearsal with pseudo-images generated by a uniform distribution $[0, 255]$. The x-axis represents the task that has just been learnt and the lines represent the network's test accuracy on the various tasks trained so far. Error bars represent the standard deviation of each data point across the 3 trials. Non-visible error bars have smaller standard deviations than their data point.

images that look similar to real images, the GAN is trained on CIFAR-10 and then SVHN while rehearsing generated images that represent CIFAR-10 (i.e the training method used in Pseudo-Recursal). Figure 3.6 and Figure 3.7 illustrates that using pseudo-rehearsal on the GAN causes it to generate images that represent the recent task (SVHN) and the previously trained task/s (CIFAR-10). Furthermore, in the case of SVHN, the pseudo-images do not appear to be noticeably different from real SVHN images. Images generated by the GAN after learning MNIST are not shown because these images are never used in pseudo-rehearsal (as there is no fourth task to learn).

Although the classification network tested is not the state of the art network for any of these datasets, it can still be trained to very respectable accuracy on all of the tasks (e.g. over 83% on CIFAR-10) without using any special tricks. The results of all the experimental conditions are displayed in Figure 3.8. The *std* condition clearly shows catastrophic forgetting because once a new task is learnt, the network does not

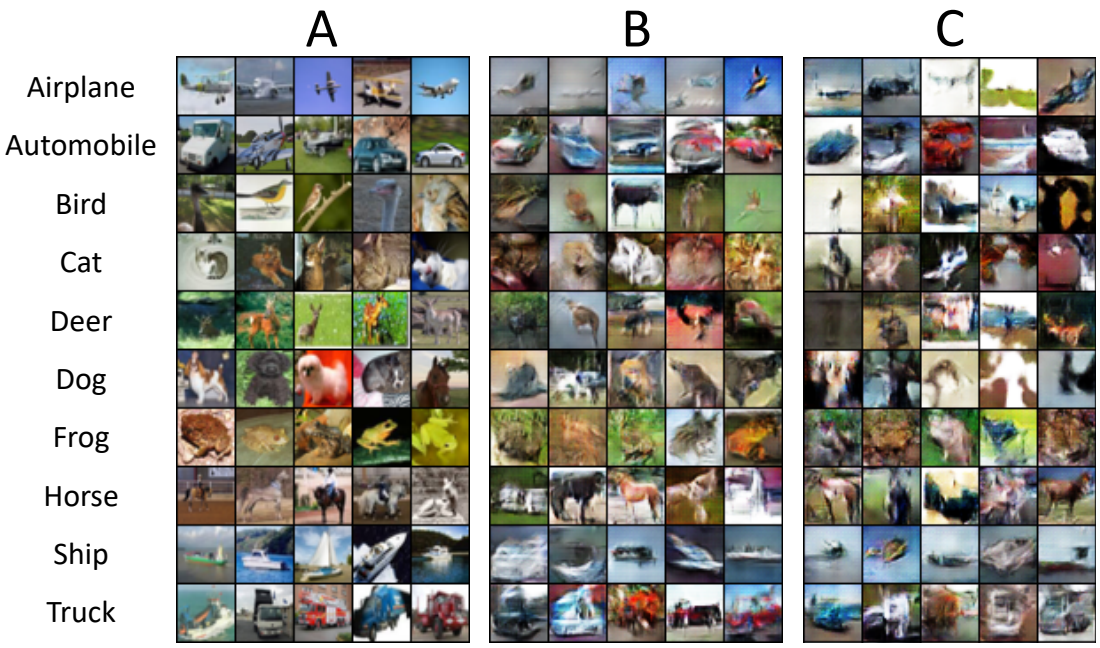

Figure 3.6: **A**: Real CIFAR-10 images. **B**: Pseudo-images generated by a GAN trained on CIFAR-10 images. **C**: Pseudo-images generated by a GAN trained on images from CIFAR-10, followed by images from SVHN along with pseudo-images representing CIFAR-10. These images are labelled by the classification network. The GAN from **C** generates images representing both SVHN and CIFAR-10, but only images representing (this is labelled by the classifier as) CIFAR-10 are included in this figure.

correctly classify any of the previous tasks' images. The fact that the accuracy drops straight to 0% on previous tasks seems dramatic. However, it is very logical for a classification network that trains using cross-entropy because when training on a new task, the previous tasks' images do not appear at all and thus, the output neurons representing those classes quickly learn that they should never activate.

As expected, the *reh* condition does not demonstrate catastrophic forgetting, as the final task accuracies increased slightly from their initial values. This condition demonstrates that the network has the capacity to learn all three tasks to a high accuracy without needing any additional units.

The *pseudo-rec* condition also overcomes the catastrophic forgetting problem as

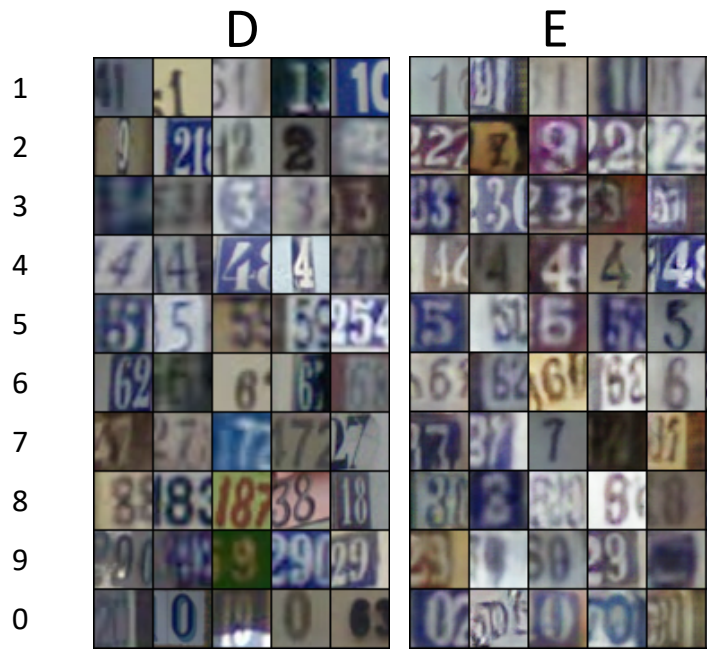

Figure 3.7: **D**: Real SVHN images. **E**: Pseudo-images generated by a GAN trained on images from CIFAR-10, followed by images from SVHN along with pseudo-images representing CIFAR-10. These images are labelled by the classification network. The GAN from **E** generates images representing both SVHN and CIFAR-10, but only images representing (this is labelled by the classifier as) SVHN are included in this figure.

it too does not experience a dramatic drop in the previous tasks' accuracy when it learns a new task. In fact, it loses no more than 1.32% accuracy each time a new task is presented and loses only 1.67% of CIFAR-10 test accuracy after learning both the other tasks. These differences in accuracy are absolute differences, which will remain consistent throughout this section. For SVHN, Pseudo-Recursal resulted in a 0.24% increase in accuracy after learning MNIST. This conveys that the network has the capability to retain almost all knowledge of previous tasks without needing to store previous data, but rather by generating approximations of it as required.

The *ewc* condition is barely resistant to the catastrophic forgetting problem, managing to correctly classify 7.11% and 9.82% of the CIFAR-10 and SVHN datasets after all three tasks have been learnt. The *online-ewc* condition does not use task specific

Figure 3.8: Average accuracy of the classification network for the *std*, *reh*, *pseudo-rec*, *ewc*, *online-ewc*, *ewc-c10*, *online-ewc-c10* and *reh-limit* conditions. The x-axis represents the task that has just been learnt and the lines represent the network's test accuracy on the various tasks trained so far. Error bars represent the standard deviation of each data point across the 3 trials. Non-visible error bars have smaller standard deviations than their data point.

weights and thus, it demonstrates as severe catastrophic forgetting as the *std* condition. The EWC variants' poor results are likely because each task's classes are represented by separate output units. Output units which represent the first task are never active for later tasks and thus, the pressure on these units to never activate on later tasks is likely greater than the pressure on these units to remember the previous task (from (online-)EWC). Therefore, in the *ewc-c*10 and *online-ewc-c*10 conditions the networks share their output units between multiple classes so that all output units are active during a training phase, regardless of the task being learnt. This led to a dramatic improvement in the EWC variants' ability to retain knowledge of previous tasks such that the *ewc-c*10 condition could classify CIFAR-10 and SVHN to 31.67% and 44.93% accuracy and the *online-ewc-c*10 condition to 12.64% and 15.19% accuracy after learning all tasks. This suggests that the EWC variants are ineffective for learning tasks which do not share their output representations but are moderately effective when they do. This is especially the case when task specific weights are used with EWC. However, findings still show that Pseudo-Recursal clearly outperforms the EWC variants as it loses only 1.67% of CIFAR-10's accuracy compared to EWC's 52.73% and online-EWC's 71.14%.

The results for the *reh-limit* condition convey that the classifier can retain the majority of its knowledge of past tasks. However, Pseudo-Recursal still clearly outperforms it, retaining 9.6% more accuracy on CIFAR-10 and 13.11% more on SVHN. This demonstrates that using the GAN model is more effective than simply remembering an equivalent subset of past items. The subset used in the *reh-limit* condition was built by simply randomly selecting items from the previous task's datasets. Although more inventive strategies exist, random selection is effective and at least presents the best results in reinforcement learning when compared to a variety of other selection strategies (Isele and Cosgun, 2018).

Compared to the EWC variants and standard pseudo-rehearsal, the main disadvantage of Pseudo-Recursal is that a generative network is required for pseudo-rehearsal to work on this deep network. However, pseudo-rehearsal is also applied to the generative model so that the size of this network is constant. Another disadvantage of Pseudo-Recursal is that it takes considerably more training time because the generator must also be trained. Furthermore, training in the classifier is done by pairing the same number of pseudo-items as novel task's items in each mini-batch, doubling the total number of items trained on. While no attempts were made in this thesis to experiment with the number of pseudo-items required in each mini-batch, experiments using sweep

rehearsal (Silver *et al.*, 2015; Robins, 1995) suggest that such a large number may not be necessary, provided that the pseudo-items are dynamic.

As previously mentioned, one of the major advantages of pseudo-rehearsal methods is that they do not constrain the network's weights to retain similar values. The weights' values can be freely changed as long as the input-output mapping remains similar for previous tasks. Originally, pseudo-rehearsal was applied to simple tasks where the pseudo-items generated by a random distribution covered the whole input space. This means that the pseudo-items represent the network's function over the whole input space such that changes that are made to accommodate the new task are as local as possible to the input space of the new task (Robins and Frean, 1998). However, a GAN generates pseudo-items near the actual inputs of previous tasks in a much larger and sparser space. Therefore, Pseudo-Recursal is advantageous over standard pseudo-rehearsal as the network is only constrained to retain similar input-output mappings around the space of previous inputs. In other parts of the space, the network is free to vary so that it can accomodate new information with even fewer restrictions.

Pseudo-Recursal requires a significant amount of temporary storage to hold either a separate copy of the neural network before training or the population of labelled pseudo-items for rehearsal. It seems implausible that the biological cognitive system would have the mechanisms to implement a storage system that could either directly copy weights to a separate but identical neural architecture or quickly and accurately store these pseudo-items. Robins (1997) suggested a more plausible implementation could have two separate weights for every connection in the network w_o and w_n. w_o holds the weights that retain knowledge of all previously learnt tasks, whereas w_n represents how these weights should be changed to learn the new task while retaining the previous tasks. Therefore, the output of the network while learning the new task is determined by the weights $w_o + w_n$, where weight updates are only applied to w_n, and pseudo-inputs are labelled by the network with weights w_o. When the new task has been successfully learnt, the weights w_o are updated with the values of $w_o + w_n$ and then w_n is reinitialised to 0. Robins (1997) found that this implementation performed indistinguishably from the basic pseudo-rehearsal method, suggesting that it would be just as effective in Pseudo-Recursal. Although the biological system is useful for hypothesising how to achieve machine intelligence, this thesis is not concerned with biological plausibility of the implementation and thus, is satisfied by simply using temporary storage.

3.4 Conclusions

This chapter has demonstrated that combining GANs with pseudo-rehearsal is an effective method for solving the catastrophic forgetting problem. Pseudo-Recursal has major advantages over other methods such as EWC because it does not require the network to grow for each new task and the network does not have any hard constraints on how individual neurons should learn the new task. Furthermore, Pseudo-Recursal was found to perform very similarly to rehearsing datasets stored from previous tasks. Given this solution was so effective in image classification, the remainder of this thesis will investigate its potential in the reinforcement learning domain.

Chapter 4

Pseudo-Rehearsal in Deep Neural Networks for Continual Reinforcement Learning

4.1 Extending Pseudo-Rehearsal to Reinforcement Learning

This chapter extends pseudo-rehearsal to deep neural networks to solve catastrophic forgetting in the reinforcement learning domain. Section 2.1.2 briefly discussed that the deadly triad causes many deep reinforcement learning methods to become unstable, including DQNs. This chapter proposes the Reinforcement-Pseudo-Rehearsal (RePR) model which further extends pseudo-rehearsal so that it can be successfully applied to reinforcement learning while still preventing instability. RePR achieves this by using a dual memory system to segregate short and long-term learning alongside replacing the cross-entropy loss functions with ones which are more suitable for continual reinforcement learning.

A dual memory system separates learning into two stages. The first stage involves the short-term memory (STM) system learning new information and, in the second stage, this information is consolidated into the long-term memory (LTM) system alongside previously learnt information. This dual memory system is biologically similar to the roles of the mammalian hippocampus and cortex previously discussed in Section 3.1.3. Dual memory systems have also been utilised in early research on catastrophic forgetting (Ans and Rousset, 1997, 2000; French, Ans, and Rousset, 2001) and

are still successfully applied in recent methods including FearNet (Kemker and Kanan, 2018), Deep Generative Dual Memory Network (Kamra, Gupta, and Liu, 2017) and Progress and Compress (Schwarz *et al.*, 2018).

4.1.1 Short-Term Memory System

Short-Term DQN

In RePR, the STM system is comprised of a DQN along with its experience replay. Each time this system is given a task to learn, it reinitialises its DQN and experience replay and learns the task as it would in Deep Q-Learning (i.e. by minimising the loss function specified in Equations 2.5-2.6). A summary of how the components in this system interact with the environment can be found in Figure 4.1. Pseudo-code demonstrating the basic training procedure for the STM system can also be found in Appendix B as Algorithm 2.

The STM system makes the acquisition of the new task simple as it can be learnt with the standard Deep Q-Learning loss function, which has already proven successful in deep reinforcement learning. Importantly, this allows the new task to be learnt without interference from previously learnt tasks. In future work, it would be sensible to replace the experience replay in the STM system with a generative model. This generative model would learn to generate samples from the current task that could be used to prevent the agent from forgetting how to act in less recent states. This extension was not investigated in this thesis because the generator would need to be trained alongside the agent and this would have dramatically increased the already extensive training times which are inevitable in deep reinforcement learning. Importantly, using the experience replay in the STM system does not invalidate any of the objectives previously outlined in Section 1.3, because the experience replay is a fixed size throughout learning and only contains data from the current task (not previous tasks).

4.1.2 Long-Term Memory System

The LTM system is comprised of a DQN and a GAN. The role of the DQN is to consolidate knowledge from the STM system about how to act in the new task while retaining knowledge about previous tasks. The GAN's role is to learn to generate items which are representative of the new task while retaining the ability to generate items which are representative of previous tasks. The GAN is also used to support the DQN in retaining knowledge of previous tasks through pseudo-rehearsal. A summary of how

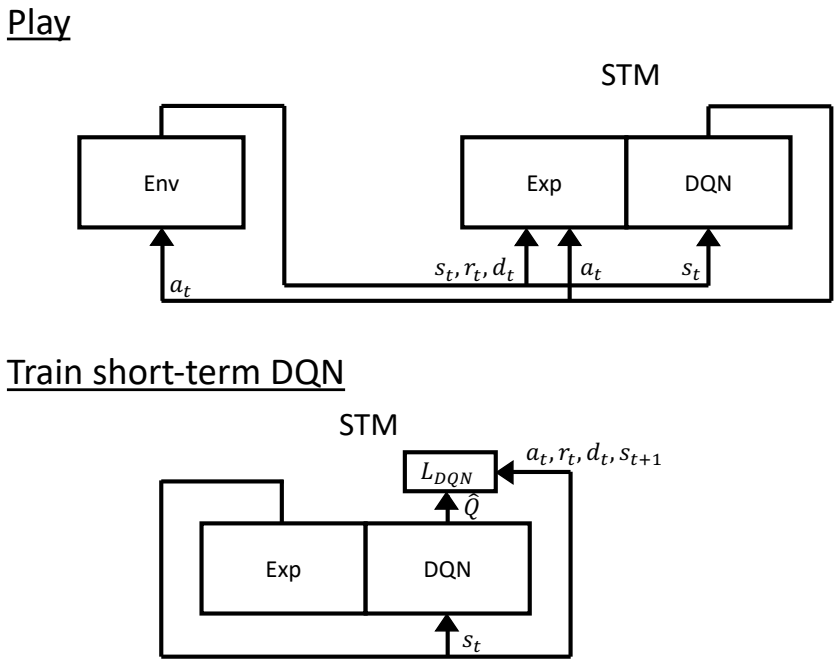

Figure 4.1: Illustration of the training procedure for RePR's STM system. The system is trained while the model is simultaneously interacting with the environment, collecting transitions in its experience replay (Exp). Mathematical notation relates to the previously specified loss function L_{DQN}.

the components in this system interact with the environment and each other can be found in Figure 4.2. Pseudo-code demonstrating the basic training procedure for the LTM system can also be found in Appendix B as Algorithm 2.

Long-Term DQN

Before training, the long-term DQN is initialised to the weights of the previous long-term DQN. The long-term DQN uses knowledge distillation[1] to learn the current task. More specifically, the long-term DQN interacts with the current environment, collecting states in the experience replay. These states are passed through the short-term DQN to obtain target Q-values (representing how the short-term agent has learnt to act in the environment). The output of the long-term DQN is then dragged toward replicating these target Q-values.

[1]In this case, the short-term DQN is the teacher and the long-term DQN is the student.

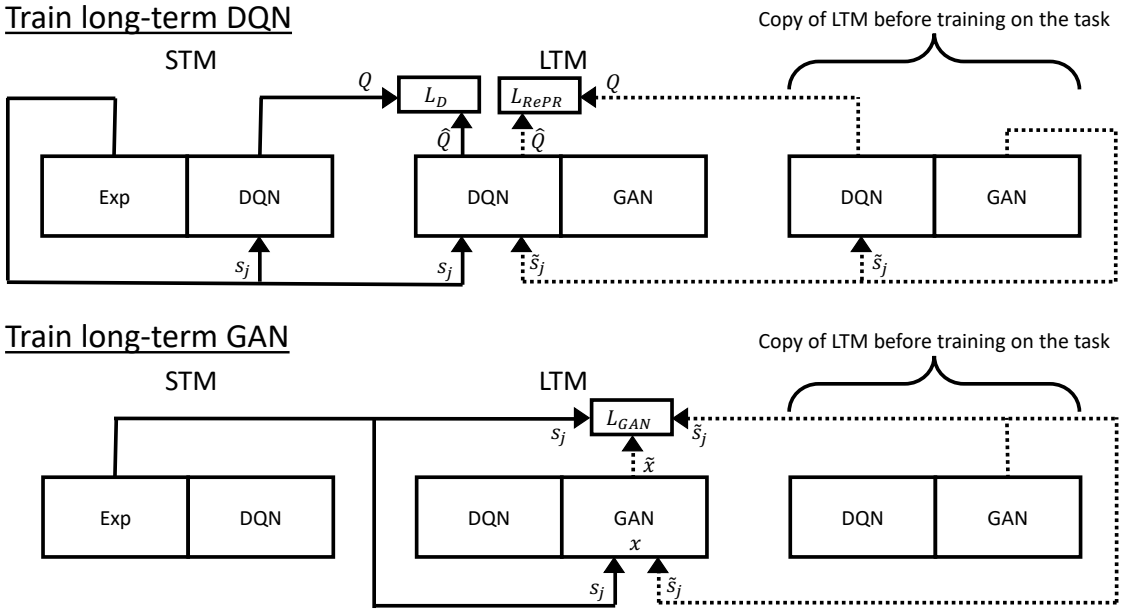

Figure 4.2: Illustration of the training procedure for RePR's LTM system. Solid lines represent the information flow of real data (for learning the new task) and dashed lines represent the information flow of generated data (for retaining previous tasks with pseudo-rehearsal). In practice, the experience replay (Exp) is not stationary while training the long-term DQN. Instead, the experience replay is storing recent transitions from the long-term DQN interacting simultaneously with the environment. The DQN and GAN are independently trained. Mathematical notation relates to the loss functions L_D and L_{RePR}, as well as $L_{DiscRePR}$ and $L_{GenRePR}$ (i.e. L_{GAN}).

The long-term DQN also uses pseudo-rehearsal to retain knowledge of previous tasks. This is achieved by generating pseudo-items from the previous GAN[2], calculating their desired Q-values by passing them through the previous long-term DQN and then training the new long-term DQN to continue outputting these values for the given items.

The specific loss functions for training the long-term DQN are:

$$L_{LTM} = \frac{1}{N}\sum_{j=1}^{N} \alpha L_{D_j} + (1-\alpha) L_{RePR_j}, \tag{4.1}$$

$$L_{D_j} = \sum_{a}^{A} \left(Q(s_j, a; \theta_i) - Q(s_j, a; \theta_i^+)\right)^2, \tag{4.2}$$

$$L_{RePR_j} = \sum_{a}^{A} \left(Q(\widetilde{s}_j, a; \theta_i) - Q(\widetilde{s}_j, a; \theta_{i-1})\right)^2, \tag{4.3}$$

where L_{D_j} is the distillation loss for teaching a new task and L_{RePR_j} is the pseudo-rehearsal loss for retaining previously learnt tasks. A state s_j is drawn from the current task's experience replay. θ_i are the weights of the long-term DQN while learning the current task, θ_i^+ are the weights of the short-term DQN after learning the current task and θ_{i-1} are the weights of the long-term DQN after learning the previous task. Pseudo-states $\widetilde{s}_j$ are representative of previously learnt tasks and are generated by the previous GAN. N is the mini-batch size, A is the set of possible actions and α is a scaling factor weighting the importance of learning the current task compared to retaining previous tasks via pseudo-rehearsal ($0 \leq \alpha \leq 1$). A large α value gives high importance to learning the current task and vice versa for a low value. $\alpha = 0.5$ weights the importance evenly. In complex sequential learning tasks, α should be set to a low value so that new knowledge is slowly consolidated into the long-term DQN, without substantially disrupting previously learnt knowledge. However, setting α too low can dramatically increase the time required to train the long-term DQN on the new task.

