Understanding Forgetting in Artificial Neural Networks

by

Dylan R. Ashley

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

© Dylan R. Ashley, 2020

Abstract

This thesis is offered as a step forward in our understanding of forgetting in artificial neural networks. ANNs are a learning system loosely based on our understanding of the brain and are responsible for recent breakthroughs in artificial intelligence. However, they have been reported to be particularly susceptible to forgetting. Specifically, existing research suggests that ANNs may exhibit unexpectedly high rates of retroactive inhibition when compared with results from psychology studies measuring forgetting in people. If this phenomenon, dubbed catastrophic forgetting, exists, then explicit methods intended to reduce it may increase the scope of problems ANNs can be successfully applied to.

In this thesis, we contribute to the field by answering five questions related to forgetting in ANNs: How does forgetting in psychology relate to ideas in machine learning? What is catastrophic forgetting? Does it exist in contemporary systems, and, if so, is it severe? How can we measure a system's susceptibility to it? Are the current optimization algorithms we use to train ANNs adding to its severity?

This work answers each of the five questions sequentially. We begin by answering the first and second of the five questions by providing an analytical survey that looks at the concept of forgetting as it appears in psychology and connects it to various ideas in machine learning such as generalization, transfer learning, experience replay, and eligibility traces.

We subsequently confirm the existence and severity of catastrophic for-

getting in some contemporary machine learning systems by showing that it appears when a simple, modern ANN (multi-layered fully-connected network with rectified linear unit activation) is trained using a conventional algorithm (Stochastic Gradient Descent through backpropagation with normal random initialization) incrementally on a well-known multi-class classification setting (MNIST). We demonstrate that the phenomenon is a more subtle problem than a simple reversal of learning. We accomplish this by noting that both total learning time and relearning time are reduced when the multi-class classification problem is split into multiple phases containing samples from disjoint subsets of the classes.

We then move on to looking at how we can measure the degree to which ANN-based learning systems suffer from catastrophic forgetting by constructing a principled testbed out of the previous multi-task supervised learning problem and two well-studied reinforcement learning problems (Mountain Car and Acrobot). We apply this testbed to answer the final of the five questions by looking at how several modern gradient-based optimization algorithms used to train ANNs (SGD, SGD with Momentum, RMSProp, and Adam) affect the amount of catastrophic forgetting that occurs during training. While doing so, we are able to confirm and expand previous hypotheses surrounding the complexities of measuring catastrophic forgetting. We find that different algorithms, even when applied to the same ANN, result in significantly different amounts of catastrophic forgetting under a variety of different metrics.

We believe that our answers to the five questions constitute a step forward in our understanding of forgetting as it appears in ANNs. Such an understanding is essential for realizing the full potential that ANNs offer to the study of artificial intelligence.

Preface

This thesis is an original work by Dylan Robert Ashley under the supervision of Dr. Richard S. Sutton. Parts of this thesis may be published under different cover in the future. The source code for all experiments appearing in this work is freely available at https://github.com/dylanashley/catastrophic-forgetting.

For all the named and nameless people who helped me on my journey

Whenever we give up, leave behind, and forget too much, there is always the danger that the things we have neglected will return with added force.

– Carl Gustav Jung in Memories, Dreams, Reflections

Acknowledgements

I have a lot of people to thank for helping me put this thesis together. First and foremost, I would like to thank my supervisor: Dr. Richard S. Sutton. His mentorship over the past several years has been critical to my success. I would not have been able to advance this thesis without his guidance and support. I would also like to thank the Alberta Machine Intelligence Institute and the whole of the Reinforcement Learning and Artificial Intelligence lab at the University of Alberta, to both of whom I similarly owe everything.

I would like to thank my committee members Dr. Patrick Pilarski and Dr. Mark Ring, who provided great questions and valuable feedback throughout the examination process. I would also like to extend this thanks to everyone involved in the administration of my program and its requirements. This includes my committee chair, Dr. Carrie Demmans Epp, as well as the Department of Computing Science and the Faculty of Graduate Studies and Research at the University of Alberta.

I want to give a special thank you to Sina Ghiassian, who provided me with invaluable mentorship throughout my program. His insight into both graduate school and the act of running empirical experiments was of inestimable value throughout the thesis writing process. I would also like to thank all the other individuals who provided me with valuable comments on earlier drafts of my thesis presentation and thesis proper: Abhishek Naik, Banafsheh Rafiee, Chen Ma, Dr. Nuanyi Liang, Han Wang, Khurram Javed, Kris De Asis, Sina Ghiassian, Tian Tian, and Vincent Liu.

While writing this thesis, I spent a lot of time working with the Graduate Students' Association at the University of Alberta. The community fostered by the GSA was a constant inspiration to me throughout my degree. So I

would like to extend a special thanks to the GSA and all the fantastic people there, especially Fahed Elian, Dr. Sasha van der Klein, Chantal Labonté, Marc Waddingham, Mohammad Shanawaz, Dr. Courtney Thomas, Julie Tanguay, Lisa Hareuther, and Melissa Woghiren.

At this point, I want to extend my sincere appreciation to Dr. Richard S. Sutton, the Natural Sciences and Engineering Research Council of Canada, the University of Alberta, and the Department of Computing Science for their generous financial support. I want to similarly thank Compute Canada for providing me with the use of their extensive computational resources that facilitated the experiments presented in this thesis.

Last, but certainly not least, I would like to thank Antoinette Meredith, Cicely Ashley, Gwylim Ashley, Keith Ashley, and the rest of my family for all their help over the years. Without them, none of this would have even been conceivable. Likewise, I want to thank all my friends who kept me sane through the hurricane that is graduate school. You are all amazing.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This work is an investigation into the tendency for a type of learning system known as an Artificial Neural Network to forget things it has previously learned. Before doing that, it is critical first to establish what artificial intelligence (AI) is, what artificial neural networks (ANNs) are, and why both merit attention.

To understand why AI merits attention, we must be clear on precisely what AI refers to. Obtaining this clarity is somewhat more challenging than we would like, as there remains much disagreement regarding what constitutes the field of AI (Russell and Norvig, 2003, pp. 1–5). In this thesis, we adopt John McCarthy's definition of AI, which states that intelligence is "the computational part of the ability to achieve goals in the world" (McCarthy, 2007, p. 2), and AI is "the science and engineering of making intelligent machines" (McCarthy, 2007, p. 2). With that definition in mind, it is apparent why AI has long captured the imagination of so many. On his deathbed, John von Neumann thought about the connection between computers and the brain (von Neumann, 1958). Alan Turing, the founder of theoretical computer science, had a keen interest in AI and ended up making made several pioneering contributions to the field, e.g., The Turing Test (Turing, 1950). That these giants gave pause to the topic should be of no surprise to anyone, as, ultimately, AI has the potential of solving many of the grand challenges facing our societies today.

Consider the problems facing many nations when they try to provide high-

quality medical care for their citizens. The primary barrier facing governments here is the simple reality that health care is expensive. In 2019, the Canadian government expected that its annual health spending would amount to 11.6% of Canada's GDP (Canadian Institute for Health Information, 2019, p. 4). This high price tag means that any reduction in the cost of providing health care would free up societal resources to use for other purposes. However, we must ask ourselves whether or not this can be done without lowering the quality of the service provided. The answer is undeniably yes, as we could achieve that with automation.

Imagine, if you will, waking up one morning a bit under the weather. You pick up your phone and start lamenting to it. After describing your symptoms, it gives you a diagnosis and recommends a course of treatment for you. If the AI is good at its job, the diagnosis it offers you will come not just from your description of the symptoms and your personal medical history, but also from the more subtle information about you that only a companion as constant as your phone will know. If the AI is especially good at its job, the effectiveness of the treatment it offers you will rival or exceed the effectiveness of whatever treatment a doctor would recommend to you. So here, AI has been successful at automating away one part of the medical care process.

As a second example of the potential of AI, consider the challenge of building self-driving cars. In the United States, there were 37,133 fatalities due to traffic accidents in 2017, and it was the leading cause of death for people aged 17 through 21 (National Highway Traffic Safety Administration, 2019, p. 6). While many traffic accidents occur due to something beyond the reasonable control of the driver, many more are the result of human error. A sufficiently advanced, fully autonomous self-driving car, however, would be able to react significantly faster than a human and consequently be substantially less prone to error in this situation. Furthermore, a self-driving car would be incapable of driving under the influence and would not be capable of distracted driving. So here, AI technology has the potential to save many lives.

Both of the aforementioned examples refer to very active areas of ongoing research. While these may be examples of yet unsolved problems, in the past

decade, there has been a number of breakthroughs on other AI problems. Most of these are due to advances relating to ANNs.

ANNs are loosely based on our knowledge regarding the underlying mechanisms of the human brain and have shown themselves to be astoundingly flexible learning systems. They have been adapted to solve a wide range of different AI problems which has resulted in substantial improvements in facial recognition technology (Taigman et al., 2014), computers playing Atari games at the level of a human (Mnih et al., 2015), computers beating professional players at the ancient game of Go (Silver et al., 2016), automatic stylization of photographs to look like they are paintings by famous artists (Gatys et al., 2016), better text translation (Vaswani et al., 2017), automatic text generation (Radford et al., 2019), and major advancements in tackling the famous protein folding problem (Senior et al., 2020). Notably, each of the aforementioned successes independently represents a major leap forward in AI.

1.1 Catastrophic Forgetting and Continual Learning

ANNs, like most learning systems, try to learn specific functions by looking at example applications of that function. One issue with ANNs is their sensitivity to the way in which they are fed these examples. Problems where all the examples are fed to the learning system at once are known as *offline* problems. In contrast, problems where the examples are fed to the learning system one after another are known as *online* problems. ANNs struggle when applied to online problems as they tend to rapidly forget previously learned information when in the presence of new information (French, 1991, p. 173). This has been called catastrophic forgetting or catastrophic interference (McCloskey and Cohen, 1989).

The presence of catastrophic forgetting in online learning is a serious issue as there are many online problems in AI that we care about. In the earlier medical example, to recommend suitable medical treatments, an AI would need to be able to learn about you moment to moment. Thus it is a problem

that fundamentally needs online solutions. As another example of an online learning problem we care about, take the challenge of building AI companions to help decrease the social isolation some individuals face. Social isolation is a major problem in society with new evidence that it may be associated with grave health concerns (Cacioppo and Cacioppo, 2014) coming at a time when societal levels of social isolation are high (Cigna, 2018, p. 2). However, to be an agreeable companion, an AI would need to again learn about a person moment to moment as well as adapt as they change. An ANN-based learning system would be able to manage catastrophic forgetting effectively if it is to learn and adapt in such a manner.

Managing catastrophic forgetting is not only useful because it can reduce how much forgetting occurs, but also because it can be used to free up resources efficiently. In the AI companion example, if the AI companion is passed from one person to another person, only part of what it has learned would usefully carry over. If the companion was passed from a wine connoisseur to a cheese connoisseur, knowing about wines might no longer be useful, but knowing about the art of tasting would. So, here the AI could free up memory by selectively forgetting useless information but still utilize the useful information it had previously acquired.

The ability to selectively forget is one of the desiderata that defines continual–sometimes called lifelong (Chen and Liu, 2018, p. 55)–learning. Continual learning systems are learning systems able to retain previously learned knowledge and apply that knowledge to new problems (Silver et al., 2013, p. 51; Ring, 1997, pp. 77–78). To perform this without using arbitrarily vast amounts of memory, some forgetting–which is necessary to compress sufficiently large amounts of information in finite memory–must occur. Many instances of continual learning systems are ANNs, e.g., Ring (1997) and Tessler et al. (2017). In such instances, effectively controlling forgetting to ensure efficient online learning is essential.

The benefits provided by online learning systems has lead to a considerable amount of work developing ways to mitigate catastrophic forgetting, e.g., Goodfellow et al. (2013), Kirkpatrick et al. (2017), Lee et al. (2017), Zenke

et al. (2017), Masse et al. (2018), Sodhani et al. (2020). However, it continues to be an unsolved issue (Kemker et al., 2018). Its persistence is partly because the catastrophic forgetting problem remains not yet well understood. Indeed, most research into catastrophic forgetting is limited to the multi-task, batch supervised learning setting (see Section 2.1). This is the same setting used by the original work looking into catastrophic forgetting, i.e, McCloskey and Cohen (1989), but is not the most connected to online learning nor to how humans and other biological organisms operate.

We propose to revisit and expand on our understanding of what forgetting, and catastrophic forgetting, is in ANNs. By furthering this understanding, we hope to better equip the field in finding ways of understanding how our algorithms can control it to their benefit. Advancing this understanding requires not just meditating on it and how it relates to other topics, but also determining how we can build a testbed to measure different facets of it. To demonstrate the value of the above, we apply our testbed to understand better whether or not contemporary step-size adaption algorithms are affecting it and, if so, how. Altogether, if realized, this enhanced understanding would represent a small step forward in the field of AI.

1.2 Research Questions and Related Contributions

Under the preceding motivation, this thesis seeks to further our understanding of forgetting and catastrophic forgetting in ANNs. With that objective, this thesis's main research questions are the following:

- How does forgetting in psychology relate to ideas in machine learning?
- What is catastrophic forgetting?
- Does catastrophic forgetting exist in contemporary machine learning systems, and, if so, is it severe?
- How can we measure how a system experiences catastrophic forgetting?

- Are the current optimization algorithms we use to train ANNs adding to the severity of catastrophic forgetting?

The related contributions of this thesis are thus

1. an analytical survey that looks at the concept of forgetting as it appears in psychology and connects it to various ideas in machine learning,

2. empirical evidence demonstrating the existence of catastrophic forgetting in some contemporary ANNs,

3. a testbed that helps understand the degree to which some ANN-based learning systems suffer from catastrophic forgetting, and

4. evidence that the choice of which modern gradient-based optimization algorithm is used to train an ANN has a significant impact on the amount of catastrophic forgetting that occurs during training.

1.3 Outline

Including the introduction, the thesis is divided into seven parts. Immediately following this outline, we prepare the reader for the later chapters by providing essential background information in Chapter 2. We then attempt to distill the concept of catastrophic forgetting and determine how it relates to other concepts in machine learning and psychology in Chapter 3. Afterwards, we provide an example of catastrophic forgetting and use it to demonstrates in what way catastrophic forgetting is a problem worth investigation in Chapter 4. Following that, we expand on Chapter 4 to construct a benchmark with which we can measure catastrophic forgetting in Chapter 5. We then apply this benchmark to look at the effect of step-size adaptation algorithms on catastrophic forgetting in Chapter 6. Finally, we conclude by reflecting on how the previous chapters have served to fulfill the promises made in Section 1.2, what the implications of that are, and what future work remains in Chapter 7.

Chapter 2

Background

Machine learning (ML) is the branch of artificial intelligence concerned with learning. More specifically, ML seeks to build autonomous systems that can learn an underlying function from looking at examples. Over the years, ML has grown from a minor topic within artificial intelligence to its dominant subfield. This growth is, in large part, a product of the success and promise of modern ML methods (see Chapter 1).

This chapter seeks to provide the necessary ML background readers will need to understand the remainder of this thesis. Individuals already well versed in ML should feel free to skip it. It is assumed that readers already have a strong background in computing science, mathematics, and statistics, as providing the necessary introduction to these topics is outside the scope of this work.

ML is a vast topic with many sub-problems and solution methods. Each sub-problem presents a distinct set of challenges, and each solution method carries with it both strengths and weaknesses. Sections 2.1, 2.2, and 2.3 provide details on the three sub-problems referenced in this work. In terms of solution methods, this work only focuses on one specific, potent solution method known as artificial neural networks, which are explained in Section 2.4.

Section 2.5 concludes the chapter by provides references to additional material for the interested reader.

Figure 2.1: Supervised learning tries to learn the function that maps the first element of pairs like this, i.e., a picture of an animal, to the second element, i.e., the name of the animal in the image. When building a learning system to solve problems like this, it is standard practice to transform this into a classification problem by mapping the names of the animals to a subset of the natural numbers.

2.1 Supervised Learning

Supervised learning (SL) is perhaps the most common problem setting in ML. SL tries to learn a function from input-output examples. Mathematically, we could say that SL learns an approximation $\hat{f}$ of $f : X \to Y$ from a set of examples $(\mathbf{x}_0, y_0)$, $(\mathbf{x}_1, y_1)$, ..., $(\mathbf{x}_k, y_k)$ where each $y_i = f(\mathbf{x}_i)$. Note that it is generally assumed that $\mathbf{x}_0$, $\mathbf{x}_1$, ..., $\mathbf{x}_k$ are Independent and Identically Distributed, but, there is no restriction against, and it is not uncommon for f to be a noisy function. Furthermore, it is not required that each $\mathbf{x}_i$ be a vector and each y_i be a scalar. It is fairly normal for both $\mathbf{x}_i$ and y_i to be higher dimensional tensors.

The nature of X and Y are undefined in the general SL problem. If Y is a set of different classes, then a learning system that maps X to Y is known as a *classifier*, and we say that the learning system is solving a *classification* problem (see Figure 2.1). The same can be said if Y is a set of *one-hot vectors*, i.e., unit vectors that each have a single non-zero element, in which case the indices of the non-zero element in the vectors form an equivalent set of classes. Alternatively, Y could be an interval in $\mathbb{R}$, in which case a learning system that maps X to Y is said to be solving a *regression* problem.

When solving SL problems, it is generally desirable to obtain an estimate of the performance of a trained learning system on unseen, novel examples. To obtain this estimate, the set of examples is partitioned into several *folds*. These folds are then split into a *training set* and a *testing set*, the latter of which is used to evaluate the learning system. When solving a classification problem, to ensure that each fold is similar to the complete dataset, the folds are generally constructed such that each fold contains roughly the same distribution of classes as the full dataset. In this instance, the folds are called *stratified folds*.

Many solution methods for SL problems involve algorithm-level parameters known as *hyperparameters*. In order to find a setting for these hyperparameters that facilitate a learning system in solving a given problem, the learning system must be applied multiple times to the problem under different such settings. To both select a good hyperparameter setting and obtain an unbiased estimate of the performance of the learning system, the aforementioned folds must be split into three sets: a training set, a *validation set*, and a testing set. The validation set is then used to evaluate the performance of each hyperparameter setting. The final performance estimate is then obtained by applying the learning system with the best hyperparameter setting to the testing set.

2.2 Online Learning

Online learning is a variant of ML where examples only become available to the learning system one after another. This can be contrasted to offline learning, where all examples are available to the learning system at all times. While, in some cases, it is possible to treat many online learning problems as offline learning problems, this is, in general, not possible. In many cases, the stream of examples has no defined termination point. In such a scenario, an offline learning approach would be infeasible with only finite memory and finite time. Also, it can be the case that, for example, as it is in reinforcement learning (see Section 2.3), the contents of the stream of examples is a product of decisions made by the learning system. When faced with such a situation, a learning

system cannot wait for all or most of the data to become available before learning.

To provide a mathematical example of online learning, consider the structure of an online SL problem. Here we would say that the objective of the learning system is to learn an approximation $\hat{f}$ of $f : X \to Y$ from a set of examples $(\mathbf{x}_0, y_0)$, $(\mathbf{x}_1, y_1)$, ..., $(\mathbf{x}_k, y_k)$ where each $y_i = f(\mathbf{x}_i)$ and $(\mathbf{x}_t, y_t)$ only becomes available to the learning system at time t. If, as opposed to storing and using multiple examples, the learning system performs an update using only $(\mathbf{x}_t, y_t)$ at each time t, we say that the learning system is an *incremental learning system*.

Note that the online learning setting does not explicitly consider whether or not the underlying distribution generating the examples changes over time. If it does we say that it is a *non-stationary* problem as opposed to a *stationary* problem.

2.3 Reinforcement Learning

The reinforcement learning (RL) problem considers an agent receiving rewards through interacting with an environment. Mathematically, RL is generally formulated as a Markov Decision Process consisting of

- a set of states $\mathcal{S}$,
- a state-dependent action set $\mathcal{A} = \cup_{s \in \mathcal{S}} \mathcal{A}(s)$,
- a transition function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ with $\forall s \in \mathcal{S}$, $a \in \mathcal{A}$, $\sum_{s' \in \mathcal{S}} p(s'|s, a) = 1$ where $p(s, a, s')$ gives the probability of action a causing a transition from state s to s', and
- a possibly stochastic reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ where $r(s, a)$ gives the reward for taking action a in state s.

In an MDP, at each timestep $t \in \mathbb{N}$, while the environment is in state $s_t \in \mathcal{S}$, the agent takes action $a_t \in \mathcal{A}(s_t)$ according to its *policy*, $\pi : \mathcal{S} \to \mathcal{A}$.

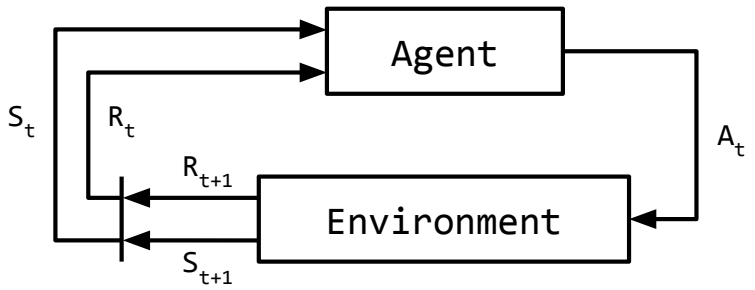

Figure 2.2: Reinforcement learning considers an agent interacting with an environment. At each step, the agent takes an action, and, in response, the environment changes its state, and rewards the agent.

In response, the environment changes its state to a new state, $S_{t+1} \in \mathcal{S}$, and the agent receives a reward R_{t+1} (see Figure 2.2).

There are two kinds of RL domains we model under the MDP framework: episodic and continuing. Episodic domains have a set of reachable states, $\mathcal{T}$, such that each state $s \in \mathcal{T}$ terminates an episode when the agents reaches it. Continuing domains, on the other hand, have no termination states. This difference has important implications when defining when defining the performance of agents.

In general, the quality of a policy is determined by the expected cumulative discounted sum of future rewards, i.e., the expected value of the *return*, when actions are selected according to that policy. The return is denoted by

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$$

where $\gamma \in [0, 1]$ is the *discount factor*. In most continuing domains $\gamma < 1$. This ensures $G_t \neq \infty$. In most episodic domains, $\gamma = 1$ which, if the expected length of an episode is less than infinity, still ensures $G_t \neq \infty$.

We call the expected value of the return given a current state s and under a policy π the *value* of state s under π. We denote this as

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

Sometimes it makes sense to talk about the value of taking specific actions in states. The expected value of the return under a policy π after action a has

been taken while in state s, is known as the *action-value* of the state-action pair (s, a) under π. We denote this as

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Action-values are generally used in the *control* setting, where the objective is to find an *optimal policy*. An optimal policy $\pi^* \in \Pi$, is a policy such that $\forall \pi \in \Pi, s \in \mathcal{S}, v_{\pi^*}(s) \geq v_\pi(s)$. In contrast to the control problem, the *prediction* setting tries to determine what the value of states are under a given policy π.

2.4 Artificial Neural Networks

Artificial neural networks (ANNs) are a learning system very loosely based on our knowledge of how networks of biological neurons operate. They consist of a sequence of layers composed of artificial neurons (see Figure 2.3). Each neuron functions by taking an aggregate of the output of neurons in the previous layer and then transmits a function of this aggregate to neurons in the next layer. Aggregation is done by weighted averaging. It is the weights in the aggregation process that determines the function the ANN computes.

The first layer in an ANN is the input layer and is the means by which an input is given to the network. Following this are any number of hidden layers and then an output layer that produces the output of the network. The number of neurons in the input and output layer thus specifies the dimensionality of the input and output of the learning system, respectively.

In addition to standard neurons, ANNs can also contain bias units. Bias units function similarly to standard neurons except that they reject all input and instead output a constant. In the function computed by an ANN, bias units serve a similar purpose to the y-intercept in the equation of a straight line ($y = mx + c$ with the y-intercept being c).

Prior to producing their output, neurons can apply a variety of transformations to the aggregate of their input. We call the functions that apply these transformations *activation functions*. Two of the most common activation functions are the identity function, i.e., $f(x) = x$, and the Rectified

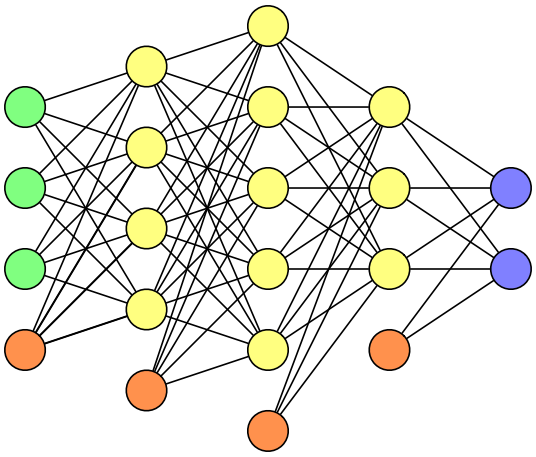

Figure 2.3: ANN with an input layer (green), three hidden layers (yellow), and an output layer (blue). Circles represent neurons, and the lines between neurons indicate the flow of information in a forward pass as computation proceeds from the input layer to the output layer. Each of the lines in the image corresponds to one weight in the network. Bias units are shown in orange. Note the lack of connections between bias units and previous layers.

