Learning Forever using Artificial Neural Networks

by

Shibhansh Dohare

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

Doctor of Philosophy

Department of Computing Science
University of Alberta

© Shibhansh Dohare, 2026
This work is licensed under CC Public Domain Mark 1.0

Abstract

In this dissertation, I study the ability of artificial neural networks to learn new things. I refer to this ability as *plasticity*. Plasticity is a desirable property of intelligent systems in the real world. Systems that interact with the real world can become outdated if they do not learn from the new information revealed to them by the extremely large and partially observable real world. All naturally intelligent systems continually learn new things to keep up with the changes in the world. Many applications like predicting markets or weather, learning human preferences, controlling factories or prosthetics, etc require AI systems that continually learn new things.

I provide direct and thorough demonstrations of loss of plasticity in deep learning systems. The demonstrations range from simple feed-forward networks to deep residual networks and cover a wide range of algorithmic choices. While seeking an understanding of the phenomenon, I found that loss of plasticity occurred simultaneously with hidden units becoming dormant and becoming similar to each other, hampering the ability of the networks to learn new things.

Next, I describe the *continual backpropagation* algorithm. I designed this algorithm, in collaboration with others, to maintain plasticity in artificial neural networks. Continual backpropagation is a simple extension of the conventional backpropagation algorithm. The conventional backpropagation has two main parts: initialization with small random weights and gradient descent at each step. In addition to gradient descent, continual backpropagation reinitializes a small fraction of units at each step. Continual backpropagation indefinitely maintained plasticity in many continual supervised learning problems.

Finally, I study plasticity of deep reinforcement learning systems. First, I show that plasticity loss has an effect even in stationary reinforcement learning problems. It is unsurprising that plasticity loss affects stationary reinforcement learning because it requires continual learning due to changing policies and bootstrapping. This effect is visible under long-term learning. I found that some commonly used RL algorithms do not scale with data. Their performance drops as they continue interacting with the environment. Plasticity loss hampers the ability of these algorithms to recover and relearn when the performance drops, resulting in agents that do not get better with experience. Continual backpropagation allows these algorithms to recover when performance drops, leading to agents whose performance keeps improving with experience. Second, I show that loss of plasticity is a significant problem in continual reinforcement learning problems, and continual backpropagation maintains plasticity in these problems.

Preface

Chapters 3, 4, 6, and 7 are based on the paper:

- Dohare, S., Hernandez-Garcia, J. F., Lan, Q. Rahman, P., Mahmood, A. R., and Sutton, R. S. (2024). Loss of plasticity in deep continual learning. *Nature* **632**, 768–774.

Fernando and I designed and conducted all the experiments for the demonstrations of loss of plasticity provided in Chapter 4. In particular, I designed the Permuted MNIST and Continual ImageNet experiments. Fernando and I jointly designed the CIFAR-100 experiments, and we both include them in our dissertations.

I developed the continual backpropagation algorithm presented in Chapter 6 in collaboration with Rich and Rupam. I conducted all the experiments to evaluate the algorithm presented in Chapter 6.

I discovered the policy collapse phenomenon in PPO and its connection with loss of plasticity presented in Chapter 7. Qingfeng and Rupam helped with conducting the experiments in Chapter 7. I wrote the paper using extensive suggestions and discussions with Rupam, Rich, and Fernando. The 2-state MDP in Chapter 7 is based on this workshop paper:

- Dohare, S., Lan, Q., and Mahmood, A. R. (2023). Overcoming Policy Collapse in Deep Reinforcement Learning. In *Sixteenth European Workshop on Reinforcement Learning*.

To my parents,

who instilled in me the confidence to work on ambitious problems.

Acknowledgements

The research I conducted during my PhD continues my master's thesis work. During the first couple of years of my PhD, we tried to publish the initial versions of the ideas of this thesis in various conferences. However, the paper was rejected from four different conferences. It would not have been possible for me to keep working on the same ideas without the support of my advisors, Richard Sutton and Rupam Mahmood. They helped me see the value of these ideas and ignore the noisy reviews. Eventually, we decided to submit the paper to a journal because the journal review process was much more suitable for our work, as it was not in the mainstream of continual learning research. After we started writing the paper, Rich spent 8 months teaching me how to write clearly. We would meet twice a week, and Rich would give me extensive feedback on my writing. In those 8 months, we only wrote three pages. However, the output of those 8 months were not those three pages, but my ability to write technical papers. At the same time, Rupam taught me how to think from first principles and design practical algorithms. I'm very grateful to Rich and Rupam for spending so much time on making me a scientist. After we finished writing our journal paper, it was Marlos who encouraged us to be more ambitious and submit our paper to Nature. My committee members, Marlos, Dale, Razvan, and Peter, gave valuable feedback on my research and encouraged me to make it more rigorous and precise. Jun at Huawei helped me think about my research from the perspective of people outside the reinforcement and continual learning communities.

I am most grateful for the work of social revolutionaries in India like Baba Saheb Dr. B. R. Ambedkar, Jotiba Phule, and Periyar E.V. Ramasamy who liberated

millions of low-caste people from the shackles of the caste system in India. I belong to a caste of untouchables called Chamar. In the Indian caste system, untouchables like Chamars are considered impure, and their touch is believed to pollute higher caste individuals. This untouchability led to severe social segregation and discrimination against low caste people, where they were not even allowed to live in the city, let alone get any education. Untouchability was widely practiced in India a couple of generations ago and is still practiced in some parts of the country. It is due to the work of social revolutionaries like Dr. Ambedkar, who is regarded as the father of the Indian Constitution, that low-caste people gained basic human rights and access to education. My parents were the first generation of my family to receive any education. I will forever be indebted to my parents for showing me the value of education and helping me pursue my interests. Thanks to my parents, brother, mama, mausis, and grandparents for helping me see what I really value in life.

The last few years of my life have been some of the best of my life, and that is mainly due to my friends. They were by my side during all the ups and downs of the last few years. I am grateful to be close friends with Graves, Nicole, Georgia, Fernando, Aman, and Charlotte; without their support, I might not have finished this degree. I am lucky to have my climbing crew by my side. They are my family in Edmonton. Thanks to Ivan, Maryna, Peter, Sam R, Andrew C, Sam G, Tanner, Andrew S, Eva, Tyler, Meagean, Charlotte, Mel, Josh, Josh E, Liz, Bekah, Cory, Sarah, Nicole Y, and Kyle for all the climbing trips, I don't see how I will ever have a day as good as our first one on Majestic in Squamish. I am glad to have had my friends in the department who shared the experiences of grad school and made it much better, Farza, Khurram, Prabhat, Leticia, Matt S, Qingfeng, Anna, Kevin, Diego, Andy, Abhishek, Alex K, Alex L, Manan, Raksha, Gabor, David, Liam, Jordan, Aidan, Esra'a, Esraa, Ehsan, Kenny and Tian. Final thanks to my ESSC multi-sport team of Em, Raf, Julia, Nick, Paul, Summer, and Kelly, who made me love every sport.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The ability to learn new things is a desirable property of intelligent systems in the real world. The real world is extremely large and partially observable. Systems usually have a very small window into the world. Due to the small size of the window relative to the size of the world, the world continually reveals new information. Systems that do not learn from this information risk becoming outdated and ineffective. In nature, animals also learn new things throughout their lives. Similarly, in industry, AI systems that continually learn new things are highly sought-after across various applications. These applications range from recommendation systems where people's preferences can change over time to robots interacting with the physical world to agents interacting with humans whose desires and motivations can change to large language models that have to accumulate new knowledge continually.

The current dominant paradigm in AI, deep learning, is specialized to problems where deployment is separate from a training phase, and systems only learn during the training phase. The most advanced models, like GPT-4 (OpenAI, 2023) and DeepSeek-R1 (DeepSeek-AI, 2025), have an extensive training phase, but they do not learn after they are deployed. If there is substantial new data to learn from, the most common strategy in practice has been to train a new model from scratch on the old and new data combined. Even when deep learning methods are applied to non-stationary settings like reinforcement learning, techniques like replay buffer and

target networks are used to make the setting nearly stationary (Mnih et al., 2015).

It is well-known that deep learning systems tend to forget what they have previously learned when they continue to learn on new data. This phenomenon, also known as "catastrophic forgetting", was first shown in the 1980s (McCloskey and Cohen, 1989; French, 1999). Since then, catastrophic forgetting has received significant attention, and many papers have been dedicated to overcoming catastrophic forgetting (Kirkpatrick et al., 2017; Yoon et al., 2018; Aljundi et al., 2019; Golkar et al., 2019; Riemer et al., 2019; Rajasegaran et al., 2019; Javed and White, 2019; Luo et al., 2023).

Although catastrophic forgetting is an important issue in some cases, the essence of continuing to learn from new data is simply to *learn from new data*. I refer to the ability to learn from new data as *plasticity*. Maintaining plasticity is essential for continual learning systems that face a non-stationary data stream. In this dissertation, I study plasticity, which is different but complementary to the more common focus on catastrophic forgetting.

My goal for this dissertation is to develop algorithms that can continually learn new things using artificial neural networks. The first question on the path to this goal is: do standard deep learning algorithms lose plasticity in continual learning problems? Somewhat surprisingly, this question had not been directly studied in the literature when I started working on this dissertation. In Chapter 3, I discuss the prior work that attempted to answer this question or provided hints towards an answer.

The first contribution of this dissertation is to provide a definitive answer to the question of loss of plasticity in deep continual learning. The demonstrations of loss of plasticity are presented in Chapter 4. These demonstrations range from simple feed-forward networks to deep residual networks. They involve various optimizers, hyperparameters, commonly used techniques in supervised learning, under- and over-parameterized networks, a wide range of rates of distribution change, and a wide range of memory constraints, from problems where information has to be processed

one data point at a time to problems where all past data can be stored, network architectures ranging from feed-forward networks to residual networks. Based on these demonstrations, I discuss various possible formalisms of the phenomenon of loss of plasticity as well as their advantages and limitations in Chapter 5.

The second contribution of this research is to develop an algorithm that can maintain plasticity. The key insight behind our algorithm is that good continual learning algorithms should do similar computations at all times. The conventional backpropagation has two main parts: initialization with small random weights and then gradient descent at each step. Our algorithm, *continual backpropagation*, is a simple extension of the conventional backpropagation algorithm. In addition to gradient descent, continual backpropagation reinitializes a small number of units during training, typically less than one per step. I fully describe the continual backpropagation algorithm, its different behaviors, and all relevant experiments in Chapter 6. In various continual supervised learning problems, continual backpropagation maintained plasticity indefinitely.

The final contribution is the study of loss of plasticity in deep reinforcement learning. Specifically, I study the relation between loss of plasticity and *policy collapse*, a phenomenon where the learned policy dramatically worsens after initial training in stationary reinforcement learning problems, in Chapter 7. I study policy collapse in PPO (Schulman et al., 2017), which is one of the most commonly used reinforcement learning algorithms. Policy collapse was not studied in the literature before this dissertation. It was probably missed because most stationary reinforcement learning experiments are stopped when the performance plateaus and the policy stops improving. On the other hand, policy collapse takes a long time to be visible, and it typically starts after the performance has plateaued for some time. Plasticity loss, in part, causes policy collapse. Once the agent's performance drops, it cannot recover due to plasticity loss. Algorithms that maintain plasticity, like continual backpropagation, can mitigate policy collapse, leading to agents that keep getting better with

experience. Additionally, I provide a demonstration of loss of plasticity in a non-stationary reinforcement learning problem using PPO. This problem requires the agent to continually change its behaviour as the environment kept changing. Continual backpropagation, along with some weight regularization, is sufficient to maintain plasticity in this problem.

Chapter 2

Background

This thesis builds on the foundations of artificial neural networks, deep learning, and reinforcement learning. In this chapter, I provide an overview of the key concepts required for understanding the thesis. Readers familiar with the basics of deep learning and reinforcement learning can skip to the next chapter.

Notationally, I represent vectors with bold lowercase letters, for example, $\mathbf{x} \in \mathbb{R}^n$ is a real-valued vector with n dimensions. The ith element of this vector is represented by $\mathbf{x}[i]$. And I use subscripts to represent the value at a specific step, for example, $\mathbf{x}_t$ represents the value of $\mathbf{x}$ at step t. Finally, I represent matrices with bold uppercase letters, for example, $\mathbf{W} \in \mathbb{R}^{m \times n}$ is a real-valued matrix with m rows and n columns.

2.1 Artificial Neural Networks

Artificial Neural Networks are mathematical functions that map an input to an output. Formally, let's denote the network by $f_{\boldsymbol{\theta}} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, where $\mathbf{x} \in \mathbb{R}^n$ is an n-dimensional input, $\mathbf{y} \in \mathbb{R}^m$ is the m-dimensional output, and $\boldsymbol{\theta} \in \mathbb{R}^d$ represents the parameters that specify the network.

The networks used in this thesis are feed-forward networks. Feed-forward networks perform a sequence of operations on the input to produce the output. One useful way to think about these networks is as a sequence of layers, where each layer is a mathematical function that contains some learnable parameters. In the simplest case,

each layer, except the last, applies a linear transformation, followed by an element-wise non-linear transformation to the input. The last layer only applies the linear transformation.

Formally, let's consider a network with L layers and denote the mathematical function of layer l as h_l. Then, for some input $\mathbf{x}$ to the layer, the output of the layer is $h_l(\mathbf{x}) = \phi(\mathbf{W}_l\mathbf{x} + \mathbf{b_l})$, where $\mathbf{W}_l$ and $\mathbf{b}_l$ are called the weight and bias parameters of the layer, respectively, and ϕ is an element-wise non-linear function, called the activation function. This holds for $l \in [1, L]$. While, $h_L(\mathbf{x}) = \mathbf{W}_L\mathbf{x} + \mathbf{b_L}$. The dimensions of $\mathbf{W}_l$ and $\mathbf{b}_l$ must be specified at the beginning to define the architecture of the network. For a network with L layers, the output of the network is $f(\mathbf{x}) = h_L(h_{L-1}(\ldots h_2(h_1(\mathbf{x}))\ldots))$.

There are many possible choices for specifying the activation function ϕ. The most common choice in deep learning is either the ReLU (Rectified Linear Unit) activation (Householder, 1941; Fukushima, 1975; Krizhevsky et al., 2012) or some variation of it. The ReLU activation is defined as $\text{ReLU}(x) = \max(0, x)$. All the networks in this thesis use the ReLU activation.

In image classification problems, it is important that the network output can be interpreted as probabilities, i.e., elements of the output vector sum to 1 and lie in $[0, 1]$. However, the raw network output described above does not satisfy these requirements. To ensure that the network's output can be treated as probabilities, when neural networks are trained on C-class classification problems, their output is normalized using the softmax function. Let $\mathbf{z} = h_L(h_{L-1}(\ldots h_2(h_1(\mathbf{x}))\ldots))$ be the raw output (logits) of the network. The softmax function is defined as

$$\text{softmax}(\mathbf{z})[i] = \frac{e^{\mathbf{z}[i]}}{\sum_{j=1}^{C} e^{\mathbf{z}[j]}} \tag{2.1}$$

for $i = 1, 2, \ldots, C$. The final output of a network with softmax is,

$$f(\mathbf{x}) = \text{softmax}(h_L(h_{L-1}(\ldots h_2(h_1(\mathbf{x}))\ldots))). \tag{2.2}$$

2.2 Training Artificial Neural Networks

In supervised learning, the goal of training artificial neural networks is to minimize some loss function $\ell : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$. The supervised learning problems used in this thesis are all image classification problems. In these problems, the network is given an input image, the goal of which is to guess the class to which the image belongs. Cross entropy loss is the standard loss function for image classification problems.

Let there be C classes, and let $\mathbf{y}[i]$ be the correct class label for a given image. $\mathbf{y}[i]$ is 1 if the image belongs to class i and 0 otherwise. The cross entropy loss function is $\ell_{\text{CE}} = -\sum_{i=1}^{C} \mathbf{y}[i] \log(\hat{\mathbf{y}}[i])$, where $\hat{\mathbf{y}}[i]$ is the probability of the image belonging to class i outputted by the network.

Modern deep learning uses the backpropagation algorithm (Rumelhart et al., 1986) for training artificial neural networks. This algorithm consists of two parts. The first is the initialization of network parameters, $\boldsymbol{\theta}$, with random numbers sampled from some distribution and the second is to perform stochastic gradient descent at each step. In stochastic gradient descent, the parameters of the network are updated as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \ell(f_{\boldsymbol{\theta}_t}(\mathbf{x}_t), \mathbf{y}_t), \tag{2.3}$$

where $\eta > 0$ is the step size parameter, $\nabla_{\boldsymbol{\theta}_t} \ell(f_{\boldsymbol{\theta}_t}(\mathbf{x}_t), \mathbf{y}_t)$ is the gradient of the loss with respect to parameters $\boldsymbol{\theta}_t$ for input $\mathbf{x}_t$ and output $\mathbf{y}_t$. In many cases, when performing stochastic gradient descent, the loss is not calculated on just one input, rather a mini-batch of input is used. In this case, mini-batch stochastic gradient descent performs the following update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{1}{|\mathcal{B}_t|} \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{B}_t} \nabla_{\boldsymbol{\theta}_t} \ell(f_{\boldsymbol{\theta}_t}(\mathbf{x}_i), \mathbf{y}_i) \tag{2.4}$$

where, $\mathcal{B}_t$ is the mini-batch at step t.

One common strategy to initialize neural network parameters is Kaiming Initialization (He et al., 2015). Kaiming initialization ensures that the variance of activations

and gradients stays the same across layers at initialization. This mitigates the exploding and vanishing gradient problems in deep networks. For a deep ReLU network, the Kaiming normal distribution for a layer with n inputs is a Gaussian distribution with mean zero and standard deviation $\frac{\sqrt{2}}{n}$.

In many applications of deep learning, a variation of stochastic gradient descent called the Adam optimizer is used (Kingma and Ba, 2015). Adam keeps an exponentially moving average of past gradients and squared gradients. Let, $\mathbf{g}_t$ be the gradient at step t, for stochastic gradient descent, $\mathbf{g}_t = \nabla_{\boldsymbol{\theta}_t} \ell(f_{\boldsymbol{\theta}_t}(\mathbf{x}_t), \mathbf{y}_t)$. And for mini-batch stochastic gradient descent, $\mathbf{g}_t = \frac{1}{|\mathcal{B}_t|} \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{B}_t} \nabla_{\boldsymbol{\theta}_t} \ell(f_{\boldsymbol{\theta}_t}(\mathbf{x}_i), \mathbf{y}_i)$. Adam updates the parameters $\boldsymbol{\theta}_t$ as

$$
\begin{aligned}
\mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\, \mathbf{g}_t \\
\mathbf{v}_t &= \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\, \mathbf{g}_t^2 \\
\hat{\mathbf{m}}_t &= \mathbf{m}_t / (1 - \beta_1^t) \\
\hat{\mathbf{v}}_t &= \mathbf{v}_t / (1 - \beta_2^t) \\
\boldsymbol{\theta}_t &= \boldsymbol{\theta}_{t-1} - \eta\, \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon)
\end{aligned} \tag{2.5}
$$

where $\mathbf{m}$ and $\mathbf{v}$ are initialized as $\mathbf{0}$s, $\beta_1, \beta_2 \in [0, 1)$ are two hyper-parameters, and ϵ is a small positive number, one commonly used value being 10^{-8}, used for mathematical stability. In practice, the default βs ($\beta_1 = 0.9, \beta_2 = 0.999$) are usually used (Abadi et al., 2016; Paszke et al., 2019) without further adjustment in both supervised learning and reinforcement learning.

One commonly used technique in deep learning is L2-regularization. It incorporates a penalty term into the loss function proportional to the squared ℓ_2-norm of the network weights. This penalty term encourages stochastic gradient descent toward solutions with smaller weight magnitudes. In our case, $\ell_{\text{reg}}(\boldsymbol{\theta}) = \ell_{CE}(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_2^2$, where λ is a hyperparameter that controls the strength of regularization. When used with stochastic gradient descent, the network parameters are updated as

$$
\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta(\mathbf{g}_t + 2\lambda \boldsymbol{\theta}_{t-1}). \tag{2.6}
$$

Shrink and Perturb is a technique based on L2-regularization (Ash and Adams, 2020). As the name suggests, Shrink and Perturb performs two operations. The first is to shrink the weights at each step, similar to L2-regularization. And second, to perturb the weights by adding random noise at each step. The update equation for Shrink and Perturb when it is used with stochastic gradient descent is

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta(\mathbf{g}_t + 2\lambda\boldsymbol{\theta}_{t-1}) + \mathbf{p}_t \tag{2.7}$$

where $\mathbf{p}$ is a vector with the same dimensions as $\boldsymbol{\theta}$, whose elements are randomly sampled from a Gaussian distribution with mean zero and standard deviation ρ. ρ is a hyperparameter of the Shrink and Perturb algorithm.

Another technique we use in this thesis is *batch normalization* (Ioffe and Szegedy, 2015). In batch normalization, the inputs to each layer are normalized using the data from the mini-batch. Application of batch normalization involves three steps during training: calculation of mean and variance, normalization, and scaling. Consider a layer with input $\mathbf{x} = [\mathbf{x}[1], \mathbf{x}[2], \ldots, \mathbf{x}[n]]$. Let the input data from the mini-batch be $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots \mathbf{x}^{(m)}$. In the first step, we calculate the mean and variance for each input dimension as

$$\boldsymbol{\mu}[j] = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)}[j] \tag{2.8}$$

$$\boldsymbol{\sigma}[j]^2 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}^{(i)}[j] - \boldsymbol{\mu}[j])^2, \tag{2.9}$$

where $\boldsymbol{\mu}[j]$ and $\boldsymbol{\sigma}[j]^2$ are the mean and variance of the jth input dimension. The next step is to normalize the inputs as

$$\hat{\mathbf{x}}^{(i)}[j] = \frac{\mathbf{x}^{(i)}[j] - \boldsymbol{\mu}[j]}{\sqrt{\boldsymbol{\sigma}[j]^2 + \epsilon}}.$$

Finally, the input is scaled and shifted as

$$\mathbf{y}^{(i)}[j] = \boldsymbol{\gamma}[j]\hat{\mathbf{x}}^{(i)}[j] + \boldsymbol{\beta}[j], \tag{2.10}$$

where $\boldsymbol{\gamma}[j]$ and $\boldsymbol{\beta}[j]$ are parameters learned through stochastic gradient descent, and ϵ is a small number used for mathematical stability. During testing, batch norm uses the running estimates of mean and variance that are calculated during training, instead of computing them from the current mini-batch. The running estimates approximate the mean and variance of the entire training set. In practice, normalization can be applied to the layer's input or before or after the activation function is applied. There is no consensus on the best place to normalize, and different applications use normalization at different places.

The final technique we use is called Dropout (Hinton et al., 2012). In Dropout, during training, a fraction of randomly chosen input dimensions of each layer are set to zero. Consider a layer with input $\mathbf{x} = [\mathbf{x}[1], \mathbf{x}[2], \ldots, \mathbf{x}[n]]$, Dropout applies the following transformation

$$\mathbf{y} = \mathbf{r} \odot \mathbf{x} \tag{2.11}$$

where $\odot$ is element-wise multiplication, and $\mathbf{r}$ is a binary mask whose elements are independently sampled from a Bernoulli distribution, where the probability of an element being zero is defined by the hyperparameter $p \in [0, 1]$. Dropout is turned off during testing, meaning all input dimensions remain active. However, because more inputs are active, they are all scaled by $(1 - p)$ to ensure that the scale remains the same during training and testing.

2.3 Reinforcement Learning

In reinforcement learning (RL), an agent learns to maximize a reward signal by interacting with an environment (Sutton and Barto, 2018). The reinforcement learning problem is generally formalized as a Markov Decision Process (MDP). Let $M = (\mathcal{S}, \mathcal{A}, \mathrm{P}, r, \gamma)$ be a Markov Decision Process which includes a state space $\mathcal{S}$, an action space $\mathcal{A}$, a state transition probability function $\mathrm{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1] \subset \mathbb{R}$, a reward signal $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in [0, 1)$.

Given an MDP, the agent's behaviour is specified by a policy π, which outputs a distribution over the action space given a state. At each time step t, the agent observes a state $S_t \in \mathcal{S}$ and selects an action A_t from $\pi(\cdot|S_t)$. The environment then transitions to the next state $S_{t+1} \in \mathcal{S}$ according to the transition function $\mathrm{P}(\cdot|S_t, A_t)$ and the agent receives a scalar reward $R_{t+1} = r(S_t, A_t) \in \mathbb{R}$. Considering an episodic task with horizon T, the return G_t is defined as the sum of discounted rewards, that is, $G_t = \sum_{k=t}^{T} \gamma^{k-t} R_{k+1}$. The action-value function of policy π give the expected return if the agent takes a specific action in a specific state and follows the policy π. It is defined as $q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a]$, $\forall (s, a) \in \mathcal{S} \times \mathcal{A}$. Similarly, the state-value function v_π maps states to expected returns, $v_\pi(s) = \mathbb{E}[G_t|S_t = s]$, $\forall s \in \mathcal{S}$. Another useful quantity is the advantage function, which is $\mathrm{Adv}_\pi(s, a) = q_\pi(s, a) - v_\pi(s)$.

The agent aims to find an optimal policy π^* that maximizes the expected return starting from some initial states. A common class of approaches to find the optimal policy is based on the policy gradient methods. Policy gradient methods directly learn a policy. Among policy gradient methods, Proximal Policy Optimization, PPO, is one of the most widely used Schulman et al. (2017). The key idea of PPO is to constrain the policy update by using a clipped surrogate objective to prevent the policy from changing too much. In practice, PPO is applied as an alternating sequence of interaction and optimization. During interaction, the agent stores all the states, actions, and rewards in a replay buffer. During optimization, the data in the buffer is used to update the policy. After the optimization phase, the data in the replay buffer is discarded and it is filled during the next interaction phase. Let's refer to the policy with network parameters $\boldsymbol{\theta}$ as $\pi_{\boldsymbol{\theta}}$ and let $\boldsymbol{\theta}_{old}$ be the network parameters at the start of an optimization phase. And let $r_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_{old}}(a_t|s_t)}$. PPO optimizes the following objective on the data in the replay buffer,

$$\ell^{PPO} = \mathbb{E}_t[min(r_t(\boldsymbol{\theta})\widehat{\mathrm{Adv}_t}, clip(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\widehat{\mathrm{Adv}_t})] \tag{2.12}$$

where, $\widehat{\mathrm{Adv}_t}$ is the estimate of the advantage at time t, and ϵ is a hyperparameter. The

advantage estimate is calculated using generalized advantage estimation (Schulman et al., 2016) that uses an additional network to approximate the value function. Note that $\pi_{\boldsymbol{\theta}_{old}}(a_t|s_t)$ is a constant during the optimization phase, the only quantity that changes is $\pi_{\boldsymbol{\theta}}(a_t|s_t)$. And it can change until it reaches the boundaries defined by the clipping function.

Chapter 3

Prior Work on Loss of Plasticity

The first step towards developing artificial neural networks that can learn forever is to ask if networks trained by deep learning methods can learn forever. This question had not been thoroughly studied before this dissertation. However, some work provided hints or partial answers to this question. In this chapter, I discuss work that was done prior to this dissertation on the issue of loss of plasticity in deep learning.

Plasticity research is equally relevant to the fields of machine learning and neuroscience. However, historically, it has received more prominent attention in neuroscience. In neuroscience, plasticity refers to the brain's ability to change. The brain exhibits synaptic plasticity, where the strength of the connections between neurons (synapses) changes in response to experience (Citri and Malenka, 2008). Moreover, there is also structural plasticity in the brain, where entirely new connections can grow between neurons (Eriksson et al., 1998). Similarly, in the machine learning literature, plasticity commonly refers to a system's ability to adapt to changes (Carpenter and Grossberg, 1988). In light of this meaning of the term plasticity, it is natural to use *loss of plasticity* to refer to the loss of the ability to adapt or the loss of the ability to learn new things.

The first evidence of loss of plasticity in deep learning comes from the psychology literature of the early 2000s. Ellis and Lambon Ralph (2000), Zevin and Seidenberg (2002), and Bonin et al. (2004) showed plasticity loss in neural networks of early 2000s

in regression problems. The experiments in these papers used a setup where a set of examples was presented to the network for a certain number of epochs, and then the training set was expanded with a second set of examples and training proceeded. The results were interesting and they showed that the error for the examples in the first training set was lower than for the second set of examples after controlling for the number of epochs. These experiments provided evidence that artificial neural networks trained by the backpropagation algorithm can lose plasticity. The main limitation of these works is that the networks used were relatively shallow by today's standards, and the algorithms used are not those that are most popular today. For example, they did not use the ReLU activation function or modern optimizers like Adam. The psychology research from early 2000s on artificial neural networks provides hints, but does not provide a clear answer to the question of whether or not modern deep learning networks exhibit loss of plasticity.

Some recent studies in the machine learning literature suggest that there is loss of plasticity in deep learning. Chaudhry et al. (2018) observed loss of plasticity in continual image classification problems. However, these results are not completely satisfactory because of a confounding variable in their experiments. In their experiments, new output units were added to the network when the network learns a new task. Meaning that the number of output units grew over time. Loss of plasticity was confounded with the effect of interference from old output units. More importantly, they found that when old output units were removed at the beginning of the next task, plasticity loss was minimal, suggesting that the plasticity loss they observed was primarily due to the effects of the old output units. The second key limitation of their study is that they did not study loss of plasticity when in a long sequence of tasks, as they only used ten tasks for their experiments.

Ash and Adams (2020) found that pretraining can be harmful in some cases. They showed that a network that is first trained on half of a dataset can lead to worse final performance than networks trained from scratch. The failure of pretraining in these

experiments is an important example of loss of plasticity and it suggested that there could be a major issue of loss of plasticity when deep learning systems face a long sequence of tasks. Berariu et al. (2021) built on Ash and Adams' work and showed in similar experiments that as the number of pretraining stages increases, the final performance gets worse. However, Berariu et al.'s experiments were still performed in a stationary problem setting with a fixed dataset, and they observed a small loss of plasticity. Although these papers hint at a fundamental problem of loss of plasticity in deep learning systems, they do not provide a fully satisfactory demonstration of the phenomena.

Concurrent with my work, some evidence for loss of plasticity in deep learning was obtained in the reinforcement learning literature. Reinforcement learning is inherently continual due to the changing behaviour of the agent, so the loss of plasticity in deep learning networks could affect modern deep reinforcement learning agents. Nikishin et al. (2022) showed in reinforcement learning problems, sometimes early learning could harm later learning. Lyle et al. (2022) also found that some deep reinforcement learning agents lose the ability to learn new functions over time. These are exciting results, but we can only draw limited conclusions from them due to the inherent complexity of modern deep reinforcement learning. All these papers from the psychology literature of the early 2000s and the recent machine learning and reinforcement learning literature provide evidence that deep-learning systems lose plasticity, but they are not direct or thorough demonstrations of the phenomenon in fully continual learning problems.

In the last few years, there has been a notable increase in the attention paid to the problem of plasticity loss in artificial neural networks. Several papers have been written adding evidence to the phenomenon, improving our understanding, and providing methods to maintain plasticity in artificial neural networks. A detailed discussion of recent papers and other related work is provided at the end of the relevant chapter.

Chapter 4

Demonstrations of Loss of Plasticity in Supervised Learning

In this chapter, I provide direct and thorough demonstrations of loss of plasticity in continual supervised learning problems. The contents of this chapter are taken from Dohare et al. (2024). All these demonstrations build on the demonstrations provided in my Master's thesis (Dohare, 2020). The main limitation of those experiments is that they used a small (single hidden layer with five units) network.

The demonstrations provided in this chapter use more commonly used networks and more realistic datasets. These demonstrations cover a wide range of memory constraints. The first demonstration is entirely online as there is no additional memory; in the second one, the learning system stores all the examples for the current classes; and in the last demonstration, the learning system stores all the examples seen so far. These experiments also cover multiple network architectures, over- and under-parameterized networks, optimizers, activation functions, and common techniques like dropout, regularization, and normalization.

4.1 Loss of Plasticity in Permuted MNIST

The first demonstration is based on the MNIST dataset (Lecun et al., 1998). MNIST is a commonly used dataset in supervised learning. It is a dataset of grayscale images of the handwritten digits from 0 to 9 together with their correct labels. It consists

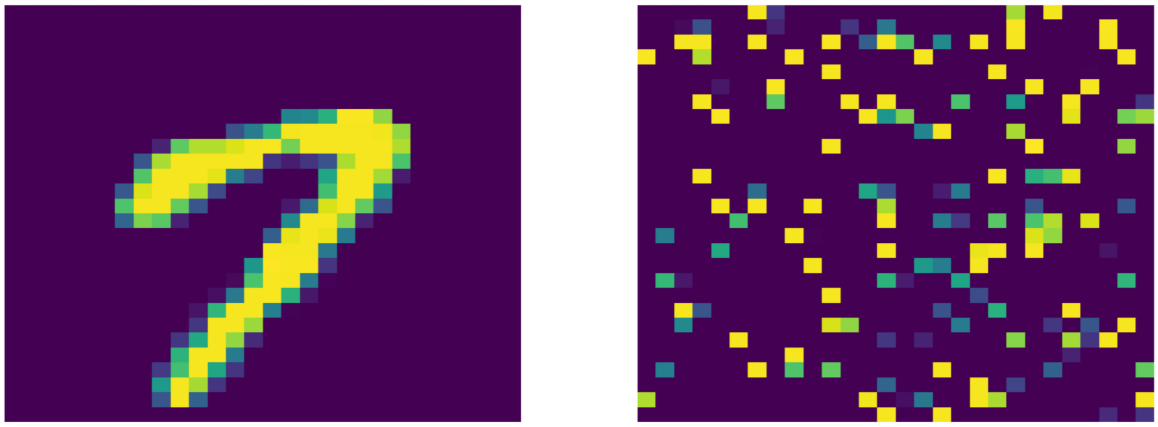

Figure 4.1: Left: An image with label '7' from the MNIST dataset. Right: A corresponding permuted image.

of 60,000 28×28 images. The left image in Figure 4.1 is an image contained in the dataset labeled by the digit 7. The smaller size of the images and the dataset compared to others used in this chapter allows smaller networks to perform well on this dataset. The small size of networks and datasets means that experiments can be performed with much less computation, which enables us to perform extensive and long-run continual learning experiments.