Long-Term GAN

The general training procedure for the GAN is identical to Pseudo-Recursal. That is, the GAN is trained to produce items from the current task, along with pseudo-items which are representative of previous tasks, as generated by the previous GAN.

[2]This GAN is trained on the previously learnt tasks and thus, its pseudo-items are representative of these tasks.

The only exception is that the GAN's training procedure is improved by changing the loss functions so that the WGAN-GP (Gulrajani, Ahmed, Arjovsky, Dumoulin, and Courville, 2017) loss function is used with a drift term (Karras, Aila, Laine, and Lehtinen, 2018) added to it. This drift term stops the discriminator's output from drifting too far away from zero for both real and fake inputs.

The GAN's training items are drawn such that:

$$x = \begin{cases} s_j, & \text{if } r < \frac{1}{T} \\ \widetilde{s}_j, & \text{otherwise} \end{cases} \tag{4.4}$$

where r is a number randomly drawn from $U[0, 1)$ and s_j is a randomly selected item in the current task's experience replay. T is the number of tasks learnt (including the new task) and $\widetilde{s}_j$ is an item generated randomly from the previous GAN. This item could represent an item from task 1 to $i - 1$.

These items are learnt using the following loss functions for the discriminator ($L_{DiscRePR}$) and generator ($L_{GenRePR}$):

$$\begin{aligned} L_{DiscRePR} = D(\widetilde{x}; \phi) - D(x; \phi) + \lambda(\|\nabla_{\hat{x}} D(\hat{x}; \phi)\|_2 - 1)^2 \\ + \epsilon_{drift} D(x; \phi)^2 + \epsilon_{drift} D(\widetilde{x}; \phi)^2, \end{aligned} \tag{4.5}$$

$$L_{GenRePR} = -D(\widetilde{x}; \phi), \tag{4.6}$$

where x is a training item specified by Equation 4.4. $\widetilde{x}$ is an item produced by the current generative model ($\widetilde{x} = G(z; \varphi)$). z is an array of latent variables $z = U(-1, 1)$. D and G are the discriminator and generator networks with the weights ϕ and φ respectively. ϵ is a random number $\epsilon \sim U(0, 1)$, $\hat{x} = \epsilon x + (1 - \epsilon)\widetilde{x}$, $\lambda = 10$ and $\epsilon_{drift} = 1e^{-6}$. During training, the weights of the discriminator and generator are updated on alternating steps using their corresponding loss function.

4.1.3 Summary of the RePR Model

In short, the RePR model comprises two systems. The first system is the STM system which contains a DQN and an experience replay. This STM system learns how to act in a new task using Deep Q-Learning (reinforcement learning). The second system is the LTM system which contains a DQN and a GAN. The short-term DQN is used to teach the long-term DQN how to act in the new task, while pseudo-rehearsal is used to remember previous tasks by rehearsing items generated from the GAN. This GAN

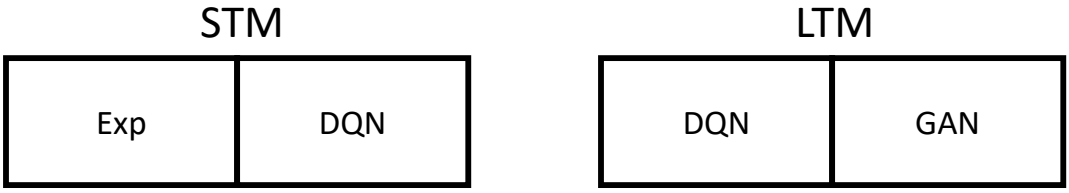

Figure 4.3: Summary of the components that make up the RePR model.

also uses pseudo-rehearsal so that it can learn to generate items representative of the new task, while also retaining the ability to generate items representative of previously learnt tasks. A summary of the components found in RePR is illustrated in Figure 4.3.

4.1.4 Related Work

The Deep Generative Dual Memory Network (Kamra *et al.*, 2017) is the model which most closely resembles RePR. It also incorporates a dual memory system alongside a generative model and pseudo-rehearsal. The Deep Generative Dual Memory Network has a number of short-term components, with each component comprising its own learner model and generative model. Each of these short-term components is taught one of the new tasks. One of the primary driving factors of this model is biological plausibility and therefore, the network uses these generative models to produce samples which are then taught to the long-term learner model and long-term generative model. In contrast, RePR only uses a single short-term learner and transfers knowledge to the LTM system using real samples from the current environment. As shown in this chapter's results (Section 4.3), real samples rather intuitively allow the new task to be more accurately transferred to LTM, although it assumes that the network still has access to the current task or, at least, its experience replay, when transferring information. The Deep Generative Dual Memory Network was primarily developed to prevent catastrophic forgetting in image classification where both the STM and LTM systems are learning through the same cross-entropy loss function. RePR has been developed for reinforcement learning and instead, changes these loss functions to promote stability in reinforcement learning. More specifically, the short-term learner uses a reinforcement learning loss function so that it can learn the new task in isolation,

whereas the long-term learner acquires the new task with mean squared error. In reinforcement learning, target output values are constantly changing with the policy of the network. Isolating reinforcement learning to the STM system is important as it allows the LTM system to learn the new task through consistent target output values, simplifying the LTM system's learning process.

Before RePR there was very limited research using pseudo-rehearsal to prevent catastrophic forgetting while sequentially learning reinforcement tasks. Caselles-Dupré, Garcia-Ortiz, and Filliat (2018) combined pseudo-rehearsal with another generative model (i.e. a Variational Auto-Encoder (Kingma and Welling, 2014)) to sequentially learn two reinforcement tasks through State Representation Learning. The two reinforcement tasks were very simple. The agent navigated a 2D world from a small 64×3 input representing the colour of objects in front of the agent. The two tasks differed only by the colour of objects the agent must collect. Authors' successfully prevented catastrophic forgetting while sequentially learning these tasks. The RePR model has been developed to prevent catastrophic forgetting in much more challenging tasks with much larger input spaces. These tasks require more powerful reinforcement learning algorithms such as Deep Q-Learning, along with deep convolutional layers for both learning how to act and generating representative inputs.

Since RePR, pseudo-rehearsal methods have also been used in models to retain; information relative to previously seen environments (Caselles-Dupré, Garcia-Ortiz, and Filliat, 2019) and experiences from previously seen environments (Ketz, Kolouri, and Pilly, 2019). However, these models do not attempt to prevent catastrophic forgetting in a reinforcement learning agent.

4.2 Methodology

The remainder of this chapter experimentally investigates the effectiveness of RePR as a solution to catastrophic forgetting while learning complex reinforcement tasks. The first experiment compares RePR to relevant baselines and state-of-the-art methods on a sequence of tasks. The second experiment aims to further confirm RePR's capabilities by investigating its forgetting on the reversal of the task sequence used in the first experiment. The final experiment analyses whether RePR's weights are multi-purposed such that weights which are important in the computation of one task are also important in other tasks.

4.2.1 Environments

In sequential reinforcement learning, each task is defined by a separate environment. The environments used in this thesis come from the Atari 2600 home video game console. This console contains 49 arcade style games which have become a test bed for deep reinforcement learning because these games include a point system which can easily be transferred into rewards given to an agent. These games are translated to environments as per Mnih *et al.* (2015). When there is no change in the games' points, this translates to a neutral reward 0. Gaining points translates to the positive reward 1 and losing points (or in some cases the opponent scoring points against you) translates to negative reward -1. The size of the action space is 18, which represents different combinations of both joystick movements and pressing of the fire button. If the agent selects an action from this space which is not allowed in the current environment, its action is replaced by randomly selecting an action from the set of valid actions. This keeps the size of the action space consistent and shared across all games learnt by the agent. Each action chosen by the agent is repeated 4 times[3] in the game and if, the game gives the agent multiple lives, an episode (current game session) terminates after a single life has been lost during training and terminates when all lives are lost during evaluation stages. The states given to the agent by the environment are representative of the 4 most recent observable frames[4]. These frames are then preprocessed by rescaling them to 84×84 and extracting the luminance. This results in every state in the environment being $84 \times 84 \times 4$, where the third channel now represents time, rather than colour channels like in the previous image classification experiments.

The Atari games used in this chapter were Road Runner, Boxing and James Bond. These were selected from the pool of games in which the DQN could outperform human performance by a wide margin (Mnih *et al.*, 2015). These games were also selected as games which the DQN implemented in this thesis could learn to a similar level as Mnih *et al.* (2015). The DQN used in RePR was based upon Mnih *et al.* (2015) with a few minor changes found to enhance the acquisition of individual tasks. These changes are explained further in the following subsections.

[3]For example, if the agent selected the action left, this action is executed 4 times (left-left-left-left) before the agent chooses another action.

[4]Observable frames exclude frames on which actions were repeated. However, some objects only appear in every second frame of the game (due to a limitation of the console) and therefore, an observed frame is actually the maximum colour value of the current frame and the previously excluded frame.

Table 4.1: DQN architecture for RePR, where CONV is a convolutional layer and FC is a fully connected layer.

DQN				
Input: $84 \times 84 \times 4$				
layer	# units/filters	filter shape	filter stride	activation
CONV	32	8×8	4×4	ReLU
CONV	64	4×4	2×2	ReLU
CONV	64	3×3	1×1	ReLU
FC	512			ReLU
FC	18			

The agent's goal in Road Runner is to run to the left of the screen away from the coyote while also collecting points and dodging a variety of hazards, including trucks. Later, the agent should also learn to lead the coyote into these hazards so that it is slowed down. In Boxing, the agent must learn to navigate a 2D boxing ring and punch an opposition boxer in the face while also avoiding being punched in the face. Positive rewards are given for landing a punch on the opponent and negative rewards for receiving a punch. Finally, James Bond has the agent learn to control a vehicle, avoid hazards and score points for shooting a particular object. In this task, the screen is moving toward the right side, forcing the agent to move in that general direction.

4.2.2 Network Architectures

The architecture of the DQN networks used in this chapter remained the same as Mnih *et al.* (2015) and can be found in Table 4.1. The architecture of the GAN's discriminator and generator can be found in Table 4.2 and Table 4.3 respectively.

4.2.3 Training and Evaluation

The primary difference between Mnih *et al.* (2015)'s DQN and the one used in this chapter is that the DQN used by Mnih *et al.* (2015) was trained by the RMSProp optimiser with gradients clipped between $[-1, 1]$, whereas the DQN in this chapter uses Tensorflow's RMSProp optimiser (without centering) with gradients clipped to their global norm. In addition, the DQN's bias weights are initialised to 0.01 and

Table 4.2: The GAN's discriminator network architecture for RePR, where CONV is a convolutional layer and FC is a fully connected layer.

Discriminator				
Input: $84 \times 84 \times 4$				
layer	# units/filters	filter shape	filter stride	activation
CONV	64	5×5	3×3	Leaky ReLU
CONV	128	5×5	2×2	Leaky ReLU
CONV	256	5×5	2×2	Leaky ReLU
FC	1			

the weights are initialised with $\mathcal{N}(0, 0.01)$, where all values greater than 2 standard deviations from the mean are re-drawn. The hyper-parameters used by the DQN are shown in Table 4.4, with changes from Mnih *et al.* (2015) emphasised in bold.

The hyper-parameters used to train the GAN can be found in Table 4.5. The last layer of the generator uses the Tanh activation function and therefore, the items the GAN is trained on are rescaled to the same space $(-1, 1)$ by applying $f(x) = 2(\frac{x}{255} - 0.5)$ to each raw pixel value. Pseudo-samples are rescaled back to the pixel space before being used for pseudo-rehearsal. Random noise $U(-10, 10)$ was also applied to real and generated items before applying rescaling and giving them to the discriminator[5].

Each time a new game is presented, the short-term DQN learns it for 20 million frames before it is taught to the long-term DQN for 20m frames and the GAN for 200,000 iterations. The loss functions used for training these components are the functions specified in Section 4.1. The one exception is that teaching the first game to the long-term DQN is skipped and instead, the network is initialised to the weights of the short-term DQN. When the LTM system is pseudo-rehearsing items from previous games, the pseudo-items are drawn from a temporary array of 250,000 samples generated by the previous GAN. The importance of learning the new task vs. retaining previous tasks is set to $\alpha = 0.55$. Other values ($\alpha = 0.35$ and $\alpha = 0.75$) were also tried and produced similar results.

After the short-term DQN is trained on a game, its final weights are set to the

[5]Applying random noise to the discriminator's input is used to improve the stability of the GAN during training (Arjovsky and Bottou, 2017).

Table 4.3: The GAN's generator network architecture for RePR, where DECONV is a deconvolutional layer and FC is a fully connected layer. Layers which use batch normalisation (Ioffe and Szegedy, 2015) before the activation function are marked with an 'x'.

Generator					
Input: 100 latent variables					
layer	# units/filters	filter shape	filter stride	batch-norm	activation
FC	$256 \times 7 \times 7$			x	ReLU
DECONV	256	5×5	3×3	x	ReLU
DECONV	128	5×5	2×2	x	ReLU
DECONV	64	5×5	2×2	x	ReLU
DECONV	4	5×5	1×1		Tanh

values which produced the largest average score over 250,000 observable frames. The final weights after training the long-term DQNs are set to the values which produced the lowest error over 250,000 observable frames. The final weights for the GAN are just the weights at the end of training. While training the short-term and long-term DQN, the network is evaluated on the current task and all previously learnt tasks after every 1m observable frames. The evaluation procedure used is based on Mnih *et al.* (2015), where the network plays each game for 30 episodes. The DQN selects its actions using an ε-greedy policy with $\varepsilon = 0.05$. This is also the evaluation procedure used when reporting the final network's results on each of the tasks, where the mean and standard deviations are calculated from the final scores in each of the 30 episodes.

4.2.4 Experimental Conditions

The first experiment compares RePR to the Deep Generative Dual Memory Network, state-of-the-art EWC variants and other useful baselines. The experimental conditions in this experiment are sequentially taught to play Road Runner, Boxing and James Bond, in that order. All of the experimental conditions are trained 3 times on the same set of seeds and results are averaged across these seeds. Unless stated otherwise, all conditions use the same dual memory system so that they can be fairly compared. The specific conditions are:

Table 4.4: Hyper-parameters for RePR's DQN model.

Hyper-parameter	Value	Description
mini-batch size	32	Number of examples drawn for calculating the stochastic gradient descent update.
replay memory size	**200,000**	Number of frames in experience replay which samples from the current game are drawn from.
history length	4	Number of recent frames given to the agent as an input sequence.
update target hz	**5,000**	Number of frames which are observed from the environment before the target network is updated.
discount factor	0.99	Discount factor (γ) for each future reward.
action repeat	4	Number of times the agent's selected action is repeated before another frame is observed.
update frequency	4	Frequency of observed frames which updates to the current network occur on.
learning rate	0.00025	Learning rate used by Tensorflow's RMSProp optimiser.
momentum	0.0	Momentum used by Tensorflow's RMSProp optimiser.
decay	**0.99**	Decay used by Tensorflow's RMSProp optimiser.
ϵ	$\mathbf{1e^{-6}}$	Epsilon used by Tensorflow's RMSProp optimiser.
replay start size	50,000	The number of frames which the experience replay is initially filled with (using a uniform random policy).
no-op max	30	Maximum number of "do nothing" actions performed at the start of an episode ($U[1, \text{no-op max}]$).
initial ε-greedy	1.0	Initial ε-greedy exploration rate.
final ε-greedy	0.1	Final ε-greedy exploration rate.
final ε-greedy frame	1,000,000	Number of frames seen by the agent before the linear decay of the exploration rate reaches its final value.

Table 4.5: Hyper-parameters for RePR's GAN model.

Hyper-parameter	Value	Description
mini-batch size	100	The number of items trained from during each mini-batch.
learning rate	0.001	Learning rate for the Adam optimiser.
β_1	0.0	First moment decay rate for the Adam optimiser.
β_2	0.99	Second moment decay rate for the Adam optimiser.
ϵ	1×10^{-8}	Epsilon value for the Adam optimiser.
decay_{bn}	0.9	Decay value for batch normalisation.
ϵ_{bn}	1×10^{-5}	Epsilon value for batch normalisation.
leak	0.2	Leakage value for Leaky ReLU.

- *std*: Learns the environments sequentially, without using methods to prevent catastrophic forgetting. This is the lower bound of performance.
- *reh*: Learns the environments sequentially, while still rehearsing real items drawn from the previous tasks' experience replays. The targets for these rehearsal items are produced by passing the items through the previous LTM system. This is the upper bound of performance.
- *RePR*: Learns the environments sequentially with the RePR model, rehearsing pseudo-items representative of the previously learnt environments.
- *PR*: Learns the environments sequentially with the same method as RePR, except the dual memory system is removed and thus, the new task is learnt using the Deep Q-Learning loss function while pseudo-items representative of previous tasks are also being rehearsed.
- *DGDMN*: Learns the environments sequentially with the Deep Generative Dual Memory Network. This condition uses a similar method to RePR, except that a GAN (identical to the one used in the LTM system) is incorporated into the STM system. This short-term GAN learns to generate data representative of only the new task. The short-term GAN is then used to generate the data which teaches both the long-term GAN and long-term DQN the new task.

- *reh-limit*: Learns the environments sequentially, while still rehearsing real items drawn from the previous tasks' experience replays. However, in this case, the number of real items available to rehearse is limited to 600 which is equivalent to the number of items that could be stored using the same amount of memory[6] as RePR's generative network (which has approximately 4m parameters/weights).

- *ewc*: Learns the environments sequentially, while retaining past knowledge with EWC. To fairly compare the methods on the same network, EWC is used without task specific weights. A grid search was used to selected the λ parameter for EWC, searching the values $\lambda = [50, 100, 150, 200, 250, \mathbf{300}, 350, 400]$, where the best parameter[7] found is indicated in bold.

- *online-ewc*: Learns the environments sequentially, while retaining past knowledge with online-EWC. Schwarz *et al.* (2018) found that the Progress and Compress method performed competitively against its online-EWC component alone. Therefore, Progress and Compress' connections between the STM and LTM systems (which presumably encourage weight sharing) were not used. This meant that the network's architecture remained the same as other experimental conditions in this section and thus, could more fairly be compared. A grid search was used to selected the λ and γ parameters for online-EWC, searching the values $\lambda = [25, \mathbf{75}, 125, 175]$ and $\gamma = [0.95, \mathbf{0.99}]$, where the best parameters found are indicated in bold.

In the second experiment RePR is compared to similar baselines while learning the reverse sequence, i.e. James Bond, Boxing and Road Runner. These conditions are identical to the *RePR* and *reh* conditions except the tasks are learnt in the reverse order and therefore, they will be referred to as *RePR-rev* and *reh-rev* respectively.

Finally, the Fisher overlap score (Kirkpatrick *et al.*, 2017) is calculated for RePR while learning the two previously mentioned sequences of games. The Fisher overlap

[6]In the previous chapter, the number of rehearsal items was calculated by assuming that each free parameter/weight used in the generative model was instead storing an input value from a real item. In this chapter, the number of bits used by every parameter/weight in the generative model is calculated and this is divided by the number of bits it would take to store a real item (i.e. the sequence of 4 images which make up a state). The latter calculation results in a larger number of stored images.

[7]The test scores for the final networks were min-max normalised using the minimum and maximum values found across all the testing episodes played while learning the tasks in STM. The parameter selected was the parameter associated with the network with the best average normalised test score across the games.

score measures the similarities between two tasks' Fisher information matrices and these similarities can then be used to estimate whether a similar set of the DQN's weights are important to the network's output. The score is bounded between 0 and 1, where a high score suggests that the network uses similar weights for determining its output on the two tasks and a low score suggests the network uses separate weights for determining the two task's outputs. The score is calculated by $1 - d^2$, where:

$$d^2(\hat{F}_1, \hat{F}_2) = \frac{1}{2} tr\Big(\hat{F}_1 + \hat{F}_2 - 2(\hat{F}_1 \hat{F}_2)^{\frac{1}{2}}\Big), \tag{4.7}$$

given $\hat{F}_1$ and $\hat{F}_2$ are the two tasks' Fisher information matrices which are normalised to have a unit trace. Fisher information matrices are approximated by Equation 2.9 using 100 batches of samples drawn from each task's experience replay.

4.3 Results and Discussion

The results for the first experiment are displayed in Figure 4.4 and Figure 4.5. The *std* condition learns the 3 tasks without attempting to prevent catastrophic forgetting. Unsurprisingly, the condition suffers dramatically from catastrophic forgetting as it essentially forgets how to play the previously learnt games and thus, at the end of the sequential learning sequence can only play the final task, James Bond, to a satisfactory standard. On the contrary, the *reh* condition learns to play new games with essentially no forgetting. This is effectively the upper bound on performance that RePR could achieve. This is because the only difference between the *reh* and *RePR* conditions is that the items rehearsed are real items from previous tasks, rather than items generated by a GAN. Therefore, it is promising that the results show that the *RePR* condition performs very similarly to the *reh* condition. The only subtle difference between these two conditions is that RePR displays a gentle decline in performance on the first task (Road Runner). After training, the final RePR model was evaluated on all of the tasks it had previously learnt. This network achieved the scores 22042 (±5375), 82 (±12) and 468 (±155), which are on par with Mnih *et al.* (2015)'s results for DQNs trained individually on the tasks, which were 18257 (±4268), 72 (±8) and 577 (±176), and considerably above human expert performance levels (7845, 4, 407). Collectively, these results convey that RePR can successfully prevent catastrophic forgetting in deep reinforcement learning.