Linear Unit, or ReLU activation function (Glorot et al., 2011; Jarrett et al., 2009; Nair and Hinton, 2010). ReLU activation is, arguably, the dominant activation function in contemporary ANN research, yet the only operation it performs is clipping values below zero: $f(x) = \max\{0, x\}$.

Using $g_{(i)}$ for the activation function of the i-th layer, $\mathbf{W}_{(i)}$ for the weights, and $\mathbf{b}_{(i)}$ for the bias weights, an n-layered neural networks takes as input some $\mathbf{z}_{(0)}$, and outputs

$$\mathbf{z}_{(n)} = g_{(n)}(\mathbf{W}_{(n)}^T \mathbf{z}_{(n-1)} + \mathbf{b}_{(n)})$$

While the above equation makes it clear that even simple ANNs have the power to represent a large class of functions, it also shows that a major obstacle to the application of ANNs is the challenge of finding a suitable configuration of weights. Thankfully, there is a considerable body of work devoted to this exact process.

2.4.1 Training ANNs

The weights in ANNs determines the function that maps their input to their output. As the main application of ANNs is finding a good approximation of

an unknown function that generated some data, and as ANNs are, in general, overparameterized (Zhang et al., 2017, p. 1), finding a good set of parameters can be challenging.

ANNs are generally trained by performing some form of gradient descent on a set of examples. Gradient descent is an iterative process over a set of parameters and thus requires some set of initial parameters before it can commence. In a neural network, there are several different ways that have been proposed for initializing the parameters of a network. The most common of these involves simply setting each parameter by sampling from a normal distribution characterized by mean zero and a small standard deviation. While often not ideal, such an initialization strategy often performs well in relatively simple settings.

A more advanced but still commonplace strategy for initializing the parameters of an ANN is Xavier initialization (Glorot and Bengio, 2010). Xavier initialization works by initializing each parameter $\theta_{i,j}$ in the network as

$$\theta_{i,j} \sim U\left[-\sqrt{\frac{6}{n_i + n_{i+1}}}, \sqrt{\frac{6}{n_i + n_{i+1}}}\right]$$

where $\theta_{i,j}$ is the j-th parameter in the i-th layer of the network, n_i is the number of parameters in the i-th layer, and $U[a, b]$ is the uniform distribution with a support consisting of all real numbers between a and b. The motivation behind this procedure is that doing so ensures roughly equal variance between layers in both forward and back propagation. While designed for deep ANNs, Xavier initialization is equally applicable to shallow networks.

Xavier initialization was designed with a particular activation function in mind (not covered here). He et al. took note of the complications this posed and created a new initialization method with similar properties under ReLU activation. Their method is known as He initialization (He et al., 2015) and remains one of the more common ways of initializing parameters in a network with ReLU activations. He initialization works by initializing each parameter $\theta_{i,j}$ in the network as

$$\theta_{i,j} \sim U\left[-\sqrt{\frac{2}{n_i}}, \sqrt{\frac{2}{n_i}}\right]$$

where $\theta_{i,j}$, n_i, and $U[a, b]$ all have the same meaning as in Equation 2.4.1. In both He initialization and Xaiver initialization, the parameters corresponding to bias units are frequently initialized differently. One such initialization strategy is to each such parameter by sampling from a normal distribution characterized by mean zero and a small standard deviation.

After being provided with some set of initial parameters, at each step in its iteration, gradient descent starts by computing the gradient of a function with respect to some parameters. Gradient descent then shifts all the parameters a step in the direction of their gradient and repeats. Since the gradient of the true function is rarely available in practice, generally gradient descent with ANNs is performed over some surrogate *loss function* describing how far the function the ANN computes is to the function described by a set of examples. We describe two such loss functions here.

Cross-entropy is a common loss function for SL classification settings. Cross-entropy compares the class probability distribution a classifier assigns to an example with the actual class that example belongs to. We can write the cross-entropy of a distribution $\hat{\mathbf{y}}$ and a one-hot encoding, $\mathbf{y}$, of the true class as

$$L(\hat{\mathbf{y}}, \mathbf{y}) = -\mathbf{y} \cdot \log(\hat{\mathbf{y}})$$

Instead of cross-entropy, in RL it is common to use the *squared Temporal-Difference error*. TD error is a one-step prediction error. Using v as the value estimate of the previous state, r as the immediate reward following a transition, γ as the discount factor, and v' as the value estimate of the next state, we can write the squared TD error as

$$L(v, r, v') = (r + \gamma v' - v)^2$$

When performing gradient descent on a loss function over a set of examples, a decision has to be made as to how many examples to include in each iteration. We call gradient descent with a single example per iteration *Stochastic Gradient Descent*. When all the examples are included in each iteration, we call it *batch gradient descent*. Finally, when each iteration considers some intermediate number of examples, we call it *mini-batch gradient descent*. Of

1: $\theta \leftarrow$ small random values
2: **while** termination criteria not met **do**
3: **for** $i = 1$ **to** n **do**
4: $\hat{\mathbf{y}}_i \leftarrow f(\mathbf{x}_i; \theta)$
5: $\theta \leftarrow \theta - \alpha \nabla_\theta L(\hat{\mathbf{y}}_i, \mathbf{y}_i)$
6: **end for**
7: **end while**

Figure 2.4: Stochastic Gradient Descent algorithm for a set of examples $(\mathbf{x}_1, \mathbf{y}_1)$, $(\mathbf{x}_2, \mathbf{y}_2)$, ..., $(\mathbf{x}_n, \mathbf{y}_n)$. When using an ANN, θ contains the weights of the network, and $f(\mathbf{x}; \theta)$ is the output of the neural network when fed $\mathbf{x}$ as input.

the three, only SGD is a purely incremental algorithm. Figure 2.4 shows the SGD algorithm as applied to an ANN.

When applying SGD to train ANNs, it is necessary to derive the gradients of the loss function with respect to the weights. Due to the structure of ANNs, this is non-trivial. One of the most significant advancements in ANNs was the popularization of the *backpropogation algorithm* (Rumelhart et al., 1986), which recursively applies the chain rule to derive gradients. As an understanding of backpropagation is not necessary to understand the contents of this thesis, we omit a full description of it here.

In contemporary research, most ANNs are not trained with vanilla SGD but are instead trained with a variant such as *SGD with Momentum*. SGD with Momentum (Qian, 1999; Rumelhart et al., 1986) applies a simple mathematical model of physical momentum to gradient descent. This momentum helps escape suboptimal *local minima*, i.e., locations in the space induced by the parameters which score better on the loss function than the surrounding area but worse than some other far away location. The SGD with Momentum algorithm as applied to an ANN can be obtained by replacing Line 5 in Figure 2.4 by

$$m \leftarrow \beta m + \alpha \nabla_\theta L(\hat{\mathbf{y}}_i, \mathbf{y}_i)$$
$$\theta \leftarrow \theta - m$$

Note that the update equations for SGD with Momentum, like SGD, has an α term but, unlike SGD, also has a β term. These are hyperparameters

for SGD. Here α is called the step-size parameter, and β is the momentum parameter.

Similar to momentum, *RMSProp* (Hinton et al., n.d.) can be viewed as a kind of adaptive step-size method. RMSProp re-scales step-sizes using an exponentially decaying average of the squared gradients. By re-scaling step-sizes like this, RMSProp naturally discourages oscillatory behaviour in weight updates. RMSProp as applied to an ANN can be obtained by replacing Line 5 in Figure 2.4 by

$$v \leftarrow \beta v + (1-\beta)(\nabla_\theta L(\hat{\mathbf{y}}_i, \mathbf{y}_i))^2$$
$$\theta \leftarrow \theta - \frac{\alpha}{\sqrt{v}+\epsilon}\nabla_\theta L(\hat{\mathbf{y}}_i, \mathbf{y}_i)$$

Note that, like SGD with Momentum, RMSProp introduces some additional hyperparameters. Here, β is a smoothing constant and ϵ is a small constant intended to prevent division by zero.

Combining SGD with Momentum and RMSProp, Adaptive Momentum Estimation, or *Adam* (Kingma and Ba, 2014), is the dominant optimization algorithm used to train ANNs. Like momentum, it keeps a running average of the gradient, and, like RMSProp, it keeps a running average of the squared gradient. Unlike both momentum and RMSProp, Adam additionally tries to correct for bias resulting from initializing the running averages with zeros, as is typically done in all three algorithms. Adam as applied to an ANN can be obtained by replacing Line 5 in Figure 2.4 by

$$m \leftarrow \beta_1 m + (1-\beta_1)\nabla_\theta L(\hat{\mathbf{y}}_i, \mathbf{y}_i)$$
$$v \leftarrow \beta_2 v + (1-\beta_2)(\nabla_\theta L(\hat{\mathbf{y}}_i, \mathbf{y}_i))^2$$
$$\hat{m} \leftarrow \frac{m}{1-\beta_1^t}$$
$$\hat{v} \leftarrow \frac{v}{1-\beta_2^t}$$
$$\theta \leftarrow \theta - \frac{\alpha}{\sqrt{\hat{v}}+\epsilon}\hat{m}$$

Here, t refers to the number of updates that have occurred previously, β_1 is analogous to β in momentum, β_2 is analogous to β in RMSProp, and ϵ is as in RMSProp.

2.5 Further Reading

A reader with a background in computing science should now be well versed in the topics necessary to understand the remainder of this thesis. The above sections, however, in the interests of brevity, represent significant simplifications of the topics they claim to cover. An interested reader wishing to improve their understanding of the above topics may wish to consult one or more of the following texts:

- Hastie et al. (2009) is an excellent reference on ML in general.
- Sutton and Barto (1998) is the classical reference when it comes to the topic of RL.
- Sutton and Barto (2018) provides an up-to-date overview of the RL problem.
- Puterman (1994) remains the standard book on Markov Decision Processes.
- Goodfellow et al. (2016) provides a contemporary, exhaustive overview of ANNs.

Chapter 3

A Meditation on Forgetting

Forgetting, i.e., the inability of a learning system to recall what was previously learned, has been a long-studied topic in several fields. Twenty-seven years before Alan Turing was born, Hermann Ebbinghaus conducted the first set of formal experimental studies looking into forgetting in humans (Ebbinghaus, 1913). Thirty-five years after Turing's death, Michael McCloskey and Neal Cohen published the first work looking into catastrophic forgetting in artificial neural networks (McCloskey and Cohen, 1989), or ANNs.

What McCloskey and Cohen had published was a report regarding a phenomenon that transpires when ANNs are applied to a not-purely-offline setting. They felt this phenomenon was worth reporting as it seemed to suggest a difference between forgetting as observed in humans and what they termed forgetting as observed in ANNs. Thus the study of catastrophic forgetting came to be as a consequence of scientists trying to ground research in ANNs using research in psychology. This sequence of events suggests that research into catastrophic forgetting cannot ignore its connection to psychological research into forgetting. This chapter is an attempt to examine this connection further and show how different, seemingly disparate ideas in machine learning are bound together by it.

3.1 The Blessing of Forgetting

Before diving into the psychological perspective on forgetting, it is essential first to address the crucial role forgetting plays in a learning system. Indeed,

forgetting is not fundamentally a bad thing and is, in many cases, a necessary part of many learning systems.

When we contemplate why forgetting may be beneficial, it is valuable to recognize that the world is a complicated place. As highlighted by the companion artificial intelligence (AI) example of Section 1.1, even just one human represents quite a bit of complexity for a learning system to deal with, and yet there are now more than 7.7 billion humans (United Nations, Department of Economic and Social Affairs, Population Division, 2019, p. 1), each one as complicated as the next. That 7.7 billion only accounts for only about 0.01 percent of the biomass on earth (Bar-On et al., 2018, p. 6507). Thus the pursuit of all knowledge is so utterly beyond hopeless that selective acquisition and retention of it is not just useful but absolutely essential. From here on, we refer to this argument as the *complex-world hypothesis*.

The complex-world hypothesis is not a cause for concern as, in most settings, much of what it is possible to learn is not useful. If you wanted to serve someone coffee, it does not matter if you know how to make a gin and tonic. Furthermore, if you never again have cause to make a gin and tonic, it also does not matter if you suddenly forget how to make one. In this way, even much of what has been learnt by a learning system can be forgotten without consequence.

Forgetting is also frequently to our advantage. In the above example, forgetting how to make a gin and tonic would free up memory to learn something new. Forgetting, in many ways, is a natural way to encourage new learning. This is particularly valuable when knowledge ends up becoming stale due to the passage of time and the changing nature of the world. Consider, for example, a doctor who has to decide what drug to administer to a patient afflicted by an uncommon medical condition. If the doctor is unaware of the usual variety of medications typically used to treat this condition, they will likely consult with the latest literature and prescribe the most effective drug currently known to treat it. If, however, they have treated this ailment before, they may be inclined to, for the purposes of efficiency, prescribe instead whatever they prescribed previously. If medical theory has changed significantly

since the last time they treated this ailment, this may end up being a mistake as the drug they prescribe in the second scenario may prove significantly less effective than what the doctor would have prescribed in the first scenario. Thus, forgetting here would help to prevent a doctor from making use of stale knowledge and has instead forced them to seek out what they may otherwise have considered irrelevant knowledge.

In the above example, the doctor who treated the condition before could be said to be stuck in local minima; the doctor in the first scenario has found a decent solution, and so is not motivated to see if there is a better one. Indeed, the presence of prior knowledge can frequently lead to an optimizer being trapped in local minima. In these kinds of situations, forgetting can encourage exploration and consequently help an optimizer escape local minima. For a second example, consider a chess-playing AI running an optimizer initialized using examples of grandmaster chess moves. These moves represent a learned strategy and, given the level of play this strategy represents, it is likely that any small changes to the strategy that can produce a superior strategy have already been explored. Thus only significant changes could ameliorate the strategy, but without sufficient encouragement to explore, e.g., by forgetting, the optimizer is unlikely ever to improve.

The aforementioned examples show that forgetting is not always a bad thing, and the complex-world hypothesis shows that forgetting is not a problem we can truly solve. Indeed, forgetting is an inevitable, and sometimes useful, phenomenon that is, in many ways, merely a part of learning. Forgetting is the discarding of old knowledge that is intrinsically linked to the acquisition of new knowledge. If you believe a coin is fair then acquires evidence to say it is, in fact, biased, then one may forget it is fair. Thus, rather than as a problem, forgetting is better described as a mechanism that is both present in, and employed by any decent learning system. The implications of this are that we should seek to understand forgetting more and design ways to analyze the properties of it exhibited by different algorithms rather than simply try to "solve" it. In this way, we may refine the forgetting tool in our algorithms to make ANNs more amenable to online learning.

3.2 Origins and Theories in Psychology

Ebbinghaus (1913) was the original formal experimental work in the study of forgetting. In order to measure forgetting, Ebbinghaus would himself memorize sequences of nonsensical syllables. They would then wait several hours and attempt to memorize the same sequences a second time. In order to measure the amount of forgetting that had occurred, Ebbinghaus would look at how much faster they could learn the sequences the second time. This way of measuring forgetting became known as the *relearning* or *savings* method (Chance, 2014, p. 351).

The relearning method of measuring forgetting captures the fact that just because someone can no longer recall exactly what they had once learnt does not mean they have forgotten it. For example, say it had been the case that Ebbinghaus could not repeat one of the sequences they learned several hours after learning it. If they could still repeat the sequence after hearing the first few syllables, then it is not wholly fair to say they had forgotten the entire sequence.

Relearning has been used in ANN research, e.g., French (1991), but is not the prevailing way of measuring forgetting in ANNs. This absenteeism is interesting as it can provide markedly different conclusions regarding the degree of forgetting (Hetherington and Seidenberg, 1989). That being said, many other metrics have been proposed since then. Recall still, however, remains the dominant one.

Since Ebbinghaus, the topic of forgetting has received considerable attention in psychology. Indeed, B. F. Skinner, one of the most notable psychologists of the 20th century, carried out some now-famous research into forgetting: Skinner (1950). To understand how contemporary psychology thinks about forgetting, though, it makes sense to start by looking at how they define it. The American Psychological Association defines forgetting to be "the failure to remember material previously learned" (VandenBos, 2015, p. 432). They go on to reference two notable theories around forgetting: *decay theory* and *interference theory*. The connections of these to modern machine learning are

explored in Section 3.3 and Section 3.4, respectively.

3.3 Decay Theory and Time

Decay theory, also called *trace-decay theory*, argues that, unless we use or rehearse it, we gradually forget learned material as time passes. As we are generally aware of some correlation between the time since we learned something and how well we remember it, this is a very natural way for us to think about forgetting. For example, we would encourage the reader to try to remember the names of their elementary school teachers. Since it has probably been several years since anyone reading this text interacted with them on a day-to-day basis, they would likely have forgotten a number of their names. Thus the passage of time has dulled the memory.

This way of thinking about time is incompatible with much of modern AI research as, in most contemporary online learning research, there is no explicit notion of time. Forgetting in ANNs is thought about as occurring over examples rather than over seconds. We can reconcile this difference by turning to ideas brought forward in McGeoch (1932). McGeoch argued that, for the brain, time itself is a sequence of events, and therefore forgetting was a direct consequence of new learning occurring. In an ANN context, decay theory is then the argument that learning from a batch of examples will always incur forgetting if that batch does not sufficiently overlap with prior learning.

Bridging the gap between decay theory and modern AI research reveals several interesting connections. Experience replay (Lin, 1992), for example, has long been an integral component of modern deep reinforcement learning algorithms, e.g., DQN (Mnih et al., 2015), where it has mainly been used as a way of mitigating catastrophic forgetting. The experience replay mechanism works by storing recent experiences in a buffer and then drawing from that buffer every so often to train an ANN. In that way, experience replay attempts to slow forgetting through directly appealing to the above conclusions about decay theory by ensuring there is an adequate amount of "rehearsing" of recent experiences.

Another interesting connection between decay theory and modern AI research comes from thinking about its connection to eligibility traces (Sutton, 1988; Sutton and Barto, 1998; Sutton and Barto, 2018). In online reinforcement learning, eligibility traces are a mechanism to help deal with the credit assignment problem (see Section 2.3). They work by maintaining a visitation tracker for each state. The tracker is set to decay as time passed but gets incremented each time the state in question is visited. This tracker then indicates the amount of time that has passed since the last time the agent visited that state. Each update is then applied once for each state and weighted by the current value of the visitation tracker for that state. In this way, visitations to states leave an impression on the eligibility trace vector that slowly decays over time.

The close association shown by the above examples implies that psychological work regarding forgetting has much to offer AI. Continuing AI research motivated by decay theory may benefit from considering what contemporary psychological research into decay theory has to say about differing rates of decay for different memories. Whether or not the decay rate between memories is shared or not shared appears to be one of the most substantial differences between interference theory and decay theory as they are applied to AI.

3.4 Interference Theory and Transfer

In contrast to decay theory, interference theory argues that it is interference between different instances of learned material that causes forgetting. That is to say, when we learn something new, it may interfere with previously learned material and thus cause us to forget either the new material or the previously learned material. For example, imagine that you have a friend called Ted and you just met someone who introduced themselves to you as Fred. It does not stretch the imagination to believe that the next time you see your new acquaintance Fred, you accidentally call him by the name of your good friend Ted. Alternatively, you might find that you start calling Ted by the name Fred! Here, interference between learned material has caused you to have difficulty

in recall, i.e., it has caused you to forget one of the two names.

In interference theory, interference is divided into two categories: *proactive interference* and *retroactive interference*. Proactive interference, also called *proactive inhibition*, refers to previous learning causing us to forget things we have just learned. In the name example, if we accidentally call Fred by the name Ted, then we have been the victim of proactive interference, as having a good friend called Ted has caused us to forget the name of our new acquaintance Fred. In contrast, retroactive interference, also called *retroactive inhibition*, refers to new learning causing us to forget things we learned previously. Again in the name example, if we accidentally called Ted by the name Fred, then we have been subjected to retroactive interference, as meeting Fred has caused us to forget about our good friend Ted.

Proactive interference is very closely related, but not the same as the phenomenon of prior learning affecting new learning. When prior learning affects new learning, rather than prior learning affecting the outcome of new learning, this is instead called the *transfer of training*. When the presence of prior learning improves new learning, it is said that *positive transfer* has occurred. When instead new learning is hampered by the presence of prior learning, it is said that *negative transfer* has occurred. Again, while proactive interference and negative transfer may seem similar, negative transfer is when old learning is detrimental to new learning, and proactive interference is when old learning is detrimental to the outcome of new learning (Reid, 1981). Similarly, the key difference between transfer of training and forgetting is that transfer of training refers the impact of old learning on new learning, and forgetting refers to the impact of new learning on the outcome of old learning.

When McCloskey and Cohen first reported the phenomenon of catastrophic forgetting, they referred to it as retroactive inhibition. What McCloskey and Cohen did in their work was train a network to perform addition on single-digit numbers. They first taught their network to add one to single-digit numbers and, once it had learned to perform that with high accuracy, they taught the network to add two to single-digit numbers. They refer to these tasks as the ones task and twos task, respectively. While they looked at a few different ways

of measuring the accuracy of the learning system during and after training, in all cases, the accuracy of the network on the ones task rapidly degraded when learning the twos task. In fact, following training, the network began to treat problems from the ones task as if they were problems from the twos task. Thus they concluded that the network displayed something akin to retroactive inhibition.

What was notable about the results of McCloskey and Cohen was that the ANNs in their experiments seemed to show more retroactive inhibition than what humans had demonstrated in Barnes and Underwood (1959). In order to understand the implications of this, though, it is essential to review the connections between McCloskey and Cohen's experiments and Barnes and Underwood's experiments.

Barnes and Underwood, like Ebbinghaus, experimented with forgetting through the medium of learning with words. Unlike Ebbinghaus, though, they had people learn to associate pairs of words. Participants were given a list of pairs where each pair on the list would consist of a nonsense syllable and a two-syllable adjective. After learning to associate all the pairs on the list, participants were given a second list containing the same nonsense syllables paired with different two-syllable adjectives. The objective of this experiment was to see what happened to the learned associations from the first list after learning the associations in the second list. They were especially interested in how the similarity of the adjectives in the first list to the adjectives in the second list affected this, i.e., they were interested in the possibility of an effect caused by transfer of training.

Barnes and Underwood measured the effect of transfer of training by running two experiments with only one key difference. In the first experiment, subjects were given two lists such that adjectives in both lists were unrelated. So, here, a participant might be given something like "aba-green" in the first list and "aba-fast" in the second list. In the second experiment, subjects were given two lists such that adjectives in the first list would always be paired with a similar adjective in the second list. So in the latter experiment, a participant might given something like "aba-happy" in the first list and "aba-sunny"

in the second list. The hope of this setup was that negative transfer would occur during the learning of the second list in the former experiment, and positive transfer would occur during the learning of the second list in the latter experiment.

In their experiments, Barnes and Underwood concluded that positive transfer only occurred when the lists were similar. Notably, participants also reported that, when the lists were similar, they used the learned associations from the first list to learn the second list. More importantly here, Barnes and Underwood concluded that when the lists were dissimilar, the more trials a participant did on the second list, the less they remembered about the first list. They showed this by directly asking the participants to write down the adjectives from both of the lists that were associated with the given nonsense syllable. The accuracy of their responses as a function of the number of trials the participants did on the second list is the comparison point used by McCloskey and Cohen.

The close relation between transfer of training and forgetting in psychology is particularly notable here as transfer of training inspired a largely unconnected topic of research in AI known as transfer learning. Transfer learning explicitly seeks to utilize prior learning related to one task to assist when learning another task (Taylor and Stone, 2009, p. 1633). The fact that the results of Barnes and Underwood suggested that positive transfer reduced retroactive interference implies the same may be true in their AI counterparts. It remains unclear to what degree this is true with Gutstein and Stump (2015) providing some evidence supporting it, and Riemer et al. (2019) provided some evidence against it.

It bears mention now that several later works support the results of McCloskey and Cohen. Hetherington and Seidenberg (1989), for example, replicated McCloskey and Cohen's experiments but determined that, after learning the twos problem, the relearning time of the ANN for the ones problem was shorter than the time it would take for the network learn a third problem. They also provided some evidence that rehearsal when learning can potentially prevent the network from forgetting how to solve some of the problems it has

learned about. In this way, they presented early motivation for something like experience replay.