We created a continual supervised learning problem using *permuted* MNIST datasets (Goodfellow et al., 2014; Zenke et al., 2017). This problem can be seen as a sequence of tasks where each task uses a new permuted MNIST dataset. An individual permuted MNIST dataset is created by permuting the pixels in the original MNIST dataset. The right image in Figure 4.1 is an example of such a permuted image. All 60,000 images are permuted in the same way to produce the new permuted MNIST dataset. Note that convolutional layers are not helpful for permuted MNIST images because the spatial information is lost.

The main strength of this problem is that we can create a near-infinite sequence of permuted MNIST datasets which allows us to study long term plasticity. For a feedforward network, all the permuted MNIST datasets are of equal difficulty. A

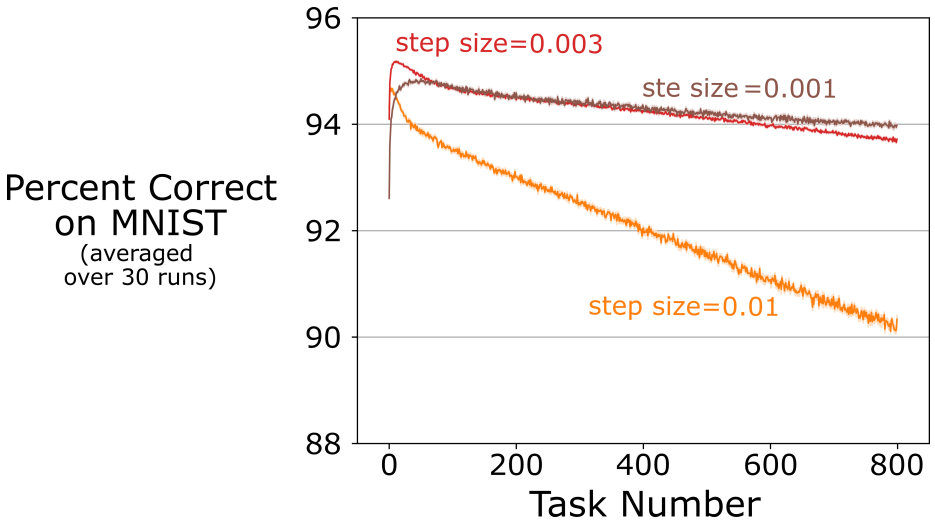

Figure 4.2: Loss of plasticity in a feed-forward network trained via backpropagation with various step sizes on Online Permuted MNIST. These results are averaged over 30 runs and the shaded regions correspond to plus and minus one standard error.

drop in performance in a sequence of permuted MNIST problems would mean that the network is losing plasticity.

We created a sequence of 800 supervised learning tasks, where each task used a different permuted MNIST dataset. Within each task, we presented all the 60,000 images one by one to the learning network in a randomized order. After completing one task, we moved to the next task and applied the same process, continuing this process across all 800 tasks. The network was trained on a single pass through the data without using any mini-batches during training. We call this problem *Online Permuted MNIST*.

We used feed-forward neural networks with three hidden layers and 2000 units in each layer for Online Permuted MNIST. For each example, the network outputs a guess for the probabilities of each of the 10 classes. These guessed probabilities are then compared to the correct label, and the cross-entropy loss is calculated. Finally, stochastic gradient descent (SGD) is performed using the computed loss. To measure the performance, we calculated the percentage of times the network correctly classified each of the 60,000 images in the task. Figure 4.2 plots this per-task performance

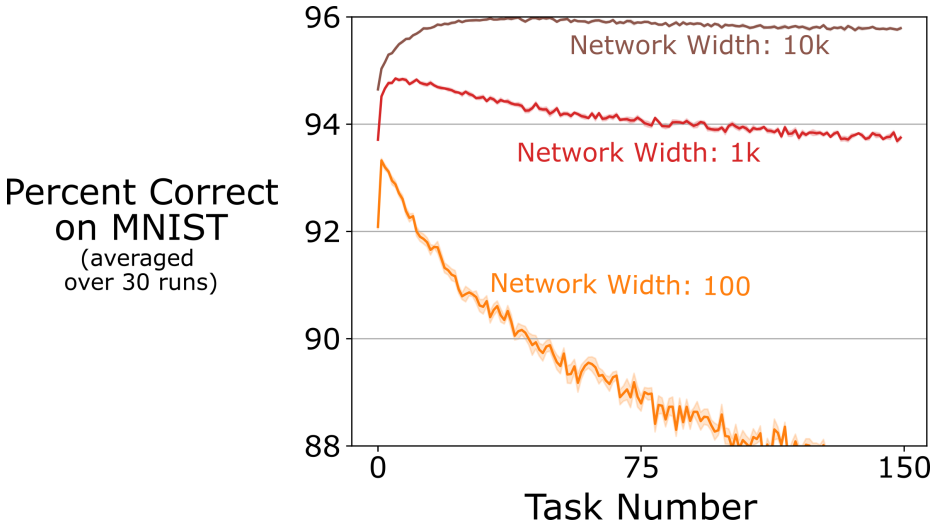

Figure 4.3: The effect of the size of the network on loss of plasticity. Smaller networks lose plasticity faster. These results are averaged over 30 runs and the shaded regions correspond to plus and minus one standard error.

measure versus task number. The network architecture utilized ReLU activation functions, and the initial weight parameters were sampled from the Kaiming distribution.

Figure 4.2 shows the evolution of performance across tasks for different settings of the step size parameter. The performance first increased across tasks, which is expected as pretraining on similar tasks can help learning on later tasks. Specifically, in Online Permuted MNIST, features in layers closer to the output layer could be reused across tasks. However, after the initial increase, the performance began falling steadily across all subsequent tasks. This continuing drop in performance means that the network is slowly losing the ability to learn from new data.

For the next set of experiments, we test how the size of the network affects loss of plasticity on a given problem. To vary the size of the nextwork, we vary its width. We test networks with 100, 1,000, and 10,000 units per layer on the online permuted MNIST problem. These networks have about 100k, 3M, and 200M parameters respectively. We ran this experiment for only 150 tasks due to the significantly longer computational time required for the 200M parameter network. Figure 4.3 shows the performance at good step sizes for each network. The loss of plasticity over extended

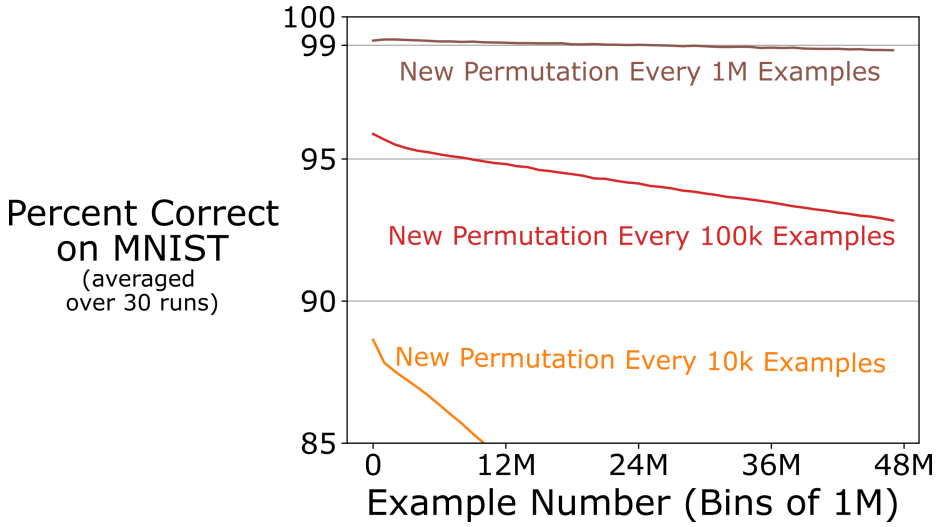

Figure 4.4: The effect of the rate of distribution change on plasticity loss. As the non-stationarity in the data increases, loss of plasticity becomes more severe. These results are averaged over 30 runs and the shaded regions correspond to plus and minus one standard error. However, the shaded region is invisible because the standard error is small.

training is most severe for smaller networks, though even the largest networks exhibit some loss of plasticity.

Finally, we studied the effect of the rate at which the task changed. Returning to the original network architecture with 2,000-unit layers, we varied the task-switching intervals. We changed permutations after every 10,000, 100,000, or 1 million examples instead of the original 60,000 examples. The experiments were run for a total of 48M examples for all three settings of task switching interval. We measured network performance on each task by calculating the percentage of correct classifications across all examples within that task. The evolution of performance is shown in Figure 4.4, which again showed that performance dropped over successive tasks. The performance drops faster as the rate of distribution changes increases, suggesting that loss of plasticity will be a more important phenomenon in settings where the distribution changes rapidly, like reinforcement learning. Altogether, these results show that the phenomenon of loss of plasticity robustly arises with conventional backpropagation. Loss of plasticity happens for a wide range of step sizes, distribution change rates,

and under and over-parameterized networks.

4.2 Correlates of Loss of Plasticity in Permuted MNIST

Let's turn our attention to understanding why backpropagation loses plasticity in Online Permuted MNIST. At the highest level, the only difference in the learning algorithm across time is the network weights. Initially, weights consisted of small random values drawn from the initialization distribution; however, following training on several tasks, these weights became specialized for the most recently encountered task. Consequently, the initial weights for each new task differ fundamentally from the weights at the very beginning of the first task. As this difference in the weights is the only difference in the learning algorithm across time, some specific characteristics of the initial weight distribution must be facilitating plasticity during early learning phases. This initial random distribution likely encompasses multiple plasticity-enabling features, such as unit diversity, activation states that avoid saturation, small magnitude parameters, etc.

As we will now show, numerous beneficial properties of the initial distribution deteriorate alongside the loss of plasticity. The degradation of each advantageous characteristic partially explains the degrading performance that we observed. We subsequently present arguments for how the deterioration of these properties may contribute to plasticity loss, along with metrics that measure the extent of each deterioration for each property. We conduct an in-depth study of the Online Permuted MNIST problem which will serve as motivation for multiple methods that could reduce plasticity loss.

The first noticeable phenomenon that occurs concurrently with the loss of plasticity is the the progressive accumulation of dead units. When a unit becomes dead, it stops propagating gradients during backpropagation. Zero gradients stops the weight coming into the unit from changing, which effectively eliminates the unit's learning

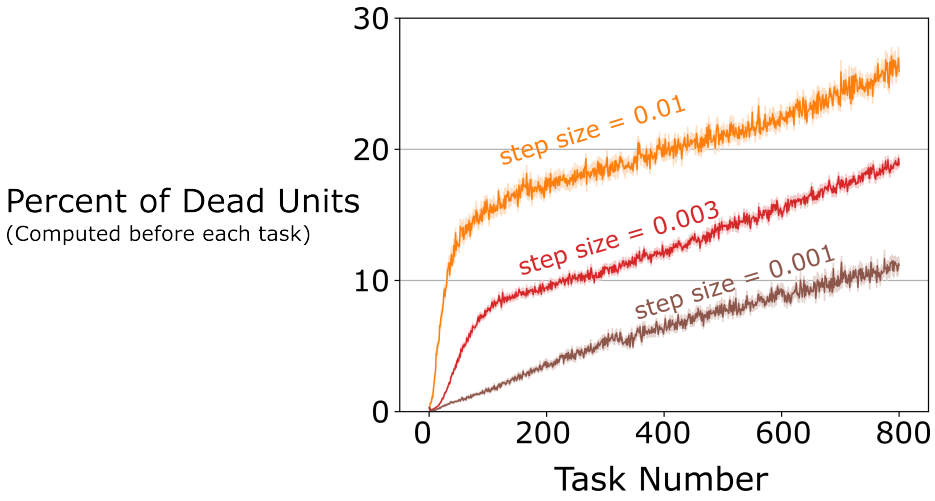

Figure 4.5: Evolution of the number of dead units in a deep network trained via backpropagation with different step sizes. The percentage of dead units in the network increases over time. These results are averaged over 30 runs and the shaded regions correspond to plus and minus one standard error.

capacity. For ReLU activation functions, this situation arises when a unit's activation remains zero across all task examples; such units are commonly termed "dead" (Lu et al., 2019; Shin and Karniadakis, 2020). We quantify dead units in ReLU networks by counting those that output zero for every example in a randomly selected sample of two thousand images evaluated at the beginning of each new task. For networks using sigmoidal activations, an analogous metric is the number of units whose outputs falls within an ϵ neighborhood of the function's saturation values for some small positive ϵ (Rakitianskaia and Engelbrecht, 2015). In this section we only focus on ReLU networks.

The evolution of dead units in the online permuted MNIST problem is plotted in Figure 4.5. It shows that the fraction of dead units increases over time. This increase is associated with the loss of plasticity in online permuted MNIST. For the largest step size, 0.01, a quarter of all units died after 800 tasks. This large increase in number of dead units directly decreases the capacity of the network.

Another concurrent phenomenon accompanying plasticity loss is the steady increase of the network's mean weight magnitude. The network's mean weight magni-

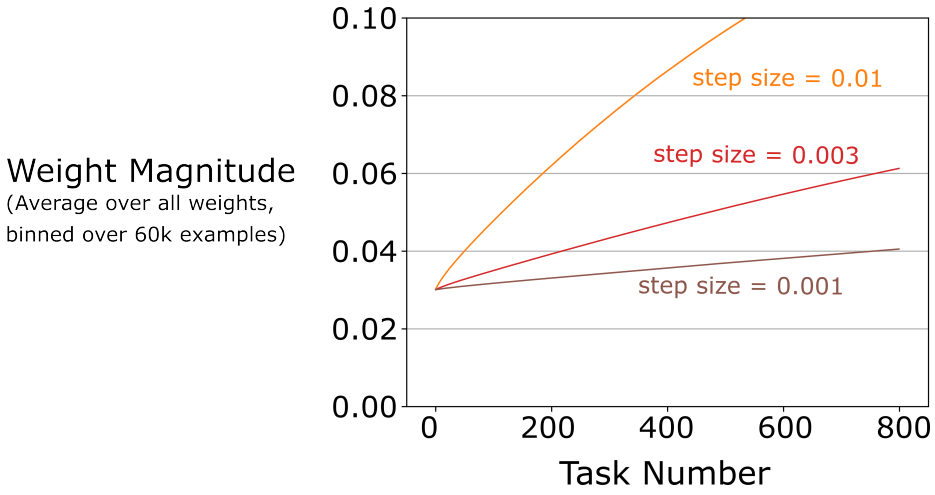

Figure 4.6: The average weight magnitude in the network increases over time. These results are averaged over 30 runs. The shaded regions corresponds to plus and minus one standard error. However, they are not visible because the shaded regions are thinner than the line width because standard error is small.

tude is the sum of the absolute values of all the network weights divided by the total number of weights. This rise of the network's mean weight magnitude in the online permuted MNIST problem is shown in Figure 4.6. The rise of the weight magnitude co-occurs with the declining plasticity shown in Figure 4.2.

The weight magnitude growth can pose a learning impediment because larger parameter values typically correlate with slower learning. Network weights directly influence the condition number of the Hessian of the loss function with respect to network parameters, $\boldsymbol{\theta}$. The Hessian's condition number is known to impact convergence rates in SGD optimization (Boyd and Vandenberghe, 2004). Therefore, increasing weight magnitudes may result in an ill-conditioned Hessian matrix, leading to slower convergence rates.

The final phenomenon accompanying loss of plasticity is the decline in the representation's effective rank. The rank of a matix refers to the number of linearly independent dimensions. Similarly, the effective rank accounts for each dimension's contribution to the transformation induced by a matrix (Roy and Vetterli, 2007). A high effective rank indicates that most matrix dimensions contribute comparably

to the transformation. Conversely, a low effective rank suggests that only a limited number of dimensions substantially influence the transformation, meaning that information contained in most dimensions is redundant.

Formally, consider a matrix $\mathbf{\Phi} \in \mathbb{R}^{n \times m}$ with singular values σ_k for $k = 1, 2, ..., q$, and $q = \max(n, m)$. Let $p_k = \sigma_k / \|\boldsymbol{\sigma}\|_1$, where $\boldsymbol{\sigma}$ is the vector containing all the singular values, and $\| \cdot \|_1$ is the ℓ^1-norm. The effective rank of matrix $\mathbf{\Phi}$, or erank($\mathbf{\Phi}$), is defined as

$$\text{erank}(\mathbf{\Phi}) \doteq \exp\left\{H(p_1, ..., p_q)\right\}, \text{where } H(p_1, ..., p_q) = -\sum_{k=1}^{q} p_k \log(p_k). \tag{4.1}$$

Note that, unlike the rank of the matrix, the effective rank is a continuous measure with a range between one and the rank of a matrix.

For neural networks, a layer's effective rank quantifies how many units are necessary to reconstruct the layer's output effectively. When a hidden layer's output has a low effective rank, the output can be reproduced by a small number of units, indicating that many units within that layer contribute minimal useful information. We estimate the effective rank using a randomly sampled set of two thousand permuted images prior to training on each new task.

In the online permuted MNIST problem, loss of plasticity is accompanied by a decrease in the average effective rank of the network, as shown in Figure 4.7. The drop in the effective rank is not inherently a problem. It has been demonstrated that gradient-based optimization naturally tends toward low-rank solutions through implicit loss function regularization or implicit rank minimization (Smith et al., 2021; Razin and Cohen, 2020). Nevertheless, a low-rank representation may serve as poor initialization for learning new things because most hidden units provide negligible novel information about new input. The declining effective rank may account for plasticity loss in our experiments in the following way. For each task, SGD finds a low-rank solution tailored to the current task, which subsequently becomes the starting point for the following task. Through this iterative process, the representation

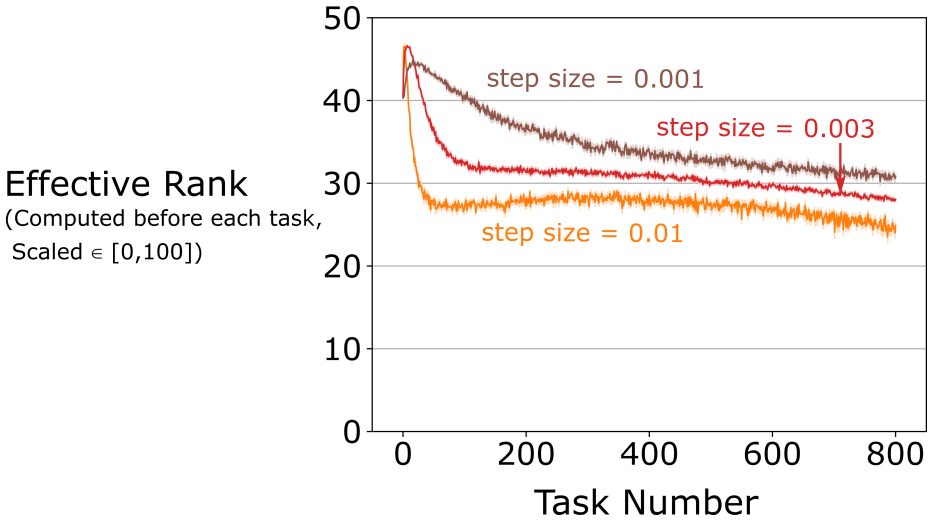

Figure 4.7: The figure depicts the evolution of the representation's effective rank for different step sizes of backpropagation. The effective rank decreases over time. The results in this plot are averaged over 30 runs. The shaded regions correspond to plus and minus one standard error.

layer's effective rank progressively diminishes after each task, constraining the range of solutions the network can immediately represent at the beginning of new tasks.

In this section, we looked deeper at the networks that lost plasticity in the Online Permuted MNIST problem. We noted that at the highest level, the only difference in the learning algorithm across time is the weights of the network, which means that the initial weight distribution possesses specific characteristics that enable early plasticity. As learning advances, the network weights diverge from their initial distribution, causing the algorithm to gradually lose plasticity. We found that loss of plasticity correlates with three phenomena: growing weight magnitudes, declining effective rank of the representation, and increasing number of dead units. Each of these correlates provide partial insights into the loss of plasticity faced by backpropagation.

4.3 Evaluating Existing Methods on Online Permuted MNIST

This section describes the effects of adding various existing methods to backpropagation on the loss of plasticity observed in the online Permuted MNIST problem. We analyzed five existing methods: L2-regularization (Goodfellow et al., 2016), Dropout (Hinton et al., 2012), online normalization (Chiley et al., 2019), Shrink and Perturb (Ash and Adams, 2020), and Adam (Kingma and Ba, 2015). The choice of L2-regularization, Dropout, normalization, and Adam was straightforward due to their widespread use in deep learning applications. Although Shrink and Perturb is less commonly used, we included it because it reduces the failure of pretraining, an instance of loss of plasticity. See Chapter 2 for details of all these methods.

To evaluate whether these techniques can alleviate loss of plasticity, we applied them to the Online Permuted MNIST problem using the identical feed-forward network architecture from the previous section. We used the same evaluation metric as before, the average online classification accuracy across all 60,000 task examples. Figure 4.8 displays the online classification accuracy of all the methods. We used a step size of 0.003 for all algorithms in this section because it performed the best for backpropagation, see Figure 4.2. We also tested how these methods affect the three correlates of loss of plasticity we identified in Section 4.2. Figure 4.9 shows the summaries of the three correlates for the loss of plasticity for the hyperparameter settings used in Figure 4.8.

Each method involves hyperparameters that directly influence its performance. Despite this dependency, we can find hyperparameter configurations that give us a good representation of the best performance for each method. We tested various combinations of hyperparameters for each method. The results for three different combinations of hyperparameters for each method are shown in Figure 4.10. While additional hyperparameter tuning could potentially lead to slightly better perfor-

mance, Figure 4.10 successfully captures the pattern of results and the behavior of these methods. We generally chose the hyperparameter value with the highest average classification accuracy across all 800 tasks. Similar to Figure 4.8, the first point in Figure 4.10 is the average classification accuracy over the first task, the next over the second task and so on.

L2-regualrization

L2-regularization adds a penalty term into the loss function that is proportional to the ℓ^2-norm of the network weights (see Equation 2.6). This penalty term encourages SGD toward solutions with smaller weight magnitudes. The performance of L2-regularization in the online permuted MNIST problem is shown in Figure 4.8 with the purple curve. The purple lines in Figure 4.9 show how the three correlates of loss of plasticity evolve with L2-regularization. With L2-regularization, weight magnitude stabilizes. As anticipated, the stable weight magnitude is associated with a lower loss of plasticity. Nevertheless, L2-regularization provides only partial protection against loss of plasticity. With L2-regularization, the proportion of dead units continues rising while the effective rank keeps decreasing, which explains the loss of plasticity with L2-regularization.

The added penalty term in the loss function introduces a hyperparameter λ that modulates the contribution of the penalty term. λ controls the peak performance and the speed of performance degradation, see Figure 4.10.

Shrink and Perturb

The next method we test is Shrink and Perturb (Ash and Adams, 2020). As the name indicates, Shrink and Perturb performs two operations: shrinking all weights and adding random Gaussian noise to these weights, see Equation 2.7 for details. The performance of Shrink and Perturb is shown in orange in Figures 4.8 and 4.9. Like L2-regularization, Shrink and Perturb prevents weight magnitude growth. Additionally, it decreases the percentage of dead units. Although its effective rank is lower than backpropagation, but it is still higher than that of L2-regularization. Shrink and

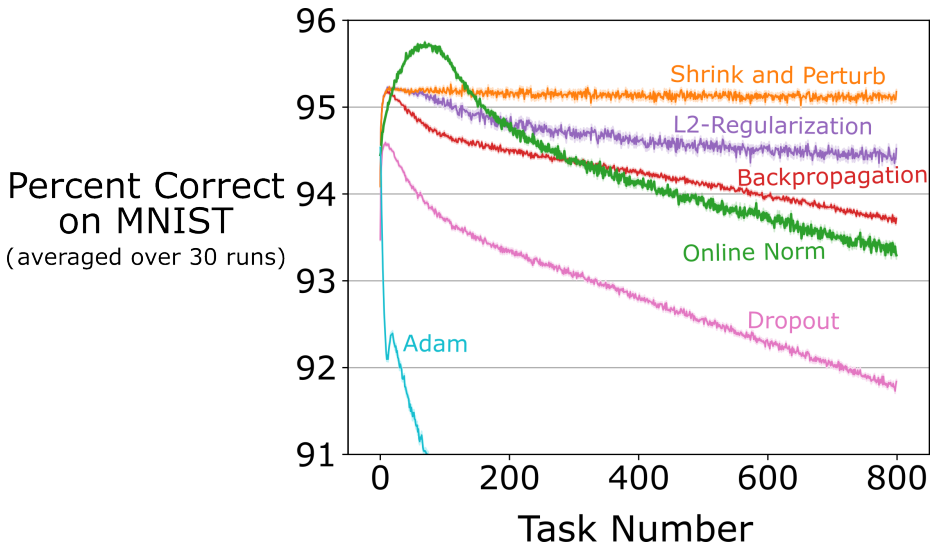

Figure 4.8: Online classification accuracy of various algorithms on Online Permuted MNIST. Three of the five methods suffer from more loss of plasticity than backpropagation. Only L2-regularization and Shrink and Perturb have higher accuracy than backpropagation throughout learning. Additionally, Shrink and Perturb have almost no drop in online classification accuracy over time.

Perturb almost completely mitigates the loss of plasticity in Online Permuted MNIST and has highest classification accuracy on the 800th task among all the methods we have tested so far.

Shrink and Perturb has two hyperparameters. First is λ, the same as in L2-regularization, and second is the variance of the noise. Figure 4.10 shows that Shrink and Perturb is sensitive to the variance of the noise. Large noise made loss of plasticity much more severe. On the other hand, small noise does not have any effect.

Dropout

Dropout is an important technique in modern deep learning (Hinton et al., 2012). The performance of Dropout is shown by pink in Figures 4.8 and 4.9. Dropout had a similar evolution of percent of dead units, weight magnitude, and effective rank as backpropagation, yet surprisingly had greater plasticity loss. Our three correlates of loss of plasticity do not explain the poor performance of Dropout, which means there are other causes of loss of plasticity.

In Dropout, the hyperparameter p refers to the probability of setting hidden units

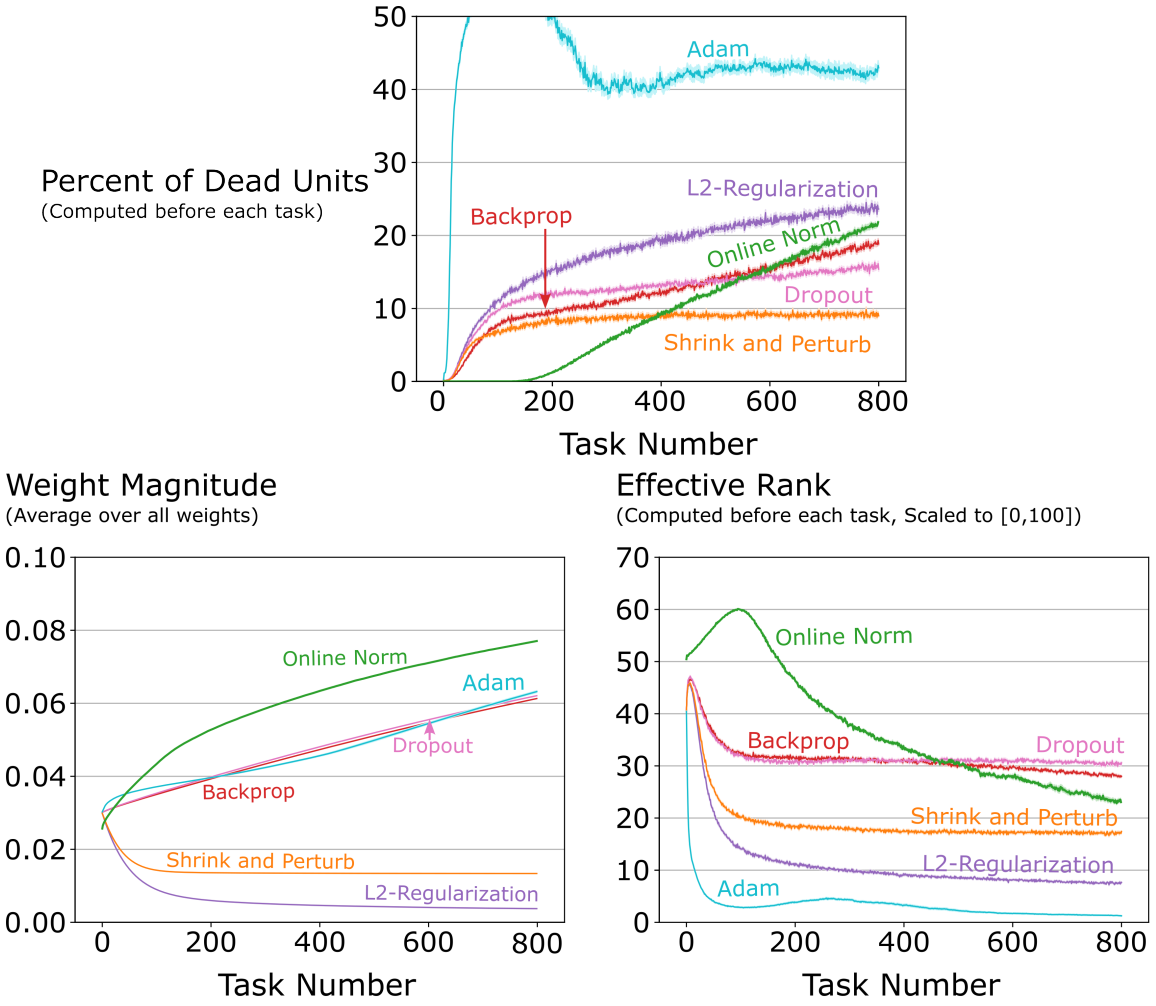

Figure 4.9: Evolution of various correlates of loss of plasticity on Online Permuted MNIST for various deep-learning algorithms. *Bottom left:* Weight magnitudes grow progressively over time across all methods except L2-regularization and Shrink and Perturb. This is because only these two methods have an explicit mechanism to stop the weights from growing. *Top:* The percentage of dead units rises over time across all methods. Shrink and Perturb keeps the fraction of dead units from growing too much. *Bottom right:* The effective rank of last layer of the representation decreases over time for all methods. Both Dropout and Shrink and Perturb stop this after around 200 tasks. The results in these plots are the average over 30 runs. The shaded regions correspond to plus and minus one standard error. For some lines, the shaded region is thinner than the line width because the standard error was small.

to zero. Among all values of p, Dropout performs best with $p = 0.01$, where its performance is almost identical to that of backpropagation. However, in Figure 4.8, we show the performance of dropout with $p = 0.1$ to show how the method performs. We found that the higher the value of p, the faster the loss of plasticity.

Online normalization

The next technique we study is batch normalization (Ioffe and Szegedy, 2015). Batch normalization changes the output of each hidden unit by normalizing and rescaling it using statistics computed from each mini-batch. We used online normalization (Chiley et al., 2019), an online variant of batch normalization, because batch normalization is not amenable to the online setting used in the Online Permuted MNIST problem. Figures 4.8 and 4.9 show the performance of online normalization in green. The network containing online normalization was expected to have fewer dead units and a higher effective rank than the network that does not. This expectation holds for earlier tasks, but both measures degrade over time. In later tasks, the network containing online normalization has a higher percentage of dead units and a lower effective rank than the network trained using backpropagation. The online classification accuracy aligns with these results. Initially, online normalization results in better online classification accuracy, but later, the classification accuracy of online normalization falls below that of backpropagation.

Online normalization contains two hyperparameters for the incremental estimation of the statistics in the normalization steps (see Equation 2.9). Changing these hyperparameters changes when the performance of the method peaked and it also has a small effect on how fast it gets to its peak performance.