Figure 4.6 displays real images from the tasks alongside pseudo-images generated by RePR's GAN. It is relatively simple for humans to discriminate between these items.

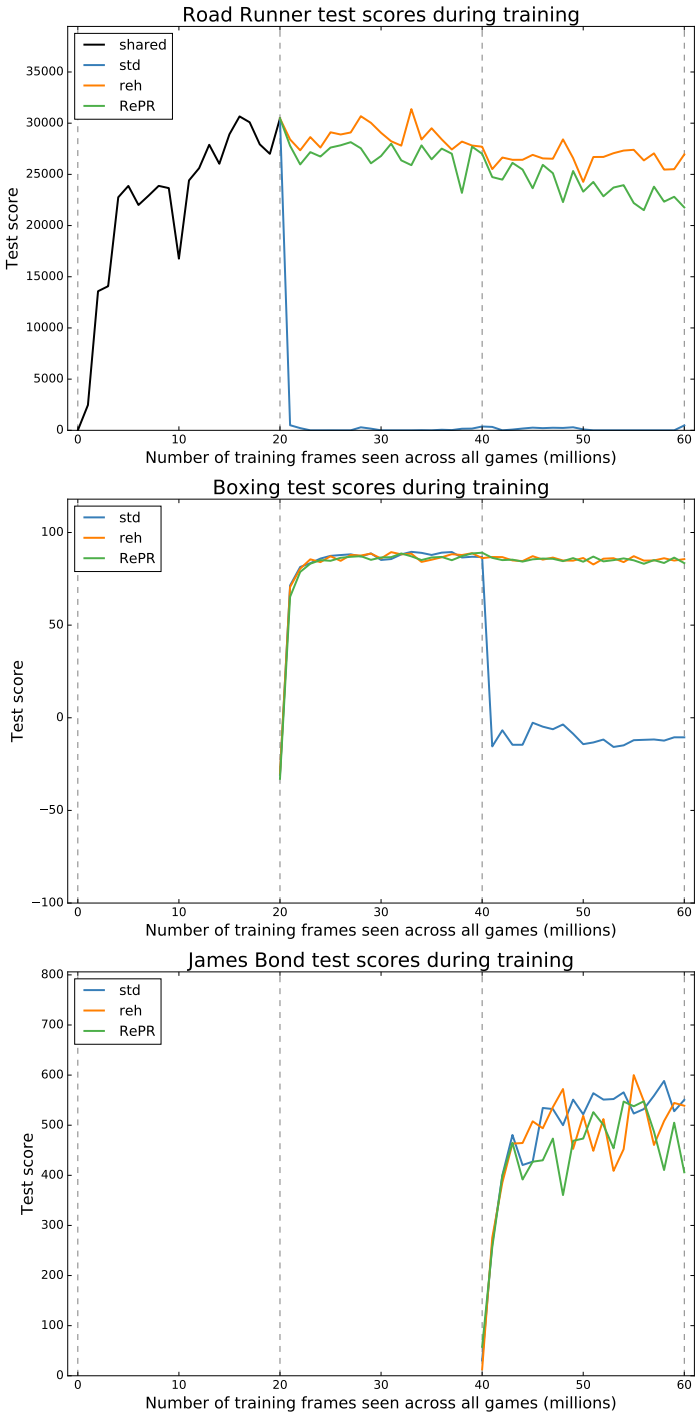

Figure 4.4: Results of the RePR model compared to the *std* and *reh* conditions. Scores are recorded by evaluating the long-term DQN after every 1m observable training frames. Task switches occur at the dashed lines, in the order Road Runner, Boxing and then James Bond.

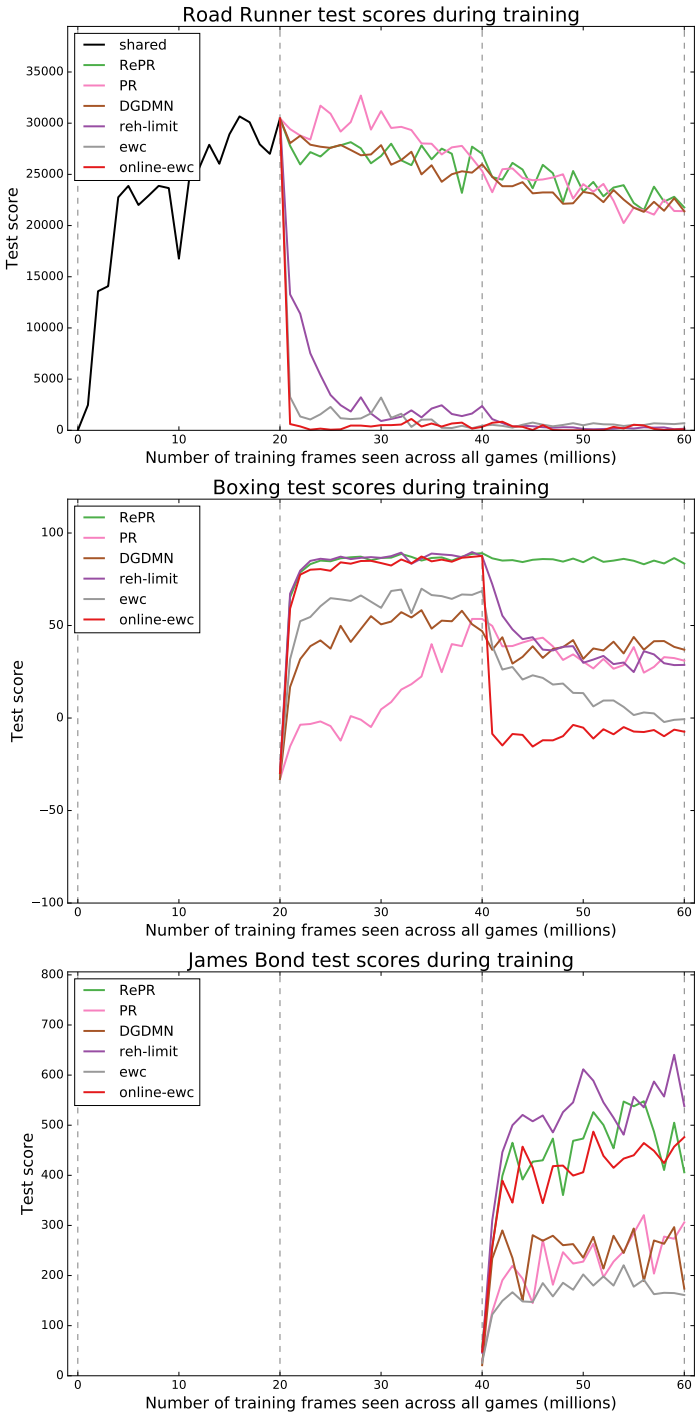

Figure 4.5: Results of the RePR model compared to the *PR*, *DGDMN*, *reh-limit*, *ewc* and *online-ewc* conditions. Scores are recorded by evaluating the long-term DQN after every 1m observable training frames. Task switches occur at the dashed lines, in the order Road Runner, Boxing and then James Bond.

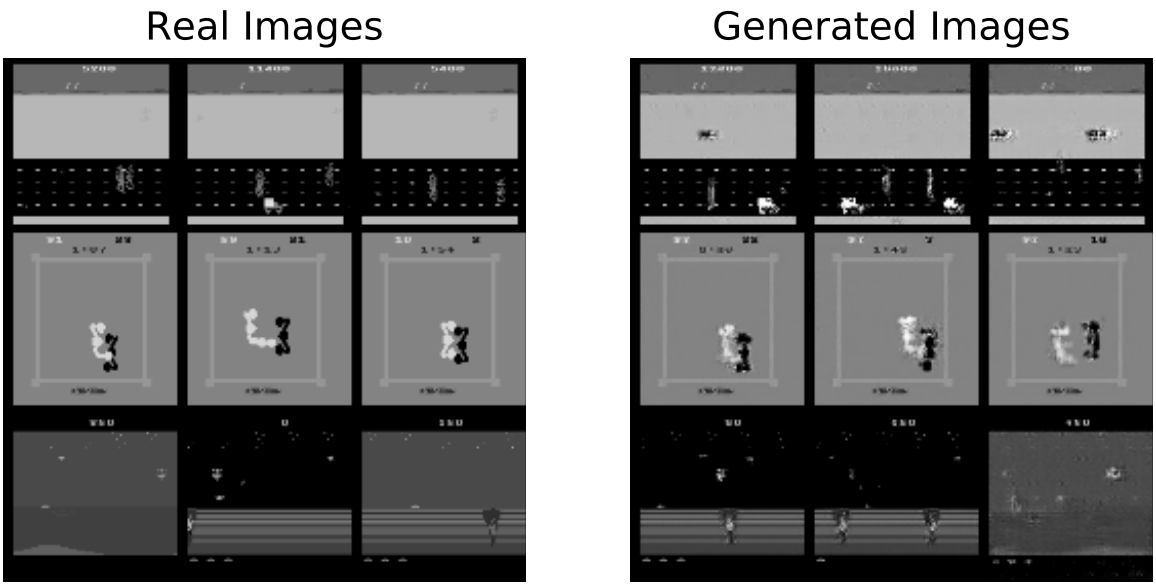

Figure 4.6: Images from states drawn randomly from the previous tasks' experience replays (real) and pseudo-states generated by a GAN after being sequentially taught Road Runner, Boxing and then James Bond. Images shown are the first image of each of the state's four frame sequence. Each row contains images from the same task.

However, the pseudo-images are still very representative of the previous games, with important features such as characters and dangers visible. Furthermore, the above results demonstrate that these features are enough to retain considerable knowledge of previous tasks.

The PR condition in Figure 4.5 resembles an ablation study[8] which demonstrates the importance of the dual memory system used in RePR. Simply removing this dual memory system prevents the model from learning new tasks to its full potential. More specifically, the PR condition demonstrates that the first task, Road Runner, can be learnt and retained similarly to RePR, but its potential to learn further tasks is interfered with and consequently, the condition has lower performance and slower convergence times. Deep Q-Learning is complicated, due, partially, to the fact that its loss function uses a dynamic network to estimate the discounted reward associated with future states. Therefore, this thesis hypothesises that the dual memory system

[8]An ablation study is used to measure the contribution of a particular component of a model by removing it.

is extremely beneficial for learning new tasks as it isolates the complex reinforcement learning task to the STM system. This system does not attempt to retain knowledge of previous games and therefore, the new task can be initially learnt through reinforcement learning without the added pressure of retention. This knowledge can then be simply transferred to the LTM system through distillation, where the target values do not change as they do in reinforcement learning.

The Deep Generative Dual Memory Network was initially designed for continual learning in image classification. This meant that extensive changes were necessary to convert this algorithm from the image classification domain to the reinforcement domain. Essentially, these changes made the model very similar to RePR with the only addition being a short-term GAN, which was used to train the LTM system on new tasks. The addition of the short-term GAN is a clear disadvantage of the Deep Generative Dual Memory Network model as it means that the LTM system is being trained with approximated data, which might not fully capture all of the intricacies in the new task. Although approximated data is capable of being used for pseudo-rehearsal, it is unlikely that its quality will be sufficient to teach the task from scratch. Results from the $DGDMN$ condition confirm this, as both Boxing and James Bond were not learnt as successfully as the $RePR$ condition, which teaches these tasks to the LTM system using real data. Furthermore, Boxing also showed an observable decrease in performance when James Bond was being learnt by the LTM system. Learning and retaining Road Runner in the LTM system does not appear to be difficult in the $DGDMN$ condition. However, this is primarily because the short-term GAN and short-term DQN had their networks copied to the LTM system after the STM system had learnt the first task. Because these networks learn from real data in the STM system, they are not affected by the disadvantages of learning the new task from generated data. The decision to copy the networks to the LTM system was made to keep the learning procedure as similar as possible to the other conditions for a fair comparison. Overall, these results strongly illustrate the advantages of RePR over the Deep Generative Dual Memory Network, as real data is shown to be more effective for training the LTM system than data generated from a second GAN.

As previously mentioned, RePR relies on access to the current task, or at least its experience replay, when transferring information to the LTM system, whereas the Deep Generative Dual Memory Network does not. Retaining the current task's experience replay until after knowledge has been transferred to the LTM system is not a particularly difficult constraint. Although the experience replay might be large, it only keeps

data from the current task and therefore, its size is fixed and will not grow with the addition of new tasks.

The *reh-limit* condition (which uses the same allocation of memory as the generator to store and rehearse real items from previous tasks) performs substantially worse than using RePR's generator. More specifically, Road Runner is completely forgotten and Boxing loses approximately half its score over the duration of training. This is important as it demonstrates that, given limited storage, using a generative model is more effective than directly storing a small sample of real items for rehearsal.

RePR was also compared to the state-of-the-art variants of EWC. The results clearly display RePR outperforming both the *ewc* and *online-ewc* conditions, which demonstrate substantial forgetting of previously taught tasks. Online-EWC was found to learn new tasks more successfully compared to EWC which was slightly more successful at retaining previous tasks. Both EWC and online-EWC showed similar catastrophic forgetting to the *std* condition (which makes no effort to prevent forgetting), with the exception of the *EWC* condition which noticeably attempts to retain Boxing, but over time still forgets how to play the game.

The results of the EWC variants were considerably poorer than the authors' originally reported. However, this is due to the sequential learning environment used in the current experiment being significantly more challenging than the experimental conditions these methods were originally reported in. Most notably, the agents had to retain the DQN's Q-values for the tasks, rather than the agent's policy. Furthermore, tasks were only learnt by the agent in one interval, rather than being allowed to revisit previously learnt tasks to relearn or continue learning them. To ensure that these algorithms could still retain previously learnt tasks in less challenging conditions, EWC and online-EWC were tested in less challenging experimental conditions. More specifically, each task was only learnt by the long-term agent for 5m observable frames and the agent had to only learn the policy from the short-term DQN by minimising cross-entropy. The results from this test, shown in Figure 4.7, displayed that both EWC (*ewc-policy*) and online-EWC (*online-ewc-policy*) could successfully retain previous tasks under these less challenging conditions. This confirms that the added difficulty of long retention times (without revisiting tasks), and requiring Q-values to be learnt, results in a more difficult sequential learning task which can more comprehensively explore the capabilities of the models.

The second experiment aims to demonstrate that RePR's retention is not simply limited to the particular sequence investigated in the first experiment. Figure 4.8

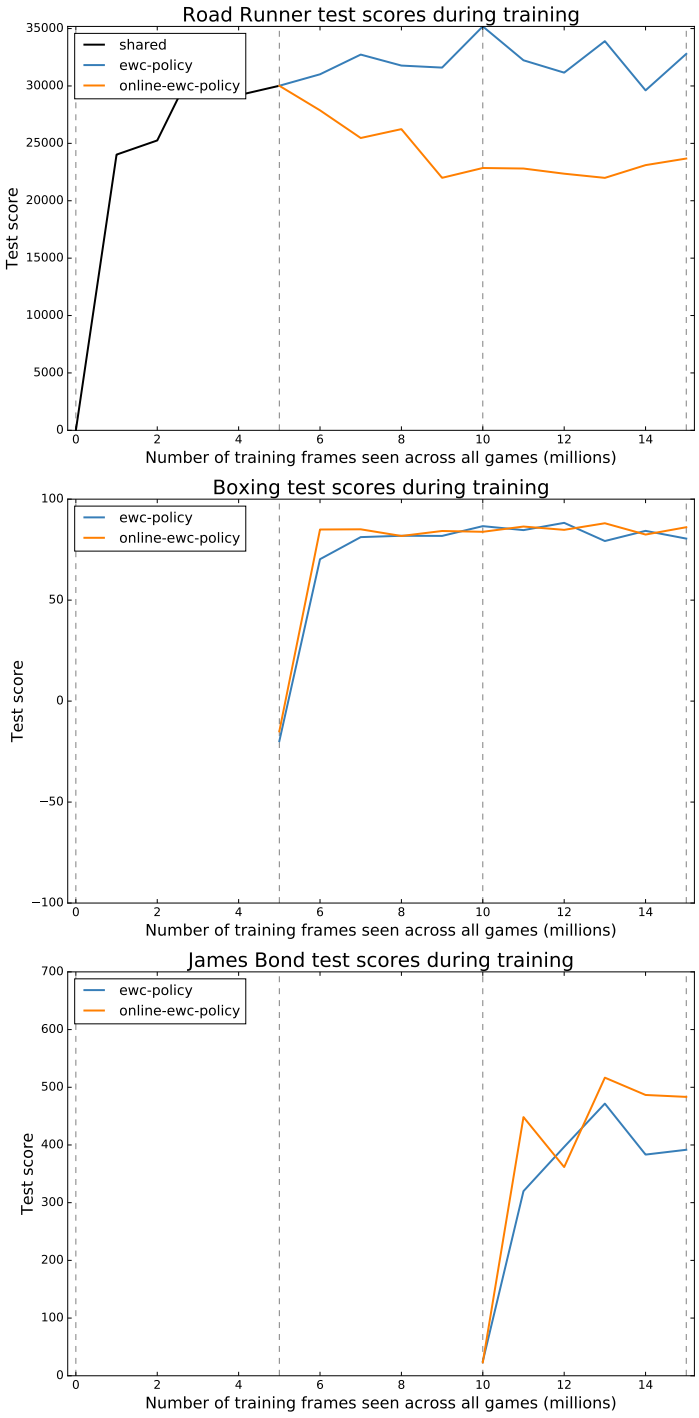

Figure 4.7: Results of the (online-)EWC implementations when learning in less challenging experimental conditions. Scores are recorded by evaluating the long-term DQN after every 1m observable training frames. Task switches occur at the dashed lines, in the order Road Runner, Boxing and then James Bond. Results were recorded with a single, consistent seed.

illustrates RePR's and rehearsal's learning on the reversed task sequence (James Bond, Boxing and Road Runner). The results convey that both methods are capable of retaining moderate knowledge of previously learnt games. RePR showed a noticeable amount of forgetting in Boxing, whereas the *reh-rev* condition did not. Furthermore, both conditions found it difficult to retain high performance in James Bond, with RePR performing marginally lower than the *reh-rev* condition on this task.

Poirier and Silver (2005) suggest that for a given set of tasks, the order in which the tasks are learnt will not affect the mean performance of the final model. However, the results from this experiment contradict this because in the reverse ordering, the model loses considerably more performance on previous tasks, without counterbalancing this by attaining greater performance on the most recently learnt task. However, this is likely because the tasks' reward functions have different scales, which result in retention of James Bond being weighted as much less important in the loss function compared to learning and retaining other tasks. Therefore, considerably more knowledge of James Bond is forgotten in the reverse ordering of tasks compared to when Road Runner must be retained in the original ordering. This issue will be further investigated in Section 5.3.

In the final experiment, RePR's Fisher overlap score is calculated for all possible task pairings when learning tasks in the order learnt in the first experiment (Road Runner, Boxing and James Bond), along with the reversed order used in the second experiment. These scores can be found in Table 4.6. When tasks were learnt in the first order, the overlap score was relatively high between Road Runner and Boxing, and relatively low between other task pairings. This suggests that the final network learnt by RePR had more similar important weights between Road Runner and Boxing than between other tasks. This could be due to either there being more similarity between the optimal policy learnt to solve these two tasks compared to James Bond, or that earlier learning is capable of sharing more computation than later task learning. To confirm the former, the Fisher overlap scores are also calculated when learning the reverse order of tasks. In this case, the score remained higher between Road Runner and Boxing than other task pairs and thus, conveyed that RePR is sharing more computation between these games due to their similarity and not due to the order they were learnt in.

Similar to Pseudo-Recursal, RePR's main advantage over popular weight constraint methods, such as EWC, is that it gives the network the freedom to change its weights as long as the resulting network still performs previously learnt games to a similar

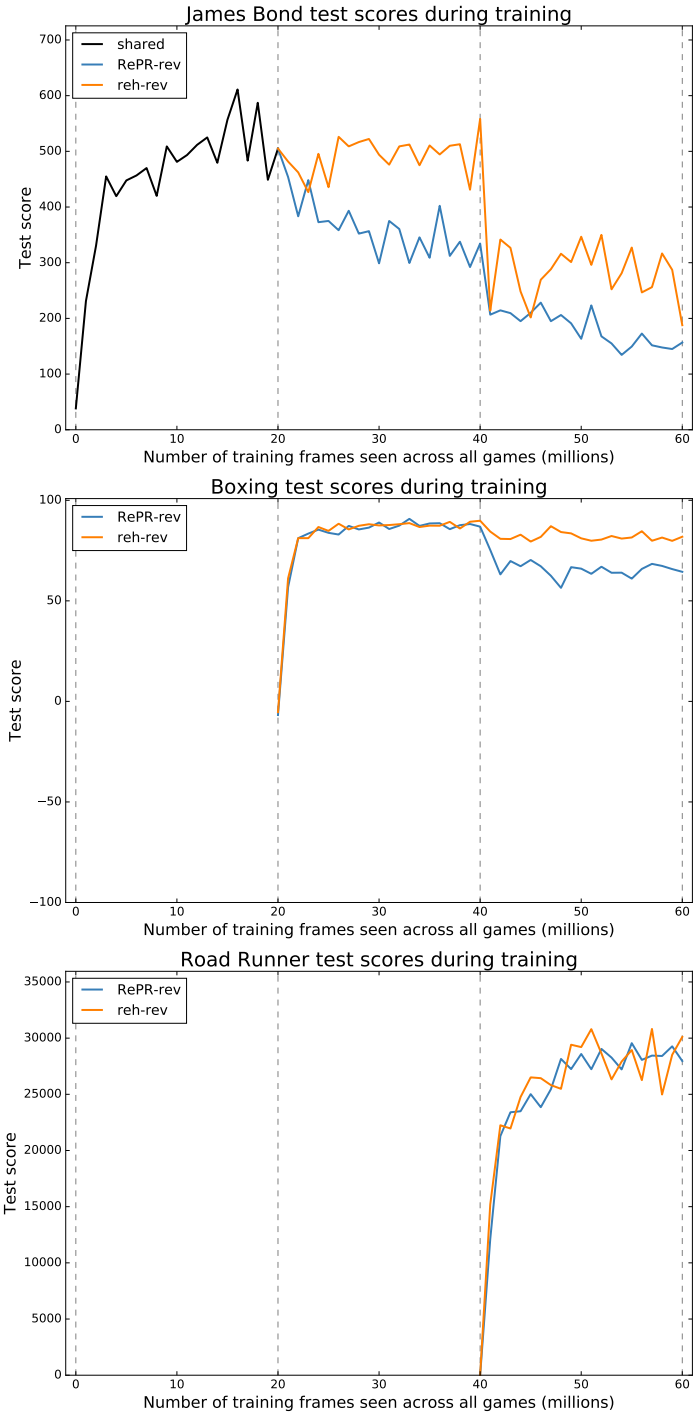

Figure 4.8: Results of the RePR model compared to the *reh-rev* condition. Scores are recorded by evaluating the long-term DQN after every 1m observable training frames. Task switches occur at the dashed lines, in the order James Bond, Boxing and then Road Runner.

