Continual Learning and Forgetting in Deep Learning Models

by

Alaa El Khatib

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2020

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Reda Alhajj Dept. of Computer Science, University of Calgary
Supervisor:	Fakhri Karray Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member:	Mark Crowley Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member:	Zhou Wang Dept. of Electrical and Computer Engineering, University of Waterloo
Internal-External Member:	Alexander Wong Dept. of Systems Design Engineering, University of Waterloo

Author's Declaration

Statement of Contributions

The work presented in this thesis was done under the supervision of Dr. Fakhri Karray. Chapter 3 is based on a journal paper that is under review. Chapter 4 is based on a published conference paper (El Khatib and Karray, 2019a) and an extended journal paper, which is under review. Chapter 5 is based on a published conference paper (El Khatib and Karray, 2019b) and another paper in preparation. I was the primary author of the aforementioned papers, all of which were co-authored with my supervisor.

Abstract

Continual learning is a framework of learning in which we aim to move beyond the limitations of standard isolated optimization of deep learning models toward a more intelligent setting, where models or agents are able to accumulate skills and knowledge, across diverse tasks and over extended periods of time, much like humans do. Like much of neural networks research, interest in continual learning has ebbed and flowed over the decades, and ultimately saw a sharp increase over the past few years, buoyed by the successes of deep learning thus far.

One obstacle that has dominated continual learning research over the years is the so-called *catastrophic forgetting* phenomenon, which refers to the tendency of neural networks to "forget" older skills and knowledge as soon as they are subsequently optimized for additional tasks. Researchers have proposed various approaches to counter forgetting in neural networks. In this dissertation, we review some of those approaches and build upon them, and address other aspects of the continual learning problem.

We make the following four contributions.

First, we address the critical role of importance estimation in fixed-capacity models, where the aim is to find a balance between countering forgetting and preserving a model's capacity to learn additional tasks. We propose a novel unit importance estimation approach, with a small memory and computational footprint. The proposed approach builds on recent work that showed that the average of a unit's activation values is a good indicator of its importance, and extends it by taking into consideration the separation between class-conditional distributions of activation values.

Second, we observe that most methods that aim to prevent forgetting by explicitly penalizing changes to parameters can be seen as *post hoc* remedies that ultimately lead to inefficient use of model capacity. We argue that taking into account the continual learning objective requires a modification to the optimization approach from the start rather than only after learning. In particular, we argue that key to the effective use of a model's capacity in the continual learning setting is to drive the optimization process toward learning more general, reusable, and thus durable representations that are less susceptible to forgetting. To that end, we explore the use of supervised and unsupervised auxiliary tasks as regularization, not against forgetting, but against learning representations that narrowly target any single classification task. We show that the approach is successful at mitigating forgetting, *even though it does not explicitly penalize forgetting.*

Third, we explore the effect of inter-task similarity in sequences of image classification tasks on the overall performance of continual learning models. We show that certain models are adversely affected when the learned tasks are dissimilar. Moreover, we show that, in those cases, a small replay memory, even 1% the size of the training data, is enough to significantly improve performance.

Fourth and lastly, we explore the performance of continual learning models in the so-called multi-head and single-head settings and approaches to narrow the gap between the two settings. We show that unlabelled auxiliary data, not sampled from any task in the learning sequence, can be used to improve performance in the single-head setting.

We provide extensive empirical evaluation of the proposed approaches and compare their performance against recent continual learning methods in the literature.

Acknowledgements

I would like to thank my supervisor, Dr. Fakhri Karray, for his support throughout my PhD studies. I appreciated the freedom I had to explore my research topic and the opportunities he gave me to gain experience while working on my thesis.

I would also like to thank my committee members, Dr. Mark Crowley, Dr. Alexander Wong, Dr. Zhou Wang, and Dr. Reda Alhajj for their feedback and their time. I also thank my former committee member Dr. Dana Kulic for her feedback on the early proposal for this work.

I am grateful to my friends and colleagues in CPAMI, Chaojie, Arief, Mahmoud, Jany, and everyone else, for their companionship, and grateful to have had the chance to meet and know Sigong Zhang, Min Meng, and Mohammad Al-Sharman during my PhD years. And I am thankful for all the people I met, no matter how briefly, who made life just a little bit more bearable. I am also thankful for my vanishing friend, Haytham. May he make a reappearance at some point.

Finally, and in the tradition of saving the best for last, I am grateful to my family—my mother and late father, my brothers and sisters—for their support and all the sacrifices they have made for me to be where I am.

Dedication

To the once big heart, big no more.

To Alaa in alternate histories and the lives that could have been.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AGI	Artificial General Intelligence 16
CNN	Convolutional Neural Networks 11
EWC	Elastic Weight Consolidation 23
FLDA	Fisher Linear Discriminant Analysis 35
GAN	Generative Adversarial Network 20
KL	Kullback-Leibler 4
LwF	Learning without Forgetting 4
NLL	Negative Log-Likelihood 13
ReLU	Rectified Linear Unit 8
RWalk	Riemannian Walk 29
SGD	Stochastic Gradient Descent 14

Chapter 1

Introduction[1]

Over the past decade, deep learning has seen many success stories in applications ranging from image classification (Krizhevsky et al., 2012) and segmentation (Chen et al., 2018) to speech recognition (Chorowski et al., 2015; Amodei et al., 2016) and machine translation (Bahdanau et al., 2015), in some cases with models rivalling human performance. These achievements, however, are almost always confined to narrowly defined, isolated applications. The ability to design models that can rival human performance in any task—what is commonly known as artificial general intelligence—remains an elusive goal. Consider, for example, a human's capacity for seemingly endless learning, their ability to accumulate skills and knowledge and learn to recognize thousands of faces, objects, words, etc., over a lifespan of decades. This capacity for *continual learning* is, as of yet, something we cannot replicate—not even remotely—in deep learning models. Partly to blame for this is a phenomenon known as catastrophic forgetting.

Catastrophic forgetting refers to the tendency of a neural network to "forget", or degrade in performance on, a task once it has been subsequently optimized for another. This implies that, absent a remedy for catastrophic forgetting, continual learning in neural networks, unlike in humans, is a process of overwriting previous learning rather one of accumulating knowledge. The ability, however, to accumulate knowledge stands to serve many practical applications.

[1]Parts of this chapter are adapted from (El Khatib and Karray, 2019a), © IEEE 2019.

1.1 Motivation

In general, continual learning—and thus countering catastrophic forgetting—is important in settings where there are constraints on the availability of training data and the way they are presented to a model. Consider, for example, a social robot tasked with assisting a human at home. Such a robot may be pre-loaded with classes of objects it can recognize (*e.g.*, chair, door, etc.), such that, as it moves around, it can "understand" its surroundings. We may be interested, however, to endow such a robot with the ability to learn to identify new classes of objects of relevance in the specific environment in which it operates. To that end, we could design the robot such that it builds and trains a new model for every new object class. Or, we could program it to store all training images for all classes, pre-loaded and new, and re-train a single multi-class classification model whenever it learns a new class. Both these strategies, however, are inefficient. The most efficient solution, both in terms of memory and computational costs, is to be able to update a single model to account for new object classes without significantly affecting its ability to recognize previously learned object classes. One can think of similar scenarios in other domains as well.

1.2 Scope

In this work, we consider continual learning of image classification tasks. That is, the ability of a neural network to learn multiple image classification tasks sequentially, and continue to perform well on all tasks learned, without access to the training data associated with old tasks. Moreover, we limit our discussion to feed-forward neural networks, in particular convolutional neural networks, optimized by gradient descent. Although the algorithms discussed and proposed are potentially applicable to other domains (*e.g.*, reinforcement learning tasks), the experiments and results we provide are restricted to this scope.

1.3 Contributions

This dissertation contains four contributions:

- A class separation-based importance estimation method, described in Chapter 3;
- An approach to countering forgetting by encouraging durable representations, described in Chapter 4;

- An analysis of the effect of inter-task similarity on the performance of continual learning models, described in Section 5.1; and
- An approach to narrowing the gap between single-head and multi-head continual learning models, described in Section 5.2.

We will describe in detail the contributions in the coming chapters. Below, however, is a brief summary of each.

1.3.1 Importance Estimation

Catastrophic forgetting is countered in the literature through different strategies. Rooted in the stability-plasticity dilemma (Grossberg, 1987), many approaches address catastrophic forgetting through a process of slowing down learning, masking, or otherwise consolidating parameters that are estimated to be important to previously learned tasks, while freeing up unused capacity to learn new tasks. Most such methods entail a process of estimating a set of values that quantify the relative "importance" of the parameters of the model. Until recently, most importance estimation methods were based in some form on the local sensitivity of the loss function to parameter changes (*i.e.*, the gradient of the loss function) (Chaudhry et al., 2018; Kirkpatrick et al., 2017; Zenke et al., 2017; Aljundi et al., 2019b).

Jung et al. (2020) recently showed that the average of activation values is an effective estimate of *unit* importance. The estimate has the appeal of simplicity (*e.g.*, it does not involve gradient computations and, hence, can be computed with forward passes only), but tends to over-estimate importance. By that we mean that it occasionally assigns high importance values to unimportant units. This leads to inefficient use of model capacity, with parameters being unnecessarily constrained, which in turn prevents the model from further learning. Jung et al. (2020) address this flaw by imposing an explicit sparsity penalty.

In this work, we build on the importance estimation method of Jung et al. (2020). We show that in layers closer to the classification layer, units tend to specialize, and class-conditional distributions of unit activations become more separable. Based on this, and rather than using the overall average of activations, we propose a class separation-based importance estimation method that draws on the ideas of Fisher discriminant linear analysis (Murphy, 2012, p. 274), as well as on the work on deeply supervised net by Lee et al. (2015). We show that the proposed approach is more efficient in its use of model capacity and performs well on a wide array of image classification tasks, outperforming a number of recent methods in certain cases.

This work is described in further details in Chapter 3. As of this writing, a journal paper based on this work is in preparation.

1.3.2 Learning Durable Representations

The methods we cited so far fall under what we call regularization-based approaches to countering forgetting (see Chapter 2.4). One could argue that most regularization-based approaches in the literature constitute a *post hoc* effort to prevent forgetting, in that only after the task is learned do we begin to consider preserving learning. At this late stage, of course, we can only resort to explicit penalties on changes to parameters.

