Safety-Oriented Stability Biases for Continual Learning

by

Ashish Gaurav

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2020

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Continual learning is often confounded by “catastrophic forgetting” that prevents neural networks from learning tasks sequentially. In the case of real world classification systems that are safety-validated prior to deployment, it is essential to ensure that validated knowledge is retained. We propose methods that build on existing unconstrained continual learning solutions, which increase the model variance or weaken the model bias to better retain more of the existing knowledge.

We investigate multiple such strategies, both for continual classification as well as continual reinforcement learning. Finally, we demonstrate the improved performance of our methods against popular continual learning approaches, using variants of standard image classification datasets, as well as assess the effect of weaker biases in continual reinforcement learning.

Acknowledgements

I would like to thank my supervisor, Prof. Krzysztof Czarnecki, for providing me with invaluable guidance and continued support throughout the program.

I wish to thank Sean Sedwards for his support, feedback and advice throughout the last year.

I would also like to thank Jaeyoung Lee, Vahdat Abdelzad, Sachin Vernekar, Aravind Balakrishnan and my colleagues at Waterloo Intelligent System Engineering Lab (WISE Lab) for their support and feedback.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Machine learning using neural networks has achieved considerable success in applications such as image recognition, game-playing, content recommendation and health-care [30]. Most of these applications require large amounts of training data and careful selection of architecture and parameters. More importantly, the learned systems often have to adapt to changing real-world requirements, and therefore require re-training. Under these circumstances, it is usually desired to retain performance on previous tasks while learning to perform well on new tasks. This is what constitutes continual learning [41].

Any strategy used for continual learning has to balance *plasticity* (the ability to learn new tasks) and *stability* (the ability to remember previous tasks). This is the well discussed stability-plasticity dilemma [48]. This dilemma can be explained in terms of the bias-variance tradeoff, another well known concept in statistical learning [11]. In this context, (model) variance characterizes the span of solutions that can be realized with a neural network. Adding bias reduces the variance of the model and can produce a better solution than the unbiased case [12]. With no bias, the continual learning model is plastic (has high variance); with strong bias, the model is stiff with respect to learning new tasks (has low variance). In this work we investigate ways to improve the retention of previously learned tasks by increasing the model variance.

We note that performance in previous work is often judged with respect to average validation accuracy across tasks. While good average validation accuracy is the most common metric to judge forgetting, our principal concern is safety. In safety-critical systems, it may not be acceptable to maintain an average validation accuracy at the cost of trading previously certified decisions for possibly correct decisions that have not been checked. Likewise, the calibration of a system may require that all classifier predictions, good or bad, remain the same. Therefore, we are motivated by the need to achieve alternative continual learning solutions that prioritize stability over plasticity. In the present work, we achieve this by considering exactly what has been forgotten and what has been learned.

This thesis is structured as follows. In Chapter 2, we provide the necessary background for the rest of the work. In Chapter 3, we summarize the continual learning approaches in existing literature and other related work. In Chapter 4, we propose simple continual learning strategies for classification, motivated by safety and the need for stability over plasticity. In Chapter 5, we extend the principle of increasing model variance to propose a continual reinforcement learning strategy. In Chapter 6, we evaluate the proposed methods on various datasets and a curriculum of RL environments. Finally in Chapter 7, we summarize and conclude the major points from the thesis.

Chapter 2

Mathematical Background

In this chapter, we define the notions of classification, reinforcement learning and continual learning as used in the rest of this work. We assume some familiarity with the basic concepts of deep neural networks and reinforcement learning. For additional reading on deep learning, we refer the reader to Goodfellow *et al.* [13]. For additional reading on reinforcement learning, we refer the reader the Sutton *et al.* [64].

2.1 Notation

Some basic notation used throughout the rest of this work is as follows.

We denote by $|\mathbf{x}|$ a vector with the same dimensions as the vector $\mathbf{x}$, such that each element in $|\mathbf{x}|$ is the absolute of the corresponding element in $\mathbf{x}$. By $\mathbf{x} \cdot \mathbf{y}$ we denote the inner product of two vectors $\mathbf{x}$ and $\mathbf{y}$. $|\mathbf{x}|^2$ denotes the element-wise square of vector $\mathbf{x}$, that is, $|\mathbf{x}|^2 := \mathbf{x} \odot \mathbf{x}$, with $\odot$ referring to the Hadamard product. $\|\mathbf{x}\|_1$ and $\|\mathbf{x}\|_2$ denote the standard L_1 and L_2 norms (scalars) of vector $\mathbf{x}$.

For a matrix $\mathbf{M}$, the operation $\mathbf{diag}(\mathbf{M})$ produces a vector that consists of the leading diagonal of $\mathbf{M}$. For a scalar x, $\text{sign}(x)$ represents the signum function, which produces 1 if $x > 0$, 0 if $x = 0$ and -1 otherwise.

We denote by $1 : i$ the sequence of indices $1, 2, \ldots, i$. The vector of weights $\boldsymbol{\theta}$ after training tasks $1 : i$ is denoted $\boldsymbol{\theta}^*_{1:i}$. The j-th element of $\boldsymbol{\theta}^*_{1:i}$ is denoted $\theta^*_{1:i,j}$.

2.2 Classification

Let a dataset be defined as a set of training examples $\mathbf{P}$, training labels $\mathbf{Q}$, validation examples $\mathbf{X}$ and validation labels $\mathbf{Y}$:

$$\mathcal{D} := (\mathbf{P}, \mathbf{Q}, \mathbf{X}, \mathbf{Y})$$

The examples in the dataset belong to one of M classes of data. By design, the validation examples and labels are never to be seen during the training. In classification, the goal is to produce a discriminative model that can best categorize unseen data for a given dataset to one of M classes (that is, maximize validation accuracy).

Deep learning provides a powerful method to construct such discriminative models. The typical setup involves using a ReLU feedforward network (*neural network*) with weights $\boldsymbol{\theta}$, where the final layer is followed by softmax instead of ReLU. Given an input x, such a network outputs a set of positive numbers ≤ 1 that approximate how likely x belongs to class m:

$$\{P_{\boldsymbol{\theta}}(y = m|x)\}_{m=1}^{M} \tag{2.1}$$

The softmax function ensures that the values sum to 1. For notational simplicity, we denote $P_{\boldsymbol{\theta}}(y = m|x)$ as $P_{\boldsymbol{\theta}}^{m}(\cdot|x)$. If the ground truth for x is $y = g$, where $(1 \leq g \leq M)$, then we use the shorthand $P_{\boldsymbol{\theta}}(\cdot|x) := P_{\boldsymbol{\theta}}^{g}(\cdot|x)$ for the predicted likelihood of label g. Note that $P_{\boldsymbol{\theta}}(y|x)$ is the output of a learned neural network which takes as input x. $P_{\boldsymbol{\theta}}(y|x)$ can be *interpreted* as the conditional probability of y given some x, but it is different from the actual conditional probability of an event y given some event x, which is denoted as $p(y|x)$.

The optimization procedure typically starts with a neural network having random weights, then changes these weights to eventually arrive at the desired classifier. To change these weights, most optimization procedures use some type of gradient descent to minimize a pre-defined loss over the training examples and labels.

Various loss functions have been previously proposed to train a neural network for classification [20]. In this work we use the cross entropy loss *aka* the negative log likelihood loss, $\mathcal{L}(\boldsymbol{\theta})$, defined over an example (x, y) as:

$$\mathcal{L}(\boldsymbol{\theta}) = -\log P_{\boldsymbol{\theta}}(\cdot|x) \tag{2.2}$$

2.3 Continual Learning for Classification

Let there be n datasets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n$ such that dataset $\mathcal{D}_i$ has K_i validation examples:

$$\mathcal{D}_i = (\mathbf{P}_i, \mathbf{Q}_i, \mathbf{X}_i, \mathbf{Y}_i) = (\mathbf{P}_i, \mathbf{Q}_i, \{x_i^{(k)}\}_{k=1}^{K_i}, \{y_i^{(k)}\}_{k=1}^{K_i}) \tag{2.3}$$

Continual learning seeks to maximize performances on all these datasets. Most commonly, this is achieved by *sequential training* on the datasets, that is, training on datasets $1, 2, \cdots, n$, in order. For any task i, the weights achieved at the end of task i ($\boldsymbol{\theta}^*_{1:i}$) should thus retain performances on tasks $1, 2, \ldots, i-1$. Therefore, notionally, $\boldsymbol{\theta}^*_{1:i}$ should minimize the cross entropy loss over datasets $\mathcal{D}_{1:i} := \mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_i$.

The continual learning objective can also be alternatively achieved by training on examples from all relevant datasets simultaneously (*joint training*). Joint training quickly becomes expensive as the number of tasks grow, but typically has the best performance across all tasks [34].

2.4 Reinforcement Learning (RL)

In reinforcement learning, an agent learns to behave in a given environment through trial-and-error. In our setting, the environment and the reward represent a Markov Decision Process $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ represents the transition probability for the transition $s \xrightarrow{a} s'$ $(s, s' \in \mathcal{S}, a \in \mathcal{A})$, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ represents the reward for the transition $s \xrightarrow{a} s'$, and $\gamma \in [0, 1]$ represents the discount factor.

$$\mathcal{T}(s, a, s') := p(s \xrightarrow{a} s') = p(s'|s, a)$$

The agent interacts with the environment in episodes. Each episode starts with an initial random state s_0. The agent observes this state and chooses an action a_0, thereby receiving a reward r_0 and transitioning into a new state s_1. After observing s_1, the agent chooses an action a_1, receives a reward r_1 and this process continues until the agent reaches a terminal state s_T. An episode can be defined as a sequence of transitions:

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \cdots s_T$$

If we assume that the agent follows a stochastic (control) policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, such that the probability of choosing an action a in state s is given by $\pi(a|s)$, then the objective of reinforcement learning is to find a policy π^* (in the family of policies Π) that maximizes the expected (infinite-horizon) long term episodic discounted reward:

$$\pi^* = \arg\max_{\pi \in \Pi} \ \mathbb{E}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim \mathcal{T}(s_t, a_t, \cdot)}} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \tag{2.4}$$

Here, the expectation is over the episodes sampled using the stochastic policy $\pi(a|s)$ and the transition function $\mathcal{T}(s, a, s')$. If the episode terminates at state s_T, then $r_T = r_{T+1} = r_{T+2} = \cdots = 0$.

There are multiple strategies to achieve the objective in (2.4). Algorithms can be *model-based*, where they learn and utilize the transition model T in action selection, or *model-free*, where a transition model is not explicitly learned. Model-based methods are known to be more sample efficient, while model-free methods are more practical for large state and action spaces.

Reinforcement learning with neural networks (*deep reinforcement learning*) usually involves learning a good parameterized approximation of certain functions which can lead to good policies. Methods like DQN [45] and Double DQN [67] focus on learning the action value function $Q_{\boldsymbol{\theta}}(s, a)$. Other methods like Dueling DQN [68], DRQN [16] and Rainbow DQN [17] use alternative network architectures and learn either $Q_{\boldsymbol{\theta}}(s, a)$, or the value estimate $V_{\boldsymbol{\theta}}(s)$, or both. Both the action value estimate and the value estimate can be trained through the Bellman loss [63].

An alternative strategy is to learn a parameterized approximation of the policy ($\pi_{\boldsymbol{\theta}}$). Policies can be updated through the policy gradient theorem, which provides a meaningful update for $\pi_{\boldsymbol{\theta}}$ by optimizing (2.4) ($\equiv \mathcal{J}(\boldsymbol{\theta})$):

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \right] \tag{2.5}$$

Here Ψ_t represents the generalized advantage function at time t in the episode. The advantage function is desired to be both low variance and low bias (controlled by λ), and therefore constructing the advantage function is an active area of research [56]. In our setting, Ψ_t is constructed as follows:

$$\hat{A}_t^{(n)} := \{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n V(s_{t+n})\} - V(s_t)$$
$$\Psi_t := (1-\lambda)\big(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \cdots\big)$$

Methods like DDPG [35] and TD3 [9] combine policy learning with learning value estimates. The learned policy acts after observing a state (behaving like an *actor*) and the value function is used in updating the policy (*critic*), and therefore such methods are called actor-critic methods.

Actor-critic style methods have recently been adapted to optimize a *surrogate* objective, instead of directly optimizing (2.4) which leads to the policy gradient theorem. Methods like TRPO [55], PPO [57], and ACKTR [70] are based on this principle.

This thesis builds on PPO, which is a well known, efficient and easy-to-implement actor-critic style algorithm that optimizes the surrogate objective. The surrogate objective was first introduced in TRPO, also called the conservative policy iteration objective (CPI). It can be obtained by changing the policy gradient update in (2.5) to use importance sampling. Basically, many trajectories are sampled using the current policy, the advantages Ψ_t are computed for states s_t in these trajectories and kept in a memory. Then, at a later stage, the policy update is constructed by estimating the gradient of the CPI objective. The CPI objective is as follows:

$$L^{CPI}(\boldsymbol{\theta}) = \mathbb{E}_{s_t}\left[\Psi_t \cdot \frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_o}(a_t|s_t)}\right] \equiv \mathbb{E}_t[\Psi_t \cdot r_t(\boldsymbol{\theta})] \tag{2.6}$$

Here Ψ_t and $\pi_{\boldsymbol{\theta}_o}(a_t|s_t)$ are sampled from the memory, and the latest policy $\pi_{\boldsymbol{\theta}}(a_t|s_t)$ is used to compute the newest probability for action a_t in state s_t.