Table 4.6: Fisher overlap scores between all possible task pairings.

Condition	Road Runner & Boxing	Road Runner & James Bond	Boxing & James Bond
RePR	0.691	0.233	0.198
RePR-rev	0.753	0.192	0.110

standard. Therefore, RePR has more freedom than these other methods to restructure itself when consolidating new knowledge. The results provided in this chapter, which compare RePR to variants of EWC, experimentally verify that this freedom is beneficial in reinforcement learning.

In this chapter and the previous one, it has been assumed that the model knows when the task it is learning switches. In many situations, this might be an unfair assumption and thus, a separate component would need to be included for detecting this change. Another potential limitation is that the task sequence which RePR was tested on was only 3 tasks long, which is relatively short compared to methods such as Progress and Compress which tested a sequence of 6 Atari games. Consequently, the above experiments do not evaluate the capacity of the GAN to identify when it fails and whether it does so gradually or drastically. Some of these questions are investigated in the following chapter.

4.4 Conclusions

RePR is the first method to apply pseudo-rehearsal to complex reinforcement learning tasks which require powerful generative models such as GANs. RePR does not directly store any data from previously learnt tasks or use task specific weights and thus, its memory requirements do not change as the number of tasks increases. The results in this chapter demonstrate RePR alleviating catastrophic forgetting for sequences of 3 reinforcement learning tasks and outperforming the popular variants of EWC used in this domain. Furthermore, RePR is also shown to promote sharing of weights, with such sharing being more prominent in similar tasks. Given the success of this solution, the next chapter will investigate some of RePR's limitations along with some improvements to the method.

Chapter 5

Further Evaluating and Improving Pseudo-Rehearsal for Continual Reinforcement Learning

This chapter aims to further evaluate and improve the RePR model. The first section investigates the performance of RePR when applied to another popular type of deep reinforcement learning known as Actor-Critic methods. The second section investigates whether RePR's LTM system needs to retain the value function, or if the policy function alone is enough to allow learning to continue on partially learnt tasks. The third section aims to improve the generator used in RePR by encouraging it to accurately reproduce image features that are important to retention. The final section evaluates RePR's performance over an extended sequence of reinforcement learning tasks, identifying whether challenging conditions result in either gradual or catastrophic forgetting.

5.1 Extending Pseudo-Rehearsal to Actor-Critic Methods

5.1.1 Actor-Critic Methods

Another very popular type of deep reinforcement learning is Actor-Critic methods (Sutton and Barto, 2017), which split reinforcement learning into two learning functions. The first is the policy function, which describes how the agent should act in an environment to maximise its reward. The other is the value function, which approximates the expected cumulative reward the agent should be able to attain from a given state. The

value function is used to encourage the agent to change its policy so that it outputs desirable actions.

Actor-Critic methods have advantages over DQNs, such as being able to be used in continuous action spaces. Importantly, these methods might also have advantages over DQNs in relation to continual learning. More specifically, separating the policy function from the value function would allow the importance of retaining each of these functions to be weighted relative to one another and in many circumstances, it may only be necessary to retain the policy function, as this thesis explores in Section 5.2.

In Actor-Critics, the policy and value functions are usually implemented by two neural networks. These two networks are not independent but, rather, share all but the final few layers of their networks. The policy network (otherwise known as the actor) takes a state as input and outputs a probability distribution over the possible actions. This distribution represents how certain the network is that each action will lead to large cumulative rewards from the given state. The action, which is taken from a given state, is randomly chosen using this probability distribution. The value network (otherwise known as the critic) takes a state as input and outputs a single value representing how advantageous that state is for maximising cumulative reward.

The network representing the value function is updated using the loss function:

$$L_{AC\text{-}value} = \big(y_t - V(s_t; \theta_t^v)\big)^2, \tag{5.1}$$

$$y_t = \begin{cases} r_t, & \text{if } d_t \\ r_t + \gamma V(s_{t+1}; \theta_t^v), & \text{otherwise} \end{cases} \tag{5.2}$$

where d_t represents whether the environment is terminal at time step t and V represents the value function network with the weights θ_t^v.

The network representing the policy function is updated using the loss function:

$$L_{AC\text{-}policy} = -\log \pi(a_t|s_t; \theta_t^\pi)\big(y_t - V(s_t; \theta_t^v)\big) + \beta \sum_{a_i}^{A} \pi(a_i|s_t; \theta_t^\pi) \log \pi(a_i|s_t; \theta_t^\pi), \tag{5.3}$$

where A is the set of possible actions and π is the network representing the policy function, with the weights θ_t^π (some of which are shared with θ_t^v). The first term in this loss function uses an advantage function to measure how the cumulative reward from taking an action compares to the value function of the policy. Actions that perform better than the policy's value function are reinforced (so that they are more

likely to be taken) and those that perform worse are punished. The second term is a regulariser which prevents the policy from deciding too quickly which action is best to take in each state. This is important as it stops the policy from converging too quickly on a local minimum. The regulariser is the entropy of the policy which is weighted with a small hyper-parameter β.

Typically the Actor-Critic learns asynchronously, where multiple agents interact with different copies of the environment and update a shared model. This makes an asynchronous Actor-Critic's learning on-policy and thus, only recent states, rewards and terminals are used when updating the networks. Each agent is not in sync when it is interacting with the environment and therefore, agents are generally at different stages (time-steps) in the environment at any one time. This is advantageous because collectively the recent data generated by the agents interacting with the environment is close to i.i.d. Therefore, an experience replay is not necessary to prevent forgetting in a single environment. However, this limits applications of the algorithm to situations where multiple independent instances of the environment exist. In many real world applications of continual learning this is not possible and therefore, this chapter opts to incorporate an experience replay to sample s_t, a_t, r_t, d_t, s_{t+1}, resulting in off-policy learning.

5.1.2 Applying Actor-Critic Methods to RePR

The previous chapter described how the RePR model could achieve sequential reinforcement learning when using DQNs as the reinforcement learner. By changing the network architectures and loss functions used by RePR, sequential learning can also be achieved with Actor-Critic methods. This specific variation of the learning model will be referred to as AC-RePR. Similar to the previously proposed RePR model, AC-RePR uses a dual memory model which contains the policy and value function networks of an Actor-Critic in the STM system, along with the experience replay. The LTM system also contains the policy and value function networks of an Actor-Critic, along with a GAN (or other generative model). A summary of how the components in the STM system and the LTM system interact with the environment and each other can be found in Figure 5.1 and Figure 5.2 respectively.

In the STM system, the Actor-Critic networks are trained through the typical reinforcement learning procedure set out in the loss functions described in Equation 5.1-5.3. In the LTM system, the GAN is trained identically to RePR and the Actor-Critic networks are trained on the new task through another adaption of knowledge

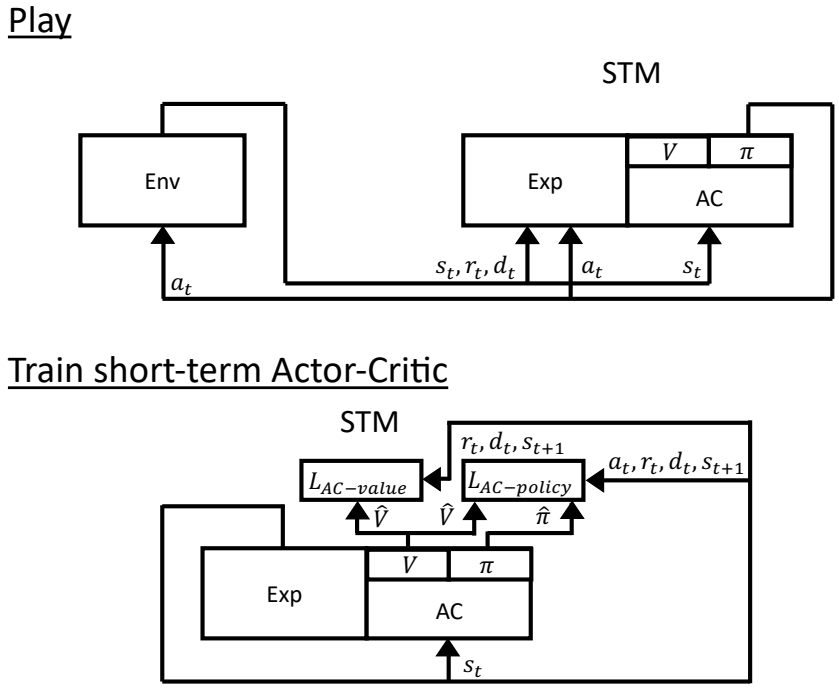

Figure 5.1: Illustration of the training procedure for AC-RePR's STM system. The system is trained while the model is simultaneously interacting with the environment, collecting transitions in its experience replay (Exp). Mathematical notation relates to the previously specified loss functions $L_{AC\text{-}value}$ and $L_{AC\text{-}policy}$.

distillation, along with pseudo-rehearsal to prevent forgetting of previous tasks. The policy function is transferred and retained through cross-entropy and the value function through mean squared error. More specifically, the loss functions for the long-term networks are:

$$L_{AC\text{-}LTM} = \frac{1}{N}\sum_{j=1}^{N} \alpha L_{AC\text{-}D_j} + (1-\alpha) L_{AC\text{-}RePR_j}, \tag{5.4}$$

$$L_{AC\text{-}D_j} = \eta\big(V(s_j;\theta_i^v) - V(s_j;\theta_i^{v+})\big)^2 + (1-\eta) CE\big(\pi(s_j;\theta_i^\pi), \pi(s_j;\theta_i^{\pi+})\big), \tag{5.5}$$

$$L_{AC\text{-}RePR_j} = \eta\big(V(\widetilde{s}_j;\theta_i^v) - V(\widetilde{s}_j;\theta_{i-1}^v)\big)^2 + (1-\eta) CE\big(\pi(\widetilde{s}_j;\theta_i^\pi), \pi(\widetilde{s}_j;\theta_{i-1}^\pi)\big), \tag{5.6}$$

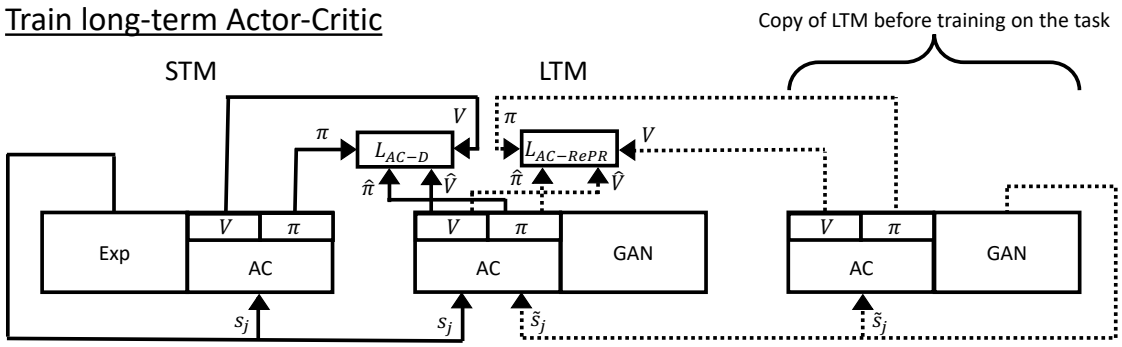

Figure 5.2: Illustration of the training procedure for AC-RePR's LTM system. Solid lines represent the information flow of real data (for learning the new task) and dashed lines represent the information flow of generated data (for retaining previous tasks with pseudo-rehearsal). In practice, the experience replay (Exp) is not stationary while training the long-term Actor-Critic. Instead, the experience replay is storing recent transitions from the long-term Actor-Critic interacting simultaneously with the environment. The GAN is independently trained with the same procedure as RePR and thus, is excluded from this figure. Mathematical notation relates to the loss functions $L_{AC\text{-}D}$ and $L_{AC\text{-}RePR}$.

where $L_{AC\text{-}D_j}$ is the distillation loss for teaching a new task and $L_{AC\text{-}RePR_j}$ is the pseudo-rehearsal loss for retaining previously learnt tasks. A state s_j is drawn from the current task's experience replay. θ_i^v and θ_i^π are the weights of the long-term Actor-Critic networks while learning the current task, θ_i^{v+} and $\theta_i^{\pi+}$ are the weights of the short-term Actor-Critic networks after learning the current task and θ_{i-1}^v and θ_{i-1}^π are the weights of the long-term Actor-Critic networks after learning the previous task. Pseudo-states $\widetilde{s}_j$ are representative of previously learnt tasks and are generated by the previous GAN. N is the mini-batch size, α is a scaling factor weighting the importance of learning the current task compared to retaining previous tasks via pseudo-rehearsal ($0 \leq \alpha \leq 1$) and η is a scaling factor weighting the importance of retaining the value function compared to the policy function ($0 \leq \eta \leq 1$). In practice, the target output patterns the policy network learns through distillation are one-hot encoded.

In short, the AC-RePR model is identical to the RePR model, except the short-term

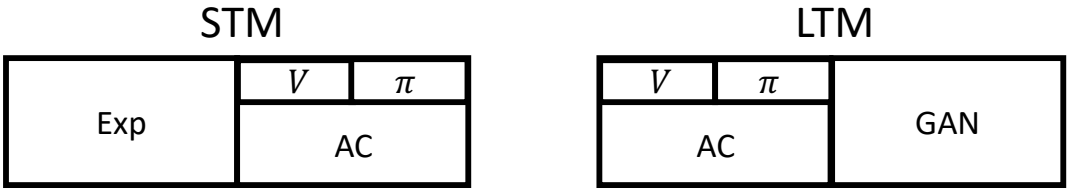

Figure 5.3: Summary of the components that make up the AC-RePR model.

and long-term DQNs are each replaced with Actor-Critics. An Actor-Critic comprises two networks each retaining either the policy or value function. The short-term Actor-Critic learns a new task using the Actor-Critic method of reinforcement learning. The long-term Actor-Critic learns the policy and value function of the new task by being taught by the short-term Actor-Critic, while also using pseudo-rehearsal to remember previous tasks' policy and value functions. Policies are retained and taught to the long-term Actor-Critic using a variation of the cross-entropy loss function, whilst value functions are retained and taught to the long-term Actor-Critic through a variation of mean squared error. Training of the GAN does not differ from RePR. A summary of the components found in AC-RePR is illustrated in Figure 5.3.

5.1.3 Methodology

The remainder of this section will experimentally investigate whether the AC-RePR model can also perform similarly to the previous RePR model. The experimental procedure is virtually identical to the previous chapter's first experiment (see Section 4.2). The only differences are the AC-RePR model's network architecture and additional hyper-parameters, as described below.

Network Architectures

The actor and critic network architectures used in the AC-RePR model are shown in Table 5.1 and Table 5.2 respectively. The architecture of the GAN used in AC-RePR is identical to RePR, as outlined in Section 4.2.2.

Table 5.1: Actor architecture for AC-RePR, where CONV is a convolutional layer and FC is a fully connected layer. Layers whose weights are shared with the critic are marked with an 'x'.

Actor					
Input: $84 \times 84 \times 4$					
layer	# units/filters	filter shape	filter stride	activation	shared
CONV	32	8×8	4×4	ReLU	x
CONV	64	4×4	2×2	ReLU	x
CONV	64	3×3	1×1	ReLU	x
FC	512			ReLU	
FC	18			Softmax	

Training and Evaluation

The training and evaluation procedure is identical to RePR. Actions are chosen using the policy network's learnt distribution, along with ϵ-greedy being used to allow further exploration once the policy has converged. The weighting of the first term in the policy's loss function ($L_{AC\text{-}policy}$) is halved to encourage the value function to converge quicker than the policy function. The additional hyper-parameters used while training AC-RePR are shown in Table 5.3. To ensure that AC-RePR is as similar as possible to RePR, the n-step variation (Mnih, Badia, Mirza, Graves, Lillicrap, Harley, Silver, and Kavukcuoglu, 2016) of Actor-Critic methods is not used.

Experimental Conditions

The conditions in this experiment are taught the environments Road Runner, Boxing and then James Bond as per the previous chapter. All of the experimental conditions are trained 3 times on the same set of seeds and results are averaged across these seeds. The *AC-RePR* condition learns these environments sequentially, with the Actor-Critic variation of the RePR model. The *RePR* condition is identical to the previous chapter's condition and thus, its results remain the same. Finally, the *RePR-policy* condition uses the RePR model to learn the tasks while only retaining the policy function in the LTM system. Similar to the AC-RePR model, the policy function is taught to and retained in the LTM system through a variation of cross-entropy, where the best

Table 5.2: Critic architecture for AC-RePR, where CONV is a convolutional layer and FC is a fully connected layer. Layers whose weights are shared with the actor are marked with an 'x'.

Critic					
Input: $84 \times 84 \times 4$					
layer	# units/filters	filter shape	filter stride	activation	shared
CONV	32	8×8	4×4	ReLU	x
CONV	64	4×4	2×2	ReLU	x
CONV	64	3×3	1×1	ReLU	x
FC	512			ReLU	
FC	1				

action in each state (as determined by the short-term DQN) is one-hot encoded before being taught to the LTM system with distillation. In this condition it was necessary to decrease α to 0.05, increasing the importance of retaining previous tasks over learning new tasks. This value was changed because learning only the policy function removes the effect the different tasks' reward functions have on increasing the importance of retaining the tasks Road Runner and Boxing compared to learning the task James Bond.

5.1.4 Results and Discussion

Figure 5.4 compares the *AC-RePR* condition to the *RePR* condition. The results convey the Actor-Critic variant of RePR performing very similarly to the DQN variant of RePR, with both variants learning the tasks to a similar standard. However, the AC-RePR model appears to retain Road Runner to a higher standard. This is likely because the evaluation results for the AC-RePR model are produced by having the actor network play the games. This network only retains the policy function which, as shown in the previous chapter, suffers less substantially from forgetting. We confirm this by introducing a further experimental condition called *RePR-policy* which uses a DQN to learn the task in the STM system but only retains the policy function in the LTM system. Consequently, this condition also retains Road Runner more successfully than the *RePR* condition. This raises the question; if performance is improved by

Table 5.3: Additional hyper-parameters for the AC-RePR model.

Hyper-parameter	Value	Description
η	0.8	Scaling factor weighting the importance of retaining the value function compared to the policy function.
initial β	0.01	Initial weighting for the policy's entropy.
final β	0.001	Final weighting for the policy's entropy.
final β frame	8,000,000	Number of frames seen by the agent before the linear decay of the weighting for the policy's entropy reaches its final value.

having a component that learns only the policy function, why should the continual learner retain the value function at all? In Actor-Critic methods, the value function is used to improve the policy. Therefore the next section will investigate whether it is important for a continual learner to retain the value function so that it can continue learning a previously seen task.

5.1.5 Conclusions

In conclusion, this section has demonstrated that the RePR model can easily be generalised to other deep reinforcement algorithms. When RePR is applied to Actor-Critic methods, the policy and value functions are learned separately by the LTM system. Because the policy suffers less from catastrophic forgetting, a minor improvement can be observed in the performance of the Actor-Critic variant compared to the previously proposed DQN variant (RePR).