We attempt in this work to preempt forgetting, as it were, by encouraging the continual learning model to learn *durable* representations, that are less susceptible to forgetting in and of themselves. In particular, we argue that the ability to learn rich, reusable representations for each task in the first place is just as crucial for efficient continual learning as the ability to safeguard learned representations *post hoc*. Moreover, we argue that key to learning such representations is the ability to draw on the *content* of training data, images in our case, irrespective of how discriminative that content is to the current classification task—an ability characteristic of unsupervised learning.

We explore multiple ways to encourage such durable representations, and show that using auxiliary reconstruction tasks is an effective way to reduce forgetting *without an explicit penalty on it*. We show that the approach is competitive with recent *post hoc* methods (Kirkpatrick et al., 2017; Chaudhry et al., 2018). Moreover, using a Kullback-Leibler (KL) divergence-based measure (Murphy, 2012), we show that the use of auxiliary reconstruction tasks has the effect of reducing the representational changes from task to task.

This work is described in further details in Chapter 4. Early results for this work are published in the Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN) (El Khatib and Karray, 2019a). As of this writing, a journal paper with additional results is in preparation.

1.3.3 Inter-Task Similarity

In this work, we explore the effect of inter-task similarity on continual learning performance. We focus on the Learning without Forgetting (LwF) model (Li and Hoiem, 2018) and show that, while it performs well in general, its performance degrades significantly when the tasks in the sequence to be learned are dissimilar.

To overcome this degradation, we explore the use of a small replay memory, in conjunction with LwF, and show that even a memory size equivalent to only 1% of the training set size is enough to significantly improve performance in cases of low inter-task similarity. Moreover, we show that the proposed mechanism is applicable to other methods, such as elastic weight consolidation (Kirkpatrick et al., 2017).

This work is described in further details in Section 5.1. As of this writing, a paper based on this work is in preparation.

1.3.4 Single-Head Performance

Much of the continual learning literature deals with improving performance in the so-called "multi-head" setting. In this setting, each test sample presented to the model is accompanied by a task identifier and the model makes predictions over the set of classes in learned in a single task only. A more challenging setting, called the "single-head" does away with task identifiers, and the model has to make predictions over the union of all classes learned across all tasks.

Performance in the single-head setting is typically worse than in the multi-head setting. This is largely due to the fact that, learning the tasks in sequence (*i.e.*, not concurrently), the model never learns to discriminate between classes from different tasks. One way to narrow the gap between single-head and multi-head performance is to use a replay memory (Chaudhry et al., 2018). This, however, has a memory cost that grows with the number of tasks. In this work, we explore the use of unlabelled auxiliary data, not associated with the learned tasks, in lieu of a replay memory, and show that the approach improves the single-head performance.

This work is described in further details in Section 5.2 and is published in the Proceedings of the 2019 International Conference on Image Analysis and Recognition (ICIAR) (El Khatib and Karray, 2019b).

1.4 Organization

The rest of this dissertation is organized as follows. Chapter 2 presents an overview of neural networks, their building blocks, and their optimization as it pertains to the proposed methods and the experiments presented in subsequent chapters, in addition to introducing and reviewing the literature on continual learning and catastrophic forgetting. Chapters 3–5 describe the four contributions summarized above. Finally, Chapter 6 concludes the dissertation.

Chapter 2

Background and Related Work

2.1 Background on Deep Learning

This chapter provides a review of the basic building blocks of neural networks that will be used in the rest of this dissertation. It also introduces the continual learning framework and catastrophic forgetting and reviews the main directions of research, old and new, concerned with mitigating or circumventing catastrophic forgetting, as well as specific methods that have been shown to aid continual learning. In the coming chapters, we will gradually expand on and formalize many of the concepts that are only briefly and informally introduced here. The intent from this chapter is to set the scene, as it were. Readers familiar with the fundamentals of neural networks may want to start from Section 2.2.

2.1.1 Deep Models and Their Many Layers

The term “deep learning” has come to be used to refer to neural networks in general, which are parametric machine learning models characterized by a layered structure. Each layer is made up of a collection of “neurons”, or units, each of which, in turn, constitutes a linear or non-linear mapping. Although commonly motivated by analogy to the animal brain, neural networks, complex and structured as they may be, can be seen simply as non-linear functions mapping inputs to outputs.

Fully Connected Layers

Fully connected layers, sometimes called dense layers, perform linear transformations of the form:

$$\mathbf{y} = W^{\top}\mathbf{x}, \tag{2.1}$$

where W is an $m \times n$ weight matrix representing the layer's parameters, $\mathbf{x}$ is the $m \times 1$ input to the layer, and $\mathbf{y}$ is its $n \times 1$ output. Figure 2.1 depicts a fully connected layer.

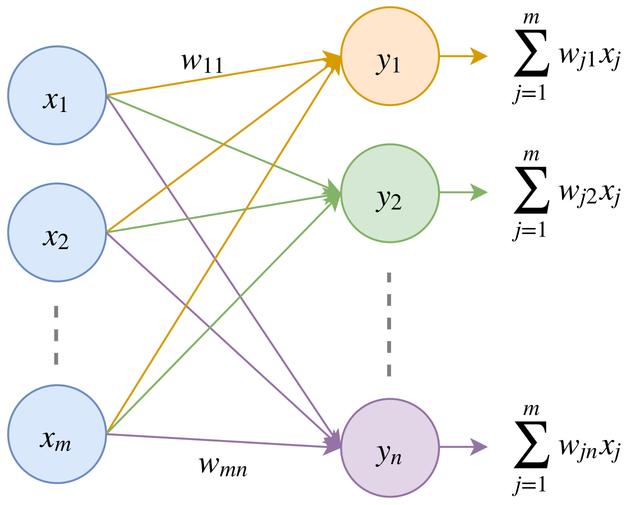
x_1
x_2
x_m
w_{11}
w_{mn}
y_1
y_2
y_n
$\sum_{j=1}^{m} w_{j1}x_j$
$\sum_{j=1}^{m} w_{j2}x_j$
$\sum_{j=1}^{m} w_{jn}x_j$

Figure 2.1: Fully connected layer.

Activation Layers

Stacking linear layers sequentially produces a still-linear overall mapping. Given the fact that real-world phenomena are seldom well represented with linear models, neural networks generally intersperse linear layers with non-linear activation layers. These activation functions are most commonly applied element-wise on the inputs. Until recently, the most common activation functions used were the sigmoid and the hyperbolic tangent. Nowadays, many have adopted the so-called Rectified Linear Unit (ReLU) activation (Goodfellow et al., 2016), given by $\max(0, x)$. Although not differentiable everywhere, the ReLU activation function has two important advantages: 1) it is computationally efficient, having a gradient that is either 1 or 0 almost everywhere; and 2) unlike the sigmoid and the hyperbolic tangent, the ReLU is non-saturating, allowing for stronger gradient signals during training.

That it kills negative inputs, and consequently the corresponding gradients flowing backwards, is a potential weakness of ReLU activation. The *leaky* ReLU is designed to alleviate this by attenuating, rather than totally eliminating, negative inputs. The output of a leaky ReLU is thus given by

$$\begin{cases} \alpha x, & \text{if } x < 0 \\ x, & \text{otherwise,} \end{cases} \tag{2.2}$$

where $0 < \alpha < 1$ is a hyperparameter.

Convolution Layers

Convolution layers (LeCun et al., 1989) can be seen as a restricted version of linear layers. In particular, convolution layers are characterized by local connectivity, in contrast to fully connected units, meaning each convolution unit is connected only to a local neighbourhood of its input (see Figure 2.2). For example, if the input is an image, a convolution unit, sometimes called a kernel or a filter, is connected only to a small region of it at a time. Spatially sampled data, such as images, exhibit local structure (*e.g.*, pixels close together in an image tend to be more correlated than those far apart) that can be exploited by such locally connected units. In addition to local connectivity, convolution units are constrained by what is known as parameter tying. One way to understand this is as if there are replicas of each kernel across different regions of the input. Alternatively, we could interpret these replicas as a single kernel "sliding" across the input. Such a sliding kernel (or a set of replicated kernels collectively) can be seen as learning a detector for a single feature (*e.g.*, an eye) that can appear across the input (*e.g.*, an image).

Mathematically, the convolution of two functions x and w defined over the same domain t is given by

$$\int_{-\infty}^{\infty} x(\tau)w(t-\tau)d\tau, \tag{2.3}$$

or, in discrete domains

$$\sum_{\tau=-\infty}^{\infty} x[\tau]w[t-\tau]. \tag{2.4}$$

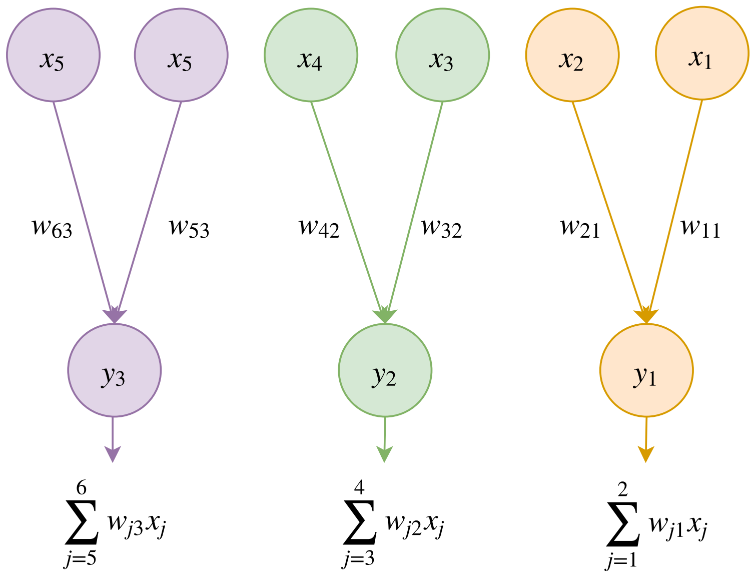

Figure 2.2: Locally connected layer. If the 3 output units are constrained to have tied parameters, then collectively they are equivalent to a single convolution unit.