PPO further clips (2.6) to produce a clipped optimization objective. This is done by clipping the ratio $r_t(\boldsymbol{\theta})$ to be within $[1-\epsilon, 1+\epsilon]$:

$$L^{CLIP}(\boldsymbol{\theta}) = \mathbb{E}_t[\min(r_t(\boldsymbol{\theta})\Psi_t, \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\Psi_t)] \tag{2.7}$$

This clipping results in a stable optimization objective, unlike TRPO. The pseudocode of the version of PPO used in Chapter 5 is described in Algorithm 1.

In our work, the policy neural network models a categorical distribution over the action space. The probability of actions correspond to the softmax probabilities of the logit outputs of the policy network. During training, actions are chosen by sampling from the categorical distribution according to the softmax probabilities. This sampling leads to exploration of novel states, as is desired in an RL algorithm. During evaluation, the action suggested by the policy corresponds to the highest softmax probability.

Algorithm 1 PPO, as described in [57]

```
for iteration = 1, 2, ... do
    for actor = 1, 2, ..., $N$ (parallel) do
        Run policy $\pi_{\boldsymbol{\theta}_o}$ in environment for $T$ timesteps
        Compute advantage estimates $\Psi_t$, $1 \leq t \leq T$
    end for
    for epoch = 1, 2, ..., $K$ do
        Optimize surrogate $L^{CLIP}(\boldsymbol{\theta})$ on $M$ samples, $M \leq NT$
    end for
end for
```

2.5 Continual RL

Similar to the classification case, we assume that there are n environments such that each environment has a pre-specified reward function. Let the i-th environment be e_i, with a state space $\mathcal{S}_i$ and an action space $\mathcal{A}_i$.

The goal of continual reinforcement learning is to obtain a parameterized policy $\pi_{\boldsymbol{\theta}}$ such that $\pi_{\boldsymbol{\theta}}$ is able to solve all the environments. However, in our setting, the learning algorithm can only sequentially access the environments, that is, in the order $e_1, e_2, \cdots, e_n$. This order defines the *training schedule* of the continual RL algorithm.

Since a single policy is expected to learn state-action mappings for different environments, the parameterized policy $\pi_{\boldsymbol{\theta}}$ learns a mapping from $\mathcal{S} \times \mathcal{A}$ to $[0, 1]$ such that $\mathcal{S} = \cup_{i=1}^{n} \mathcal{S}_i$ and $\mathcal{A} = \mathcal{A}_i; \forall i$. We assume that the dimensionality of the state space stays the same between the environments.

Chapter 3

Related Work

In this chapter, we mention and categorize existing continual learning literature. In Section 3.1, we discuss the stability plasticity dilemma. Following this, we broadly categorize the continual learning approaches for classification into three approaches [48], that are discussed in Section 3.2, Section 3.3 and Section 3.4. Miscellaneous approaches are discussed in Section 3.5. Since this thesis is more closely related to regularization approaches, we additionally discuss some of these approaches in greater depth in Section 3.3.1 (Elastic Weight Consolidation) and Section 3.3.2 (Synaptic Intelligence). Finally, we discuss the continual reinforcement learning literature in Section 3.6.

3.1 The Stability-Plasticity Dilemma

Deep networks excel at individual tasks when large datasets are available and networks start from random configurations. However, when trained sequentially across different datasets, performance on the past tasks usually drops, which is otherwise known as catastrophic forgetting [40, 42]. Catastrophic forgetting has been extensively studied previously [8, 14].

There is typically a tradeoff between the ability of a feedforward network to learn new tasks (*plasticity*) and its ability to remember learned tasks (*stability*). This is the well known stability-plasticity dilemma, which has been studied in the context of biological as well as artificial networks [4, 43, 15].

Too much plasticity leads to catastrophic forgetting, but too much stability is also not desirable. This is the other side of the stability-plasticity dilemma, also called the *entrenchment effect*. The most probable reason for too much stability is that age-of-acquisition of data matters. For example, Ellis *et al.* [6] and Smith *et al.* [61] observed that when patterns are entered into training early, the network structures itself into a certain configuration and loses some of its plasticity. Even if the frequency of the late-learned pattern is greater, it is difficult for the network to overcome this loss in plasticity.

3.2 Architectural Approaches

Architectural approaches incrementally grow the network to learn the new task through the added capacity, that is, the untrained weight parameters. We discuss some popular architectural strategies as follows.

Zhou *et al.* [74] propose an incremental algorithm to learn features from large scale online data. Specifically, new nodes are added to the feature representation and similar features are merged when suitable.

Xiao *et al.* [71] propose a training algorithm that grows a network both incrementally and hierarchically, by grouping similar classes and organizing them into levels.

Progressive nets [54] instantiate a new column or parallel network to add capacity for learning a new task. This parallel network is connected to the original column(s) through a (non-linear) adapter function.

Yoon *et al.* [72] propose Dynamically Expandable Networks (DEN), which is an efficient online strategy that can learn multiple tasks compactly, by dynamically changing (increasing) the capacity of the network as needed. Learn to Grow [33] uses neural architectural search to grow the network for new tasks.

iCARL [51] is a method for class-incremental learning that stores exemplars per output class. For classification, it compares the input to mean of exemplars for each class. Further, exemplars are maintained through a selection algorithm, combined with distillation and rehearsal.

3.3 Regularization Approaches

Regularization approaches assume a fixed network architecture and regularize changes to crucial weights, so that the network can learn to perform well on the new task by changing less significant weights.

Less-forgetting learning [21], Learning without Forgetting (LwF) [34], Active Long Term Memory Networks (A-LTMs) [10] are all regularization strategies based on fine-tuning approaches, where a part of the network is kept common between the tasks. More specifically, LwF and A-LTMs use knowledge distillation [18] to preserve old knowledge and learn a separate head per task. When training on a new dataset, the network outputs for the other tasks (head outputs) are computed in advance, and an extra term is added to the training loss such that the head outputs don't change.

Another set of approaches consists of fixing the network structure and only changing the parameters that are not significant to the past configurations. Elastic weight consolidation [26, 19, 27] achieves this by formulating the CL problem from a Bayesian perspective and proposing a simple quadratic penalty that weighs the change in each parameter by its Fisher importance, thereby preventing the network from changing too much from previously learned configurations. EWC is further explained in Section 3.3.1. Synaptic Intelligence [73] uses a similar loss; however, it computes the weight for change in each parameter in an online manner. SI is further explained in Section 3.3.2. Liu *et al.* [36] propose improvements to EWC through network reparametrization. Chaudhry *et al.* [3] propose EWC++, which is a faster online version of EWC that combines the benefits of both EWC and SI.

3.3.1 Elastic Weight Consolidation (EWC)

In EWC [26, 19, 27], the continual learning problem is formulated from a Bayesian perspective. Learning a new task while retaining past performance is equivalent to maximizing a posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:i})$ given a prior $p(\boldsymbol{\theta}|\mathcal{D}_{1:i-1})$ such that the posterior and prior are as similar as possible, that is, the KL divergence between them is minimized. Here, the prior and the posterior can be thought of as the weight likelihood distributions before and after seeing the dataset $\mathcal{D}_i$. Through the optimization process, the network arrives at weights $\boldsymbol{\theta}^*$ which represents the maximum likelihood estimate of the weight distribution. For the prior and the posterior distributions, these estimates are $\boldsymbol{\theta}^*_{1:i-1}$ and $\boldsymbol{\theta}^*_{1:i}$. EWC arrives at $\boldsymbol{\theta}^*_{1:i}$ from $\boldsymbol{\theta}^*_{1:i-1}$ through simultaneous posterior maximization and KL minimization:

$$\begin{aligned}\boldsymbol{\theta}^*_{1:i} &= \min_{\boldsymbol{\theta}} \left\{ \mathcal{L}_i(\boldsymbol{\theta}) + \lambda \mathrm{KL}\big[\, p(\boldsymbol{\theta}|\mathcal{D}_{1:i-1}) \,||\, p(\boldsymbol{\theta}|\mathcal{D}_{1:i}) \,\big] \right\} \\ &\equiv (\approx) \min_{\boldsymbol{\theta}} \left\{ \mathcal{L}_i(\boldsymbol{\theta}) + \frac{\lambda}{2} \sum_j F_{jj} \cdot (\theta_j - \theta^*_{1:i-1,j})^2 \right\} \end{aligned} \tag{3.1}$$

Here, $\mathcal{L}_i(\boldsymbol{\theta})$ refers to the likelihood loss on the i-th dataset, and F_{jj} refers to the j-th diagonal element of the empirical Fisher information matrix $\mathbf{F}$, evaluated at $\boldsymbol{\theta}^*_{1:i-1}$ over the datasets $\mathcal{D}_{1:i-1}$. The Fisher information measures the sensitivities of the predicted probabilities for the correct labels with respect to the network parameters $\boldsymbol{\theta}$, and therefore allows identifying the weights that have the most influence on these predictions. The elements of $\mathbf{F}$ are defined as follows:

$$\begin{aligned} g_{ij} &:= \left\{ \frac{\partial}{\partial \theta_i} \log P_{\boldsymbol{\theta}}(\cdot|x) \right\} \cdot \left\{ \frac{\partial}{\partial \theta_j} \log P_{\boldsymbol{\theta}}(\cdot|x) \right\} \\ F_{ij} &:= \mathbb{E}_{(x,y)\sim\mathcal{D}_{1:i-1}} \left[g_{ij} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*_{1:i-1}} \right] \end{aligned}$$

This follows from the assumption that the difference between the two weight estimates is small, which allows for a second order Taylor approximation of the KL-divergence. More concretely, if we have seen a few more examples since $\mathcal{D}_{1:i-1}$ and the set of all seen examples is $\widetilde{\mathcal{D}}_{1:i-1}$, and our network's parameters changed by $\Delta\boldsymbol{\theta}$, then:

$$\begin{aligned} \mathrm{KL}\left[\, p(\boldsymbol{\theta}|\mathcal{D}_{1:i-1}) \,||\, p(\boldsymbol{\theta}|\widetilde{\mathcal{D}}_{1:i-1}) \,\right] &= \mathrm{KL}\left[\, P_{\boldsymbol{\theta}^*_{1:i-1}}(\cdot|x) \,||\, P_{\boldsymbol{\theta}^*_{1:i-1}+\Delta\boldsymbol{\theta}}(\cdot|x) \,\right] \\ &\approx \frac{1}{2} \Delta\boldsymbol{\theta}^T \mathbf{F}\, \Delta\boldsymbol{\theta} \\ &\approx \frac{1}{2} |\Delta\boldsymbol{\theta}|^2 \cdot \mathbf{diag}(\mathbf{F});\ \text{diagonal approximation} \end{aligned} \tag{3.2}$$

The interested reader may find a complete derivation of the relationship (3.2) between the KL divergence and the Fisher information in Kullback *et al.* [29].

3.3.2 Synaptic Intelligence (SI)

SI [73] introduces a surrogate regularization loss that has similar form as EWC, but uses a strength term Ω^i_j for θ_j, task i (instead of F_{jj}) that can be maintained online:

$$\omega_j^i := \int_{t_{i-1}}^{t_i} \frac{\partial \mathcal{L}}{\partial \theta_j} \frac{\partial \theta_j}{\partial t} dt$$

$$\Delta_j^i := \theta_j(t_i) - \theta_j(t_{i-1})$$

$$\Omega_j^i := \sum_{k<i} \frac{\omega_j^k}{(\Delta_j^k)^2 + \zeta}$$

$$\boldsymbol{\theta}_{1:i}^* = \min_{\boldsymbol{\theta}} \left\{ \mathcal{L}_i(\boldsymbol{\theta}) + c \sum_j \Omega_j^i (\theta_j - \theta_{1:i-1,j}^*)^2 \right\} \tag{3.3}$$

The strength Ω_j^i for parameter θ_j is proportional to how much it affects the loss in previous tasks (ω_j^k) and inversely proportional to how much it changed in previous tasks ($(\Delta_j^k)^2$). If θ_j affects the loss too much or hasn't been changed much previously, then it is deemed important. c is another type of strength parameter (like λ in EWC) that can be used to scale the regularization term.

3.4 Memory Approaches

Memory approaches store examples from each task being learned and then learn a new task while simultaneously maximizing performance on each of the stored memories. Storing memories is inspired from biological systems. The Complementary Learning Systems (CLS) theory [52] states that intelligent animals typically have dual memory systems, one for short term and one for long term. The other concept that memory approaches borrow is *rehearsal*, wherein information has to be replayed to be assimilated long term [52]. To rehearse data, continual learning approaches typically either store a subset of the data, or learn a generative model of the data.

Shin *et al.* [60] propose Generative Replay as a method of rehearsal. This involves learning a deep generative model (GAN) of previously seen data, alongside a classifier. The classifier is trained on the new dataset, but is also fed generated data from the generative model. Fearnet [23], inspired by CLS theory, similarly maintains multiple networks, but uses a generative autoencoder for replay.

Gradient Episodic Memory (GEM) [37], maintains an episodic memory, or a subset of every dataset (task) seen within an overall memory budget. While learning a new task, GEM ensures that the change in weights does not increase the loss on a previous dataset by solving a quadratic program.

3.5 Miscellaneous Approaches

Some approaches do not directly fall into any of the previously discussed categories, or fall into more than one categories. We discuss such existing work in this section.