Adam

Adam (Kingma and Ba, 2015) is necessary for a complete evaluation of alternative methods, as it ranks among the most valuable tools in contemporary deep learning. Adam is a variant of stochastic gradient descent that uses an estimate of the first moment of the gradient scaled inversely by an estimate of the second moment to

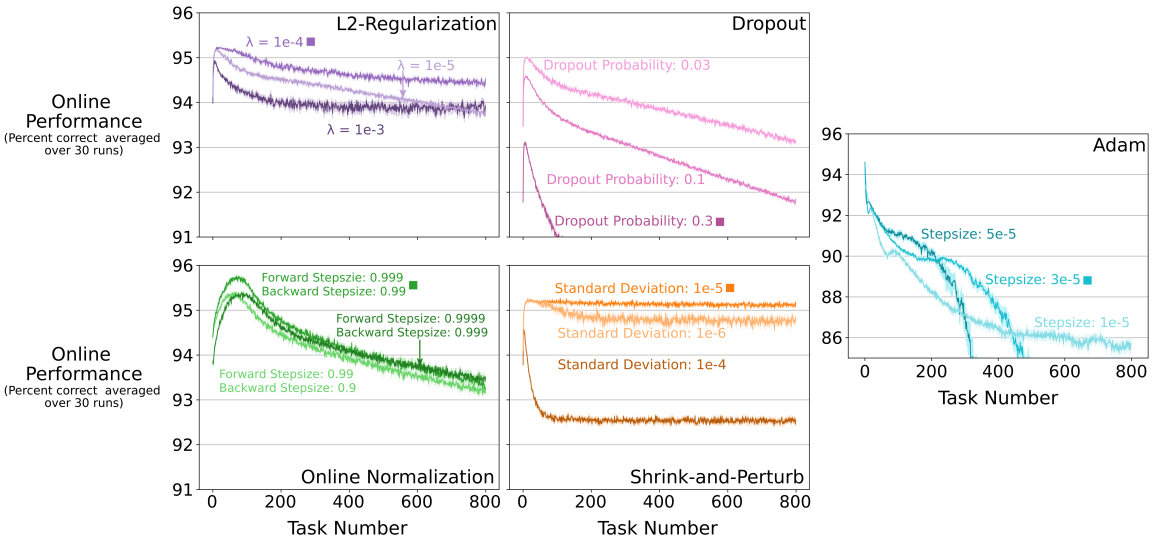

Figure 4.10: Parameter sensitivity of various algorithms on Online Permuted MNIST. Starting from the top left and proceeding clockwise, the online classification accuracy of backpropagation with L2-regularization, Dropout, Adam, Shrink and Perturb, and Online Normalization for various hyperparameter settings. We show the performance of three different hyperparameter settings for each method. The parameter settings used in Figure 4.8 are marked with a solid square next to their label. The results correspond to an average of over 30 runs for settings marked with a solid square and 10 runs for the rest. The solid lines represent the mean and the shaded regions correspond to plus and minus one standard error.

update the weights, see Equation 2.5. The performance of Adam is shown with cyan in Figures 4.8 and 4.9. Adam experiences catastrophic plasticity loss, with performance dropping dramatically. Adam also significantly worsens all three correlates of loss of plasticity. Adam has a similar weight magnitude to backpropagation. However, there is a drastic drop in the effective rank early during training and the percentage of dead units plateaus around 40%.

Adam has two hyperparameters, β_1, β_2, which are used to keep incremental estimates of the mean and square of the gradient. We used the standard of these hyperparameters, which was proposed in the original paper, i.e., $\beta_1 = 0.9, \beta_2 = 0.999$. Figure 4.10 shows the performance of Adam for different values of the step-size parameter, η. Different values of step-size change how fast the performance collapsed.

Many standard methods substantially exacerbate plasticity loss. Adam's effect on network plasticity is particularly dramatic. Networks trained with Adam have a large fraction of dead units and quickly lose almost all of their representation diversity, as measured by the effective rank. Similarly, other widely-used methods like Dropout and normalization also worsen plasticity loss. Normalization initially performs better than backpropagation, but later it has a sharper drop. Dropout simply degrades performance, the higher the Dropout probability, the larger the loss of plasticity. These results mean that some of the most successful deep learning tools do not work well in continual learning, highlighting the need to focus on developing tools specifically for continual learning.

We achieved some success in preserving plasticity within deep neural networks. L2-regularization and Shrink and Perturb both mitigate plasticity loss, with Shrink and Perturb proving particularly effective as almost entirely overcame loss of plasticity in Online Permuted MNIST. However, both methods are somewhat sensitive to hyperparameter values. They reduce plasticity loss for a small range of hyperparameters, while worsening it for other values. This sensitivity to hyper-parameters can limit the practical application of these methods to continual learning. Furthermore,

Shrink and Perturb does not completely address the three correlates of plasticity loss, it has a lower effective rank than backpropagation, and a significant fraction of units are dead.

4.4 Loss of Plasticity in Continual ImageNet

Imagenet is a large database of images and their labels (Deng et al., 2009). It has been influential throughout machine learning and it played a pivotal role in the rise of deep learning (Krizhevsky et al., 2012). ImageNet allowed researchers to conclusively show that deep learning can solve a problem like object recognition, a hallmark of intelligence. Classical machine learning methods had a classification error rate of 25% (Russakovsky et al., 2015) when deep-learning methods reduced it to below 5% (Hu et al., 2018), the human error rate on the dataset.

The ImageNet database contains millions of images labeled with nouns (classes) representing various animals. The standard ImageNet task is to guess the correct label for a given image. Our ImageNet dataset includes 1,000 classes, with 700 images per class. In the standard way to use this dataset, it is split into training and test partitions. A learning system is first trained on a set of images and their labels, then training is stopped and performance is measured on a separate set of test images from the same classes. We divided the 700 images for each class into 600 images for a training set and 100 images for a test set. We used the down-sampled 32×32 version of the ImageNet dataset, as is often done to save computation (Chrabaszcz et al., 2017).

We constructed a sequence of binary classification tasks by randomly selecting two classes from the dataset to adapt ImageNet to continual learning while minimizing all other changes. For example, the first task might be to separate crocodiles and guitars, and the second might be to separate cats from dogs. We created about half a million binary classification tasks this way, because our dataset has 1000 classes. For each task, we trained a deep learning network on images from the training sets

of the two selected classes, then evaluated its performance on a separate test set for the same classes. Specifically, the network was first trained on the 1,200 images from the training set of and then its classification accuracy was measured on the test set of 200 images. The training consisted of several passes through the training set, called epochs. For each task, all learning algorithms performed 250 epochs on the training set using mini-batches of 100 images. Following training and testing on one task, the next task began with a different pair of classes. We call this problem *Continual ImageNet*. In Continual ImageNet, task difficulty remains constant over time. A drop in performance would mean the network is losing its learning ability.

We applied a wide variety of standard deep learning networks to Continual ImageNet and tested many learning algorithms and parameter settings. Network performance on each task was assessed by measuring the percentage of correctly classified test images. The results are shown in Figure 4.11. In Figure 4.12, we plot the average classification accuracy during training on a task. These results are representative; they are for a feed-forward convolutional network and for a training procedure, using unmodified backpropagation, that performed well on this problem in the first few tasks. The network had three convolutional-plus-max-pooling layers followed by three fully connected layers. The final layer consisted of just two units, the *heads*, corresponding to the two classes. The details of the network architecture are presented in Table A.1 in Appendix A. At task changes, the input weights of the heads were reset to zero. This head resetting can be interpreted as introducing new heads for new classes. This resetting of the output weights is not ideal for studying plasticity, as the learning system gets access to privileged information on the timing of task changes (and we do not use it in other experiments in this thesis). We use it here because it is the standard practice in deep continual learning for this type of problem where the learning system faces a sequence of independent tasks (van de Ven et al., 2022).

In this problem, we reset the network head at each task's beginning. For a linear network, this means resetting the entire network, which explains why linear network

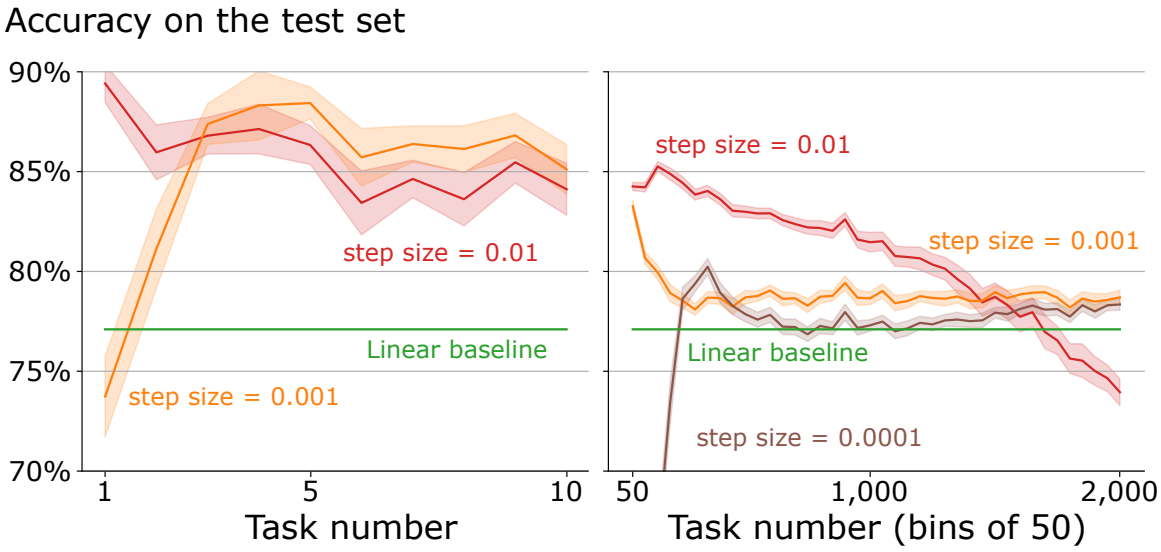

Figure 4.11: Loss of plasticity with backpropagation on Continual ImageNet. The test accuracy is measured at the end of each tasks. The first plot shows performance over the first ten tasks, which sometimes improved initially before declining. The second plot shows performance over 2000 tasks, over which the loss of plasticity was extensive. The results are averaged over 30 runs and the shaded region represents plus and minus standard error.

performance does not degrade in Continual ImageNet. Since the linear network serves as a baseline, obtaining a low-variance performance estimate is desirable. The value of this baseline is obtained by averaging over thousands of tasks. This averaging gives us a much better estimate of its performance than other networks.

The network was trained using stochastic gradient descent (SGD) with momentum on the cross-entropy loss and randomly initialized once prior to the first task. The momentum hyper-parameter was 0.9. We tested various step size parameters for backpropagation but only presented the performance for step sizes 0.01, 0.001, and 0.0001 for clarity of Figures 4.11 and 4.12. We performed 30 runs for each hyper-parameter value, varying the sequence of tasks and other randomness. Across different hyper-parameters and algorithms, the same sequences of pairs of classes were used.

Although these networks learned up to 88% correct on the test set of the early tasks (Figure 4.11, left panel), by the 2000th task they had lost substantial plasticity

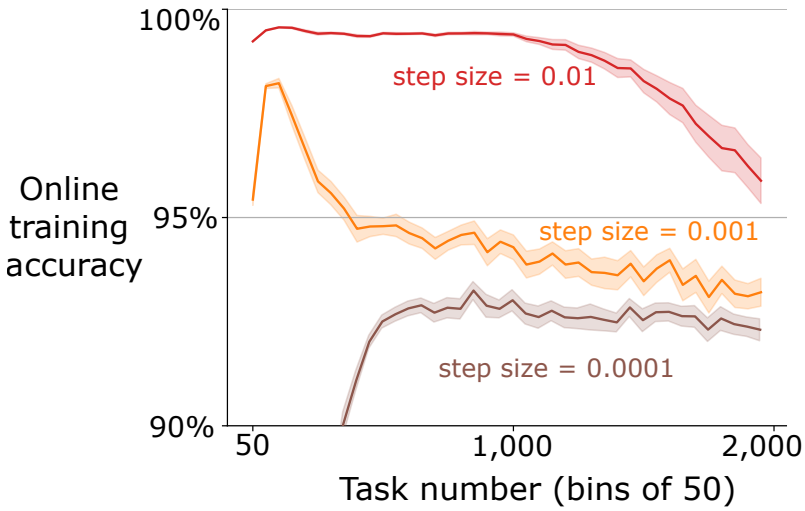

Figure 4.12: The online training accuracy also drops for backpropagation on Continual ImageNet. This means that the network struggles to even reduce the training loss in later tasks. The results are averaged over 30 runs and the shaded region represents plus and minus standard error.

for all values of the step size parameter (right panel). Some step sizes performed the best on the first two tasks, but then fell on subsequent tasks, eventually reaching a level below a linear network. For other step sizes, performance rose initially and then fell, and was only slightly better than the linear network after 2000 tasks. The drop in classification accuracy on the test set is accompanied by a similar drop in the online training accuracy (Figure 4.12). The online training accuracy for a task is the average accuracy on all the data points on which the network is being trained. We found this to be a common pattern in our experiments: for a well-tuned network, performance first improves, then falls substantially, ending near or below the linear baseline. We have observed this pattern for many network architectures, parameter choices, and optimizers. The specific choice of network architecture, algorithm parameters, and optimizers affects when the performance starts to drop, but a severe performance drop occurred for a wide range of choices. The failure of standard methods to learn better than a linear network in later tasks is substantial evidence that deep learning methods can fail in continual learning problems.

4.5 Plasticity Loss in Class-Incremental Learning

For the final demonstration, we chose to use residual networks, class-incremental continual learning, and the CIFAR-100 dataset. *Residual networks* (He et al., 2016) included layer-skipping connections in addition to the usual layer-to-layer connections of conventional convolutional networks. Today residual networks are more widely used and produce better results than strictly layered networks. We used residual networks in a *class-incremental continual learning* (Rebuffi et al., 2017) problem which involves sequentially adding new classes while testing on all classes seen so far. In our demonstration, we started with training on five classes and then successively added more, five at a time, until all 100 were available. After each addition, the networks were trained and performance was measured on all available classes. We continued training on the old classes (unlike in most work in class-incremental learning) to focus on plasticity rather than on forgetting.

We used the CIFAR-100 dataset to create the class-incremental learning problem. The dataset consists of 100 classes with 600 images each. The 600 images for each class were divided into 450 images to create a training set, 50 for a validation set and 100 for a test set. Note that the network is trained on all data from all classes available at present. First, it is trained on data from just five classes, then from all ten classes and so on, until finally, it is trained from data from all 100 classes simultaneously.

In this demonstration, we used an 18-layer residual network with a variable number of heads, adding heads as new classes were added. The details of the network architecture are presented in Table A.2 in Appendix A. We also used additional deep learning techniques, including batch normalization, data augmentation, L2 regularization, and learning rate scheduling. We call this our *base deep-learning system*.

The weights of convolutional and linear layers in the 18-layer residual network were initialized using Kaiming initialization, the weights for the batch-norm layers were

initialized to one and all of the bias terms in the network were initialized to zero. Each time five new classes were made available to the network, five more output units were added to the final layer of the network. The weights and biases of these output units were initialized using the same initialization scheme. The weights of the network were optimized using SGD with a momentum of 0.9, a weight decay of 0.0005 and a mini-batch size of 90.

After each increment, the network was trained for 200 epochs, for a total of 4,000 epochs across all 20 increments. We used a learning rate schedule that reset at the start of each increment. For the first 60 epochs of each increment, the learning rate was set to 0.1, then to 0.02 for the next 60 epochs, then 0.004 for the next 40 epochs, and to 0.0008 for the last 40 epochs. During the 200 epochs of training for each increment, we kept track of the network with the best accuracy on the validation set. To prevent overfitting, at the start of each new increment, we reset the weights of the network to the weights of the best-performing (on the validation set) network found during the previous increment; this is equivalent to early stopping for each different increment.

We used several steps of data preprocessing before the images were presented to the network. First, the value of all the pixels in each image was rescaled between 0 and 1 by dividing by 255. Then, each pixel in each channel was centred and rescaled by the average and standard deviation of the pixel values of each channel, respectively. Finally, we applied three random data transformations to each image before feeding it to the network: randomly horizontally flip the image with a probability of 0.5, randomly crop the image by padding the image with 4 pixels on each side, and randomly cropping to the original size, and randomly rotate the image between 0 and 15 degrees. The first two steps of preprocessing were applied to the train, validation, and test set, but the random transformations were only applied to the images in the train set.

As more classes are added, correctly classifying images becomes more difficult,

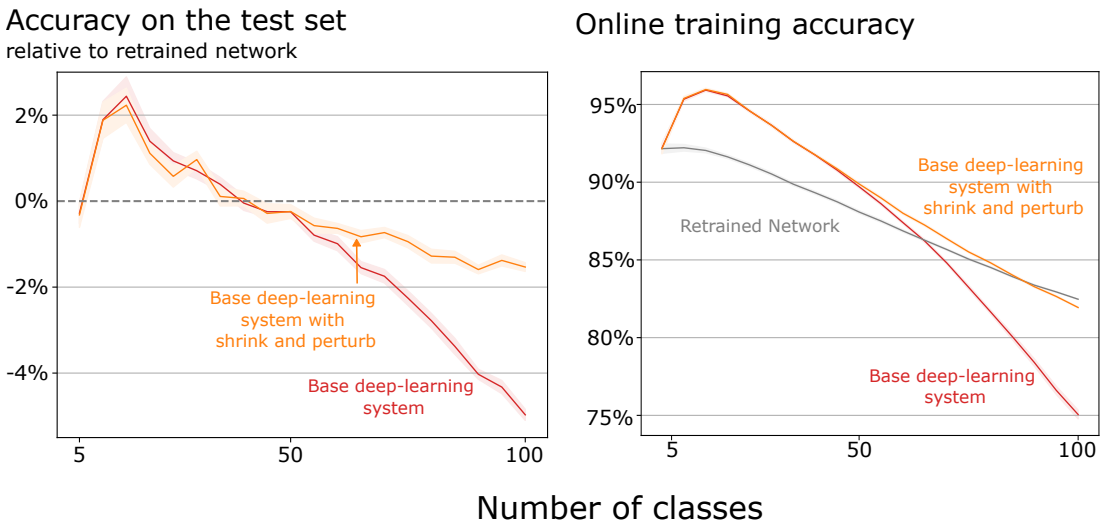

Figure 4.13: Loss of plasticity in class-incremental CIFAR-100. Initially, incremental training produced benefits compared to a network retrained from scratch, but after 50 classes it produced a substantial loss of plasticity in the base deep-learning system. The Shrink and Perturb algorithm lost less plasticity.

and classification accuracy would decrease even if the network maintained its ability to learn. To factor out this effect, we compare the accuracy of our incrementally-trained networks with networks that were retrained from scratch on the same subset of classes. For example, the network that was trained first on five classes, and then on all ten classes, is compared to a network retrained from scratch on all ten classes. If the incrementally-trained network performs better than a network retrained from scratch, then there is a benefit due to training on previous classes, and if it performs worse, then there is a genuine loss of plasticity.

The red line in the left panel of Figure 4.13 shows that incremental training was initially better than retraining, but after forty classes the incrementally-trained system showed loss of plasticity that became increasingly severe. By the end, when all 100 classes were available, the accuracy of the incrementally-trained base system was 5% lower than the retrained network (a performance drop equivalent to that of removing a major algorithmic advance, such as batch normalization). We observed a similar trend for online training accuracy, see right panel of Figure 4.13. The loss of

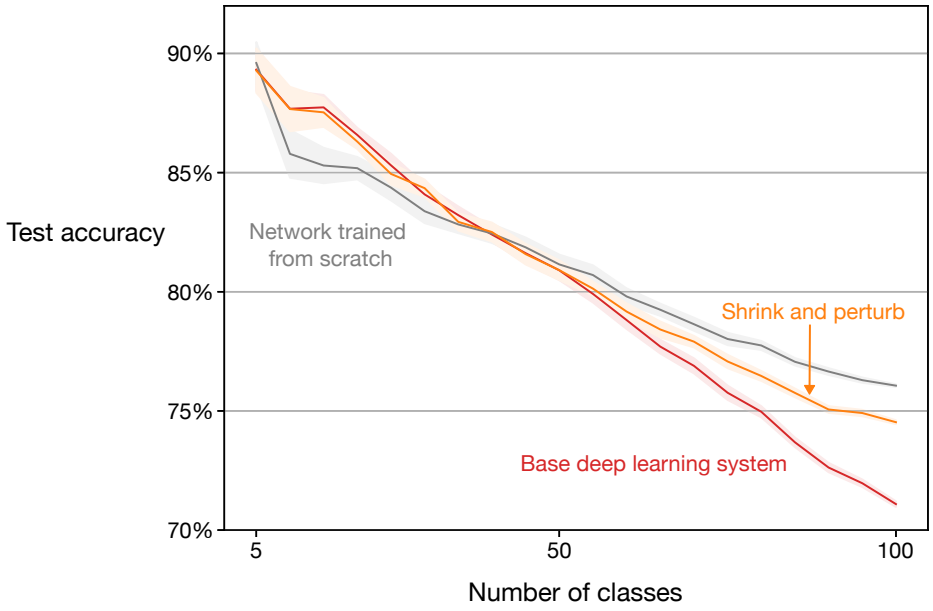

Figure 4.14: Test accuracy in class-incremental CIFAR-100. As more classes are added, the classification becomes harder and algorithms naturally show decreasing accuracy with more classes. Each line corresponds to the average of 15 runs.

plasticity was less severe if Shrink and Perturb was added to the learning algorithm (in the incrementally-trained network), as shown by the orange line in Figures 4.13 and 4.13. For completeness, Figure 4.14 shows the test accuracy of each algorithm in each different increment. The final accuracy of the base deep-learning system on all 100 classes was about 71%, while that of the network trained from scratch was about 76%.

We tested several hyperparameters to ensure the best performance for each different algorithm with our specific architecture. For the base system, we tested values for the weight decay parameter in 0.005, 0.0005, 0.00005. A weight-decay value of 0.0005 resulted in the best performance in terms of area under the curve for accuracy on the test set over the 20 increments. For Shrink and Perturb, we used the weight-decay value of the base system and tested values for the standard deviation of the Gaussian noise in $\{10^{-4}, 10^{-5}, 10^{-6}\}$; 10^{-5} resulted in the best performance.

We plot the evolution of dormant units in the network and the stable rank of the representation in the penultimate layer of the network. We call a unit to be dormant

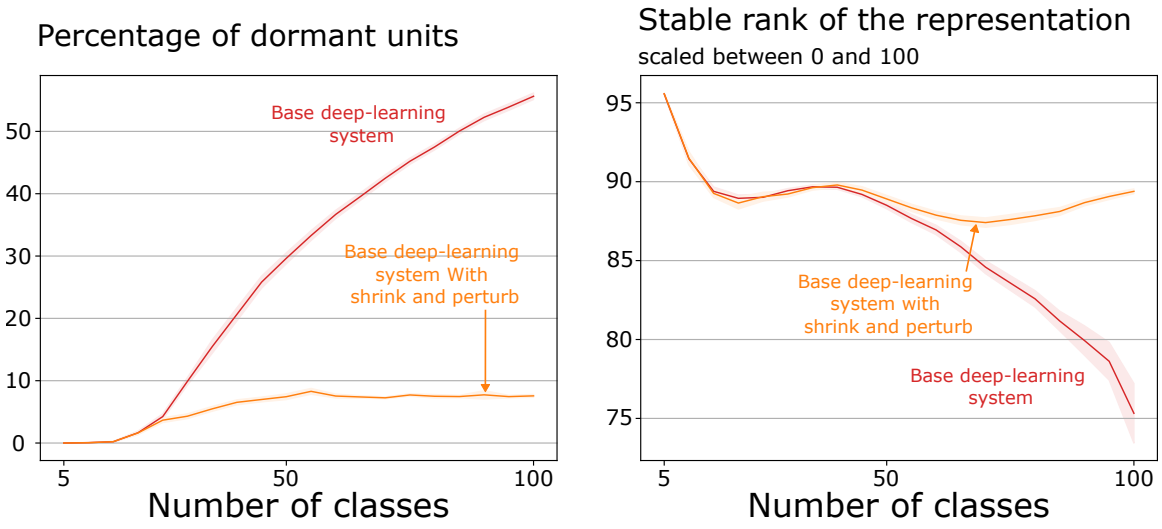

Figure 4.15: *Left*: A dormant unit in a network is one that is active less than 1% of the time. The number of these increases rapidly with the base deep-learning system, but less so with Shrink and Perturb. *Right*: A low stable rank means a network's units do not provide much diversity. The base deep-learning system loses much more diversity than Shrink and Perturb.

if it is active less than 1% of the time. For a matrix $\Phi \in \mathbb{R}^{n \times m}$ with singular values σ_k sorted in descending order for $k = 1, 2, ..., q$, and $q = \max(n, m)$, the stable rank is $\min \left\{ k : \frac{\Sigma_i^k \sigma_i}{\Sigma_i^d \sigma_i} > 0.99 \right\}$ (Yang et al., 2019).

Figure 4.15 revealed a similar trend as we saw in the permuted MNIST problem seen in Section 4.2. In the 18-layer residual network, the fraction of dormant units dramatically increases. By the time of the last increment, more than 50% of the units in the network are dormant (left panel), meaning that effectively half of the network is not contributing to learning. Similarly, the stable rank of the representation drops to about 75 by the last increment, meaning that a large part of the representation is not providing unique information. The orange lines in the Figure 4.15 show that Shrink and Perturb, significantly reduced the number of dormant units in the network and kept a high stable rank.

This demonstration used larger networks and required more computation, but still we were able to do extensive systematic tests. We found a robust pattern to the

results that was similar to what we found in ImageNet and Permuted MNIST. In all cases, deep learning systems exhibit substantial loss of plasticity.

4.6 Discussion

The phenomenon of loss of plasticity has gained significant attention in the literature since we first made our work publicly available in 2021 (Dohare et al., 2021). Various people have studied loss of plasticity from different perspectives and added evidence to the phenomenon in continual supervised learning (Lyle et al., 2023; Nikishin et al., 2023; Kumar et al., 2024; Dohare et al., 2023; Lewandowski et al., 2023; Lee et al., 2023; Elsayed and Mahmood, 2024; Lee et al., 2024; Dohare et al., 2024; Lewandowski et al., 2025b), reinforcement learning problems (Nikishin et al., 2022; Lyle et al., 2022; Abbas et al., 2023; Sokar et al., 2023; Nikishin et al., 2023; Lyle et al., 2024a; Delfosse et al., 2024; Ahn et al., 2025), multi-agent systems (Zang et al., 2025), and modality-incremental learning of large foundation models (Zhang et al., 2025).

There are many desirable properties for an efficient continual- learning system (Veniat et al., 2021; Verwimp et al., 2024). It should be able to keep learning new things, control what it remembers and forgets, have good computational and memory efficiency and use previous knowledge to speed up learning on new data. The choice of the benchmark affects which property is being focused on. Most benchmarks and evaluations in this chapter only focused on plasticity but not on other aspects, such as forgetting and speed of learning. For example, in Continual ImageNet, previous tasks are almost never repeated, which makes it effective for studying plasticity but not forgetting. In permuted MNIST, consecutive tasks are largely independent, which makes it suitable for studying plasticity in isolation. However, this independence means that previous knowledge cannot substantially speed up learning on new tasks. On the other hand, in class-incremental CIFAR-100, previous knowledge can substantially speed up learning of new classes. Overcoming loss of plasticity is an important, but still the first step towards the goal of fast learning on future data. Once we have

networks that maintain plasticity, we can develop methods that use already acquired knowledge to speed up learning on future data.

In the last few years, significant progress has been made in our understanding of the phenomenon of loss of plasticity. Loss of plasticity can be generally categorized in two different cases (Lee et al., 2024). The first is where the algorithm is unable to optimize new objectives, this corresponds to the loss of the ability to reduce training losses for new tasks. The second is where the system can keep optimizing new objectives but lose the ability to generalize (Ash and Adams, 2020; Berariu et al., 2021). The type of loss of plasticity studied in this dissertation is largely because of the loss of the ability to optimize new objectives, as can be seen by reducing online training accuracies. However, it is unclear if the two types of plasticity loss are fundamentally different or if the same mechanism can explain both phenomena. Some methods, such as Shrink and Perturb, that are able to bring back all advantages of random initialization, are effective for mitigating both types of loss of plasticity. Future work that improves our understanding of plasticity and finds the underlying causes of both types of plasticity loss will be valuable.

In this chapter, we identified three correlates of loss plasticity, namely, increasing number of dormant units, decreasing stable rank, and growing weight magnitudes. Few recent works have focused on identifying mechanisms that cause the loss of plasticity. They have mostly focused on the form of plasticity loss where the learning system loses the ability to optimize. Lyle et al. (2024b) say that it is best to think of the loss of plasticity being caused by multiple independent causes, and they show that no single mechanism fully explains the loss of plasticity in all cases. They show that the different mechanisms for loss of plasticity include growth of weight magnitude, varying output scale (as happens when predicting value functions in reinforcement learning), and pre-activation distribution shift (this covers the case of dormant units). Their work argues that no single cause can explain the loss of plasticity. In contrast, more recent work by Lewandowski et al. (2023) shows that in many cases, a reduction

in the number of curvature directions in the loss landscape coincides with the loss of plasticity. And they suggest that a reduction in the number of curvature directions could explain loss of plasticity on its own. It remains to be seen why exactly gradient descent loses a number of curvature directions. Finding deeper mechanisms for loss of plasticity remains an important area of research for improving our understanding of the phenomenon and potentially developing better methods for mitigating the issue.

Loss of plasticity might also be connected to the lottery ticket hypothesis (Frankle and Carbin, 2019). The hypothesis states that randomly initialized networks contain subnetworks that can achieve performance close to that of the original network with a similar number of updates. These subnetworks are called winning tickets. We found that, in continual-learning problems, the effective rank of the representation at the beginning of tasks reduces over time. In a sense, the network obtained after training on several tasks has less randomness and diversity than the original random network. The reduced randomness might mean that the network has fewer winning tickets. And this reduction in the number of winning tickets might explain loss of plasticity. Our understanding of loss of plasticity could be deepened by fully exploring its connection with the lottery ticket hypothesis.

Loss of plasticity is a critical factor when learning continues for many tasks, but it might be less important if learning happens for a small number of tasks. Usually, the learning system can take advantage of previous learning in the first few tasks. For example, in class-incremental CIFAR-100 (Fig. 2), the base deep-learning systems performed better than the network trained from scratch for up to 40 classes. This result is consistent with deep-learning applications in which the learning system is first trained on a large dataset and then fine-tuned on a smaller, more relevant dataset. Plasticity-preserving methods may still improve performance in such applications based on fine-turning, but we do not expect that improvement to be large, as learning happens only for a small number of tasks. We have observed that deep-learning systems gradually lose plasticity, and this effect accumulates over tasks. Loss of

plasticity becomes an important factor when learning continues for a large number of tasks; in class-incremental CIFAR-100, the performance of the base deep-learning system was much worse after 100 classes.

4.7 Conclusion

Deep learning is a useful tool for settings where learning occurs in a special training phase and not afterwards. However, in settings where learning continues for a long time, we showed that deep learning does not work. By deep learning, we refer to the commonly used algorithms for learning in artificial neural networks and by not work, we mean that they lose the ability to learn new things over time. In this chapter, we provided demonstrations of loss of plasticity using datasets where deep learning methods have been successful. We showed such loss of plasticity in a wide range of problems. Our experiments covered cases that include a wide range of memory constraints, from online permuted MNIST on one end, where no old data can be stored, to class-incremental CIFAR-100, where all the data is stored and used for learning. These experiments included multiple network architectures, from simple feedforward networks in online permuted MNIST to deep residual networks in class-incremental CIFAR-100. We demonstrated loss of plasticity with over- and under-parameterized networks and different optimizers using the online permuted MNIST problem. We also found that methods like dropout, regularization, and normalization are insufficient to overcome the loss of plasticity in online permuted MNIST or class-incremental CIFAR-100. Overall, these chapter provide substantial evidence that loss of plasticity is a widespread problem in deep learning.

Chapter 5

Formalizing the Phenomenon of Loss of Plasticity

In the last chapter, we provided various demonstrations of loss of plasticity in continual supervised learning problems. Those demonstrations involved a sequence of tasks. In some problems, like Online Permuted MNIST and Continual ImageNet, the tasks were, on average, of equal difficulty. The worsening performance over time in a sequence of equally difficult tasks demonstrated a loss of plasticity. On the other hand, in the class-incremental CIFAR-100, the tasks got more difficult over time. In this problem, the worse performance of the learning system compared to a retrained system was evidence of loss of plasticity. We also saw that many units can die inside the network during continual learning. Dead units, by definition, do not contribute to the network and have lost all their plasticity because they cannot learn new things.

In this chapter, we look at various attempts at formalizing the phenomenon. We look at the pros and cons of these formalisms in defining the phenomenon. The intuition gained from the demonstrations in the last chapter will help us check if a given formalism captures all instances of loss of plasticity.

The first question on the path to formalization is: Who loses plasticity? Is it a property of the network or optimizer (eg, SGD or Adam) or the entire learning system? Consider the case where the optimizer's step size decreases to zero in a continual learning problem. This system loses plasticity as it loses the ability to learn

new things. It is unreasonable to say that the network has lost plasticity because it can learn new things if we use a different optimization algorithm. The remaining question is: Is loss of plasticity a property of the optimizer or the entire learning system? To answer this, consider a case where the network contains just one linear layer, it uses stochastic gradient descent and faces a sequence of regression problems. Lewandowski et al. (2025b) showed that there is no loss of plasticity in this setting. It is a case where stochastic gradient descent does not lose plasticity; this means that loss of plasticity is not a property of the optimizer. By the process of elimination, it is most appropriate to think of loss of plasticity as a property of the entire learning system.