The term convolution layers, however, is a misnomer: what is called convolution in the literature on neural networks, and what is implemented in machine learning libraries, is in fact cross-correlation, given by

$$\sum_{\tau=-\infty}^{\infty} x[\tau]w[t+\tau]. \tag{2.5}$$

Many applications entail processing multidimensional data. For example, images are 2-dimensional (2D) spatially, and 3D considering the "channels" dimension that encodes colour. Convolution can be extended to handle such multidimensional data. For example, 2D convolution (by which we mean, hereinafter, cross-correlation), often used with image data, is given by

$$\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} x[i,j]w[m+i,n+j]. \tag{2.6}$$

In the case of RGB (colour) images, 2D convolution is applied per-channel (*i.e.*, across the spatial dimensions with 3 distinct 2D kernels), and the 3 resultant *feature maps* are summed together.

Convolutional Neural Networks (CNN) have been applied most successfully to spatially sampled data, such as images. Recently, however, they have been shown to perform well also on sequential data, such as text, which, in contrast to sampled data, do not easily bend to interpretation in frequency domain.

Pooling Layers

Pooling layers tend to follow convolution layers in CNNs. Many variants of pooling layers exist, but their function is generally one, and that is to summarize feature maps into more compact versions, thus reducing computational costs. A pooling layer replaces small regions in feature maps with summary statistics, such as the maximum or the average. Figure 2.3 shows an example of applying max pooling to a 4×4 image.

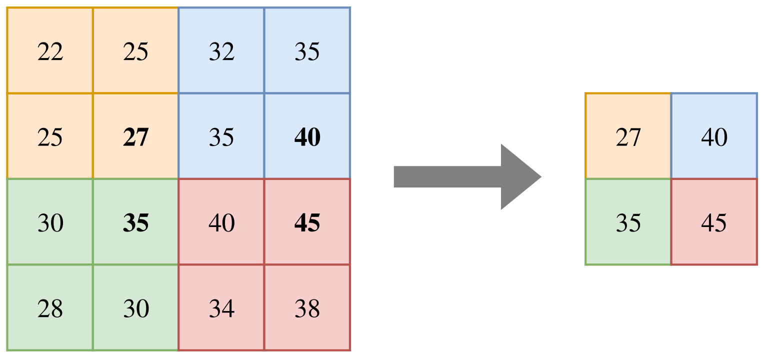

Figure 2.3: 2D max pooling with a 2×2 kernel and a *stride* of 2. The values shown represent pixel intensities.

In addition to computational savings, pooling introduces a degree of translation invariance in a model. This is useful in image classification applications, for example, because when processing images, the pixel-exact positions of features (*e.g.*, eyes in a face) are not generalizable.

Transposed Convolution Layers

A situation arises often where there is a need to "invert" a convolution layer (Dumoulin et al., 2017). This is particularly relevant in encoder-decoder models (such as the one proposed in Chapter 4). Transposed convolution layers (Dumoulin and Visin, 2016) serve this purpose only insofar as shape is concerned. In fact, transposed convolution *is* convolution, only with the input padded with zeros to achieve a desired output shape consistent with inversion. Nonetheless, transposed convolution, which is sometimes confusingly called deconvolution, is not a mathematical inverse of convolution—which *is* called deconvolution.

Normalization layers

Owing to the difficulties involved in optimizing neural networks, especially deep ones, many heuristics appear in the literature that have been shown empirically to lead to better solutions and/or faster convergence. Among those are normalization layers. In the coming chapters, we will make use of a version called batch normalization (Ioffe and Szegedy, 2015). Premised on the observation that changes to the distributions of the inputs to a network's layers can have a negative effect on the optimization, batch normalization normalizes layer inputs using batch statistics such that they approximate a standard normal distribution throughout training.

Normalizing the input, however, can also have unintended consequences. For example, it can constrain the input to a sigmoid activation to remain within the linear region of the sigmoid. Noting this, the authors augment the normalization step with a learned linear transformation, guaranteeing that the batch normalization layer can learn an identity mapping if needed. Thus batch normalization is given by

$$y = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{2.7}$$

where μ and σ are the batch mean and standard deviation, respectively; ϵ is a small number added for numerical stability; and γ and β are learned parameters. Note that the normalization is applied per parameter, and thus ignores correlations between parameters.

To obtain a deterministic inference function after training (*i.e.*, one that does not change across batches), batch statistics are replaced with population statistics.

Batch normalization have been shown empirically to lead to faster convergence and better solutions in some cases.

2.1.2 Training Neural Networks

The first step toward training or optimizing a neural network is selecting a loss function. The loss function quantifies how well a neural network performs on the task at hand. The optimization of neural networks is an iterative process that involves computing the gradient of the loss with respect to the network parameters. This necessitates that the loss be differentiable with respect to the network parameters.

The choice of loss function is dependent on the nature of the task. The most commonly used loss function for regression tasks (where the output is a continuous variable) is the mean squared error, defined as

$$L(\hat{Y}, Y) = \frac{1}{ND} \sum_{i=1}^{N} \sum_{j=1}^{D} (\hat{Y}_{ij} - Y_{ij})^2, \tag{2.8}$$

where $\hat{Y}$ is the prediction, Y the true value, and D and N are the output dimension and the number of samples, respectively.

In classification settings, where the output is a categorical variable, we use the Negative Log-Likelihood (NLL), sometimes called cross entropy, as the loss. For a single sample, NLL is given by

$$L(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{i} \mathbf{y}_i \log(\hat{\mathbf{y}}_i), \tag{2.9}$$

where i indexes the class, $\mathbf{y}$ is the one-hot-encoded true class, and $\hat{\mathbf{y}}$ is the predicted probability distribution over the classes. Note that, given $\mathbf{y}$ is a one-hot-encoded vector, the summand is nonzero only for a single value of i, the one corresponding to the true class. When processing more than one sample at a time, the loss is averaged over the samples.

Training a neural network is equivalent to minimizing the loss function. This is generally a non-convex optimization problem. Neural networks thus are normally optimized iteratively using gradient descent:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}, \tag{2.10}$$

where η, called the *learning rate*, controls the magnitude of the update in each iteration. The partial derivative of the loss with respect to a parameter, $\frac{\partial L}{\partial w}$, is computed using the chain rule of derivatives, starting from the output layer and traversing the network backwards. This process is known in the neural networks literature as *backpropagation*.

Mini-Batch and Stochastic Gradient Descent

While the loss we seek to minimize is the average over all the samples in the training set, for practical reasons we normally update based on mini batches of the training set. The mini batch loss is an estimate of the training set loss. Being cheaper to compute (along with its gradients), and due to redundancies in the training set, mini-batch loss leads to faster convergence. When an update is made per training sample, the optimization is known as Stochastic Gradient Descent (SGD).[1]

2.1.3 Overfitting and Regularization

Optimizing a neural network involves minimizing a loss function over a training set of samples. However, our interest is generally in the expected performance of the network on future samples, not on training samples. This dichotomy between what we actually optimize and what we want to optimize adds another layer of complexity—in addition to non-convexity—to the process of training neural networks.

The general practice in training neural networks is to use a proxy set, called a *validation* set, separate from the training set and not used in the optimization process, to gauge the expected future performance of a network. When monitoring both training and validation error rates during training, we commonly note that validation error follows but lags behind training error (*i.e.*, validation error tends to be higher). In many cases, this trend continues only up to a point, after which the two error rates diverge, with training error decreasing and validation error increasing. This phenomenon is called *overfitting*, and overcoming it is arguably the crux of learning.

Overfitting occurs whenever a high-capacity model is coupled with insufficient training data. This results in the model "memorizing" the peculiarities of the training set, such as any noisiness or outliers in the data, that are not reflected in the validation set or any future test data. Given a small training set, it is usually straight-forward to fit a high-capacity model to it with an error rate approaching 0%. Such a model, however, would not *generalize* to future data. Alternatively, we could choose a low-capacity model that cannot overfit. However, such an approach can introduce a bias to the model (*e.g.*, choosing a linear model when the data cannot be modelled linearly) and generally leads to poorer performance compared to more sophisticated approaches.

[1]Note, however, that there is no consensus on this naming convention. Some authors use SGD to refer to the case where a subset of more than one sample is used per update, while others still use it to refer to gradient descent in general.

This trade-off between the capacity of a model and the bias introduced by it is known in the statistical learning literature as the *bias-variance dilemma* (Geman et al., 1992). Given data generated from the model $y = f(x) + \epsilon$, where ϵ is a zero-mean noise with a variance of σ^2, the expected mean squared error (MSE) between any learned model, $\hat{f}$, and the true model on a test sample can be decomposed as

$$\mathbf{E}[(y - \hat{f}(x))^2] = \underbrace{\mathbf{E}^2[\hat{f}(x) - f(x)]}_{bias^2(\hat{f})} + \underbrace{\mathbf{E}[\hat{f}^2(x)] - \mathbf{E}^2[\hat{f}(x)]}_{variance(\hat{f})} + \sigma^2. \tag{2.11}$$

Rather than starting with a low-capacity model, overfitting is commonly countered by *regularizing* high-capacity models. Regularization refers to constraining the optimization of machine learning models, usually by penalizing model complexity. L2 regularization, sometimes called weight decay, penalizes the L2 norm of the parameters of a model. Thus for a task loss L_{task}, we minimize

$$L_{\text{task}} + \lambda \|\mathbf{w}\|_2^2, \tag{2.12}$$

where λ is a regularization coefficient, and $\mathbf{w}$ represents the vectorized model parameters. The addition of an L2 penalty drives the parameters toward lower values. For a linear model, it can be shown that it is equivalent to adding a zero-mean Gaussian prior on the parameters. L1 regularization, on the other hand, penalizes the L1 norm in the same way. It is commonly used to enforce sparsity.

While regularization is often used to refer to penalties to overfitting, it can be more generally understood to refer to any constraints imposed on the optimization of machine learning models. Over the coming chapters, we will introduce various regularization penalties designed to counter forgetting in continual learning models.

2.2 Continual Learning

Neural networks are normally optimized for, and subsequently deployed to handle, a single task. Between optimization and deployment, the networks are not altered or optimized for other tasks, and thus retain their optimized parameters. Different tasks are handled with different models trained in this way. The knowledge learned by any model does not affect the learning process of any subsequent task, which in turn does not alter previously acquired knowledge in other models. This framework of learning, where each model learns a single task, is sometimes called isolated learning (Chen and Liu, 2016). Contrast this with the human learning process, where acquired knowledge aids future learning and is continually being refined and expanded by new experiences. If the goal of artificial intelligence research is to advance ever closer toward human intelligence—ultimately to achieve what is known as Artificial General Intelligence (AGI)—it is necessary to move away from the isolated learning framework, where models are tailored to specific tasks, toward the more flexible and extendable continual learning framework.