Neuroevolution PathNet [7] uses a genetic algorithm to evolve pathways in a network. While the topology is fixed in many non-genetic approaches, PathNet models freeze task relevant pathways to avoid catastrophic forgetting.

Combination of architectural and regularization approaches AR1 [39] - combines an existing architectural approach (CWR+) with a regularization approach (SI). Similarly, Progress & Compress [58] uses progressive networks to maintain two columns – one for newest task, and one for the past tasks, and then consolidates the new information into the past column using EWC, which is a regularization method.

Dropout Networks trained with dropout are also less prone to the catastrophic forgetting problem [14]. Incremental Moment Matching [32] provides a few practical techniques for continual learning using dropout networks.

Attention Masks Serra *et al.* [59] propose Hard Attention to the Task (HAT), in which hard attention masks are learned for different tasks to preserve old knowledge without affecting the learning for the newest task. Mallya *et al.* [38] propose PackNet, which uses network pruning through binary masks to learn new tasks.

Variational Methods Variational Continual Learning (VCL) [46] combines methods in online variational inference and Monte Carlo variational inference into a general framework for continual learning. The proposed ideas are shown to work both in discriminative and generative continual settings.

3.6 Approaches for Continual RL

This section aims to summarize some of the known continual strategies that work in the RL setting. Unfortunately, there are relatively only few works in continual RL, compared to continual learning literature for classification.

Actor-Mimic [49] demonstrate that pretraining a Q network on one reinforcement learning task can lead to faster convergence on a different task. Berseth *et al.* [2] discuss curriculum learning strategies for continual RL, and propose Progressive Learning and Integration via Distillation (PLAID) as a strategy for combining multiple expert policies continually.

Progressive Nets [54] demonstrate the viability of constructive architectures for continual reinforcement learning on A3C, performing better than finetuning strategies. EWC [26] show how a simple DQN can learn multiple games appearing in a random schedule through the elastic criterion discussed in Section 3.3.1. Progress & Compress [58] combine progressive networks with EWC to achieve a scalable strategy for continual RL. Distral [65] is another approach similar to Progress & Compress that maintains a shared policy across tasks, which is used to guide individual policies. After learning a policy, the knowledge is re-distilled into the shared policy using policy distillation [53].

Tessler *et al.* [66] develop a hierarchical approach to lifelong learning using options, where knowledge is accommodated as skills in a "deep skill network" using skill distillation, a variation of policy distillation.

Kaplanis *et al.* [22] model neurons using a synaptic model that was devised to maximize the signal-to-noise ratio of memories over time in a population of synapses, and apply the idea to Q networks.

Chapter 4

CL Strategies for Classification

In this chapter, we investigate the effect of the stability bias, and propose strategies that use an alternative bias. The motivation is to achieve solutions that retain validated knowledge more strongly, and therefore prioritize stability over plasticity.

In Section 4.1, we explain the decomposition of the regularized continual learning gradient update. In Section 4.2, we quantify the absolute amount of forgetting that occurs over a dataset and use it to propose a weaker stability bias. In Section 4.3, we explain how this quantification can be used to preserve the output labels in specific use cases, depending on the requirement. Afterwards, we propose a few simple strategies for continual learning. In Section 4.4, we propose direct minimization as a family of strategies to directly minimize the absolute amount of forgetting. In Section 4.5, we extend direct minimization strategies to achieve finer control over the amount of forgetting. In Section 4.6, we propose Fisher freezing as an alternative way to achieve a larger span of solutions. The corresponding experiments can be found in Section 6.1.

4.1 Components of Regularized CL Updates

Regularized approaches to continual learning assume a fixed capacity of the network and jointly optimize two objectives per task through a combined loss $\mathcal{L}(\boldsymbol{\theta})$ as follows (G is the regularization loss):

$$\mathcal{L}(\boldsymbol{\theta}) := \mathcal{L}_i(\boldsymbol{\theta}) + \lambda G_{1:i-1}(\boldsymbol{\theta}, \boldsymbol{\theta}^*_{1:i-1}) \tag{4.1}$$

$$\boldsymbol{\theta}^*_{1:i} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{4.2}$$

To incorporate the knowledge for an example (x, y), the optimization step computes a gradient update:

$$\Delta\boldsymbol{\theta} \propto -\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \tag{4.3}$$

Note that many optimization methods use additional strategies to converge faster to a solution, like momentum [62], adaptive gradients [5] and moment estimation [25]. However, on their own they do not offer any particular advantage for continual learning. Hence, for simplicity, we just discuss the first order gradient update on the regularized continual learning objective, that is $-\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$. This gradient update is sufficiently informative to provide useful insights about how these methods preserve existing knowledge.

With the regularized loss, the gradient update breaks down into two components: $-\nabla_{\boldsymbol{\theta}}\{\mathcal{L}_i(\boldsymbol{\theta})\}$ and $-\nabla_{\boldsymbol{\theta}}\{\lambda G_{1:i-1}(\boldsymbol{\theta}, \boldsymbol{\theta}^*_{1:i-1})\}$. The former can be interpreted as the *plasticity update*, which tries to optimize the current task's objective, while the latter can be interpreted as the *stability update*, which tries to maintain existing knowledge.

Plasticity Update Throughout this work, we use the cross-entropy loss (2.2) to compute the plasticity update. The cross-entropy objective is a suitable plasticity update, since it produces exponentially strong updates as $P_{\boldsymbol{\theta}}(\cdot|x) \to 0$.

$$-\nabla_{\boldsymbol{\theta}}\{\mathcal{L}_j(\boldsymbol{\theta})\} \propto \frac{1}{P_{\boldsymbol{\theta}}(\cdot|x)} \tag{4.4}$$

Stability Update Regularization methods in the literature that use a second order Taylor approximation on the KL divergence as the regularization loss, like (3.1), or a form like (3.3) construct a stability update that resembles the form $-\mathbf{a} \cdot (\boldsymbol{\theta} - \boldsymbol{\theta}^*)$. EWC [28] constructs a regularization loss $G_{1:i-1}(\boldsymbol{\theta})$ by multiplying each $(\theta_j - \theta^*_{1:i-1,j})^2$ by the corresponding diagonal term F_{jj} in the Fisher information matrix. The squared displacement of the weight θ_j from the previous weight $\theta^*_{1:i-1,j}$ is scaled by the Fisher importance of the weight. The stability update therefore has the form:

$$-\nabla_{\boldsymbol{\theta}}\{\lambda G_{1:i-1}(\boldsymbol{\theta}, \boldsymbol{\theta}^*_{1:i-1})\} = -\lambda \sum_j F_{jj}(\theta_j - \theta^*_{1:i-1,j})$$

This update can be understood as an elastic constraint (much like a spring) which forces θ_j to be close to $\theta_{1:i-1,j}$. The strength of this stability update for θ_j depends on λ, the Fisher information term F_{jj} for θ_j and the magnitude of $(\theta_j - \theta^*_{1:i-1,j})$. The direction of this update is always towards $\theta^*_{1:i-1,j}$. SI [73] uses a similar form as EWC, but calculates the per parameter importance Ω^i_j in a different way as explained previously. However, the form of the stability update remains similar to EWC:

$$-\nabla_{\boldsymbol{\theta}}\{\lambda G_{1:i-1}(\boldsymbol{\theta}, \boldsymbol{\theta}^*_{1:i-1})\} = -2c \sum_j \Omega^i_j(\theta_j - \theta^*_{1:i-1,j})$$

This is still the elastic constraint, except that the per-parameter importance is Ω^i_j, rather than the Fisher importance. There are other approaches that use a similar form of update but use a different strategy to compute the constant of regularization, for example, EWC++ [3].

We note that while the elastic constraint and its variants constitute a good stability bias for continual learning, they may be stronger than is needed. Specifically, they minimize the KL divergence between the posterior and prior but do not directly consider the amount of forgetting, which affects the preservation of (validated) knowledge.

4.2 Absolute Amount of Forgetting

In this section, we quantify the amount of forgetting as a network learns new data. This quantification provides a generic strategy to define the per-parameter importance in the stability update. More concretely, we provide an approximate upper bound on the absolute amount of forgetting over a dataset.

After learning a task i, the weights of the network are $\boldsymbol{\theta}^*_{1:i}$. For simplicity, let $\boldsymbol{\theta}^*_{1:i} \equiv \boldsymbol{\theta}^*$ and that afterwards, at any point in the sequential training process, the weights are at $\boldsymbol{\theta}^* + \Delta\boldsymbol{\theta}$. Assuming $\Delta\boldsymbol{\theta}$ is small, we can apply a first order Taylor approximation of the individual predicted likelihood $P^m_{\boldsymbol{\theta}^*+\Delta\boldsymbol{\theta}}$ in the neighborhood of $\boldsymbol{\theta} = \boldsymbol{\theta}^*$:

$$P^m_{\boldsymbol{\theta}^*+\Delta\boldsymbol{\theta}} \approx P^m_{\boldsymbol{\theta}^*} + \Delta\boldsymbol{\theta} \cdot (\nabla_{\boldsymbol{\theta}} P^m_{\boldsymbol{\theta}})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} \tag{4.5}$$

The individual predicted likelihood $P^m_{\boldsymbol{\theta}}$ on an example $x \in \mathcal{D}_i$ changes by the magnitude

$$|\Delta P^m_{\boldsymbol{\theta}^*}| = |P^m_{\boldsymbol{\theta}^*+\Delta\boldsymbol{\theta}}(\cdot|x) - P^m_{\boldsymbol{\theta}^*}(\cdot|x)|. \tag{4.6}$$

On average, the magnitude change in the individual predicted likelihood $P^m_{\boldsymbol{\theta}}$ over the dataset $\mathcal{D}_i$ is given by the expectation:

$$\begin{aligned}
\mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\left|P^m_{\boldsymbol{\theta}^*+\Delta\boldsymbol{\theta}}(\cdot|x) - P^m_{\boldsymbol{\theta}^*}(\cdot|x)\right|\right] &= \mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\left|\Delta\boldsymbol{\theta}\cdot(\nabla_{\boldsymbol{\theta}}P^m_{\boldsymbol{\theta}})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}\right|\right] \\
&\text{(empirical averaging over } N \text{ samples)} \\
&\approx \frac{1}{N}\sum_{(x,y)\in\mathcal{D}_i}\left|\sum_{j=1}^{\dim(\boldsymbol{\theta})}\Delta\theta_j\frac{\partial P^m_{\boldsymbol{\theta}}(\cdot|x)}{\partial\theta_j}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}\right| \\
&\leq \frac{1}{N}\sum_{(x,y)\in\mathcal{D}_i}\sum_{j=1}^{\dim(\boldsymbol{\theta})}\left|\Delta\theta_j\right|\left|\frac{\partial P^m_{\boldsymbol{\theta}}(\cdot|x)}{\partial\theta_j}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}\right| \\
&= \sum_{j=1}^{\dim(\boldsymbol{\theta})}\left|\Delta\theta_j\right|\frac{1}{N}\sum_{(x,y)\in\mathcal{D}_i}\left|\frac{\partial P^m_{\boldsymbol{\theta}}(\cdot|x)}{\partial\theta_j}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}\right| \\
&= |\Delta\boldsymbol{\theta}|\cdot\mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\left|\nabla_{\boldsymbol{\theta}}P^m_{\boldsymbol{\theta}}(\cdot|x)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}\right|\right] \qquad (4.7) \\
&\equiv \sum_j C^m_j\|\theta_j - \theta^*_{1:i,j}\|_1 \equiv u^m(\mathcal{D}_i, \boldsymbol{\theta}^*_{1:i}, \boldsymbol{\theta}) \qquad (4.8)
\end{aligned}$$

At every task i, we can minimize $u^m(\mathcal{D}_{i'}, \boldsymbol{\theta}^*_{1:i'}, \boldsymbol{\theta})$ directly for each of the previous datasets i', and this minimization should mitigate catastrophic forgetting (C^m_j is parameter importance for θ_j) for the network's outputs for label m ($1 \leq m \leq M$). This constitutes our *minimization criterion*. We extend the notation to use u^m when computing this upper bound on P^m, and u when computing it for the ground truth.

An approximate upper bound for the squared change in $P^m_{\boldsymbol{\theta}}$ (*aka* the L_2 variant) can be similarly obtained:

$$\mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\left(P^m_{\boldsymbol{\theta}^*+\Delta\boldsymbol{\theta}}(\cdot|x) - P^m_{\boldsymbol{\theta}^*}(\cdot|x)\right)^2\right] \lessapprox |\Delta\boldsymbol{\theta}|^2\cdot\mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\left(\nabla_{\boldsymbol{\theta}}P^m_{\boldsymbol{\theta}}(\cdot|x)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}\right)^2\right] \tag{4.9}$$

A Weaker Constraint If this absolute amount of forgetting is a part of the stability update, then the stability update corresponds to a weaker constraint, compared to the elastic constraint in EWC:

$$
\begin{aligned}
-\nabla_{\boldsymbol{\theta}}\{\lambda G_{1:i}(\boldsymbol{\theta}, \boldsymbol{\theta}^*_{1:i-1})\} &= -\nabla_{\boldsymbol{\theta}}\{\lambda u^m(\mathcal{D}_{i-1}, \boldsymbol{\theta}^*_{1:i-1}, \boldsymbol{\theta})\} \\
&\equiv -\nabla_{\boldsymbol{\theta}}\left\{\lambda \sum_j C^m_j \|\theta_j - \theta^*_{1:i-1,j}\|_1\right\} && (4.10) \\
&= -\lambda \sum_j C^m_j \operatorname{sign}(\theta_j - \theta^*_{1:i-1,j}) && (4.11)
\end{aligned}
$$

Unlike the elastic constraint, the strength of this stability update for θ_j depends on λ and on C^m_j, but not on the magnitude of $(\theta_j - \theta^*_{1:i-1,j})$; the direction of this update is always towards $\theta^*_{1:i-1,j}$. This is therefore a weaker stability bias than the elastic constraint.