5.2 Continuing Learning a Partially Learnt Task

Both the long-term and short-term agents in the RePR model are DQNs. This means that the STM system learns each game by learning each state's Q-values. Then, this knowledge is transferred to the LTM system so that it learns the Q-values from the current game's states, while also retaining the Q-values from the previous games' states. These Q-values represent both the policy and value function of the network. Therefore,

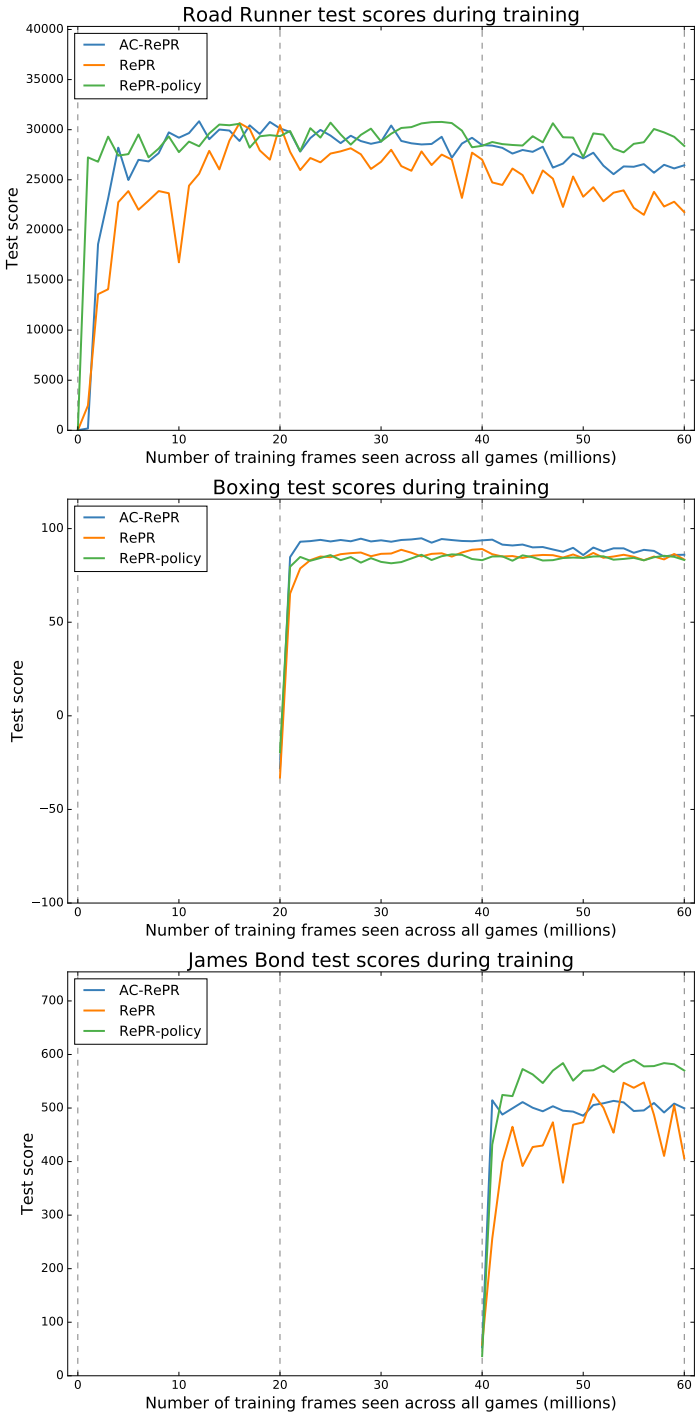

Figure 5.4: Results comparing the *AC-RePR* condition to the *RePR* and *RePR-policy* conditions. Scores are recorded by evaluating the long-term agent after every 1m observable training frames. Task switches occur at the dashed lines, in the order Road Runner, Boxing and then James Bond.

they include all the information learnt about an environment. If the model was required to further learn one of the previously seen tasks, the model could use all the information in the LTM system to continue learning from where it had stopped, rather than having to relearn the majority of the task again.

In the previous section, the AC-RePR model uses Actor-Critic methods to separate the agent's policy and value functions. Both of these functions are then transferred to the LTM system. Again this provides the LTM system with all the knowledge learnt from previous tasks.

Some of the previously proposed algorithms in this field do not transfer both the policy and value function to the LTM system. For example, Progress and Compress (Schwarz *et al.*, 2018) also uses Actor-Critic methods so that the STM system learns the policy and value function separately. However, the LTM system is only taught the policy function. The value function is important for learning the policy function, but if an optimal policy has already been learnt the value function is not required. The Progress and Compress algorithm assumes that if learning was to continue on a previously seen task, the value function could be quickly relearnt. The experiment in this section aims to investigate whether a continual learner can continue learning a familiar task, without its LTM system retaining the task's value function.

5.2.1 Methodology

Training and Evaluation

The models being trained and evaluated in this section are identical to the RePR and AC-RePR model from previous sections. All of the hyper-parameters also remain the same, except the ϵ-greedy value is only linearly decayed for the first 350,000 frames in unseen environments and β (weighting the policy's entropy) is only linearly decayed for the first 5m frames in unseen environments. The other main differences to the previous chapter's methods are in the training procedure. More specifically, the STM system is taught Road Runner for 1m frames, which is then copied to the LTM system. Next, the STM system is reinitialised and Boxing is learnt for 5m frames. This knowledge is then transferred to the LTM system for 10m frames. Finally, learning of Road Runner is continued by reinitialising the STM system with the weights from the LTM system and then further learning the game for 4m frames[1]. The STM system's knowledge can

[1]As per usual, the STM system is only trying to learn the current game and thus, does not use pseudo-rehearsal.

then be transferred to the LTM system using another 10m frames of learning.

Experimental Conditions

All of the experimental conditions are trained 3 times on the same set of seeds and results are averaged across these seeds. The conditions are as follows:

- *RePR-partial*: Learns the environments with the RePR model, retaining the Q-values (representing both the policy and value function) in the LTM system.
- *RePR-partial-policy*: Learns the environments with the RePR model, retaining only the policy function in the LTM system. The policy function is taught to and retained in the LTM system through a variation of cross-entropy, where the best action in each state (as determined by the short-term DQN) is one-hot encoded before being taught to the LTM system with distillation. Before continuing learning Road Runner, all layers in the short-term DQN are reinitialised to the values of the long-term DQN, except for the output layer.
- *AC-RePR-partial*: Learns the environments with the Actor-Critic variation of the RePR model, retaining both the policy and value function in the LTM system.
- *AC-RePR-partial-policy*: Learns the environments with the Actor-Critic variation of the RePR model, retaining only the policy function in the LTM system. This means that the non-shared layers in the short-term critic network are reinitialised to random values before further learning Road Runner.

5.2.2 Results and Discussion

In this experiment, all conditions successfully learnt Boxing to approximately the same standard. The results of the conditions learning Road Runner are illustrated in Figure 5.5 for RePR and Figure 5.6 for AC-RePR. In the *RePR-partial* condition, the LTM system retains the DQN's Q-values. Results confirm that retaining this information allows the short-term DQN to effectively continue learning Road Runner. However, the *RePR-partial-policy* condition only retains the policy function in the LTM system. This condition shows that retaining the policy function alone is not enough for a DQN to continue learning a familiar task because further learning is hindered after reinitialising the short-term DQN's weights with the values in the LTM system. This is unsurprising as the weights retaining the policy function are likely to be substantially

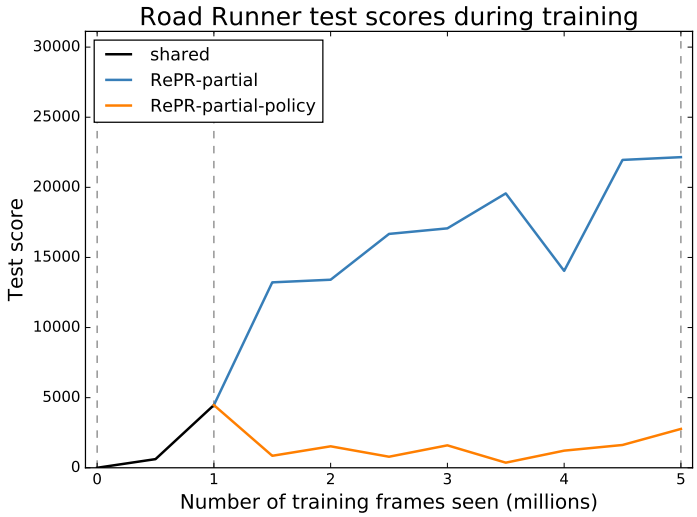

Figure 5.5: Results illustrating the scores attained while learning Road Runner in RePR's STM system. Results are shown for RePR when either the policy and value function is retained in the LTM system or just the policy function is retained. Scores are recorded by evaluating the short-term agent after every 500,000 observable training frames. Dashed lines indicate different learning intervals. In the first interval, Road Runner is learnt for the first time by a newly initialised network. In the second interval, the model is initialised to the long-term network's weights and learning on Road Runner is continued.

different to the weights which had initially learnt the Q-values in the short-term DQN. Therefore, reinitialising the short-term network with the majority of these weights would start the network at a poorer point for continuing learning than if those weights had been randomly initialised.

Surprisingly, there appears to be no difference between the *AC-RePR-partial* and *AC-RePR-partial-policy* conditions, where both conditions could successfully continue learning Road Runner to approximately the same standard. This means that in the *AC-RePR-partial-policy* condition, the STM system was able to quickly relearn the value function so that it could use it to further improve the network's policy. Overall, these results are important as they suggest that when using a reinforcement learning algorithm that separates the policy function from the value function, it is only necessary

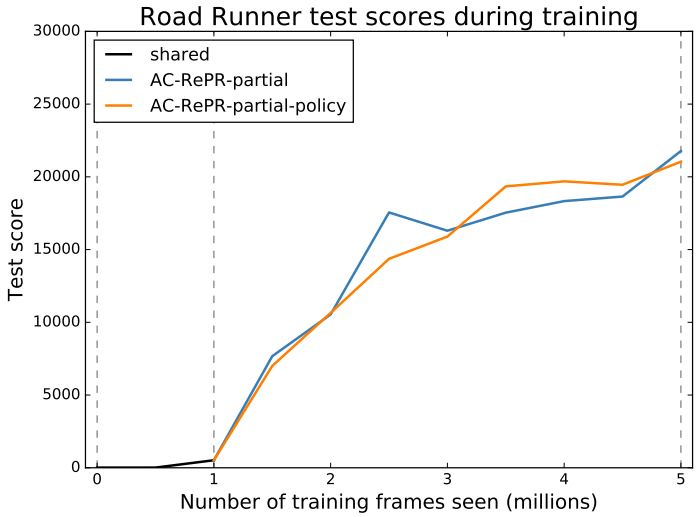

Figure 5.6: Results illustrating the scores attained while learning Road Runner in AC-RePR's STM system. Results are shown for AC-RePR when either the policy and value function is retained in the LTM system or just the policy function is retained. Scores are recorded by evaluating the short-term agent after every 500,000 observable training frames. Dashed lines indicate different learning intervals. In the first interval, Road Runner is learnt for the first time by a newly initialised network. In the second interval, the model is initialised to the long-term network's weights and learning on Road Runner is continued.

to retain the policy function in the LTM system.

In this experiment, the tasks were learnt with relatively large amounts of training data over relatively long training intervals (a minimum of 1m observable frames). O'Quinn, Silver, and Poirier (2005) found that continual learning, while repetitively switching between two tasks, was more difficult when learning with a large number of shorter intervals than the equivalent, smaller number of longer intervals. Therefore, future work could extend the methods in this section to analyse the models' ability to continue learning (with and without retaining the value function), while switching between tasks in shorter intervals.

5.2.3 Conclusions

In conclusion, this section has shown that continual learning methods which use reinforcement algorithms that do not separate the policy and value function should retain the joint function in the LTM system, so that it is possible for them to further learn previous tasks. This section has also provided evidence confirming that in continual learning methods which use reinforcement algorithms that do separate the policy and value function, it is only necessary to retain the policy function. This is because the value function can be quickly relearnt in the STM system so that the model is capable of continuing to learn a familiar task without disruption.

5.3 Prioritising Generating Important Features for Pseudo-Rehearsal

In the Atari 2600 games and many other environments, there are many features in the input that are relatively redundant to the agent when learning how to perform. For example, in Road Runner the repeating background pattern (road and desert with cacti) is not as important as the positions of the road runner, coyote, points and obstacles/dangers. Therefore, it is more important for the generator to accurately reproduce important features than less important ones. This section aims to improve the RePR algorithm by encouraging a GAN to produce pseudo-items designed specifically for retention. When the agent is struggling to retain previous tasks, these improved pseudo-items could provide more useful information about the previous task and thus, reduce catastrophic forgetting. Furthermore, this information might be beneficial to the generator because, when the GAN's capacity to effectively learn a new task has been exceeded, the generator can prioritise which features from already learnt tasks do not need to be reproduced accurately, freeing some units to learn the important features from the new task.

In sequential learning tasks, the goal is to retain the information learnt by the long-term agent. In RePR, the generator is only necessary to provide the agent with pseudo-data to help it retain this information. The generator is trained after the agent and therefore, the agent can be used to inform the generator which features in the data are important for it to reproduce.

The difficult question is; how can the agent inform the generator of important features? When the agent is taught a task, it learns to recognise only features important

to solving the task through stochastic gradient descent. This is achieved by changing the network's weights. Therefore, the activation patterns produced by these weights can be used to inform the generator how the agent solves the task and thus, can be used to encourage the generator to produce pseudo-items which give similar patterns.

When the generator learns with a reconstruction loss function (such as a Variational Auto-Encoder (Kingma and Welling, 2014)), the generator is directly learning to reproduce tasks' data. This means that each example from the training data can be fed through the agent and then a simple regulariser can be used so that the generator reconstructs examples that also produce similar activation patterns in the agent. However, when the generator is a GAN, this is not possible because training data is not used to update the weights of the generator. Instead, this section introduces a second discriminator into the GAN model which incorporates information from the agent into the generator.

5.3.1 Improving the Generative Model

This section proposes the model Generating Representations using Importance for Reinforcement-Pseudo-Rehearsal (GRIm-RePR) to improve the generator for continual learning. This model introduces a second discriminator into RePR. The input to this discriminator is the activation patterns from one of the early layers of the agent. These activation patterns are produced by passing real and generated examples through the agent's network and thus, the discriminator must solely use these activation patterns to distinguish between real and generated items. Figure 5.7 illustrates how the activation patterns from the agent's DQN are passed to the second discriminator.

The generative network in GRIm-RePR is updated so that it produces items which fool both of the discriminators. Fooling the second discriminator is given a much higher weighting in this loss function so that the generator is particularly encouraged to focus on reproducing items relevant to retention. However, initial results showed that giving some weight to the first discriminator was also important for producing realistic items.

The new loss function for the generator and the second discriminator are:

$$L_{gen} = -D(\widetilde{x};\phi) - \beta D(\widetilde{a};\Phi), \tag{5.7}$$

$$\begin{aligned} L_{disc_2} = D(\widetilde{a};\Phi) - D(a;\Phi) + \lambda(\|\nabla_{\hat{a}} D(\hat{a};\Phi)\|_2 - 1)^2 \\ +\epsilon_{drift} D(a;\Phi)^2 + \epsilon_{drift} D(\widetilde{a};\Phi)^2, \end{aligned} \tag{5.8}$$

where the weights of the second discriminator are Φ. A returns the activations from

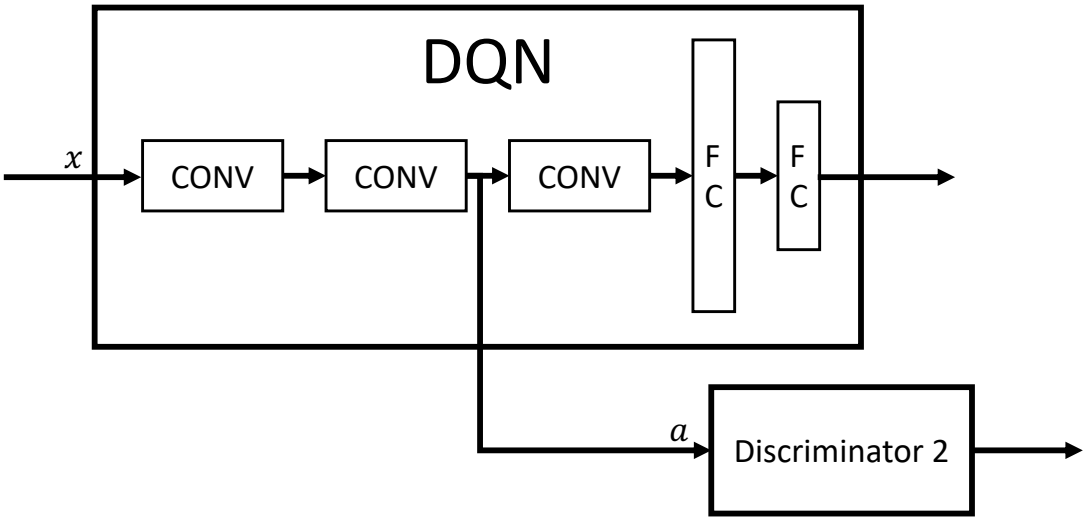

Figure 5.7: Illustration of how information (activation patterns) from the long-term DQN is given to the GAN's second discriminator in GRIm-RePR. CONV is a convolutional layer and FC is a fully connected layer.

the second layer of RePR's long-term DQN, which has the weights θ. Using this, $a = A(x;\theta)$, $\widetilde{a} = A(G(z;\varphi);\theta)$ and $\hat{a} = \epsilon a + (1-\epsilon)\widetilde{a}$. β is set to 1000 in this section's experiments. The weights of both the discriminators and the weights of the generator are updated on alternating steps.

In a neural network, the early layers will contain an abundance of low level features important to the task, with many other irrelevant features filtered out (Zeiler and Fergus, 2014; Yosinski, Clune, Nguyen, Fuchs, and Lipson, 2015). Therefore, it is the activations from the second layer which have been chosen to be fed to the second discriminator to help it generate useful items.

5.3.2 Related Work

Other methods that improve the generative capabilities of a model by providing it with additional information do exist. For example, conditional GANs (Mirza and Osindero, 2014) extend both the generator and discriminator to accommodate a class label. For the generator, this label informs the network which class it should be producing an image from. For the discriminator, this label informs the network which class the

image should be from. Therefore, the discriminator can learn to differentiate between different classes' images. An image which the discriminator thinks should be in a different class than it was told is likely to be fake, such that this information can assist it in categorising real and fake images. In a GAN, the discriminator competes against the generator and therefore, improving the discriminator by providing this additional information also improves the generator.

A modified Auxiliary Conditional GAN has been used to similarly improve GANs in continual image classification (Rios and Itti, 2019). This is where the classifier (continual learner) is incorporated inside of the GAN architecture. This results in the discriminator having $k + 1$ output units, where there are k classes being classified and an additional output representing whether the network believes an item is real or fake. One disadvantage of this method is that it has the discriminator share weights with the continual learning model. This can add unnecessary competition on the network to solve both classification and discrimination in one model. Furthermore, the generator in this model requires a class label to be sampled and then given as input. Given class labels do not exist in reinforcement learning and that the output of the agent cannot easily be sampled, this model cannot simply be applied to reinforcement learning.

In super-resolution, the resolution of an image is increased by passing it through a network. GANs have been particularly successful in this task. For this, the generator learns to reconstruct a high resolution image from a low resolution input. SRGAN (Ledig, Theis, Huszár, Caballero, Cunningham, Acosta, Aitken, Tejani, Totz, Wang, *et al.*, 2017) improves this GAN architecture by providing the generator with additional information. The generative network is updated using training samples and thus, the network can be regularised so that its reconstructed images produce similar features to their real super-resolution ones. The features are compared by passing both the reconstructed and real images through the VGG (Simonyan and Zisserman, 2015) network (a deep classification network pre-trained on ImageNet) and then minimising the Euclidean distance between their corresponding activation patterns. This additional constraint greatly improves the quality of the network's reconstructions. This idea has similarly been applied to the super-resolution of videos (Lucas, Katsaggelos, Lopez-Tapia, and Molina, 2018).

Hou, Shen, Sun, and Qiu (2017) has used a similar method in Variational Auto-Encoders (Kingma and Welling, 2014) to improve the quality of generated faces. This model uses a small number of latent variables to produce random faces representative of the CelebA dataset. Authors introduced a constraint so that their decoder/generator

was encouraged to produce similar VGG (Simonyan and Zisserman, 2015) activation patterns for reconstructed images than real ones, which consequently improved the generative quality of their model. However, in this method, and the above super-resolution ones, the generator is attempting to reconstruct real training examples. This is not the case for the GAN used in RePR and therefore, similar methods cannot be used to improve the GAN in RePR.

5.3.3 Methodology

The remainder of this section will experimentally investigate whether the GRIm-RePR model can outperform RePR. As per the previous chapter, all experiments train DQN agents and not Actor-Critics. The first experiment is identical to the previous chapter's first experiment (see Section 4.2), except: the task sequence is changed so that the models now learn Pong and then Boxing; and the GRIm-RePR model is introduced with its additional discriminator network (illustrated below). The second experiment improves retention by simply normalising the environments' Q-values that are being taught to the LTM system. When normalisation is used, RePR is found to completely retain Pong and thus, no difference between RePR and GRIm-RePR can be observed. This prompts a third experiment to demonstrate that the quality of the generations are still improved in GRIm-RePR when normalisation is used. This third experiment trains a freshly initialised agent with either pseudo-items generated from RePR's GAN or GRIm-RePR's GAN. Essentially, this experiment measures how much information is contained in the GAN's generations by having an agent learn the task from scratch. In this experiment, the generated pseudo-items have their target output labelled by a DQN agent which has already been taught the task. The GRIm-RePR model improves its generations by using the activations from the same DQN to provide information about which features are most important to generate correctly. To investigate whether these features remain consistent across DQNs, another condition is added where a separate DQN (pre-trained on the task) is used for labelling pseudo-items, such that it does not match (mismatch) the DQN whose activation patterns provide GRIm-RePR's improved GAN with importance information.

Environments

The experiments in this section utilise the environment Pong which is another Atari 2600 video game. The agent's goal in this game is to move its paddle so that it hits the

Table 5.4: The GAN's second discriminator network architecture for GRIm-RePR, where CONV is a convolutional layer and FC is a fully connected layer.

Discriminator 2				
Input: $9 \times 9 \times 64$				
layer	# units/filters	filter shape	filter stride	activation
CONV	64	5×5	1×1	Leaky ReLU
CONV	128	5×5	1×1	Leaky ReLU
CONV	256	5×5	1×1	Leaky ReLU
FC	1			

ball past the opponent's paddle. Every time the ball gets past the opponents paddle, the agent gets a point. Every time the ball gets past the agent's paddle, the opponent gets a point. This game is translated into a reinforcement learning environment through the same procedure described in Section 4.2.1.

Network Architectures

The only additional network architecture in this section's experiments is the second discriminator's network, which can be found in Table 5.4. Similar to the first discriminator, noise is added to the second discriminator's input. This involves applying the function $f(x) = \max(0, x + \mathcal{N}(0, 0.33\sigma))$ to each of the input unit's values, where σ is the standard deviation of the input unit across the current mini-batch.

Training and Evaluation

The first experiment trains and evaluates the models in the same way as the previous chapter's first experiment. The only exception is that the tasks learnt are Pong and then Boxing. Pong is introduced here as it appears to be a particularly difficult task to retain. Presumably, this is because the components important for the generator to learn to reproduce are relatively small (e.g. the ball).

The second experiment extends this by normalising the Q-values that the short-term DQN is teaching to the long-term DQN. The mean and standard deviation used in this normalisation is approximated using 1,000 batches from the experience replay,

which is passed through the STM system before training the LTM system.