The focus of this dissertation is on supervised learning tasks, and on image classification, in particular. Moreover, we consider neural network models only, and thus focus our exposition on their peculiarities. We note, however, that continual learning, or lifelong learning, is a more general framework that is not necessarily restricted to a single domain of tasks or family of models.

Consider a set of K tasks $\mathcal{T} = \{T_0, T_1, ..., T_k, ..., T_{K-1}\}$, each of which is an image classification problem with an associated data set $\{(\mathbf{x}_n^k, y_n^k)\}_{n=0}^{N-1}$. Under our restricted continual learning framework, we are interested in learning a single function, $\hat{f}$, parameterized by a neural network,[2] that maximizes the average accuracy on all tasks

$$\max_{\hat{f}} \frac{1}{K} \sum_{k=0}^{K-1} \mathbf{E}([y^k = \hat{f}(\mathbf{x}^k)]), \tag{2.13}$$

where $[\cdot]$ is the Iverson bracket notation, which evaluates to 1 if the condition in the bracket is true and to 0 otherwise. The k^{th} expectation in the summation is taken with respect to the test distribution of the k^{th} task. In addition, the tasks are to be presented to the model, and must be learned, sequentially: once the model is trained on a task it will no longer have access to the data associated with it (except for evaluation purposes). The challenge, then, is to maintain performance on a task once the network has been subsequently optimized for other tasks.

Below we discuss three other learning frameworks—online learning, transfer learning, and multi-task learning—that bear a resemblance to continual learning.

2.2.1 Relation to Online Learning

Between the two extremes of isolated learning and continual learning lies online learning. Here, a model is optimized for a single task and is updated continuously or periodically as new data *for the same task* become available. The distinction between online learning

[2]There is more to say about the output layer, which in one setting may be task-specific. See Section 5.2 for more details.

and continual learning is for practical purposes, as ultimately both update a conditional distribution of the form $p(y|\mathbf{x};\mathbf{w})$ in response to new data. In practice, however, continual learning stands as a more challenging problem, as the perturbations to model parameters induced by learning new tasks can be significantly larger. Other practical considerations, such as learning tasks with different output layers, adapting to different input domains, and combining the set of learned tasks, justify treating continual learning as a distinct problem.

Both online and continual learning are referred to as incremental learning in the literature.

2.2.2 Relation to Transfer Learning

Transfer learning, like continual learning, entails adapting a model's parameters sequentially for a set of tasks, usually only two. Unlike with continual learning, however, the only performance we seek to optimize with transfer learning is performance on the final task in the sequence. In other words, pre-training on the initial task or set of tasks acts to position the optimization of the final task at a favourable initial point; but once training on the final task begins, all other previous tasks become irrelevant.

Transfer learning is used to overcome scarcity of resources and/or data by making use of models pre-trained on large data sets and adapting them to different but related data sets. For transfer learning to bear fruit, the source and target domains must share at least some of their low-level features.

2.2.3 Relation to Multi-Task Learning

Multi-task learning is similar to continual learning in that in both frameworks we are interested in optimizing a model for a set of tasks. The main difference is that learning occurs simultaneously in the multi-task framework: the model parameters are optimized for all tasks jointly. In a way, each task here can be seen as a regularizer with respect to the remaining tasks. The approach proposed in Chapter 4 exploits this effect of multi-task learning as regularization to aid the continual learning process.

2.3 Catastrophic Forgetting

Catastrophic forgetting is a rather sensational expression of the fact that performance of a neural network on a task—as one would expect—hinges on the preservation of the parameters optimized for that task. In the course of learning multiple tasks sequentially, parameters optimized for a task are modified when the model learns a new task, causing performance on the initial task to drop. In general, as the number of intervening tasks between optimization and testing increases, the parameters drift farther and farther from their optimized values, and consequently the performance drop increases.

On the face of it, catastrophic forgetting may seem inherent to the way neural networks are optimized. After all, there is no obvious reason why performance on a task would persist once the model has been optimized for another. There are, however, a number of reasons to expect persistent performance in continual learning models.

Consider, for example, a set of image classification tasks. We expect a large overlap in the low-level features (*e.g.*, oriented edges), as well as some of the higher-level ones, among the tasks. Consequently, while sequential optimization for multiple tasks in this case does still modify model parameters from one task to another, the overlap in features among the tasks should lead to a higher degree of stability in the learned representations.

Another way of approaching this is to note that neural network models are often over-parameterized, and thus can have more capacity than is needed to learn all the tasks in a set. The question then becomes, how can the model make efficient use of its available capacity such that it can learn all the tasks in a set? This is known in the literature as the *stability-plasticity dilemma* (Grossberg, 1987). In other words, how can the model consolidate what it has learned so far (stability) while simultaneously freeing up unused capacity to learn additional tasks (plasticity)?

2.4 Literature Review

Researchers began to take notice of catastrophic forgetting around the time back-propagation was adopted as a method to compute gradients in neural networks and update weights (Rumelhart et al., 1986). McCloskey and Cohen (1989) observed that training neural networks sequentially led to old knowledge being overwritten by more recently learned representations. They termed this phenomenon *catastrophic interference.* The dramatic epithet "catastrophic" was used to emphasize that the forgetting of a task can be severe even in response to small changes in the corresponding optimized parameters

(French, 1999; Kolen and Pollack, 1990). With the advances made in deep learning research over the past decade, catastrophic forgetting has gained increased attention as the next problem to be re-examined.

Research efforts to contain or mitigate catastrophic forgetting in continual learning models generally follow one (or more) of three strategies: rehearsal, compartmentalization, and/or regularization.

2.4.1 Rehearsal-based continual learning

Rehearsal was one of the earliest strategies to be put forward in the literature to counter catastrophic forgetting (Ratcliff, 1990; Robins, 1993). It refers to the process of periodically training a model with samples from previously learned tasks as the model learns new tasks. As updates corresponding to new data are interleaved or averaged with updates corresponding to old data, catastrophic forgetting is checked and the drop in performance on old tasks tempered. Rehearsal requires that training data from all tasks learned be stored to be used in future re-training. As such, it can be considered a brute-force approach.

There has been recent work on optimal ways to update memory samples (Aljundi et al., 2019a). Chrysakis and Moens (2020), for example, explore memory sample selection in cases imbalanced data.

Pseudo-rehearsal is an extension of rehearsal, introduced to circumvent the latter's need for access to training data from previous tasks. Like rehearsal, it acts to drive the model to maintain performance on previously learned tasks through a process of periodic re-training. However, unlike rehearsal, it does so *without* access to training data from previous tasks. Researchers have proposed many pseudo-rehearsal approaches, but they all generally use surrogate data as a substitute for training data from previously learned tasks. In one approach, surrogate data are generated by passing random noise through the model and using the corresponding output as target labels (Robins, 1995; Frean and Robins, 1999). As the model learns a new task, it is concurrently trained with the surrogate data. This drives the model to maintain the same outputs for the surrogate data as the target labels, which implicitly constrains the model parameters.

Li and Hoiem (2016) and Furlanello et al. (2016) independently propose using the new task's training data, instead of random data, as the surrogate data. Their model[3] is made up of layers shared across tasks and a set of task-specific layers (this is referred to as the "multi-head" setting of continual learning and will be described further in the coming

[3]This description is based on the model described in (Li and Hoiem, 2018).

chapters). Prior to learning a new task, its input training data are passed through the shared layers and the task-specific layers to generate a set of outputs, Y_o^i, one corresponding to each old task i. These generated outputs are stored and used while learning the current task to minimize forgetting. The model is optimized to minimize a weighted sum of losses of the form

$$L(Y_n, \hat{Y}_n) + \lambda_o \sum_i L_{\mathrm{KD}}(Y_o^i, \hat{Y}_o^i), \tag{2.14}$$

where the first term corresponds to the loss on the current training task and the second term is a penalty on the changes to the input-output mapping represented by the model for previous tasks. We used the subscript $_{KD}$ to denote the distillation loss (Hinton et al., 2015) used by the authors.

Jung et al. (2018) propose a similar strategy but with the penalty imposed on changes to the mapping up to the penultimate layer of the network (as opposed to the full input-output mapping).

The underlying assumption of pseudo-rehearsal methods—and key to their success—is that preserving the input-output mapping represented by the model for the surrogate data (be they random noise or the training data for a subsequent task), has the effect of preserving the input-output mapping for the data of previous tasks. For this to be true, though, the surrogate data must approximately represent a random sample from the distribution of the training data of previous tasks. While using the training data of the current training task as a proxy for previous training data is more likely to satisfy this condition than using random noise, this is nonetheless not guaranteed. We address this issue further when we discuss the effect of inter-task similarity on continual learning performance in Section 5.1.

A better solution would be to learn to generate samples from the distribution of previous tasks. In recent years, adversarial training has become a powerful framework for learning generative models (Goodfellow et al., 2014). Shin et al. (2017) propose using a Generative Adversarial Network (GAN) to generate surrogate data for previous tasks. In other words, they propose using a pair of models, one trained to learn all the tasks and another, the generator, trained to learn the distribution of all the tasks seen so far. When a new task is introduced, the training data associated with it are augmented with data for previous tasks generated by the generator. This augmented data set is used to re-train the solver and the generator to account for the new task. It is clear that, given a perfect generator, the learned model approaches the performance of a multi-task model (that is, a model trained on all the tasks jointly). Thus continual learning approaches that learn to generate samples from the learned tasks hinge only on the successful learning of such generators.

Pseudo-rehearsal can be seen as a form of regularization, where the penalty is a function of the output rather than a function of its parameters. Unlike regularization approaches that we will review in the coming sections, however, pseudo-rehearsal approaches in a way circumvent, rather than solve, the catastrophic forgetting issue. Continual learning models based on pseudo-rehearsal can be trained from scratch for each new task, since they have access to training data (that is, surrogate data) for all previously learned tasks. Thus, while the real training data for each task is accessible only when that task is first learned, the model, viewed during learning a single new task, by using surrogate data, is actually trained in a multi-task framework, which is not subject to catastrophic forgetting.

In other words, in contrast to the strategies described in the coming sections, rehearsal methods pass on knowledge about previous tasks through the surrogate data rather than through the model's learned parameters.