To understand the nature of the solutions obtained by this weaker update, we consider a simple two task example (Sim-EMNIST, but for two tasks). Using the method described in Section 4.4, we plot the task 1 and task 2 accuracies on the x and y axes and obtain a figure as shown in Figure 4.1. We observe that as the strength of λ increases, more of the task 1 accuracy is preserved, but at some point the performance on task 2 decreases. Of the two variants, the L_1 constraint obtains sparse solutions, which are almost always more restrictive on forgetting task 1. This is by design, since the goal is to minimize forgetting on previous task(s) more strongly, as explained later. The vertical dotted line corresponds to the maximum accuracy achieved at the end of training task 1. The dashed diagonal line represents the line across which the best equivalent overall accuracy will lie ($x + y = c$ for some constant c).

The point at which the performance on task 2 decreases corresponds to the optimum of the stability-plasticity curve. To understand this curve, in Figure 4.2 we plot the final average validation accuracy for this two task example across a wide range of λ. As can be observed, for both the L_1 and L_2 versions of absolute expected amount of forgetting, there are different values of λ at which the approach achieves (joint) best stability and plasticity. This is achieved with lower values of λ for the L_1 variant than for the L_2 variant.

Analogy with Regression Models The regularized continual learning objective has a similar shape to the objective in biased regression models. Regression models typically

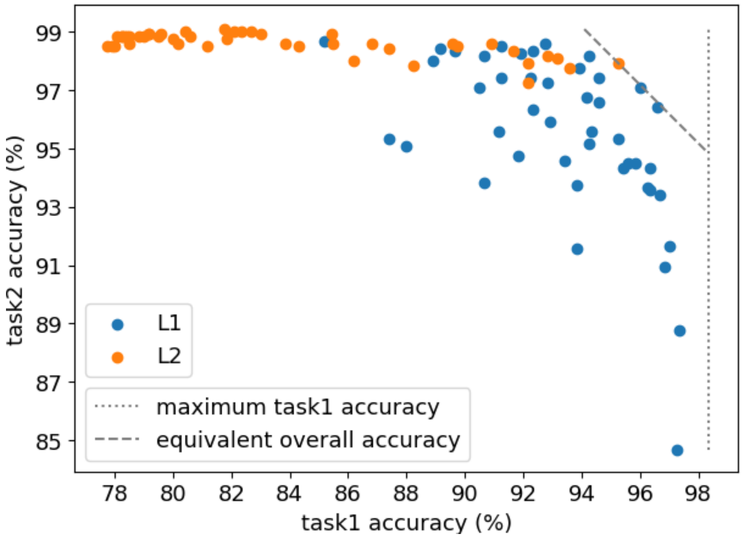

Figure 4.1: The nature of solutions obtained with L_2 vs L_1 constraints on Sim-EMNIST with 2 tasks, 5 seeds and different strengths ($\lambda \in [1, 10^4]$). L_1 solutions are more spread out (higher variance) and preserve the previous behavior more strongly than traditional L_2 strategies for the same strengths. See Section 4.2.

minimize an ordinary least-squares objective OLS($\boldsymbol{\theta}$) but suffer from a high variance problem. The most popular ways to reduce this variance is through LASSO and ridge methods, which introduce a bias onto the estimate of regression weights $\boldsymbol{\theta}$ through a regularization term. LASSO adds an L_1 term $\|\boldsymbol{\theta}\|_1$ as the regularization term, while ridge adds an L_2 term $\|\boldsymbol{\theta}\|_2^2$ as the regularization term. The effect of these biases is that the solution satisfies $\|\boldsymbol{\theta}\|_1 \leq t$ or $\|\boldsymbol{\theta}\|_2^2 \leq t$, for some small t, with LASSO producing a sparse solution, that is, a solution with fewer non-zero weights. From a Bayesian standpoint, this is because LASSO shrinkage is equivalent to a Laplace prior, while ridge shrinkage is equivalent to a Gaussian prior. Since a Laplace prior is more peaky towards its mean, the solution is more likely to be the mean (and near the mean) than with a Gaussian prior.

Similarly, regularized continual learning minimizes the loss for the most recent task $\mathcal{L}_i(\boldsymbol{\theta})$, while ensuring that $G_{1:i-1}(\boldsymbol{\theta}, \boldsymbol{\theta}^*_{1:i-1}) \leq t$. When using the approximate KL-divergence for G, it yields the Bayesian interpretation (as explained in EWC) that the KL-divergence between the prior $p(\boldsymbol{\theta}|\mathcal{D}_{1:i-1})$ and the posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:i})$ should be $\leq t$ while simultaneously minimizing the cross-entropy loss. When using the absolute amount of the forgetting, it can be understood as trying to keep the absolute amount of forgetting from previous

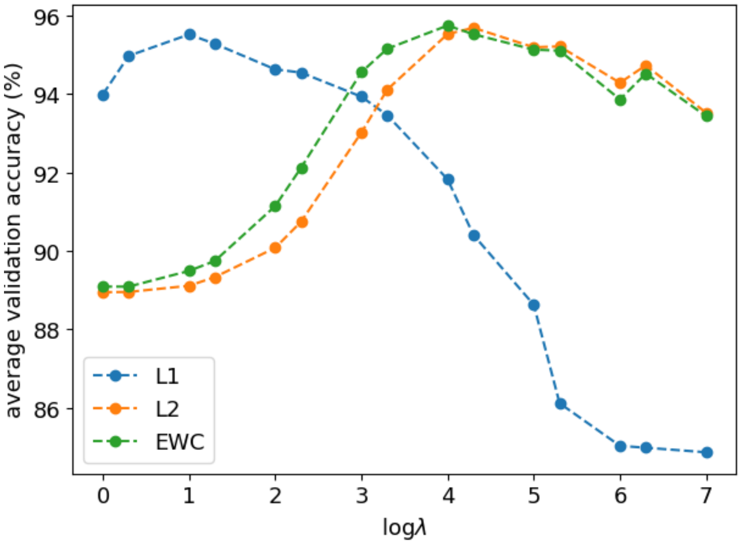

Figure 4.2: The effect of λ on the final average validation accuracy for DM-L_1 (Case III), DM-L_2 (Case III) and EWC, for Sim-EMNIST, two tasks, 10 seeds. The different approaches achieve the optimum joint stability and plasticity at different strengths of λ.

datasets within a certain t. With the L_1 version of the constraint, more θ_j are likely to not change than in the L_2 version.

Equivalence with KL Approaches If the classification behavior corresponding to the ground truth label is desired to be preserved, the minimization criterion has a similar form to KL approaches, which use a loss based on the KL-approximation as stated earlier. More concretely, the squared L_2 version of our upper bound is smaller than the second-order approximate KL divergence (with the diagonal of the empirical Fisher). This means that when using the expectation term instead of the Fisher, the stability updates are smaller in magnitude, and thus, less strong. This interpretation of per-parameter updates achieves a slightly higher model variance, which is aligned with our goal. The proof, which follows from (4.9) is as follows:

$$
\begin{aligned}
\mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\left(P_{\boldsymbol{\theta}^*+\Delta\boldsymbol{\theta}}(\cdot|x)-P_{\boldsymbol{\theta}^*}(\cdot|x)\right)^2\right] &\lessapprox |\Delta\boldsymbol{\theta}|^2\cdot\mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\left|\nabla_{\boldsymbol{\theta}}P_{\boldsymbol{\theta}}(\cdot|x)|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}\right|^2\right] \\
&\leq |\Delta\boldsymbol{\theta}|^2\cdot\mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\left|\nabla_{\boldsymbol{\theta}}\log P_{\boldsymbol{\theta}}(\cdot|x)|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*}\right|^2\right] \\
&= \sum_j F_{jj}(\theta_j-\theta_j^*)^2
\end{aligned}
$$

4.3 Extension to Different Use Cases

The formulation of the upper bound in terms of the output label allows for its adaptation to different use cases. Specifically, real world systems have different motivations for continual learning, which are dependent on the *requirement* specification. For example, in an outlier detection system, it may be desired to preserve the reject class more strongly than the rest of the classes, since retaining knowledge about the outlier examples is of the highest priority. Therefore, more generally, depending on the requirement, it may be desired to preserve the network behavior expressed by all or some of the output neurons. We identify four such use cases, which we illustrate in Figure 4.3 and describe below.

Case I We can preserve the entire set of predicted likelihoods from $\boldsymbol{\theta}^*$ to $\boldsymbol{\theta}^*+\Delta\boldsymbol{\theta}$, which penalizes changes to any individual predicted likelihood. This is the most restrictive version of the criterion and can be achieved by regularizing a sum over $1\leq m\leq M$ of the individual changes:

$$
u_{\mathrm{I}}(\mathcal{D}_i,\boldsymbol{\theta}^*_{1:i},\boldsymbol{\theta}) := \sum_{1\leq m\leq M} u^m(\mathcal{D}_i,\boldsymbol{\theta}^*_{1:i},\boldsymbol{\theta}) \tag{4.12}
$$

Case II We can preserve the change in predicted likelihood for the confident label(s) at $\boldsymbol{\theta}^*$, which typically corresponds to the highest individual probability in $\{P^m_{\boldsymbol{\theta}^*}(\cdot|x)\}_{m=1}^M$. This may be desired in tasks related to safety-critical systems, where a network has been safety-calibrated at deployment and now needs to add some more knowledge without violating previously satisfied safety calibrations. To achieve this, we can first compute an expectation over $|(\Delta P^m_{\boldsymbol{\theta}^*})P^m_{\boldsymbol{\theta}^*}|$ rather than the formulation in (4.6):

$$
\mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\left|(\nabla_{\boldsymbol{\theta}}P^m_{\boldsymbol{\theta}}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*_{1:i}})P^m_{\boldsymbol{\theta}^*_{1:i}}\right|\right] \equiv \bar{u}^m(\mathcal{D}_i,\boldsymbol{\theta}^*_{1:i},\boldsymbol{\theta}) \tag{4.13}
$$

Since the network's outputs represent the confidence of the x being a certain label, highly confident $P^m_{\boldsymbol{\theta}^*}$ ends up contributing more to the expectation. Then, a sum over $1 \leq m \leq M$ of the confidence weighted upper bound $\bar{u}^m$ can be used:

$$u_{\mathrm{II}}(\mathcal{D}_i, \boldsymbol{\theta}^*_{1:i}, \boldsymbol{\theta}) := \sum_{1 \leq m \leq M} \bar{u}^m(\mathcal{D}_i, \boldsymbol{\theta}^*_{1:i}, \boldsymbol{\theta}) \tag{4.14}$$

Case III We can preserve the absolute change in predicted likelihood for the ground truth by directly regularizing $u_{\mathrm{III}}(\mathcal{D}_i, \boldsymbol{\theta}^*_{1:i}, \boldsymbol{\theta}) := u(\mathcal{D}_i, \boldsymbol{\theta}^*_{1:i}, \boldsymbol{\theta})$. This corresponds to the typical motivation, which is to maintain a stable average validation accuracy.

Case IV We can partially preserve the change in predicted likelihood for the ground truth, that is, penalize the change $P_{\boldsymbol{\theta}^*}(\cdot|x) = 1 \to P_{\boldsymbol{\theta}^*+\Delta\boldsymbol{\theta}}(\cdot|x) = 0$, but allow the change $P_{\boldsymbol{\theta}^*}(\cdot|x) = 0 \to P_{\boldsymbol{\theta}^*+\Delta\boldsymbol{\theta}}(\cdot|x) = 1$ for the ground truth predicted likelihood. This applies the penalty only when a correctly classified x at $\boldsymbol{\theta}^*$ (high confidence on the ground truth) becomes incorrectly classified (lower confidence on the ground truth) at $\boldsymbol{\theta}^* + \Delta\boldsymbol{\theta}$. Using previously defined notation, this yields:

$$\begin{aligned} u_{\mathrm{IV}}(\mathcal{D}_i, \boldsymbol{\theta}^*_{1:i}, \boldsymbol{\theta}) &:= \bar{u}(\mathcal{D}_i, \boldsymbol{\theta}^*_{1:i}, \boldsymbol{\theta}) \\ &= \mathbb{E}_{(x,y)\sim\mathcal{D}_i}\left[\left|(\nabla_{\boldsymbol{\theta}} P_{\boldsymbol{\theta}}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*_{1:i}}) P_{\boldsymbol{\theta}^*_{1:i}}\right|\right] \end{aligned} \tag{4.15}$$

4.4 Algorithm: Regularizing Amount of Forgetting

The upper bound(s) on the amount of forgetting can be directly used in the stability update. With the formulation now being understood in terms of the amount of forgetting, we need to ensure that the sum of absolute amounts of forgetting is $\leq t$ from all previous datasets. We can encode this objective by choosing a $G_{1:i-1}(\boldsymbol{\theta})$ in (4.1) as follows:

$$G_{1:i-1}(\boldsymbol{\theta}) := \sum_{1 \leq i' \leq i-1} g(\mathcal{D}_{i'}, \boldsymbol{\theta}^*_{1:i'}, \boldsymbol{\theta}) \equiv \sum_{1 \leq i' \leq i-1} g_{i'}$$

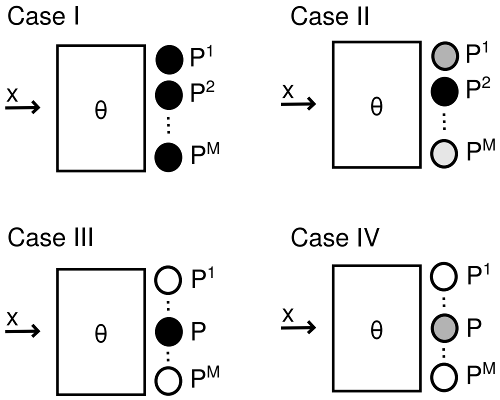

Figure 4.3: Use cases I-IV. The intensity of the output neurons denote their contribution. Case I preserves all the outputs. Case II preserves the outputs in proportion to their confidence. Case III preserves only the ground truth output. Case IV preserves the ground truth in proportion to its confidence.