To formalize the phenomenon, consider online supervised learning problems, where the learning network, $f_{\boldsymbol{\theta}} : \mathbb{R}^n \to \mathbb{R}^m$, receives a sequence of training examples, $(\mathbf{x}_t, \mathbf{y}_t)$, where $\mathbf{x}_t \in \mathbb{R}^n$ and $\mathbf{y}_t \in \mathbb{R}^m$. Recall from Chapter 2 that $\boldsymbol{\theta}$ represents the network parameters and the loss function is $\ell : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$. Let $(\mathbf{x}_t, \mathbf{y}_t)$ be sampled from some distribution p_t. The learning objective at time t is $J_t(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x}_t, \mathbf{y}_t) \sim p_t}[\ell(f_{\boldsymbol{\theta}_t}(\mathbf{x}_t), \mathbf{y}_t)]$. The learning algorithms changes the network parameter to minimize $J_t(\boldsymbol{\theta})$. In many cases, p_t does not change at each step. For example, in the experiments we used in the previous chapter, p_t stayed the same for duration of a task. However, in general, in continual learning, p_t can change at each step.

5.1 Loss of Plasticity as Decreasing Performance

The first idea for formalizing loss of plasticity is to define it as a drop in performance over a sequence of tasks. Let's assume that the examples can be seen as a sequence of tasks, where the defining characteristic of a task is that distribution from which examples $(\mathbf{x}_t, \mathbf{y}_t)$ are sampled remains the same. Let task τ last from $t \in \{(\tau - 1)T + 1, (\tau - 1)T + 2, \ldots, \tau T\}$, for some constant T, and let the distribution for this task

be p_τ. The expected loss on task τ is

$$\frac{1}{T}\sum_{t=(\tau-1)T+1}^{\tau T} \mathbb{E}_{(\mathbf{x}_t,\mathbf{y}_t)\sim p_\tau}[\ell(f_{\boldsymbol{\theta}_t}(\mathbf{x}_t),\mathbf{y}_t)]. \tag{5.1}$$

In this setting, we assume that tasks are of equal difficulty. An increase in expected loss across tasks in this setting means the system loses the ability to optimize new objectives. This is a useful definition of the phenomenon. It fully captures the drop in performance that we saw in the Online Permuted MNIST and Continual ImageNet problems in the last chapter.

However, this definition does not capture the case of class-incremental learning, where the problem's difficulty changes over time. In class-incremental problems, the classification problem becomes more difficult over time, and the performance drops even if the algorithm does not lose plasticity. This definition falls short of a fully satisfactory formalization of the phenomenon, as it does not capture the case where the difficulty of the problem varies over time. Lewandowski et al. (2025a) used this definition and also pointed out the limitation of this definition.

5.2 Loss of Plasticity as Worse Performance than Retraining

In class-incremental CIFAR-100, we said that a learning algorithm has lost plasticity if it performed worse than a randomly initialized network trained on the same task. We can formalize that intuition in a similar manner to the previous section. Again, let's assume that the examples can be seen as a sequence of tasks, where the distribution from which examples $(\mathbf{x}_t, \mathbf{y}_t)$ are sampled remains the same during a task. However, this time, the tasks are not of equal length and difficulty. Let task τ last from $t \in \{t_{\tau-1}+1, t_{\tau-1}+2, \dots t_\tau\}$, and $t_0 = 0$. Note that unlike last section, tasks are of different lengths in this setting, which requires us to explicitly specify the start time of task as $t_{\tau-1}$. Let the distribution of examples for this task be p_τ. Again, the

expected loss for a learning system on task τ is

$$\frac{1}{t_\tau - t_{\tau-1}} \sum_{t=t_{\tau-1}+1}^{t_\tau} \mathbb{E}_{(\mathbf{x}_t, \mathbf{y}_t) \sim p_\tau}[\ell(f_{\boldsymbol{\theta}_t}(\mathbf{x}_t), \mathbf{y}_t)]. \tag{5.2}$$

In this setting, we say that the system has lost plasticity if its expected loss during a task is larger than if the same system used a network that was randomly initialized at $t_{\tau-1} + 1$, i.e.,

$$\frac{1}{t_\tau - t_{\tau-1}} \sum_{t=t_{\tau-1}+1}^{t_\tau} \mathbb{E}_{(\mathbf{x}_t, \mathbf{y}_t) \sim p_\tau}[\ell(f_{\boldsymbol{\theta}_t}(\mathbf{x}_t), \mathbf{y}_t)] - \mathbb{E}_{(\mathbf{x}_t, \mathbf{y}_t) \sim p_\tau}[\mathbb{E}_{\boldsymbol{\theta'}_{t_{\tau-1}+1}}[\ell(f_{\boldsymbol{\theta'}_t}(\mathbf{x}_t), \mathbf{y}_t)]] > 0 \tag{5.3}$$

where $\boldsymbol{\theta'}_t$ are the parameters of an alternative network at time t. The alternative network is randomly initialized at then $t_{\tau-1}+1$ and then trained using the same algorithm from that point onward. This definition of loss of plasticity captures the performance of backpropagation for all three problems we used in the previous chapter. In all three problems, the performance of backpropagation is worse at the last task than at the first task. Lyle et al. (2023) used a similar definition of the phenomenon.

This definition fails to capture an important case of the phenomenon, which is the pretraining-finetuning setting. In many deep learning applications, the network is first trained on a large dataset and then fine-tuned on a small dataset of interest. The small dataset generally only has a few hundred or thousand examples. A network trained on just the small dataset has very poor performance. In this case, if the pretrained system has lost some plasticity, its performance will still be substantially better than a system trained from scratch, see Figure 5.1 for an illustration. The pretrained system could have lost plasticity if some fraction, say 10%, of the units died out in its network. In this case, even if the system loses plasticity, it will perform better than a system trained from scratch, and the current definition of loss of plasticity will not capture this case.

todo: Add a copy of Figure 4.10 here for illustration This definition also fails to capture a key aspect of the phenomenon, i.e., reducing performance. Sometimes

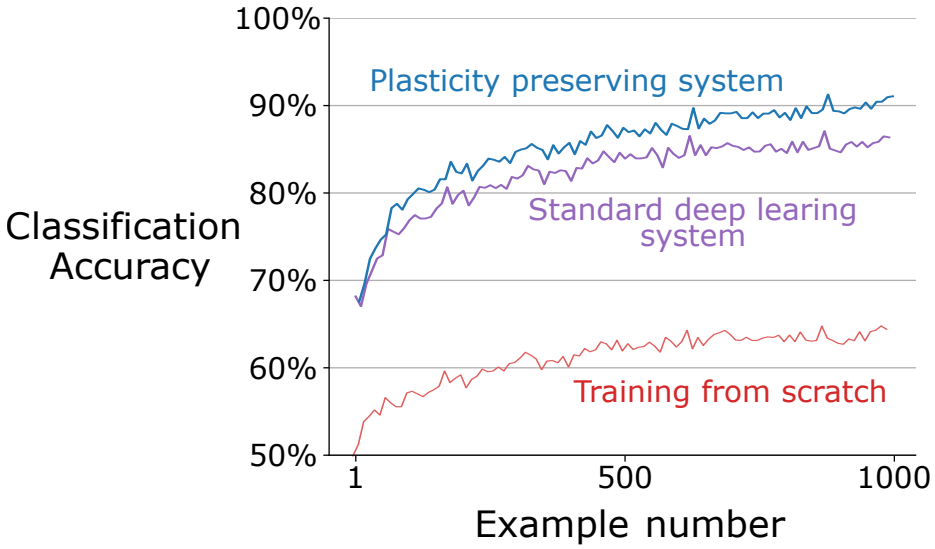

Figure 5.1: Hypothetical performance of various algorithms in a pretraining-finetuning problem. A randomly initialized network trained on the fine-tuning data does not perform well. The base deep learning system was pretrained on a large dataset, substantially outperforming training from scratch. However, it might have lost plasticity during training. A plasticity-preserving algorithm with the standard deep learning system can outperform the base system. Comparing performance to a network trained from scratch does not tell us if a system has lost plasticity in this case. In practice, it is difficult to answer whether a system has lost plasticity in a pretraining-finetuning problem.

the performance of a network starts to drop, but it never gets below that of a network trained from scratch (see L2 regularization in Figure 4.10). In such cases, there is some loss of plasticity, but this definition of loss of plasticity fails to capture this case. Although the current definition is helpful and captures a wide range of cases seen in the last chapter, it is insufficient.

5.3 Plasticity Loss as Increasing Dynamic Regret

The final definition of loss of plasticity combines the key ideas of both previous definitions. It defines loss of plasticity as a drop in performance with respect to a baseline. However, this time, the baseline is not retraining but the optimal parameters. Let $\boldsymbol{\theta}_t^*$ be the value of the parameter vector that minimizes $J_t(\boldsymbol{\theta})$. Note that $\boldsymbol{\theta}_t^*$ can be

different at each step. Then the average expected dynamic regret is

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{(\mathbf{x}_t,\mathbf{y}_t)\sim p_t}[\ell(f_{\boldsymbol{\theta}_t}(\mathbf{x}_t),\mathbf{y}_t)] - \mathbb{E}_{(\mathbf{x}_t,\mathbf{y}_t)\sim p_t}[\ell(f_{\boldsymbol{\theta}_t^*}(\mathbf{x}_t),\mathbf{y}_t)]. \tag{5.4}$$

In this case, a system is called to lose plasticity if the average expected regret increases over time. Farias and Jozefiak (2025) proposed this definition of loss of plasticity. It captures the phenomenon for all the examples we saw in the previous chapter and the pretraining-finetuning setting discussed in the last section. In practice, $\boldsymbol{\theta}^*$ is not available, which limits the utility of this definition in practice. However, some lower bounds of optimal performance, like the performance of a retrained network, are available in many cases. The one unspecified component in this definition is the distribution p_t. We must carefully think about specific distribution before we can call an increase in regret a loss of plasticity. One case where an increasing regret does not mean a loss of plasticity is when the magnitude for data points increases over time, for example, if $\mathbb{E}[\|\mathbf{x}_t\|_2] = 10 * \mathbb{E}[\|\mathbf{x}_{t-1}\|_2]$. In this case, the regret can increase just because the magnitude of the data point increases, but there is no loss of the ability to learn. Despite its limitations, this definition captures the key intuition behind our informal definition, i.e. loss of the ability to learn new things.

5.4 Conclusion and Discussion

In this chapter, we looked at three formal definitions of loss of plasticity as well as the limitations of each definition. Defining the phenomenon as a reduction in performance is useful in cases where the problem's difficulty stays the same over time. Nevertheless, it fails in cases where the problem's difficulty changes over time. On the other hand, defining the phenomenon as worse performance compared to a retrained network fails in the pretraining-finetuning setting. Finally, defining the phenomenon as increasing dynamic regret can fail in cases when the scale of data distribution increases over time. However, I believe loss of plasticity as increasing dynamic regret is the most appropriate formalization of the phenomenon because it captures the key idea: losing

the ability to learn new things.

The definitions in this chapter have one major difference from the experimental setting used in Chapter 4, namely, the use of held-out test sets. We used held-out test sets in Chapter 4 for two reasons. First, we wanted to stay close to the standard deep learning setting. Second, the existing supervised learning datasets are not sufficiently rich and complex to capture the generalization requirements of real-world data streams. The real world contains long data streams with abundant data, but the system still needs to generalize well to achieve good performance. However, existing supervised learning datasets are too small to effectively simulate the need for generalization. Held-out test sets allow us to evaluate the generalization capabilities of learning systems without needing massive labeled datasets. As the field of continual learning grows, the need for held-out test sets will diminish as we find more realistic data streams without any separation between a training and test set.

Chapter 6

Maintaining Plasticity via Selective Reinitialization

Previous chapters in this thesis have focused on establishing and understanding the phenomenon of plasticity loss. The next step is to develop algorithms that can maintain plasticity in continual learning problems. To this end, we develop the *continual backpropagation* algorithm. This chapter describes the algorithm and evaluates its effectiveness on all the problems where we observed loss of plasticity in Chapter 4.

We start with motivation and a description of the continual backpropagation algorithm. Then, we deeply dive into the algorithm to understand its hyperparameters and the behaviors they induce. Next, we evaluate the algorithms on various problems where we observed loss of plasticity and look at how it affects various properties of the networks. We end with a discussion of algorithms that have been built on top of continual backpropagation or have been proposed to overcome loss of plasticity.

6.1 Description of Continual Backpropagation

In Chapter 4, we learned that Shrink and Perturb reduces the loss of plasticity in many cases. The injection of variability into the network through the Shrink and Perturb process reduces dormancy and increase the diversity of the representation. However, the continual injection of randomness in Shrink and Perturb is tied to the idea of shrinking the weights. There exists prior work (Mahmood and Sutton, 2013;

Mahmood, 2017) that proposed a more direct way of continually injecting randomness by selectively reinitializing low-utility units in the network. But the ideas presented were not developed for deep networks and could not be used with modern deep learning. We fully developed the idea of selective reinitialization so it can be used with modern deep learning. The resulting algorithm is continual backpropagation and it combines conventional backpropagation with selective reinitialization.

In one sense, continual backpropagation is a simple and natural extension of the conventional backpropagation algorithm to continual learning. The conventional backpropagation algorithm has two parts: initialization with small random weights and gradient descent at each step. This algorithm is designed for the stationary setting, where learning only happens once. The initialization provides variability initially, but, as we have seen, in continual learning, variability tends to be lost, as well as plasticity along with it. To maintain the variability, our new algorithm, continual backpropagation, reinitializes a small number of units throughout training. The key principle behind continual backpropagation is that good continual learning algorithms should do time-symmetric computations, that is, similar computations at all times.

Continual backpropagation selectively reinitializes low-utility units in the network. Our utility measure, called the *contribution utility*, is defined for each connection or weight and each unit. The basic intuition behind the contribution utility is that the magnitude of the product of units' activation and outgoing weight gives information about how valuable this connection is to its consumers. If a hidden unit's contribution to its consumer is small, its contribution can be overwhelmed by contributions from other hidden units. In such a case, the hidden unit is not useful to its consumer. We define the contribution utility of a hidden unit as the sum of the utilities of all its outgoing connections. The contribution utility is measured as a running average of instantaneous contributions with a decay rate, η. In a feed-forward network, the

contribution-utility, $\mathbf{u}_l[i]$, of the ith hidden unit in layer l at time t is updated as

$$\mathbf{u}_l[i] \leftarrow \eta * \mathbf{u}_l[i] + (1-\eta) * |\mathbf{h}_{l,t}[i]| * \sum_{k=1}^{n_{l+1}} |\mathbf{W}_{l,t}[i,k]|, \tag{6.1}$$

where $\mathbf{h}_{l,t}[i]$ is the output of the i^{th} hidden unit in layer l at time t, $\mathbf{W}_{l,t}[i,k]$ is the weight connecting the i^{th} unit in layer l to the k^{th} unit in layer $l+1$ at time t, n_{l+1} is the number of units is layer $l+1$.

When a hidden unit is reinitialized, its incoming weights are randomly sampled from the same distribution that was used to initialize the weights in the beginning, and its outgoing weights are set to zero. Initializing the incoming weights randomly injects variability into the network. At the same time, initializing the outgoing weights as zero ensures that the newly added hidden units do not affect the already learned function. However, initializing the outgoing weight to zero makes the new unit vulnerable to immediate reinitialization as it has zero utility. To protect new units from immediate reinitialization, they are protected from a reinitialization for *maturity threshold*, m, number of updates.

Continual backpropagation finds low-utility units in each layer of the network. This means that the utility of a unit is only compared to other units in the same layer. The hyperparameter *replacement rate*, $\rho \in [0,1]$, controls the number of units that are initialized per layer per step. The replacement rate is the fraction of eligible units that are reinitialized in every layer at each step. A unit is called *eligible* if its age exceeds the maturity threshold, m, where m is a non-negative integer. For example, if ρ is 0.001 and the layer has n_e eligible units, and let $n_e = 1000$, then at that step, $\rho * n_e$ is one. This means that one unit would be reinitialized at that step. ρ is typically very small, meaning that $\rho * n_e$ is less than one. In such cases, $\rho * n_e$ is accumulated in another variable called c. When c becomes greater than one, the unit with the smallest utility is reinitialized. For example, if ρ is 10^{-5}, and $n_e = 1000$, then $\rho * n_e$ is just 0.01. In this case, one unit is reinitialized after every hundred steps.

The final algorithm combines conventional backpropagation with selective reini-

Algorithm 1: Continual backpropagation (CBP) for a feed-forward network with L hidden layers

Set: step size α, replacement rate ρ, decay rate η, and maturity threshold m (e.g. 10^{-4}, 10^{-4}, 0.99, and 100)

Initialize: Initialize the weights $\mathbf{W}_0, ..., \mathbf{W}_L$. Let, $\mathbf{W}_l$ be sampled from a distribution d_l

Initialize: Utilities $\mathbf{u}_1, ..., \mathbf{u}_L$, accumulated number of features to reinitialize $c_1, ..., c_l$, and ages $\mathbf{a}_1, ..., \mathbf{a}_L$ to 0

for *each input* x_t **do**

- **Forward pass:** pass input through the network, get the prediction, $\hat{y}_t$
- **Evaluate:** Receive loss $l(f_{\boldsymbol{\theta}_t}(\mathbf{x}_t), \mathbf{y}_t)$
- **Backward pass:** update the weights using stochastic gradient descent
- **for** *layer* l *in* $1 : L$ **do**
  - **Update age:** $\mathbf{a}_l \mathrel{+}= 1$
  - **Update unit utility:** Using Equation 6.1
  - **Find eligible units:** $n_{eligible}$ = Number of units with age greater than m
  - **Update accumulated number of units to reinitialize:** $c_l = c_l + n_{eligible} * \rho$
  - **if** $c_l > 1$ **then**
    - **Unit to reinitialize:** Find the unit with smallest utility, let its index be r
    - **Initialize input weights:** Reset the input weights $\mathbf{W}_{l-1}[:, \mathbf{r}]$ using samples from d_l
    - **Initialize output weights:** Set $\mathbf{W}_l[\mathbf{r}, :]$ to zero
    - **Initialize utility, unit activation, and age:** Set $\mathbf{u}_l[r]$ and $\mathbf{a}_l[r]$ to 0
    - **Update accumulated number of units to reinitialize:** $c_l = c_l - 1$

tialization to continually inject random units from the initial distribution. Continual backpropagation performs a gradient-descent and selective reinitialization step at each update. Algorithm 1 specifies the continual backpropagation algorithm for a feed-forward network. In cases where the learning system uses mini-batches, the instantaneous contribution utility can be used instead of keeping a running average to save computation.

The selective reinitialization in continual backpropagation can also be understood as a search process in the space of representation units or artificial neurons. Continual backpropagation builds on a long line of research on conducting search in the space of representation units (Selfridge, 1958; Mucciardi and Gose, 1966; Klopf and Gose, 1969; Holland and Reitman, 1977; Holland, 1992; Kaelbling, 1993; Stanley and Miikkulainen, 2002; Mahmood and Sutton, 2013; Mahmood, 2017). The idea of searching in the space of representations was first proposed by Selfridge (1958). They proposed to use a network of representation units. The final output of the network is a linear combination of the representation units, and the weights for the linear combination are learned by zeroth-order optimization. The utility measure proposed by Selfridge (1958) is similar to ours; it used the magnitude of the outgoing weights of a unit as the measure of utility. However, it is not the same as ours because our utility measure also considers the activity of a unit. In continual backpropagation, new units are randomly reinitialized. However, they proposed more complicated methods to generate new units, which included non-linear combinations of already useful units.

Mucciardi and Gose (1966) and Klopf and Gose (1969) built on the work by Selfridge (1958). They showed the effectiveness of representation search methods over methods that used a fixed representation. Note that the representation units in these works differ from those commonly used in modern artificial neural networks. For example, Mucciardi and Gose (1966) used units with binary inputs and a linear threshold activation function. Mucciardi and Gose (1966) used the same utility measure as proposed by Selfridge (1958). Similar to continual backpropagation, Muc-

ciardi and Gose (1966) proposed setting the outgoing weights of new units to zero. Klopf and Gose (1969) built on the work by Mucciardi and Gose (1966) and evaluated various utility measures and found that a utility measure based on the magnitude of the product of a unit's output and its outgoing weight performed the best. Note that this utility measure is exactly the same as the one used in continual backpropagation. Unlike Selfridge (1958) but similar to continual backpropagation, Klopf and Gose (1969) used randomly generated new units. We can see an outline of continual backpropagation emerge from these old papers, as they already contain some of the key ideas of continual backpropagation: utility measure, setting outgoing weights to zero, and random generation of new units.

More recent work by Mahmood and Sutton (Mahmood and Sutton, 2013; Mahmood, 2017) further refined and scaled up the idea of representation search. They developed a new search process and introduced concepts protecting units using a maturity threshold and continually replacing a small fraction of units using a replacement rate. Their work used networks with two linear layers with Linear Threshold Unit (LTU) (McCulloch and Pitts, 1943) in between. Similar to prior work, the representation layer in their network was learned using the search process but not gradient descent. All the prior work, starting from Selfridge's 1958 paper (Seidenberg and McClelland, 1989), laid the foundations for continual backpropagation.

Continual backpropagation is the first combination of the old idea of representation search and modern deep learning. All the prior work on representation search had two significant limitations. First, they did not use modern artificial neural networks, they typically used networks with a single representation layer that did not learn using gradient descent. Second, their networks generally only had a single output, so the utility measure could not be applied to cases where a unit has multiple consumers. We overcome these limitations and make the idea of representation search compatible with modern artificial neural networks. Continual backpropagation works with arbitrary feed-forward networks.

6.2 Different Behaviors of Continual Backpropagation

In this section, we take a deeper look at the hyperparameters of the continual backpropagation algorithm. These hyperparameters induce different behaviors of the algorithm. On one extreme, the algorithm simply find dormant units and reinitializes them. On the other end it acts as a search algorithm in the space of units.

One way to intuitively understand hyperparameters of continual backpropagation is in terms of the number of units eligible for replacement in a given layer called n_e. The evolution of n_e over time gives an insight into the behaviour of the algorithm. Note that only a certain number of units would be eligible for replacement at a given time because the age of other units will be less than the maturity threshold m. For example, at step 0, all units are protected from replacement and n_e is 0. Similarly, at step $m + 1$, all units units become eligible for replacement and n_e is equal to n, where n is the total number of units in the layer. The first replacement happens a few steps after $m + 1$. At that point, one unit is replaced and $n - 1$ units are eligible for replacement.

Figure 6.1 shows the evolution of n_e for a wide range of values of replacement rate, ρ and maturity threshold, m. In all cases, the number of units in the layer is 1000. The figure shows an interesting trend; after some initial oscillation, n_e plateaus at some value. In continual learning, the initial oscillation does not matter as much. The most important thing in the long term is the value at which n_e plateaus.

Finding the stable value of n_e is not difficult. At the stable point of n_e, the size of the pool of ineligible units in the layer stays stable. This means that the rate at which new units are added to the pool must equal the rate at which units leave the pool. New units continually join the pool of ineligible units because their age is zero, and they are protected from replacement. They are added to the pool at a rate of $n_e * \rho$ per step.

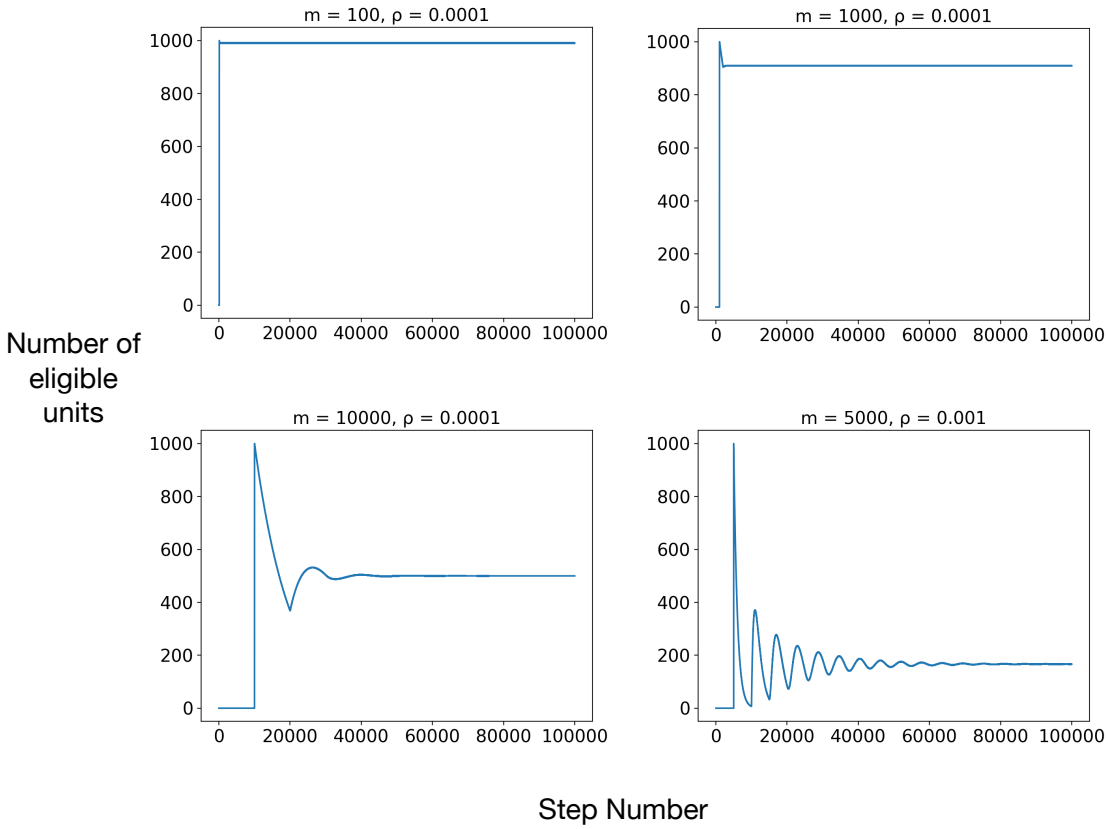

Figure 6.1: The evolution of n_e for a wide range of hyperparameters ρ and m of continual backpropagation. For some settings of hyperparameters n_e oscillates initially. But for all hyperparameters n_e eventually plateaus at some value.

A unit leaves the pool when its age becomes larger than m. Note that once the process reaches the equilibrium, the age of units in the pool is uniformly distributed between 0 and m, and at every step, the age of $\frac{1}{m}$ fraction of the pool becomes larger than m. Because the size of the pool is $n - n_e$, the rate at which units leave the pool is $\frac{n-n_e}{m}$. Equating the rate of new units joining the pool and older units leaving the pool of ineligible units gives us,

$$n_e * \rho = \frac{n - n_e}{m} \tag{6.2}$$

$$m * n_e * \rho = n - n_e$$

$$n_e(1 + m * \rho) = n$$

$$n_e = \frac{n}{1 + m * \rho} \tag{6.3}$$

Equation 6.3 describes the stable value of n_e in terms of hyperparameters ρ and m. When $m * \rho$ is significantly smaller than one, n_e is very close to n, meaning that almost all the units in the layer are eligible at the given step. On the other hand, if $m * \rho$ is close to one, then n_e is much smaller than n, meaning that a significant fraction of the units are protected at that step. The stable value of n_e given by the equation 6.3 also matches the empirical values observed in Figure 6.1. For example, for $m = 100$ and $\rho = 10^{-5}$, n_e plateaus at 999. Similarly, for $m = 5000$ and $\rho = 10^{-3}$, n_e plateaus at 167, which is approximately equal to $1000/(1 + 5000 * 10^{-3})$.

Different values of n_e and m give rise to very different behaviors of continual backpropagation. On the one extreme, n_e can be almost equal to n, as in the top left panel of Figure 6.1. This happens when $m * \rho$ is significantly smaller than one. In such cases, m is generally small and new units do not have enough time to grow their outgoing weights before they become eligible for reinitialization. Meaning they generally have the lowest utility (see Equation 6.1) when the time comes to reinitialize a unit. The only other type of units that can be reinitialized in this case are dormant units because they have small utility due to low activity. In this case, when the

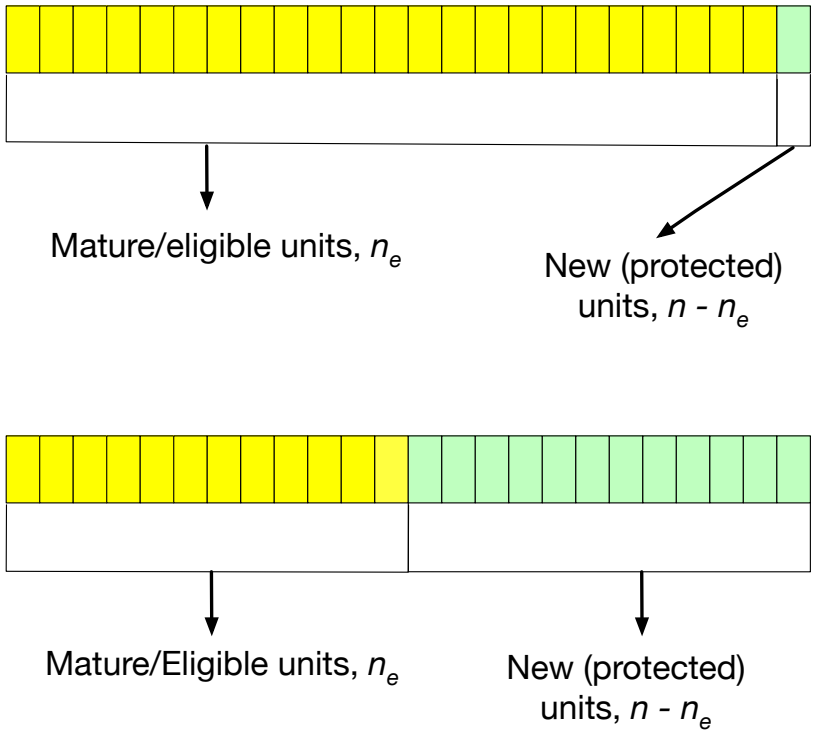

Figure 6.2: A graphical depiction of two behaviours of continual backpropagation. *Top:* $m * \rho$ is small and very few units are protected. In such a case, generally, m is also small, and the algorithm only reinitializes low utility units like dormant ones. *Bottom:* $m * \rho$ equals one, and half of the units are protected from replacement. In such a case, m is generally large, and the algorithm performs an aggressive search process where units with non-negligible utilities can be reintialized.

algorithm has to reinitialize a unit, it is either a dormant unit or the youngest unit. If there are no dormant units, the youngest unit can keep getting reinitialized, and the algorithm has minimal impact on the network.

Another interesting case is when continual backpropagation leads to an aggressive search process. This happens when $m*\rho$ is close to one, for example, $m = 10,000$ and $\rho = 10^{-4}$. In this case, half of the units in the layer are protected, and half are eligible for reinitialization; see Figure 6.2 for a depiction. In such cases, m is typically large and new units have enough time to grow their outgoing weights to have large utilities. In this case, when a unit is reinitialized, it often has non-negligible utility. This means

that the threshold for a unit not to be reinitialized is much higher. This behaviour and setting of hyperparameters of continual backpropagation are particularly powerful in reinforcement learning problems, as we will see in the next chapter.

6.3 Evaluating Continual Backpropagation

In this section, we evaluate continual backpropagation on all the problems where we observed loss of plasticity in Chapter 4. We start with the Online Permuted MNIST problem, where we apply continual backpropagation to a feed-forward network as described in Algorithm 1.

For the Continual ImageNet problem, we apply continual backpropagation to a convolutional network. When using continual backpropagation for a convolutional network, we treat each convolutional filter similar to how we treat a unit in a feed-forward network. The key difference between a unit in a feed-forward network and a convolutional filter is that a unit outputs one number while a filter outputs a matrix. We define the instantaneous utility of a convolution filter as the product of the sum of absolute values in its output matrix and the sum of the absolute values in its outgoing weights. In a convolutional network, the contribution-utility, $\mathbf{u}_l[i]$, of the ith filter in layer l at time t is updated as

$$\mathbf{u}_l[i] \leftarrow \eta * \mathbf{u}_l[i] + (1-\eta) * \sum_{m=1}^{M} \sum_{n=1}^{N} |\mathbf{H}_{l,t}[i,m,n]| * \sum_{k} |\mathbf{W}_{l,t}[i,k]|, \tag{6.4}$$

where $\mathbf{H}_{l,i,t}$ is the output of the i^{th} filter in layer l at time t, M and N are the number of rows and columns of $\mathbf{H}$ respectively, $\mathbf{W}_{l,t}[i,k]$ is the k^{th} outgoing weight of the filter. Note that this equation is equivalent to Equation 6.1, but for convolutional filters.

Finally, we apply continual backpropagation to a residual network in CIFAR 100. The residual consists of convolutional filters and skip connections that cross over multiple layers in the network. To apply continual backpropagation to the residual

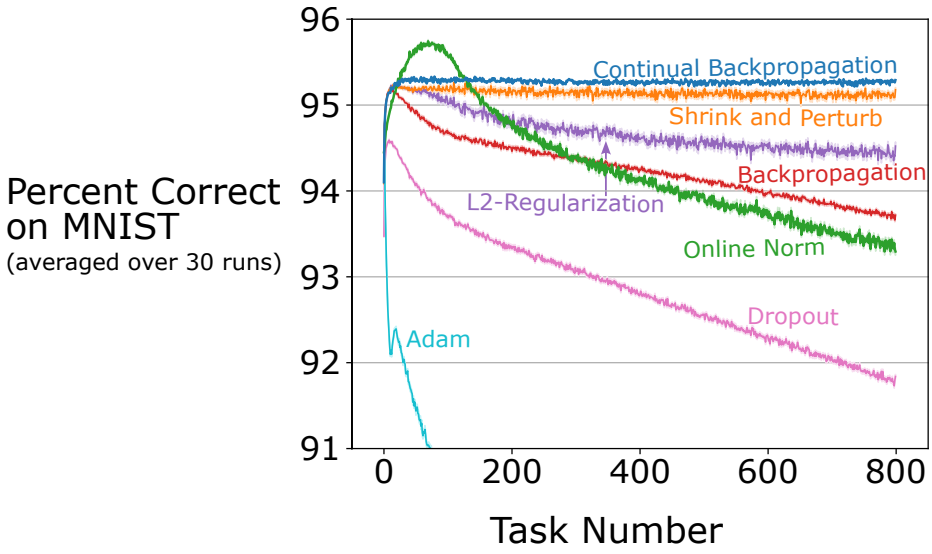

Figure 6.3: The online classification accuracy of various algorithms on Online Permuted MNIST. The performance of all algorithms except continual backpropagation degrades over time.

network, we treat convolutional filters as discussed above and ignore skip connections because they do not have a learnable weight but rather a constant weight of one.