Ratcliff (1990) also built scientific support for McCloskey and Cohen results. Ratcliff verified their results by first training an autoencoder on one batch of random vectors, then training it incrementally on new, novel random vectors. Their key conclusion was that the ability of the autoencoder to recognize a specific vector would decrease in proportion to how many phases of training had occurred since the autoencoder had last been trained on that vector.

Thus, with the work of McCloskey and Cohen and others, it is clear that ANNs do indeed exhibit some amount of retroactive interference, though the circumstances and degree of it remain somewhat unclear. It also suggests that psychological work into mitigating forgetting in humans could inspire new methods of mitigating catastrophic forgetting in ANNs. To truly pursue that though, requires a deeper understanding of the precise nature of catastrophic forgetting in ANNs, something this work hopes to advance.

3.5 Synaptic Plasticity and Generalization

Unsurprisingly, neuroscience research, or the study of biological neural networks, has, for many years, had close ties to the study of ANNs. Many significant advancements, such as Convolutional Neural Networks, the mechanisms behind many recent breakthroughs in Computer Vision (Goodfellow et al., 2016, p. 326), are based on neuroscience results. When it comes to forgetting, Abraham and Robins (2005) is perhaps the most notable recent neuroscience study with implications for research into ANNs.

The question Abraham and Robins sought to answer in their work is whether or not synaptic weights, i.e., the strength of the connections between neurons in biological neural networks, are stable or not. If there is this *synaptic stability*, then it would imply that the encoding of individual memories in the brain remains largely unchanged as time passes. If, on the other hand, there was not a significant amount of synaptic stability, i.e., there was *synaptic*

plasticity, then the encoding of memories in the brain would change as new learning occurs. The presence of synaptic plasticity would imply that large segments of our memories are being affected whenever new learning occurs.

The answer to the aforementioned question bears significant implications for research into ANNs. The connection of ANNs to biological neural networks implies that, if neuroscience derives new results regarding the behaviour of biological neural networks, it follows that AI might make further progress by attempting to determining under what conditions ANNs display the same behaviour. In this instance, the more synaptic stability biological neural networks display, the more we may want to encourage local representations, i.e., sparse representations, in ANNs, and vice-versa.

While the answer to the aforementioned question may have implications for research into ANNs, interestingly, the authors actually used experiments with an ANN to provide evidence for synaptic plasticity. To determine to what degree ANNs exhibit synaptic stability, they, as with Ratcliff, trained an autoencoder on random inputs online. They looked at the degree by which weights changed when trained on only new examples at each step. They then compared this to the degree by which weights changed when trained on the first k examples at step k. They concluded that the latter training scheme produces greater weight changes but also achieved higher accuracy. Thus, in their experiment, synaptic plasticity was necessary for strong performance from the ANN.

Abraham and Robins draw on the above evidence, as well as many additional results in neuroscience, to conclude that there is evidence for both synaptic stability and synaptic plasticity. They go on to conclude that there is still not enough evidence yet to determine the degree to which the brain demonstrates one or the other. For research into ANNs, this conclusion supports the idea that any event should at most make significant changes to some, but not all weights. In other words, there should be some amount of, but not total, locality in the representations.

The notion of locality as a desirable property of ANNs has been explored before. French (1991) argued that the overlap of representations in ANNs,

i.e., their ability to generalize, is one of the causes of catastrophic forgetting (p. 173). French proposed a novel way of creating more local representations in ANNs online. They went on to show that, under their experimental setup, locality reduced the degree of catastrophic forgetting exhibited by the ANN. Later results, e.g., Liu et al. (2019) and Srivastava et al. (2014), provided some verification of this conclusion that locality reduces forgetting.

French's results are intuitive as it is quite clear that local representations should, in general, reduce the degree to which different pieces of learning interfere with one another. This is true even if the learning system is not an ANN. For example, a tabular representation, i.e., using one-hot vectors as a representation, will naturally minimize interference. At the same time, a tabular representation also minimizes generalization. Locality is indeed the opposite of generality, and thus the more a learning system generalizes, the less locality it exhibits. It follows then that more synaptic plasticity leads to better generalization but at the cost of an increase in forgetting, a constant supported by Riemer et al. (2019). This constant lends support to Abraham and Robins's conclusions and provides a solid rationale as to why neither pure synaptic stability or pure synaptic plasticity should be sought in ANNs.

Chapter 4

An Example of Catastrophic Forgetting

In this chapter, we seek to give an empirical example of catastrophic forgetting in contemporary artificial neural networks (ANNs) and consequently provide rigorous scientific validation for its existence and the meaningfulness of its effect. From Chapter 3, we know that catastrophic forgetting refers to when new learning causes rapid forgetting of previously learned material, i.e., retroactive interference. Thus we want to establish experimentally whether high rates of retroactive interference occur when training modern ANNs.

The objective of showing that catastrophic forgetting exists permits us tremendous flexibility in experimental design. However, confirming how meaningful its effect, i.e., showing that its something we should care about, necessitates a more precise construction. We can demonstrate that it is not a niche phenomenon by ensuring our design conforms to contemporary practices in the structure and study of ANNs, e.g., multi-layered feedforward networks trained using backpropagation. We can then confirm that these results align with some previous work, such as McCloskey and Cohen (1989) and Hetherington and Seidenberg (1989), to lend them additional validity.

4.1 Experimental Setup

In order to demonstrate catastrophic forgetting in ANNs, we will have to assemble and justify several experimental components including a suitable

1. data-stream to feed to the networks,

2. ANN architecture to be trained,

3. optimization algorithm for training weights in the ANN, and

4. set of metrics to quantify retroactive interference.

In order to determine what makes a component suitable, it is useful to establish the desiderata of this experiment. To begin, for aforementioned reasons, we require the construction of this experiment to bear strong similarities to the structure of conventional experiments with ANNs. That means we must utilize widely-used datasets, build a network using components which are near-ubiquitous in the literature, avoid niche optimization algorithms, and only utilize common metrics for measuring catastrophic forgetting. We work through collecting components that satisfy these constraints in Sections 4.1.1, 4.1.2, 4.1.3, and 4.1.4, respectively.

4.1.1 Assembling a Data-stream

When considering how to assemble the data-stream, we must note that current belief suggests that catastrophic forgetting is, at least in part, a consequence of how the dataset is presented to the learning system. Thus while we want to use a well-known dataset to fill the data-stream, we would also like to ensure the data-stream we form with it does test the network's resistance to catastrophic forgetting. The easiest way to do this is to ensure non-stationarity exists within the data-stream.

When considering what dataset to use, we would be well-served by turning to the Modified National Institute of Standards and Technology dataset, or MNIST (LeCun et al., 1998). MNIST is one of the most ubiquitous datasets in ANN research and has been referred to as "the drosophila of machine learning" by Geoffrey Hinton (Goodfellow et al., 2016, p. 20). MNIST consists of 28×28 hand-written digits (see Figure 4.1). The conventional task for a learning system with MNIST is to predict what the digit is given the image.

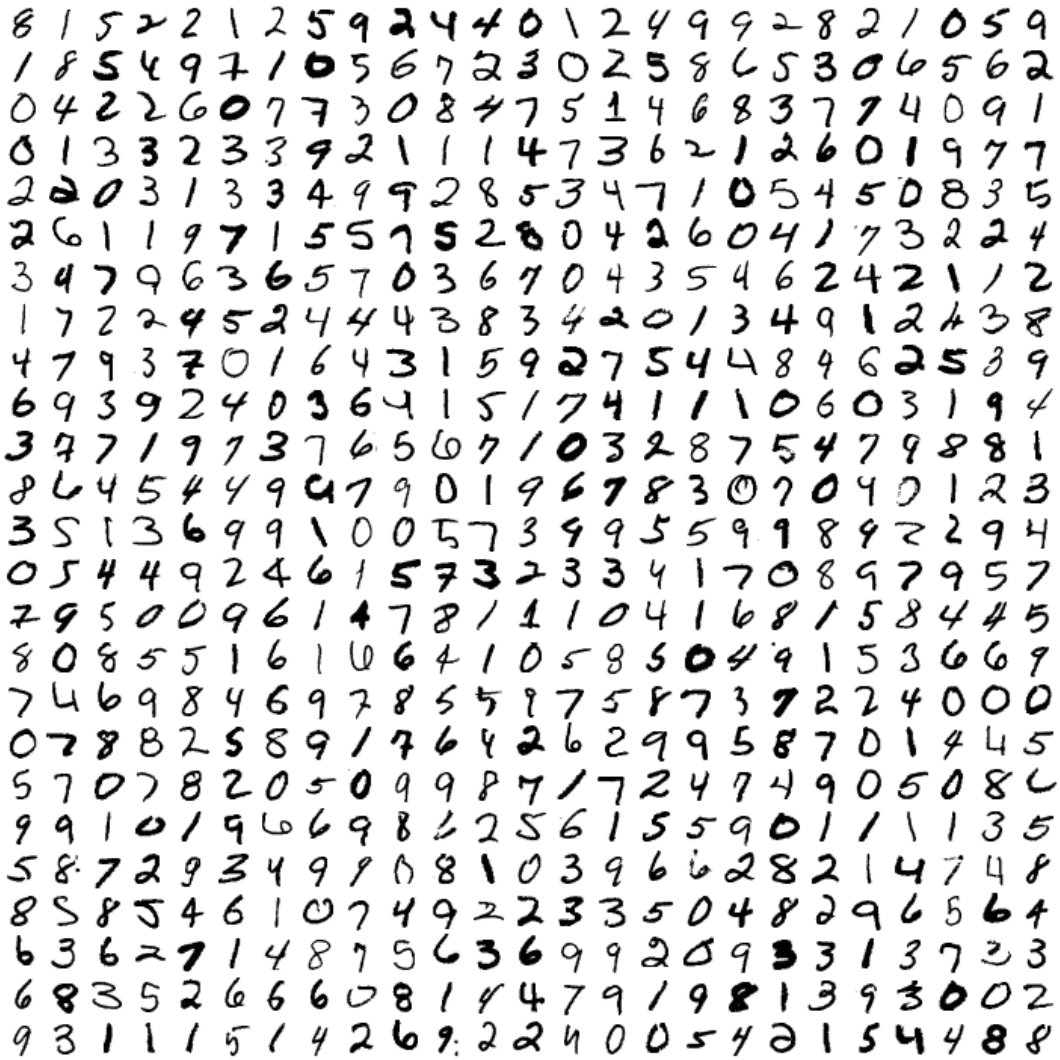

Figure 4.1: Some of the handwritten digits as they appear in the MNIST dataset. Each digit appears in the dataset as a labelled 28×28 greyscale image.

Fold \ Digit	0	1	2	3	4	5	6	7	8	9
0	593	675	596	614	585	543	592	627	586	595
1	593	675	596	613	585	542	592	627	585	595
2	593	674	596	613	584	542	592	627	585	595
3	592	674	596	613	584	542	592	627	585	595
4	592	674	596	613	584	542	592	627	585	595
5	592	674	596	613	584	542	592	626	585	595
6	592	674	596	613	584	542	592	626	585	595
7	592	674	596	613	584	542	592	626	585	595
8	592	674	595	613	584	542	591	626	585	595
9	592	674	595	613	584	542	591	626	585	594
Holdout	980	1135	1032	1010	982	892	958	1028	974	1009

Table 4.1: Distribution of digits in MNIST after dividing it into a holdout set and ten stratified folds. Note that each fold contains roughly the same number of each digit.

As image classification under MNIST is a supervised learning task, the first step in constructing our data-stream is to separate it into folds. We use stratified folds to ensure similarity across experimental settings. Furthermore, as MNIST is large enough, in all of our experiments, we choose to explicitly prevent the same sample from appearing twice to the learning system in a single run. This measure necessitates folds that have several thousand digits each. The exact distribution of the resulting separation is shown in Table 4.1.

To build our data-stream out of the MNIST dataset, we must now divide it into two tasks. These tasks will then be presented sequentially to the network in separate phases. Here we separate the ones-and-twos from MNIST to create the first task: a two-class classification task where a learning system must decide from an image whether that image is of a one or a two. Similarly, we separate the threes-and-fours from MNIST to create a second task. We can then observe any forgetting that occurs after first providing the network with samples from one task, then providing it with samples from the other task. At each step, we can measure the network's ability to perform on a given task by checking its accuracy on novel samples from a set of test folds. While when testing we can provide the network with all the digits in the test folds at once, the same is not true when training the network. Indeed, the number of digits

the network is provided at each step has a considerable effect on the learning time.

For the reasons mentioned in Chapter 1, we are ultimately interested in learning systems that can interact with the real world. While a sizable gap exists between the real world and MNIST, to at least be more compatible with the real world, it makes the most sense for us to present samples to the ANN one-by-one. While we do not preclude a learning system that chooses to collect samples and wait before performing updates, requiring that is severely limiting.

With the number of samples at each step decided, we must now consider when the data-stream will swap between tasks. As we do so, it is valuable to recognize that learning something after already mastering it, i.e., overlearning, can profoundly affect human forgetting (Chance, 2014, p. 366). So, while we want to ensure the network has sufficient time to learn each task before it changes, we do not want this to lead to overlearning. Having that could favour systems that learn quickly, which, in turn, would produce results requiring more complex interpretation.

We can achieve the required balance of tasks by keeping track of the network's running accuracy on a task and then swapping tasks once this running average exceeds some threshold. In practice, we thus require the network, at each time t, to make a guess $\hat{y}_t$ when given an image x_t, but before being observing y_t. We require the running average of correct guesses to exceed 90% before swapping tasks. This necessitates the network demonstrating mastery of the task but remains forgiving of early initial mistakes. To prevent predictors that always guess uniformly at random from the set of possible answers, i.e., random predictors, being able to achieve this accuracy regularly, we also require the learning system to maintain this accuracy for several steps before moving to the next task. If we require it to maintain this accuracy for five steps, the probability of a random predictor making five correct guesses at the start of learning and therefore moving to the next stage quickly is $2^{-5} = 3.125\%$. Since this experiment will need multiple runs to establish any conclusions with reasonable statistical confidence, these rare occurrences

should have little effect on the final results of the experiment.

Speaking of multiple runs, to test the full range of hypotheses we present in Section 4.2, we consider three different orderings for the tasks. In experiment $E_{(1,2)}$ the ones-and-twos task appears in the first phase, followed by the threes-and-fours task, then the ones-and-twos task appears again, and, finally, the threes-and-fours task appears one last time. In experiment $E_{(3,4)}$, we use the same phase structure as experiment $E_{(1,2)}$ but reverse their ordering. In experiment $E_{(1,2,3,4)}$, which we use to test basic assumptions of this setup, we drop the switching altogether and present all four digits to the network in a single phase. In all cases, we use two of the folds for training and two of them for testing. To eliminate any effect from the minor difference in fold sizes, we restrict phases to 2500 examples. We sweep over permutations of the samples in the folds for training and report averages.

4.1.2 Constructing a Network

When constructing a network to use in our experiments, we must ensure we create a network that has the capacity to solve the problem but simultaneously try to ensure the network is small; a small network is desirable as it can be easier to understand its behaviour, but if the network is too small then it will not be able to solve the problem no matter how much training it receives. Furthermore, it is well-known that ANNs without hidden layers harbour severe limitations (Goodfellow et al., 2016, p. 169), and so are infrequently used. The consequence of this is that we will need to have at least one hidden layer in our network.

With the above considerations in mind, we use a feedforward ANN with three fully-connected layers: an input layer with 784 units, a hidden layer with 100 units, and an output layer with 4 units. The images in the MNIST dataset are greyscale pictures containing $28 \times 28 = 784$ pixels. Thus the simplest preprocessing-free input layer for MNIST contains exactly 784 units. Similarly, since, as noted in Section 4.1.1, our data-stream specifies a four-class classification problem, the simplest output layer contains 4 units.

One notable feature of such an output layer in this setting is that, as noted

by Farquhar and Gal (2018, Section 6.3.2.), having output neurons that are not shared by tasks can have a significant impact on the degree of forgetting. While having non-shared output neurons should, in theory, enforce some amount of neural stability and thus decrease the effect of forgetting, since we only seek to demonstrate the existence of catastrophic forgetting and its severity, if we are able to observe a high degree of catastrophic forgetting, then this limitation of our architecture is immaterial. Furthermore, the use of non-shared output neurons does create a more traditional setting for MNIST classification. So, provided we avoid algorithms that explicitly exploit this neural stability, it is easier to justify using non-shared output neurons here.

When constructing the way by which information flows through the network, we must determine how neurons in layers are connected and what activation functions they use. Since only the hidden layer requires a non-identity activation function here, we use the contemporary and familiar ReLU activation for all the units in the hidden layer. To establish connections between layers, the simplest and most common system we could use is to connect each neuron in each layer to each neuron in the subsequent layer. Doing this means that our network will have a total of $784 \times 100 + 100 \times 4 = 78800$ parameters. This is a small network and yet will likely have the representational capacity to solve the given problem.

One final decision about the architecture that remains to be made is how the parameters will be initialized. We adopt a common initialization strategy that initializes each weight independently as a sample from a Gaussian with mean zero and standard deviation equal to 0.1. As the exact initialization can have a significant impact on performance, we, as we did with permuting the data-stream in Section 4.1.1, sweep over random seeds and report an average.

4.1.3 Picking an Optimization Algorithm

Several algorithms that can be used to train ANNs were mentioned in Section 2.4.1. The simplest fully-online one of these is Stochastic Gradient Descent. Moreover, SGD with Momentum, RMSProp, and Adam are all variants of SGD with added embellishments. This consistency means that SGD forms

the foundation by which almost all ANNs are trained and consequently is an excellent choice for determining the existence and severity of catastrophic forgetting.

To use SGD, we must select a loss function to minimize. As described in Section 2.4.1, cross-entropy represents an excellent choice here for the same reason SGD is a good choice of algorithm: because it is both simple and near-ubiquitous in the literature. Altogether, using cross-entropy with SGD helps ensure that the way the weights in the network are optimized is on par with much of the contemporary literature.

In addition to a loss function, SGD needs one more quantity to be specified: a step-size. Using a common strategy, we select a fixed step-size by trying each of 2^0, 2^{-1}, ..., 2^{-18}. We then, for each of experiments $E_{(1,2)}$, $E_{(3,4)}$, and $E_{(1,2,3,4)}$, select whichever step-size minimized the average total time spend cumulatively in all phases while achieving the desired accuracy in each phase. We ran the experiment with 50 seeds to perform this step-size selection procedure, then used the resulting step-size with 500 other seeds to generate the results reported in Section 4.3. The folds used in both training and testing for the step-size selection were disjoint from the folds used later. Furthermore, each seed was additionally used to permute the samples from the folds (as mentioned in Section 4.1.1) and initialize the networks (as described in Section 4.1.2).

4.1.4 Selecting Metrics

We want to select metrics that can confirm catastrophic forgetting exists and provide us with some insight into how severe an issue it is. In Section 3.4, we described two ways in which forgetting has been previously measured: retention and relearning time. Retention refers to the performance on a first, previously-mastered task directly after mastering a second task. Relearning time refers to how much more rapid it is to master a task for a second time after first mastering that task then mastering a second task. Of these two, retention is the more common one appearing in the catastrophic forgetting literature. Furthermore, retention is sufficient to confirm the existence of forgetting as if retention does decrease as time goes on, then it is true that some

form of forgetting has occurred. Further on that, in specific tasks, retention actually defines the precise kind of forgetting that is important. For these reasons, we use retention here as our principal way of measuring catastrophic forgetting.

While retention provides an excellent metric to show the existence and severity of catastrophic forgetting, it is important to note that, as demonstrated in Hetherington and Seidenberg (1989), using retention alone cannot give an accurate picture of the full nature of forgetting in ANNs. Thus, in addition to using retention, we also employ relearning time to confirm whether or not forgetting is absolute in our experiments. If it is absolute, we would expect the relearning time for a task to not differ from the amount of time it took to learn the task initially. If the time it takes to learn the first task the second time around does differ, then catastrophic forgetting is not as straightforward as a reversal of learning. Catastrophic forgetting can then be considered a more complex phenomenon whereby obtaining comparable measurements of catastrophic forgetting may be non-trivial in many problems of interest.

In this setting, retention is measured as the accuracy on the testing folds for ones-and-twos after the second phase has been completed in experiment $E_{(1,2)}$, and as the accuracy on the testing folds for threes-and-fours after the second phase has been completed in experiment $E_{(3,4)}$. Similarly, relearning time would be measured in both experiments $E_{(1,2)}$ and $E_{(3,4)}$ as the number of steps needed to complete the first phase as a function of the number of steps needed to complete the third phase.

4.2 Hypotheses

With the above experimental setup in hand, we now formalize the specific hypotheses we will be testing with this experiment. These hypotheses are described in plain English below and presented formally in Table 4.2. In Table 4.2, and in later sections, we use $D_{(1+2)}$ to refer to the testing folds for ones-and-twos, and we use $D_{(3+4)}$ to refer to the testing folds for threes-and-fours. We also use $E_{(1,2)}$, $E_{(3,4)}$, and $E_{(1,2,3,4)}$ as defined in Section 4.1.1.

Number	Null Hypothesis	Alternative Hypotheses
H1	In experiment $E_{(1,2)}$, the average accuracy of the learning system on $D_{(1+2)}$ is less than 90% after the first phase of training.	In experiment $E_{(1,2)}$, the average accuracy of the learning system on $D_{(1+2)}$ is greater than or equal to 90% after the first phase of training.
H2	In experiment $E_{(1,2)}$, the average accuracy of the learning system on $D_{(3+4)}$ is less than 90% after the second phase of training.	In experiment $E_{(1,2)}$, the average accuracy of the learning system on $D_{(3+4)}$ is greater than or equal to 90% after the second phase of training.
H3	In experiment $E_{(1,2,3,4)}$, the average accuracy of the learning system on $D_{(1+2)}$ is less than 90% after training.	In experiment $E_{(1,2,3,4)}$, the average accuracy of the learning system on $D_{(1+2)}$ is greater than or equal to 90% after training.
H4	In experiment $E_{(1,2,3,4)}$, the average accuracy of the learning system on $D_{(3+4)}$ is less than 90% after training.	In experiment $E_{(1,2,3,4)}$, the average accuracy of the learning system on $D_{(3+4)}$ is greater than or equal to 90% after training.
H5	In experiment $E_{(1,2)}$, the average accuracy of the learning system on $D_{(1+2)}$ is greater or equal to 90% after the second phase of training.	In experiment $E_{(1,2)}$, the average accuracy of the learning system on $D_{(1+2)}$ is less than 90% after the second phase of training.
H6	In experiment $E_{(3,4)}$, the average accuracy of the learning system on $D_{(3+4)}$ is less than 90% after the first phase of training.	In experiment $E_{(3,4)}$, the average accuracy of the learning system on $D_{(3+4)}$ is greater than or equal to 90% after the first phase of training.
H7	In experiment $E_{(3,4)}$, the average accuracy of the learning system on $D_{(1+2)}$ is less than 90% after the second phase of training.	In experiment $E_{(3,4)}$, the average accuracy of the learning system on $D_{(1+2)}$ is greater than or equal to 90% after the second phase of training.
H8	In experiment $E_{(3,4)}$, the average accuracy of the learning system on $D_{(3+4)}$ is greater or equal to 90% after the second phase of training.	In experiment $E_{(3,4)}$, the average accuracy of the learning system on $D_{(3+4)}$ is less than 90% after the second phase of training.

Number	Null Hypothesis	Alternative Hypotheses
H9	In experiment $E_{(1,2)}$, the average number of steps required to complete the third phase is greater than or equal to the average number of steps required to complete the first phase.	In experiment $E_{(1,2)}$, the average number of steps required to complete the third phase is less than the average number of steps required to complete the first phase.
H10	In experiment $E_{(1,2)}$, the average number of steps required to complete the fourth phase is greater than or equal to the average number of steps required to complete the second phase.	In experiment $E_{(1,2)}$, the average number of steps required to complete the fourth phase is less than the average number of steps required to complete the second phase.
H11	The average number of total steps required for the learning system in experiment $E_{(1,2,3,4)}$ is not different to the average number of total steps required for the learning system to pass both the first and second phase of experiment $E_{(1,2)}$.	The average number of total steps required for the learning system in experiment $E_{(1,2,3,4)}$ is different from the average number of total steps required for the learning system to pass both the first and second phase of experiment $E_{(1,2)}$.

Table 4.2: Null and alternative hypotheses to be tested. Each of the hypothesis pairs either checks standard assumptions made about our experimental setup, tries to answer a question of interest, or seeks to provide some insight into the phenomenon of catastrophic forgetting if it exists.

Each of the hypotheses we test either checks standard properties of our experimental setup (H1, H2, H3, H4, H6, H7, H8), answers a question of interest (H5), or provides some insight into the phenomenon of catastrophic forgetting if it exists (H9, H10, H11).