The third experiment teaches a new agent to play either Pong or Road Runner using generated pseudo-items. This is achieved by firstly training a DQN to play the game (for 20m frames) while collecting data in an experience replay. This data, and in GRIm-RePR the DQN too, is then used to teach RePR or GRIm-RePR's generative model for 200,000 iterations, so that it can reproduce pseudo-items representative of the task. Next, the generative model is used to produce 200,000 pseudo-inputs, which are passed through the trained DQN so that their target Q-values are attained and normalised. Distillation is then used to teach these Q-values to a newly initialised DQN, so that it too can play the game.

Experimental Conditions

All of the experimental conditions are trained 3 times on the same set of seeds and results are averaged across these seeds. The first experiment's conditions are as follows:

- *RePR*: Learns Pong and Boxing sequentially with the RePR model, using the standard GAN architecture to generate pseudo-items for rehearsal. This condition does not use Q-value normalisation.
- *GRIm-RePR*: Learns Pong and Boxing sequentially with the GRIm-RePR model, utilising a second discriminator to improve the quality of the pseudo-items rehearsed. This condition does not use Q-value normalisation.
- *reh*: Learns Pong and Boxing sequentially, utilising real items for rehearsal. This condition does not use Q-value normalisation.

The second experiment's conditions are identical, except Q-value normalisation is used. Therefore, the conditions are labelled as *RePR-norm*, *GRIm-RePR-norm* and *reh-norm*. Finally, the third experiment investigates the quality of the RePR and GRIm-RePR models' generations by using their pseudo-items to train a task to a new DQN. The experimental conditions for this experiment are:

- *RePR-norm-scratch*: Learns an environment from scratch using pseudo-data generated by RePR's standard GAN, which has been trained on data from the task. This condition uses Q-value normalisation.
- *GRIm-RePR-norm-scratch*: Learns an environment from scratch using pseudo-data generated by GRIm-RePR's improved GAN, which has been trained on

data from the task. In this condition, the DQN agent used to inject additional information into the GAN is the same DQN agent used to teach the task to the new agent. This condition uses Q-value normalisation.

- *GRIm-RePR-norm-scratch-mismatch*: Learns an environment from scratch using pseudo-data generated by GRIm-RePR's improved GAN, which has been trained on data from the task. In this condition, two DQN agents are originally trained on the task. The first DQN is used to inject additional information into the GAN, while the second DQN is used to teach the task to the new agent. This condition uses Q-value normalisation.

5.3.4 Results and Discussion

The results of the first experiment are shown in Figure 5.8. When Boxing is being learnt, the *reh* condition shows some initial forgetting of Pong, but most of this is recovered through further rehearsal of Pong. Both the *RePR* and *GRIm-RePR* conditions demonstrate substantial initial forgetting of Pong. Both conditions gradually recover some of its ability to play the game, although the *GRIm-RePR* condition recovers to a much higher performance. This is important as it suggests that providing additional information to the generator has improved the generative capabilities of the GAN.

One interesting observation from the results was the initial forgetting present. This was hypothesised to be due to the tasks' Q-value functions interfering with one another. More specifically, the average Q-value in Pong was observed to be approximately 2, whereas in Boxing it was approximately 18. This results in Boxing being weighted as about 9 times more important in the agent's loss function[2] and thus, Pong is immediately forgotten when Boxing is first being learnt. This prompted the second experiment which investigated whether normalising the Q-values being taught to the LTM system stopped the initial forgetting seen in Pong.

The results for the second experiment are illustrated in Figure 5.9. The results conveyed no observable differences between conditions in both their retention of Pong and their capability to learn Boxing. However, having the importance of these tasks more evenly weighted appears to cost the network's ability to learn Boxing to the same

[2]For a two task sequence this could be solved by changing the α value weighting the importance between learning the new task and retention. However, for longer task sequences this solution is not possible.

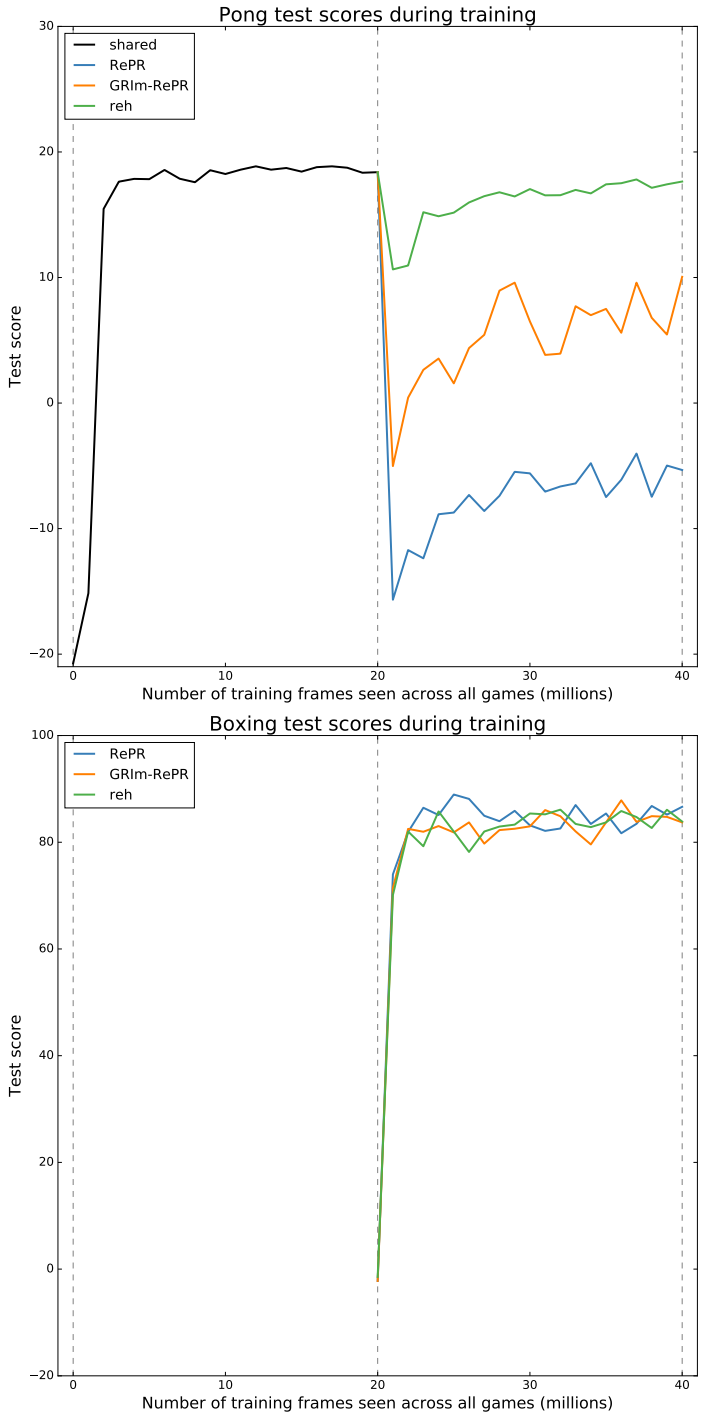

Figure 5.8: Results of the *GRIm-RePR* condition compared to the *RePR* and *reh* conditions, where all conditions do not use normalisation. Scores are recorded by evaluating the long-term DQN after every 1m observable training frames. Task switches occur at the dashed lines, in the order Pong and then Boxing.

standard as the previous experiment. Most interestingly, none of the 3 conditions had noticeably lower performance on Pong after learning the new task. Therefore, the differing Q-value functions of the tasks were causing Pong to be initially forgotten in the previous experiment. This shows that improving the generative model was not necessary to remember Pong. However, the previous results still suggest that the quality of GRIm-RePR's generations are likely better than RePR, as they could be used to relearn more forgotten knowledge of Pong. Therefore, it is likely that GRIm-RePR will be especially beneficial in situations where it is particularly challenging for the continual learner to retain knowledge.

Pilot tests also investigated whether interference between tasks could alternatively be minimised by standard normalising their reward functions. However, the mean and standard deviation of the rewards that the agent could eventually attain from the environment is not known before training. Therefore, this normalisation must be implemented by using the current mean and standard deviation of the rewards in the experience replay buffer. This mean and standard deviation can then be used to standard normalise the rewards sampled from the experience replay. However, as the agent's policy improves, it gets more rewards and thus, the mean and standard deviation of the experience replay buffer changes. This constant changing does not lend well to reinforcement learning, making some environments (e.g. Pong whose score could not be improved from its pre-initialised state) particularly more challenging to learn.

The third experiment more specifically investigates the quality of the models' generations by using them to teach a task from scratch. This experiment was conducted for both the Pong and Road Runner tasks and the results can be found in Figure 5.10 and Figure 5.11 respectively. For both of these tasks, the generative network used by the GRIm-RePR model can be observed to outperform RePR, being able to teach the task to a considerably higher standard on Pong.

One interesting observation made from the third experiment was that the *GRIm-RePR-norm-scratch-mismatch* condition did not perform as well as the *GRIm-RePR-norm-scratch* condition. More specifically, when the DQN used for improving the GAN did not match the DQN used for teaching, the new agent did not learn the task as well as when they did match. However, the *GRIm-RePR-norm-scratch-mismatch* condition did appear to more quickly learn the task compared to the *RePR-norm-scratch* condition, especially on Pong. It's surprising that the *GRIm-RePR-norm-scratch-mismatch* condition does not perform as well because this

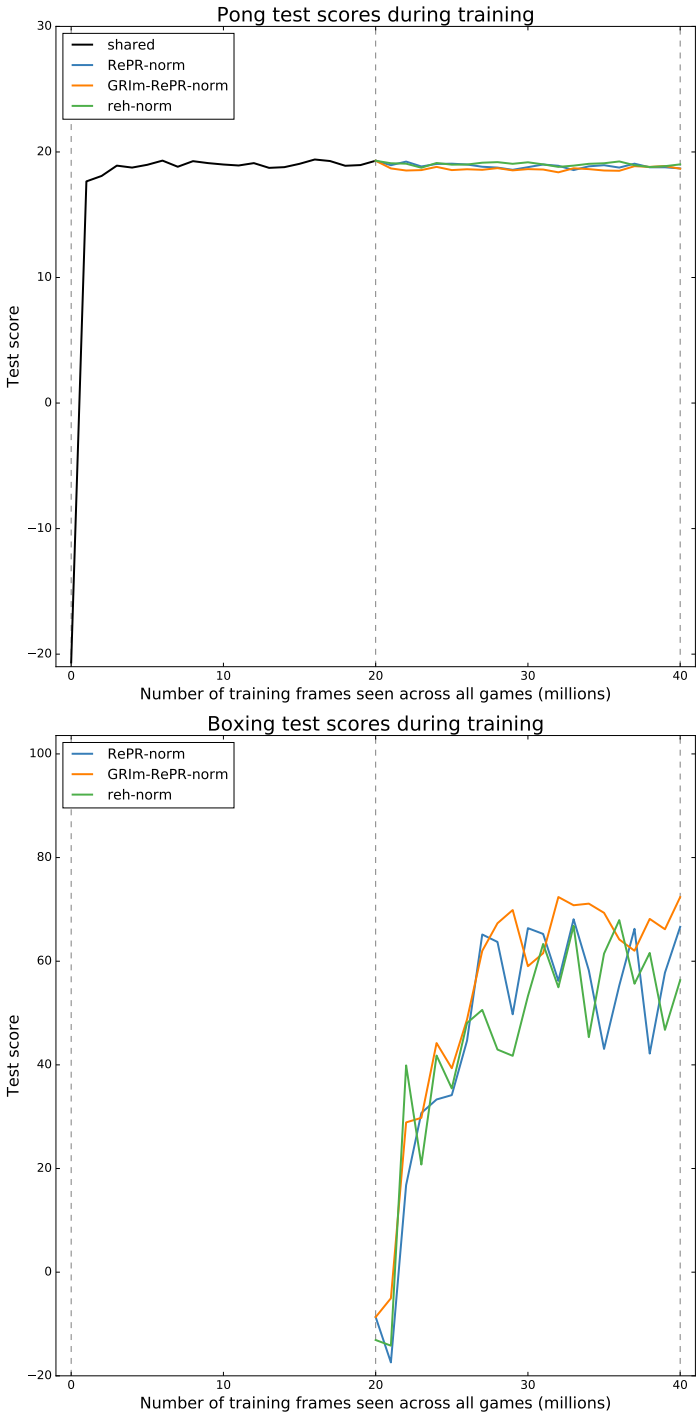

Figure 5.9: Results of the *GRIm-RePR-norm* condition compared to the *RePR-norm* and *reh-norm* conditions, where all conditions use normalisation. Scores are recorded by evaluating the long-term DQN after every 1m observable training frames. Task switches occur at the dashed lines, in the order Pong and then Boxing.

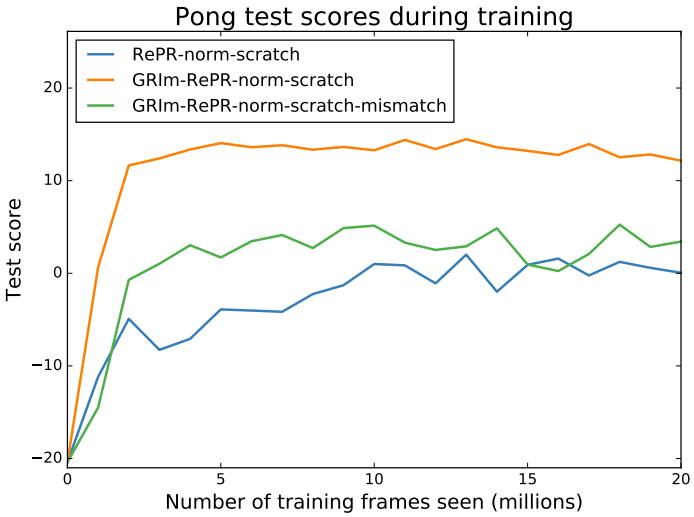

Figure 5.10: Results of RePR compared to GRIm-RePR (with the DQN used to train the new agent matching or mismatching the DQN used to train the GAN), where all conditions use normalisation. Scores are recorded by evaluating the new DQN on Pong after every 1m observable training frames.

suggests that the learnt features that are important to one DQN are different to the features that are important to another DQN trained to play the same game under the same training conditions. Although it is disappointing that the quality of generations does not improve regardless of whether the DQNs match, this is not a limitation when using GRIm-RePR for continual learning because the same DQN agent should be accessible when training the generative model, as well as when doing pseudo-rehearsal.

There is significant room for future work in improving generative models for pseudo-rehearsal. However, there was not enough time to investigate further in this thesis. Such future work could include extending the GRIm-RePR model by feeding more layers of information from the agent to the generative model, as well as investigating other ways this information could be calculated and given to the generator. One particular domain that could have interesting applications is network interpretability methods. These methods hope to indicate how important each network's input is to its output. When these methods improve, they could be particularly helpful in providing beneficial information to the generator.

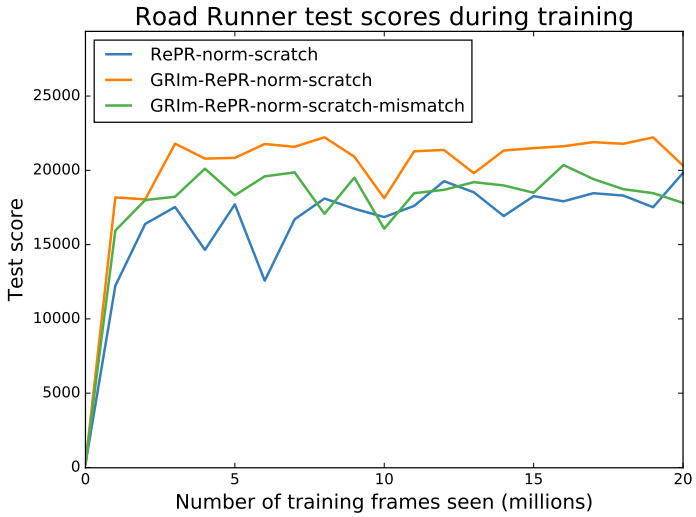

Figure 5.11: Results of RePR compared to GRIm-RePR (with the DQN used to train the new agent matching or mismatching the DQN used to train the GAN), where all conditions use normalisation. Scores are recorded by evaluating the new DQN on Road Runner after every 1m observable training frames.

5.3.5 Conclusions

In conclusion, this section proposed a method called GRIm-RePR which aims to improve the quality of pseudo-items. This is achieved by providing information from the agent to the generative model, referring to how important features in the input examples are to the agent's policy. Learning of important features is encouraged by the loss function of the network and thus, pseudo-items produced by the generative model are more effective for retention. Experimental results confirm that GRIm-RePR is capable of outperforming RePR, especially when forgetting occurs and relearning is necessary. Furthermore, normalising Q-values was also found to prevent forgetting by minimising the interference between tasks.

5.4 Evaluating Pseudo-Rehearsal on a Larger Task Sequence

So far, RePR and its variants have only been shown in relatively short, but still challenging, sequences of reinforcement learning tasks. GANs can be relatively unstable to train due to vanishing gradients and mode collapse (Li, Madry, Peebles, and Schmidt, 2018). Therefore, the RePR model has the potential to catastrophically fail once the model is under high load. This section challenges RePR further by extending the sequence of learning tasks to 6 Atari 2600 games. The aim is to investigate the limitations of RePR by more specifically investigating whether the model fails gracefully or catastrophically once the generative model has reached its capacity. It is also hypothesised that the advantages of the GRIm-RePR model over the RePR model will be more observable under these difficult conditions.

Furthermore, the Maximum Mean Discrepancy (MMD) (Gretton, Borgwardt, Rasch, Schölkopf, and Smola, 2012) value is computed to measure the similarity between the distribution of real input items and RePR's pseudo-items. MMD measures the similarity between two distributions by computing the distance between samples within and between the two distributions using kernels. The closer the MMD value is to zero[3], the more similarity there is between the distributions. More specifically, the MMD value is calculated by:

$$MMD^2 = \frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{j \neq i}^{m} k(x_i, x_j) + \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k(y_i, y_j) - \frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k(x_i, y_j), \tag{5.9}$$

where x and y are samples from the first and second distribution respectively. The number of samples used from each distribution are m and n respectively and $k(\cdot, \cdot)$ is a kernel (or sum of kernels).

5.4.1 Methodology

The first experiment in this section is identical to the experiment in the previous chapter (see Section 4.2), except: an extended task sequence is learnt; a larger short-term and long-term DQN is used; the long-term DQN is trained for longer, using normalised

[3]Negative MMD values are treated as zero.

Q-values; and learning of the new task is set to be slower. These changes are described in more detail below.

The second experiment compares the MMD value for batches of real items, RePR's pseudo-items and items generated by a normal distribution. An MMD value is calculated for each stage in the extended training sequence[4], where the GAN used to produce pseudo-items is the one trained in this section's first experiment (i.e. the *RePR-extend* condition below).

Environments

In the previous chapter, the environments learnt were Road Runner, Boxing and James Bond. This sequence is further extended using the following Atari 2600 video games: Pong (which has been described in the previous section), Atlantis and Qbert. All three of these tasks are ones in which a DQN can outperform a human (Mnih *et al.*, 2015). In Atlantis, the agent is required to protect Atlantis city from enemy space ships by choosing when to fire 3 stationary cannons at them. The ships move horizontally across the screen and if they survive 4 passes, they will destroy one of the city's bases (starting with the central cannon). The game ends when all bases have been destroyed by the invasion. In Qbert, the agent controls a character (called Qbert), who starts at the top of a pyramid of cubes and must jump diagonally around the cubes, changing them all to a certain colour to clear the level. Some levels require Qbert to jump on the cubes multiple times to get them to the correct colour. In later levels, the cubes will cycle through multiple colours when Qbert jumps on them, even once they have reached the target colour. There are also several enemies that impede Qbert, most importantly the enemy Coily which tries to catch Qbert and take a life from him. There are also floating platforms which Qbert can jump on to temporarily escape enemies and return to the top of the pyramid. These Atari games are translated into reinforcement learning environments through the same procedure described in Section 4.2.1.

Network Architectures

Pilot tests showed that the DQN architecture used in the previous chapter was too small to effectively retain the extended sequence of tasks, even when rehearsing with real data. Therefore, the DQN architecture used in this section was enlarged by doubling both the number of filters used in convolutional layers and the number of units in

[4]Results after learning the sixth task (Qbert) are excluded because these pseudo-items are not used to rehearse another task.

Table 5.5: Enlarged DQN architecture for training on an extended task sequence, where CONV is a convolutional layer and FC is a fully connected layer.

DQN				
Input: $84 \times 84 \times 4$				
layer	# units/filters	filter shape	filter stride	activation
CONV	64	8×8	4×4	ReLU
CONV	128	4×4	2×2	ReLU
CONV	128	3×3	1×1	ReLU
FC	1024			ReLU
FC	18			

the fully connected layer. This resulted in the architecture shown in Table 5.5. The architectures of the GANs used in RePR and GRIm-RePR remain the same as the architectures described in the models' corresponding sections.

Training and Evaluation

The training and evaluation procedure remains the same as the previous chapter, except for a few minor differences. The first difference being the Q-values transferred from the short-term DQN to the long-term DQN are normalised using the procedure described in Section 5.3.3. Training of the long-term DQN on each task is extended from 20m frames to 40m frames. Finally, assimilation of the new task into the LTM system is slowed down by setting α, scaling the importance of learning the new task compared to retaining previous tasks, to be 0.05.

Experimental Conditions

The first experiment's conditions are as follows:

- *std-extend*: Learns the extended sequence of environments, without using methods to prevent catastrophic forgetting. This is the lower bound of performance.
- *reh-extend*: Learns the extended sequence of environments, utilising real items for rehearsal. This is the upper bound of performance.

- *RePR-extend*: Learns the extended sequence of environments with the RePR model, using the standard GAN architecture to generate pseudo-items for rehearsal.