In Online Permuted MNIST, we used the same network as in Section 4.3. This network had 3 hidden layers with 2000 units each. We used SGD with a step-size of 0.003 to train this network, as it was the best performing step-size with backpropagation in Figure 4.2. For continual backpropagation, we show the online classification accuracy for various values of replacement rates and a fixed maturity threshold of 100. As we saw in the previous section, replacement rate controls how rapidly units are reinitialized in the network. For example, a replacement rate of 10^{-6} and maturity threshold of 100 for our network with 2000 hidden units in each layer would mean replacing one unit in each layer after every 501 examples. The online classification accuracy of continual backpropagation on Online Permuted MNIST is presented in Figure 6.3. Among all the algorithms, only continual backpropagation and Shrink and Perturb have non-degrading performance. Additionally, continual backpropagation is stable for a wide range of replacement rates as shown in Figure 6.4. Both continual backpropagation and Shrink and Perturb enable small weight magnitudes

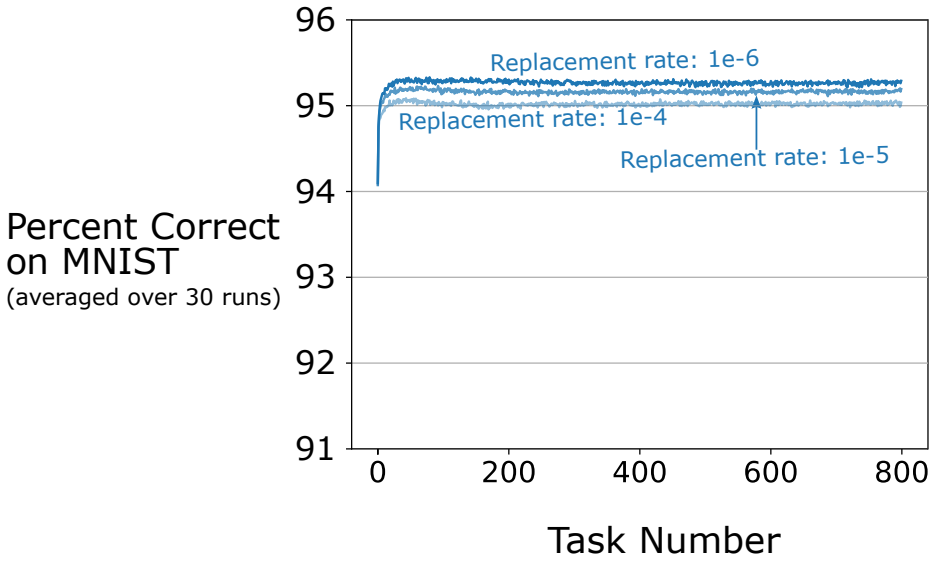

Figure 6.4: The performance of continual backpropagation for a wide range of replacement rates on Online Permuted MNIST. Continual backpropagation maintains a good level of performance for a wide range of replacement rates.

and diversity of representation by their design.

Let us take a deeper look at the network that is learning via continual backpropagation. The evolution of the correlates of loss of plasticity is shown in Figure 6.5. Continual backpropagation mitigates all three correlates of loss of plasticity. It has almost no dead units, stops the network weights from growing, and maintains a high effective rank across tasks. All algorithms that stop the weights from growing reduced loss of plasticity. This supports our claim that low weight magnitudes are important for maintaining plasticity. The algorithms that maintain low weight magnitudes were continual backpropagation, L2-regularization, and Shrink and Perturb. Shrink and Perturb and continual backpropagation have an additional advantage over L2-regularization: they inject randomness into the network. This injection of randomness leads to a higher effective rank and lower number of dead units, which leads to these algorithms outperforming L2-regularization. However, continual backpropagation injects randomness selectively, effectively removing all dead units from the network and leading to a higher effective rank. The smaller number of dead units and higher effective rank explains the better performance of continual backpropagation.

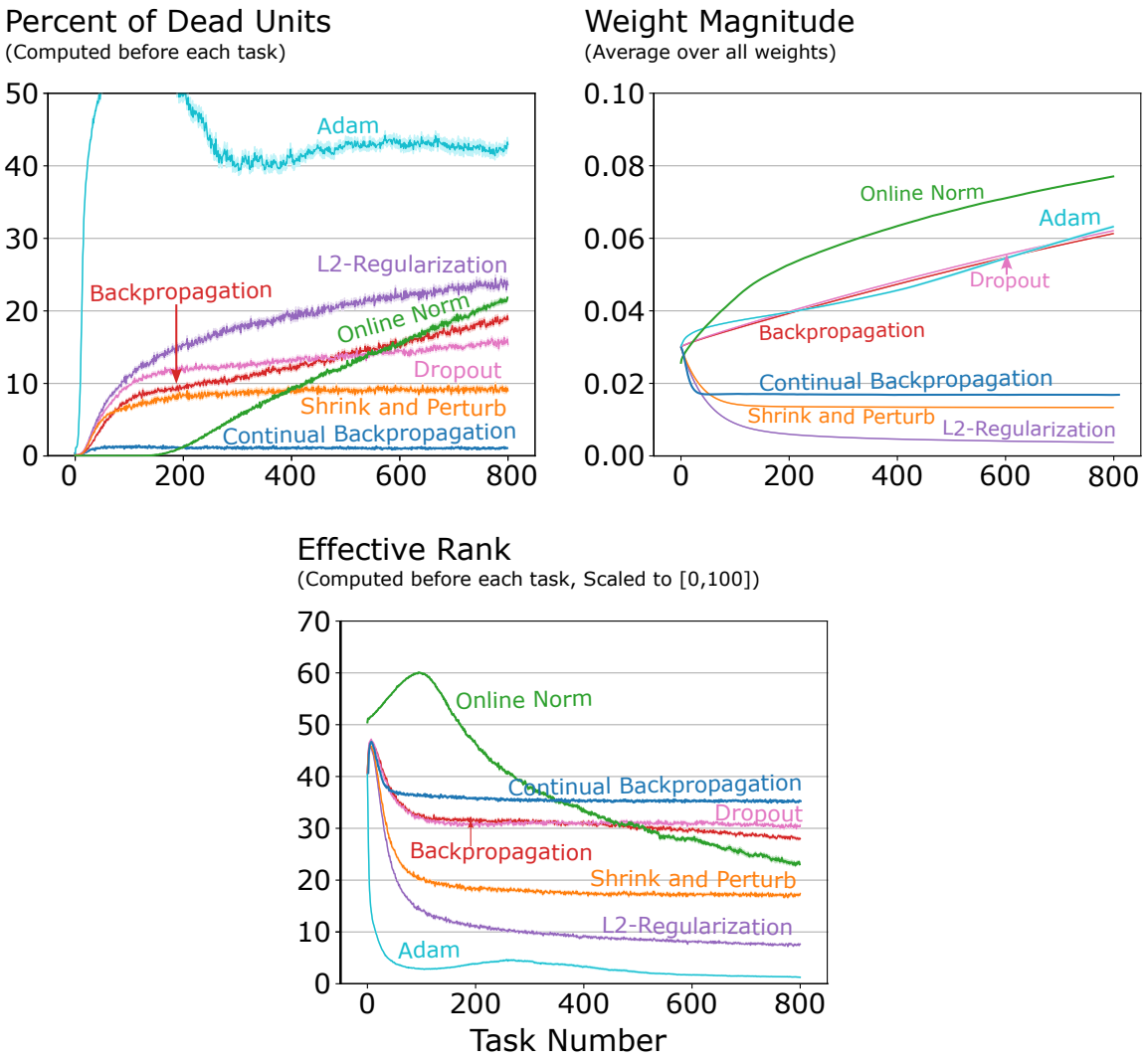

Figure 6.5: A deeper look into various qualities of a deep network on Online Permuted MNIST using different algorithms. *Top Left:* Over time, the percentage of dead units increases in all methods except for continual backpropagation. It has almost zero dead units throughout learning, and this happens because dead units have zero utility so they are quickly reinitialized. *Top Right:* The average magnitude of the weights increases over time for all methods except for L^2-Regularization, Shrink and Perturb, and continual backpropagation. And, these are the three best-performing methods. This means that non-increasing weights are important for maintaining plasticity. *Bottom:* The effective rank of the representation of all methods drops over time. However, continual backpropagation maintains a higher effective rank than both backpropagation and Shrink and Perturb. Among all the algorithms only continual backpropagation maintains a high effective rank, non-increasing weight magnitude, and low percent of dead units.

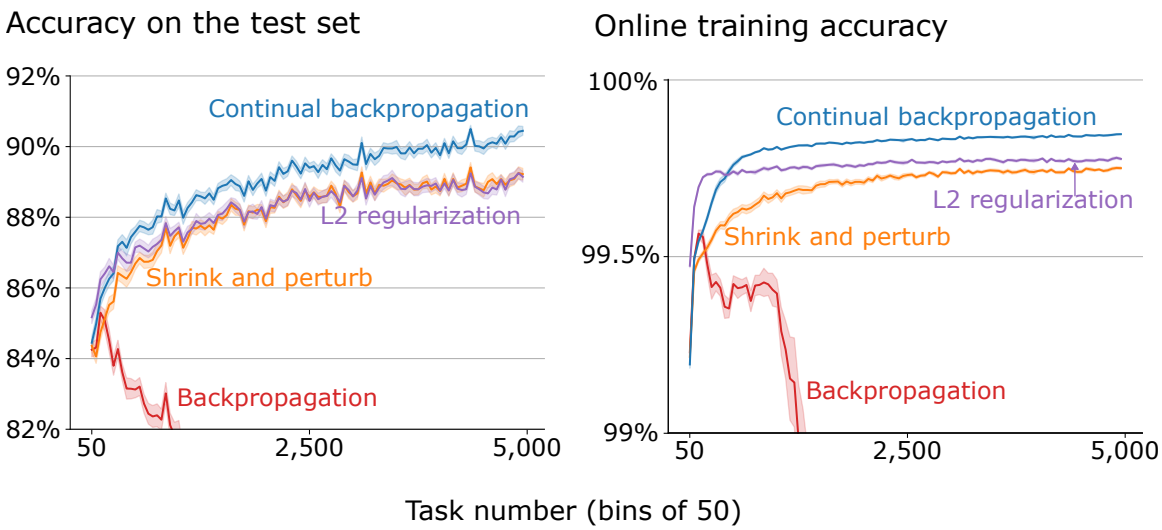

Figure 6.6: Continual backpropagation outperforms many commonly used algorithms and fully maintains plasticity on Continual ImageNet. It performs well on the test set as well as the training data. Its performance at the end of 5000 tasks is even better than on the first task.

Then we tested continual backpropagation on Continual ImageNet. We also tried L2 regularization, and Shrink and Perturb on Continual ImageNet, as these are the only two methods that reduced loss of plasticity in Permuted MNIST. For all algorithms, we present the performance of the hyperparameter value that had the largest average classification accuracy on the test set over 5000 tasks. The classification accuracy on the test set of various algorithms on Continual ImageNet is shown in the left panel of Figure 6.6. And the online training accuracy is shown in the right panel of Figure 6.6. The results are averaged over thirty runs.

The first point in Figure 6.6 is the average accuracy on the first 50 tasks; the next is the average accuracy over the next 50 tasks, and so on. For continual backpropagation, we used a maturity threshold of 100 and a replacement rate of $3 * 10^{-4}$. The details of hyperparameters for all other algorithms and their selection procedure are described in Table B.1 in Appendix B.

Continual backpropagation fully mitigates the loss of plasticity in Continual ImageNet. Its classification accuracy on the test set of the 5000th task is better than on

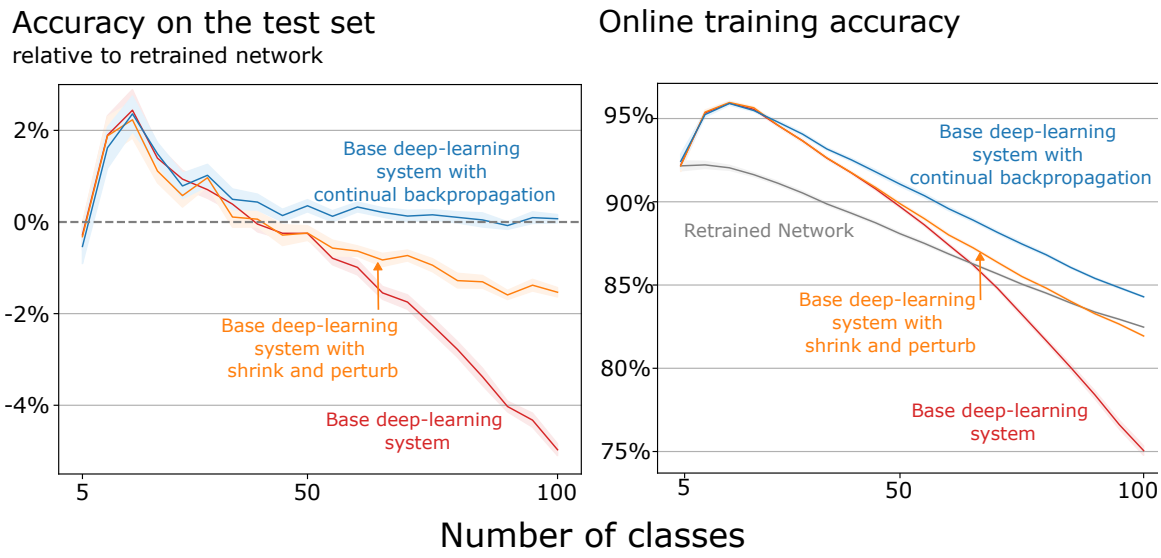

Figure 6.7: Continual backpropagation fully maintains plasticity on class-incremental CIFAR-100. *Left:* Its accuracy on the test set at the end of each increment is always better than or equal to that of a network trained from scratch. *Right:* Its online training accuracy is better than the network trained from scratch. Online training accuracy captures the speed of learning in addition to the final accuracy. The higher online training accuracy of continual backpropagation means that continual backpropagation learns faster than the network trained from scratch and has about 2% higher online training accuracy. All results are averaged over 30 runs and the shaded region represents plus and minus one standard error.

the first. It also outperforms existing techniques like L2-regularization and Shrink and Perturb. Additionally, by the 5000th task it also has the hightest average accuracy during training.

Finally, we applied continual backpropagation with a residual network in class-incremental CIFAR-100. Figure 6.7 shows the performance of various algorithms on class-incremental CIFAR-100. Among all the algorithms, only continual backpropagation maintains plasticity, while all other algorithms lose plasticity (left panel of Figure 6.7). The right panel of Figure 6.7 shows the online training accuracy of all algorithms. Continual backpropagation has about 2% higher online training accuracy than the retrained network. Online training accuracy captures the speed of learning of an algorithm. Continual backpropagation learns faster than the network trained

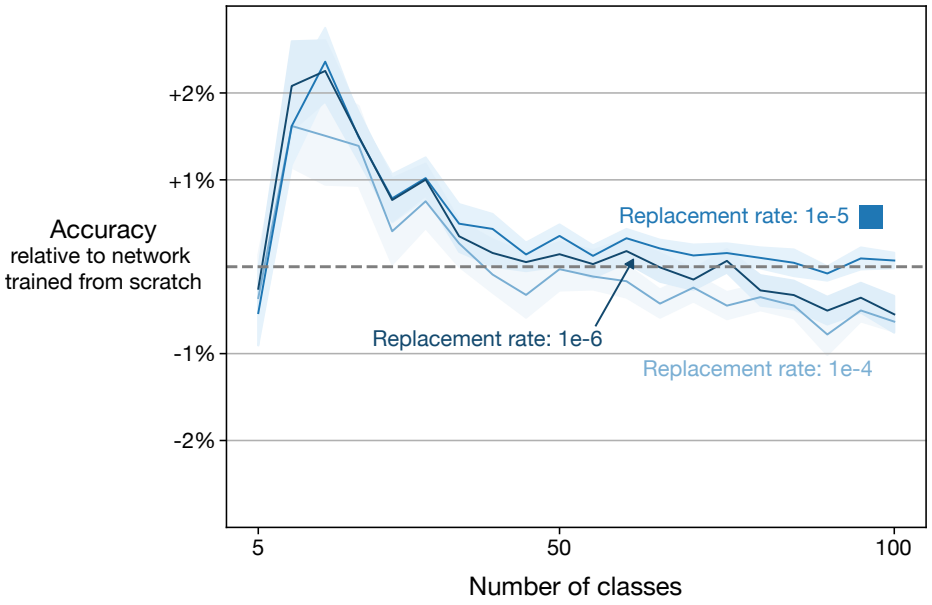

Figure 6.8: Continual backpropagation performs well for a wide range of replacement rates on class-incremental CIFAR-100. All results are averaged over 30 runs and the shaded region represents plus and minus one standard error.

from scratch, even though they both have the same final performance. In Figure 6.8, we show the performance of continual backpropagation for various values of replacement rate. It performs well for all values of replacement rate. However, only for a replacement rate of 10^{-5}, it performed better than the network trained from scratch. We found that maturity threshold of 1000 performed best in this problem.

We took a deeper look at the network trained by continual backpropagation in Figure 6.9. The blue lines in the Figure show the evolution of percentage of dormant units and the stable rank of the representation of the network trained by continual backpropagation. In continual backpropagation has almost no dead units and it maintains a high stable rank throughout learning.

In this section, we evaluated continual backpropagation and found that it fully maintains plasticity in Continual ImageNet, Online Permuted MNIST, and class-incremental CIFAR-100. Continual backpropagation outperforms all the existing methods on all three continual learning problems. It is much less sensitive to its

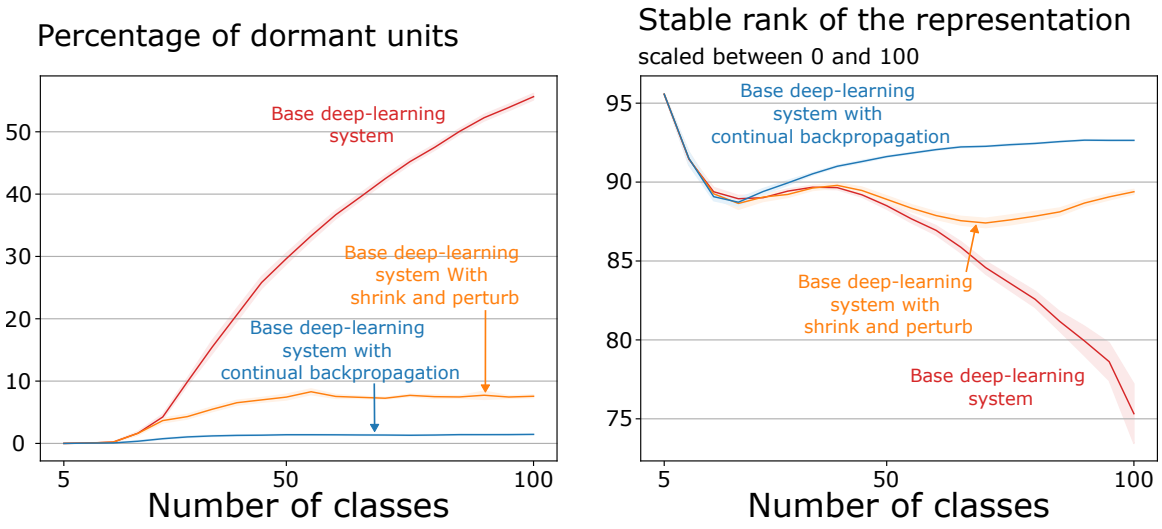

Figure 6.9: Continual backpropagation fully maintains plasticity on class-incremental CIFAR-100. Additionally, it has almost no dead units and it maintain a high stable rank.

hyperparameters than other algorithms like L2-regularization and Shrink and Perturb. It also mitigates all three correlates of plasticity as it maintains a low average weight magnitude, a very small percentage of dead units, and a high effective rank. The results in this section are consistent with the idea that non-increasing weights are important for maintaining plasticity and that a continual injection of variability further mitigates loss of plasticity. Although Shrink and Perturb adds variability to all weights, continual backpropagation does so selectively, which seems to make it better at maintain plasticity.

6.4 Discussion of Other Plasticity Preserving Algorithms

The key idea behind continual backpropagation is to maintain plasticity by selectively reinitializing units in the network. In the last few years, some work has built on this idea and developed new algorithms for finding low utility units (Sokar et al., 2023; Farias and Jozefiak, 2025; Liu et al., 2025). Sokar et al. (2023) proposed an algorithm

that reinitialize units with low activity. In their case, a unit is considered to have low activity if its average activation is smaller than some small fraction of the average activation in the layer. They showed that their proposed algorithm can significantly improve the final performance and sample efficiency of deep reinforcement learning algorithms. Recent work by Liu et al. (2025) shows that low activation might not be a good indication of the low utility of a unit in complex network architectures that involve normalization layers and non-ReLU activations. Instead, they show that a small incoming gradient is a better indicator of low utility and that reinitializing units with relatively small incoming gradients effectively maintains plasticity with complex network architectures.

Farias and Jozefiak (2025) developed a new idea for selecting units for reinitialization. In continual backpropagation, a unit is compared to other units in the layer to decide if it should be reinitialized. However, the method developed by Farias and Jozefiak (2025) compares units to their past selves. The key idea behind their algorithm is to increase the resetting probability of a unit if its activity drops over time. Their experiments show that their algorithm can be more effective at maintaining plasticity in continual supervised learning problems than continual backpropagation and other methods designed for maintaining plasticity. Additionally, they theoretically analyze a simple case where the goal is to learn a target ReLU. They show that their reinitializing algorithm can learn continually learn a sequence of target ReLUs. However, even with L2 regularization, gradient descent can sometimes fail. However, the algorithm they used requires keeping track of a large window of past feature activations to maintain the distribution of inter-firing times. This results in a memory complexity of $\mathrm{O}(w * l)$ for their algorithm, where w represents the width of the network and l denotes the length of the window. In supervised learning experiments, they used window lengths of a few thousand, making the memory requirement of the window much larger than that of the network. There is an approximate version of the algorithm with memory complexity $\mathrm{O}(w)$. In the next chapter, we compare

this approximate version with continual backpropagation in a reinforcement learning environment.

One natural variation of the idea of selectively reinitializing units is to be more granular and selectively reinitialize weights. Extending the idea of reinitializing units to attention layers in transformer models is non-trivial, which means that continual backpropagation can not be applied to the full transformer models. Hernandez-Garcia et al. (2025) developed a new algorithm for reinitializing weights for maintaining plasticity. Their algorithm performs about as well as continual backpropagation with residual networks and successfully maintains plasticity with transformer models. Concurrently, Hofmann et al. (2025) showed that selective weight reinitialization could be a useful addition even in stationary settings.

Continual backpropagation continually injects randomness into the network. The idea of continually injecting randomness in present in deep learning in the form of perturb gradient descent (Jin et al., 2017). This algorithm adds random noise to the weights at each step. Shrink and Perturb (Ash and Adams, 2020) extends the idea of perturb gradient descent to continual learning problems. The key difference between these algorithms and continual backpropagation is that continual backpropagation injects randomness selectively, and the magnitude of randomness is larger. As we have seen in this thesis, Shrink and Perturb can effectively maintain plasticity. Galashov et al. (2024) developed a new version of the Shrink and Perturb algorithm where the amount of randomness added to each weight in the network is controlled by a variable which is continually learned from the data stream. Their results show that their method can outperform Shrink and Perturb as well as many other plasticity-preserving algorithms. This idea could be extended to continual backpropagation to adapt its hyperparameters to the changes in the data stream.

In settings where the learning system has a large memory and can store old data, one of the most straightforward ways to maintain plasticity is to reset the entire network or large parts of it. One such setting is deep reinforcement learning, where large

replay buffers allow the agent to store a large part of its old experience. In reinforcement learning, the idea of periodically resetting the network has proven to be very powerful in the last few years. Many papers have shown that resetting the network can dramatically improve the sample efficiency and the final performance of reinforcement learning agents (Nikishin et al., 2022, 2023; D'Oro et al., 2023; Schwarzer et al., 2023). However, these methods are limited to cases where large replay buffers are available and are ineffective in cases where the learning system can not store a large amount of old data.

Another class of methods that have been proposed to maintain plasticity involve architectural changes. Abbas et al. (2023) showed that the concatenated ReLU activation can be very effective at maintaining plasticity in continual deep reinforcement learning as the concatenated ReLU activation gets rid of dormant neurons by design. A line of work by Lyle et al. (2023; 2024b; 2024a) shows that layer normalization combined with well-tuned L2 regularization effectively maintains plasticity in a wide range of continual learning problems. This is because L2 regularization ensures that weights do not grow over time, and layer normalization can generally ensure no dormant units in the network.

The last class of methods developed for maintaining plasticity involve different types of regularization methods. Kumar et al. (2024) proposed the idea of regularizing toward the initial values of the weights to ensure that properties of initialization are maintained throughout learning. They found that this method can be effective in many continual supervised learning problems. Lewandowski et al. (2025a) developed spectral regularization, which forces the maximum singular value of each layer to stay close to one. This ensures a diverse flow of gradients throughout the network. They show that spectral regularization effectively maintains plasticity and generally outperforms regularizing toward the initial values. Most recently, Chung et al. (2024) proposed a regularization method that forces the weight of different units in the network to remain orthogonal. This orthogonality ensures that the units remain diverse

throughout learning. They found that this regularization method outperforms many other normalization and regularization algorithms in continual reinforcement learning, including those based on layer normalization, Shrink and Perturb, and regularization towards initialization. Their results suggest that diversity of representation is useful for maintaining plasticity.

6.5 Discussing Related Ideas in Machine Learning

Previous works on the importance of initialization have focused on finding the correct weight magnitude to initialize the weights. It has been shown that it is essential to initialize the weights so that the gradients do not become exponentially small in the initial layers of a network and the gradient is preserved across layers (He et al., 2015; Glorot and Bengio, 2010). Furthermore, initialization with small weights is critical for sigmoid activations as they may saturate if the weights are too large (Sutskever et al., 2013). Despite all this work on the importance of initialization, the fact that its benefits are only present initially but not continually has been overlooked, as these papers focused on cases in which learning has to be done just once, not continually.

Continual backpropagation uses a utility measure to find and replace low-utility units. One limitation of continual backpropagation is that the utility measure is based on heuristics. Although it performs well, future work on more principled utility measures will improve the foundations of continual backpropagation. Our current utility measure is not a global measure of utility as it does not consider how a given unit affects the overall represented function. One possibility is to develop utility measures in which utility is propagated backwards from the loss function. The idea of utility in continual backpropagation is closely related to connection utility in the neural-network-pruning literature. Various papers (LeCun et al., 1989; Han et al., 2016; Gale et al., 2019; Liu et al., 2020) have proposed different measures of connection utility for the network-pruning problem. Adapting these utility measures to mitigate loss of plasticity is a promising direction for new algorithms, and some recent work

by Elsayed and Mahmood (2024) has already made progress in this direction.

The idea of adding new units to neural networks is present in the continual-learning literature (Yoon et al., 2018; Zhou et al., 2012; Rusu et al., 2016). This idea is usually manifested in algorithms that dynamically increase the size of the network. For example, one method (Rusu et al., 2016) expands the network by allocating a new sub-network whenever there is a new task. These methods do not have an upper limit on memory requirements. Although these methods are related to the ideas in continual backpropagation, none are suitable for comparison, as continual backpropagation is designed for learning systems with finite memory, which are well suited for lifelong learning. And these methods would therefore require non-trivial modification to apply to our setting of finite memory.

The idea of selective reinitialization is similar to the emerging idea of dynamic sparse training (Mocanu et al., 2018; Bellec et al., 2018; Evci et al., 2020). In dynamic sparse training, a sparse network is trained from scratch and connections between different units are generated and removed during training. Removing connections requires a measure of utility, and the initialization of new connections requires a generator similar to selective reinitialization. The main difference between dynamic sparse training and continual backpropagation is that dynamic sparse training operates on connections between units, whereas continual backpropagation operates on units. Consequently, the generator in dynamic sparse training must also decide which new connections to grow. Dynamic sparse training has achieved promising results in supervised and reinforcement-learning problems (Chen et al., 2021; Sokar et al., 2022; Graesser et al., 2022), in which dynamic sparse net- works achieve performance close to dense networks even at high sparsity levels. Dynamic sparse training is a promising idea that can be useful to maintain plasticity.

One common strategy to deal with non-stationary data streams is reinitializing the network entirely. In the Online Permuted MNIST experiment, full reinitialization corresponds to a performance that stays at the level of the first point Figure 6.3.

In this case, continual backpropagation outperforms full reinitialization as it takes advantage of what it has previously learned to speed up learning on new data. In ImageNet experiments, the final performance of continual back- propagation is only slightly better than a fully reinitialized network (the first point for backpropagation in the left panel of Figure 6.6). However, Figure 6.6 does not show how fast an algorithm reaches the final performance in each task. We observed that continual backpropagation achieves the best accuracy ten times faster than a fully reinitialized network on the 5,000th task of Continual ImageNet, ten epochs versus about 125 epochs. Furthermore, continual backpropagation could be combined with other methods that mitigate forgetting, which can further speed up learning on new data. Recent work by Verwimp et al. (2025) shows that the plasticity-preserving method Shrink and Perturb significantly speeds up learning in later tasks. They show that in some image classification problems, incrementally learning via Shrink and Perturb can learn twice as fast as learning from scratch, leading to massive computational savings.

Some recent works have focused on quickly adapting to the changes in the data stream (Finn et al., 2017; Wang et al., 2017; Nagabandi et al., 2019). However, the problem settings in these papers were offline as they had two separate phases, one for learning and the other for evaluation. To use these methods online, they have to be pretrained on tasks that represent tasks that the learner will encounter during the online evaluation phase. This requirement of having access to representative tasks in the pretraining phase is not realistic for lifelong learning systems as the real world is non-stationary, and even the distribution of tasks can change over time. These methods are not comparable with those we studied in our work, as we studied fully online methods that do not require pretraining.

There are two main goals in continual learning: maintaining stability and maintaining plasticity (Caruana, 1997; Ring, 1998; Parisi et al., 2019; Kumar et al., 2025). Maintaining stability is concerned with memorizing useful information and main-

taining plasticity is about finding new useful information when the data distribution changes. Current deep-learning methods struggle to maintain stability as they tend to forget previously learned information (McCloskey and Cohen, 1989; French, 1999). Many papers have been dedicated to maintaining stability in deep continual learning (Kirkpatrick et al., 2017; Yoon et al., 2018; Aljundi et al., 2019; Golkar et al., 2019; Riemer et al., 2019; Rajasegaran et al., 2019; Javed and White, 2019). We focused on continually finding useful information, not on remembering useful information. Our work on loss of plasticity is different but complementary to the work on maintaining stability. Continual backpropagation in its current form does not tackle the forgetting problem. Its current utility measure only considers the importance of units for current data. One idea to tackle forgetting is to use a long-term measure of utility that remembers which units were useful in the past. Developing methods that maintain both stability and plasticity is an important direction for future work.

When continual backpropagation reinitializes a unit, its weights are randomly sampled from a distribution. However, new weights can also be selected in other ways. Prior work on representation search contains algorithms where new units are initialized in other ways (Holland and Reitman, 1977; Holland, 1992; Kaelbling, 1993; Stanley and Miikkulainen, 2002; Whiteson and Stone, 2006). One idea is to initialize new units that take pre-existing useful units as inputs and create a non-linear combination of them (Kaelbling, 1993). Another idea is to create new units as variants of original units (Holland and Reitman, 1977; Stanley and Miikkulainen, 2002). In our context, one limitation of these works is that they do not use artificial neural networks, but rather some other form of representation units. Recent work has already shown promising results for generating new features that use existing useful features as inputs in artificial neural networks (Javed, 2025). Further exploring these ideas for smarter generation of new units with continual backpropagation is a promising direction for future work.

Continual backpropagation injects randomness into the network and performs a

type of search process in the space of representation units. The idea of searching through a trial-and-error process is widespread in sciences. Perhaps the most famous version of this idea is the field of natural evolution, where evolution is best thought of as selective survival of off-springs who are random variants of their parent(s) (Darwin, 1859). This idea is also present in fields of psychology and behavior (Thorndike, 1911; Campbell, 1960; Dennett, 1975). Solution methods for stochastic approximation and optimization also use variations of this ides (Kashyap et al., 1970; Powell, 1977). Continual backpropagation is a version of this idea for artificial neural networks and it opens many research directions to further refine this idea.