To begin, we test if the ones-and-twos problem is solvable by the network (H1). We then test if the threes-and-fours problem is solvable by the network after the network has previously solved the one-and-twos problem (H2). This latter test also checks whether or not the initialization created by learning the one-and-twos problem prohibits learning the threes-and-fours problem. We go on to test whether or not both of the above hypotheses depend on the ordering of the tasks (H6, H7).

In addition to testing whether the problems are solvable by the network independently, we also test whether the network has the capacity to solve both problems simultaneously (H3, H4). If this is the case, then if any forgetting occurs, it is not a result of the network lacking the representational capacity to learn a solution to both problems at the same time. To determine whether the nature of transfer is different in these two cases, we also test whether solving both tasks sequentially takes a different amount of time than solving both tasks independently (H11).

Next, we try to answer the key question we are asking with these experiments; we next test whether or not some form of retroactive inhibition is occurring, i.e., we check whether or not catastrophic forgetting exists. We do so by observing what the retention of the network is with respect to what it learned in the first phase, after the second phase has been completed (H5). We additionally test to ensure that, if forgetting occurs, it is not merely a consequence of the ordering of the problems (H8).

If forgetting has occurred, then we try to answer the question of whether or not catastrophic forgetting as measured by retention differs from catastrophic forgetting as measured by relearning (H9, H10). If so, then a simple reversal of learning is not what occurred as the network's weights are holding onto a part of what otherwise appears to have been forgotten. If that is the case, it suggests that measuring catastrophic forgetting may be non-trivial as retention alone does not fully capture the phenomenon of catastrophic forgetting.

4.3 Results

We test each of the hypotheses in Table 4.2 using the experiment described in Section 4.1. Figure 4.2 shows the results of this experiment. We calculate p-values for each hypothesis using either a one or two sample t-test as appropriate. Based on the results of this, we reject all null hypotheses in H1 through H11 in favour of their alternative hypotheses. For this, as a consequence of all p-values being smaller than 0.0001, we report a family-wise error rate of less than 0.01 using Bonferroni corrections.

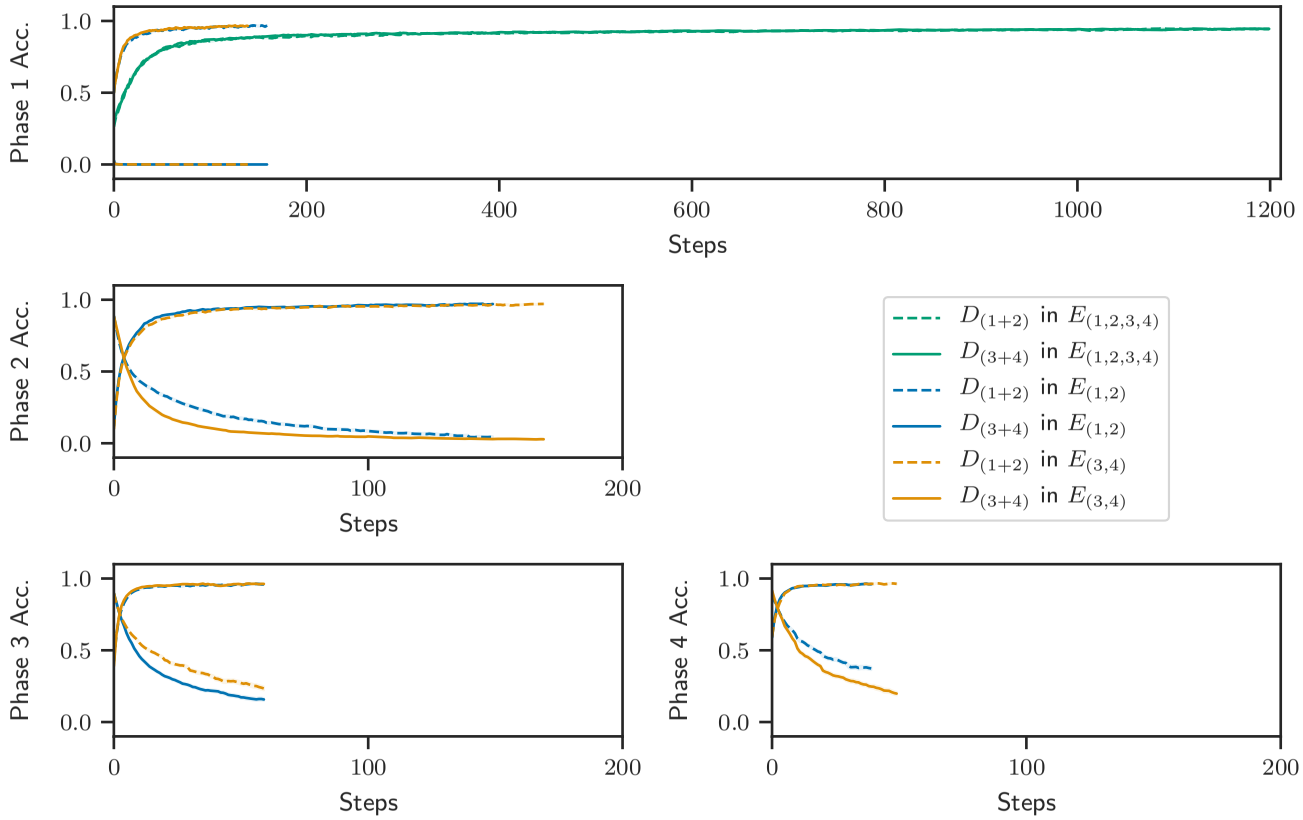

Figure 4.2: Performance on both tasks as a function of phase and step in phase. Lines are averages of all runs currently in that phase and are only plotted for steps where at least half of the runs for a given experiment are still in that phase. Standard error is shown with shading.

Experiment	Phase 1	Phase 2	Phase 3	Phase 4
$E_{(1,2)}$	116.87±2.64	123.38±2.05	48.80±1.46	35.27±1.33
$E_{(3,4)}$	105.70±2.47	142.50±2.44	47.18±1.37	35.23±1.32
$E_{(1,2,3,4)}$	1041.06±9.63	N/A	N/A	N/A

Table 4.3: Average number of steps needed to complete each phase in each experiment.

There are several key observations to be made while looking at Figure 4.2. First, each of experiments $E_{(1,2)}$, $E_{(3,4)}$, and $E_{(1,2,3,4)}$ was able to successfully complete each phase in a reasonable number of steps. For reference, we provide the average phase lengths in Table 4.3. Notably, while experiments $E_{(1,2)}$ and $E_{(3,4)}$ required relatively similar amounts of time for each phase, experiment $E_{(1,2,3,4)}$ took considerably longer to complete the first phase. While the optimal step-size for experiment $E_{(1,2,3,4)}$ was 0.03125 as opposed to 0.0625 for both experiments $E_{(1,2)}$ and $E_{(3,4)}$, the step-size cannot be responsible for this difference as the selection method outlined in Section 4.1.3 allowed both methods the opportunity to pick the step-size that minimized the total time. In the case of experiment $E_{(1,2,3,4)}$, this means that, unlike in experiments $E_{(1,2)}$ and $E_{(3,4)}$, the step-size used directly minimized the time it took to complete the first phase.

Further on the number of steps in each phase, both experiments $E_{(1,2)}$ and $E_{(3,4)}$ took substantially less time to get through all four phases than experiment $E_{(1,2,3,4)}$ took to get through one. For experiment $E_{(3,4)}$, the second phase was the longest, and for experiment $E_{(1,2)}$, both the first and second phases were the longest. That being said, both the first and second phase for experiment $E_{(1,2)}$ took less time than experiment $E_{(3,4)}$ in the second phase and more time than experiment $E_{(3,4)}$ in the first phase. On that, for both experiments $E_{(1,2)}$ and $E_{(3,4)}$, the fourth phase was the shortest and the third phase, the second shortest. In both of these instances, though, both experiments $E_{(1,2)}$ and $E_{(3,4)}$ took comparable amounts of time.

As hypothesized, catastrophic forgetting is observed in the second, third, and fourth phase for both experiments $E_{(1,2)}$ and $E_{(3,4)}$. This is demonstrated in both Table 4.4, which shows the retention, and Table 4.5, which shows the

Experiment	Phase	Accuracy on $D_{(1+2)}$	Accuracy on $D_{(3+4)}$
$E_{(1,2)}$	1	0.9558±0.0030	0.0000±0.0000
	2	**0.0814±0.0053**	0.9707±0.0011
	3	0.9636±0.0021	0.2411±0.0088
	4	0.4907±0.0095	0.9584±0.0019
$E_{(3,4)}$	1	0.9575±0.0031	0.0000±0.0000
	2	**0.0294±0.0023**	0.9706±0.0011
	3	0.9680±0.0009	0.3652±0.0090
	4	0.3944±0.0121	0.9460±0.0035

Table 4.4: Accuracy on each test dataset directly after completing a phase as a function of the experiment. Values shown in bold represent the retention metric.

Experiment	Relearning
$E_{(1,2)}$	4.15±0.19
$E_{(3,4)}$	3.57±0.16

Table 4.5: Length of the first phase as a function of the third under each optimizer in each experiment. These values represent the relearning metric.

relearning time. If no forgetting had occurred, we would expect retention to be close to 0.9 and relearning time here to be close to average time spent in the first phase divided by the minimum number of steps needed to complete a phase: five. Interestingly, the rate of forgetting, while in all cases very severe, varied drastically between both experiments $E_{(1,2)}$ and $E_{(3,4)}$. In addition, by referring to Figure 4.3, we can see it varied a lot between phases as well. The differences between the phases can be partially explained by the relearning time being shorter than the original learning time for both tasks. Further on that, the relearning time in the fourth phase was less than the relearning time in the third phase. Also, in both the third and fourth phases, the rate of forgetting was more severe on $D_{(3,4)}$ than $D_{(1,2)}$.

4.4 Discussion

The results provided in Section 4.3 allow us to reach several conclusions. First, as high rates of retroactive interference were exhibited, catastrophic forgetting exists. Second, as demonstrated by the near absolute loss of the ability to per-

form the first task after learning the second, it is a severe problem. The careful experimental design here provides evidence that the standard construction of contemporary ANNs does not meaningfully mitigate catastrophic forgetting. Furthermore, this phenomenon can occur in standard problems that we currently use ANNs to solve, e.g., image classification under MNIST. Thus the key objective of this chapter, being to validate its existence and demonstrate its severity, is achieved. That being said, several additional, interesting conclusions are also supported by the results.

Hetherington and Seidenberg had previously observed that the relearning time seemed to paint a different picture then retroactive interference when it came to forgetting in ANNs. The results of our experiment confirm that. The lower relearning time shows that, while retention was inhibited, forgetting was in no way absolute. This conclusion may indicate that the nature of the forgetting exhibited by ANNs may not disadvantage them in all settings. For example, if the ANN were being applied to a multi-task decision-making setting where the first few actions in each task mattered little, this type of forgetting might not be a meaningful issue.

The relearning time during the fourth phase being lower than the relearning time during the third phase also may imply that the relearning time may decrease as the network repeats tasks. To check this, we ran experiment $E_{(1,2)}$ in the same manner with the same hyperparameters as before except that we repeat the sequence of tasks a second time. Table 4.6 shows the number of steps in each phase for this experiment. This supplementary experiment provides clear evidence that, under some conditions, the relearning time will continue to decrease as the same task is repeated, even if other tasks occur in between repetitions. However, these results are preliminary, and further investigation is required to determine what the aforementioned conditions are. Such inquiry, though, remains beyond the scope of this work.

It was entirely unexpected to see the ANN take so much longer to learn both tasks simultaneously. This occurrence may indicate that forgetting is allowing the system to make more efficient use of its capacity. However, it is also true that, when learning the tasks separately, there will be a lower

Phase	Steps
1	116.87±2.64
2	123.38±2.05
3	48.80±1.46
4	35.27±1.33
5	27.43±1.12
6	21.55±0.85
7	21.35±0.99
8	16.23±0.66

Table 4.6: Steps per phase in experiment $E_{(1,2)}$ when the sequence of tasks is repeated a second time. Odd-numbered phases are the ones-and-twos task; even number phases are the threes and four task.

average interference between samples within a phase. This low interference is reminiscent of curriculum learning, which takes advantage of situations where simpler tasks allow the network to learn exponentially more rapidly. This speedup amplifies the motivation behind reducing catastrophic interference as doing so could potentially provide us with more sample efficient online, i.i.d., single-task learning. Still, this hypothesis remains entirely conjecture at this point, and future work must be conducted to verify or refute it with any degree of certainty.

The considerably different behaviour of the learning system when the tasks were reversed, compounded with the minor difference in performance over the permutations of the dataset or initial weights of the network, carries with it several implications. It, firstly, shows that minor differences in the dataset can have a profound impact on both the quantity and rate of forgetting exhibited by the network. This strong dependency on the dataset, as also noted by Kemker et al. (2018), implies that considerable care and restraint must be taken when reaching conclusions regarding the precise degree of forgetting exhibited by a learning system.

The differences in behaviour also confirm that we were correct in being concerned about the effect of overlearning. When the reversal of the tasks caused a phase to be completed more quickly, the following phase appears to exhibit a higher amount of forgetting, and vice-versa. This relation between time spent learning and subsequent forgetting may have consequences

when using different optimization algorithms, e.g., SGD with Momentum, or rehearsal methods, e.g., experience replay, as these could naturally lead to overlearning. This conclusion, however, is at odds with the benefits of rehearsal demonstrated by Hetherington and Seidenberg. Thus further work should be conducted to understand the relationship between overlearning and catastrophic forgetting.

Chapter 5

Building a Testbed

In this chapter, we expand on the contents of the last chapter to create a testbed for catastrophic forgetting. Our work towards this is motivated by the recognition that, as outlined in Chapters 3 and 4, catastrophic forgetting remains a notably subtle problem. Both Hetherington and Seidenberg (1989) and the results from Chapter 4 demonstrate that only considering one family of metrics for catastrophic forgetting, e.g., retention, limits one's ability to understand the phenomenon. Furthermore, as noted by Kemker et al. (2018), both changes to the metric used to quantify it, even principled ones, or changes to the experimental setting are likely to result in a different story being told by the results; there is strong evidence that catastrophic forgetting cannot be adequately studied under a single setting or with a single metric. Thus catastrophic forgetting cannot be sufficiently well understood in a limited experimental setting, and a principled testbed, consisting of both multiple settings and metrics, must be employed.

Few previous attempts have been made to explicitly construct proper testbeds for measuring catastrophic forgetting, but many different scenarios have been employed to measure it. We discuss a few of these testbeds and scenarios here. However, it is worth noting that most modern research on catastrophic forgetting focuses on strategies for mitigating it and, consequently, most recent work only includes a demonstration showing that specific strategies are capable of partially mitigating it under strict environmental conditions. For example, Kirkpatrick et al. (2017), one of the more influential works in the

area, looked at three distinct settings when presenting their method, Elastic Weight Consolidation. However, their focus was not on understanding catastrophic forgetting as exhibited by EWC and, so, when directly measuring catastrophic forgetting, they limited their analyses to the retention metric. Kemker et al. (2018) later showed that their method was much more vulnerable to catastrophic forgetting than their analysis claimed.

Kemker et al. provided a testbed for catastrophic forgetting, which considered several different settings and multiple metrics. Their main contribution was using this to demonstrate that the superiority of one method over another method is very dependent on both the setting and metric experimented with. However, like Kirkpatrick et al., they limited themselves to retention-based metrics and, unlike Kirkpatrick et al., batch settings.

Goodfellow et al. (2013) sought to empirically understand catastrophic forgetting as several different learning systems experienced it. In the process of doing so, they developed a small testbed. However, like Kemker et al. (2018), they only considered multi-task supervised learning batch classification settings and only looked at retention as a metric. Their main objective was to understand whether or not activation functions affected catastrophic forgetting and whether or not a specific technique known as dropout (Srivastava et al., 2014) was useful in partially mitigating it.

With the above examples in mind, in order to construct a principled testbed for catastrophic forgetting, we start by first identifying the limitations of the experimental setting from Chapter 4 in Section 5.1 and look at what can be done to eliminate them. From this analysis, we decide on introducing two new settings in Sections 5.2 and 5.3, respectively. We then formalize a set of metrics for each setting in Section 5.4. In Chapter 6, we apply the testbed we construct here to ameliorate our understanding of how step-size adaptation methods impact catastrophic forgetting.

5.1 A Retrospective on Limitations

Determining what the limitations the experimental setup described in Section 4.1 would have if rebranded as a testbed and then building on that gives us the shortest path to a principled testbed. By far, the biggest issue with using the experimental setup described in Section 4.1 as a testbed is that it only considers a multi-task supervised learning setting. In multi-task supervised learning, the samples within a phase are typically i.i.d., and so samples are only weakly correlated with both their successors and predecessors. The lack of a strong temporal correlation is significant when it comes to artificial neural networks, as it is well-known that they struggle to learn when exposed to such correlation (Mnih et al., 2015, p. 529). This is an issue as there are many settings where incremental methods cannot avoid strong temporal correlation in their data-stream, e.g., reinforcement learning. The implication of this is that, as we care about learning systems that can operate under the temporally-correlated data-streams in the real world, we need to either modify the MNIST setting or include an additional setting with this property in the testbed.

A second issue with the previous experimental design is that, as noted by Farquhar and Gal (2018, Section 6.3.2.), the presence of non-shared output neurons can have a significant impact on catastrophic forgetting. However, using shared output neurons would represent a fundamental change in the task the network is attempting to solve. In the previous experiment, this would have meant the network was solving a two-class rather than a four-class classification problem. Such an experimental design would not have been in line with Barnes and Underwood (1959). Thus, the presence of the shared output neurons does not represent an undesirable element of the experimental design. That being said, if we were to include an additional setting, we should endeavour to ensure it either uses shared output neurons or circumvents this problem entirely as the above concern could then be put to rest.

The third and final shortcoming of the previous experimental design is that, while it vaguely explores a second metric, it really only quantified for-

getting using one metric. Kemker et al. (2018) notes that changes to the metric, even principled ones, are likely to result in a different story being told by an experiment. Thus, in addition to the metrics previously explored, we need to consider additional metrics if we hope to construct a testbed that can holistically demonstrate the catastrophic forgetting experienced by a method.

Resolving all the above is most easily accomplished by including both additional setting in the testbed and additional metrics to measure catastrophic forgetting. This change implies several other alterations to the previous experimental design. First, the value of looking at different orderings of tasks is substantially diminished if we include additional settings. Thus this can be safely discarded. We can also dispense with the experiment including both tasks in one phase, i.e., experiment $E_{(1,2,3,4)}$, if we opt to use the same network as before. Second, as we are now interested in using additional metrics to measure catastrophic forgetting, such as relearning time, it makes sense to optimize hyperparameters such that they minimize the time it takes the learning agent to move through all four phases. The results shown in Section 4.3 suggest that this may not produce a particularly pronounced effect as the third and fourth phases seemed to contribute less to the total time than the first and second phases. Thus, such a change is unlikely to result in a meaningful change in the other metrics. We refer to the result of applying the above changes to the experimental setup described in Section 4.1 as the MNIST setting.

When constructing a testbed, for the reasons mentioned in Chapter 1, it is desirable to limit ourselves to incremental settings. While a learning system may choose to employ buffers to instead treat the problem as a mini-batch problem, the incremental setting is the most natural setting to consider catastrophic forgetting and so there is little incentive to ground a testbed in such a consideration.

5.2 Electing a Second Setting

To determine what a second setting for the testbed should be, we need to reflect on what properties we want it to have. From the above, we know that

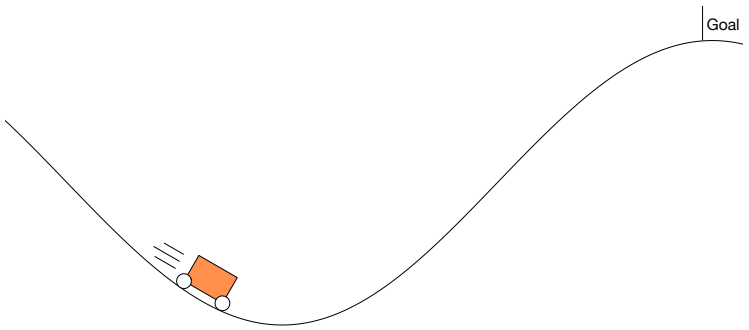

Figure 5.1: The Mountain Car setting simulates a car (shown in orange) whose objective is to reach the goal on the right. The car starts at the bottom of the valley and must rock back and forth in order to climb the mountain. Note that the car is prevented from falling off the left edge of the world by an invisible wall.

we want a setting that has a strong correlation between consecutive samples. Thus we do not want a second multi-task supervised learning setting. For obvious reasons, we also want to ensure that the setting we use is both simple and well-studied. We do not expect the second setting will enable the testbed to be all-encompassing; it is not desirable that the second setting is complicated enough to be able to capture an excessively large number of additional settings. Such an expectation is impossible to satisfy, and any attempt to do so will either make the testbed unnecessarily cumbersome or render the results of an application of the testbed incredibly challenging to interpret.

One setting that satisfies all the above desiderata is the Mountain Car setting (Moore, 1990; Sutton and Barto, 1998). Mountain Car is a popular, classic reinforcement learning setting that models a car trying to climb a hill (see Figure 5.1). The car starts at the bottom of a valley and lacks sufficient power to make it up the mountain by acceleration alone. Instead, it must rock back and forth to build up sufficient momentum to climb the mountain.

Formally, Mountain Car is an undiscounted episodic domain where, at each step, the car measures its position $p \in [-1.2, 0.6]$ and velocity $v \in [-0.07, 0.07]$, and then either accelerates in the direction of the goal, decelerate, or does neither. To capture the idea that the car should reach the goal quickly, it receives a reward of -1 at each step. The episode ends when $p \geq 0.5$. If, at

any point, $p \leq -1.2$, then p is set to be equal to -1.2 and v is set to be equal to 0. This last rule simulates the effect of it harmlessly hitting an impassable wall. With this last rule in mind, the position and velocity of the car in Mountain Car is updated at each step according to the following equations:

$$p_{t+1} = p_t + v_{t+1}$$
$$v_{t+1} = v_t + 0.001a_t - 0.0025\cos(3p_t)$$

where $a_t = 0$ when decelerating, $a_t = 2$ when accelerating, and $a_t = 1$ when the action selected is to do neither.

As we have no specific interests in evaluating learning systems that learn a good policy here, we are free to assign a fixed policy to the setting as it appears in the testbed. Doing so would not interfere with any of the aforementioned desiderata. It is desirable, however, that the policy we select should produce an interesting and meaningful data stream. One such policy is to have the car accelerate in the direction of movement. This policy does introduce an edge case where the velocity is zero, but provided there is sufficient randomness in the initialization of episodes, the probability of the car maintaining zero velocity for two consecutive steps is effectively zero. Thus it is fair to simply let the car neither accelerate or decelerate when its velocity is zero.

We plot the state-values in Mountain Car under the above policy in Figure 5.2. The key takeaway from this figure is that the episode lengths follow a distinct pattern characteristic of strong policies in Mountain Car. Additionally, no evidence appears here that the policy ever performs excessively poorly. Altogether this suggests that, while potentially not optimal, the above policy remains a strong policy.

The last two details we need to resolve before we can include Mountain Car in the testbed is how episodes will be initialized and how performance on the domain can be evaluated. For the former, we follow Sutton and Barto (1998) in initializing each episode with $v = 0$ and p selected uniformly from $[-0.6, 0.4)$. For the latter, we can measure the Root Mean Squared Value

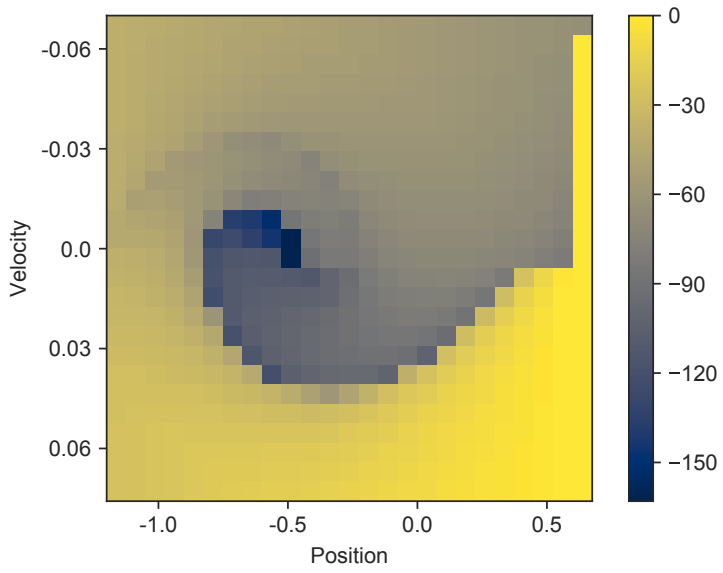

Figure 5.2: Values of states in Mountain Car setting when the policy the car follows is to always accelerate in the direction of movement. Note that the value of a state in Mountain Car is the negation of the expected number of steps before the car reaches the goal.