Here, g can be either $u_{\text{I}}, u_{\text{II}}, u_{\text{III}}, u_{\text{IV}}$ for each of the previous datasets, depending on the requirement. After finishing the training for a particular dataset $\mathcal{D}_i$, $g(\cdot)$ can be computed and added to $G_{1:i-1}(\boldsymbol{\theta})$ to produce $G_{1:i}(\boldsymbol{\theta})$. With this definition, the objective can optimized as expressed in (4.2).

To evaluate which version of the stability bias (L_1 or L_2) is appropriate for knowledge preservation, we conduct experiments with the L_1, L_2 and the elastic-net variants of the objective. The L_1 and L_2 variants are described in Section 4.2, and the elastic-net variant is a weighted combination of these two variants [75]; we use an equally weighted combination, which we refer to as $E0.5$. We refer to the family of these methods as *direct minimization* (DM) strategies.

4.5 Algorithm: Finer Control over Forgetting

Minimization through the Lagrange method of multipliers produces a minimum for $\mathcal{L}_i(\boldsymbol{\theta})$, such that the absolute amount of forgetting over datasets $\mathcal{D}_{1:i-1}$ is bounded by some t, that is $G_{1:i-1}(\boldsymbol{\theta}) \leq t$. Without any additional assumptions on the solution, the value of t is determined by the minimization procedure. However, it is possible to alter the minimization procedure to achieve more control over the amount of forgetting t, and hence the degree of preservation. While this introduces a stronger bias and thereby reduces the variance of the model, it also gives us a hyperparameter to find the precise t at which the solution is optimal for a certain stability bias.

Specifically, let us assume that we do not want $G_{1:i-1}(\boldsymbol{\theta})$ to exceed some c. We can attach a scalar identity term $\mathbb{I}$ to the cross-entropy objective as follows:

$$\mathcal{L}_i(\boldsymbol{\theta}) \cdot \mathbb{I}(G_{1:i-1}(\boldsymbol{\theta}) \leq c) + \lambda G_{1:i-1}(\boldsymbol{\theta}) \tag{4.16}$$

As long as $G_{1:i-1}(\boldsymbol{\theta}) \leq c$, this training objective is equivalent to the description in (4.1) and (4.2). When $G_{1:i-1}(\boldsymbol{\theta}) > c$, the training objective directly minimizes $G_{1:i-1}(\boldsymbol{\theta})$ until $G_{1:i-1}(\boldsymbol{\theta}) \leq c$. This forces the optimizer to re-focus on finding a solution that first satisfies $G_{1:i-1}(\boldsymbol{\theta}) \leq c$, and thereby forces the optimization to not exceed a certain c (stability is enforced). A stricter objective can be obtained by individually considering the different g within G:

$$\mathcal{L}_i(\boldsymbol{\theta}) \cdot \mathbb{I}(g_1(\boldsymbol{\theta}) \leq c_1, \cdots, g_{i-1}(\boldsymbol{\theta}) \leq c_{i-1}) + \lambda G_{1:i-1}(\boldsymbol{\theta}) \tag{4.17}$$

In our experiments, we use this stricter objective, maintaining a forgetting threshold (c_i) per task i. We initialize $c_i \leftarrow c^{(1)}$ and then incrementally increase this forgetting threshold as we see more tasks, that is, $c_i \leftarrow c_i + c^{(2)}$ per new task seen. Through the hyperparameter search, we can obtain a minimum c which produces the best solution for a given stability bias.

4.6 Algorithm: Fisher Freezing

With any regularization strategy, all the weights are always updated, even if the changes to some weights are very small. This can perturb sensitive weights, such as early layer weights [50]. Even if this perturbation is only small, small perturbations can add up over multiple tasks and eventually affect the classifier likelihood irreversibly.

The upper bound of change in classifier likelihood for a dataset $\mathcal{D}_i$ is dependent on two terms (see (4.7)), $|\Delta\boldsymbol{\theta}|$ and the expectation of the absolute gradients. To minimize the change in classifier likelihood, we may opt to minimize $|\Delta\boldsymbol{\theta}|$ more conventionally, by freezing the most important weights. This reduces the magnitude of $|\Delta\boldsymbol{\theta}|$ and therefore results in a smaller change in classifier predictions. Other strategies in the literature have tried freezing weights [54], but our approach to freezing is a more principled way to choose which weights to freeze, and directly enforces sparsity in $|\Delta\boldsymbol{\theta}|$.

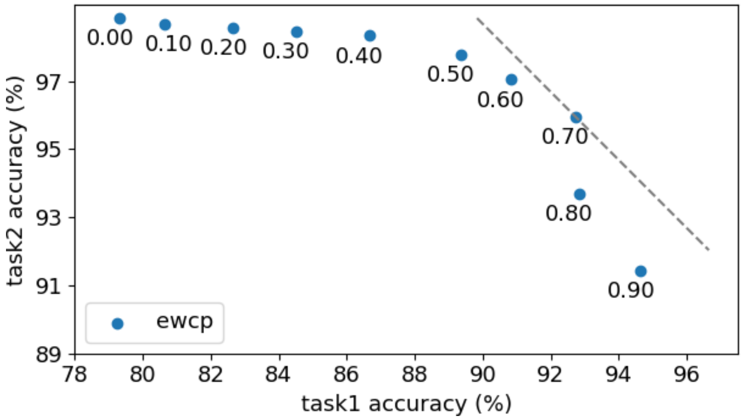

Figure 4.4: Fisher freezing on EWC applied to Sim-EMNIST; average of 5 seeds with $\lambda = 1$. As we freeze more weights, task 1 accuracy increases, while task 2 accuracy decreases. The trend of solutions is similar to that obtained using L_1.

Specifically, we compute the Fisher information matrix $F(\boldsymbol{\theta}^*_{1:i})$ and choose the top p-percentile parameters $\boldsymbol{\theta}_p \subseteq \boldsymbol{\theta}$. For these parameters, we ensure that the optimizer does not update their values. To assess the effects of this kind of freezing separately from the aforementioned L_1 criterion, we freeze weights on EWC. In our experiments, we refer to this method as *EWC-p*.

As an example, we plot the task 1 vs task 2 accuracy in Figure 4.4 for EWC with freezing. The nature of solutions with increasing p is similar to that depicted in Figure 4.1. An increase in p therefore reduces the plasticity of the model, which is expected. This shows that a higher span of solutions (higher variance) can be achieved directly through EWC, by introducing this hyperparameter p.

Chapter 5

Extension to Continual RL

In this chapter, we propose a simple extension of the ideas presented in Chapter 4 to reinforcement learning. In Section 5.1, we discuss the originally proposed application of the EWC objective (Section 3.3.1) to continual reinforcement learning as presented in Kirkpatrick *et al.* [26]. In Section 5.2, we explain the difference between the original approach and our implementation, which uses PPO. In Section 5.3, we propose a strategy to weaken the stability bias. The relevant experiments can be found in Section 6.2.

5.1 EWC: Original Implementation

Originally, Kirkpatrick *et al.* [26] extended EWC to a DQN setup [45, 67]. A random schedule of 10 Atari environments was used, and the environments were visited multiple times. The problem statement also involved learning which task is currently being seen. This task determination was learned using the Forget Me Not (FMN) algorithm [44].

Automatic Task Determination The implementation of FMN used by Kirkpatrick *et al.* [26] learns a Hidden Markov Model (HMM) to automatically determine the current task during the training process. The hidden variable of the HMM is a categorical context c which corresponds to one of the tasks (environments). The HMM evolves according to:

$$p(c, t+1) = \sum_{c'} p(c', t) \cdot \Gamma(c, c')$$

Here α is the probability of task switch, that is, $\Gamma(c, c') = \alpha$ if $c \neq c'$, else $\Gamma(c, c') = 1 - \alpha$.

Each task is further associated with a generative model that can predict the probability $p(o|c,t)$ of an observation o. Since Kirkpatrick *et al.* [26] conduct experiments with Atari environments, this generative model is based on factored multinomial distributions which explain the probability of each pixel in the observation space, which leads to a Dirichlet model. The value of α used is inversely proportional to time.

The training is divided into windows. In each window, the generative models corresponding to all environments are updated with the evidence as proposed in FMN [44]. At the end of the window, the best model that explains the current task is kept, while other models are reverted to their state at the beginning of the window.

Given such a model per task, the current task can be inferred at time t as follows. If the observations so far are $o_1, o_2, \cdots, o_t$, then

$$p(c|o_1, o_2, \cdots, o_t) \propto \sum_{c'} p(c', t-1) \cdot \Gamma(c, c') \cdot p(o|c, t)$$

The context c with the highest probability is inferred to be the current task label.

Computing the Fisher Matrix The Fisher Matrix is computed at every task switch, and a quadratic penalty as explained in Section 3.3.1 is added to the training objective per task after 20M frames of training (total training occurs for 50M frames). For every inferred task, Kirkpatrick *et al.* [26] maintain a short-term memory buffer B. Then, when the task switch happens, the empirical Fisher Matrix F is computed over 100 randomly sampled experiences from the short-term buffer. Unfortunately, the authors don't provide any further details regarding the Fisher Matrix computation for the reinforcement learning case, and it is not immediately clear what the likelihood function represents.

Learning without task determination Kirkpatrick *et al.* [26] also investigate the effect of learning when the task sequence is known, that is, without automatic task determination. The improvement was not found to be significant compared to the FMN case.

5.2 PPO-EWC

In this section, we explain the differences in our implementation over the implementation in Kirkpatrick *et al.* [26]. We chose to implement EWC with PPO instead of DQN, due to the lack of a reference implementation and preliminary experiments with DQN which suggested orders of magnitude slower convergence than PPO. While EWC has been shown to work with DQN in Kirkpatrick *et al.* [26], the authors do not investigate the effect of the strength of the stability update, which is the focus of our work.

The pseudocode of our adaptation of EWC to PPO is given in Algorithm 3, and it builds upon the baseline pseudocode given in Algorithm 2.

Efficient Convergence Our algorithm adopts EWC to the Clipped PPO algorithm [57], explained in Section 2.4. PPO is an actor-critic style algorithm, that is easy to implement, works with discrete action spaces, and is usually considered to be practically more efficient in convergence than DQNs [45, 67].

Training Schedule The original EWC [26] is tested against multiple Atari games in a randomized order, such the same game can be revisited. This interleaving of experiences is very similar to joint training and does not demonstrate the viability of EWC in settings where the same environment cannot be visited again.

In our implementation, we evaluate EWC on a curriculum of simple autonomous driving tasks, going from easy to difficult. We argue that this is a more natural use case for continual learning, compared to the randomized schedule. We note that curriculum learning has been previously investigated [1], although usually in the context of vision and language tasks.

Categorical vs Gaussian Policy Autonomous driving tasks are usually modeled as environments with continuous action spaces. Previous continual learning work(s) have proposed methods for continuous action space environments [2]. With PPO, continuous action space policies are typically modeled as (diagonal covariance) Gaussian policies, implemented as neural networks which take in a state s and produce a mean $\boldsymbol{\mu}_{\boldsymbol{\theta}}(s)$ and variance $\boldsymbol{\sigma}_{\boldsymbol{\theta}}(s)$ for each continuous action parameter. During training, actions are sampled from this distribution:

$$a \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(s), \boldsymbol{\sigma}_{\boldsymbol{\theta}}(s))$$

This sampling achieves an effect similar to exploration. During evaluation, the mode $\boldsymbol{\mu}_{\boldsymbol{\theta}}(s)$ of the distribution is directly used as the action (most likely action).

While a Gaussian policy can act as the control policy for choosing actions in PPO, it cannot be used to compute the Fisher Matrix. This is because the Fisher Matrix usually requires the likelihood and the score function to be defined, and the Gaussian policy is a generative model. Therefore, we chose to discretize our action space, and use a Categorical policy, which is a discriminative model whose likelihood and score can be easily computed through softmax. DQNs typically "correspond" to a Categorical policy.

Preserving Q functions Deep RL methods are typically either value function approximation methods (no policy) or actor critic style algorithms (both policy and value). In the case of value approximation methods, the policy is constructed from the learned $Q_{\boldsymbol{\theta}}$ function as:

$$\pi(s) = \arg\max_{a \in \mathcal{A}} Q_{\boldsymbol{\theta}}(s, a)$$

Continually preserving the $Q_{\boldsymbol{\theta}}$ function therefore also preserves the policy in these methods [49, 26, 22].