2.4.2 Continual learning by compartmentalizing knowledge

In this section, we discuss approaches that counter catastrophic forgetting by reducing representation overlap across tasks. Early on, the representation overlap characteristic of distributed models like neural networks was identified as one of the causes of catastrophic forgetting (Murre, 1992; French, 1991). As a result, many research efforts went into designing models that reduce representation overlap, whether at the input layer or at the hidden layers. One suggestion was to use "activation sharpening" (French, 1991)—the process of slightly amplifying large activations and attenuating small ones during optimization, with the aim of encouraging orthogonality of representations through sparsity. (French (1994) subsequently criticized encouraging sparsity, suggesting it has an adverse effect on representational capacity.) Since then, researchers have proposed more sophisticated approaches with the same goal.

The underlying idea common to these approaches, to an extent, is to compartmentalize knowledge within the neural network such that learning a new task modifies only a subset of the network's parameters without significantly affecting the representations learned from previous tasks. There are generally two strategies to achieve this: 1) to start with a small network and gradually grow it (by adding new units or columns) as new tasks are encountered; or 2) to start with a high-capacity network and limit the share of the parameters dedicated to each task by masking or pruning. In either case, the end result is that different subsets of the model's parameters get optimized with respect to different tasks, and that parameters do not get modified after they have been assigned to and optimized for a task.

Terekhov et al. (2015) propose what they call block-modular neural networks, which gradually grow to cope with new tasks (Rusu et al. (2016) propose a very similar model, which they call progressive neural networks. For a new task, they instantiate a "block" of new layers (more commonly called a column in the literature) to be joined with the existing network. The units in the new layers have two types of connections: 1) vertical connections to the units in the preceding and following layers within the new block; and 2) lateral connection to the units in the corresponding preceding layer in the existing network. Only the newly added connections can be optimized for the new task, whereas existing connections remain set to their previously learned values. Thus the new layers, through the lateral connections, can make use of representations that were previously learned if they are relevant to the new task. However, existing connections cannot be modified in learning the new task. The end result is that each task has a corresponding block or column optimized for it, while simultaneously having access to the layers optimized for previous tasks. The sharing of knowledge through lateral connections serves to make the model more efficient than having an isolated model per task. Nonetheless, the fact that the model grows with the number of tasks learned represents a major weakness of this strategy.

The alternative strategy is to start with a high-capacity model and "divide" it across multiple tasks. Mallya and Lazebnik (2018) propose a strategy of pruning and masking to limit the share of the model's parameters optimized for each task. In this strategy, a model is trained on the first task, then its weight are pruned. They employ a simple pruning approach based on the magnitude of the parameters learned, setting the smallest-magnitude weights to zero. After pruning, the remaining weights are fine-tuned and a binary mask indicating which weights are relevant to the current task is stored. The process is repeated for subsequent tasks, using the previously pruned weights, until all the model's parameters have been optimized. As with previous approaches we reviewed, new tasks have access to previously optimized weights, but cannot modify them. While it overcomes the growing model issue of the previous strategy, this approach is limited by the fact that model parameters eventually run out, preventing subsequent learning. Moreover, there is a need to store a binary mask over all model parameters for each task learned, which creates a memory overhead that grows with the number of tasks.

In a somewhat similar approach, Mallya et al. (2018) propose to learn task-specific binary masks in an end-to-end fashion. Here, the first learned task serves as a base task, and the parameters optimized for it are not modified subsequently. When a new task is to be learned, they learn a binary mask over the parameters (in addition to an output layer), whose values, nonetheless, remain unchanged. As each task has an associated mask, and the base parameters do not change with additional tasks, the model is not subject to catastrophic forgetting. However, the representational capacity of such a model hinges on the parameters learned for the first task, which is rather arbitrary.

Yoon et al. (2018) propose a dynamically expanding network to alleviate model capacity saturation, expanding the model when the loss on a new task hangs above a pre-defined threshold. Zhang et al. (2020) propose a similarly expandable network, however, they advocate for using AutoML for automatic architecture search when expanding the network.

2.4.3 Regularization-based continual learning

When a model is optimized for a task, performance on that task hinges on the preservation of the optimized model parameters. Most regularization approaches work by imposing a penalty on deviating from the weights optimized for previous tasks. Not all optimized parameters, however, are equally important for "remembering" a task. As such, these approaches tend to penalize modifications to important weights more heavily, resulting in a compromise between maintaining previous parameter values and the ability to learn new tasks by utilizing free (that is, less constrained) parameters. (Note how the masking approaches described in the previous section can be viewed, to an extent, in terms of regularization with a binary penalty on parameters.)

The most naïve regularization approach is to anchor the solution for a task around the solution to the previous task with an isotropic penalty. For example, one could penalize $\|\theta^{t+1} - \bar{\theta}^t\|_2^2$, the L2-norm between solutions to consecutive tasks ($\bar{\cdot}$ indicates the final parameter values after optimizing for that task)

$$\min_{\theta^{t+1}} L(\theta^{t+1}) + \lambda \|\theta^{t+1} - \bar{\theta}^t\|_2^2, \tag{2.15}$$

where λ is a regularization coefficient. To learn more than two tasks, more penalties are added, each anchored at the solution to a different previous task. This approach, however, is overly restrictive in that it constraints even irrelevant parameters. The effect is compounded as more tasks are learned, and the inefficient use of capacity quickly paralyzes the model, preventing future learning. Noting this, Kirkpatrick et al. (2017) propose Elastic Weight Consolidation (EWC), in which the strength of the regularization is tied to the importance of the parameter to previous tasks. The main contribution of EWC lies in how they estimate parameter importance. For a data set $\mathcal{D}$, consisting of 2 training sets $\mathcal{D}_A$ and $\mathcal{D}_B$ corresponding to 2 tasks A and B, one could write the log-posterior probability as

$$\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}_B|\theta) + \log p(\theta|\mathcal{D}_A) - \log p(\mathcal{D}_B). \tag{2.16}$$

When optimizing for task B, we normally try to maximize $\log p(\mathcal{D}_B|\theta)$, the log-likelihood. The posterior $p(\theta|\mathcal{D}_A)$ summarizes all the information contained in task A, including parameter importance with respect to that task; and were we to somehow estimate this posterior, we can use it to take task A into account while learning task B. EWC approximates the posterior by a Gaussian distribution, centred at the optimized parameters of task A, $\bar{\theta}^A$, and with a diagonal precision matrix. The precision values are estimated using a diagonal Fisher information matrix. Thus, the regularized cost function for task B becomes

$$\log p(\mathcal{D}_B|\theta) + \lambda \sum_i F_i(\theta_i - \bar{\theta}_i^A)^2, \tag{2.17}$$

where i indexes model parameters, and F_i is the precision corresponding to the i^{th} parameter. Intuitively, the penalty represents the steepness of the loss function along the direction of each parameter, and thus the importance of the parameter (*i.e.*, parameters that can vary without affecting the the loss have a small effective regularization coefficient λF_i).

In the same vein as EWC, French and Chater (2002) had previously proposed using the hessian of the loss function to estimate its steepness around previous optimized parameters. Most other importance-based regularization approaches differ only in how they estimate parameter importance (Zenke et al., 2017).

Farajtabar et al. (2020) propose orthogonal gradient descent, an approach to countering forgetting by restricting gradient descent updates of parameters to directions in the parameter space that do not affect performance on previous tasks.

In some cases, regularization can be implicit. Serra et al. (2018), for example, use parameter importance to scale per-parameter updates rather than adding an explicit penalty. In their model, parameter importance is learned concurrently with the current task through an attention mechanism. That is, layer activations are multiplied element-wise with learned, almost-binary attention vectors. The attention vectors are encouraged to be sparse, thus leading to efficient use of capacity. Parameters corresponding to high attention vector values are deemed important and hence their gradient descent updates are scaled down.

Finally, we noted earlier that some pseudo-rehearsal approaches can be interpreted in terms of regularization (Li and Hoiem, 2016; Shmelkov et al., 2017). For example, when the training data $(\mathbf{x}^B, y^B)$ for the current task are augmented with some surrogate data for a previous task,$(\mathbf{x}^A, y^A)$, the optimized cost function can be seen as a regularized objective

$$L(\hat{y}^B, y^B) + \lambda L(\hat{y}^A, y^A). \tag{2.18}$$

We end this section by highlighting that most of the approaches we reviewed here have a common weakness: overcoming catastrophic forgetting appears in the continual learning framework as an afterthought, a remedy applied after learning to prevent forgetting. In Chapter 4, we will propose an alternative, more preemptive approach, in which we attempt to learn representations from the start that are relatively less susceptible to catastrophic forgetting.

2.5 Summary

In this chapter, we reviewed neural networks, their building blocks, and how they are optimized. Moreover, we introduced the continual learning framework and discussed how catastrophic forgetting arises in this context. We also reviewed the literature on overcoming catastrophic forgetting. Starting with the next chapter, we begin describing the main contributions of this dissertation.

Chapter 3

Estimating Importance in Continual Learning Models

We reviewed in the previous chapter the various approaches researchers have taken to counter catastrophic forgetting in continual learning models. We saw that in one approach a regularization term may be used to penalize changes to model parameters, and thus reduce forgetting. It is an approach that addresses forgetting in fixed capacity models and absent any replay memory. With only a finite number of parameters, fixed architecture, and no access to rehearsal data, countering catastrophic forgetting becomes entirely dependent on the update mechanism of model parameters.

In the most basic regularization strategy, one could penalize changes to all the parameters of the model after it had learned its first task. The problem with this strategy, however, is that it prevents subsequent learning. And hence, while the model may not forget, it also cannot learn, which defeats the continual learning purpose.

As we noted in surveying the literature in the previous chapter, a more intelligent way to penalize changes to the model is to scale a per-parameter penalty by an estimate of the parameter importance. Parameters that are deemed important are strongly penalized, while unimportant parameters are allowed to change freely. In this way, the model achieves a balance between countering forgetting and preserving its capacity to learn additional tasks.

This raises the question: how do we define importance and how can we estimate it?

Researchers have proposed various answers to this question, some defining importance on a per-parameter basis (Kirkpatrick et al., 2017; Chaudhry et al., 2018), while others assigned importance per unit or filter (Jung et al., 2020; Aljundi et al., 2019b). In most cases, however, importance is derived, one way or another, from the sensitivity of the loss function to changes to parameters or units.