6.6 Discussing Connections to Neuroscience

Perhaps the most exciting connections to the ideas of continual backpropagation are not in the machine learning literature but in neuroscience. Similar to continual backpropagation, new biological neurons are born in adult brains, a phenomenon called neurogenesis (Eriksson et al., 1998). New neurons created during neurogenesis are highly plastic, and they are used to quickly learn new things (Altman, 1963; Eriksson et al., 1998). A common view is that new neurons are used to learn new things, while preserving old neurons and knowledge. This provides a way to balance plasticity with stability (Aimone et al., 2010). In contrast, continual backpropagation, in its current form, does not provide a way to preserve old knowledge. However, it is possible that new methods can be developed that also preserve old artificial neurons and knowledge based on some utility metric to balance plasticity with stability.

Power law remembering (Fusi et al., 2005; Benna and Fusi, 2016) in neuroscience provides interesting directions to maintain both plasticity and stability in artificial neural networks. The cascade model by Fusi et al. (2005) proposes that each synapse has multiple weights, instead of a single weight. There are shallow weights that are highly adaptive to new information, and there are deeper weights that change much more slowly and are robust to catastrophic forgetting. Some ideas from continual

backpropagation can be used along with the cascade model to maintain both plasticity and stability in continual learning. Low-utility shallow weights can be reset using a continual backpropagation type method to inject plasticity into the network, while the cascading of information through different weights can slowly accumulate knowledge in deeper weights.

Continual backpropagation and Shrink and Perturb perform operations that are largely independent of the input. There is evidence that synapses, which are similar to weights in artificial neural networks, in some parts of the brain change independently of any signal (Ziv and Brenner, 2018). A large fraction of the synapses reset every day. Some estimates suggest resetting can be as significant as 1-3% per day (Kasai et al., 2021). Some of the synapses change through a process that is surprisingly similar to the Shrink and Perturb algorithm (Kasai et al., 2021). The purpose of resetting and Shrink and Perturb in the brain is unclear. There are ongoing efforts to test if these processes help with plasticity. If it is found that resetting and Shrink and Perturb in the brain help with maintaining plasticity, it would be a great example of the synergy between the fields of neuroscience and artificial intelligence. And perhaps it would mean that continually doing similar computations is a fundamental principle underlying intelligence.

6.7 Conclusion

In this chapter, we presented and evaluated the continual backpropagation algorithm. Continual backpropagation combines gradient descent with selective reinitialization. The results show that continual backpropagation, and to some extent Shrink and Perturb, enable artificial neural networks to maintain plasticity. Continual backpropagation and Shrink and Perturb add a source of randomness into the network, which was absent in standard deep learning. This randomness mitigates the three correlates of loss of plasticity that we found in Chapter 4 and maintains plasticity in all problems. Furthermore, we saw that continual backpropagation learns faster

than the retrained network on class-incremental CIFAR-100. We saw two different behaviors of continual backpropagation, where, depending on its hyperparameters, it acts as an algorithm that reinitializes dormant units or performs an aggressive search process. The selective reinitialization of continual backpropagation can be thought of as a search process in the space of representation units. Continual backpropagation builds on a long line of work on representation search in machine learning, bringing this idea to modern deep learning (Selfridge, 1958; Mucciardi and Gose, 1966; Klopf and Gose, 1969; Mahmood and Sutton, 2013). Continual backpropagation is the first version of the idea of representation search in modern deep learning. Further development of this idea can improve the effectiveness of modern deep learning in continual learning problems.

Chapter 7

Plasticity Loss in on-policy Deep Reinforcement Learning

This thesis has focused on the loss of plasticity in continual supervised learning problems because it is the simplest setting where we can establish and study the phenomenon of loss of plasticity without encountering other confounders. However, continual learning is perhaps more natural for reinforcement learning. In many reinforcement learning problems, it is natural to expect the environment to change over time. Additionally, the agent's behaviour changes over time as it learns to get more reward. This makes the agent's data stream non-stationary even if the environment does not change. Furthermore, modern reinforcement learning generally uses temporal difference learning methods where the target for learning is also a learned quantity, which further necessitates continual learning. For these reasons, it is important to study the phenomenon of loss of plasticity in reinforcement learning.

In this chapter, we study the phenomenon of policy collapse and its connection with loss of plasticity. Policy collapse refers to the phenomenon where the learned policy can dramatically worsen after some initial training as the agent continues to interact with the environment. We focus on on-policy reinforcement learning algorithms as they typically use the least amount of additional memory which makes them most susceptible to loss of plasticity. Particularly, we study the PPO algorithm (Schulman et al., 2017). In this chapter, we first demonstrate policy collapse in a

Mujoco environment, and then we take a deep dive into the phenomenon in a 2-state MDP. Finally, we study algorithms to overcome the policy collapse and explore its connection to loss of plasticity.

7.1 Policy Collapse

AI systems that take advantage of available data and computation tend to outperform systems that do not (Sutton, 2019). Historically, in games like Chess and Go, systems that utilize the available data and computation have defeated all other systems (Campbell et al., 2002; Silver et al., 2016). Most recently, large language models like GPT-4 (OpenAI, 2023) have dramatically outperformed previous natural language processing systems, primarily due to the amount of data and computation used by them. The need to improve performance with data is particularly relevant for reinforcement learning systems (Sutton and Barto, 2018) as they experience a potentially unending data stream.

Unfortunately, the performance of many existing deep reinforcement learning algorithms does not always improve with more experience. The policy learned by these algorithms can dramatically worsen as the agent continues interacting with the environment, a phenomenon we call *policy collapse*. The evidence of policy collapse is scattered throughout the reinforcement learning literature. For instance, policy collapse can be observed in several reinforcement learning algorithms such as DQN, PPO, and DDPG, as shown in papers by Schaul et al. (2016, Figure 7), Henderson et al. (2018, Figure 2), and Tassa et al. (2018, Figure 4), respectively.

Although we can observe policy collapse in several papers, it has not been pointed out and studied in the literature. As the first step, we now establish that policy collapse can occur in the widely used Proximal Policy Optimization (PPO) algorithm. Its variants have been used in many applications ranging from robotics (OpenAI et al., 2019) to post-training of large language models to improve their reasoning capabilities (DeepSeek-AI, 2025). Additionally, PPO is computationally cheap, allowing us to

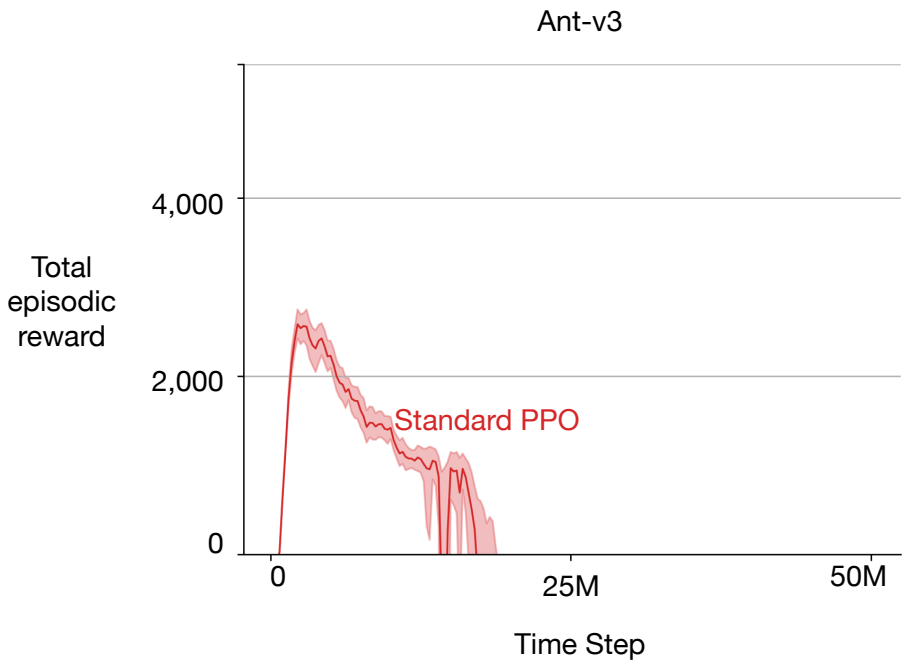

Figure 7.1: PPO on Ant-v3. After initial learning, the policy learned by PPO kept degrading, and its performance dropped below what it was in the beginning. PPO did not scale with experience because instead of improving, its performance decreased with more experience. These results are averaged over 30 runs. The solid lines represent the mean, and the shaded regions correspond to a 95% bootstrapped confidence interval.

perform thorough and long experiments.

In the first experiment, we test if PPO scales with experience on a standard Mujoco environment, *Ant-v3*. Usually, PPO is only trained for 1-3 million time steps on Mujoco environments. However, we ran PPO with standard setting of its hyperparameters for 50M time steps. Two separate networks were used for the policy and the value function, and both had two hidden layers with 256 units. We used the undiscounted episodic return as the measure of performance. The results of the experiments are shown in Figure 7.1.

The x-axis in the plots is the time step, and the y-axis is the undiscounted episodic return in bins of 100k times steps. The first points in the plots are the average return for the episodes in the first 100k time steps, and the next point is the average return

for the episodes in the next 100k time steps and so on. We performed 30 independent runs, and the shaded region shows the 95% bootstrapped confidence interval. In all experiments in this chapter, we report the 95% bootstrapped confidence interval as recommended by Patterson et al. (2024) for reinforcement learning problems.

The performance of PPO improved for the first few million time steps, then it hit a plateau, and finally, it dropped to a level below what it had in the beginning. Another thing to note is that once the performance dropped, it did not improve, suggesting that the agent might have lost plasticity, which is the ability to learn new things. These results mean that PPO does not scale with experience as its performance degrades instead of improving. It point out a major problem with PPO: it is not stable during training, and its scalability is limited due to policy collapse.

7.2 A Deeper Look at Policy Collapse in a 2-state MDP

To overcome policy collapse, we first need to understand what happens to the learning agent when the policy collapses. However, fully understanding policy collapse in modern deep reinforcement learning algorithms in standard environments is difficult because deep reinforcement learning algorithms have many interacting parts, such as bootstrapping, off-policy learning, function approximation, exploration, and changing policy. Additionally, modern deep reinforcement learning algorithms have dozens of hyper-parameters, and a wrong setting of any one of them could be causing policy collapse. To make matters worse, the environments where deep reinforcement learning algorithms are tested are extremely complicated and computationally expensive, which makes it impossible to do a full grid search over the hyperparameter space.

In this section we explore policy collapse in a simple MDP. The MDP consists of two states, the agent can take two actions, left and right, in both states. Both actions take the agent to the terminal state. However, the reward associated with each action is different. The MDP is shown in Figure 7.2A. After termination, the agent starts

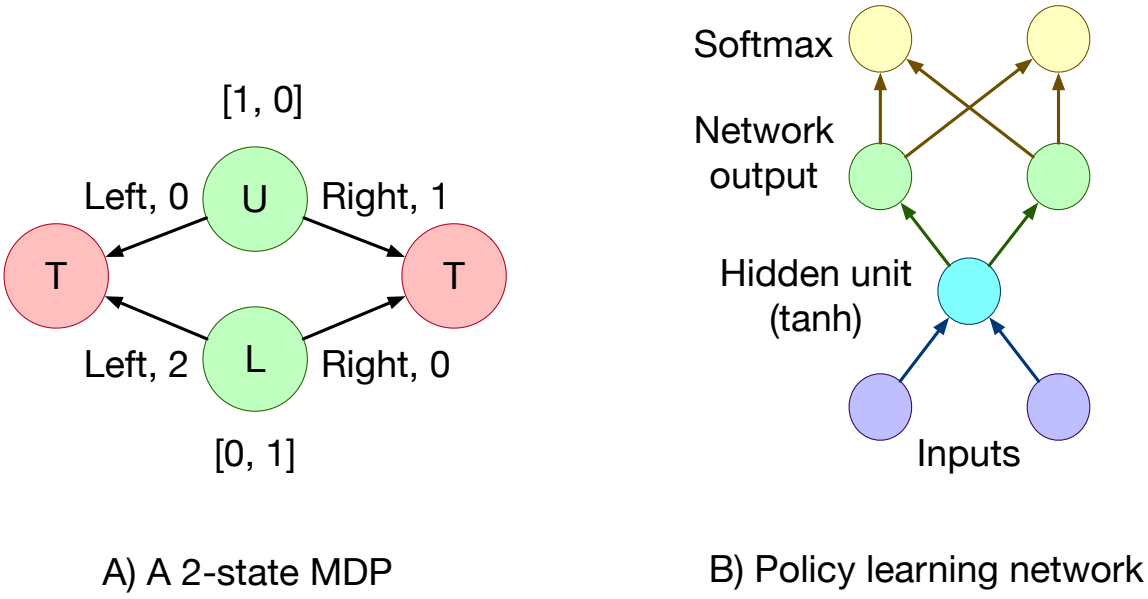

Figure 7.2: *Left:* A 2-state MDP. *Right:* The network used by the learning agent to represent the policy.

with equal probability in both states. There are four deterministic policies for this MDP, and the best one is to choose the right action in state U and left action in state L. The expected return for optimal policy is 1.5, while for other deterministic policies, it is 1.0, 0.5, and 0.0.

We trained an agent using PPO on this MDP. The agent used a neural network to learn a policy. The neural network had one hidden layer with one unit and tanh activation. The input to the network is a two-dimensional vector, which is $[1, 0]$ when the agent is in state U and $[0, 1]$ when the agent is in state L. The action probabilities are obtained by passing the network outputs into a softmax operator. Figure 7.2B shows the policy learning network. The agent used a different network with one hidden layer and one hidden unit to learn a value function for the current policy.

We use the expected return as the performance measure in this experiment. Because we have access to the entire state space, we can calculate the exact expected return instead of approximating it using the actual return. We performed 500 independent runs for this experiment. The results are shown in Figure 7.3.

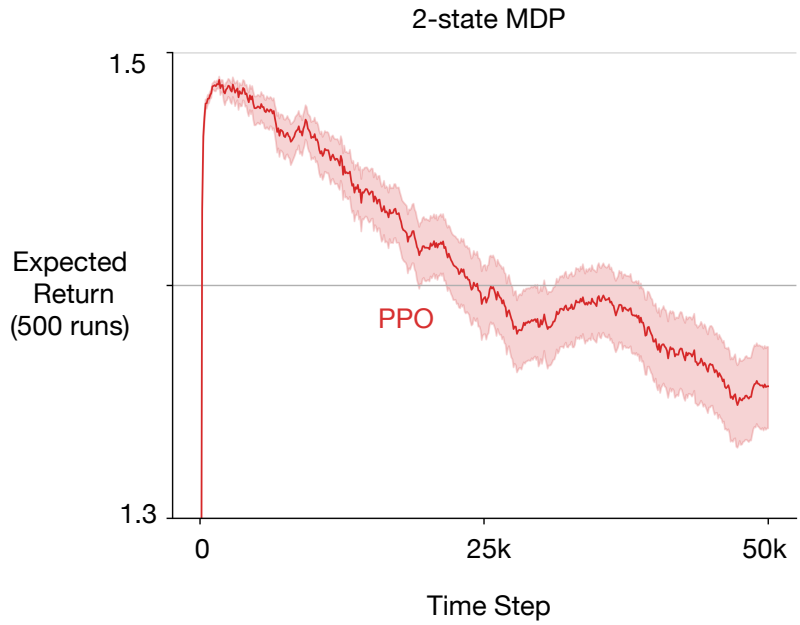

Figure 7.3: The performance of PPO on the 2-state MDP. A 2-dimensional vector represents the states of the MDP. When PPO learns using the network shown in Figure 7.2B, it lacks stability, and its performance degrades as the agent continues interacting in the MDP.

The performance of PPO in Figure 7.3 follows the same trend as its performance in the Mujoco environments. The performance of PPO improves at first, then it plateaus at a high level and finally drops. There are two notable things when we look at an individual run of PPO in Figure 7.4A. First, the agent gets stuck at different sub-optimal policies. And second, the learned policy is very unstable. It fluctuates rapidly between different policies.

The agent uses a neural network and a softmax policy parameterization. As the optimal policy is deterministic, the optimal values of the weights have infinite magnitude. This means that the gradient will force the outgoing weights in the network to grow. The probability of taking a different action will decrease exponentially as the weights increase. Once the weights become large enough, the agent almost never takes an exploratory action. Figure 7.4B shows the evolution of the outgoing weights of the policy network, as suspected, the weights kept increasing over time. The large

magnitude of outgoing weights explains why the learned policy gets stuck at different deterministic policies.

The sudden jumps in the policy in Figure 7.4A suggest that there might be sudden large changes in the representation provided by the single hidden unit. Figure 7.4C provides evidence for this hypothesis. Figure 7.4C shows the absolute difference in the output of the hidden unit for the two states. Let's call the output of the hidden unit for a given state the representation of the state. Note that when the difference in the representation of the two states is small, the states will look similar to the final layer. Figure 7.4 shows that sudden jumps in the policy happen whenever the difference in the state representation changes dramatically. The sudden changes in the representation should follow large changes in the input weights, which we observe in Figure 7.4D. Figure 7.4E shows the average magnitude of the gradients of the input layer. Perhaps surprisingly, there are large changes in the input weights even when the gradient is small, such as at step 17495.

The large updates, even when the gradient is small, are due to the Adam optimizer. Adam keeps running averages of the first and second moments of the gradient. The averages use β_1 and β_2 to control the importance of recent gradients in the average. In this experiment, we set $\beta_1 = 0.9$ and $\beta_2 = 0.999$, which are the most commonly used values of βs in deep reinforcement learning. In the individual run, when the policy changes at time step 17495, there is a sudden non-zero gradient. The sudden non-zero gradient happened because the agent took an exploratory action. Assuming that the gradient before and after time step 17495 is exactly zero, then during the next ten updates, these values of β_1 and β_2 would lead to an update that is 20 times larger than the gradient That is because the first update will be 3.16 times the gradient, and the future updates will decay by a factor of 0.9 (β_1), as β_2 is very large. This large weight change, even when the gradient was small, can lead to a large change in the policy, which explains why the learned policy fluctuates so abruptly.

Policy collapse in the 2-state MDP gave us various insights into the phenomenon.

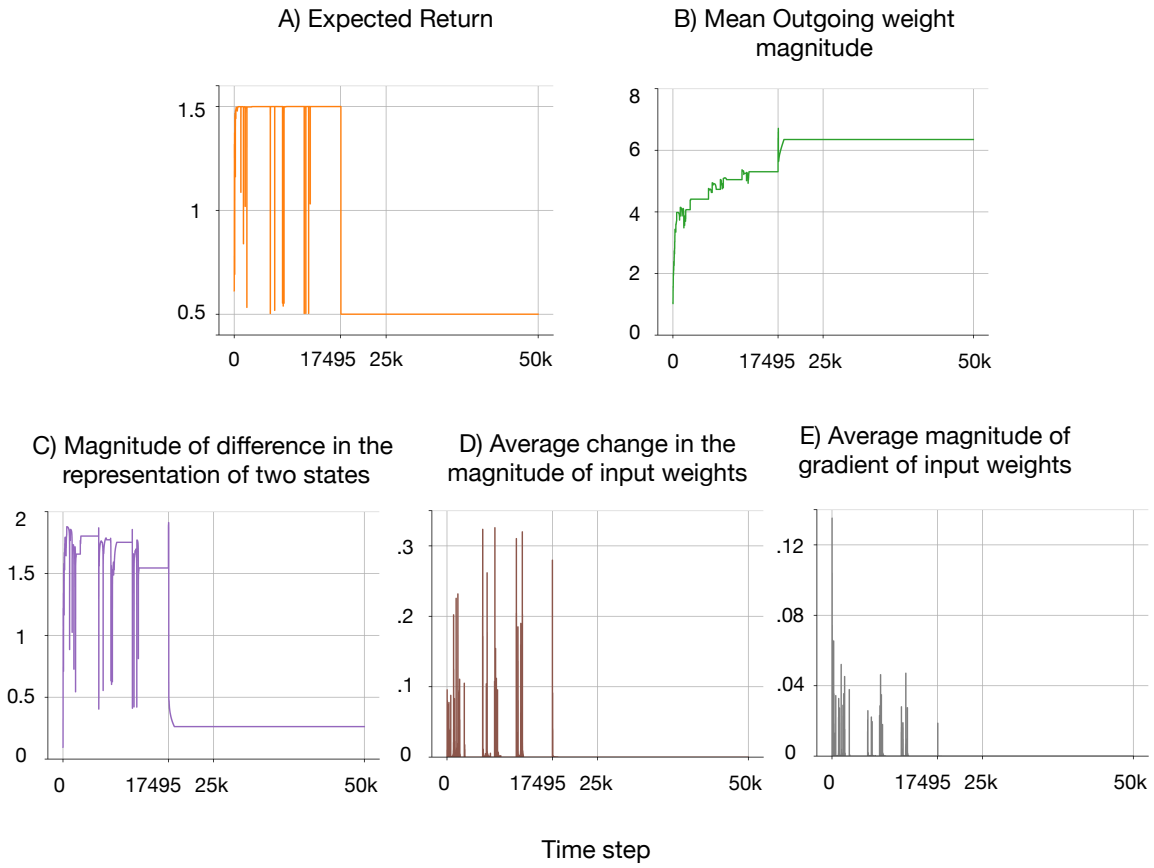

Figure 7.4: A deep look at one specific run of PPO on the 2-state MDP. Figures plot the evolution of different quantities. The magnitude of the output weights of the network kept increasing (Figure B) because the optima lie at infinity, making it difficult to try exploratory actions. Once the representation of the two states became sufficiently similar, at time step 17495 (Figure C), the agent kept taking the same action in both states. This resulted in the agent getting stuck at a sub-optimal policy (Figure A). The sudden large changes in input weights (Figure D) caused sudden large changes in the representation and the learned policy. These large changes were caused by the standard use of the Adam optimizer, which caused large weight changes even when the gradient was small (Figure E).

We learned that the agent gets stuck at sub-optimal policies when the representation does not separate the two states and outgoing weights become too large. And we found that the instability in PPO is due to the standard use of Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$), which caused sudden changes in the policy even when the gradient was small. These sudden changes caused the agent to forget the previously learned policy. This instability caused by the standard use of Adam could be the reason why Adam has been observed to cause more forgetting than SGD (Ashley et al., 2021). Similar to our analysis, Lyle et al. (2023) found that the standard use of Adam causes issues learning from a non-stationary stream of data. They showed that standard use of Adam worsens the loss of plasticity. In complement to that, we showed that the standard use of Adam also causes forgetting.

7.3 Reducing Policy Collapse with Tuned Adam

Insights from policy collapse in the 2-state MDP guide us towards a potential solution to policy collapse. If we can tackle the problem of larger-than-intended updates, we might be able to mitigate the issue of policy collapse. Lyle et al. (2023) suggested using equal values for β_1 and β_2. Recall that these parameters control the rate of the running averages for the first and second moments of the gradient. We call Adam with equal values for β_1 and β_2 *tuned Adam*.

First, we tested if PPO with tuned Adam can overcome policy collapse in the 2-state MDP. The experiment design and hyperparameters were the same as in the previous section. For tuned Adam, we kept $\beta_1 = \beta_2$ and tried values of $0.9, 0.99, 0.999$, and found that $\beta_1 = \beta_2 = 0.99$ performed best in this problem. We conducted 500 independent runs on this problem. The results are plotted in Figure 7.5. The shaded region in the graph shows the 95% boot-strapped confidence interval.

The data in Figure 7.5 shows that PPO with tuned Adam can maintain a good level of performance. Tuned Adam mitigated policy collapse with PPO in the 2-state MDP. Note that the performance of PPO with tuned Adam was stable. This is not

surprising as tuned Adam fixes the source of instability. Although it gets to a high level of performance, it does not find the optimal policy, a return of 1.5, in all runs, as its average performance is about 1.48.

We now test if tuned Adam can overcome policy collapse in the Ant environment. The experiment design is the same as in Section 7.1. The results of this experiment are shown in Figure 7.6. The data in Figure 7.6 shows the PPO with tuned Adam performs significantly better than standard PPO, but even its performance worsens after initial improvement.

To verify if tuned Adam has a similar effect of stabilizing learning in the Ant environment, we measured the the largest total weight change in the network during a single update cycle for bins of 1M steps. We plot this quantity in Figure 7.7. The first

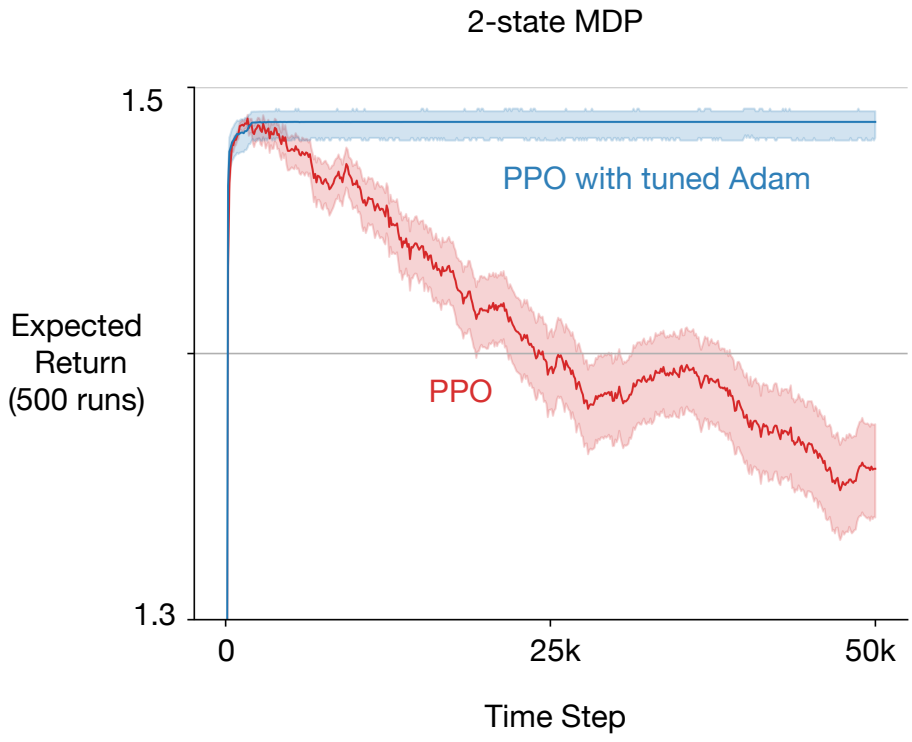

Figure 7.5: PPO with tuned Adam on the 2-state MDP. Tuned Adam uses the same rate for keeping the averages for the first and second moments of the gradient. Tuned Adam successfully mitigate policy collapse.Although, there is still room for improvement as neither algorithm gets to the optimal policy (return of 1.5) in all runs.

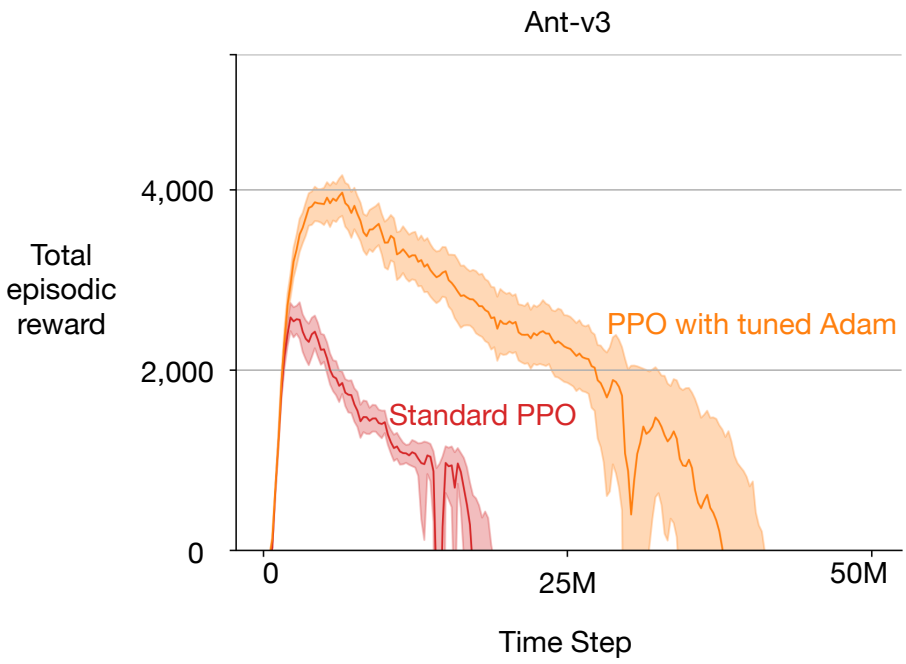

Figure 7.6: PPO with tuned Adam on Ant-v3. Tuned Adam substantially improves the performance of PPO, but there is still a significant drop in performance over time. These results are averaged over 30 runs. The solid lines represent the mean, and the shaded regions correspond to a 95% bootstrapped confidence interval.

point in the plots shows the largest total weight change during a single update cycle in the first 1M steps. The second point shows the largest weight change in the network during a single update cycle the second 1M steps and so on. The Figure 7.7 shows that standard Adam consistently causes very large updates to the weights which can destabilize learning, while tuned Adam with $\beta_1 = \beta_2 = 0.99$ has significantly smaller updates which leads to more stable learning. The failure of standard Adam with PPO is similar to the failure of standard Adam in Permuted MNIST.

The failure of PPO with tuned Adam in the Ant problem suggests that there are additional factors at play with the larger network in the Ant problem. We look deeper into the policy networks in Figure 7.8. We plot the evolution of weight magnitudes, dormant units, and stable rank in the policy network. The results are quite interesting, and they show that as training continues, the weights in the network get larger, the rank of the representation decreases, and a large fraction of units become dormant

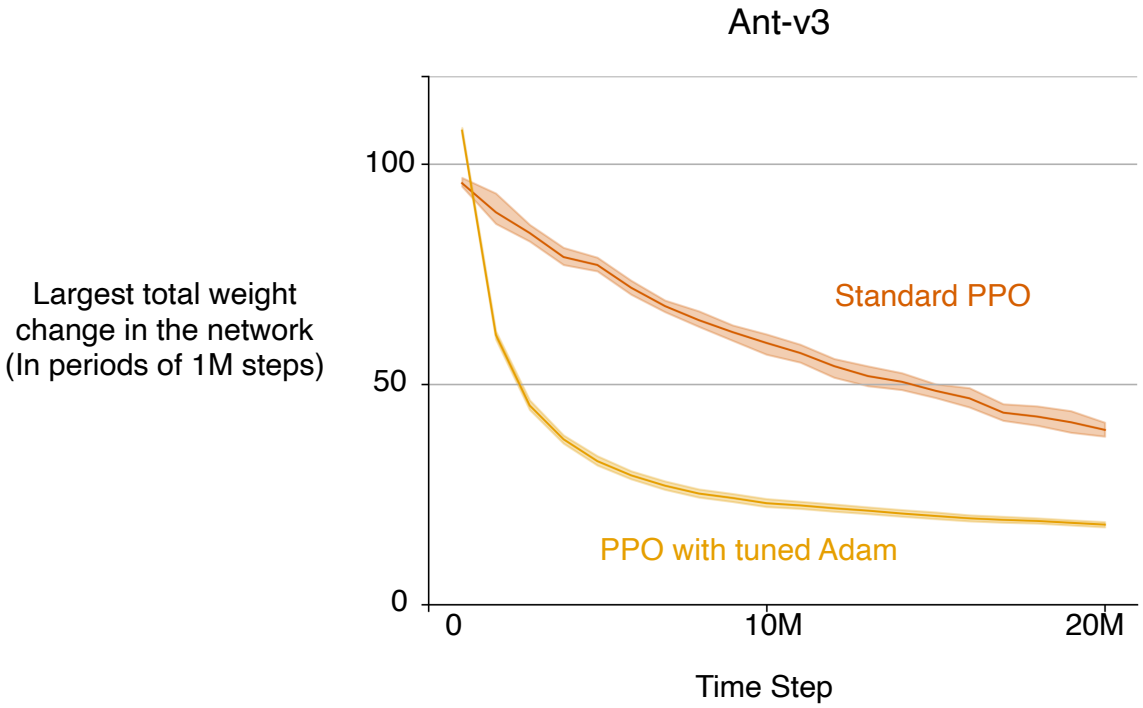

Figure 7.7: PPO with standard Adam leads to larger updates in the policy network compared with tuned Adam ($\beta_1 = \beta_2 = 0.99$) in Ant-v3, similar to what we saw in the 2-state-MDP. These large updates explains why PPO with tuned Adam is more stable than standard PPO in Ant-v3. These results are averaged over 30 runs. The solid lines represent the mean, and the shaded regions correspond to a 95% bootstrapped confidence interval.