In the case of actor critic style algorithms, previously proposed methods typically share some initial layers between the policy function π and the Q network [54, 58]. In our implementation, we do not share layers between π and Q, and instead opt to learn these networks separately. With separate networks, Q just reflects the corresponding value estimate for π, and therefore it is sufficient to just preserve π across multiple environments. That is, we just remember "how to act" across environments, and not the reasoning (value estimate) behind the actions.

Computing the Fisher Matrix Since we are not working with a randomized continual schedule, automatic task determination is not necessary. Also, since our experiments are not on Atari, we do not learn a generative multinomial model for the observations. We instead compute the Fisher Matrix over states from successful trajectories during training. The set of successful trajectories should produce a distribution of states that can be used to compute the likelihood.

Algorithm 2 PPO-Baseline

```
Given sequence of environments $e_1, e_2, \cdots, e_n$
for environment = $e_1, e_2, \cdots, e_n$ do        ▷ Current environment is $e_i$
    Initialize critic $Q$
    for iteration = 1, 2, ... do
        for actor = 1, 2, ..., $N$ (parallel) do
            Run policy $\pi_{\boldsymbol{\theta}_o}$ in environment $e_i$ for $T$ timesteps
            Compute advantage estimates $\Psi_t$, $1 \leq t \leq T$
        end for
        for epoch = 1, 2, ..., $K$ do
            Optimize surrogate $L^{CLIP}(\boldsymbol{\theta})$ on $M$ samples, $M \leq NT$
        end for
    end for
end for
```

Algorithm 3 PPO-EWC

```
Given sequence of environments $e_1, e_2, \cdots, e_n$
Initialize $L^{EWC}(\boldsymbol{\theta}) := 0$
for environment = $e_1, e_2, \cdots, e_n$ do        ▷ Current environment is $e_i$
    Initialize critic $Q$
    Initialize set of successful trajectories $\mathcal{M}_i := \phi$ for $e_i$
    for iteration = 1, 2, ... do
        for actor = 1, 2, ..., $N$ (parallel) do
            Run policy $\pi_{\boldsymbol{\theta}_o}$ in environment $e_i$ for $T$ timesteps
            Compute advantage estimates $\Psi_t$, $1 \leq t \leq T$
            Update $\mathcal{M}_i$ with successful trajectories seen so far
        end for
        for epoch = 1, 2, ..., $K$ do
            Optimize surrogate $L^{CLIP}(\boldsymbol{\theta}) + \lambda \cdot L^{EWC}(\boldsymbol{\theta})$ on $M$ samples, $M \leq NT$
        end for
    end for
    Compute Fisher Matrix $F_i$ for states $s \in \mathcal{M}_i$
    Snapshot the current set of weights $\boldsymbol{\theta}$ as $\boldsymbol{\theta}^*_{1:i}$
    $L^{EWC}(\boldsymbol{\theta}) \leftarrow L^{EWC}(\boldsymbol{\theta}) + \mathbf{diag}(F_i) \cdot |\boldsymbol{\theta} - \boldsymbol{\theta}^*_{1:i}|^2$
end for
```

5.3 Weakening the Bias

To extend the principle of a weaker stability bias as discussed in Chapter 4, we note how continual RL is different from continual learning for classification. Most importantly, the procedure of optimization in RL involves changing the policy and choosing random actions, which is a trial-and-error strategy. In fact, obtaining a policy through PPO-EWC (Algorithm 3) is still difficult, mainly because the stability update due to $L^{EWC}(\boldsymbol{\theta})$ changes the policy and detracts from the usual exploration strategy.

A simple way to relax the stability bias is to do stability updates every δ epochs ($\delta \geq 1$). As δ increases, the push towards $\boldsymbol{\theta}^*_{1:i}$ becomes weaker. A very high δ should correspond to no stability update, that is PPO-Baseline (Algorithm 2). We investigate different values of δ in our experiments.

Chapter 6

Experiments

In this chapter, we evaluate the methods proposed in Chapter 4 and 5. The code is available at `github.com/ashishgaurav13/stability-biases`.

6.1 Experiments for Classification

In this section, we evaluate the methods proposed in Chapter 4 and compare their performance with other popular KL based approaches in continual learning. We evaluate the following strategies:

Baseline The usual baseline involves training with just the likelihood loss, that is, no regularization (no stability bias).

EWC We use the version of EWC that accumulates Fisher information matrices as well as adds a quadratic loss term per task, as described in [26, 19, 27]. We implemented this method from scratch, following the description in the related papers. EWC has been previously compared to the unweighted L_2 regularization $\sum_j (\theta_j - \theta^*_{1:i-1,j})^2$. This is different to the L_2 variant described in this work, which has a per-parameter importance.

SI Synaptic Intelligence strategy was described in [73]. The original code released by the authors was adapted to the various datasets.

DM-I, II, III, IV This was proposed in Section 4.4, a strategy that directly regularizes the amount of forgetting, for L_1, L_2 and the elastic-net variant.

DM-I, II, III, IV with fine control This was proposed in Section 4.5, similar to the previous strategy but with additional hyperparameters that allow finer control over forgetting; evaluated for L_1, L_2 and the elastic-net variant.

EWC-p The "Fisher freezing" strategy described in Section 4.6; implemented on EWC.

Note that we do not evaluate joint training for classification as it has been previously investigated in the context of classification [34] and is known to be expensive as the number of tasks grow.

Method	P-MNIST	S-MNIST	Sim-EMNIST	CIFAR100	Sim-CIFAR100
Baseline	55.6 (1.04)	63.4 (0.38)	75.4 (1.15)	37.5 (5.74)	76.2 (0.49)
EWC	93.9 (0.30)	70.8 (2.65)	89.6 (2.99)	61.7 (2.41)	83.8 (2.06)
SI	92.6 (0.75)	78.3 (2.65)	91.4 (1.21)	62.1 (1.34)	83.7 (0.88)
EWC-p	94.5 (0.26)	72.0 (2.80)	89.2 (2.95)	65.4 (3.36)	**85.0 (1.35)**
DM-L_1 (best)	95.0 (0.30)	77.2 (1.96)	88.6 (2.97)	65.6 (4.88)	84.3 (1.64)
DM-E0.5 (best)	95.0 (0.23)	**80.3** (2.09)	88.9 (1.93)	**65.9** (2.15)	84.4 (0.97)
DM-L_2 (best)	94.3 (0.32)	71.2 (3.43)	89.5 (2.64)	61.2 (2.07)	83.6 (0.60)
DM-L_1 (best, fine)	**95.1** (0.13)	80.0 (1.88)	**93.0** (0.76)	65.8 (4.80)	83.9 (1.77)
DM-E0.5 (best, fine)	95.0 (0.19)	**80.3** (2.10)	91.6 (1.06)	65.2 (1.81)	84.5 (0.97)
DM-L_2 (best, fine)	94.4 (0.29)	69.0 (0.85)	89.4 (2.89)	61.2 (2.07)	83.9 (2.04)

Table 6.1: Mean and std final average validation accuracies (%) across 5 seeds for the best hyperparameters for all methods described in Section 6.1 and for all the datasets mentioned in Section 6.1.2. The details of the hyperparameter search are mentioned in Section 6.1.1. Methods are ranked by their mean validation accuracy across 5 seeds. The best results came from different cases (I-IV) with no conclusive trend, as reported in Section 6.1.3

6.1.1 Training Methodology

Training for each strategy is performed on feedforward ReLU networks with 2 hidden layers ($h = 128, \eta = 0.0001$), for 20 epochs. For hyperparameter search, we evaluate all methods on a single random seed, then choose a parameter that has the highest average validation accuracy. The final results (mean and standard deviation) are averaged over 5 seeds, using the best parameter. Table 6.1 shows the performance (final average validation accuracy) of the proposed methods.

We use the Adam optimizer for our experiments. Constants searched for EWC include $\lambda \in \{1, 10^1, 10^2, 10^3, 10^4\}$. For DM-I, II, III, IV (with and without finer control), we searched for $\lambda \in \{1, 10^1, 10^2, 10^3, 10^4\}$, $c^{(1)} \in \{0.025, 0.05, \cdots 0.10\}$ and $c^{(2)} \in \{0.0, 0.025, 0.05, \cdots 0.10\}$. For the CIFAR100 datasets, we searched for $\lambda \in \{1, 10^1, 10^2, \cdots 10^7\}$. For EWC-p, we searched for $p \in \{0.1, 0.2, 0.3, \cdots 0.9\}$. For SI, we searched for $c \in \{0.01, 0.1, 0.5, 1, 2\}$ and $\zeta \in \{0.001, 0.01, 0.1, 1\}$.

For the CIFAR100 experiments we use the embeddings from a pretrained Resnet-v1 model which achieves $\approx 70\%$ accuracy on the 100-class classification.

6.1.2 Datasets

We evaluate on the following datasets:

Permuted MNIST 5 tasks; first task corresponds to classifying the MNIST dataset of handwritten digits, and every other task corresponds to classifying images generated by permuting the pixels of the first task according to a fixed random permutation (fixed for the entire task); used in [28, 73, 47, 33].

Split MNIST 5 tasks, where every task is a 2-class classification. The tasks are labels 0/1, 2/3, 4/5, 6/7, and 8/9 from the MNIST dataset; used in [3, 69].

Similar EMNIST Hand-picked labels from the EMNIST dataset such that the classification tasks look approximately similar; 4 tasks, 3-class classification, task labels are 2/O/U, Z/8/V, 7/9/W, and T/Q/Y.

CIFAR100 Realistic image dataset with 5 tasks; 3-class classification; task labels are 0/1/2, 3/4/5, 6/7/8, 9/10/11, and 12/13/14.

Similar CIFAR100 As CIFAR100, but tasks are chosen from superclasses such that the labels per task correspond to the coarse classes. We chose the coarse classes to be "aquatic mammals", "food containers" and "household furniture". Since each coarse class contains 5 superclasses, this corresponds to 3-class classification spanning over 5 tasks.

Two of the datasets of our choice are similar continual datasets. The effect of task similarity has been discussed previously [24], but these discussions consider permutated data to be similar tasks. On the other hand, our investigation considers different datasets to be similar if the labels draw from some common superclass distribution of data. This is of practical significance since in real world classification systems, we usually desire our classification ability to persist when the newer classification tasks are label-wise similar to the previous data. For example, we expect a classifier that classifies between cars and motorbikes to easily (continually) learn the distinction between trucks and bikes.

Additionally, we note that a lot of the work in continual learning is evaluated on incremental classes, but with incremental classes the network is expected to remember just the labels that have been seen over multiple tasks. However, remembering multi-class classification requires remembering the differences between all the classes in each task. This is our rationale for the choice of few-class classification spanning over 4 or 5 tasks.

6.1.3 Results

Our numerical results are given in Table 6.1 and 6.2. We report the following insights.

Baselines and Existing Approaches As expected, the baselines for all the datasets incur catastrophic forgetting. This forgetting is less for the similar datasets Sim-EMNIST and Sim-CIFAR100, because the classification tasks are related, that is, learning the first task is enough to perform decently on following tasks. EWC and SI significantly improve upon the baseline accuracies.

DM-L_1 vs DM-L_2 We find that in almost all the datasets (an exception being Sim-EMNIST) the L_1 variant finds a better overall solution over multiple tasks, compared to the L_2 variant. In the case of fine control, this still holds on almost all the datasets, but the exception is Sim-CIFAR100. Note that even in the exceptional cases, the mean accuracies differ by a very small amount.

Fine Control We expect the fine-control version of DM-L_1 and DM-L_2 to perform the best in all datasets, because with the correct hyperparameters it can find the optimum which jointly maximizes both plasticity and stability. We indeed observe this for the grayscale datasets, where the improvements are good, but this does not hold in the case of the CIFAR100 datasets. We speculate that this is because of the relatively coarse granularity of the hyperparameter search for $c^{(1)}$ and $c^{(2)}$. Since the λ search for the CIFAR100 datasets was already computationally expensive, we chose not to repeat the search with finer granularity. Nevertheless, the best values for the fine methods are still better than EWC and SI.

Relative Retention in Cases I–IV Given the variance in our results, we find no clear empirical order of retention using DM-L_1. While each case responds to λ in a different

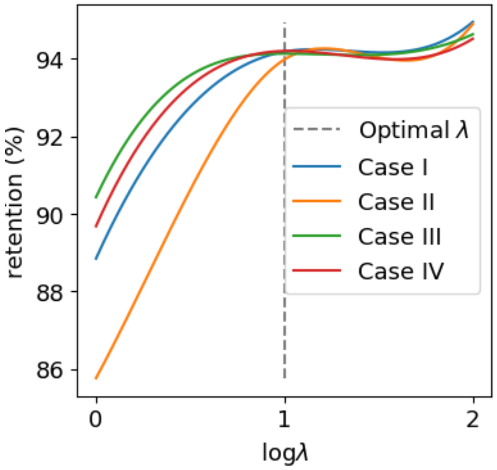

Figure 6.1: Average performance (smoothed) of Cases I–IV on Sim-EMNIST; 2 tasks with DM-L_1, 5 seeds ($\lambda \in [1, 10^2]$). Retention is defined as the percentage of classifier predictions on task 1 that stay the same after task 2 has been trained.

way, we observe that at optimal λ, the cases have approximately equal retention across all datasets. We illustrate this in Figure 6.1 for Sim-EMNIST, for two tasks.