Recently, Jung et al. (2020) proposed using the average activation value of a unit as an estimate of its importance. The approach has its advantages, including the low memory and computational cost of being a per-unit rather than per-parameter estimate, that it only requires a single pass over the training samples, and that, unlike many methods that estimate importance from the loss gradient, it does not require back-propagation. However, it also suffers from a significant disadvantage, namely, that it has a tendency to assign high importance values to unimportant units (as we will show). This tendency is more pronounced in network layers closer to the output classification layer. Assigning high importance values to unimportant units leads to inefficient usage of model capacity, which in turn leads to intransigence (*i.e.*, resistance to learning) (Chaudhry et al., 2018).

In this chapter, we propose a novel importance estimate that addresses the limitations of using the average activations estimate by Jung et al. (2020), while maintaining its positive aspects. We observe empirically that units in the layers closer to the output classification layer tend to specialize and become more discriminative. Moreover, as the average activation value for a unit is not an accurate indicator of the degree to which it is discriminative (as we will show), we propose replacing the activations average importance estimate for layers closer to the output with what we call a *class separation* importance estimate. The proposed estimate assigns higher importance values to units that are more discriminative and lower values to units that are less discriminative (even if the latter have an overall high average activations across classes). We hypothesize that discriminative units are more important to preserving learned skills and avoiding forgetting. Moreover, for layers that are farther away from the output layer, we propose an explicit penalty, similar to the work on deeply supervised nets by Lee et al. (2015), that induces the model to learn discriminative units at any hidden layer.

We evaluate the proposed approach on a diverse set of task sequences and compare its performance against several parameter and unit importance estimation methods. We show that the proposed approach outperforms the other methods in most cases, all while having a small memory and computational cost. We also show that it alleviates the intransigence problem of the activations average method.

The rest of this chapter is organized as follows. Section 3.1 provides an overview of the continual learning setting and reviews the relevant literature. Section 3.2 outlines importance estimation and formulations of regularization-based continual learning methods. Section 3.3 describes the proposed approach. We provide experimental results in

Section 3.4 and conclude the chapter in Section 3.5.

3.1 Background and Related Work

This section formally introduces the continual learning problem and reviews the literature on catastrophic forgetting. In surveying the literature, we will discuss briefly the concept of importance, which we formalize in Section 3.2.

Recall from Section 2.2 that a continual learning framework consists of a sequence of tasks $[T_0, T_1, T_2, \cdots]$, with associated training data sets $[D_0, D_1, D_2, \cdots]$. Each training data set D_k consists of a set of training samples $\{(\mathbf{x}_i^k, y_i^k)\}_{i=0}^{N_k-1}$, where N_k is the number of training samples in the k^{th} data set.

In this work, we focus solely on image classification tasks. Therefore, each T_k corresponds to an image classification task, $\mathbf{x}_i^k$ is the i^{th} training sample for T_k, and y_i^k is the corresponding target class.

Moreover, we consider only mutually exclusive tasks. In other words, the classes appearing in D_k appear only in that training set.

The training sets are available, or presented to the model, one after the other, such that the model loses access to D_k after learning task T_k. The goal in this framework is to preserve the performance of the model on all the tasks in the sequence as it progresses to learn the tasks. In other words, at all times, the objective of continual learning is to maximize:

$$\frac{1}{M} \sum_{k=0}^{M-1} \mathbf{E}([y^k = \hat{y}^k]), \tag{3.1}$$

where M is the total number of training tasks in the sequence that the model has been trained on (including the active training task), $\hat{y}^k$ is the class predicted by the model, and the $[\cdot]$ is the Iverson bracket notation, which evaluates to 1 if the condition in the bracket is true and to 0 otherwise. The k^{th} expectation in the summation is taken with respect to the test distribution of the k^{th} task. That is, we seek to train the model on the task sequence such that the average accuracy on all the tasks it is trained on remains maximized throughout the training process. This despite the constraint that each training data set D_k is available only for one part of the training sequence.

We consider in this work the multi-head setting, in which each task T_k has a corresponding output "head" (*i.e.*, a set of output units), and in which the model is assumed to have perfect knowledge of the correct head to use for each test input sample. In other words, for each test sample, the model has to make a decision by predicting a class from the set of classes in one of the heads only (by contrast, in the single-head setting, the model has to predict from the union of all the sets of classes of all the output heads). We will discuss the difference between the single-head and multi-head settings in more details in Section 5.2.

As we discussed earlier, the objective of continual learning to maintain performance on all learned tasks is hindered by catastrophic forgetting (we will formally define forgetting when we introduce the evaluation metrics we use in Section 3.4.3). Continual learning researchers have proposed different approaches to address forgetting, including architecture-based approaches (Terekhov et al., 2015; Rusu et al., 2016; Mallya and Lazebnik, 2018; Mallya et al., 2018), replay-based approaches (Ratcliff, 1990; Robins, 1993; Li and Hoiem, 2016; Furlanello et al., 2016; Shin et al., 2017; Li and Hoiem, 2018), and regularization-based approaches (refer to Section 2.4 for a broader review of the literature).

Regularization-based approaches, which are most relevant to our work, involve explicit penalties on changes to optimized parameters. EWC (Kirkpatrick et al., 2017) introduced a per-parameter importance estimate based on a diagonalized Fisher information matrix. The original EWC method has since seen many extensions. Huszar (2018) proposed a variant of EWC that keeps track of a single set of importance values (rather than one set per learned task) and uses a single penalty centered at the most recent optimized parameter vector. Schwarz et al. (2018) proposed an online version of EWC with lower computational costs. Riemannian Walk (RWalk) (Chaudhry et al., 2018) combines an online version of EWC with a path integral-based estimate of importance (Zenke et al., 2017). Most of these methods estimate parameter importance from the loss gradient (Aljundi et al., 2019b). More recently, Jung et al. (2020) proposed using the average activation value of a unit as a measure of its importance. This approach has 2 advantages over the previous methods: 1) it assigns importance to units rather than parameters, and thus incurs less memory costs; and 2) it involves only a single forward pass through the training data and does not require back-propagation, and thus incurs low computational cost to compute. In the coming sections, we will examine this importance estimate more closely, identify some of its weak points, and propose a method that builds on it while addressing its limitations.

3.2 Importance Estimation

As we saw in Chapter 2, the standard cost function used in classification tasks is the negative log-likelihood (NLL) or cross-entropy loss function. For task T_k, this is given by:

$$L^k_{\text{cls}} = -\frac{1}{N_k}\sum_{i=0}^{N_k-1}\sum_{j=0}^{C_k-1} p^k_{i,j}\log(\hat{p}^k_{i,j}), \tag{3.2}$$

where C_k is the number of classes in task T_k, $p^k_{i,j}$ is the target probability that sample $\mathbf{x}^k_i$ belongs to class j (for a one-hot encoded target, this evaluates to 1 for only one class, and evaluates to 0 for the remaining classes), and $\hat{p}^k_{i,j}$ is the corresponding probability predicted by the model.

During optimization, we seek to minimize this loss function with respect to the model's parameters. In the multi-head continual learning, it is common to have a set of parameters shared across tasks, θ, and a set of task-specific parameters, θ^k, associated with each task. With this convention, the model's prediction is given by:

$$\mathbf{z}^k_i = f_{\theta^k}(g_\theta(\mathbf{x}^k_i)). \tag{3.3}$$

In this work, g_θ represents the core network: a sequence of convolution and fully connected layers that are shared across all tasks. f_{θ^k}, on the other hand, represents a set of output fully connected units (*i.e.*, a linear transformation) that is specifically optimized for each task. Given $\mathbf{z}^k_i$, the predicted probabilities are given by a softmax over the model's output:

$$\hat{p}^k_{i,j} = \frac{e^{z^k_{i,j}}}{\sum_m e^{z^k_{i,m}}}. \tag{3.4}$$

Hence, it is clear that, for a single sample, the loss function $L^k_{\text{cls}}(\mathbf{x}^k_i, y^k_i)$ is a parametric function of both θ and θ^k: $L^k_{\text{cls}}(\mathbf{x}^k_i, y^k_i; \theta, \theta^k)$.

For two tasks, T_0 and T_1, learned one after the other, the model begins by optimizing θ and θ^0 for T_0. At the risk of introducing confusing notation, let's denote the optimized value of θ after learning task T_0 by $\theta^{*(0)}$. To subsequently learn T_1, the model optimizes again θ, as well as a separate set of task-specific parameters θ^1. As a consequence of this re-optimization, one would expect θ to start drifting away from $\theta^{*(0)}$, leading to worsening performance on T_0 (*i.e.*, we say the model forgets task T_0).

Perhaps the most naïve way to prevent this worsening of performance on T_0 while learning T_1 is to add an L2 regularization penalty to the cost function L^1_{cls} of the form:

$$L^1_{\text{reg}} = ||\theta - \theta^{*(0)}||^2_2, \tag{3.5}$$

to obtain the regularized cost function:

$$L^1 = -\frac{1}{N_1}\sum_{i=0}^{N_1-1}\sum_{j=0}^{C_1-1} p^1_{i,j}\log(\hat{p}^1_{i,j}) + \lambda||\theta - \theta^{*(0)}||^2_2, \tag{3.6}$$

where λ is a hyper-parameter controlling the regularization strength. Thus, the L2 penalty prevents changes away from the optimized parameter vector $\theta^{*(0)}$, which in turn prevents forgetting T_0. The downside to this penalty, however, is that it is isotropic: it penalizes all individual parameters with the same strength. This means that the model will have no capacity—no free parameters—left to learn T_1 or subsequent tasks. This is a phenomenon sometimes referred to as *intransigence* or *negative forward transfer* (Chaudhry et al., 2018).

3.2.1 Per-Parameter Importance

A more intelligent approach would be to penalize changes to parameters that are "important" to T_0 while allowing less important parameters to change more freely. To do this, recent methods estimate the importance of each parameter in the model and apply a per-parameter penalty, to obtain a loss function of the form:

$$L^1 = -\frac{1}{N_1}\sum_{i=0}^{N_1-1}\sum_{j=0}^{C_1-1} p^1_{i,j}\log(\hat{p}^1_{i,j}) + \lambda\sum_{r=0}^{P-1}\Omega_r||\theta_r - \theta^{*(0)}_r||^2_2, \tag{3.7}$$

where we use P to denote the total number of parameters in the shared part of the network and Ω_r represents a penalty on changes to the r^{th} parameter. Researchers have proposed many different approaches to estimate Ω_r, including the previously mentioned EWC and RWalk. Most of these approaches, however, use in some form the partial derivatives of the loss with resepct to a parameter, $\frac{\partial L}{\partial \theta_r}$, to estimate the importance of that parameter.