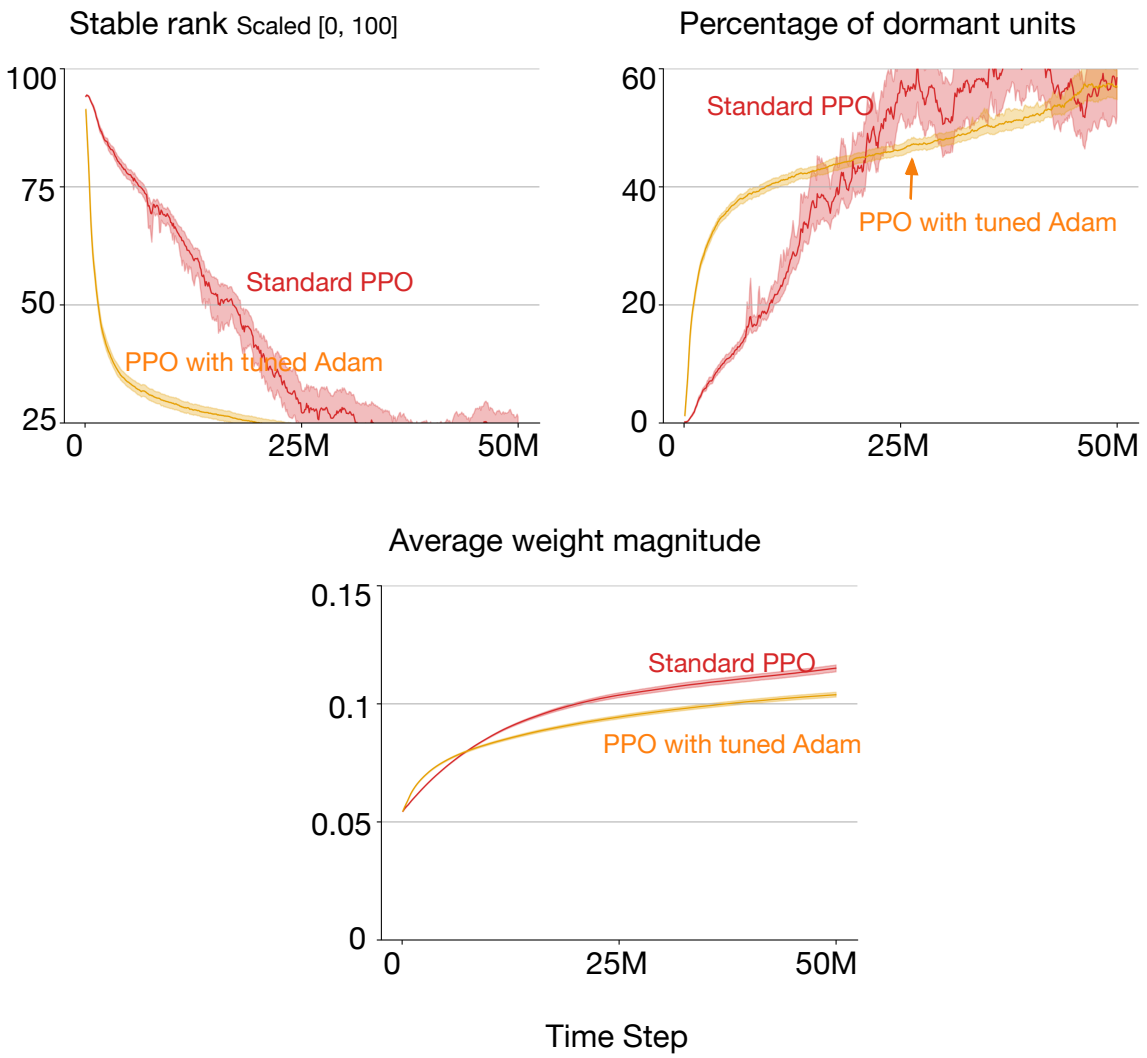

Figure 7.8: A closer look inside the policy network trained by PPO with tuned Adam on Ant-v3. These plots reveal a similar pattern as in continual supervised learning. The network continually loses plasticity as its weights keep growing, the fraction of dormant units increases, and the stable rank of the representation decreases. Although tuned Adam stabilizes weight updates, it does not mitigate the loss of plasticity. These results are averaged over 30 runs. The solid lines represent the mean, and the shaded regions correspond to a 95% bootstrapped confidence interval.

over time. These results are strikingly similar to the correlates of loss of plasticity we saw in Chapter 4. They show that the networks are losing plasticity and suggest that plasticity-injecting algorithms like continual backpropagation might mitigate policy collapse.

7.4 Overcoming Policy Collapse using Continual Backpropagation

In the last section, we saw that networks trained via PPO with tuned Adam lose plasticity over time. A natural next step is to test if continual backpropagation can mitigate this loss of plasticity in reinforcement learning and if it can overcome policy collapse. We tested both L2 regularization and continual backpropagation (along with a small amount of L2 regularization) applied to PPO with tuned Adam. The performance of these algorithms on the Ant problem is shown in Figure 7.9. For all the experiments presented in this section, all algorithms other than standard PPO used tuned Adam with $\beta_1 = \beta_2 = 0.99$.

The data in Figure 7.9 shows that both L2 regularization and continual backpropagation mitigate policy collapse. However, only the performance of continual backpropagation improves over time, while PPO with L2 regularization plateaus at a sub-optimal policy. PPO with continual backpropagation scales with experience as its performance keeps improving over time.

In Figure 7.10, we plot the correlates of loss of plasticity for all the algorithms. The Figure 7.10 shows that both PPO with continual backpropagation and L2 regularization significantly reduce the fraction of dormant units and maintain a high stable rank. Additionally, both algorithms have a non-growing average weight magnitude. However, L2 regularization only worked well when aggressive regularization was used with PPO (10^{-3}). The algorithm still suffered from policy collapse for other smaller regularization values, and for larger values, it never achieved good performance. This strong value regularization strongly restricted the space of policies the agent could

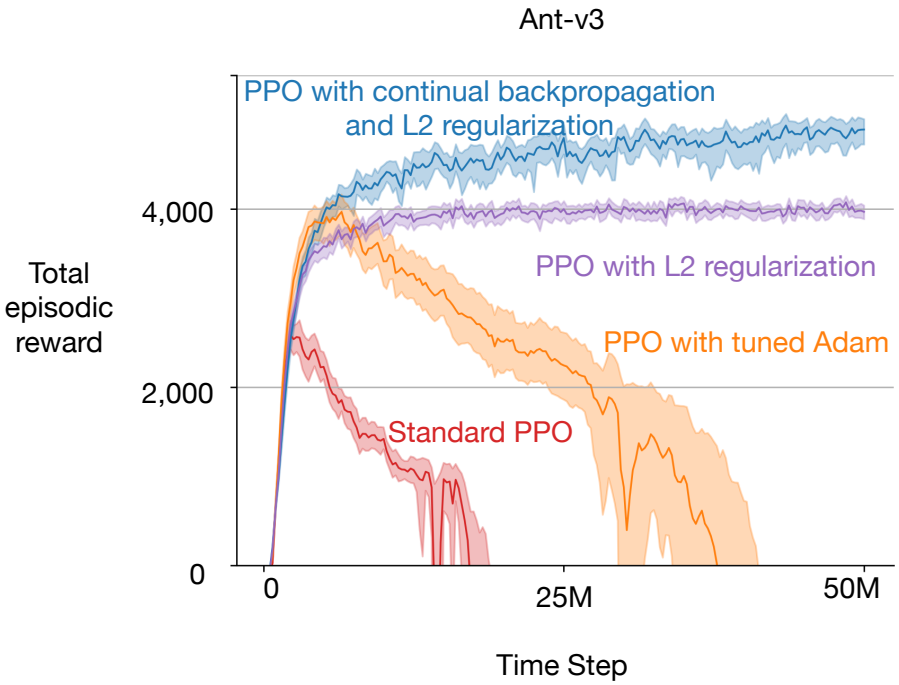

Figure 7.9: The performance of various algorithms on Ant-v3. All algorithms other than standard PPO use tuned Adam. These results are averaged over 30 runs. The solid lines represent the mean, and the shaded regions correspond to a 95% bootstrapped confidence interval. PPO with tuned Adam substantially improves over standard PPO but still shows a significant drop in performance over time. Performance of PPO with L2 regularization plateaus at a suboptimal level. The performance of PPO with continual backpropagation (and weak regularization) keeps improving.

represent and prevented it from reaching better policies. On the other hand, continual backpropagation works well with weaker regularization, allowing the algorithm to represent better policies.

We also evaluated PPO and its variants in two additional environments: Hopper-v3 and Walker-v3. The results for these experiments are presented in Figure 7.11. The results mirrored those from Ant-v3; standard PPO suffered from a dramatic degradation in performance, where its performance dropped dramatically. However, this time, L2 regularization did not fix the issue in all cases; there was some performance degradation with L2 in Walker-v3. PPO, with continual backpropagation and L2 regularization, completely fixed the issue in all environments. Note that the only difference between our experiments and what is typically done in the literature

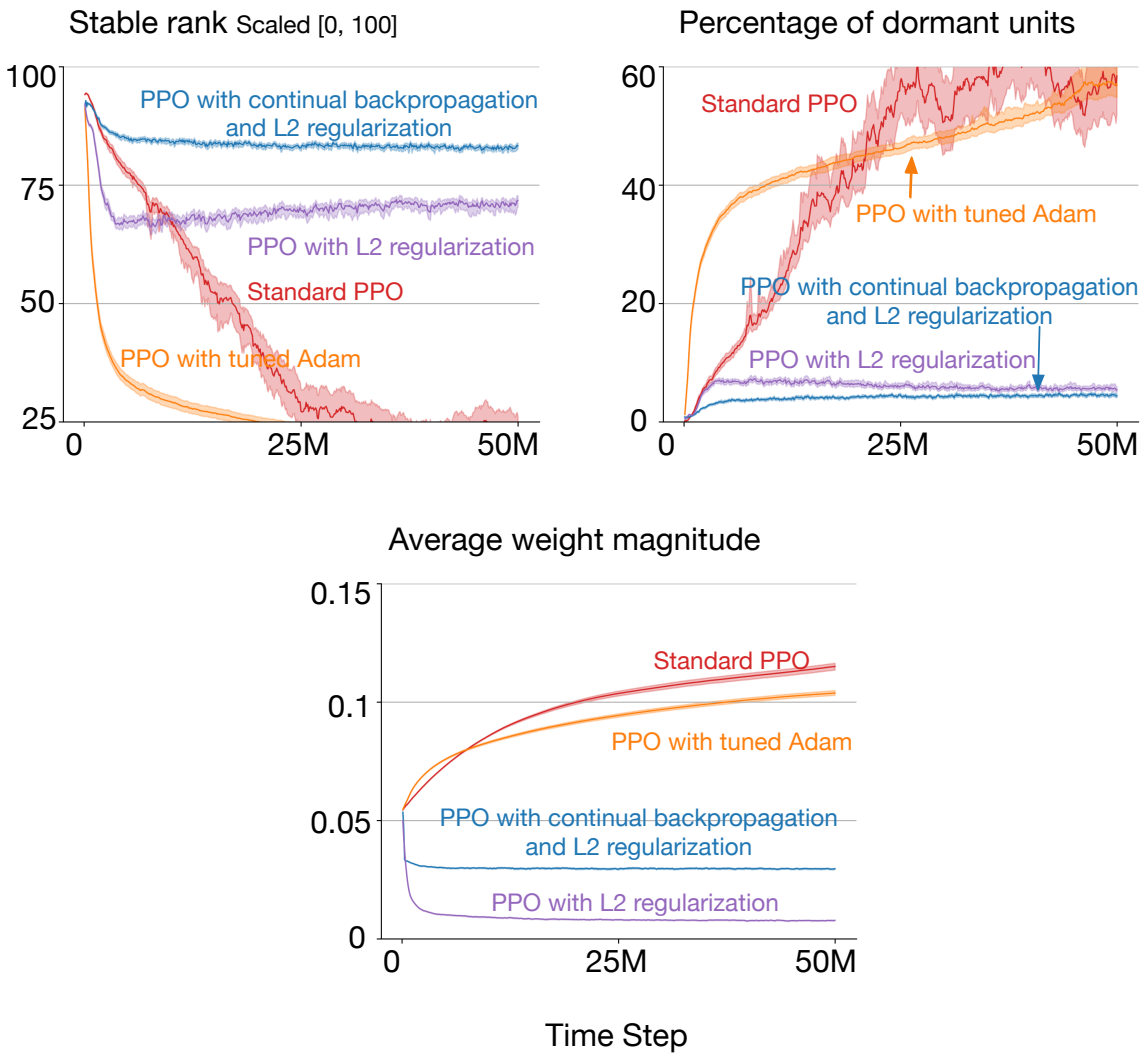

Figure 7.10: The evolution of three correlates of plasticity in the Ant problems. Continual backpropagation and L2 regularization mitigate all three correlates. Additionally, continual backpropagation maintains a higher stable rank and higher average weight magnitude. These results are averaged over 30 runs. The solid lines represent the mean, and the shaded regions correspond to a 95% bootstrapped confidence interval.

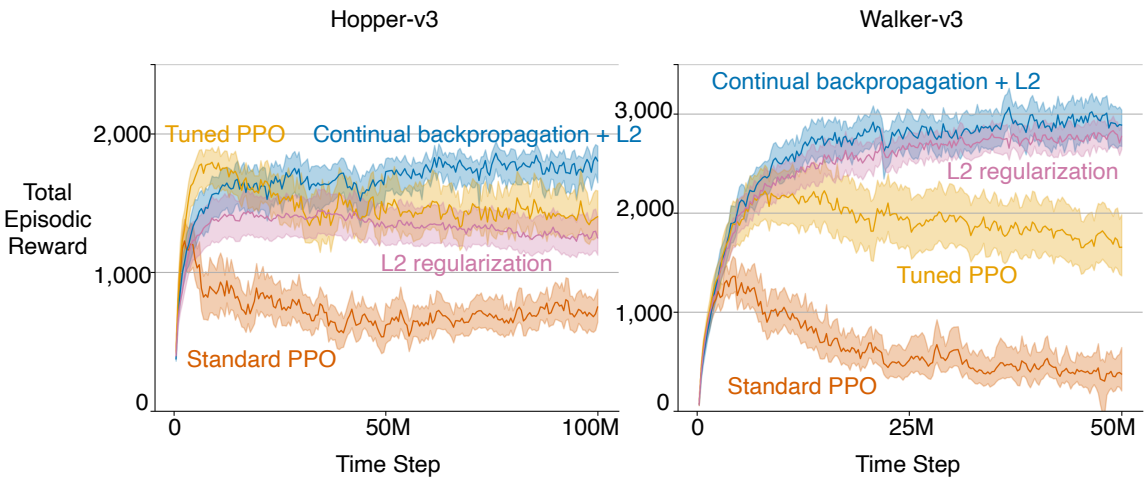

Figure 7.11: Performance of all algorithms on Hopper-v3 and Walker-v3. Similar to the Ant environment, the performance of PPO and PPO with tuned Adam drops over time in Hopper-v3 and Walker-v3. However, unlike in the Ant environment, the performance of PPO with L2 regularization gets worse over time in Hopper-v3. On the other hand, PPO with continual backpropagation and L2 regularization can keep improving over time. These results are averaged over 30 runs. The solid lines represent the mean, and the shaded regions correspond to a 95% bootstrapped confidence interval.

is that we run the experiments for longer. Typically, these experiments are only done for 3M steps, but we ran these experiments for up to 100M steps.

We tried a wide range of hyper-parameters for continual backpropagation in all the environments. The common trend was that continual backpropagation required a large value of maturity threshold in these reinforcement learning experiments. A maturity threshold of at least 1000 was required for good performance in reinforcement learning. Note that this is larger than the good value of the maturity threshold for the continual supervised learning experiments in Chapter 4. A maturity threshold of 100 was good in supervised learning problems. The optimal replacement rate and maturity threshold value for all three environments were 10^{-4} and 10^4, respectively.

Recall from Chapter 6 that one way to understand the behaviour of continual backpropagation is to look at the number of eligible units for replacement. For replacement rate and maturity threshold of 10^{-4} and 10^4, there are 128 eligible units

in each layer of 256 units. This means that continual backpropagation is performing an aggressive search process in this problem, reinitializing units with non-negligible utility. We found that continual backpropagation with a small value of maturity threshold (close to 100) did not mitigate policy collapse in this problem. Continual backpropagation with a small maturity threshold generally behaves like an algorithm that only removes dormant units. A high value of maturity threshold in these environments suggests that just removing dormant units is insufficient for maintaining plasticity, and a more aggressive search process is needed in these problems.

ReDo (Sokar et al., 2023) and Self-normalizing Resets (SNR) (Farias and Jozefiak, 2025) are variations of continual backpropagation that are explicitly designed to remove dormant units from networks. We want to test if the intuition we gained from the hyperparameters of continual backpropagation is useful. Failure of ReDo, SNR, and continual backpropagation with a small maturity threshold will support the intuition that removing dormant units is insufficient for maintaining plasticity in these problems.

We tested ReDo and SNR on Ant-v3. ReDo requires two parameters: threshold and the reinitialization period. We tested ReDo for all combinations of threshold in $\{0.01, 0.03, 0.1\}$ and reinitialization period in $\{10, 10^2, 10^3, 10^4, 10^5\}$; a threshold of 0.1 and with reinitialization period of 10^2 performed the best. Similar to ReDo, SNR requires two parameters: rejection percentile and window size for threshold updates. We tested SNR for all combinations of rejection percentile in $\{0.1, 0.01, 0.001\}$ and window size of $\{10^2, 10^3, 10^4\}$. All combinations of these parameters had statistically similar performance. This aligns with the results by Farias and Jozefiak (2025), where they showed that SNR is highly insensitive to its parameters. ReDo, SNR and continual backpropagation were used with weight decay of 10^{-4}. The performance of PPO with ReDo and PPO with SNR is plotted in Figure 7.12. For SNR, we plot the performance for a rejection percentile of 0.1 and a window length of 1000.

Another method that has been recently proposed to mitigate loss of plasticity is

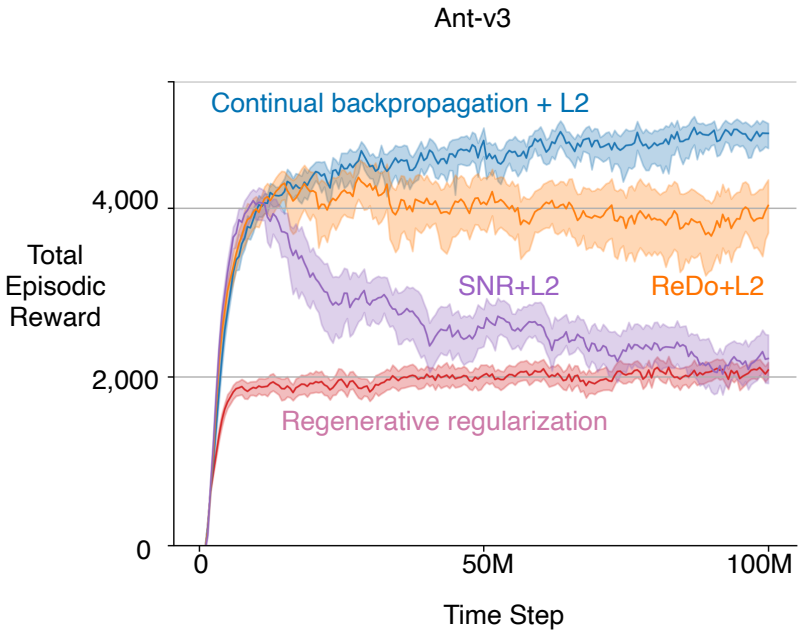

Figure 7.12: Comparison of continual backpropagation, ReDo, SNR, and regenerative regularization on Ant-v3. ReDo and SNR are selective reinitialization methods that use different utility measures and reinitialization strategies than continual backpropagation. Only the performance of PPO with continual backpropagation and L2 regularization keeps improving over time. These results are averaged over 30 runs. The solid lines represent the mean, and the shaded regions correspond to a 95% bootstrapped confidence interval.

regenerative regularization. In regenerative regularization, the weights are regularized towards their initial value. We also tested regenerative regularization on Ant-v3. Regenerative regularization requires a parameter that controls the strength of regularization. We test regenerative regularization with regularization strength parameter in $\{1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. Regenerative regularization with regularization strength of 0.1 performed the best. We did not use weight decay towards zero with regenerative regularization. The performance of PPO with regenerative regularization is plotted in Figure 7.12.

Figure 7.12 shows that PPO with ReDo and L2 regularization performs significantly better than standard PPO. However, it still suffers from slight performance degradation, and its performance is worse than PPO with continual backpropagation and L2 regularization. On the other hand, PPO with SNR and L2 regularization suf-

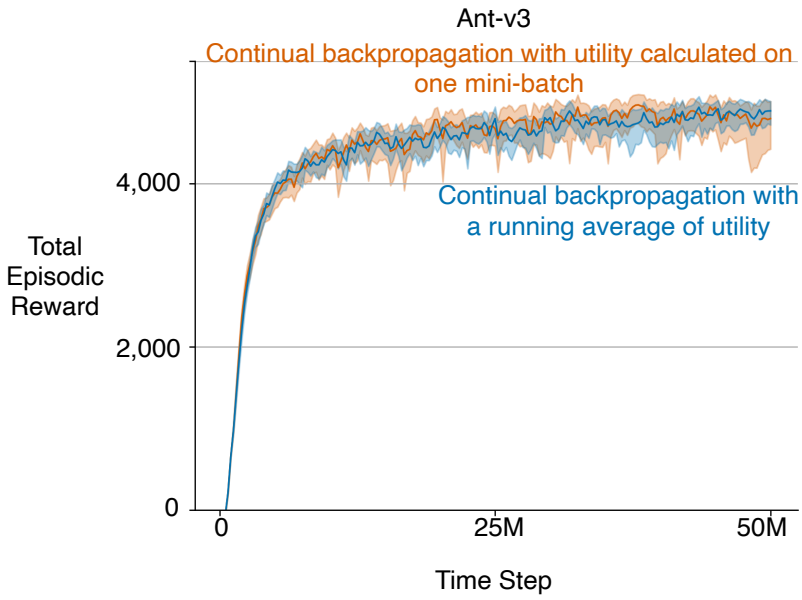

Figure 7.13: Comparison of two forms of utility in continual backpropagation in Ant-v3. The first form of utility calculated utility over just one mini-batch, while the other kept a running average of utilities over mini-batches. Both variations have similar performance. The utility in continual backpropagation can be calculated over just one mini-batch without a performance drop, reducing its computational requirements. These results are averaged over 30 runs. The solid lines represent the mean, and the shaded regions correspond to a 95% bootstrapped confidence interval.

fers from significant performance degradation. Although the policy does not entirely collapse with SNR. Finally, PPO with regenerative regularization does not suffer from performance degradation, but its performance is always poor.

The failure of ReDo and SNR suggests that just removing dormant units is insufficient to maintain plasticity in this problem, and a more aggressive search process, as provided by continual backpropagation with a large maturity threshold, is required in these problems. Additionally, regenerative regularization only mitigated performance degradation for aggressive regularization. However, when regularization is too aggressive, the weights in the network become too restricted, preventing the network from representing effective policies.

Recall from Algorithm 1 in Chapter 6 that continual backpropagation uses a running average of instantaneous utility to estimate true utility. In these reinforcement learning problems, data is processed in mini-batches. This allows us to estimate the

true utility using the samples in the mini-batch without keeping a running average. Removing the need to keep a running average reduces the computational requirement for continual backpropagation. In the next experiment, we test if continual backpropagation with utility calculated over one mini-batch is sufficient for good performance. The results are presented in Figure 7.13, and they show that continual backpropagation has the same performance on Ant whether we use utility calculated over a mini-batch or if we keep running average over mini-batches.

This section showed that plasticity-injecting algorithms like continual backpropagation can maintain plasticity in reinforcement learning problems. This plasticity, in turn, is sufficient to overcome policy collapse. Continual backpropagation maintained a high stable rank, a low percentage of dormant units and a non-growing weight magnitude. PPO with continual backpropagation scaled with experience in three Mujoco environments, and it outperforms ReDo, which is another selective reinitialization algorithm. Additionally, continual backpropagation works just as well if we only use one mini-batch to calculate utility, which reduces its computational usage.

7.5 Maintaining Plasticity in Non-Stationary Reinforcement Learning

In this section, we study loss of plasticity in a non-stationary reinforcement learning problem. In one sense, non-stationary reinforcement learning is the most challenging setting we study in this thesis because it involves many sources of non-stationary. There is non-stationarity due to a changing environment, the continually changing data distribution due to the agent's changing behaviour, and the use of temporal difference learning. All the sources of non-stationarity make it a challenging problem for continual learning algorithms.

We use the ant problem to create a non-stationary problem. In the ant problem, a simulated ant-like robot is tasked with moving forward as rapidly and efficiently as possible. The agent receives a reward depending on the forward distance travelled

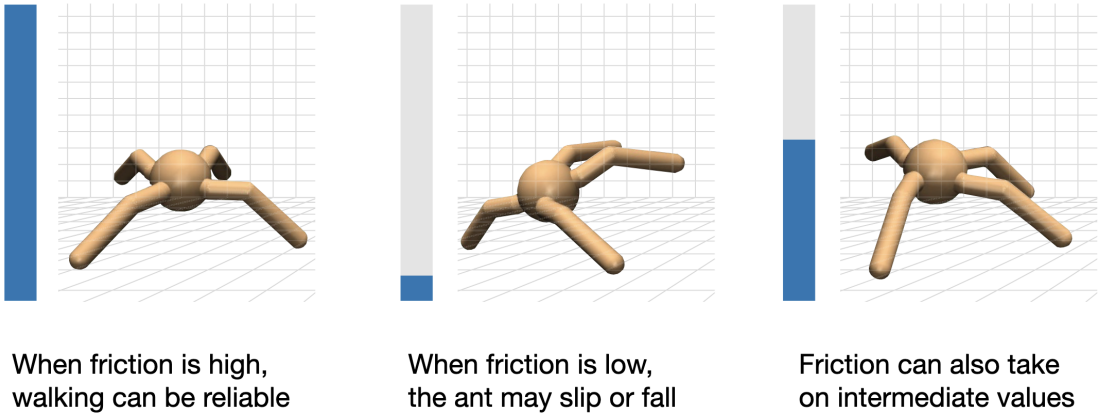

Figure 7.14: We make a non-stationary reinforcement learning problem by changing the friction between the simulated ant robot and the ground. The changing friction forces the agent to learn different behaviours to walk on different surfaces.

by the robot. One way to make this task non-stationary is to change the friction between the agent and the ground. The agent must adapt its walking method each time the friction changes to cover the most distance in an episode, see Figure 7.14 for an illustration. This will force the agent to learn different behaviours for different friction values, similar to how humans learn to walk differently on different surfaces like ice, sand, paved ground, and dirt trails. Specifically, the coefficient of friction between the feet of the ant and the floor is changed after every 2 million time steps. We changed the coefficient of friction by sampling it log-uniformly from the range $[0.02, 2]$, using a logarithm with base 10. The coefficient of friction changed at the first episode boundary after 2M time steps had passed since the last change.

We evaluate standard PPO, PPO with tuned Adam ($\beta_1 = \beta_2 = 0.99$), PPO with tuned Adam and L2-regularization, and PPO with tuned Adam and continual back-propagation and L2-regularization on this problem. We conducted 100 independent runs for all algorithms. We needed 100 runs to establish statistical significance because different friction values across different runs add another source of uncertainty in the results. For all algorithms, we used the same 100 sequences of frictions; this ensures that the problem's difficulty is the same for all algorithms. The results of this experiment are plotted in Figure 7.15.

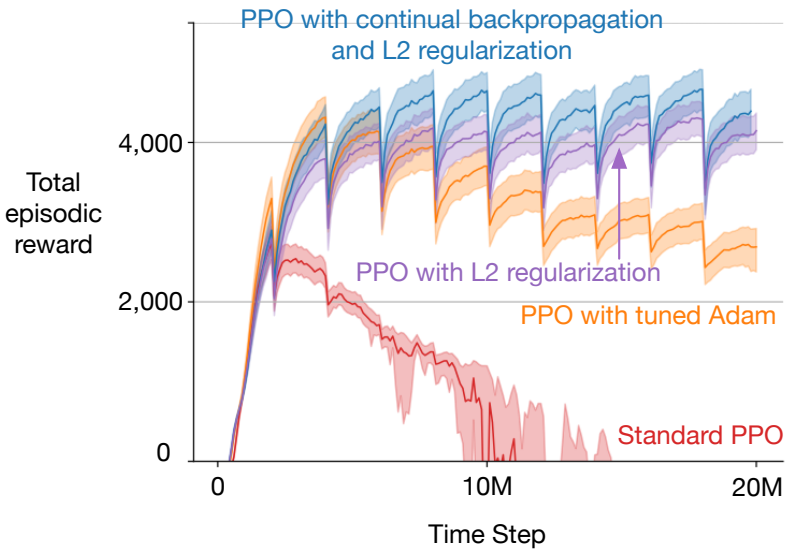

Figure 7.15: Performance of various algorithms on the Ant problem with changing friction. These results are averaged over 100 runs. The solid lines represent the mean, and the shaded regions correspond to a 95% bootstrapped confidence interval. Standard PPO algorithm fails catastrophically on the non-stationary ant problem. Similar to the stationary Ant problem, when we set $\beta_1 = \beta_2 = 0.99$ for Adam, then the failure is less severe, but adding continual backpropagation or L2 regularization is necessary to perform well indefinitely.

The performance curves in Figure 7.15 are jagged, similar to a saw tooth. The sudden drops in performance correspond to the change in friction. Every time the friction changes, the old behaviour is not as effective, which results in a low performance. The performance of standard PPO follows the same trend as in the stationary Ant problem; see Figure 7.9. It performs well initially, but over time, its performance drops, and it performs worse than it did in the beginning. PPO with tuned Adam performs much better than standard PPO, but even its performance worsened after the 2nd change in friction. PPO with L2 regularization and PPO with continual backpropagation and L2 regularization kept adapting and maintained a high level of performance as the friction kept changing. We also tested shrink-and-pertrub on this problem and found that it did not provide a significant performance improvement over L2 regularization.

In this section, we evaluated various algorithms in non-stationary reinforcement

learning algorithms. The results followed a similar trend as the stationary problems we studied earlier in the chapter. Standard PPO failed catastrophically under extended learning. Properly tuned, Adam improved the performance, but plasticity-preserving algorithms like L2 regularization and Continual backpropagation were needed to keep adapting to the changes in the environment.

7.6 Discussion

Following our work, Moalla et al. (2024) took a deeper dive into policy collapse with PPO and provided valuable insights into the phenomenon. We saw in this chapter that as learning continues, features saturate, and the representation of different states becomes similar. Moalla et al. (2024) found that these close representations affect the ability of PPO to stay in the trust region, and PPO makes updates that break the trust region. Because of the loss of plasticity, the learning system cannot recover and learn good representations that can separate different states and learn the optimal behaviour for all states.

One important result from this chapter is that setting $\beta_1 = \beta_2$ in Adam is crucial for stabilizing long-term learning in reinforcement learning. This year, many papers have come out that argue that Adam with $\beta_1 = \beta_2$ is the best choice for large-scale learning (Zhao et al., 2025; Shah et al., 2025; Orvieto and Gower, 2025). In a large-scale empirical study of language models, Orvieto and Gower (2025) find that Adam with $\beta_1 = \beta_2$ is the best choice for Adam's parameters. They show that Adam with $\beta_1 = \beta_2$ has a special property, where the update of Adam is $sign(m_k)/(1 + m_k^2/\sigma_k^2)$, where m_k is the estimate of the mean (momentum) and σ_k is the estimate of variance of the gradient. This means that Adam's update is bounded, and the signal-to-noise ratio controls the size of the update. They show that if $\beta_2 > \beta_1$ (which is the case in all applications of Adam), the updates can be significantly larger than the gradient. The bounded size of Adam's updates is important for non-stationary problems to ensure that the network remains stable.

Loss of plasticity expresses itself in various forms in deep reinforcement learning. Some work found that deep reinforcement learning systems can lose their generalization abilities in the presence of non-stationarities (Igl et al., 2021). A reduction in the effective rank, similar to the rank reduction in CIFAR-100, has been observed in some deep reinforcement learning algorithms (Kumar et al., 2021). Nikishin et al. (2022) showed that many reinforcement learning systems perform better if their network is occasionally reset to its naive initial state, retaining only the replay buffer. This is because the learning networks got worse than a reinitialized network at learning from new data. Recent work has improved performance in many reinforcement learning problems by applying plasticity-preserving methods (Sokar et al., 2023; Nikishin et al., 2023; D'Oro et al., 2023; Schwarzer et al., 2023; Lee et al., 2023; Delfosse et al., 2024). These works focused on deep reinforcement learning systems that use large replay buffers. Our work complements this line of research as we studied systems with much smaller replay buffers. Loss of plasticity is most relevant for systems that use small or no replay buffers, as large buffers can hide the effect of new data.

In recent years, there has been an increase in the interest in continual reinforcement learning (Abel et al., 2023). Abbas et al. (2023) showed that standard deep reinforcement learning algorithms suffer from a dramatic loss of plasticity when tasked to learn a sequence of Atari games. Some work has also studied various plasticity-preserving algorithms for on-policy learning in non-stationary reinforcement learning problems (Chung et al., 2024; Juliani and Ash, 2024). Ahn et al. (2025) studied loss of plasticity in conjunction with catastrophic forgetting in continual reinforcement learning.

7.7 Conclusion

In Mujoco environments and a 2-state MDP, we showed that the PPO algorithm does not scale with experience. After initial performance improvement, PPOs' performance starts to drop until they eventually perform worse than the initial random policy. A closer look at the algorithm in the 2-state MDP revealed that the default setting of the

Adam optimizer causes large updates, destabilizing the network and causing policy collapse. Fixing these large updates by setting $\beta_1 = \beta_2$ in Adam stabilized the learning and mitigated policy collapse in the 2-state MDP. This finding suggests that directly taking up tools from supervised learning is not always good for reinforcement learning problems. We should be more careful when borrowing tools from other domains. Schlegel et al. (2023) made a similar observation, but for recurrent networks. They showed that there is a big difference between the best design choices for recurrent networks in supervised and reinforcement learning problems.