- *GRIm-RePR-extend*: Learns the extended sequence of environments with the GRIm-RePR model, utilising a second discriminator to improve the quality of the pseudo-items rehearsed.

Due to extensive training times, these conditions were only tested on a single, consistent seed.

The second experiment's conditions are as follows:

- *Real*: The MMD value when computing the similarity between two batches of real items drawn from all currently learnt tasks' experience replays. This condition provides a baseline for what the MMD value is for batches of data from the same distribution.

- *GAN*: The MMD value when computing the similarity between a batch of real items drawn from all currently learnt tasks' experience replays and a batch of pseudo-items produced by a GAN which has learnt the current sequence of tasks.

- *Norm*: The MMD value when computing the similarity between a batch of real items drawn from all currently learnt tasks' experience replays and a batch of items produced by a normal distribution. The normal distribution generates items by using the mean and standard deviation for each input variable, approximated over 20,000 states drawn from all currently learnt tasks' experience replays.

Each condition in the second experiment is repeated for 10 trials using a batch size of 100. A large number of 1 dimensional Gaussian Radial Basis Kernels were used with $\sigma^2 = [1 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}, 1, 5, 10, 15, 20, 25, 30, 35, 100, 1 \times 10^{3}, 1 \times 10^{4}, 1 \times 10^{5}, 1 \times 10^{6}]$.

5.4.2 Results and Discussion

The results of the first experiment are displayed in Figure 5.12. Consistent with previous experiments, the *std-extend* condition shows severe forgetting of all previously learnt tasks, where the final performance of all these tasks is similar to if no learning had occurred. This is except for the most recently learnt task, which is learnt to a

high performance level. The *reh-extend* condition displays some gradual forgetting of previously learnt tasks, with more recent tasks retained to a higher performance. The final rehearsal trained model scored 17180, 59, 653, 17, 55813 and 7960 on the learnt tasks respectively. Importantly, this result shows that the DQN model has the capability to successfully learn the sequence of tasks without dramatic forgetting.

When the RePR model was challenged, it showed varying amounts of forgetting. On Road Runner, forgetting became noticeable when the fourth task was being learnt, with performance drastically decreasing and then partially recovering to a score of around 18,000. However, once the fifth task was introduced, performance drastically decreased again, but this time the quality of the pseudo-items for this task was not high enough to recover from. While Qbert was being learnt, the *reh-extend* condition also suffered from drastic forgetting of Road Runner, but in this case, the network was able to partially recover from it. This suggests that Road Runner was particularly difficult to retain over this extended sequence, regardless of the condition. RePR conveyed less substantial forgetting on Boxing, with performance degrading over time until the task was essentially forgotten after all tasks were learnt. James Bond did not show forgetting until the fifth task, from which the network only gradually forgot some of its ability to perform the task. Pong was the only task in which substantial catastrophic forgetting could be immediately observed, with the agent's performance quickly dropping to around -5. In Atlantis, the RePR model displayed some gradual forgetting, losing less than half of its performance while successfully learning the final task Qbert.

Overall, the results of RePR showed that when the model is challenged beyond its capabilities, forgetting is usually gradual. That is, the model only partially forgets how to perform previously learnt tasks and thus, performance moderately decreases. However, this was not the case for all tasks, especially Pong which displayed very little retention. Although RePR displayed noticeable retention of previously learnt tasks, the model still underperformed compared to rehearsal with real data. This means that the generative model struggled to learn to produce data from new tasks, while retaining previous tasks.

Results suggest that the generator particularly struggled to produce items from certain games, mainly Road Runner and Pong. Images from both of these games were likely difficult for the GAN to effectively produce as small objects are present in those images. To attain a high score, it is extremely important for the agent to remember how to respond to these small objects. In Road Runner, these are the small circles on

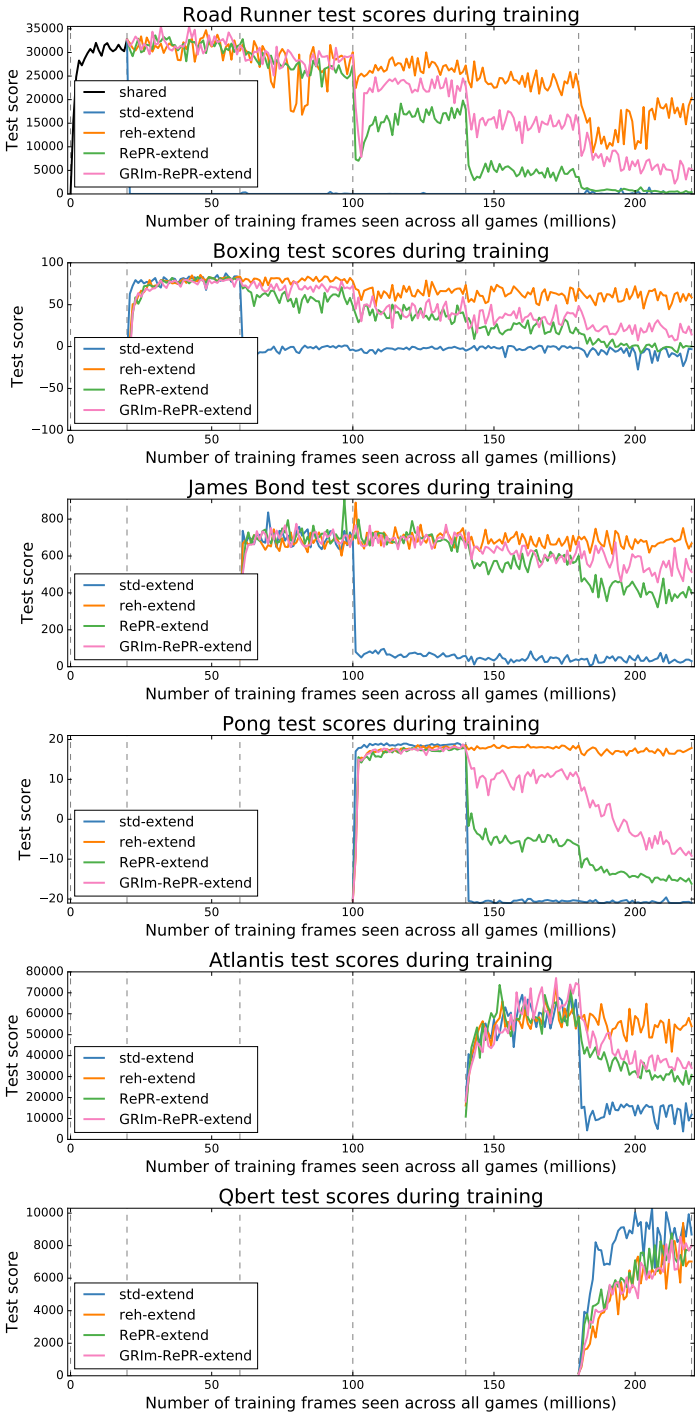

Figure 5.12: Results comparing the RePR and GRIm-RePR models to the *std-extend* and *reh-extend* conditions for an extended task sequence. Scores are recorded by evaluating the long-term agent after every 1m observable training frames. Task switches occur at the dashed lines, in the order Road Runner, Boxing, James Bond, Pong, Atlantis and then Qbert.

the ground which the Road Runner collects to attain reward. In Pong, this is the ball which the agent must move the paddle towards. These objects are more difficult for the generator to produce, due to them being small and difficult for the discriminator to detect. Therefore, the quality of these tasks' images for retention will be poorer causing more substantial forgetting in these tasks when the model is challenged.

The GRIm-RePR model outperformed the RePR model, showing only gradual forgetting when challenged beyond its capabilities. More specifically, the GRIm-RePR model retained noticeably higher performance on all previously learnt games compared to RePR, with the difference being most extreme in Road Runner and Pong. Overall, this result suggests that injecting extra information into the GAN does improve the quality of the images produced by the GRIm-RePR model and consequently increases its retention capabilities. However, the GRIm-RePR model still underperformed compared to the *reh-extend* condition, suggesting that further improvements to the GAN would lead to additional increases in the model's retention capabilities.

The generative model in RePR and GRIm-RePR uses pseudo-rehearsal to retain its ability to produce data from previous tasks. The more times pseudo-rehearsal is used on the GAN, the less realistic the data it produces. This is because the generator rehearses pseudo-items, not real data, and thus, errors build up as the generator learns from more and more abstracted data. This can explain why, on longer task sequences, the RePR variants demonstrate a steeper deterioration in performance compared to rehearsal of real items, which is much less noticeable when fewer tasks are learnt.

The results of the second experiment are displayed in Figure 5.13. This figure shows that the MMD values for the *Real* and *GAN* conditions are relatively similar, whereas the MMD value for the *Norm* condition is considerably higher. This suggests that the pseudo-items produced by the GAN are similar to the real distribution, particularly when compared to the similarity between items produced by a normal distribution and the real distribution. However, it is surprising that the MMD value for the *GAN* condition does not considerably increase with the number of tasks learnt. This contrasts with the results of the first experiment, which clearly showed that the GAN struggled to generate reasonable pseudo-items when increasing the number of tasks learnt. This suggests that the MMD value is a poor measure of similarity between these distributions and this is likely because the kernels are a poor measure of distance between items (especially when 1 dimensional kernels are used on flattened 3 dimensional items). The Inception score (Salimans *et al.*, 2016) was also considered as a measure of similarity (due to its prominence in GAN research). However, this score relies on real items being

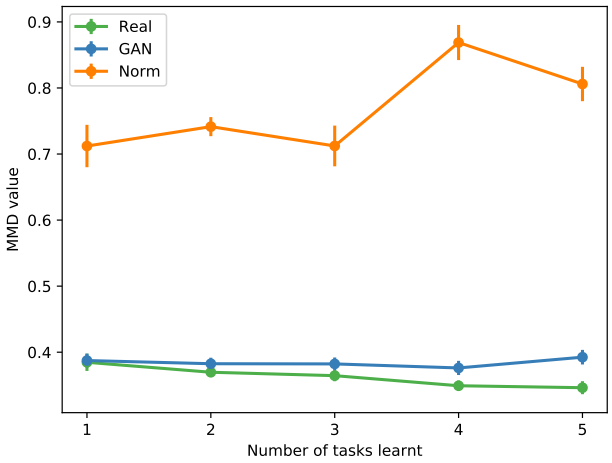

Figure 5.13: Results comparing the MMD values for the *Real*, *GAN* and *Norm* conditions at each stage of learning the extended task sequence. Error bars represent the standard deviation of each data point across the 10 trials.

evenly distributed into classes, which is not the case for the actions (classes) in these reinforcement learning tasks.

5.4.3 Conclusions

In conclusion, RePR was observed to gradually forget an extended sequence of tasks more steeply than when real data is used in rehearsal. This was believed to be due to the generative model using pseudo-rehearsal such that errors in generations compounded over time. Furthermore, substantial catastrophic forgetting was also present for RePR in tasks which were more difficult to generate items effective for retention. Alternatively, the GRIm-RePR model demonstrated less gradual forgetting of previously learnt tasks compared to the RePR model and furthermore, did not display any instances of severe catastrophic forgetting.

Chapter 6

Conclusion

A key quality of biological intelligence is its ability to do continual learning, where it can learn over time, integrating new knowledge, without catastrophically forgetting old knowledge. However, when artificial neural networks update to accomodate new knowledge, they have the tendency to completely overwrite previously attained knowledge and thus, catastrophically forget. This is disadvantageous to the model and restricts its ability to continually learn in a real world environment. Overcoming this catastrophic forgetting problem is important as it will allow neural networks to effectively build upon and improve its current knowledge. Furthermore, knowledge in one domain can often be applied to other domains. Therefore, overcoming this problem is also beneficial as knowledge from multiple domains could be compressed into a single model, which might also decrease the learning time for new information.

This thesis primarily aimed to solve the catastrophic forgetting problem in the reinforcement domain, where the neural network is challenged with a sequence of complex reinforcement tasks to learn. This is an important domain in which to solve the catastrophic forgetting problem because it is through reinforcement learning that a neural network has the capabilities to learn to positively interact in real world environments. Another important objective was for the method to prevent catastrophic forgetting without: increasing in memory size as the number of tasks expands; revisiting previously learnt tasks; or directly storing data from previous tasks.

Pseudo-rehearsal was identified as a promising solution to catastrophic forgetting, which this thesis built upon while satisfying the above objectives. Pseudo-rehearsal uses a random number generator to produce pseudo-inputs which can be passed through the model to attain target outputs. These input-output pairings (pseudo-items) can represent what the network has learnt from previous tasks. This means that pseudo-items

can be practiced alongside new tasks, so that new tasks are learnt without interfering with previously attained knowledge. Pseudo-rehearsal has major advantages over many other methods for preventing catastrophic forgetting because it constrains the overall function of the network to remain the same, without constraining the actual weights of the network. This gives the network the freedom to change dramatically and therefore, is less restrictive on how the network integrates new knowledge.

In more complex tasks, like those commonly learnt by deep neural networks, standard pseudo-rehearsal is no longer effective. This is because the pseudo-items do not represent previous tasks and consequently, they do not represent what the network has learnt about those tasks. Therefore, this thesis improves upon the ideas of pseudo-rehearsal, so that it can be used to prevent catastrophic forgetting in deep neural networks.

Initial work on improving pseudo-rehearsal was applied to the image classification domain. The method, termed Pseudo-Recursal, extended pseudo-rehearsal by introducing a generative network. This generative network learnt to randomly produce data, which was representative of previously learnt tasks, so that this data could be used as pseudo-items. These pseudo-items could then be used to retain knowledge of previous classification tasks. The pseudo-items could also be fed back to the generator so that it could learn to produce data representative of new tasks, without catastrophically forgetting how to produce data representative of previous tasks. Results showed the Pseudo-Recursal method successfully preventing catastrophic forgetting, outperforming the competing EWC methods and preforming similarly to when rehearsing real data.

Pseudo-rehearsal was further extended to the reinforcement domain. The model, termed RePR, split learning into two systems. The STM system contained a DQN which only learnt the current task through reinforcement learning. The LTM system contained a DQN and GAN, both of which were taught the new task by the STM system, while retaining previous knowledge through pseudo-rehearsal. The GAN was the generative model, which produced pseudo-items similar to Pseudo-Recursal. Isolating reinforcement learning to the STM system was found to be particularly beneficial because it made the training procedure considerably easier for the LTM system. Results from the RePR model demonstrated it was substantially more successful in preventing catastrophic forgetting compared to the EWC and the Progress and Compress method, along with performing very similarly to when the model rehearsed real items.

The later work in this thesis has further improved and evaluated the RePR model.

Firstly, the model was successfully generalised to Actor-Critic methods (another popular deep reinforcement learning method). This separated the policy from the value function, which consequently improved the policy retained by the model and meant the LTM system did not have to learn the value function to be able to continue learning from partially learnt tasks. GRIm-RePR was also proposed, which improved RePR by informing the generator about which features in images were important to the agent's policy. Finally, the RePR and GRIm-RePR models were evaluated on an extended sequence of tasks. Results confirmed that GRIm-RePR outperforms RePR but also demonstrated the models moderately suffering from forgetting on this extended sequence. However, forgetting was predominantly gradual, with some severe forgetting by the RePR model on tasks that were more difficult to generate important features for.

This thesis aimed to prevent catastrophic forgetting in neural networks with a model that used a consistent memory size (i.e. did not expand with the number of tasks learnt) and did not revisit previously learnt tasks nor directly store data from those tasks. This was achieved by introducing a generative network into previously established pseudo-rehearsal methods. The generative network learnt to generate data representative of previously seen tasks. This generated data was rehearsed so that previous tasks were not revisited and did not have raw data stored from them. Furthermore, the models proposed in this thesis did not scale as the number of tasks increased. More specifically, the size of the networks, number of networks and training time were all fixed in each experiment and thus, did not increase with the number of tasks currently learnt. Also, the number of pseudo-items generated and used in pseudo-rehearsal did not change throughout learning.

The goal of the generator is to produce items which are representative of the real dataset. Therefore, storing real items for rehearsal should always be more successful at preventing catastrophic forgetting. However, the proposed pseudo-rehearsal methods have advantages over rehearsal. Firstly, results in this thesis demonstrate that given a relatively small fixed memory allocation, it is more beneficial to use a generative model to produce pseudo-items than it is to directly store real items for rehearsal. Secondly, there might be legal/privacy reasons which do not allow the storage of real data (e.g. in the medical profession). Finally, it is implausible that the brain hardcodes real data to solve the catastrophic forgetting problem in human intelligence and therefore, the development of biologically plausible methods can be more intriguing.

RePR relies heavily on the capabilities of its generative network. Therefore, future

work improving generative models will be beneficial to pseudo-rehearsal and further minimising catastrophic forgetting. These improvements could not only enhance the overall quality of the data produced by these generative models, but also enhance the quality specific to pseudo-rehearsal. This latter improvement has been touched on by the GRIm-RePR model proposed in this thesis. However, there is still potential for an abundance of research investigating other ways to calculate and pass information to the generator so that it can be tailored specifically to pseudo-rehearsal.

Another avenue for future research is investigating whether pseudo-rehearsal can be used to solve the catastrophic forgetting that occurs when a reinforcement agent learns a single task. The stream of data a reinforcement agent is learning from is not i.i.d. and therefore, it is currently necessary to prevent catastrophic forgetting by using an experience replay or having multiple agents produce data by interacting with copies of the environment. Alternatively, future work might investigate whether using both pseudo-rehearsal and a generative model could also prevent this type of catastrophic forgetting.

In summary, this thesis has investigated how pseudo-rehearsal could be used to prevent catastrophic forgetting. This thesis found that introducing a generative model, along with other components, allowed pseudo-rehearsal to successfully prevent catastrophic forgetting in both the image classification and deep reinforcement learning domains. The proposed methods were found to out-compete state-of-the-art solutions. However, the methods are primarily limited by the capability of their generator and thus, if pseudo-rehearsal is to completely solve the catastrophic forgetting problem, future research into improving this generator is necessary.

References

Abraham, W. C., Jones, O. D., and Glanzman, D. L. (2019). Is Plasticity of Synapses the Mechanism of Long-Term Memory Storage? *npj Science of Learning*, *4*(1). https://doi.org/10.1038/s41539-019-0048-y.

Abraham, W. C. and Robins, A. (2005). Memory Retention - The Synaptic Stability Versus Plasticity Dilemma. *Trends in Neurosciences*, *28*(2), 73–78.

Ans, B. and Rousset, S. (1997). Avoiding Catastrophic Forgetting by Coupling Two Reverberating Neural Networks. *Comptes Rendus de l'Académie des Sciences - Series III - Sciences de la Vie*, *320*(12), 989–997.

Ans, B. and Rousset, S. (2000). Neural Networks with a Self-Refreshing Memory: Knowledge Transfer in Sequential Learning Tasks without Catastrophic Forgetting. *Connection Science*, *12*(1), 1–19.

Arjovsky, M. and Bottou, L. (2017). Towards Principled Methods for Training Generative Adversarial Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Hk4_qw5xe.

Atkinson, C., McCane, B., Szymanski, L., and Robins, A. (2018a). Pseudo-Recursal: Solving the Catastrophic Forgetting Problem in Deep Neural Networks. *arXiv preprint arXiv:1802.03875*.

Atkinson, C., McCane, B., Szymanski, L., and Robins, A. (2018b). Pseudo-Rehearsal: Achieving Deep Reinforcement Learning without Catastrophic Forgetting. *arXiv preprint arXiv:1812.02464*.

Atkinson, C., McCane, B., Szymanski, L., and Robins, A. (2019). GRIm-RePR: Prioritising Generating Important Features for Pseudo-Rehearsal. *arXiv preprint arXiv:1911.11988*.

Baddeley, B. (2008). Reinforcement Learning in Continuous Time and Space: Interference and Not Ill Conditioning is the Main Problem when Using Distributed Function Approximators. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *38*(4), 950–956.

Bellman, R. (1966). Dynamic Programming. *Science*, *153*(3731), 34–37.

Berseth, G., Xie, C., Cernek, P., and Van de Panne, M. (2018). Progressive Reinforcement Learning with Distillation for Multi-Skilled Motion Control. In *International Conference on Learning Representations*. `https://openreview.net/forum?id=B13njo1R-`.

Carpenter, G. A. and Grossberg, S. (1988). The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network. *Computer*, *21*(3), 77–88.

Caselles-Dupré, H., Garcia-Ortiz, M., and Filliat, D. (2018). Continual State Representation Learning for Reinforcement Learning Using Generative Replay. *arXiv preprint arXiv:1810.03880*.

Caselles-Dupré, H., Garcia-Ortiz, M., and Filliat, D. (2019). S-TRIGGER: Continual State Representation Learning via Self-Triggered Generative Replay. *arXiv preprint arXiv:1902.09434*.

Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. S. (2018). Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence. In *European Conference on Computer Vision*, 556–572. Springer International Publishing.

Coop, R., Mishtal, A., and Arel, I. (2013). Ensemble Learning in Fixed Expansion Layer Networks for Mitigating Catastrophic Forgetting. *IEEE Transactions on Neural Networks and Learning Systems*, *24*(10), 1623–1634.

Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, *2*(4), 303–314.

Draelos, T. J., Miner, N. E., Lamb, C. C., Cox, J. A., Vineyard, C. M., Carlson, K. D., Severa, W. M., James, C. D., and Aimone, J. B. (2017). Neurogenesis Deep Learning: Extending Deep Networks to Accommodate New Classes. In *International Joint Conference on Neural Networks*, 526–533.

Fowler, B. and Silver, D. L. (2011). Consolidation Using Context-Sensitive Multiple Task Learning. In *Advances in Artificial Intelligence*, 128–139. Springer Berlin Heidelberg.

Frean, M. R. and Robins, A. (1999). Catastrophic Forgetting in Simple Networks: An Analysis of the Pseudorehearsal Solution. *Network: Computation in Neural Systems*, *10*(3), 227–236.

French, R. M. (1997). Pseudo-Recurrent Connectionist Networks: An Approach to the 'Sensitivity-Stability' Dilemma. *Connection Science*, *9*(4), 353–380.