3.2.2 Per-Unit Importance

Instead of computing and tracking per-parameter importance values, some approaches estimate the importance of units (*e.g.*, a convolution or a fully connected unit). One advantage of this approach is that it is less computationally demanding. Another is that it is more open to interpretation to consider units rather than individual weights (we can more easily understand units as filters or feature detectors).

Recent work by Jung et al. (2020) explored the use of the average value of a unit activation on the training set samples as a measure of its importance. In other words, for unit l, we estimate an importance value as:

$$\Omega_l^k = \frac{1}{N_k} \sum_{i=0}^{N_k-1} h_l(\mathbf{x}_i^k), \tag{3.8}$$

where $h_l(\mathbf{x}_i^k)$ denotes the output of unit l for training sample $\mathbf{x}_i^k$. We will refer to this approach as *activations average* in the rest of this chapter.

Our proposed approach to estimating unit importance will take the activations average approach by Jung et al. (2020) as a starting point. In the next section, we will highlight the latter's primary weakness, and describe the approach proposed in this work.

3.3 Proposed Approach

While the activations average approach has its advantages, it can be inefficient in its use of model capacity. This is more clearly noticeable in the model's layers closer to the output. As an example, consider the results shown in Fig. 3.1 and Fig. 3.2. The figures show the activation values for the units in the last 2 layers before the classification layer for a 2-class classification problem. On the right side of each figure, we see the average activation value corresponding to each unit (shown in green). We also see a measure of how discriminative each unit is (shown in blue). We will discuss these figures in more detail in the next section, but for now we observe that the units with the highest activations average are not necessarily the units that are most discriminative. In fact, some of the units with the lowest activations average are among the most discriminative, while some of the units with the highest activations average are among the least discriminative.[1] This means that assigning a high importance estimate to the units with high activations average is unnecessarily using up the model's capacity, by preventing the parameters of those units from changing, while at the same time missing important units that *should* be preserved. In other words, using activations average as an importance estimate leads to both intransigence as well as a degree of forgetting.[2]

[1]Note that this ignores correlations between the units, which is a simplification common in importance estimation approaches, given the high computational and memory costs of accounting for them.

[2]Jung et al. (2020) propose a sparsity-inducing approach in their paper to address the intransigence issue. Our focus in this work, however, is on the importance estimate only.

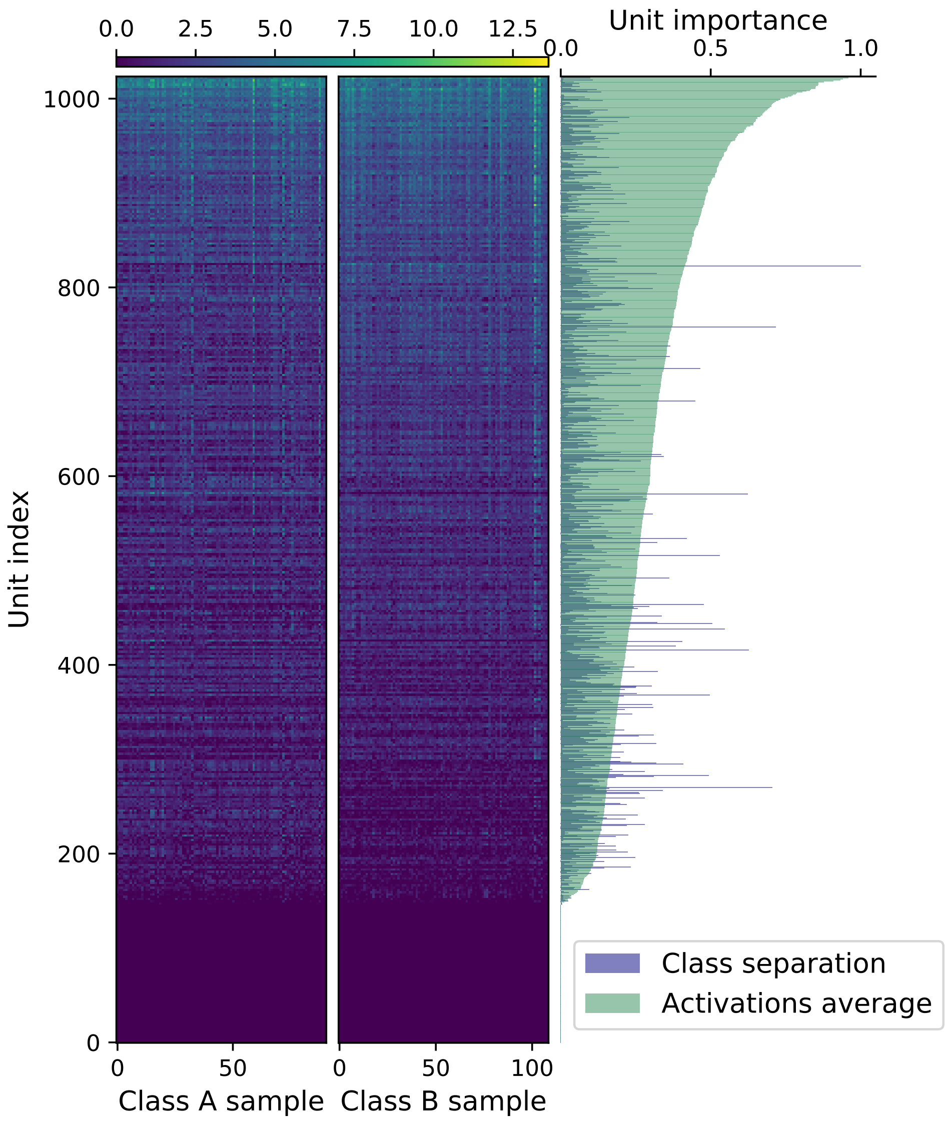

Figure 3.1: Left: activation values for the first fully connected layer for the 11th task in the c100-2 task sequence (refer to Section 3.4). The activation values are shown for all units and training samples. The training samples for the 2 classes are shown separately. Right: The importance values assigned by both methods to the corresponding unit. All plots are ordered according to the activations average values.

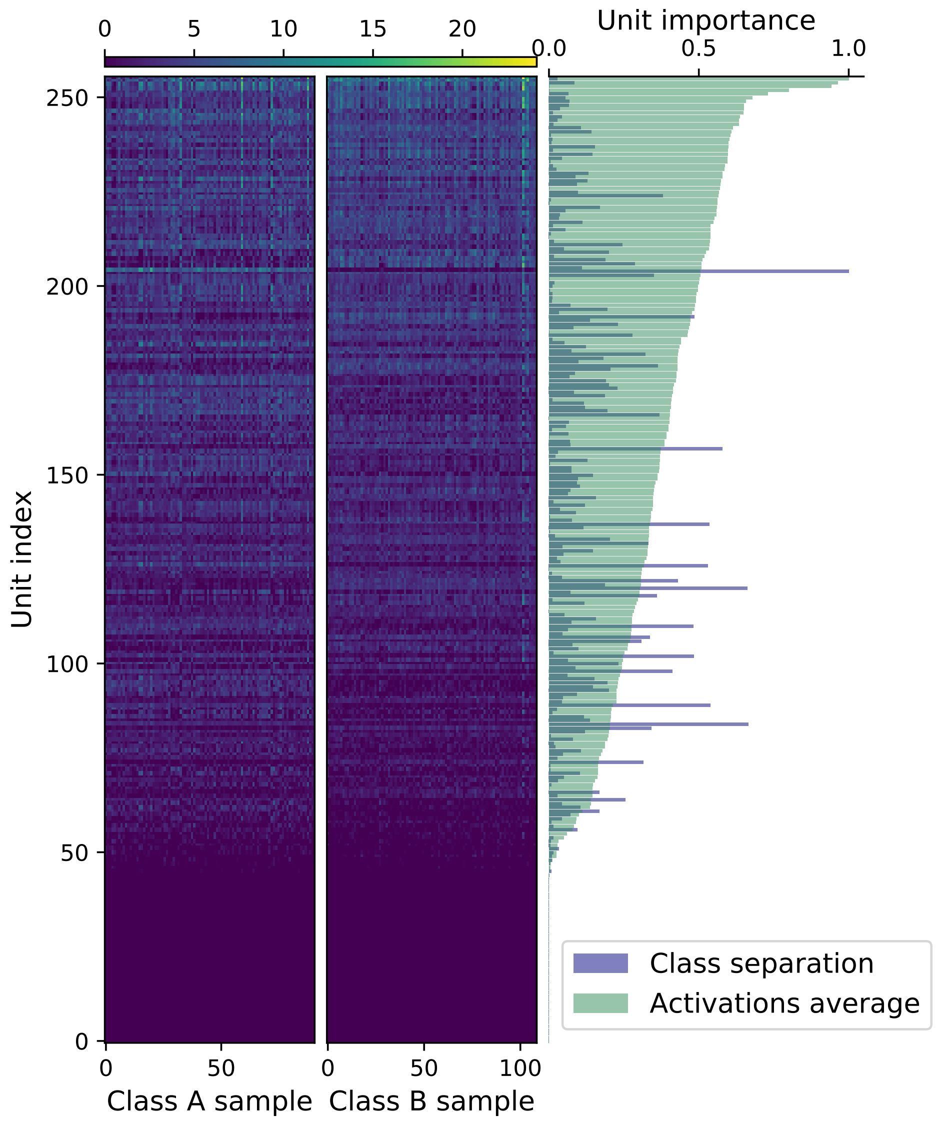

Figure 3.2: Left: activation values for the second fully connected layer for the 11th task in the c100-2 task sequence (refer to Section 3.4). The activation values are shown for all units and training samples. The training samples for the 2 classes are shown separately. Right: The importance values assigned by both methods to the corresponding unit. All plots are ordered according to the activations average values.

In view of this observation, we propose using a measure of how discriminative a unit is as an estimate of its importance—especially for layers closer to the classification layer. The emphasis on layers closer to the output aligns with how neural networks are thought to operate: units in the initial layers learn common low-level features, while later layers specialize and learn to identify higher-level, discriminative features. The emphasis is also driven by empirical observation that supports this understanding of neural networks.