We found that PPO loses plasticity in Mujoco environments. Similar to continual supervised learning, a large part of the network became dormant, the rank of the representation dropped, and the weights in the network increased. This reduced diversity in the representation meant that states started to look similar, which caused the system to make updates that affected too many states and worsened the policy. The loss of plasticity, in turn, meant that the system could not recover and relearn a good representation. Continual backpropagation overcame the loss of plasticity and enabled PPO to scale with experience.

Chapter 8

Conclusion and Future Work

The first key result of this dissertation is that standard deep learning methods do not work in continual learning problems. By standard deep learning methods, we mean the existing standard algorithms designed for learning from a fixed dataset. And by not working, we mean they lose the ability to learn new things over time. We provided evidence of loss of plasticity in both supervised and reinforcement learning problems. Our demonstrations of loss of plasticity included problems with a wide range of memory constraints, optimizers, activation functions, hyperparameters, under- and over-parameterized networks, rates of distribution change, network architectures, and various additional techniques like dropout, regularization and normalization. Altogether, these demonstrations provide substantial evidence of loss of plasticity in deep continual learning.

The second result is that algorithms like continual backpropagation and shrink and perturb that continually inject randomness and variability into the network can enable artificial neural networks to learn forever. Continual backpropagation maintained plasticity in all supervised and reinforcement learning problems we tested. Continual backpropagation performs a search process in the space of units. For some settings of its hyperparameters, the search can be slow, removing only near zero-utility units like dormant units. While for some other hyperparameters, the search can be much more aggressive, and units with non-negligible utility get replaced. Continual backprop-

agation learned faster than retraining from scratch. It also mitigated all correlates of loss of plasticity that we observed. It maintained non-growing weights, removed all dormant units from the network, and maintained a high stable rank throughout learning. By continually injecting new units into the network, continual backpropagation enables the benefits of initialization throughout learning, which enables artificial neural networks to learn continually.

8.1 Directions for Future Work

In recent years, very large-scale machine learning models have shown impressive capabilities (OpenAI, 2023; DeepSeek-AI, 2025). However, these systems are only trained once and do not learn continually. One direction for future work is to test whether these systems can maintain plasticity when learning from new data. Some recent results suggest that these systems might be losing plasticity (Zhang et al., 2025; Springer et al., 2025). However, the plasticity of these large-scale models has not been systematically studied. Such studies will be useful theoretically as they will deepen our understanding of these models. They will also be valuable from a practical perspective because continually learning models can be much more cost-efficient at absorbing new information on the internet than models retrained from scratch.

Machine learning systems that can continue to adapt to changes in their data stream can enable many new applications. Traditionally, machine learning applications have been limited to cases where the system does not have to learn new things, because deep learning systems struggle to learn new things. However, in recent years, some companies like Lyft and RL Core have started to use continual learning in applications. Progress in learning systems that can maintain plasticity enables applications in forecasting in highly complicated domains like stock markets or weather. Continual adaptation is also important for developing control systems in factories, as there is a need for continual learning due to regular wear and tear. Problems requiring on-device learning, like prosthetics, robots, and deep space exploration, require

continual adaptation due to the ever-changing real world.

Continual backpropagation combines search with gradient descent. However, it is just the first version of this idea, and there are many open questions. It is unclear what the best way to measure the utility is. Should the utility be based on the effect of a unit on the current data point or its effect on the entire future data stream? Another open question is whether we can use utility to protect useful information and mitigate forgetting. A long-term utility measure instead of instantaneous utility can be useful to protect useful information from catastrophic forgetting. Perhaps the most important question is how do we scale this form of search. The search in continual backpropagation is quite slow, as only one unit is typically replaced once every few hundred steps. If more units are replaced, the network's output is severely affected, and the performance worsens. Wide networks can be useful for scaling search, as more units can be replaced without severely affecting the network's output. Another idea is to use gradient descent to learn through a core/backbone network and perform search in a large fringe. Future versions of continual backpropagation might significantly improve the capabilities of continual learning systems and act as a powerful complement to gradient descent-based deep learning.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*.

Abbas, Z., Zhao, R., Modayil, J., White, A., and Machado, M. C. (2023). Loss of Plasticity in Continual Deep Reinforcement Learning. In *Conference on Lifelong Learning Agents*.

Abel, D., Barreto, A., Roy, B. V., Precup, D., van Hasselt, H., and Singh, S. (2023). A Definition of Continual Reinforcement Learning. In *Advances in Neural Information Processing Systems 37*.

Ahn, H., Hyeon, J., Oh, Y., Hwang, B., and Moon, T. (2025). Reset & Distill: A Recipe for Overcoming Negative Transfer in Continual Reinforcement Learning. In *13th International Conference on Learning Representations*.

Aimone, J. B., Deng, W., and Gage, F. H. (2010). Adult Neurogenesis: Integrating Theories and Separating Functions. *Trends in Cognitive Sciences*, 14(7):325–337.

Aljundi, R., Belilovsky, E., Tuytelaars, T., Charlin, L., Caccia, M., Lin, M., and Page-Caccia, L. (2019). Online Continual Learning with Maximal Interfered Retrieval. In *Advances in Neural Information Processing Systems 33*.

Altman, J. (1963). Autoradiographic Investigation of Cell Proliferation in the Brains of Rats and Cats. *The Anatomical Record*, 145(4):573–591.

Ash, J. and Adams, R. P. (2020). On Warm-Starting Neural Network Training. In *Advances in Neural Information Processing Systems 33*.

Ashley, D. R., Ghiassian, S., and Sutton, R. S. (2021). Does the Adam Optimizer Exacerbate Catastrophic Forgetting? Preprint at https://arxiv.org/abs/2102.07686.

Bellec, G., Kappel, D., Maass, W., and Legenstein, R. (2018). Deep Rewiring: Training Very Sparse Deep Networks. In *6th International Conference on Learning Representations*.

Benna, M. K. and Fusi, S. (2016). Computational Principles of Synaptic Memory Consolidation. *Nature Neuroscience*, 19(12):1697–1706.

Berariu, T., Czarnecki, W., De, S., Bornschein, J., Smith, S., Pascanu, R., and Clopath, C. (2021). A Study on the Plasticity of Neural Networks. Preprint at https://arxiv.org/abs/2106.00042.

Bonin, P., Barry, C., Méot, A., and Chalard, M. (2004). The Influence of Age of Acquisition in Word Reading and Other Tasks: A Never Ending Story? *Journal of Memory and Language*, 50(4):456–476.

Boyd, S. P. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Campbell, D. T. (1960). Blind Variation and Selective Survival as a General Strategy in Knowledge-processes. *Self-organizing Systems*, 2:205–231.

Campbell, M., Hoane Jr, A. J., and Hsu, F.-h. (2002). Deep Blue. *Artificial Intelligence*, 134(1-2):57–83.

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

Caruana, R. (1997). Multitask Learning. *Machine Learning*, 28(1):41–75.

Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. (2018). Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence. In *Proceedings of the European Conference on Computer Vision*.

Chen, T., Cheng, Y., Gan, Z., Yuan, L., Zhang, L., and Wang, Z. (2021). Chasing Sparsity in Vision Transformers: An End-to-end Exploration. In *Advances in Neural Information Processing Systems 35*.

Chiley, V., Sharapov, I., Kosson, A., Koster, U., Reece, R., Samaniego de la Fuente, S., Subbiah, V., and James, M. (2019). Online Normalization for Training Neural Networks. In *Advances in Neural Information Processing Systems 33*.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2017). A Downsampled Variant of ImageNet as an Alternative to the CIFAR Datasets. Preprint at https://arxiv.org/abs/1707.08819.

Chung, W., Cherif, L., Precup, D., and Meger, D. (2024). Parseval Regularization for Continual Reinforcement Learning. In *Advances in Neural Information Processing Systems 38*.

Citri, A. and Malenka, R. C. (2008). Synaptic Plasticity: Multiple Forms, Functions, and Mechanisms. *Neuropsychopharmacology*, 33(1):18–41.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London.

DeepSeek-AI (2025). DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. Preprint at https://arxiv.org/abs/2501.12948.

Delfosse, Q., Schramowski, P., Mundt, M., Molina, A., and Kersting, K. (2024). Adaptive Rational Activations to Boost Deep Reinforcement Learning. In *12th International Conference on Learning Representations*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-scale Hierarchical Image Database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*.

Dennett, D. C. (1975). Why the Law of Effect Will Not Go Away. *Journal for the Theory of Social Behaviour*.

Dohare, S. (2020). The Interplay of Search and Gradient Descent in Semi-stationary Learning Problems. Master's thesis, University of Alberta.

Dohare, S., Hernandez-Garcia, J. F., Lan, Q., Rahman, P., Mahmood, A. R., and Sutton, R. S. (2024). Loss of Plasticity in Deep Continual Learning. *Nature*, 632(8026):768–774.

Dohare, S., Hernandez-Garcia, J. F., Rahman, P., Mahmood, A. R., and Sutton, R. S. (2023). Maintaining Plasticity in Deep Continual Learning. Preprint at https://arxiv.org/abs/2306.13812.

Dohare, S., Sutton, R. S., and Mahmood, A. R. (2021). Continual Backprop: Stochastic Gradient Descent with Persistent Randomness. Preprint at https://arxiv.org/abs/2108.06325.

D'Oro, P., Schwarzer, M., Nikishin, E., Bacon, P.-L., Bellemare, M. G., and Courville, A. (2023). Sample-Efficient Reinforcement Learning by Breaking the Replay Ratio Barrier. In *11th International Conference on Learning Representations*.

Ellis, A. W. and Lambon Ralph, M. A. (2000). Age of Acquisition Effects in Adult Lexical Processing Reflect Loss of Plasticity in Maturing Systems: Insights from Connectionist Networks. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 26(5):1103.

Elsayed, M. and Mahmood, A. R. (2024). Addressing Catastrophic Forgetting and Loss of Plasticity in Neural Networks. In *12th International Conference on Learning Representations*.

Eriksson, P. S., Perfilieva, E., Björk-Eriksson, T., Alborn, A.-M., Nordborg, C., Peterson, D. A., and Gage, F. H. (1998). Neurogenesis in the Adult Human Hippocampus. *Nature Medicine*, 4(11):1313–1317.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. (2020). Rigging the Lottery: Making All Tickets Winners. In *37th International Conference on Machine Learning*.

Farias, V. and Jozefiak, A. D. (2025). Self-Normalized Resets for Plasticity in Continual Learning. In *13th International Conference on Learning Representations*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *34th International Conference on Machine Learning*.

Frankle, J. and Carbin, M. (2019). The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *7th International Conference on Learning Representations*.

French, R. M. (1999). Catastrophic Forgetting in Connectionist Networks. *Trends in Cognitive Sciences*, 3(4):128–135.

Fukushima, K. (1975). Cognitron: A Self-organizing Multilayered Neural Network. *Biological Cybernetics*, 20(3):121–136.

Fusi, S., Drew, P. J., and Abbott, L. F. (2005). Cascade Models of Synaptically Stored Memories. *Neuron*, 45(4):599–611.

Galashov, A., Titsias, M., György, A., Lyle, C., Pascanu, R., Teh, Y. W., and Sahani, M. (2024). Non-Stationary Learning of Neural Networks with Automatic Soft Parameter Reset. *Advances in Neural Information Processing Systems 38*.

Gale, T., Elsen, E., and Hooker, S. (2019). The State of Sparsity in Deep Neural Networks. Preprint at https://arxiv.org/abs/1902.09574.

Glorot, X. and Bengio, Y. (2010). Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *13th International Conference on Artificial Intelligence and Statistics*.

Golkar, S., Kagan, M., and Cho, K. (2019). Continual Learning via Neural Pruning. In *Real Neurons & Hidden Units: Future Directions at the Intersection of Neuroscience and Artificial Intelligence, Workshop at Advances in Neural Information Processing Systems 33*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I., Mirza, M., Xiao, D., and Aaron Courville, Y. B. (2014). An Empirical Investigation of Catastrophic Forgeting in Gradient-Based Neural Networks. In *2nd International Conference on Learning Representations*.

Graesser, L., Evci, U., Elsen, E., and Castro, P. S. (2022). The State of Sparse Training in Deep Reinforcement Learning. In *39th International Conference on Machine Learning*.

Han, S., Huizi, M., and Dally, W. J. (2016). Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-level Performance on ImageNet Classification. In *IEEE International Conference on Computer Vision*.

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

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep Reinforcement Learning That Matters. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

Hernandez-Garcia, J. F., Dohare, S., Luo, J., and Sutton, R. S. (2025). Reinitializing Weights Vs Units for Maintaining Plasticity in Neural Networks. In *Conference on Lifelong Learning Agents*.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving Neural Networks by Preventing Co-adaptation of Feature Detectors. Preprint at https://arxiv.org/abs/1207.0580.

Hofmann, M., Becker, M. F. P., Tetzlaff, C., and Mäder, P. (2025). Concept Transfer of Synaptic Diversity from Biological to Artificial Neural Networks. *Nature Communications*, 16(1):1–16.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. MIT Press.

Holland, J. H. and Reitman, J. S. (1977). Cognitive Systems Based on Adaptive Algorithms. *ACM Sigart Bulletin*, (63):49–49.

Householder, A. S. (1941). A Theory of Steady-state Activity in Nerve-fiber Networks: I. Definitions and Preliminary Lemmas. *The Bulletin of Mathematical Biophysics*, 3:63–69.

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-Excitation Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Igl, M., Farquhar, G., Luketina, J., Boehmer, W., and Whiteson, S. (2021). Transient Non-stationarity and Generalisation in Deep Reinforcement Learning. In *9th International Conference on Learning Representations*.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *32nd International Conference on Machine Learning*.

Javed, K. (2025). *Real-time Reinforcement Learning for Achieving Goals in Big Worlds*. PhD thesis, University of Alberta.

Javed, K. and White, M. (2019). Meta-Learning Representations for Continual Learning. In *Advances in Neural Information Processing Systems 33*.

Jin, C., Ge, R., Netrapalli, P., Kakade, S. M., and Jordan, M. I. (2017). How to Escape Saddle Points Efficiently. In *34th International Conference on Machine Learning*.

Juliani, A. and Ash, J. T. (2024). A Study of Plasticity Loss in On-Policy Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems 38*.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. MIT Press.

Kasai, H., Ziv, N. E., Okazaki, H., Yagishita, S., and Toyoizumi, T. (2021). Spine Dynamics in the Brain, Mental Disorders and Artificial Neural Networks. *Nature Reviews Neuroscience*, 22(7):407–422.

Kashyap, R. L., Blaydon, C. C., and Fu, K. S. (1970). Stochastic Approximation. *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, 66:329–356.

Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*.

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

Klopf, A. H. and Gose, E. (1969). An Evolutionary Pattern Recognition Network. *IEEE Transactions on Systems Science and Cybernetics*, 5(3):247–250.

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

Kumar, A., Agarwal, R., Ghosh, D., and Levine, S. (2021). Implicit Under-Parameterization Inhibits Data-Efficient Deep Reinforcement Learning. In *9th International Conference on Learning Representations*.

Kumar, S., Marklund, H., Rao, A., Zhu, Y., Jun Jeon, H., Yueyang, L., and Van Roy, B. (2025). Continual learning as computationally constrained reinforcement learning. *Foundations and Trends in Machine Learning*, 18(5):913–1053.

Kumar, S., Marklund, H., and Van Roy, B. (2024). Maintaining Plasticity in Continual Learning via Regenerative Regularization. In *Conference on Lifelong Learning Agents*.

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

LeCun, Y., Denker, J., and Solla, S. (1989). Optimal Brain Damage. In *Advances in Neural Information Processing Systems 2*.

Lee, H., Cho, H., Kim, H., Gwak, D., Kim, J., Choo, J., Yun, S.-Y., and Yun, C. (2023). PLASTIC: Improving Input and Label Plasticity for Sample Efficient Reinforcement Learning. In *Advances in Neural Information Processing Systems 37*.

Lee, H., Cho, H., Kim, H., Kim, D., Min, D., Choo, J., and Lyle, C. (2024). Slow and Steady Wins the Race: Maintaining Plasticity with Hare and Tortoise Networks. In *41st International Conference on Machine Learning*.

Lewandowski, A., Bortkiewicz, M., Kumar, S., György, A., Schuurmans, D., Ostaszewski, M., and Machado, M. C. (2025a). Learning Continually by Spectral Regularization. In *13th International Conference on Learning Representations*.

Lewandowski, A., Schuurmans, D., and Machado, M. C. (2025b). Plastic Learning with Deep Fourier Features. In *13th International Conference on Learning Representations*.

Lewandowski, A., Tanaka, H., Schuurmans, D., and Machado, M. C. (2023). Curvature Explains Loss of Plasticity. Preprint at https://arxiv.org/abs/2312.00246.

Liu, J., Wu, Z., Obando-Ceron, J., Castro, P. S., Courville, A., and Pan, L. (2025). Measure Gradients, Not Activations! Enhancing Neuronal Activity in Deep Reinforcement Learning. Preprint at https://arxiv.org/abs/2505.24061.

Liu, J., Xu, Z., Shi, R., Cheung, R. C. C., and So, H. K. (2020). Dynamic Sparse Training: Find Efficient Sparse Network From Scratch With Trainable Masked Layers. In *8th International Conference on Learning Representations*.

Lu, L., Shin, Y., Su, Y., and Karniadakis, G. E. (2019). Dying ReLU and Initialization: Theory and Numerical Examples. Preprint at https://arxiv.org/abs/1903.06733.

Luo, Y., Yang, Z., Meng, F., Li, Y., Zhou, J., and Zhang, Y. (2023). An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-tuning. Preprint at https://arxiv.org/abs/2308.08747.

Lyle, C., Rowland, M., and Dabney, W. (2022). Understanding and Preventing Capacity Loss in Reinforcement Learning. In *10th International Conference on Learning Representations*.

Lyle, C., Zheng, Z., Khetarpal, K., Martens, J., van Hasselt, H. P., Pascanu, R., and Dabney, W. (2024a). Normalization and Effective Learning Rates in Reinforcement Learning. In *Advances in Neural Information Processing Systems 37*.

Lyle, C., Zheng, Z., Khetarpal, K., van Hasselt, H., Pascanu, R., Martens, J., and Dabney, W. (2024b). Disentangling the Causes of Plasticity Loss in Neural Networks. In *Conference on Lifelong Learning Agents*.

Lyle, C., Zheng, Z., Nikishin, E., Avila Pires, B., Pascanu, R., and Dabney, W. (2023). Understanding Plasticity in Neural Networks. In *Proceedings of the 40th International Conference on Machine Learning*.

Mahmood, A. (2017). *Incremental Off-policy Reinforcement Learning Algorithms*. PhD thesis, University of Alberta.

Mahmood, A. R. and Sutton, R. S. (2013). Representation Search through Generate and Test. In *AAAI Workshop: Learning Rich Representations from Low-Level Sensors, Volume 10*.

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

McCulloch, W. S. and Pitts, W. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

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

Moalla, S., Miele, A., Pyatko, D., Pascanu, R., and Gulcehre, C. (2024). No Representation, No Trust: Connecting Representation, Collapse, and Trust Issues in PPO. In *Advances in Neural Information Processing Systems 38*.

Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. (2018). Scalable Training of Artificial Neural Networks with Adaptive Sparse Connectivity Inspired by Network Science. *Nature Communications*, 9(1):1–12.

Mucciardi, A. N. and Gose, E. E. (1966). Evolutionary Pattern Recognition in Incomplete Nonlinear Multithreshold Networks. *IEEE Transactions on Electronic Computers*, EC-15(2):257–261.

Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. (2019). Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning. In *7th International Conference on Learning Representations*.

Nikishin, E., Oh, J., Ostrovski, G., Lyle, C., Pascanu, R., Dabney, W., and Barreto, A. (2023). Deep Reinforcement Learning with Plasticity Injection. In *Advances in Neural Information Processing Systems 37*.

Nikishin, E., Schwarzer, M., D'Oro, P., Bacon, P.-L., and Courville, A. (2022). The Primacy Bias in Deep Reinforcement Learning. In *39th International Conference on Machine Learning*.

OpenAI (2023). GPT-4 Technical Report. Preprint at https://arxiv.org/abs/2303.08774.

OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Dkebiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with Large Scale Deep Reinforcement Learning. Preprint at https://arxiv.org/abs/1912.06680.

Orvieto, A. and Gower, R. (2025). In Search of Adam's Secret Sauce. Preprint at https://arxiv.org/abs/2505.21829.

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

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 32.

Patterson, A., Neumann, S., White, M., and White, A. (2024). Empirical Design in Reinforcement Learning. *Journal of Machine Learning Research*, 25(318):1–63.

Powell, M. J. D. (1977). Restart Procedures for the Conjugate Gradient Method. *Mathematical Programming*, 12:241–254.

Rajasegaran, J., Hayat, M., Khan, S. H., Khan, F. S., and Shao, L. (2019). Random Path Selection for Continual Learning. In *Advances in Neural Information Processing Systems 33*.

Rakitianskaia, A. and Engelbrecht, A. (2015). Measuring Saturation in Neural Networks. In *IEEE Symposium Series on Computational Intelligence*, pages 1423–1430.

Razin, N. and Cohen, N. (2020). Implicit Regularization in Deep Learning May Not Be Explainable by Norms. In *Advances in Neural Information Processing Systems 34*.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. (2017). iCaRL: Incremental Classifier and Representation Learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Riemer, M., Cases, I., Ajemian, R., Liu, M., Rish, I., Tu, Y., , and Tesauro, G. (2019). Learning to Learn Without Forgetting By Maximizing Transfer and Minimizing Interference. In *7th International Conference on Learning Representations*.

Ring, M. B. (1998). CHILD: A First Step Towards Continual Learning. In *Learning to Learn*, pages 261–292. Springer.

Roy, O. and Vetterli, M. (2007). The Effective Rank: A Measure of Effective Dimensionality. In *15th European Signal Processing Conference*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive Neural Networks. Preprint at https://arxiv.org/abs/1606.04671.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized Experience Replay. In *4th International Conference on Learning Representations*.

Schlegel, M. K., Tkachuk, V., White, A. M., and White, M. (2023). Investigating Action Encodings in Recurrent Neural Networks in Reinforcement Learning. *Transactions on Machine Learning Research*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *4th International Conference on Learning Representations*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. Preprint at https://arxiv.org/abs/1707.06347.

Schwarzer, M., Obando Ceron, J. S., Courville, A., Bellemare, M. G., Agarwal, R., and Castro, P. S. (2023). Bigger, Better, Faster: Human-level Atari with Human-level Efficiency. In *40th International Conference on Machine Learning*.

Seidenberg, M. S. and McClelland, J. L. (1989). A Distributed, Developmental Model of Word Recognition and Naming. *Psychological Review*, 96(4):523.

Selfridge, O. G. (1958). Pandemonium: A Paradigm for Learning. *Mechanization of Thought Processes: Proceedings of a Symposium Held at the National Physical Laboratory*, page 511–531.

Shah, I., Polloreno, A. M., Stratos, K., Monk, P., Chaluvaraju, A., Hojel, A., Ma, A., Thomas, A., Tanwer, A., Shah, D. J., et al. (2025). Practical Efficiency of Muon for Pretraining. Preprint at https://arxiv.org/abs/2505.02222.

Shin, Y. and Karniadakis, G. E. (2020). Trainability of ReLU Networks and Data-dependent Initialization. *Journal of Machine Learning for Modeling and Computing*, 1(1):39–74.

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., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529(7587):484–489.

Smith, S. L., Dherin, B., Barrett, D., and De, S. (2021). On the Origin of Implicit Regularization in Stochastic Gradient Descent. In *9th International Conference on Learning Representations*.

Sokar, G., Agarwal, R., Castro, P. S., and Evci, U. (2023). The Dormant Neuron Phenomenon in Deep Reinforcement Learning. In *Proceedings of the 40th International Conference on Machine Learning*.

Sokar, G., Mocanu, E., Mocanu, D. C., Pechenizkiy, M., and Stone, P. (2022). Dynamic Sparse Training for Deep Reinforcement Learning. *The 31st International Joint Conference on Artificial Intelligence*.

Springer, J. M., Goyal, S., Wen, K., Kumar, T., Yue, X., Malladi, S., Neubig, G., and Raghunathan, A. (2025). Overtrained Language Models Are Harder to Fine-Tune. Preprint at https://arxiv.org/abs/2503.19206.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the Importance of Initialization and Momentum in Deep Learning. In *30th International Conference on Machine Learning*.

Sutton, R. (2019). The Bitter Lesson. Incomplete Ideas (Blog) at http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind Control Suite. Preprint at https://arxiv.org/abs/1801.00690.

Thorndike, E. L. (1911). *Animal Intelligence*. The Macmillan Company.

van de Ven, G. M., Tuytelaars, T., and Tolias, A. S. (2022). Three Types of Incremental Learning. *Nature Machine Intelligence*, 4(12):1185–1197.

Veniat, T., Denoyer, L., and Ranzato, M. (2021). Efficient Continual Learning with Modular Networks and Task-Driven Priors. In *9th International Conference on Learning Representations*.

Verwimp, E., Aljundi, R., Ben-David, S., Bethge, M., Cossu, A., Gepperth, A., Hayes, T. L., Hüllermeier, E., Kanan, C., Kudithipudi, D., Lampert, C. H., Mundt, M., Pascanu, R., Popescu, A., Tolias, A. S., van de Weijer, J., Liu, B., Lomonaco, V., Tuytelaars, T., and van de Ven, G. M. (2024). Continual Learning: Applications and the Road Forward. *Transactions on Machine Learning Research*.

Verwimp, E., Hacohen, G., and Tuytelaars, T. (2025). Same Accuracy, Twice As Fast: Continual Learning Surpasses Retraining From Scratch. Preprint at https://arxiv.org/abs/2502.21147.

Wang, Y.-X., Ramanan, D., and Hebert, M. (2017). Growing a Brain: Fine-tuning by Increasing Model Capacity. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Whiteson, S. and Stone, P. (2006). Evolutionary Function Approximation for Reinforcement Learning. *Journal of Machine Learning Research*, 7(31):877–917.

Yang, Y., Zhang, G., Xu, Z., and Katabi, D. (2019). Harnessing Structures for Value-based Planning and Reinforcement Learning. In *7th International Conference on Learning Representations*.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. (2018). Lifelong Learning with Dynamically Expandable Networks. In *6th International Conference on Learning Representations*.

Zang, Z., Sun, C., Liu, L., Sun, F., and Zheng, C. (2025). Loss of Plasticity: A New Perspective on Solving Multi-Agent Exploration for Sparse Reward Tasks. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual Learning Through Synaptic Intelligence. In *34th International Conference on Machine Learning*.

Zevin, J. D. and Seidenberg, M. S. (2002). Age of Acquisition Effects in Word Reading and Other Tasks. *Journal of Memory and Language*, 47(1):1–29.

Zhang, D., Qi, S., Xiao, X., Chen, K., and Wang, X. (2025). Merge Then Realign: Simple and Effective Modality-Incremental Continual Learning for Multimodal LLMs. Preprint at https://arxiv.org/abs/2503.07663.

Zhao, R., Morwani, D., Brandfonbrener, D., Vyas, N., and Kakade, S. M. (2025). Deconstructing What Makes a Good Optimizer for Autoregressive Language Models. In *13th International Conference on Learning Representations*.

Zhou, G., Sohn, K., and Lee, H. (2012). Online Incremental Feature Learning with Denoising Autoencoders. In *Artificial Intelligence and Statistics*.

Ziv, N. E. and Brenner, N. (2018). Synaptic Tenacity or Lack Thereof: Spontaneous Remodeling of Synapses. *Trends in Neurosciences*, 41(2):89–99.

Appendix A: Network Architectures

Table A.1: Network architecture used for the Continual ImageNet problem. The network consists of three convolutional layers followed by three fully connected layers.

Layer 1: Convolutional + Max-Pooling			
Number of Filters	32	Activation	ReLU
Convolutional Filter Shape	(5,5)	Convolutional Filter Stride	(1,1)
Max-Pooling Filter Shape	(2,2)	Max-Pooling Filter Stride	(1,1)
Layer 2: Convolutional + Max-Pooling			
Number of Filters	64	Activation	ReLU
Convolutional Filter Shape	(3,3)	Convolutional Filter Stride	(1,1)
Max-Pooling Filter Shape	(2,2)	Max-Pooling Filter Stride	(1,1)
Layer 3: Convolutional + Max-Pooling			
Number of Filters	128	Activation	ReLU
Convolutional Filter Shape	(3,3)	Convolutional Filter Stride	(1,1)
Max-Pooling Filter Shape	(2,2)	Max-Pooling Filter Stride	(1,1)
Layer 4: Fully Connected			
Output Size	128	Activation	ReLU
Layer 5: Fully Connected			
Output Size	128	Activation	ReLU
Layer 6: Fully Connected			
Output Size	2	Activation	Linear

Table A.2: Details of the 18 layer residual network used for the Class Incremental CIFAR-100 problem. Convolutional layers used a kernel size of (3,3), reshape layers used a kernel size of (1,1), and the pool layer used a kernel size of (4,4).

Layer Name	Layer Type	Layer Parameters	Receives Inputs From
Input Layer	-	Input Shape: (3, 32, 32)	-
Conv 1	Conv + Batch Norm + ReLU	Out Filters: 64, Stride: 1	Input Layer
Conv 2	Conv + Batch Norm + ReLU	Out Filters: 64, Stride: 1	Conv 1
Conv 3	Conv + Batch Norm	Out Filters: 64, Stride: 1	Conv 2
Act 3	ReLU Activation	-	Conv 3 + Conv 1
Conv 4	Conv + Batch Norm + ReLU	Out Filters: 64, Stride: 1	Act 3
Conv 5	Conv + Batch Norm	Out Filters: 64, Stride: 1	Conv 4
Act 5	ReLU Activation	-	Conv 5 + Act 3
Downsample 6	Conv + Batch Norm	Out Filters: 128, Stride: 2	Act 5
Conv 6	Conv + Batch Norm + ReLU	Out Filters: 128, Stride: 2	Act 5
Conv 7	Conv + Batch Norm	Out Filters: 128, Stride 1	Conv 6
Act 7	ReLU Activation	-	Conv 7 + Downsample 6
Conv 8	Conv + Batch Norm + ReLU	Out Filters: 128, Stride 1	Act 7
Conv 9	Conv + Batch Norm	Out Filters: 128, Stride 1	Conv 8
Act 9	ReLU Activation	-	Conv 9 + Act 7
Downsample 10	Conv + Batch Norm	Out Filters: 256, Stride: 2	Act 9
Conv 10	Conv + Batch Norm + ReLU	Out Filters: 256, Stride: 2	Act 9
Conv 11	Conv + Batch Norm	Out Filters: 256, Stride: 1	Conv 10
Act 11	ReLU Activation	-	Conv 11 + Downsample 10
Conv 12	Conv + Batch Norm + ReLU	Out Filters: 256, Stride: 1	Act 11
Conv 13	Conv + Batch Norm	Out Filters: 256, Stride: 1	Conv 12
Act 13	ReLU Activation	-	Conv 13 + Act 11
Downsample 14	Conv + Batch Norm	Out Filters: 512, Stride: 2	Act 13
Conv 14	Conv + Batch Norm + ReLU	Out Filters: 512, Stride: 2	Act 13
Conv 15	Conv + Batch Norm	Out Filters: 512, Stride: 1	Conv 14
Act 15	ReLU Activation	-	Conv 15 + Downsample 14
Conv 16	Conv + Batch Norm + ReLU	Out Filters: 512, Stride: 1	Act 15
Conv 17	Conv + Batch Norm	Out Filters: 512, Stride: 1	Conv 16
Act 17	ReLU Activation	-	Conv 17 + Act 15
Pooling Layer	Average Pooling + Flatten Layer	Output Size: 512	Act 17
Output Layer	Linear	Output Size: 10	Pooling Layer

Appendix B: Hyperparameters

Table B.1: Hyperparameter in Continual ImageNet. Values used for the grid searches to find the best set of hyper-parameters for all algorithms tested on Continual ImageNet. The best-performing set of values for each algorithm is boldened.

Algorithm Name	Stepsize	Weight Decay / Replacement Rate	Noise Variance
L2-Regularization	0.1, **0.03**, 0.01, 0.003	$3*10^{-5}$, 10^{-5}, $3*10^{-6}$, 10^{-6}	-
Shrink and Perturb	0.03, **0.01**	$3*10^{-5}$, 10^{-5}, $3*10^{-6}$, 10^{-6}	10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}
Continual Backpropagation	0.03, **0.01**	$3*10^{-3}$, 10^{-3}, $3*10^{-4}$, $1*10^{-4}$, $3*10^{-5}$	-

Table B.2: Hyper-parameters for PPO.

Name	Default Value
Policy network	(256, ReLU, 256, ReLU, Linear) + Standard deviation variable
Value network	(256, ReLU, 256, ReLU, Linear)
Buffer size	2048
Number of epochs	10
Mini-batch size	128
GAE, λ	0.95
Discount factor, γ	0.99
Clip parameter	0.2
Input normalization	False
Advantage normalization	True
Value function loss clipping	False
Gradient clipping	False
Optimizer	Adam
Optimizer step size	10^{-4}
Optimizer ε	10^{-8}