French, R. M., Ans, B., and Rousset, S. (2001). Pseudopatterns and Dual-Network Memory Models: Advantages and Shortcomings. In *Connectionist Models of Learning, Development and Evolution*, 13–22. Springer London.

Gais, S., Albouy, G., Boly, M., Dang-Vu, T. T., Darsaud, A., Desseilles, M., Rauchs, G., Schabus, M., Sterpenich, V., Vandewalle, G., *et al.* (2007). Sleep Transforms the Cerebral Trace of Declarative Memories. *Proceedings of the National Academy of Sciences*, *104*(47), 18778–18783.

Gallistel, C. R. and Matzel, L. D. (2013). The Neuroscience of Learning: Beyond the Hebbian Synapse. *Annual Review of Psychology*, *64*(1), 169–200.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, Volume 15, 315–323.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. (2014). An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks. In *International Conference on Learning Representations*. `https://openreview.net/forum?id=oXSw7laxwUpln`.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, Volume 27, 2672–2680. Curran Associates, Inc.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A Kernel Two-Sample Test. *Journal of Machine Learning Research*, *13*(25), 723–773.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved Training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*, Volume 30, 5767–5777. Curran Associates, Inc.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 770–778.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*.

Hong, D., Li, Y., and Shin, B.-S. (2019). Predictive EWC: Mitigating Catastrophic Forgetting of Neural Network through Pre-Prediction of Learning Data. *Journal of Ambient Intelligence and Humanized Computing*. `https://doi.org/10.1007/s12652-019-01346-7`.

Hou, X., Shen, L., Sun, K., and Qiu, G. (2017). Deep Feature Consistent Variational Autoencoder. In *IEEE Winter Conference on Applications of Computer Vision*, 1133–1141.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, Volume 37 of *Proceedings of Machine Learning Research*, 448–456. PMLR.

Isele, D. and Cosgun, A. (2018). Selective Experience Replay for Lifelong Learning. In *AAAI Conference on Artificial Intelligence*, 3302–3309.

Jung, H., Ju, J., Jung, M., and Kim, J. (2018). Less-Forgetful Learning for Domain Expansion in Deep Neural Networks. In *AAAI Conference on Artificial Intelligence*, 3358–3365.

Kamra, N., Gupta, U., and Liu, Y. (2017). Deep Generative Dual Memory Network for Continual Learning. *arXiv preprint arXiv:1710.10368*.

Kaplanis, C., Shanahan, M., and Clopath, C. (2018). Continual Reinforcement Learning with Complex Synapses. In *Proceedings of the 35th International Conference on Machine Learning*, Volume 80 of *Proceedings of Machine Learning Research*, 2497–2506. PMLR.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018). Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Hk99zCeAb.

Kemker, R. and Kanan, C. (2018). FearNet: Brain-Inspired Model for Incremental Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJ1Xmf-Rb.

Kemker, R., McClure, M., Abitino, A., Hayes, T. L., and Kanan, C. (2018). Measuring Catastrophic Forgetting in Neural Networks. In *AAAI Conference on Artificial Intelligence*, 3390–3398.

Ketz, N., Kolouri, S., and Pilly, P. (2019). Continual Learning Using World Models for Pseudo-Rehearsal. *arXiv preprint arXiv:1903.02647*.

Kim, D., Bae, J., Jo, Y., and Choi, J. (2019). Incremental Learning with Maximum Entropy Regularization: Rethinking Forgetting and Intransigence. *arXiv preprint arXiv:1902.00829*.

Kim, H.-E., Kim, S., and Lee, J. (2018). Keep and Learn: Continual Learning by Constraining the Latent Space for Knowledge Preservation in Neural Networks. In *Medical Image Computing and Computer Assisted Intervention*, 520–528. Springer International Publishing.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*. https://openreview.net/forum?id=33X9fd2-9FyZd.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., *et al.* (2017). Overcoming Catastrophic Forgetting in Neural Networks. *Proceedings of the National Academy of Sciences*, *114*(13), 3521–3526.

Kobayashi, T. (2018). Check Regularization: Combining Modularity and Elasticity for Memory Consolidation. In *Artificial Neural Networks and Machine Learning*, 315–325. Springer International Publishing.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, Volume 25, 1097–1105. Curran Associates, Inc.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeCun, Y., Cortes, C., and Burges, C. J. C. (1998). The MNIST Database of Handwritten Digits. `http://yann.lecun.com/exdb/mnist/`.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., *et al.* (2017). Photo-Realistic Single Image Super-Resolution using a Generative Adversarial Network. In *IEEE Conference on Computer Vision and Pattern Recognition*, 105–114.

Legg, S. and Hutter, M. (2007). Universal Intelligence: A Definition of Machine Intelligence. *Minds and Machines*, *17*(4), 391–444.

Li, J., Madry, A., Peebles, J., and Schmidt, L. (2018). On the Limitations of First-Order Approximation in GAN Dynamics. In *Proceedings of the 35th International Conference on Machine Learning*, Volume 80 of *Proceedings of Machine Learning Research*, 3005–3013. PMLR.

Li, Z. and Hoiem, D. (2018). Learning without Forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(12), 2935–2947.

Lopez-Paz, D. and Ranzato, M. (2017). Gradient Episodic Memory for Continual Learning. In *Advances in Neural Information Processing Systems*, Volume 30, 6467–6476. Curran Associates, Inc.

Louie, K. and Wilson, M. A. (2001). Temporally Structured Replay of Awake Hippocampal Ensemble Activity During Rapid Eye Movement Sleep. *Neuron*, *29*(1), 145–156.

Lucas, A., Katsaggelos, A. K., Lopez-Tapia, S., and Molina, R. (2018). Generative Adversarial Networks and Perceptual Losses for Video Super-Resolution. In *25th IEEE International Conference on Image Processing*, 51–55.

Marochko, V., Johard, L., and Mazzara, M. (2017). Pseudorehearsal in Value Function Approximation. In *Agent and Multi-Agent Systems: Technology and Applications*, 178–189. Springer International Publishing.

Marochko, V., Johard, L., Mazzara, M., and Longo, L. (2018). Pseudorehearsal in Actor-Critic Agents with Neural Network Function Approximation. In *IEEE 32nd*

International Conference on Advanced Information Networking and Applications, 644–650.

McClelland, J. L., McNaughton, B. L., and O'Reilly, R. C. (1995). 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*(3), 419–457.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In *Psychology of Learning and Motivation*, Volume 24, 109–165. Elsevier.

Mellado, D., Saavedra, C., Chabert, S., and Salas, R. (2017). Pseudorehearsal Approach for Incremental Learning of Deep Convolutional Neural Networks. In *Computational Neuroscience*, 118–126. Springer International Publishing.

Mirza, M. and Osindero, S. (2014). Conditional Generative Adversarial Nets. *arXiv preprint arXiv:1411.1784*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning*, Volume 48 of *Proceedings of Machine Learning Research*, 1928–1937. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., *et al.* (2015). Human-Level Control through Deep Reinforcement Learning. *Nature*, *518*(7540), 529–533.

O'Quinn, R. J., Silver, D. L., and Poirier, R. (2005). Continued Practice and Consolidation of a Learning Task. In *International Conference on Machine Learning Workshop on Meta-Learning*.

Pallier, C., Dehaene, S., Poline, J.-B., LeBihan, D., Argenti, A.-M., Dupoux, E., and Mehler, J. (2003). Brain Imaging of Language Plasticity in Adopted Adults: Can a Second Language Replace the First? *Cerebral Cortex*, *13*(2), 155–161.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). Continual Lifelong Learning with Neural Networks: A Review. *Neural Networks*, *113*, 54–71.

Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2016). Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=4zXscyOrI_m.

Pascanu, R. and Bengio, Y. (2014). Revisiting Natural Gradient for Deep Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=vz8AumxkAfz5U.

Poirier, R. and Silver, D. L. (2005). Effect of Curriculum on the Consolidation of Neural Network Task Knowledge. In *IEEE International Joint Conference on Neural Networks*, Volume 4, 2123–2128.

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *International Conference on Learning Representations*. https://arxiv.org/abs/1511.06434.

Ratcliff, R. (1990). Connectionist Models of Recognition Memory: Constraints Imposed by Learning and Forgetting Functions. *Psychological Review*, *97*(2), 285–308.

Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., and Tesauro, G. (2019). Learning to Learn without Forgetting by Maximizing Transfer and Minimizing Interference. In *International Conference on Learning Representations*. https://openreview.net/forum?id=B1gTShAct7.

Rios, A. and Itti, L. (2019). Closed-Loop Memory GAN for Continual Learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 3332–3338. International Joint Conferences on Artificial Intelligence Organization.

Robins, A. (1995). Catastrophic Forgetting, Rehearsal and Pseudorehearsal. *Connection Science*, *7*(2), 123–146.

Robins, A. (1997). Maintaining Stability during New Learning in Neural Networks. In *IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, Volume 4, 3013–3018.

Robins, A. and Frean, M. R. (1998). Local Learning Algorithms for Sequential Tasks in Neural Networks. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, *2*(6), 221–227.

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., and Wayne, G. (2019). Experience Replay for Continual Learning. In *Advances in Neural Information Processing Systems*, Volume 32, 350–360. Curran Associates, Inc.

Rosenblatt, F. (1957). *The Perceptron, a Perceiving and Recognizing Automaton (Project Para)*. Cornell Aeronautical Laboratory. `https://books.google.co.nz/books?id=P_XGPgAACAAJ`.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Representations by Back-Propagating Errors. *Nature*, *323*, 533–536.

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2016). Policy Distillation. In *International Conference on Learning Representations*. `https://arxiv.org/abs/1511.06295`.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive Neural Networks. *arXiv preprint arXiv:1606.04671*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016). Improved Techniques for Training GANs. In *Advances in Neural Information Processing Systems*, Volume 29, 2234–2242. Curran Associates, Inc.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized Experience Replay. In *International Conference on Learning Representations*. `https://arxiv.org/abs/1511.05952`.

Schwarz, J., Luketina, J., Czarnecki, W. M., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2018). Progress & Compress: A Scalable Framework for Continual Learning. In *Proceedings of the 35th International Conference on Machine Learning*, Volume 80 of *Proceedings of Machine Learning Research*, 4528–4537. PMLR.

Shin, H., Lee, J. K., Kim, J., and Kim, J. (2017). Continual Learning with Deep Generative Replay. In *Advances in Neural Information Processing Systems*, Volume 30, 2990–2999. Curran Associates, Inc.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., *et al.* (2016).

Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, *529*(7587), 484–489.

Silver, D. L. and Mahfuz, S. (2020). Generating Accurate Pseudo Examples for Continual Learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 1035–1042.

Silver, D. L., Mason, G., and Eljabu, L. (2015). Consolidation Using Sweep Task Rehearsal: Overcoming the Stability-Plasticity Problem. In *Advances in Artificial Intelligence*, 307–322. Springer International Publishing.

Silver, D. L. and Mercer, R. E. (2002). The Task Rehearsal Method of Life-Long Learning: Overcoming Impoverished Data. In *Advances in Artificial Intelligence*, 90–101. Springer Berlin Heidelberg.

Silver, D. L. and Poirier, R. (2004). Sequential Consolidation of Learned Task Knowledge. In *Advances in Artificial Intelligence*, 217–232. Springer Berlin Heidelberg.

Silver, D. L., Yang, Q., and Li, L. (2013). Lifelong Machine Learning Systems: Beyond Learning Algorithms. In *AAAI Spring Symposium Series*, 49–55.

Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*. `https://arxiv.org/abs/1409.1556`.

Sorrells, S. F., Paredes, M. F., Cebrian-Silla, A., Sandoval, K., Qi, D., Kelley, K. W., James, D., Mayer, S., Chang, J., Auguste, K. I., *et al.* (2018). Human Hippocampal Neurogenesis Drops Sharply in Children to Undetectable Levels in Adults. *Nature*, *555*(7696), 377–381.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958.

Sutton, R. S. and Barto, A. G. (2017). Reinforcement Learning: An Introduction (2nd Edition). complete draft.

Thorndike, E. L. (1898). Animal Intelligence: An Experimental Study of the Associative Processes in Animals. *The Psychological Review: Monograph Supplements*, *2*(4). `https://doi.org/10.1037/h0092987`.

Thrun, S. and Mitchell, T. M. (1995). Lifelong Robot Learning. *Robotics and Autonomous Systems*, *15*(1), 25–46.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. In *AAAI Conference on Artificial Intelligence*, 2094–2100.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning*, Volume 48 of *Proceedings of Machine Learning Research*, 1995–2003. PMLR.

Wen, S. and Itti, L. (2019). Beneficial Perturbation Network for Continual Learning. *arXiv preprint arXiv:1906.10528*.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. (2015). Understanding Neural Networks through Deep Visualization. *arXiv preprint arXiv:1506.06579*.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In *Computer Vision*, 818–833. Springer International Publishing.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual Learning through Synaptic Intelligence. In *Proceedings of the 34th International Conference on Machine Learning*, Volume 70 of *Proceedings of Machine Learning Research*, 3987–3995. PMLR.

Zhang, J., Zhang, J., Ghosh, S., Li, D., Tasci, S., Heck, L., Zhang, H., and Kuo, C.-C. J. (2020). Class-Incremental Learning via Deep Model Consolidation. In *IEEE Winter Conference on Applications of Computer Vision*, 1120–1129.

Appendix A

Pseudocode for Pseudo-Recursal

```
foreach d in datasets do
    net = train_net(net, d, gan)
    gan = train_gan(gan, d)
end

def train_net(net, d, gan):
    new_net = copy(net)
    while new_net is not converged do
        update_new_net(new_net, d, gan, net)
    end
    return new_net
end

def update_new_net(new_net, d, gan, net):
    x, y = d.sample(batchsize)
    L_N = CE(new_net.get_outputs(x), y)
    if gan is not initialised then
        loss = L_N
    else
        z = uniform_sample(−1, 1, (batchsize, n_latents))
        prev_x = gan.gen.get_outputs(z)
        prev_y = net.get_outputs(prev_x)
        L_PRec = CE(new_net.get_outputs(prev_x), prev_y)
        loss = L_N + L_PRec
    end
    new_net.SGD_step(loss)
end
```

```
def train_gan(gan, d):
    new_gan = initialise_gan()
    new_gan.n_tasks = 1
    if gan is initialised then
        new_gan.n_tasks = gan.n_tasks
    end
    while new_gan is not converged do
        update_new_gan(new_gan, d, gan)
    end
    new_gan.n_tasks = new_gan.n_tasks + 1
    return new_gan
end

def update_new_gan(new_gan, d, gan):
    x, y = d.sample(batchsize/new_gan.n_tasks)
    if new_gan.n_tasks > 1 then
        z = uniform_sample(−1, 1, (batchsize − (batchsize/new_gan.n_tasks),
         n_latents))
        prev_x = gan.gen.get_outputs(z)
        x = concatenate(x, prev_x)
    end
    z = uniform_sample(−1, 1, (batchsize, n_latents))
    L_DiscPRec = CE(new_gan.disc.get_outputs(new_gan.gen.get_outputs(z)),
     zeros) + CE(new_gan.disc.get_outputs(x), ones)
    new_gan.disc.SGD_step(L_DiscPRec)
    for i in range(2) do
        z = uniform_sample(−1, 1, (batchsize, n_latents))
        L_GenPRec = CE(new_gan.disc.get_outputs(new_gan.gen.get_outputs(z)),
         ones)
        new_gan.gen.SGD_step(L_GenPRec)
    end
end
```

Algorithm 1: Pseudocode for training the Pseudo-Recursal model. *datasets* is a list of the datasets to be learnt sequentially. *net*, *gan*.disc and *gan*.gen are the classifier, discriminator and generator networks respectively. *zeros* and *ones* are arrays of length *batchsize* containing either zeros or ones respectively. In practice, the algorithm used for collecting Pseudo-Recursal's results stores an array of pseudo-items which are generated by the GAN before training on a new task rather than generating them on the fly during training.

Appendix B

Pseudocode for RePR

```
foreach env in environments do
    stm_agent = initialise_stm_agent()
    stm_agent.target_net = copy(stm_agent.pred_net)
    stm_agent = train_agent(stm_agent, env, exp, stm_max_iter, None, None)
    if ltm_agent is not initialised then
        ltm_agent = copy(stm_agent)
    else
        ltm_agent = train_agent(stm_agent, env, exp, ltm_max_iter, ltm_agent,
          gan)
    end
    gan = train_gan(gan, exp, gan_max_iter)
end

def train_gan(gan, exp, gan_max_iter):
    new_gan = initialise_gan()
    new_gan.n_tasks = 1
    if gan is initialised then
        new_gan.n_tasks = gan.n_tasks
    end
    iter = 0
    while iter < gan_max_iter do
        if iter is even then
            update_disc(new_gan, exp, gan)
        else
            update_gen(new_gan)
        end
        iter = iter + 1
    end
    new_gan.n_tasks = new_gan.n_tasks + 1
    return new_gan
end
```

```
def train_agent(stm_agent, env, exp, max_iter, ltm_agent, gan):
    if ltm_agent is None then
        agent = stm_agent
    else
        agent = ltm_agent
        prev_ltm_agent = copy(ltm_agent)
    end
    exp.clear()
    s_t = env.reset()
    iter = 0
    while iter < max_iter do
        a_t = agent.pred_net.predict_action(s_t)
        r_t, d_t, s_t = env.take_action(a_t)
        exp.add(a_t, r_t, d_t, s_t)
        if ltm_agent is None then
            update_stm_agent(stm_agent, exp, iter)
        else
            update_ltm_agent(ltm_agent, exp, stm_agent, gan, prev_ltm_agent)
        end
        if d_t then
            s_t = env.reset()
        end
        iter = iter + 1
    end
    return agent
end
```

def update_stm_agent(stm_agent, exp, $iter$)**:**
    a, r, d, s, $s^{'}$ = exp.sample($batchsize$)
    $loss = 0$
    foreach a_t, r_t, d_t, s_t, s_{t+1} **in** zip(a, r, d, s, $s^{'}$) **do**
        if d_t **then**
            $y_t = r_t$
        else
            $y_t = r_t + \gamma \max_{a_{t+1}} stm\_agent$.target_net.get_outputs($s_{t+1}$)[$a_{t+1}$]
        end
        $loss = loss + (y_t - stm\_agent$.pred_net.get_outputs($s_t$)[$a_t$])^2
    end
    stm_agent.pred_agent.SGD_step($loss$)
    if $iter$ **divisible by** $update\_target\_freq$ **then**
        stm_agent.target_net = copy(stm_agent.pred_net)
    end
end

def update_ltm_agent(ltm_agent, exp, stm_agent, gan, $prev\_ltm\_agent$)**:**
    a, r, d, s, $s^{'}$ = exp.sample($batchsize$)
    z = uniform_sample(-1, 1, ($batchsize$, $n\_latents$))
    $\widetilde{s} = gan$.gen.get_outputs(z)
    $loss = 0$
    foreach s_j, $\widetilde{s}_j$ **in** zip(s, $\widetilde{s}$) **do**
        L_D = sum((ltm_agent.pred_net.get_outputs(s_j) − stm_agent.pred_net.get_outputs(s_j))^2)
        L_{RePR} = sum((ltm_agent.pred_net.get_outputs($\widetilde{s}_j$) − $prev\_ltm\_agent$.pred_net.get_outputs($\widetilde{s}_j$))^2)
        $loss = loss + \alpha L_D + (1 - \alpha) L_{RePR}$
    end
    ltm_agent.pred_agent.SGD_step($loss$)
end

def update_disc(new_gan, exp, gan)**:**
    a, r, d, s, $s^{'}$ = exp.sample($batchsize/new\_gan$.n_tasks)
    $x = s$
    if new_gan.n_tasks > 1 **then**
        z = uniform_sample(-1, 1, ($batchsize - (batchsize/new\_gan$.n_tasks), $n\_latents$))
        $\widetilde{x}$ = gan.gen.get_outputs(z)
        x = concatenate(x, $\widetilde{x}$)
    end
    z = uniform_sample(-1, 1, ($batchsize$, $n\_latents$))
    ϵ = uniform_sample(0, 1, $batchsize$)
    $loss = 0$
    foreach x_j, z_j, ϵ_j **in** zip(x, z, ϵ) **do**
        $\widetilde{x}_j$ = new_gan.gen.get_outputs(z_j)
        $\hat{x}_j = \epsilon_j x_j + (1 - \epsilon_j)\widetilde{x}_j$
        $disc\_real$ = new_gan.disc.get_outputs(x_j)
        $disc\_fake$ = new_gan.disc.get_outputs($\widetilde{x}_j$)
        $disc\_xhat$ = new_gan.disc.get_outputs($\hat{x}_j$)
        $gradient\_penalty = \lambda(\|\text{grads}(disc\_xhat, \hat{x}_j)\|_2 - 1)^2$
        $loss = loss + disc\_fake - disc\_real + gradient\_penalty + \epsilon_{drift} disc\_real^2 + \epsilon_{drift} disc\_fake^2$
    end
    new_gan.disc.SGD_step($loss$)
end

def update_gen(new_gan)**:**
    z = uniform_sample(-1, 1, ($batchsize$, $n\_latents$))
    $\widetilde{x}$ = new_gan.gen.get_outputs(z)
    $loss = 0$
    foreach $\widetilde{x}_j$ **in** $\widetilde{x}$ **do**
        $loss$ = $loss$ - new_gan.disc.get_outputs($\widetilde{x}_j$)
    end
    new_gan.gen.SGD_step($loss$)
end

Algorithm 2: Pseudocode for training RePR. *environments* is a list of the environments to be sequentially learnt and *exp* is an experience replay. *stm_agent* contains a predictor network and a target network, whereas the *ltm_agent* uses only a predictor network. *gan*.disc and *gan*.gen are the GAN's discriminator and generator networks respectively. α weights the importance of learning the new task vs. retaining previous tasks. In practice, the algorithm used for collecting RePR's results stores an array of pseudo-items which are generated by the GAN before training on a new task rather than generating them on the fly during training.