The Effect of Freezing Important Weights Compared to EWC, EWC-p always produces an improved solution, except in the case of Sim-EMNIST, where it remains close to EWC. It outperforms the other methods for Sim-CIFAR100. As can be seen in Table 6.2, we note that for the grayscale datasets the best degree of preservation p is between $20\% - 40\%$, while for the realistic image datasets, the best degree of preservation is around $60\% - 80\%$. This is in agreement with the strength of the stability bias λ for the realistic image datasets for EWC, which are also high, meaning that the realistic images require stronger preservation to register improvements.

6.2 Experiments for RL

In this section, we report our results for the Continual RL algorithms proposed in Chapter 5. Specifically, we evaluate PPO-EWC (Algorithm 3) for different stability biases δ against PPO-Baseline (Algorithm 2) and joint training. Our environments are based on a version of WiseMove [31], a hierarchical framework to investigate safe reinforcement learning, using incremental learntime verification of temporal logic constraints.

Dataset	**Degree of Preservation** (p)
P-MNIST	40%
S-MNIST	40%
Sim-EMNIST	20%
CIFAR100	80%
Sim-CIFAR100	60%

Table 6.2: Optimal degree of preservation p for evaluated datasets, where p refers to the percentage of weights that are frozen for all tasks after the first task.

6.2.1 Environments

We use a curriculum of four environments, relating to control in an autonomous driving setting. These environments increase in difficulty. In all these environments, the vehicle to be controlled (ego) starts from the left side and has to travel forward, without colliding into any other vehicle and staying on the road, preferably within the lane. Other vehicles are stopped (parked). More specific details are as follows.

`OneStoppedCarOpposite` (Figure 6.2) There is one stopped car on the opposite lane as ego. The goal is to avoid any crashes with this car, and reach the rightmost end, in the same lane.

`OneStoppedCar` (Figure 6.3) There is one stopped car on the same lane as ego. The goal is to avoid any crashes with this car, preferably by changing lane and reach 10 metres beyond this car.

`TwoStoppedCars` (Figure 6.4) There are two stopped cars on the same lane as ego. The goal is to avoid any crashes with any of the cars, preferably by changing lane and reach 10 metres beyond the last car.

`ThreeStoppedCars` (Figure 6.5) There are two stopped cars on the same lane as ego, followed by a car on the opposite lane. The goal is to avoid any crashes with any of the cars, preferably by changing lanes (twice) and reach 10 metres beyond the last car.

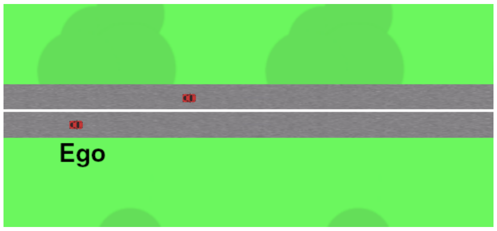

Figure 6.2: `OneStoppedCarOpposite`

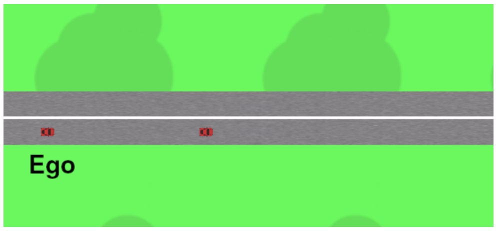

Figure 6.3: `OneStoppedCar`

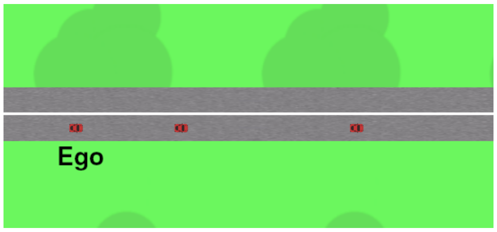

Figure 6.4: `TwoStoppedCars`

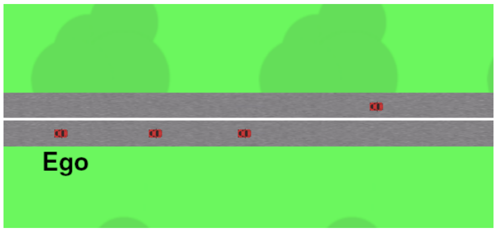

Figure 6.5: `ThreeStoppedCars`

Reward Specification The reward is specified as follows. A collision results in a -100 reward. Going out of the road boundaries also results in a -100 reward. Stopping results in a -100 reward as well. Every timestep spent in the environment results in a -1 reward. The agent is expected to terminate successfully within $T_E = 150$ timesteps. Reaching the target location successfully results in a $100 + T_E$ reward.

State Space The state space for all the environments consists of 7 ego features and 6 propositional features, depending on the environment. The ego features are: x, y position coordinates, speed of the car v, magnitude of acceleration a, car angle θ, steering rate ψ and the change of steering rate $\dot{\psi}$. The propositional features (Table 6.3, 6.4) consist of boolean variables that encode specific details about the current state. Note that only the ego features are normalized.

Action Space The action space for all the environments corresponds to 13 discrete actions that can be used to control the car (Table 6.5). These discrete actions are various combinations of acceleration $a \in [-2, 2]$ and rate of change of steering angle $\dot{\psi} \in [-1, 1]$.

OneStoppedCarOpposite	OneStoppedCar	TwoStoppedCars	ThreeStoppedCars
stopped	stopped	stopped	stopped
out_of_bounds	out_of_bounds	out_of_bounds	out_of_bounds
near_goal	-	past_car_a	past_car_a
collided	-	past_car_b	past_car_b
-	past_car	collided_a	collided_a
-	collided	collided_b	collided_b

Table 6.3: Boolean propositional features for the environments in curriculum, in the order that they are appended to the feature vector. A blank proposition refers to a dummy feature, added to ensure that the dimensionality of the state space is constant across the environments.

Feature	**Description**
stopped	True when ego is stopped.
out_of_bounds	True when ego goes outside the two horizontal lanes.
near_goal	True when ego reaches the rightmost end of the road.
collided*	True when ego collides with another vehicle.
past_car*	True when ego goes 10 distance units beyond a specific other vehicle.

Table 6.4: Description of the boolean propositional features.

Simplifications In all the following graphs, x-axis denotes training progress. Different seeds converge in different number of time-steps, so we try for 200k steps per environment and obtain 100 time points per training procedure. Further, 10 seeds are tried, but the algorithm halts when the algorithm is unable to learn an environment in the continual sequence. Actual rewards are in the [-100, 100] range, but to understand how many environments are remembered at any instant, we normalize these rewards to [0, 1].

6.2.2 Joint Training

We jointly train by interleaving environment evaluation in PPO (Algorithm 1), switching between all the environments uniformly. Joint training is supposed to have the best performance given enough training time. We try joint training with 2, 3, 4 environments for 400k, 600k and 800k time steps, which is equivalent training time for continual learning with 2, 3 and 4 environments. We report the average and individual performance curves during training in Figures 6.6 and 6.7.

Action	Description
`[amax, psidotmax]`	Turn left (max steering intensity) with max acceleration.
`[amax, 0.75*psidotmax]`	Turn left (75% steering intensity) with max acceleration.
`[amax, 0.50*psidotmax]`	Turn left (50% steering intensity) with max acceleration.
`[amax, 0.25*psidotmax]`	Turn left (25% steering intensity) with max acceleration.
`[amax, 0]`	Move forward (no steering) with max acceleration.
`[amax, -0.25*psidotmax]`	Turn right (25% steering intensity) with max acceleration.
`[amax, -0.50*psidotmax]`	Turn right (50% steering intensity) with max acceleration.
`[amax, -0.75*psidotmax]`	Turn right (75% steering intensity) with max acceleration.
`[amax, -psidotmax]`	Turn right (max steering intensity) with max acceleration.
`[0.50*amax, 0]`	Move forward (no steering) with 50% of max acceleration.
`[0, 0]`	No acceleration or steering.
`[-0.50*amax, 0]`	Decelerate; 50% of max deceleration.
`[-amax, 0]`	Max deceleration.

Table 6.5: Description of discrete actions that make up the action space. Constants used are `amax = 2` and `psidotmax = 1`.

We observe that joint training is unable to reach configurations that solve all the environments satisfactorily. With more training time, joint training should be able to achieve successes across all environments.

6.2.3 PPO-Baseline vs PPO-EWC

Next, we compare the performance of PPO-Baseline (Algorithm 2) with PPO-EWC (Algorithm 3). In PPO-EWC, the stability update is applied every 8 training epochs ($\delta = 8$). We report the average and individual performance curves during training in Figures 6.8, 6.9, 6.10, and 6.11.

We observe that the stability update interferes with the exploration strategy, which corresponds to a dip in performance at the beginning of training a new task.

Interestingly, PPO-Baseline is able to (accidentally) find a configuration that solves all the environments, although only for one seed. PPO-EWC is able to solve all the environments across more number of seeds (not all seeds).

The optimal λ for our setting seems to be $\lambda = 1$. With higher λ, the stability update interferes a lot with the training, and is sometimes even unable to solve beyond 2 environments.

6.2.4 The Effect of δ

In Section 5.3, we proposed applying the stability update every δ number of training epochs, to weaken the stability bias. To understand the effect of δ, we plot the average number of environments solved across 20 seeds by PPO-EWC against different λ, for different values of δ (Figure 6.12). We try $\delta = 1, 2, 4, 8$.

We observe that sparsely applying the stability update results in a better performance beyond the optimal λ. Specifically, $\delta = 8$ performs better than applying the stability update at every training epoch, that is $\delta = 1$. There is also a downward trend showing that the number of environments solved decreases with higher λ, but this trend only holds beyond the stability plasticity optimum, which is observed to be near $\lambda = 1$. For λ less than the optimum, there is no clear trend.

Additionally, we find that PPO-Baseline is only able to solve 1.4 environments on average across 10 seeds, which is much lower than PPO-EWC.

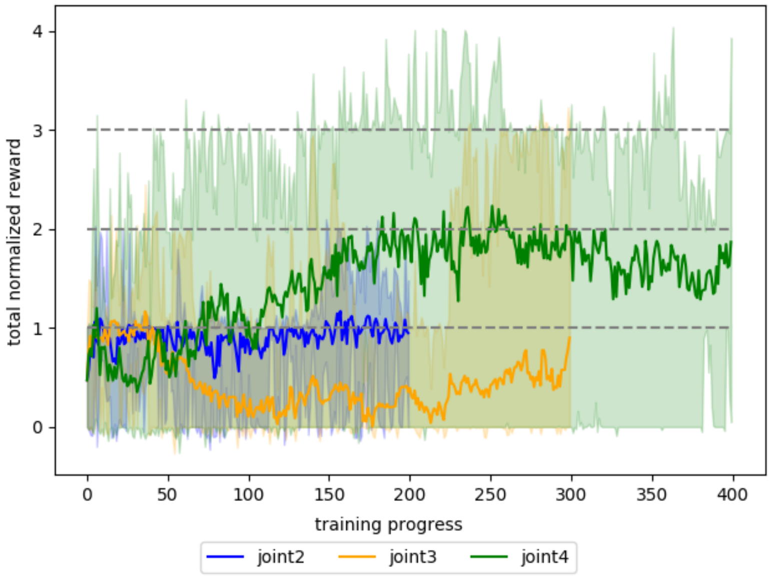

Figure 6.6: Average performance for joint training with 2, 3, 4 environments.

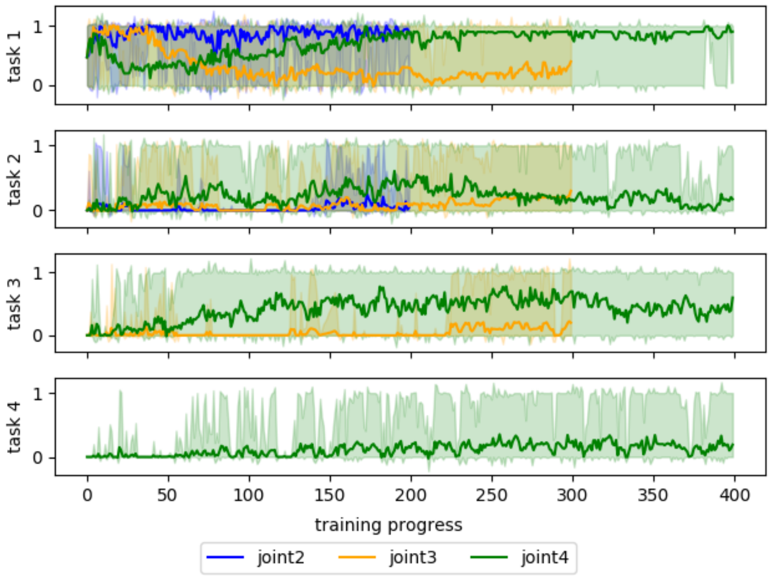

Figure 6.7: Taskwise performance for joint training with 2, 3, 4 environments.