Error, or RMSVE under the above policy which is defined to be

$$RMSVE = \sqrt{\sum_{s \in \mathcal{S}} d_\pi(s)(\hat{v}_\pi(s) - v_\pi(s))^2}$$

where $\mathcal{S}$ is the set of all states, $d_\pi(s)$ is the proportion of time above policy π spends in state s, $\hat{v}_\pi(s)$ is the value estimate for state s under π, and $v_\pi(s)$ is the true value of state s under π.

Mountain Car has an unlimited number of states, so we approximate RMSVE here by using a test set of states. We follow Ghiassian et al. (2020) in generating this set by repeatedly running episodes to create a trajectory containing $10,000,000$ transitions. We then sample 500 test states from this trajectory uniformly and with replacement. The full trajectory and resulting sample are shown in Figure 5.3.

From the trajectory shown in Figure 5.3, it is clear that the policy and initialization procedure we selected are leading the car through a large area of the state space. This behaviour is desirable here. To ground this belief, though, it makes sense to consider a comparison point. It seems natural that we could get a more even coverage of the state space by forcing episodes to

begin by sampling a position and velocity uniformly at random from the state space. Using this initialization procedure with the above policy, we obtain the trajectory shown in Figure 5.4. Clearly, trajectories under this modified initialization procedure end up exploring the state space less evenly. Thus, the previous initialization procedure and policy are more useful for our purposes.

5.3 Finding a Third Setting

While the Mountain Car setting presented in the previous section removes many potential limitations from the testbed, it has some shortcomings when included in the testbed. For reasons mentioned later in Section 5.4, the Mountain Car setting is more limited than the MNIST setting in terms of which measures of catastrophic forgetting it is compatible with. This deficiency renders analysis under the Mountain Car setting somewhat more limited than may be desirable. We can easily and effectively mitigate this by introducing a third setting that retains some similarities with Mountain Car. In this way, greater verification of conclusions reached through the application of the testbed can occur.

For a third setting, we start by noting that the desiderata from Section 5.2 are also satisfied by the Acrobot setting (DeJong and Spong, 1994; Spong and Vidyasagar, 1989; Sutton, 1995). Like Mountain Car, Acrobot is a popular, classic reinforcement learning setting. It models a double pendulum combating gravity in an attempt to invert itself (see Figure 5.5). The pendulum moves through the application of force to the joint connecting the two pendulums. However, not enough force can be applied to smoothly push the pendulum such that it becomes inverted. Instead, like in Mountain Car, the force must be applied in such a way that the pendulums build momentum by swinging back and forth.

Formally, Acrobot is an undiscounted episodic domain where, at each step, the acrobot measures the sin and cos of the angles of both joints as well as their velocities. A fixed amount of force can then be optionally applied to the joint between the two pendulums in either direction. Like with Mountain Car,

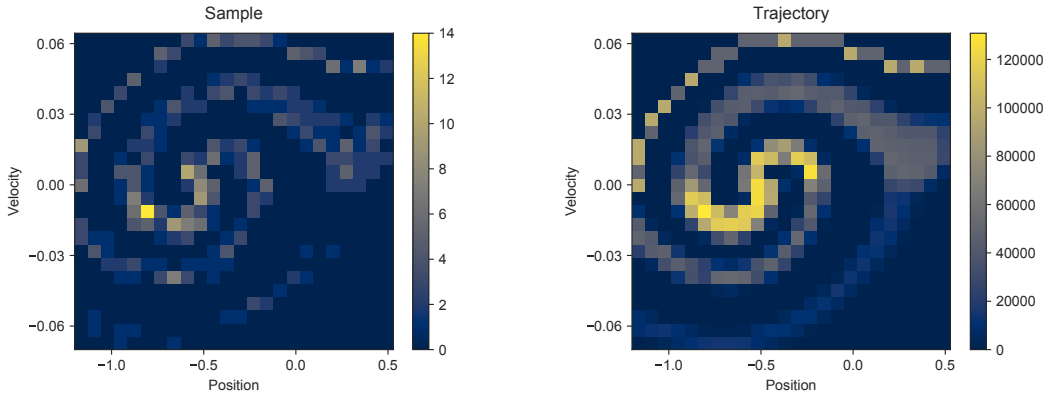

Figure 5.3: On the right is the distribution of states when initialization is done by setting $v = 0$ and selecting p uniformly from $[-0.6, 0.4)$. On the left is a uniform sample from this distribution that can be used for testing purposes. Note the distinctive pattern which covers a wide area of the state space.

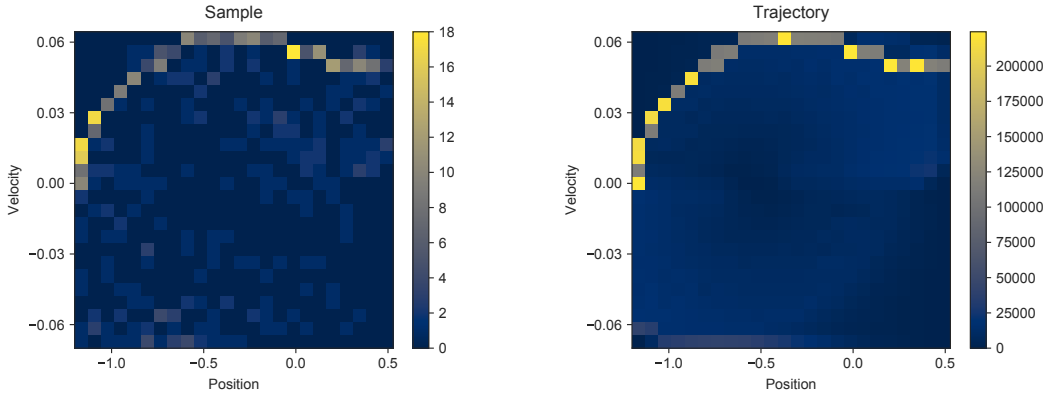

Figure 5.4: An alternative distribution of states and test states when initialization is done by selecting v and p uniformly from their range of possible values. Note the locality of the pattern this generates.

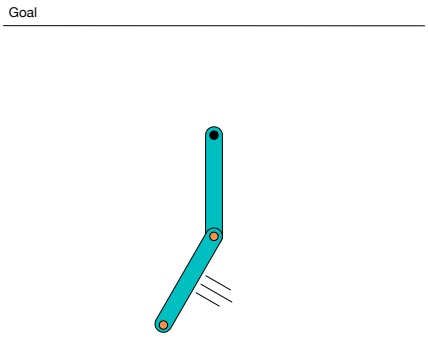

Figure 5.5: The Acrobot setting simulates a double pendulum whose objective is to place the end of the outer pendulum above a goal line. Force is applied to the joint between the two pendulums. The pendulums must rock back and forth in order for the outer pendulum to reach the goal.

the acrobot receives a reward of -1 at each step. Both pendulums have equal lengths, and episodes terminate when the end of the second pendulum is at least the pendulum's length above the pivot. The velocity of the inner joint angle in radian per second is bounded by $[-4\pi, 4\pi]$, and the velocity of the outer joint angle is bounded by $[-9\pi, 9\pi]$.

We use the OpenAI Gym implementation of Acrobot (Brockman et al., 2016), which is based on the RLPy version (Geramifard et al., 2015). The most significant difference between the RLPy implementation of Acrobot and the one used in Sutton (1995) is that it uses Runge-Kutta integration to provide more accuracy in approximating the system dynamics. The equations of motion that describe the pendulum movements under this setup are significantly more complicated than the equations for Mountain Car, and so are omitted here. The original equations of motion can be found on page 1044 of Sutton (1995), and the implementation we use can be found at https://github.com/openai/gym/blob/master/gym/envs/classic_control/acrobot.py.

Like for Mountain Car, we fix the policy of the agent. However, finding a good, simple rule-based policy for Acrobat is not as straightforward. Inspired by the policy we used in Mountain Car, we adopt a policy whereby force is applied at each step according to the direction of motion of the inner joint. To deal with situations where centripetal force renders the inner pendulum

effectively immobile, we augment this policy with the rule that no force is applied if the outer joint's velocity is at least ten times greater than the velocity of the inner joint.

We ran the above policy for 1,000,000 episodes and observed an average episode length of 156.0191 with a standard deviation of 23.4310 steps. The maximum number of steps in an episode was 847, and the minimum was 109. Thus this policy displays sufficient consistency to be useful for learning but enough variability to ensure a reasonably heterogeneous data-stream.

As before, we must still determine how we can evaluate performance in this new setting. For consistency, we follow the exact same procedure as in Section 5.2.

5.4 Measuring Catastrophic Forgetting

When comparing methods, there is a clear need to consider several alternate ways of measuring catastrophic forgetting, if only because the metric used to quantify catastrophic forgetting has a tremendous impact on the amount of forgetting reported. Kemker et al. (2018) previously observed that and our contrasting of retention and relearning in Chapter 4 corroborates this. Furthermore, ideally, each of these metrics would be somewhat distinct from one another. This distinctiveness is necessary in order to be able to reach broad, valid conclusions through their application.

When considering what would be good metrics to include in our testbed, it makes sense to prioritize heterogeneity and validity. That is to say, we should select metrics that are distinct from one another but can all be said to either measure catastrophic forgetting or a learning system's susceptibility to it. This criteria makes the retention metric and relearning metric of Chapter 4 an excellent starting point. Both have connections to psychological research (see Section 3.2 and Section 3.4, respectively), already exists in the literature as metrics for measuring catastrophic forgetting, and tell different stories. However, while some attempts have been made under specific conditions, e.g., Fedus et al. (2020), it is not so clear how these can be applied in

both a general and a principled manner to a setting without clear task boundaries. So, while both can be, and should be, included in the testbed as a way of measuring catastrophic forgetting, further metrics will be required for the additional settings introduced in Sections 5.2 and 5.3.

To determine what additional metrics we can include in the testbed that do not rely on clear task boundaries, we can turn to the work of French. In French (1991), the authors put forward the claim that catastrophic forgetting is a consequence of overlapping representations (p. 173), i.e., generalization. We previously explored this idea in Section 3.5. As a way of measuring the degree of generalization exhibited by a network–and therefore, theoretically, the degree of catastrophic forgetting it is susceptible to–French put forward *activation overlap*. The activation overlap of a network for two samples is simply the shared activation of the network on both samples.

Activation overlap was originally conceived for dense activation functions where most units would exhibit some amount of activation on any given sample. Thus, the activation overlap for a specific unit is the minimum activation of that unit under each sample. To make it more amenable to modern networks with sparser activation, e.g., networks using ReLU activation, we interpret the activation overlap of a network with respect to two samples as the dot product of the activations of the hidden units under each of the samples. The dot product has been previously used to estimate the representational overlap of networks, e.g., Kornblith et al. (2019), and thus this is both simple and acceptably in line with contemporary thought when measuring the activation overlap. To avoid confusion between this and activation overlap as originally defined by French, we refer to the former as activation similarity here. Importantly, this naming distinction is not representative of any conceptual difference.

Mathematically, we can write the activation similarity of a network with hidden units h_0, h_1, ..., h_n with respect to two samples $\mathbf{a}$ and $\mathbf{b}$ as

$$s(\mathbf{a}, \mathbf{b}) = \frac{1}{n} \sum_{i=0}^{n} g_{h_i}(\mathbf{a}) \cdot g_{h_i}(\mathbf{b})$$

where $g_{h_i}(\mathbf{x})$ is the activation of the hidden unit h_i with a network input $\mathbf{x}$.

A more contemporary measure of catastrophic forgetting than activation

similarity is *pairwise interference* (Ghiassian et al., 2020; Liu, 2019; Riemer et al., 2019). Pairwise interference seeks to explicitly measure how much a network learning from one sample would interfere with learning on another sample. In this way, it corresponds to the tendency for a network under its current weights to demonstrate positive transfer and retroactive inhibition. Thus, pairwise interference considers the stability-plasticity approach to catastrophic forgetting. Mathematically, the pairwise interference of a network with respect to two samples $\mathbf{x}_t$ and $\mathbf{x}_i$ at some instant t can be written as

$$PI(\theta_t; \mathbf{x}_t, \mathbf{x}_i) = J(\theta_{t+1}; \mathbf{x}_i) - J(\theta_t; \mathbf{x}_i)$$

where $J(\theta_t; \mathbf{x}_i)$ is the performance of the optimizer on the objective function J for $\mathbf{x}_i$ and $J(\theta_{t+1}; \mathbf{x}_i)$ is the performance on J for $\mathbf{x}_i$ after performing an update at time t using $\mathbf{x}_t$ as input. Note that pairwise interference could be positive. In such a scenario, the effect of positive transfer is overwhelming any instances of retroactive inhibition.

Both activation similarity and pairwise interference, unlike retention and relearning, are suitable for use in a setting without clearly defined task boundaries, e.g., Mountain Car and Acrobot. However, they are only defined in terms of pairs of examples. Thus some derivation must be done to produce equivalent setting-wide metrics. In all cases, we follow Ghiassian et al. (2020) to generate a setting-wide metric by getting the average of the metrics between all pairs in a set of examples. For MNIST, we obtain this set by sampling ten examples from each of the four classes in the test set. For Mountain Car, we overlay a 6×6 evenly-spaced grid over the state space (with position only up to the goal position) and then using the center points of the cells in this grid as examples. Finally, for Acrobot, we generate this set by sampling 180 random states uniformly from the state space.

Chapter 6

The Impact of Step-size Adaptation

In this chapter, we apply the testbed developed in the last chapter to answer a question regarding catastrophic forgetting in artificial neural networks (ANNs). Here, we will try to determine whether or not step-size adaptation methods have a meaningful effect. While previous work has been done explicitly looking at how other aspects of a learning system influence it, e.g., activation functions or dropout, little work, if any, has been done explicitly looking at the influence of the step-size adaptation methods used to train most contemporary ANNs. This absence is in contrast to the potential significance of their effect.

Step-size adaptation methods remain dominant in batch, offline settings where forgetting is rarely a concern. However, they are often blindly applied to online–sometimes incremental–settings where forgetting remains a major concern. Thus an answer to the above can potentially yield insight relevant to a large body of contemporary work.

In addition to determining the existence of the effect of step-size adaptation methods on catastrophic forgetting, we would also like to use this opportunity to empirically understand the relationship between the metrics in the testbed. The degree of conflict between these metrics would imply the degree to which catastrophic forgetting is a subtle phenomenon.

With the above goals in mind, we first outline our experimental setup and what we aim to observe under it in Section 6.1. We then present the results

of the experiment in Section 6.2 and discuss the meaning and implication of these results in Section 6.3.

6.1 Experimental Setup

To satisfy the objectives of this chapter, we need to assemble and justify several experimental components. Unlike in Chapter 4, we will be working with a formal testbed. The value of this is evident here as, despite conducting a more complicated experiment, utilizing the testbed means there are fewer additional components we need to select. The only remaining components we have to decide on are

1. ANN architectures for each of the three settings, and
2. the optimization algorithms with adaptive step-sizes to use for training weights in the ANNs.

We finalize each of the above components in Sections 6.1.1 and 6.1.2, respectively. After fixing these, we discuss what we are interested in observing under this setup in Section 6.1.3.

6.1.1 Choosing Architectures

When deciding on network architectures, our objectives are much the same as they were in Section 4.1.2. We still want a modern ANN that has the capacity to solve the problem and yet is sufficiently small to make subsequent analysis of its behaviour easy. This desideratum means that, for the MNIST setting, we can use the architecture described in Section 4.1.2.

For the Mountain Car and Acrobot settings, it is vital to keep in mind that ANNs often struggle with temporally-correlated data. Thus, rather than naively constructing architectures from scratch, we would be better-served by borrowing architectures from the literature that satisfy the above desideratum. For the Mountain Car setting, we can turn to Ghiassian et al. (2020). Their architecture for Mountain Car is similar to the architecture we are employing in the MNIST setting. The major differences between them, apart from the

necessary changes to the input and output layer, are that the architecture employed by Ghiassian et al. uses only 50 hidden units in the hidden layer and initializes their non-bias units with Xavier initialization. Despite these differences, the architecture they employ remains simple and modern, making it ideal for our purposes.

For the Acrobot setting, to assert the above desideratum, we can model our network architecture after Liu (2019). Doing so results in a network consisting of an input layer, an output layer, and two hidden layers with ReLU activation. The input layer of this network has 6 units, the first hidden layer has 32, the second hidden layer has 256, and the output layer has 1. To be consistent with Liu (2019), we must initialize the non-bias units with He initialization. However, as with the network we decided on for the Mountain Car setting, bias units can be initialized by sampling from a normal distribution with mean 0 and a standard deviation equal to 0.1.

6.1.2 Deciding on Step-size Adaptation Methods

To decide what step-size adaptation methods to include in our experiment, first recall that we are most interested in whether step-size adaptation methods affect catastrophic forgetting in ANNs if the answer affects a large body of contemporary research. Thus we must aim to select only step-size adaptation methods that remain prominent in the literature. Three such methods were mentioned in Section 2.4.1: SGD with Momentum, RMSProp, and Adam. While we must establish some basis by which we can believe that these could affect catastrophic forgetting to justify including them here, if we can do so, their prevalence makes them ideal for our purposes.

There are clear reasons to believe that all three of the above step-size adaptation methods could affect catastrophic forgetting in ANNs. The simplest reason is that all three of them maintain some form of a gradient trace that is employed when updating parameters. Thus, in an incremental setting, a single new sample will affect not only the immediate update but also several subsequent updates. Under decay theory (see Section 3.3), this rehearsal should reduce catastrophic forgetting. In a setting such as MNIST, however,

it is also possible that a change in task could result in these traces leading to an acceleration towards a set of parameters only optimal for the new task. Thus there is some reason to believe that each of the above three methods has, at least theoretically, both the potential to increase or decrease catastrophic forgetting.

We are interested in seeing if such an effect exists in practice, what direction it takes, and whether or not it is actually meaningful. To do so, for the MNIST setting, as in Section 4.1.3, we select one α for each of the above optimizers by trying each of 2^{-3}, 2^{-4}, ..., 2^{-18}. As the Mountain Car setting and Acrobot setting are likely to be harder for the ANN to learn, we instead select one α for each setting by instead trying each of 2^{-3}, $2^{-3.5}$, ..., 2^{-18}. For SGD with Momentum, we set β to 0.9: a commonly used value in the literature (Ruder, 2016, p. 4). For Adam, we fix β_1 to be 0.9 and β_2 to be 0.999 as recommended by Adam's creators (Kingma and Ba, 2014). To be consistent with Adam, for RMSProp, we set β to 0.999. While Hinton recommends setting β to 0.9 in RMSProp, we found that setting β to 0.9 severely harmed learning in both the Mountain Car and Acrobot setting. For RMSProp and Adam, we set ϵ to 10^{-8}.

As before, we ran each experiment with 50 different seeds to perform the α selection procedure. We then used the resulting α with 500 other seeds to generate the results reported in Section 6.2. In the case of the MNIST setting, the folds used in both training and testing for the hyperparameter selection were disjoint from the folds used later. For the Mountain Car and Acrobot setting, each run consisted of 500 episodes, as in Ghiassian et al. (2020). Each seed was additionally used to initialize the networks and, in the MNIST setting, permute the samples from the folds.

To verify the consistency of our conclusions under different hyperparameters, we repeat the above experiment setting with different values of β for SGD with Momentum and RMSProp. For SGD with Momentum, we try setting β to 0.81 and 0.9. For RMSProp, we try setting β to 0.81, 0.9, and 0.99. We also conduct a sensitivity analysis for α in the original experiment.

6.1.3 Variables of Interest

Unlike in Chapter 4, we do not formalize hypotheses here. We are interested here in simply observing the nature of the effect of step-size adaptation methods on catastrophic forgetting in ANNs and how well the testbed does in displaying this. It is not actually clear here what a formal, statistically-testable yet useful hypothesis would be to accomplish this at such a stage.

The above means that we are most interested in observing whether or not any of the metrics report a clear difference between the amount of forgetting exhibited by each of the step-size adaptation methods. For the retention case, this means the accuracy on $D_{(1+2)}$ directly after the second phase in the MNIST setting. For relearning, this means the length of the third phase as a function of the first phase in the MNIST setting, i.e., the speedup. For activation similarity, this means the activation similarity after each phase in the MNIST setting, and, in the Mountain Car and Acrobot setting, both an average activation similarity after each episode, and the final activation similarity after all episodes have elapsed. Note that the final activation similarity can be used to estimate what the average activation similarity would be if more episodes were run. For pairwise interference, we want to observe the same settings and summary groupings as with activation similarity. In reference to the above, we want to observe how each of the above metrics ranks the different methods and whether or not there is agreement between the metrics.

6.2 Results

We present our results in four sections. Sections 6.2.1, 6.2.2, and 6.2.3 gives the results pertaining to the MNIST, Mountain Car, and Acrobot settings, respectively. Section 6.2.4 then looks at how the above results change with different hyperparameters. We discuss all these results later in Section 6.3.

6.2.1 Reading Writing With MNIST

Table 6.1 provides the average number of steps each optimizer took to complete each phase. Of the four optimizers, RMSProp was consistently the fastest in

Optimizer	Phase 1	Phase 2	Phase 3	Phase 4
Adam	82.98±1.78	161.58±1.80	136.14±1.78	110.78±1.45
Momentum	135.88±2.86	192.18±2.38	155.03±2.67	116.55±1.90
RMSProp	**60.19±1.25**	**100.08±1.28**	**49.29±1.11**	**24.54±0.81**
SGD	105.67±2.26	120.82±1.97	52.12±1.51	29.81±0.90

Table 6.1: Average number of steps each of the four optimizers took to complete each phase. Smallest values are shown in bold.

each phase, though all the optimizers were able to complete each phase in a reasonable number of steps. Recall that $E_{(1,2,3,4)}$ from Chapter 4 took an order of magnitude longer to get through the first phase than anything shown here. Additionally, all the optimizers took more time, on average, to complete the second phase as compared to the first phase. Similarly, they all took less time to complete the fourth phase as compared to the second phase. Notably, though, only RMSProp and SGD took, on average, less time to complete the third phase as compared to the first phase.

Figure 6.1 shows the accuracies of the four optimizers over each phase. Clearly, changes in accuracies are relatively smooth in each phase. This figure also suggests that RMSProp and SGD seem to be forgetting at a distinctly slower rate than Adam or Momentum. To confirm that we can refer to Table 6.2, which provides the accuracy on both datasets after each phase. As expected, high accuracy is achieved by each of the optimizers on whatever dataset was presented in that phase. As previously defined, the retention metric is the accuracy on $D_{(1+2)}$ after the second phase. When comparing optimizers, RMSProp shows by far the highest retention. RMSProp is followed by SGD and then by both Adam and SGD with Momentum. Both showed similar retention. While outside the definition of retention, this trend is preserved when looking at the accuracy of the optimizers on $D_{(3+4)}$ after the third phase, and $D_{(1+2)}$ after the fourth phase. Also somewhat interesting, consistent with what we noted in Section 4.4, the accuracy on $D_{(1+2)}$ after the fourth phase is much higher than the accuracy on $D_{(1+2)}$ after the second phase.

Figure 6.2 compares the distribution of phase lengths for the first and third phases, and Table 6.3 provides the corresponding relearning metric, i.e., the

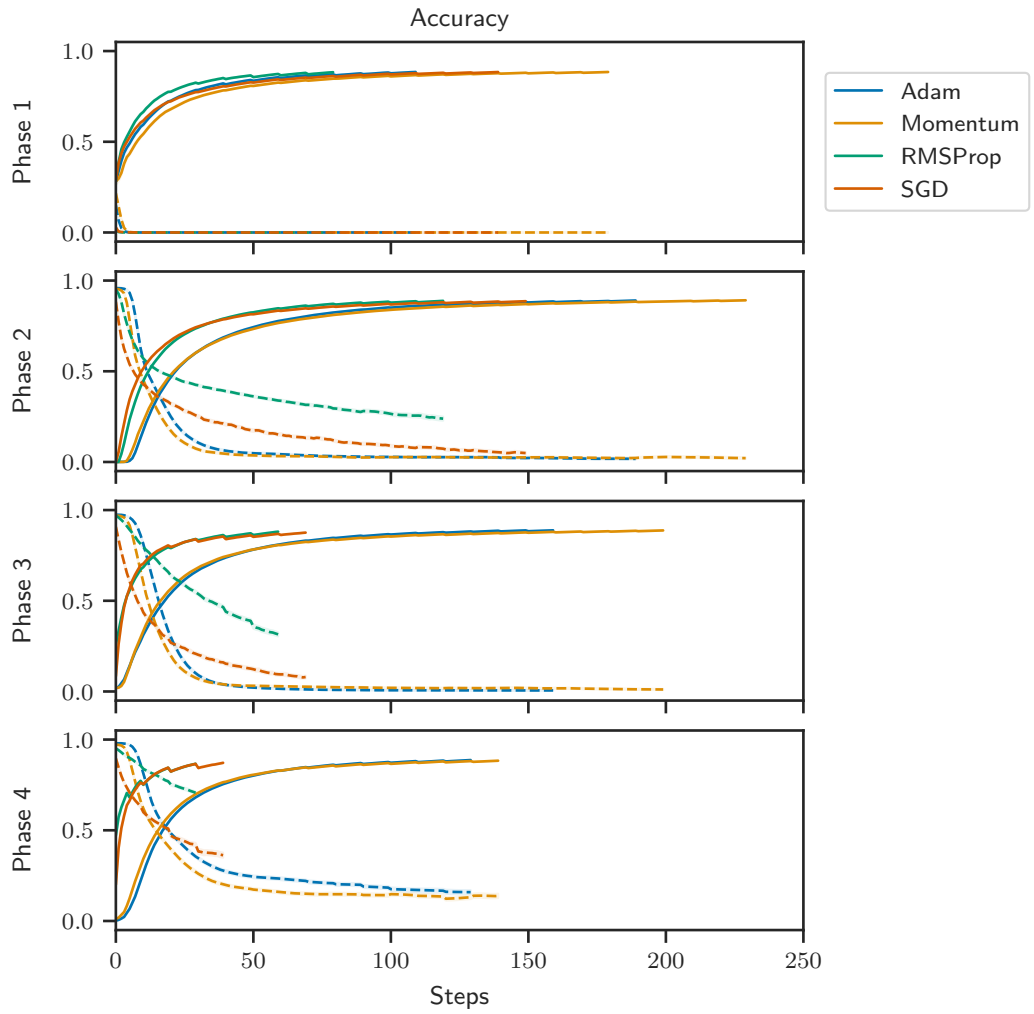

Figure 6.1: Performance of the four optimizers as a function of phase and step in phase in the MNIST setting. Lines are averages of all runs currently in that phase and are only plotted while at least half of the runs for a given optimizer are still in that phase. Solid lines show the current running accuracy of the learning system in that phase. Dashed lines show the accuracy on the test set corresponding to the task not in that phase. So in phases one and three, the dashed line is the accuracy on $D_{(3+4)}$. Likewise, in phases two and four, the dashed line is the accuracy on $D_{(1+2)}$. Standard error is shown with shading.

Optimizer	Phase	Accuracy on $D_{(1+2)}$	Accuracy on $D_{(3+4)}$
Adam	1	0.9580±0.0031	0.0000±0.0000
	2	**0.0200±0.0025**	0.9741±0.0007
	3	0.9790±0.0006	0.0059±0.0010
	4	0.2032±0.0068	0.9778±0.0006
Momentum	1	0.9533±0.0035	0.0000±0.0000
	2	**0.0177±0.0023**	0.9689±0.0010
	3	0.9701±0.0015	0.0171±0.0027
	4	0.1527±0.0068	0.9703±0.0011
RMSProp	1	0.9573±0.0032	0.0000±0.0000
	2	**0.2635±0.0061**	0.9756±0.0005
	3	0.9615±0.0028	0.4645±0.0101
	4	0.7767±0.0062	0.9362±0.0025
SGD	1	0.9570±0.0032	0.0000±0.0000
	2	**0.0768±0.0048**	0.9711±0.0009
	3	0.9638±0.0024	0.1931±0.0093
	4	0.5123±0.0093	0.9564±0.0018

Table 6.2: Accuracy on each test dataset in the MNIST setting directly after completing a phase as a function of the optimizer. Values shown in bold represent the retention metric.

Optimizer	Relearning
Adam	0.67±0.02
Momentum	1.18±0.07
RMSProp	1.80±0.09
SGD	3.43±0.18

Table 6.3: Length of the first phase as a function of the third under each optimizer in the MNIST setting. These values represent the relearning metric.

average ratio between the length of the third phase and the first phase. While RMSProp may have been dominant under the retention metric, SGD is dominant here. When considering this, note the differences in the variability and size of the first phase lengths under RMSProp and SGD visible in Figure 6.2. Further on the differences between this and our observations relating to the retention metric, a clear gap between Adam and SGD with Momentum is visible here. To this point, Adam actually has a relearning speed slower than its original learning.

Figure 6.3 shows the activation similarity and pairwise interference for the four optimizers as a function of phase and step in phase. Recall that, for

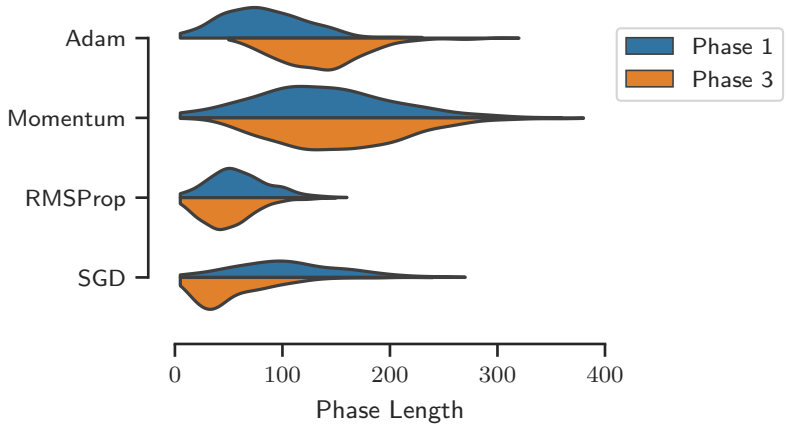

Figure 6.2: Split violin plot showing the distribution of the lengths of the first and third phase as a function of the optimizer. The width of a bar at each point provides an estimate for how frequently the optimizer will take the corresponding number of steps to complete the phase.

both activation similarity and pairwise interference, a high value indicates a high degree of forgetting or susceptibility to forgetting. Consistent with the retention and relearning metric, Adam exhibits the highest amount of activation similarity here. However, in contrast to the retention and relearning metric, RMSProp seems to exhibit the second highest. Only minimal amounts are displayed with both SGD and SGD with Momentum.

When compared with activation similarity, looking at the pairwise interference reported here produces less straightforward analysis. In the first phase Adam and SGD with Momentum exhibit less pairwise interference than SGD and RMSProp. This ordering is then reversed in the later phases in which SGD consistently displays less pairwise interference than RMSProp. RMSProp, in turn, displays less pairwise interference than Adam and SGD with Momentum. When resolving this, it is useful to recall that the weights in the first phase are initialized randomly, which results in a somewhat different meaning for pairwise interference in the first phase compared to the later phases.

6.2.2 Rocking up the Hill With Mountain Car

Figure 6.4 shows the post-episode performance, activation similarity, and pairwise interference for each of the four optimizers in the Mountain Car setting.

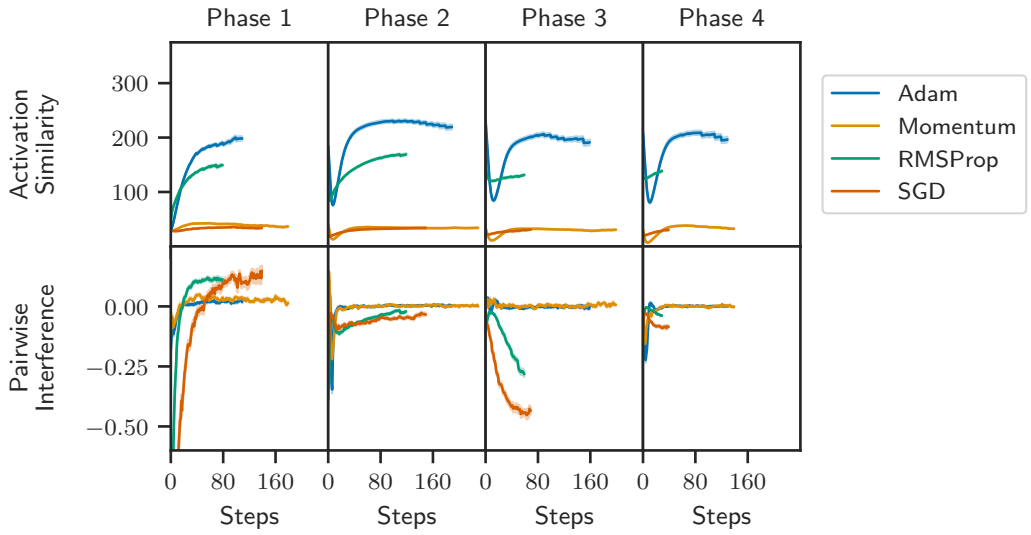

Figure 6.3: Activation similarity and pairwise interference exhibited by the four optimizers as a function of phase and step in phase in the MNIST setting. Lines are averages of all runs currently in that phase and are only plotted for steps where at least half of the runs for a given optimizer are still in that phase. Standard error is shown with shading.

To determine whether or not the optimizer has been able to achieve an acceptable performance in Mountain Car, we compare their performance to the performance of a learning system that, at any given step, outputs a prediction equal to the average return so far observed up to that step, i.e., a constant predictor. When a learning system outperforms a constant predictor, it means that the learning system is making use of the input it receives when generating an output. Here, all of the four optimizers were able to consistently outperform the constant predictor in later episodes.

Table 6.4 shows the mean and final post-episode values for activation similarity and pairwise interference. As shown here and in the preceding figure, Adam exhibited both the highest mean and final activation similarity. As such, Adam was, on average, susceptible to the most catastrophic forgetting both during learning and after learning. In contrast, SGD with Momentum exhibited both the least mean activation similarity and the least final activation similarity. This ranking vaguely corresponds to the ordering induced by activation similarity under the MNIST setting. However, a noticeable gap has

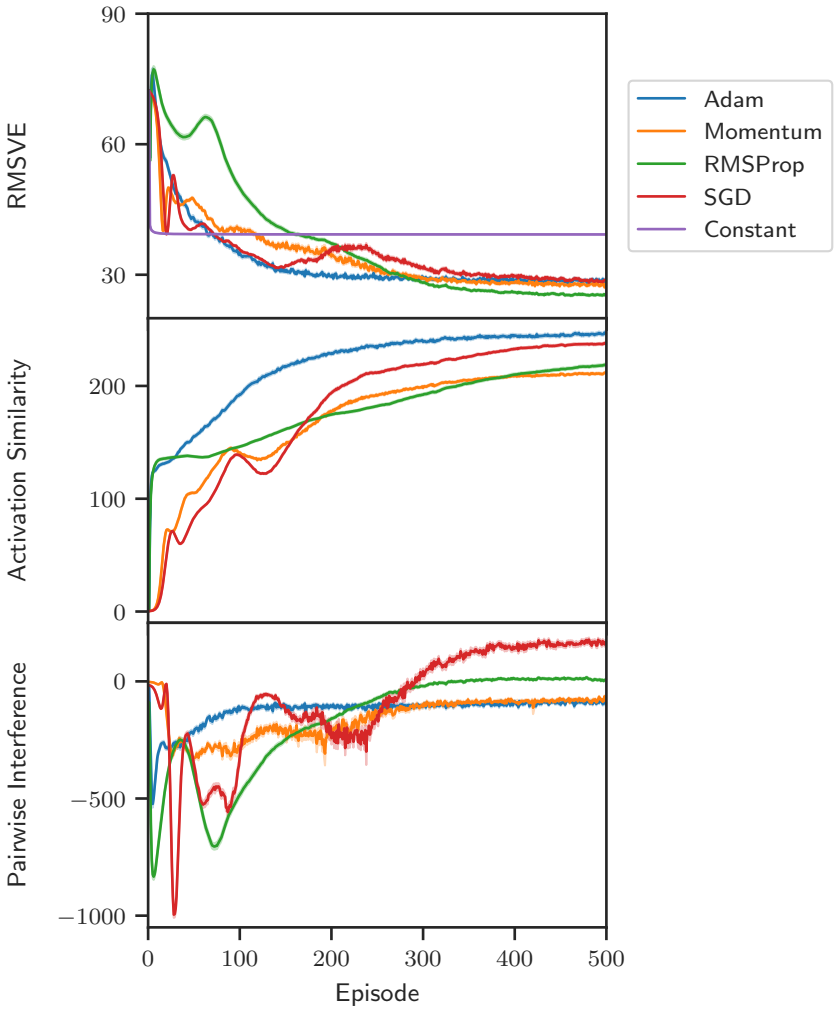

Figure 6.4: Performance and interference metrics for the four optimizers as a function of episode in the Mountain Car setting. Lines are averages of all runs, and standard error is shown with shading.

Metric	Optimizer	Mean Value	Final Value
Activation Similarity	Adam	218.32±0.90	247.32±1.32
	Momentum	**170.64±0.40**	**211.50±0.78**
	RMSProp	179.36±0.63	218.73±1.03
	SGD	180.50±0.45	237.93±0.83
Pairwise Interference	Adam	-124.82±0.88	**-84.27±5.28**
	Momentum	-156.11±1.20	**-80.05±4.97**
	RMSProp	**-168.70±2.80**	10.32±3.33
	SGD	-76.75±3.02	170.77±8.64

Table 6.4: Average and final post-episode activation similarity and pairwise interference as a function of the optimizer in the Mountain Car setting. Smallest values are shown in bold.

formed between SGD and SGD with Momentum. Additionally, while SGD exhibited slightly less mean activation similarity than RMSProp, looking at Figure 6.4, it is clear that this is transient. Thus we can safely conclude RMSProp exhibits more activation similarity in the Mountain Car setting than SGD, further disputing the ranking under activation similarity in the MNIST setting.

When considering pairwise interference, the mean and final values have less correspondence than with activation similarity. Looking at the mean pairwise interference, one should note that Figure 6.4 shows that pairwise interference did not change smoothly in early episodes. This somewhat diminishes the utility of looking at the mean pairwise interference as such values are then highly dependent on the number of episodes. When looking at only later episodes or final pairwise interference, SGD experienced the most, and SGD with Momentum and Adam tied for experiencing the least. This is the first instance so far where Adam has been meaningfully ranked as experiencing the least catastrophic forgetting.

6.2.3 Defying Gravity With Acrobot

Figure 6.5 shows the post-episode performance, activation similarity, and pairwise interference for each of the four optimizers in the Acrobot setting. Like with Mountain Car, we compare the performance of the four optimizers to the performance of a constant predictor. Here, all the optimizers consistently

Metric	Optimizer	Mean Value	Final Value
Activation Similarity	Adam	46.88±0.24	72.69±0.41
	Momentum	69.44±0.32	81.06±0.84
	RMSProp	**34.42±0.23**	**49.54±0.38**
	SGD	50.33±0.20	69.67±0.36
Pairwise Interference	Adam	-287.47±4.27	-64.52±2.55
	Momentum	-861.73±6.41	**-562.92±17.96**
	RMSProp	-398.41±8.07	-74.13±12.39
	SGD	**-1150.31±12.37**	-463.38±16.35

Table 6.5: Average and final post-episode activation similarity and pairwise interference as a function of the optimizer in the Acrobot setting. Smallest values are shown in bold.

outperform the constant predictor in later episodes but vary quite substantially in the magnitude by which they do so. Throughout training, RMSProp seemed to outperform the other optimizers, and SGD with Momentum seemed to struggle the most.

Table 6.5 shows the mean and final post-episode values for both activation similarity and pairwise interference. High consistency can be observed between the mean and final values here. As also visible in Figure 6.5, SGD with Momentum exhibits both the highest mean and final activation similarity, whereas RMSProp exhibits the least. Adam and SGD exhibit similar amounts. This ordering disagrees with every previous ordering induced by activation similarity where Adam was consistently the worst.

When looking at pairwise interference in the Acrobot setting, much less episode-to-episode variability is observed than in the Mountain Car setting. This reduction in variability may correspond to the lower variability seen in RMSVE early on. With respect to the final pairwise interference, SGD and SGD with Momentum seemed to exhibit the least followed by RMSProp and Adam, which exhibited similar amounts. Moving to mean pairwise interference, the ordering remains generally preserved, but some gaps appear, causing SGD to experience less mean pairwise interference than SGD with Momentum and RMSProp to experience less mean pairwise interference than Adam.

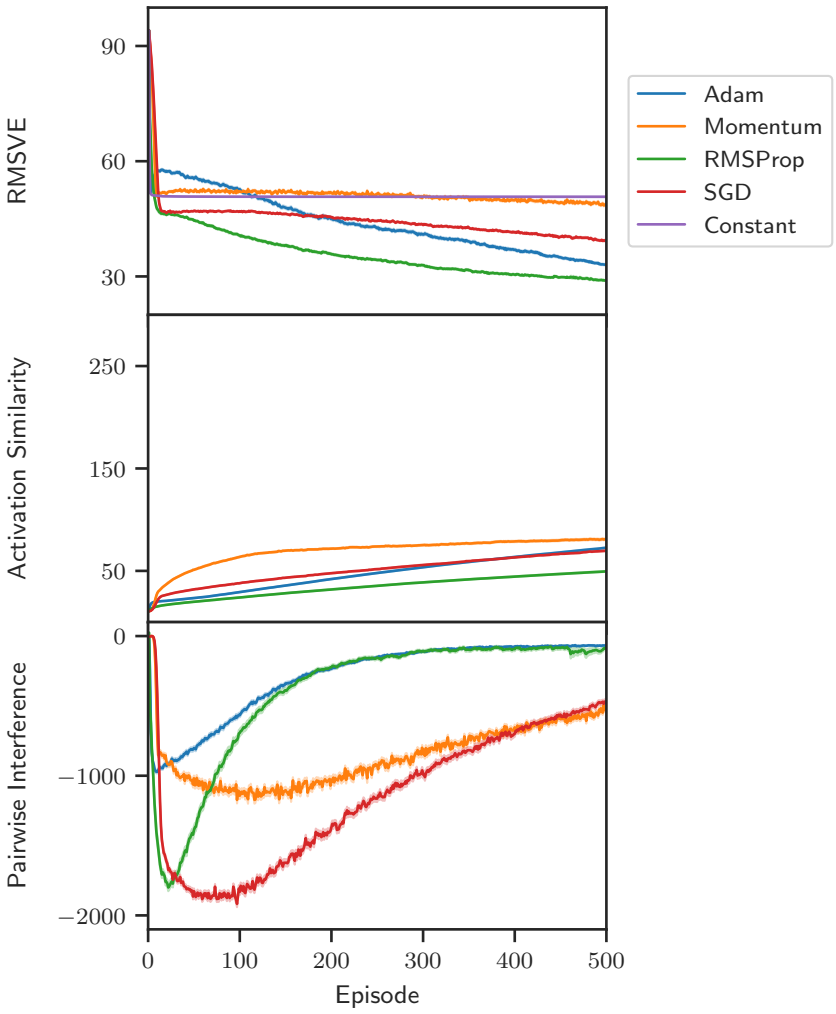

Figure 6.5: Performance and interference metrics for the four optimizers as a function of episode in the Acrobot setting. Lines are averages of all runs, and standard error is shown with shading.

6.2.4 The Effect of Hyperparameters

The results shown in the previous sections show the behaviour of the optimizers under one set of hyperparameters. With the exception of α, the hyperparameters were all set to fixed values that align with contemporary practice. α was selected by sweeping over multiple values and selecting the one that performed the best in the given setting under the given optimizer. In MNIST, this meant moving through the four phases the fastest, and, in Mountain Car and Acrobot, this meant minimizing the area under the RMSVE curve.

While we were careful to select hyperparameters to ensure they represent a standard setup, it is valuable to consider how sensitive these results are to minor variations in these hyperparameters. That being said, it is infeasible to empirically analyze the effect of every hyperparameter here: there are thousands of network configurations that have been explored in recent works. For this reason, we limit our investigation to α and β. That is to say, we explore the sensitivity of the above results for the four optimizers with respect to α and the sensitivity of the above results for SGD with Momentum and RMSProp with respect to β. While we do not analyze the effect of β_1 and β_2, note that both of these hyperparameters are respectively related to β in SGD with Momentum and RMSProp. Furthermore, variations in β_1 and β_2 are relatively uncommon in, at least, the supervised learning literature.

We begin by looking at the effect of β. Table 6.6 shows how the retention and relearning metrics in the MNIST setting vary as β changes. Under SGD with Momentum, there is a clear trend showing retention and relearning increasing as β tends to lower values. Recall that a lower β brings SGD with Momentum closer to SGD. Under RMSProp, this trend as it pertains to the retention metric is reversed with higher values of β leading to higher retention. Recall that RMSProp previously showed the highest rate of retention, followed by SGD. Looking at relearning under RMSProp does not produce a clear message, though, as both high and low values of β seem to produce worse relearning. Again, recall that SGD previously showed the fastest relearning time, followed by RMSProp.

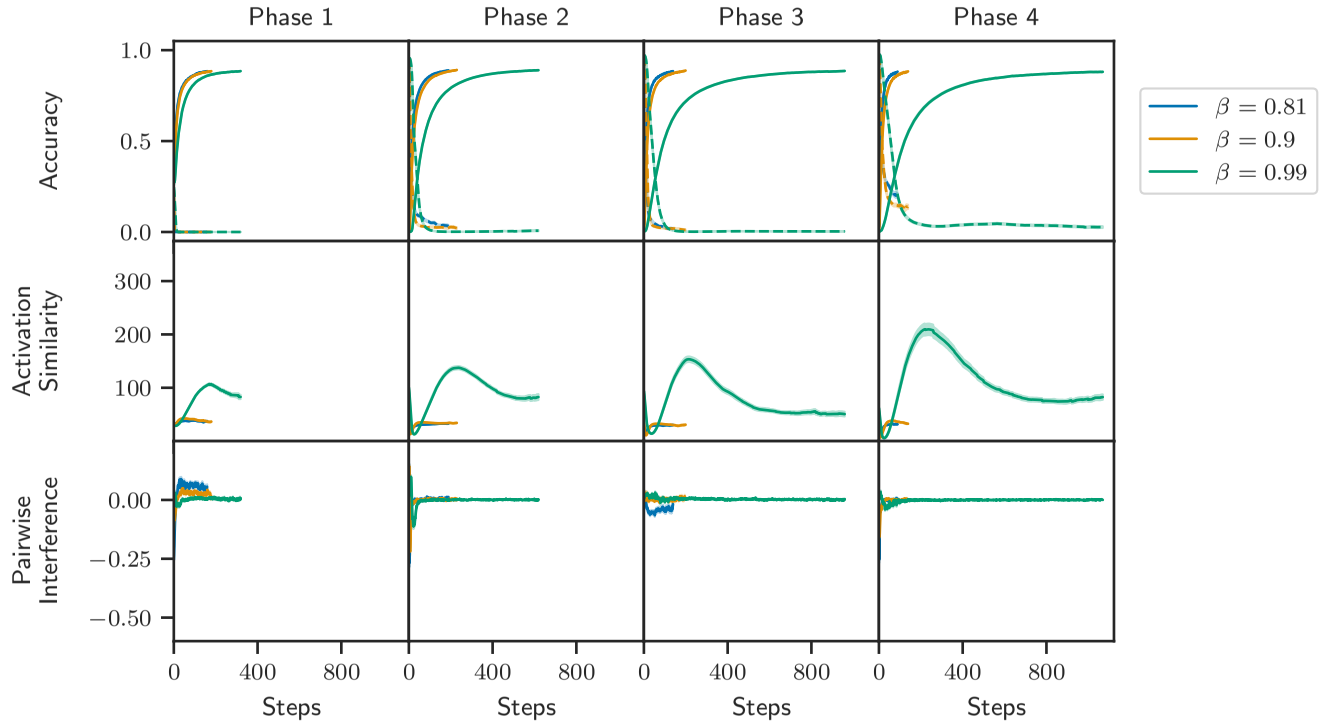

Figure 6.6: Accuracy, activation similarity, and pairwise interference exhibited by SGD with Momentum under different values of β as a function of phase and step in phase in the MNIST setting. Lines are averages of all runs currently in that phase and are only plotted while at least half of the runs for a given optimizer are still in that phase. Standard error is shown with shading.