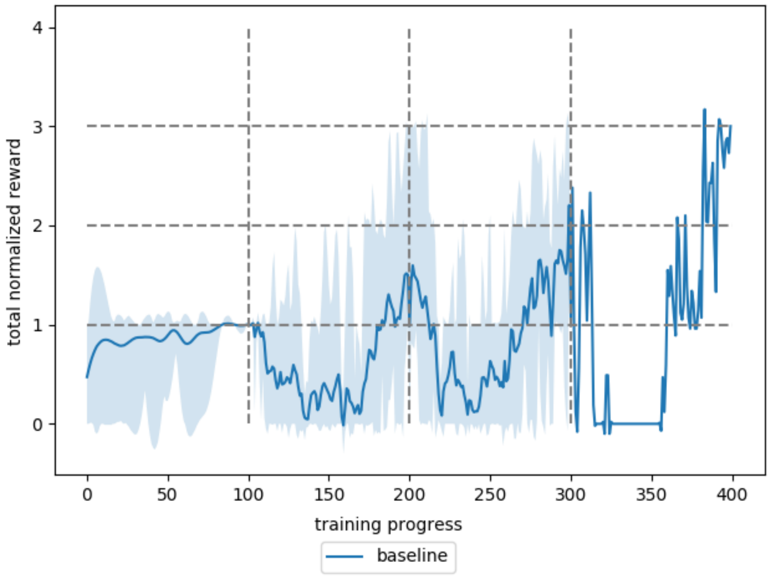

Figure 6.8: Average performance for PPO-Baseline with 4 environments.

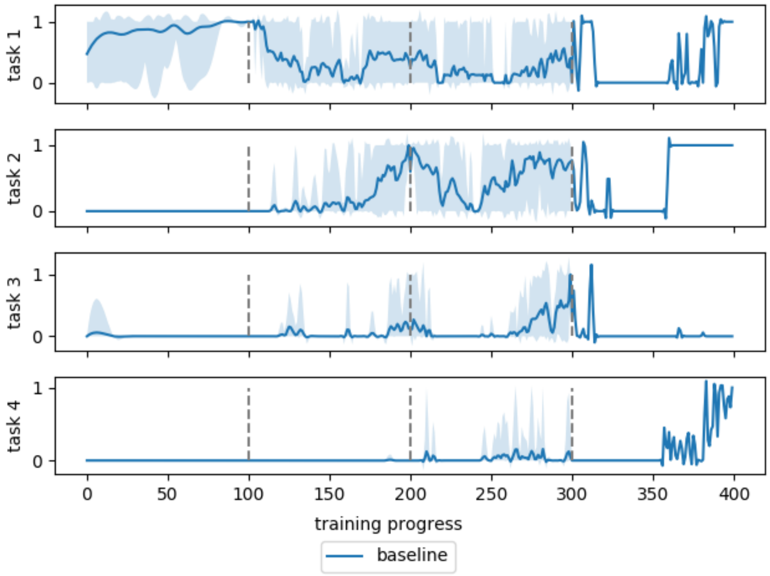

Figure 6.9: Taskwise performance for PPO-Baseline with 4 environments.

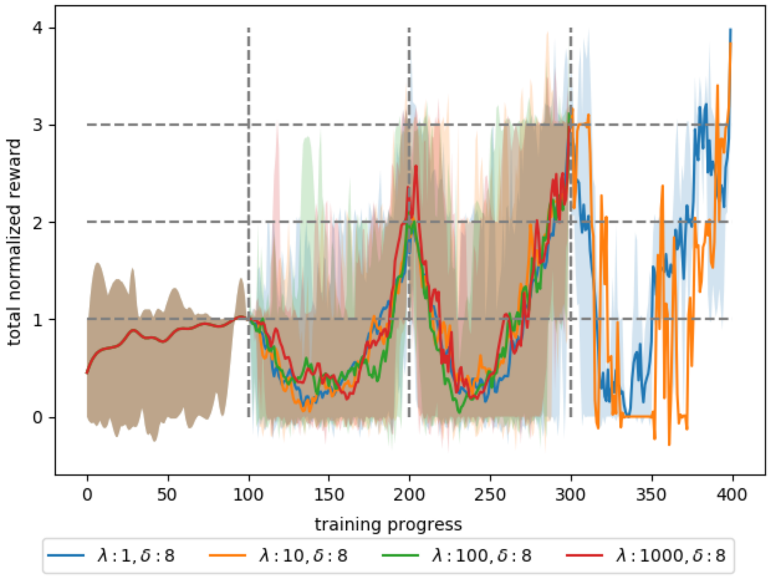

Figure 6.10: Average performance for PPO-EWC with 4 environments.

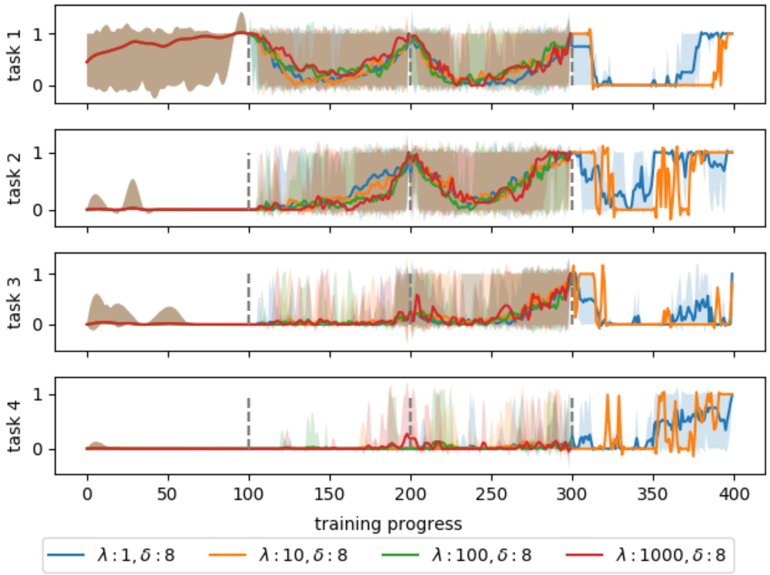

Figure 6.11: Taskwise performance for PPO-EWC with 4 environments.

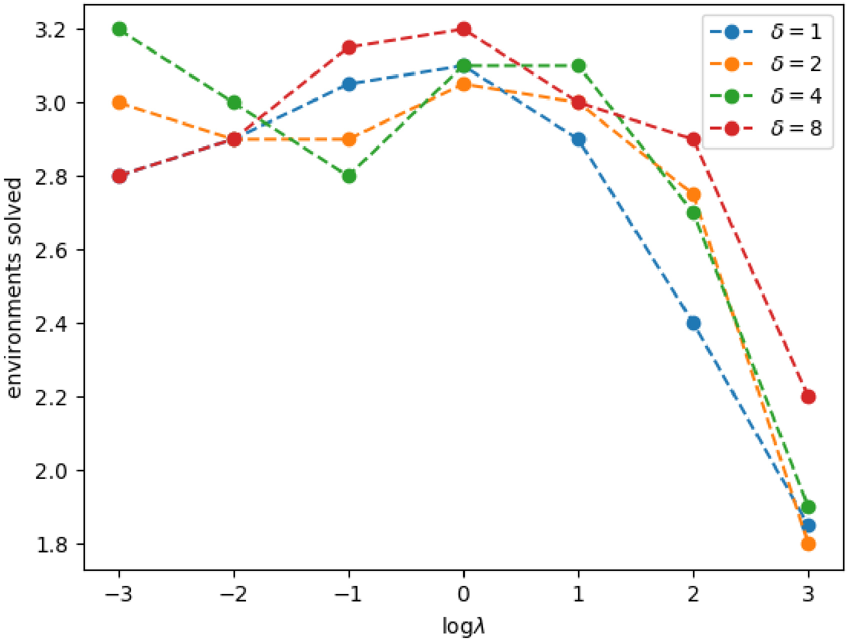

Figure 6.12: Number of environments solved against various values of λ, for $\delta = 1, 2, 4, 8$.

Chapter 7

Conclusions

In the context of real world classification systems, catastrophic forgetting may not just cause performance deterioration, but also cause a loss of validated knowledge. Existing (regularization) strategies to mitigate catastrophic forgetting typically minimize the elastic criterion, but do not explicitly prioritize stability over plasticity.

In this thesis, we explored alternative strategies for continual learning and investigated the effect of altering the stability bias.

- We formulated the continual learning problem to directly minimize an approximate upper bound on the absolute amount of forgetting, which led to a weaker bias for continual classification.
- We proposed three applications of this approximation (DM, DM-fine, EWC-p). We further demonstrated different variants of network preservation corresponding to different system requirements (Cases I-IV).
- We evaluated the proposed methods for continual classification against popular approaches in continual learning (EWC, SI) on various grayscale and colored datasets.
- We adapted EWC to an actor-critic style algorithm (PPO), and investigated the effect of sparsely applying the stability update, to weaken the bias. We compared our method to a baseline and the joint training approach.

As evidenced by our experiments, we observe that weakening the stability bias can lead to better preservation of past knowledge, and often produce solutions with better performance.

We see several possible directions for future research. First, this work considers only informal notions of stability, plasticity and knowledge retention, so it would useful to quantify these more formally. Second, despite its very slow convergence, it would be interesting to pursue the comparison of DQN-EWC [26] with our proposed base RL algorithm, PPO-EWC. Finally, the differential equations for the gradient descent dynamics using the L_1 and the L_2 updates can be better examined to concretely understand the nature of the obtained solutions.

References

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[2] Glen Berseth, Cheng Xie, Paul Cernek, and Michiel Van de Panne. Progressive reinforcement learning with distillation for multi-skilled motion control. *arXiv preprint arXiv:1802.04765*, 2018.

[3] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018.

[4] Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.

[5] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[6] Andrew W Ellis, Lambon Ralph, and A Matthew. 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, 2000.

[7] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[8] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[9] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[10] Tommaso Furlanello, Jiaping Zhao, Andrew M Saxe, Laurent Itti, and Bosco S Tjan. Active long term memory networks. *arXiv preprint arXiv:1606.02355*, 2016.

[11] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[12] Gerd Gigerenzer and Henry Brighton. Homo heuristicus: Why biased minds make better inferences. *Topics in Cognitive Science*, 1(1):107–143, 2009.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[14] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[15] Stephen Grossberg. Adaptive resonance theory: How a brain learns to consciously attend, learn, and recognize a changing world. *Neural Networks*, 37:1–47, 2013.

[16] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

[17] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[18] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[19] Ferenc Huszár. Note on the quadratic penalties in elastic weight consolidation. *Proceedings of the National Academy of Sciences*, page 201717042, 2018.

[20] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *Schedae Informaticae*, 25:49–59, 2016.

[21] Heechul Jung, Jeongwoo Ju, Minju Jung, and Junmo Kim. Less-forgetting learning in deep neural networks. *arXiv preprint arXiv:1607.00122*, 2016.

[22] Christos Kaplanis, Murray Shanahan, and Claudia Clopath. Continual reinforcement learning with complex synapses. *arXiv preprint arXiv:1802.07239*, 2018.

[23] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.

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

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[27] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Reply to huszár: The elastic weight consolidation penalty is empirically valid. *Proceedings of the National Academy of Sciences*, 115(11):E2498–E2498, 2018.

[28] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 114 13:3521–3526, 2016.

[29] Solomon Kullback. *Information theory and statistics.* Courier Corporation, 1997.

[30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[31] Jaeyoung Lee, Aravind Balakrishnan, Ashish Gaurav, Krzysztof Czarnecki, and Sean Sedwards. Wisemove: A framework to investigate safe deep reinforcement learning

for autonomous driving. In David Parker and Verena Wolf, editors, *Quantitative Evaluation of Systems*, pages 350–354, Cham, 2019. Springer International Publishing.

[32] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in neural information processing systems*, pages 4652–4662, 2017.

[33] Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *International Conference on Machine Learning*, pages 3925–3934, 2019.

[34] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[35] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[36] Xialei Liu, Marc Masana, Luis Herranz, Joost Van de Weijer, Antonio M Lopez, and Andrew D Bagdanov. Rotate your networks: Better weight consolidation and less catastrophic forgetting. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2262–2268. IEEE, 2018.

[37] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

[38] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[39] Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. *Neural Networks*, 116:56–73, 2019.

[40] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[41] M. W. McCloskey. Catastrophic interference in connectionist networks: The sequential learning problem" the psychology. 1989.

[42] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[43] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4:504, 2013.

[44] Kieran Milan, Joel Veness, James Kirkpatrick, Michael Bowling, Anna Koop, and Demis Hassabis. The forget-me-not process. In *Advances in Neural Information Processing Systems*, pages 3702–3710, 2016.

[45] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[46] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[47] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *International Conference on Learning Representations*, 2018.

[48] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 2019.

[49] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[50] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2847–2854. JMLR. org, 2017.

[51] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[52] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[53] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[54] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[55] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[56] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[57] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[58] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4535–4544, 2018.

[59] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pages 4555–4564, 2018.

[60] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[61] Mark A Smith, Garrison W Cottrell, and Karen L Anderson. The early word catches the weights. In *Advances in neural information processing systems*, pages 52–58, 2001.

[62] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[63] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[64] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* 2018.

[65] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4496–4506, 2017.

[66] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[67] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[68] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003, 2016.

[69] Felix Wiewel and Bin Yang. Localizing catastrophic forgetting in neural networks. *arXiv preprint arXiv:1906.02568*, 2019.

[70] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5279–5288, 2017.

[71] Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, Yuxin Peng, and Zheng Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 177–186. ACM, 2014.

[72] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[73] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR. org, 2017.

[74] Guanyu Zhou, Kihyuk Sohn, and Honglak Lee. Online incremental feature learning with denoising autoencoders. In *Artificial intelligence and statistics*, pages 1453–1461, 2012.

[75] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.