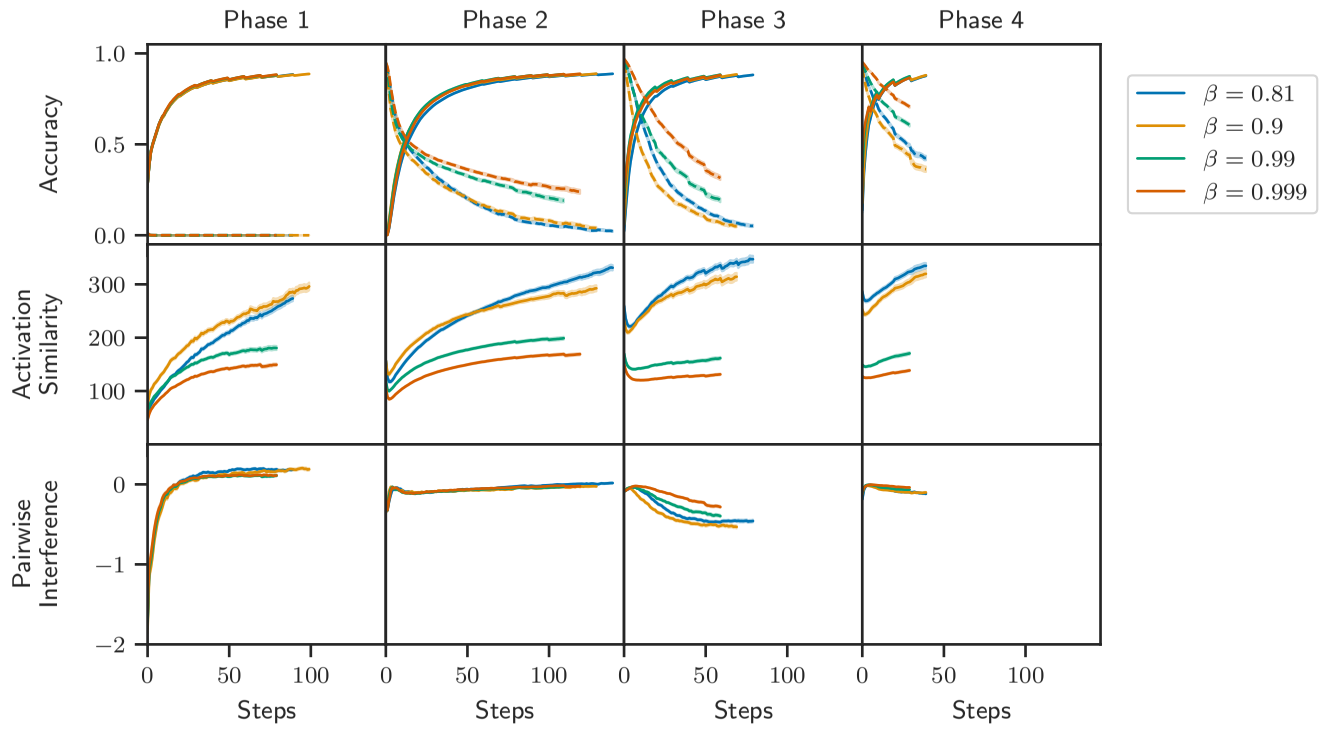

Figure 6.7: Accuracy, activation similarity, and pairwise interference exhibited by RMSProp under different values of β as a function of phase and step in phase in the MNIST setting. Lines are averages of all runs currently in that phase and are only plotted while at least half of the runs for a given optimizer are still in that phase. Standard error is shown with shading.

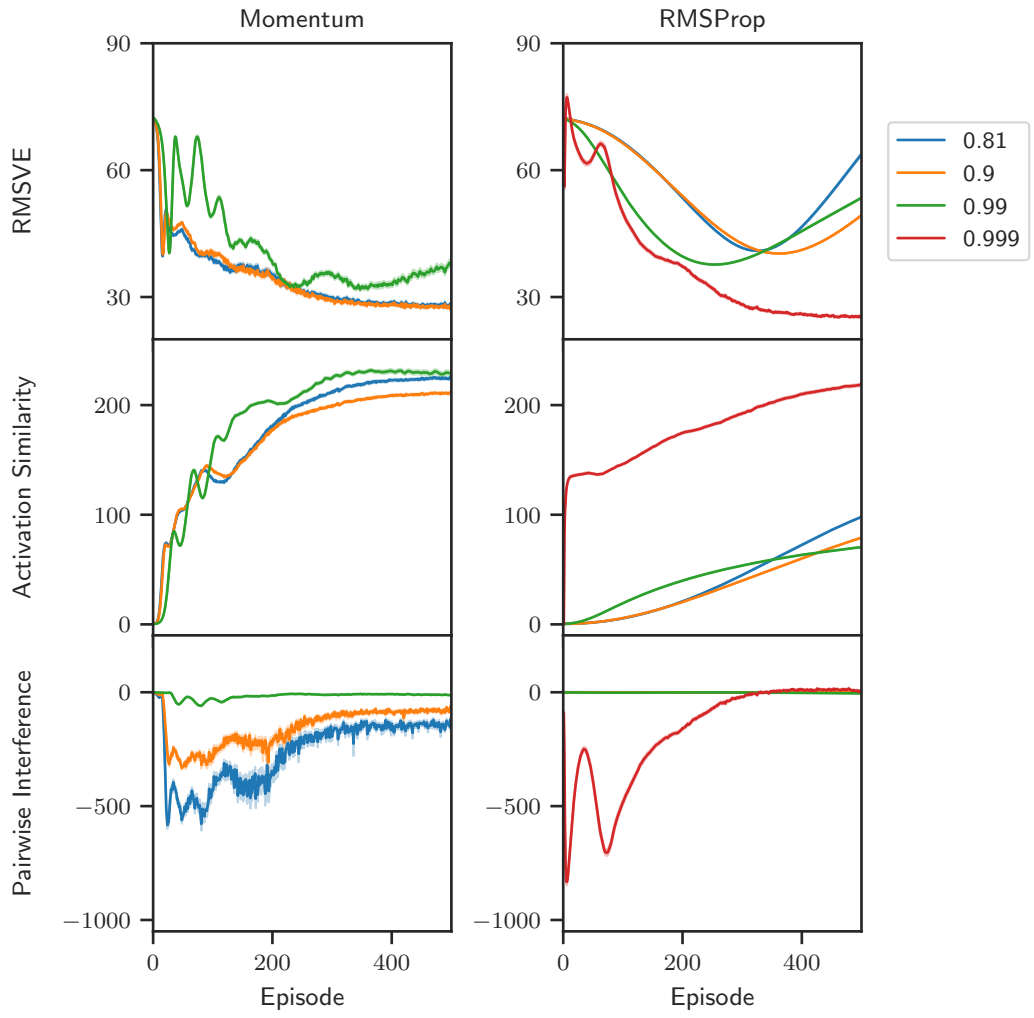

Figure 6.8: Performance and interference metrics for SGD with Momentum and RMSProp as a function of β and episode in the Mountain Car setting. Lines are averages of all runs, and standard error is shown with shading.

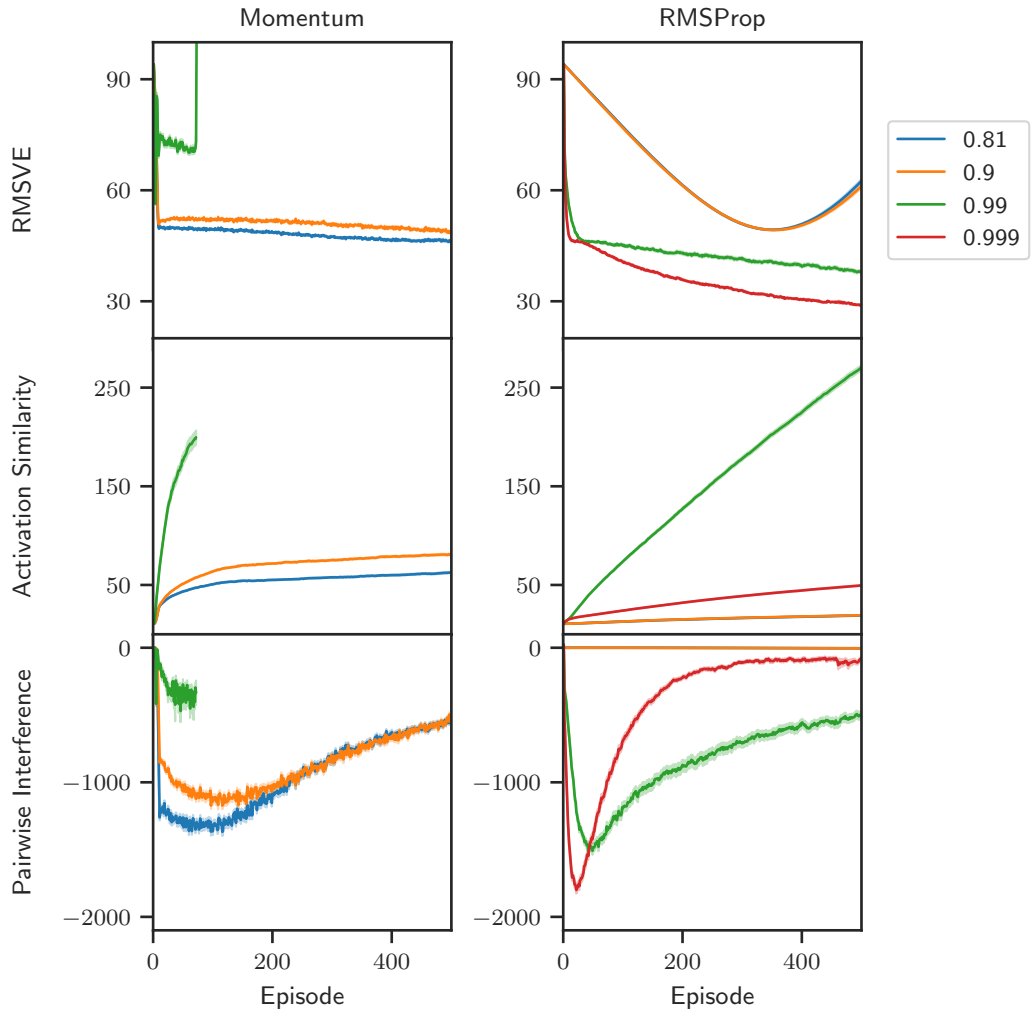

Figure 6.9: Performance and interference metrics for SGD with Momentum and RMSProp as a function of β and episode in the Acrobot setting. Lines are averages of all runs, and standard error is shown with shading. Note here that some test runs using SGD with Momentum under $\beta = 0.99$ experienced numerical instability. Lines for activation similarity and pairwise interference corresponding to such instability are only plotted up to the first such an occurrence.

Optimizer	β	Retention	Relearning
Momentum	0.81	**0.0378±0.0035**	**1.5635±0.0577**
	$\star$ 0.9 $\star$	0.0177±0.0023	1.1817±0.0749
	0.99	0.0032±0.0010	0.3553±0.0115
RMSProp	0.81	0.0336±0.0034	1.5068±0.0685
	0.9	0.0687±0.0045	**2.1629±0.1016**
	0.99	0.2263±0.0065	2.0151±0.0911
	$\star$ 0.999 $\star$	**0.2635±0.0061**	1.8019±0.0851

Table 6.6: Retention and relearning metrics exhibited by SGD with Momentum under different values of β in the MNIST setting. Smallest values are shown in bold. Values corresponding to the results shown in Sections 6.2.1, 6.2.2, and 6.2.3 are starred.

Figures 6.6 and 6.7 show how the accuracy, activation similarity, and pairwise interference vary as β changes in the MNIST setting for SGD with Momentum and RMSProp, respectively. For SGD with Momentum, it is clear that higher values of β produce slower learning and, consistent with retention and relearning, worse activation similarity. Simultaneously, lower values of β tended to produce little difference in learning speed but markedly higher rates of activation similarity. However, under both SGD with Momentum and RMSProp, no meaningful change appears in the pairwise interference when β changed.

Moving onto the Mountain Car setting, Figure 6.8 shows how β affects the RMSVE, activation similarity, and pairwise interference under both SGD with Momentum and RMSProp. For SGD with Momentum, β seems to play little roll in the activation similarity exhibited by the system, yet simultaneously lower values do seem to produce much lower amounts of pairwise interference. Looking at RMSProp is more challenging as only under $\beta = 0.999$ was it able to outperform the constant predictor consistently. Thankfully, little difference is observed in the interference metrics between $\beta = 0.81$, $\beta = 0.9$, and $\beta = 0.99$.

Figure 6.9 shows the same data as Figure 6.8 but for the Acrobot setting. Notably, unlike in Mountain Car, here, SGD with Momentum encountered numerical instability in one or more of the runs. This missing data somewhat hampers interpretation of the results. Despite that, there are still clear trends visible here that higher values of β produced more activation similarity and

more pairwise interference under SGD with Momentum. Under RMSProp, as before, the learning system failed to consistently outperform the constant predictor with certain values of β. Looking at only $\beta = 0.99$ and $\beta = 0.999$, higher values of beta seemed to be associated with less activation similarity and more pairwise interference. The non-convergent values of $\beta = 0.81$ and $\beta = 0.9$ seem to contradict this though. Thus no clear conclusion can be drawn here.

All of the optimizers appearing in this chapter maintain a hyperparameter α, which either served as a step-size or pseudo step-size. The value of α can be roughly described as what the magnitude of an update should be after observing some quantity of error. As such, α has a major effect on the performance of the learning system. We previously selected a value of α for each of the optimizers using the standard method: performing a sweep of different values and selecting the value according to what performed the best. Despite this, it would be useful to know whether or not small variations in α would produce different conclusions. This would verify to what degree different granularity sweeps would affect what we report here.

To begin, Figure 6.10 shows how the average post-phase accuracy and the length of phases varies with α in the MNIST setting. As with later results, some optimizer-α pairs were not able to effectively solve the problem, and so are omitted. For the pairs that did, we can observe that, while drastically different values of α produced widely varying behaviours, similar values of α tended to present similar behaviour. Figure 6.11 shows that this extends to the retention and relearning metrics. Interestingly, the shapes of the curves for each optimizer under both retention and relearning were similar. Here, SGD with Momentum seemed to only slightly be affected by α, but a pronounced effect with varying patterns is visible for each of the other three optimizers.

Again in the MNIST setting, Figure 6.12 shows how the final post-phase activation similarity and pairwise interference varied with α. Again here there is local smoothness under α. However, while activation similarity showed clear patterns of either minimal dependency on α or a strong positive correlation with α, more erratic patterns appear with pairwise interference. Most promi-

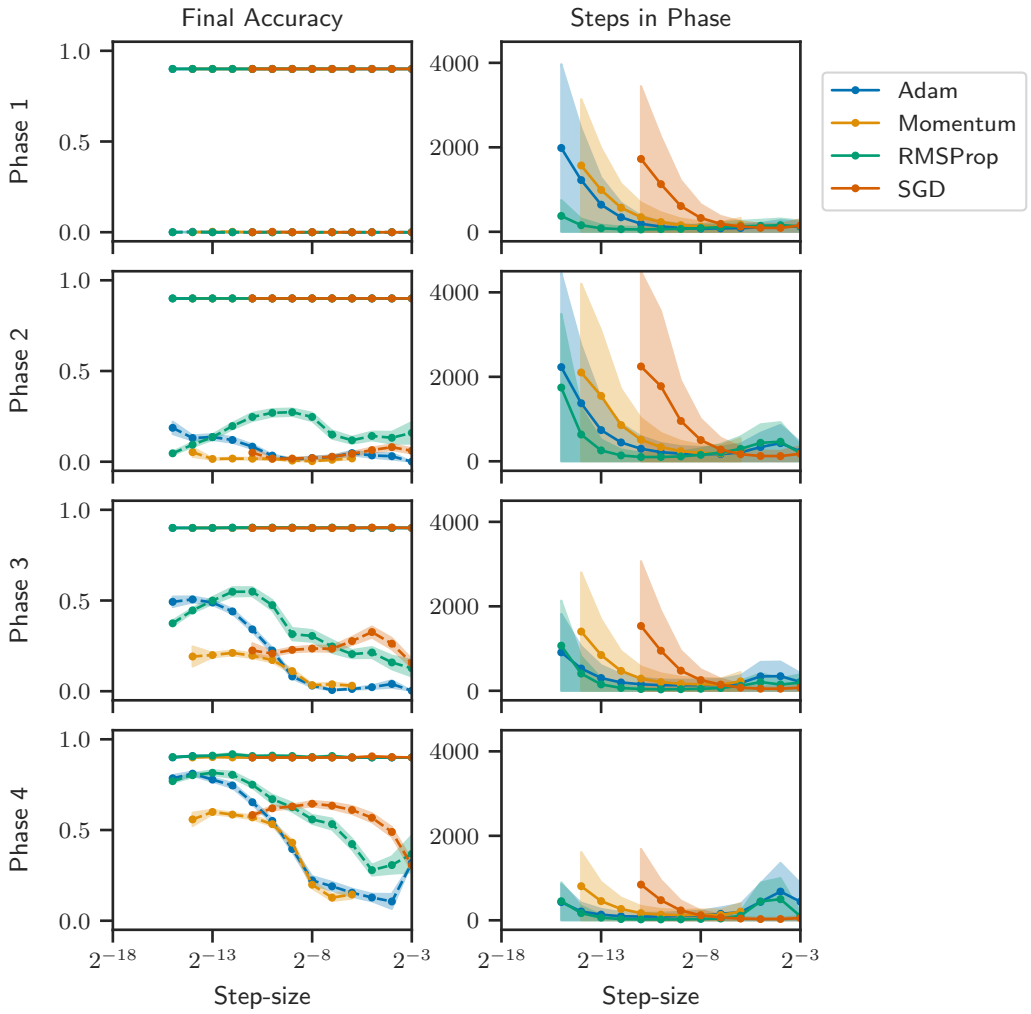

Figure 6.10: Final accuracy and number of steps needed to complete each phase in the MNIST setting for each of the four optimizers as a function of α. Other hyperparameters were set as they were in Figure 6.1. Lines are averages of all runs, and standard error is shown with shading. For final accuracy, solid lines show the running accuracy of the learning system in that phase and dashed lines show the accuracy on the test set corresponding to the task not in that phase. Note that in the final accuracy for the first phase, all solid lines overlap, and all dashed lines overlap. Lines are only drawn for values of α in which no run under the optimizer resulted in numerical instability.

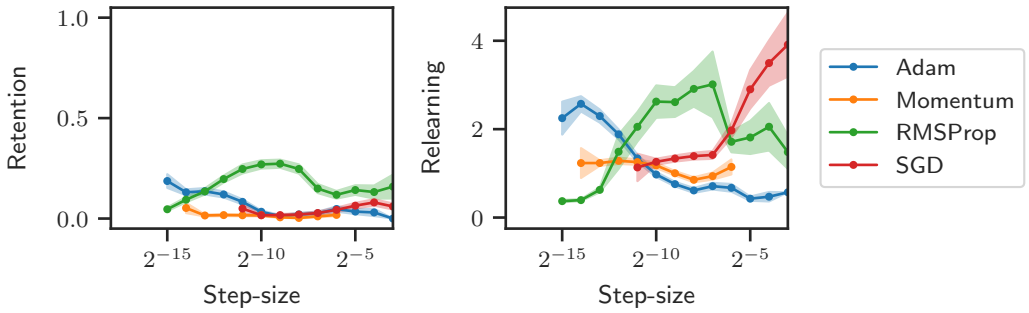

Figure 6.11: Retention and relearning metrics in the MNIST setting for each of the four optimizers as a function of α. Other hyperparameters were set as they were in Figure 6.1. Lines are averages of all runs, and standard error is shown with shading. Lines are only drawn for values of α in which no run under the optimizer resulted in numerical instability.

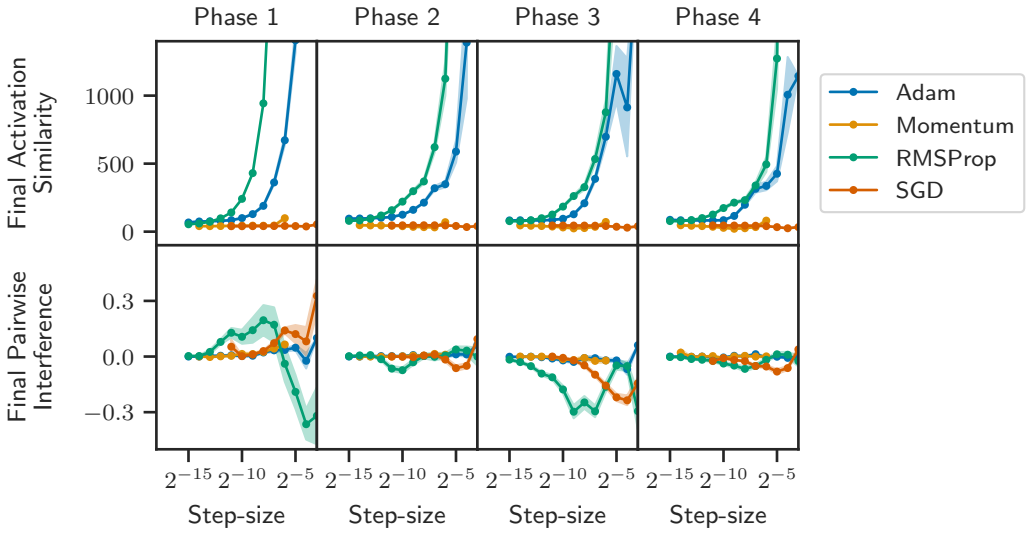

Figure 6.12: Final activation similarity and pairwise interference in the MNIST setting for each of the four optimizers as a function of α. Other hyperparameters were set as they were in Figure 6.1. Lines are averages of all runs, and standard error is shown with shading. Lines are only drawn for values of α in which no run under the optimizer resulted in numerical instability.

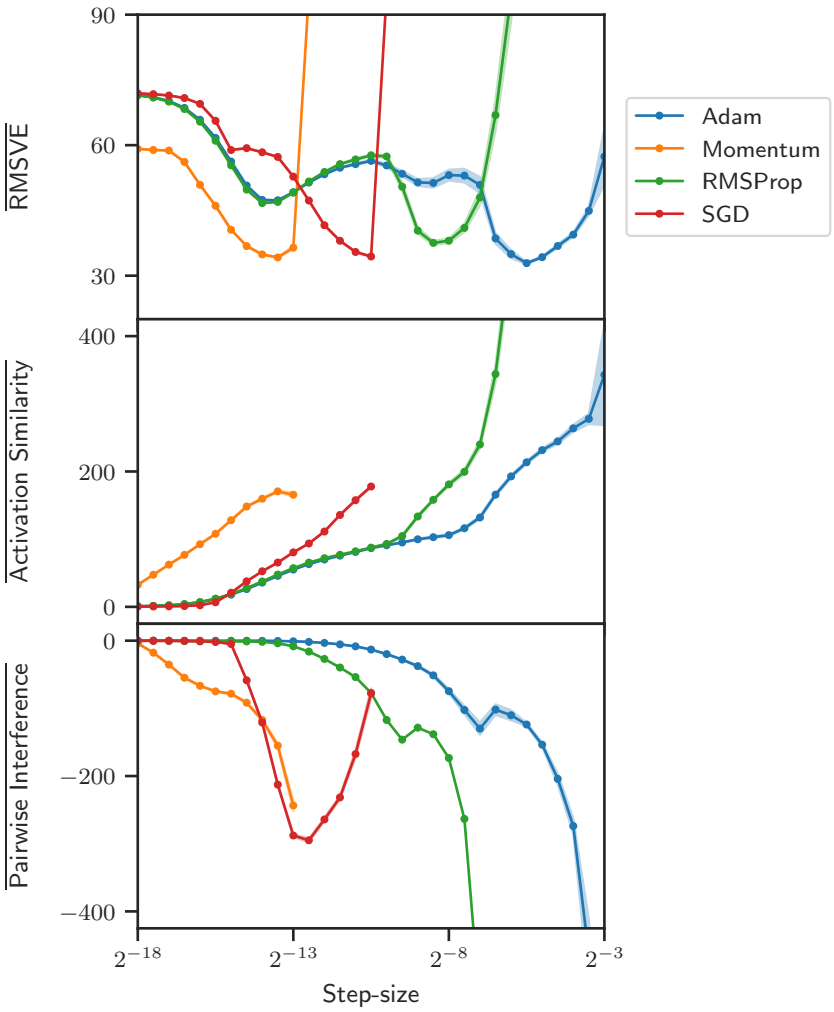

Figure 6.13: Mean performance and interference metrics in the Mountain Car setting for each of the four optimizers as a function of α. Other hyperparameters were set as they were in Figure 6.4. Lines are averages of all runs, and standard error is shown with shading. Both SGD and SGD with Momentum encountered numerical instability issues with certain values of α. Lines for activation similarity and pairwise interference are drawn so as to exclude these values.

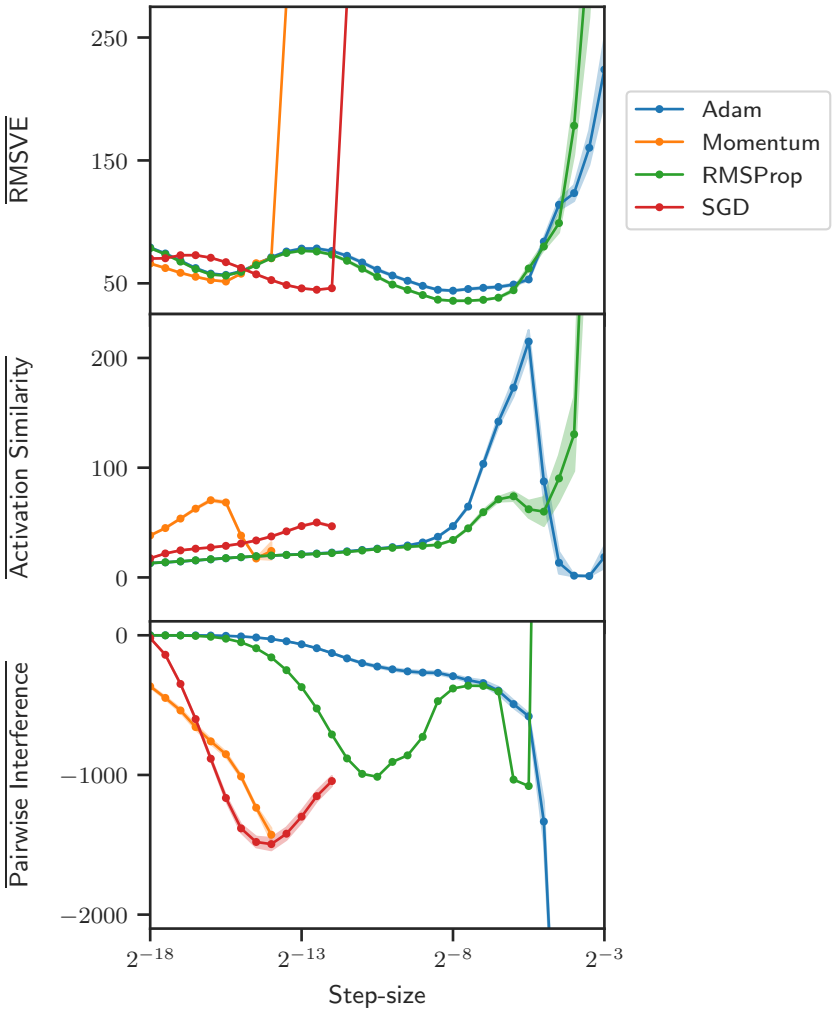

Figure 6.14: Mean performance and interference metrics in the Acrobot setting for each of the four optimizers as a function of α. Other hyperparameters were set as they were in Figure 6.5. Lines are averages of all runs, and standard error is shown with shading. Both SGD and SGD with Momentum encountered numerical instability issues with certain values of α. Lines for activation similarity and pairwise interference are drawn so as to exclude these values.

nently, in the first phase, RMSProp swaps between minimal pairwise interference, to positive pairwise interference, then to negative pairwise interference as α grows.

For the Mountain Car setting, Figure 6.13 shows how the mean RMSVE, activation similarity, and pairwise interference varied with α. As we observed when looking at β before, some of the optimizers experienced numerical instability under certain values of α. These cases are omitted here but are consistent with previous work (Ghiassian et al., 2020, pp. 441–443). Like with the MNIST setting, little local differences appear with regards to the mean RMSVE and the mean catastrophic forgetting measures, but major variations can be observed with drastically varying values of α. For activation similarity, we can observe that higher values of α were associated with a higher mean value. In contrast, higher values of α also seemed to produce lower values of pairwise interference for all the optimizers, with the exception of SGD.

Finally, Figure 6.14 provides the same data as Figure 6.13 but for the Acrobot setting. Immediately obvious here is that changes to α seemed to produce much of the same behavioural changes with respect to RMSVE as with the Mountain Car setting. However, less clear trends are visible for the mean catastrophic forgetting measures. Thankfully, local consistency still appears to be mostly preserved outside of extremely large values of α.

6.3 Discussion

The results provided in Section 6.2 allow us to reach several conclusions. First and foremost, as we observed a number of differences between the different step-size adaptation methods over a variety of metrics and in a variety of settings, we can safely conclude that there can be no doubt that step-size adaptation does have a meaningful and large effect on catastrophic forgetting in ANNs. The generic but careful construction of the experiments implies that this effect is likely impacting a large amount of contemporary research in the area.

As shown by Table 6.7, in most of our experiments, Adam appeared to be

amongst the most vulnerable to catastrophic forgetting. We hypothesize that Adam's high rate of forgetting may be a consequence of Adam being loosely defined as a union of SGD with Momentum and RMSProp. As a unification of these two methods, Adam may be particularly vulnerable when either of the methods is particularly vulnerable. This conjecture aligns with our observations where, in many of the previous results, either RMSProp or SGD with Momentum was particularly vulnerable to catastrophic forgetting, and Adam's behaviour often vaguely matched the worse one, e.g., see Figure 6.3. However, Adam's implementation here includes a bias correction mechanism usually skipped over in implementations of SGD with Momentum and RMSProp. Thus, further inquiry is needed to formally confirm or refute this, and, for now, it remains conjecture.

When looking at SGD with Momentum and RMSProp under different values of β, we saw evidence that β has a pronounced effect on the amount of catastrophic forgetting an algorithm experienced. In most cases, this dependence was smooth, i.e., similar values of β produced similar results. We also noted that the optimizer seemed to play a more substantial effect here, e.g., the best retention and relearning scores for SGD with Momentum we observed were still only roughly as good as the worst such scores for RMSProp.

By observing the α selection procedure, we were able to obtain consistent, strong evidence that catastrophic forgetting has a major dependence on α. Thankfully, this dependence appears to be very smooth, given the optimizer, under most metrics and in most settings. This dependence still suggests that α is necessary to consider when measuring the amount of catastrophic forgetting experienced by an optimizer, and higher consideration should be given to sensitivity analysis in the future. Thankfully, our results also suggest that small differences in α can be disregarded as they are unlikely to have a profound effect on any of the metrics.

One metric that we explored was activation similarity. Recall that activation similarity is equivalent to the activation overlap metric proposed in French (1991) and is only named differently to avoid future confusion regarding the mathematical way we calculate it. In the Mountain Car and Acrobot setting,

Optimizer	Retention	Relearning	Activation Similarity			Pairwise Interference		
			MNIST	Mountain Car	Acrobot	MNIST	Mountain Car	Acrobot
Adam	=3	4	4	4	3	=3	**=1**	=3
Momentum	=3	3	**=1**	**1**	4	=3	**=1**	**1**
RMSProp	**1**	2	3	2	**1**	2	3	=3
SGD	2	**1**	**=1**	3	2	**1**	4	2

Table 6.7: Rough summary of how each of the optimizers was ranked under each metric and setting in Section 6.2. For Mountain Car and Acrobot, rankings under activation similarity and pairwise interference use their final values.

activation similarity showed properties of being a useful metric as rankings under activation similarity seemed to correspond better than under pairwise interference to rankings under RMSVE. However, while French (1991) argued that more activation overlap is the cause of catastrophic forgetting and so can serve as a viable metric for it (p. 173), in the MNIST setting activation similarity seemed to be in opposition to the well-established retention and relearning metrics. These results suggested that, while Adam suffers a lot from catastrophic forgetting, so too does RMSProp. Together, this suggests that catastrophic forgetting cannot be a consequence of activation similarity alone. Further studies must be conducted to understand why the unique representation learned by RMSProp here leads to it performing well on the retention and relearning metrics despite having a greater representational similarity.

On the consistency of the results, the variety of rankings we observed in Section 6.2 validate previous concerns regarding the challenge of measuring catastrophic forgetting. Between settings, as well as between different metrics in a single setting, vastly different rankings were produced. While each setting and metric was meaningful and thoughtfully selected, little agreement appeared between them. Thus, we can conclude that, as we hypothesized, catastrophic forgetting is a subtle phenomenon that cannot be characterized by only limited metrics or limited problems. Thus, it is insufficient to utilize scores on a single metric (or a single family of metrics) to conclude which of a set of metrics best mitigates catastrophic forgetting. One could go as far as to say that it is better to use such metrics to understand the nature of the catastrophic forgetting exhibited by a learning system rather than to show one algorithm's superiority over another in mitigating it. As forgetting should be considered, in general, unavoidable (see Section 3.1), research effort would be more effectively used by doing so.

Chapter 7

Conclusion

In this thesis, we sought to revisit and expand our understanding of what forgetting and catastrophic forgetting are in artificial neural networks (ANNs), a learning system loosely based on the brain that is behind several recent breakthroughs in artificial intelligence. We sought to do so by answering five questions of interest: How does forgetting in psychology relate to machine learning (ML) ideas? What is catastrophic forgetting? Does it exist in contemporary systems, and, if so, is it severe? How can we measure a system's susceptibility to it? Are the modern algorithms we use to train ANNs affecting the phenomenon?

To understand the relationship between forgetting in psychology and various ML ideas, **we examined how both fields think about forgetting**. We started by mapping out the origins of the study of forgetting in each field. We then noted that forgetting in both psychology and ML is not necessarily a negative phenomenon and, in fact, serves a critical role in any good learning system. Afterwards, we delved into how both decay theory and interference theory, two perspectives about forgetting in psychology, unexpectedly connect to several topics in ML, such as transfer learning. **We noted here that catastrophic forgetting originally referred to retroactive interference, or the phenomenon of new learning interfering with older learning.** We then connected this all to neuroscience research by noting how the nature of connections between neurons suggests that generalization is crucial in studying catastrophic forgetting.

To validate the existence and show forgetting and catastrophic forgetting is a problem worthy of further inquiry, we set up an experiment using a contemporary ML system. **We showed that catastrophic forgetting appears in large quantities, even when using a modern dataset, with a modern ANN trained with a modern algorithm, and measured with a principled metric.** We also noted here that the phenomenon of catastrophic forgetting was much more subtle than a simple reversal of learning. At the same time, we showed the value of having algorithms that could effectively control forgetting by showing that learning problems one at a time can lead to faster learning overall.

Following the above, we tried to understand how we can determine a system's susceptibility to catastrophic forgetting, and simultaneously determine if the modern algorithms we use to train ANNs are affecting this phenomenon. In doing so, **we built a testbed we could use to understand the nature of the catastrophic forgetting exhibited by a method.** We build this by first carefully assembling three data-streams from the MNIST dataset, the Mountain Car problem, and the Acrobot problem. This variety ensured that we covered explicitly multi-task settings as well as single-task settings with temporally correlated examples. We then assembled several different ways of measuring forgetting to ensure a diverse set of perspectives on catastrophic forgetting was represented. Afterwards, **we used this testbed to show that contemporary optimizations algorithms used to train ANNs are meaningfully affecting the amount of catastrophic forgetting occurring. We also showed that the nature of the catastrophic forgetting occurring differed and, consequently, showed that the phenomenon of catastrophic forgetting cannot be explained using only one existing metric.** As these algorithms we employed are widely used in modern research related to catastrophic forgetting, this result has implications for a large body of contemporary research.

In conclusion, this work sought to answer five questions about catastrophic forgetting so as to further our understanding of it. Altogether, we believe that it has succeeded in giving a satisfactory answer to each of the aforementioned

questions. As such, this work is offered as a step forward for the field of artificial intelligence.

7.1 Implications

The main implications of this work concern both the application of ANNs to problems susceptible to catastrophic forgetting, and the study of forgetting and catastrophic forgetting in ANNs. The results of this study suggest that catastrophic forgetting remains a severe problem in contemporary ANNs. They also suggest that when applying ANNs to problems susceptible to catastrophic forgetting, users should be wary of the optimization algorithm they use to train the network. This is especially true when that algorithm is Adam. It is less true when that algorithm is SGD.

In addition to the above, the results of this study also imply that, when studying forgetting or catastrophic forgetting in ANNs, a holistic perspective must be considered; we have shown that catastrophic forgetting is a subtle phenomenon and not an exact problem. This study builds on previous work to reinforce the conclusion that considering only limited metrics on limited settings can only produce very limited conclusions regarding its nature. The testbed provided here gives one way of partially overcoming this concern, as we have shown it can provide a more holistic view of forgetting and catastrophic forgetting than much previous work.

7.2 Future Work

It is important to recognize here that the results of this study, as with most studies, are not fully conclusive, and, as always, additional research should be conducted to verify further and farther understanding of the claims put forward here. In addition, much work remains to be done to apply the conclusions of this work so as to reach meaningful ends.

While we constructed a principled testbed that used various metrics to quantify catastrophic forgetting, we only applied it to answer whether one set of particular mechanisms affected catastrophic forgetting. Moreover, no

attempt was made to use the testbed to examine the effect of mechanisms specifically designed to mitigate catastrophic forgetting. The decision to not focus on such methods was made as Kemker et al. (2018) already showed that these mechanisms' effectiveness varies substantially as both the setting changes and the metric used to quantify catastrophic forgetting changes. Kemker et al., however, only considered the retention metric in their work, so some value exists in again comparing these methods using the testbed provided here.

Our results regarding the testbed remained limited in that they only tested ANNs with one hidden layer. Contemporary deep learning often utilizes networks with many–sometimes hundreds–of hidden layers. Ghiassian et al. (2020) showed that this might not be the most impactful factor in catastrophic forgetting (p. 444). However, how deeper networks affect the nature of catastrophic forgetting seems largely unexplored. Applying the provided testbed to this question would yield important insights for the field.

One final opportunity for future research lies in the fact that, while we explored several settings and multiple metrics for quantifying catastrophic forgetting, there are many other, more complicated settings, as well as several still-unexplored metrics which also quantify catastrophic forgetting, e.g., Fedus et al. (2020). Whether the results of this work extend to significantly more complicated settings remains an important open question, as is the question of whether or not these results carry over to the control case of the reinforcement learning problem.

7.3 Closing Thoughts

Clarke (1973) unintentionally described ANNs the best when they said that "any sufficiently advanced technology is indistinguishable from magic" (p. 21). Artificial intelligence has managed to enter a golden age by moulding some simple cognitive hypotheses into something suitable for the structure of our current computers. It is of no surprise to anyone, though, that such a translation has inevitably created learning systems with unexpected behaviour and, consequently, generated countless new scientific questions. This must be viewed

as a fantastic opportunity for new, impactful research. While the results of this thesis compound our belief that the challenge of building strong learning systems is a truly immense one, we must remember that progress on this challenge is not only of immense value to society but is also a species-wide objective defined in our very nature.

References

Abraham, W. C., & Robins, A. (2005). Memory retention – the synaptic stability versus plasticity dilemma. *Trends in Neurosciences*, *28*(2), 73–78. https://doi.org/10.1016/j.tins.2004.12.003.

Barnes, J. M., & Underwood, B. J. (1959). "Fate" of first-list associations in transfer theory. *Journal of Experimental Psychology*, *58*(2), 97–105. https://doi.org/10.1037/h0047507.

Bar-On, Y. M., Phillips, R., & Milo, R. (2018). The biomass distribution on earth. *Proceedings of the National Academy of Sciences*, *115*(25), 6506–6511. https://doi.org/10.1073/pnas.1711842115.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI gym*. arXiv. https://arxiv.org/abs/1606.01540.

Cacioppo, J. T., & Cacioppo, S. (2014). Social relationships and health: The toxic effects of perceived social isolation. *Social and Personality Psychology Compass*, *8*(2), 58–72. https://doi.org/10.1111/spc3.12087.

Canadian Institute for Health Information. (2019). *National health expenditure trends, 1975 to 2019*. https://www.cihi.ca/sites/default/files/document/nhex-trends-narrative-report-2019-en-web.pdf.

Chance, P. (2014). *Learning and behavior* (7th ed.). Cengage Learning.

Chen, Z., & Liu, B. (2018). *Lifelong machine learning* (2nd ed.). Morgan & Claypool Publishers. https://doi.org/gd8g2p.

Cigna. (2018). *Cigna U.S. loneliness index*. https://www.cigna.com/assets/docs/newsroom/loneliness-survey-2018-full-report.pdf.

Clarke, A. C. (1973). *Profiles of the future* (2nd ed.). Harper & Row.

DeJong, G., & Spong, M. W. (1994). Swinging up the acrobot: An example of intelligent control. *Proceedings of the 1994 American Control Conference*, *2*, 2158–2162. https://doi.org/10.1109/ACC.1994.752458.

Ebbinghaus, H. (1913). *Memory: A contribution to experimental psychology* (H. A. Ruger & C. E. Bussenius, Trans.). Teachers College Press. (Original work published 1885).

Farquhar, S., & Gal, Y. (2018). *Towards robust evaluations of continual learning*. arXiv. https://arxiv.org/abs/1805.09733.

Fedus, W., Ghosh, D., Martin, J. D., Bellemare, M. G., Bengio, Y., & Larochelle, H. (2020). *On catastrophic interference in atari 2600 games*. arXiv. https://arxiv.org/abs/2002.12499.

French, R. M. (1991). Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, 173–178. https://cognitivesciencesociety.org/wp-content/uploads/2019/01/cogsci_13.pdf.

Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image style transfer using convolutional neural networks. *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2414–2423. https://doi.org/10.1109/CVPR.2016.265.

Geramifard, A., Dann, C., Klein, R. H., Dabney, W., & How, J. P. (2015). RLPy: A value-function-based reinforcement learning framework for education and research. *Journal of Machine Learning Research*, *16*(46), 1573–1578. http://jmlr.org/papers/v16/geramifard15a.html.

Ghiassian, S., Rafiee, B., Lo, Y. L., & White, A. (2020). Improving performance in reinforcement learning by breaking generalization in neural networks. *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*, 438–446. http://ifaamas.org/Proceedings/aamas2020/pdfs/p438.pdf.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, *9*, 249–256. http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, *15*, 315–323. http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

Goodfellow, I. J., Bengio, Y., & Courville, A. C. (2016). *Deep learning*. MIT Press.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., & Bengio, Y. (2013). *An empirical investigation of catastrophic forgetting in gradient-based neural networks*. arXiv. https://arxiv.org/abs/1312.6211.

Gutstein, S., & Stump, E. (2015). Reduction of catastrophic forgetting with transfer learning and ternary output codes. *Proceedings of the 2015 International Joint Conference on Neural Networks*, 1–8. https://doi.org/10.1109/IJCNN.2015.7280416.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer. https://doi.org/10.1007/978-0-387-84858-7.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the 2015 IEEE International Conference on Computer Vision*, 1026–1034. https://doi.org/10.1109/ICCV.2015.123.

Hetherington, P. A., & Seidenberg, M. S. (1989). Is there 'catastrophic interference' in connectionist networks? *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, 26–33. https://

cognitivesciencesociety.org/wp-content/uploads/2019/01/cogsci_11.pdf.

Hinton, G. E., Srivastava, N., & Swersky, K. (n.d.). *RMSProp: Divide the gradient by a running average of its recent magnitude* [PDF slides]. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Jarrett, K., Kavukcuoglu, K., Ranzato, M., & LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? *Proceedings of the 2009 IEEE International Conference on Computer Vision*, 2146–2153. https://doi.org/10.1109/ICCV.2009.5459469.

Kemker, R., McClure, M., Abitino, A., Hayes, T. L., & Kanan, C. (2018). Measuring catastrophic forgetting in neural networks. *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 3390–3398. https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16410.

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization.* arXiv. https://arxiv.org/abs/1412.6980.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N. C., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., & Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, *114*(13), 3521–3526. https://doi.org/10.1073/pnas.1611835114.

Kornblith, S., Norouzi, M., Lee, H., & Hinton, G. E. (2019). Similarity of neural network representations revisited. *Proceedings of the 36th International Conference on Machine Learning*, *97*, 3519–3529. http://proceedings.mlr.press/v97/kornblith19a/kornblith19a.pdf.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791.

Lee, S., Kim, J., Jun, J., Ha, J., & Zhang, B. (2017). Overcoming catastrophic forgetting by incremental moment matching. *Advances in Neural Information Processing Systems*, *30*, 4652–4662. http://papers.nips.cc/paper/7051-overcoming-catastrophic-forgetting-by-incremental-moment-matching.pdf.

Lin, L. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*(3-4), 293–321. https://doi.org/10.1007/BF00992699.

Liu, V. (2019). *Sparse Representation Neural Networks for Online Reinforcement Learning* [Master's thesis, University of Alberta]. https://era.library.ualberta.ca/items/b4cd1257-69ae-4349-9de6-3feed2648eb1.

Liu, V., Kumaraswamy, R., Le, L., & White, M. (2019). The utility of sparse representations for control in reinforcement learning. *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, 4384–4391. https://doi.org/10.1609/aaai.v33i01.33014384.

Masse, N. Y., Grant, G. D., & Freedman, D. J. (2018). Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, *115*(44), E10467–E10475. https://doi.org/10.1073/pnas.1803839115.

McCarthy, J. (2007). *What is artificial intelligence?* John McCarthy's Home Page. http://www-formal.stanford.edu/jmc/whatisai.pdf.

McCloskey, M., & Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, *24*, 109–165. https://doi.org/10.1016/S0079-7421(08)60536-8.

McGeoch, J. A. (1932). Forgetting and the law of disuse. *Psychological Review*, *39*(4), 352–370. https://doi.org/10.1037/h0069819.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236.

Moore, A. W. (1990). *Efficient memory-based learning for robot control* [Doctoral dissertation, University of Cambridge]. https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-209.pdf.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning*, 807–814. http://www.icml2010.org/papers/432.pdf.

National Highway Traffic Safety Administration. (2019). *2017 quick facts*. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812747.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. Wiley. https://doi.org/10.1002/9780470316887.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, *12*(1), 145–151. https://doi.org/10.1016/S0893-6080(98)00116-6.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language models are unsupervised multitask learners*. OpenAI. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Ratcliff, R. (1990). Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, *97*(2), 285–308. https://doi.org/10.1037/0033-295X.97.2.285.

Reid, E. (1981). Negative transfer and proactive interference: Some confusion in introductory psychology texts. *Teaching of Psychology*, *8*(2), 109–110. https://doi.org/10.1207/s15328023top0802_15.

Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., & Tesauro, G. (2019). Learning to learn without forgetting by maximizing transfer

and minimizing interference. *Proceedings of the International Conference on Learning Representations*. https://openreview.net/forum?id=B1gTShAct7.

Ring, M. B. (1997). CHILD: A first step towards continual learning. *Machine Learning*, *28*(1), 77–104. https://doi.org/10.1023/A:1007331723572.

Ruder, S. (2016). *An overview of gradient descent optimization algorithms*. arXiv. https://arxiv.org/abs/1609.04747.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0.

Russell, S. J., & Norvig, P. (2003). *Artificial intelligence - a modern approach* (2nd ed.). Prentice Hall.

Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W. R., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Kohli, P., Jones, D. T., Silver, D., Kavukcuoglu, K., & Hassabis, D. (2020). Improved protein structure prediction using potentials from deep learning. *Nature*, *577*(7792), 706–710. https://doi.org/10.1038/s41586-019-1923-7.

Silver, D. L., Yang, Q., & Li, L. (2013). Lifelong machine learning systems: Beyond learning algorithms. *Lifelong Machine Learning: Papers from the 2013 AAAI Spring Symposium*, 49–55. https://www.aaai.org/ocs/index.php/SSS/SSS13/paper/view/5802.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489. https://doi.org/10.1038/nature16961.

Skinner, B. F. (1950). Are theories of learning necessary? *Psychological Review*, *57*(4), 193–216. https://doi.org/10.1037/h0054367.

Sodhani, S., Chandar, S., & Bengio, Y. (2020). Toward training recurrent neural networks for lifelong learning. *Neural Computation*, *32*(1), 1–35. https://doi.org/10.1162/neco_a_01246.

Spong, M. W., & Vidyasagar, M. (1989). *Robot dynamics and control*. Wiley.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44. https://doi.org/10.1007/BF00115009.

Sutton, R. S. (1995). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in Neural Information Processing Systems*, *8*, 1038–1044. http://papers.nips.cc/paper/1109-

generalization-in-reinforcement-learning-successful-examples-using-sparse-coarse-coding.pdf.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (1st ed.). MIT Press.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

Taigman, Y., Yang, M., Ranzato, M., & Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, 1701–1708. https://doi.org/10.1109/CVPR.2014.220.

Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, *10*(56), 1633–1685. http://jmlr.org/papers/v10/taylor09a.html.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., & Mannor, S. (2017). A deep hierarchical approach to lifelong learning in minecraft. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 1553–1561. https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14630.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*(236), 433–460. https://doi.org/10.1093/mind/LIX.236.433.

United Nations, Department of Economic and Social Affairs, Population Division. (2019). *World population prospects 2019: Highlights*. https://population.un.org/wpp/Publications/Files/WPP2019_Highlights.pdf.

VandenBos, G. R. (2015). *Apa dictionary of psychology* (2nd ed.). American Psychological Association. https://doi.org/10.1037/14646-000.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*, 5998–6008. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

von Neumann, J. (1958). *The computer and the brain* (1st ed.). Yale University Press.

Zenke, F., Poole, B., & Ganguli, S. (2017). Continual learning through synaptic intelligence. *Proceedings of the 34th International Conference on Machine Learning*, *70*, 3987–3995. http://proceedings.mlr.press/v70/zenke17a/zenke17a.pdf.

Zhang, C., Bengio, S., Hardt, M., Recht, B., & Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. *Proceedings of the International Conference on Learning Representations*. https://openreview.net/forum?id=Sy8gdB9xx.