3.3.1 Class Separation as an Estimate of Importance

To quantify how discriminative a unit is, we draw inspiration from the literature on Fisher Linear Discriminant Analysis (FLDA) (Murphy, 2012, p. 274). FLDA is a method used to find the linear projection of a set of features to a low-dimension space that is most conducive to classification purposes. That is, the low-dimension space that maximizes class separation. To that end, FLDA is formulated to maximize the distance between the class means in the projection space and minimize the within-class spread of each class. For 2 classes, this is equivalent to:

$$\max \frac{(\mu_1 - \mu_0)^2}{\sigma_1^2 + \sigma_0^2}, \tag{3.9}$$

where μ_j and σ_j^2 denote class j mean and variance in the projected space, respectively.

The objective of FLDA is to *find* the projection that maximizes this quantity. In this work, however, we will use (3.9) as a measure of each unit's importance. Hence, for unit l and 2 classes, we have:

$$\mu_{l,j}^k = \frac{1}{N_{k,j}} \sum_{i:p_{i,j}^k=1} h_l(\mathbf{x}_i^k), \tag{3.10}$$

where $\mu_{l,j}^k$ is the activations average of class j training samples for unit l after learning task T_k, $N_{k,j}$ is the number of training samples from class j, and $p_{i,j}^k$ is j^{th} element of the one-hot encoded target of the class label y_i^k.

Likewise, we compute class j spread for unit l after learning task T_k as:

$$\sigma_{l,j}^k = \frac{1}{N_{k,j} - 1} \sum_{i:p_{i,j}^k=1} (h_l(\mathbf{x}_i^k) - \mu_{l,j}^k)^2, \tag{3.11}$$

where we divide by $N_{k,j} - 1$ to compute the unbiased sample variance.

With class means and variances computed, we estimate importance for unit l after task T_k by:

$$\Omega_l^k = \frac{(\mu_{l,1}^k - \mu_{l,0}^k)^2}{\sigma_{l,1}^k + \sigma_{l,0}^k}. \tag{3.12}$$

The description so far has addressed the regularization up to the second task only, where there is only a single penalty centred around the first task's optimized parameter vector. There are mainly 2 ways in the literature by which researchers address subsequent tasks: 1) a separate penalty centred around every task's optimized parameters (*e.g.*, the original EWC approach (Kirkpatrick et al., 2017), or 2) a single penalty centred around the last task's optimized parameter vector (*e.g.*, RWalk (Chaudhry et al., 2018; Huszar, 2018)). We follow the second approach in this paper. Therefore, starting with the third task in the sequence, we use:

$$L_{\text{sep}}^k = \sum_l \tilde{\Omega}_l^{k-1} \sum_{r_l} ||\theta_{r_l} - \theta_{r_l}^{*(k-1)}||_2^2, \tag{3.13}$$

where r_l is used to index parameters of unit l and $\tilde{\Omega}_l^{k-1}$ is a normalized weighted average of importance parameters:

$$\hat{\Omega}_l^{k-1} = \alpha(\Omega_l^{k-1} + \hat{\Omega}_l^{k-2}) \tag{3.14}$$

$$\tilde{\Omega}_l^{k-1} = \frac{\hat{\Omega}_l^{k-1}}{\max_{n:n\in L_l} \hat{\Omega}_n^{k-1}}, \tag{3.15}$$

where α is a hyper-parameter and we introduce L_l to denote the set of units in the layer containing unit l. That is, the normalization is applied on a layer-by-layer basis.

We refer to this proposed importance estimation method as *class separation*.

3.3.2 Enhancing Class Separation via Hidden Layer Supervision

We noted earlier that the class separation-based importance estimates are especially targeted at the layers closer to the output classification layer. This is because we expect those layers to be most discriminative under standard gradient descent optimization.

In order to apply the class separation method on earlier layers more effectively, we explicitly induce those layers to be more discriminative by introducing a supervision signal at the targeted hidden layers. The approach draws on the "deeply supervised nets" work by Lee et al. (2015), where they introduce the concept of "companion losses" at the hidden layers, in order to increase network transparency, discriminative ability of hidden layer features, and overall performance of deep networks.

In our work, we use a single-layer softmax classifier at each targeted hidden layer that is trained to take as input the hidden layer's activations and predict the class labels for the task's training data.

Let the activation values for hidden layer u for input sample $\mathbf{x}_i^k$ be $\mathbf{h}_{u,i}^k = q_u(\mathbf{x}_i^k)$, where we use $q_u(\cdot)$ to denote the mapping from the input layer to the output of hidden layer u. We add as a regularization term to the overall objective function the following classification loss:

$$L_{\text{hidden}}^k = -\frac{1}{N_k} \sum_u \sum_{i=0}^{N_k-1} \sum_{j=0}^{C_k-1} p_{i,j}^k \log(\hat{s}_{u,i,j}^k), \tag{3.16}$$

where $\hat{s}_{u,i,j}^k$ are the probabilities predicted by hidden layer u's classifier:

$$\hat{s}_{u,i,j}^k = \frac{e^{(w_u^{k\top} \mathbf{h}_{u,i}^k)_j}}{\sum_m e^{(w_u^{k\top} \mathbf{h}_{u,i}^k)_m}}, \tag{3.17}$$

where w_u^k is the single-layer classifier's weight vector for hidden layer u and task T_k. Note that these hidden layer classifiers are only needed during training, and are discarded after learning each task.

Putting together (3.2), (3.13), and (3.16), we obtain the full proposed objective function:

$$L^k = L_{\text{cls}}^k + \lambda_{\text{s}} L_{\text{sep}}^k + \lambda_{\text{h}} L_{\text{hidden}}^k, \tag{3.18}$$

where λ_s and λ_h are hyper-parameters.

3.4 Experimental Findings

In this section, we report experimental findings for the proposed approach, evaluating it against recent continual learning methods. We begin by describing the experimental setting and the task sequences used. Then we provide hyper-parameter and other implementation details. In Subsection 3.4.3, we describe the evaluation metrics used and formally define forgetting and intransigence. Finally, we provide detailed results for accuracy and capacity usage.

3.4.1 Task Sequences

We evaluate the proposed approach in the setting of image classification tasks. We limit this work to binary classification tasks and design the task sequences to cover a variety of scenarios, to test the effectiveness of each method under various conditions. The tasks are drawn from the following data sets: CIFAR10, CIFAR100 (Krizhevsky, 2009), MNIST (LeCun et al., 1998), KMNIST (Clanuwat et al., 2018), FashionMNIST (Xiao et al., 2017), STL-10 (Coates et al., 2011), and SVHN (Netzer et al., 2011).

We define the following sequences of tasks:

- c10-2: CIFAR10 classes, 2 classes per task: $[(0, 1), (2, 3), \cdots]$. 5 tasks.
- c100-2: CIFAR100 classes, 2 classes per task. 50 tasks.
- c10-k-2: 2 classes from CIFAR10 followed by 2 classes from KMNIST, repeated: $[\text{CIFAR10}(0, 1), \text{KMNIST}(0, 1), \text{CIFAR10}(2, 3), \cdots]$. 10 tasks.
- k-c10-2: Same as c10-k-2, but starting with $[\text{KMNIST}(0, 1), \cdots]$. 10 tasks.
- k-2: KMNIST classes, 2 classes per task. 5 tasks.
- diverse-2: CIFAR10 classes, followed by KMNIST, then MNIST, then FMNIST, and then STL-10, 2 classes per task. 25 tasks.

The sequences are chosen to test performance with long and short sequences, similar, dissimilar, easy and difficult tasks, and with different sequence orderings.

For all tasks and sequences, we use 100 training samples per class. We tune hyperparameters on a validation set extracted from the original training set associated with each data set, using an 80/20 split. We evaluate the models on the test set of each data set. For the diverse-2 task sequence, we use 100 test samples per class for evaluation. The remaining sequences use the full test sets. All input images are scaled to be $3 \times 32 \times 32$ in size (gray scale images are replicated across the 3 channels).

3.4.2 Implementation Details

We use the following architecture for all networks: 3 convolution layers, with 128 units each, followed by 2 fully connected layers, with 1024 and 256 units respectively. All units use ReLU activation. All convolution layers are followed by batch normalization, and the first 2 convolution layers are also followed by 2×2 max-pooling.

We use the Adam optimizer with a learning rate of $1e - 4$ for all experiments. We repeat each experiment 10 times and report average values.

For all sequences, we use a batch size of 100. For c100-2 and diverse-2, we train the model for 200 batches per task, while for the remaining sequences, we train for 400 batches.

We compare the proposed approach against a vanilla baseline model, L2-regularized model, EWC, RWalk, and the activations average method in (3.8).

For L2 and EWC, we use a regularization coefficient of $1e4$ and $1e7$, respectively, for all experiments. We report results for the EWC version proposed by Huszar (2018). For RWalk, we use a regularization coefficient of 1.0. The activations average method uses the same normalization and averaging process outlined for the proposed class separation method in (3.13)–(3.15). We use a regularization coefficient of 10.0 for the activations average method for all task sequences except c100-2 and diverse-2, which use a value of 100.0. Additionally for both activations average and class separation, we set α in (3.14) to 1.0 for all sequences except c100-2 and diverse-2, where it is set to 0.5.

For the proposed class separation approach, we set λ_s and λ_h in (3.18) to 1.0 and 0.01, respectively. We apply class separation only to the fully connected layers, while the convolution layers are regularized with activations average-based importance. We apply hidden layer supervision only to the first fully connected layer. For convolution layers, the activations averages are computed over the norm of the feature maps.

3.4.3 Evaluation Metrics

We make use of 3 metrics to evaluate the performance of the models: average task accuracy, forgetting, and intransigence (Chaudhry et al., 2018). While average accuracy gives an overview of overall performance, reporting forgetting and intransigence values gives us more insight into the workings of each method.

By average task accuracy, we mean the average accuracy of the model on all the tasks learned up to and including the active training task. Because tasks in a sequence vary in difficulty, the average accuracy is not sufficient to understand the extent of forgetting and intransigence.

To define forgetting,[3] let's first denote by $A_t^{t'}$ the accuracy of the model on task t having

[3]The definitions of forgetting and intransigence used in this work are based on those provided by Chaudhry et al. (2018), with minor deviations.

learned up to task t' (we assume $t \leq t'$). And let the accuracy of the model on task t at any time after learning it be denoted by A_t. Forgetting of task t, then, is defined as:

$$F_t = A_t^t - A_t. \tag{3.19}$$

Intransigence is a measure of the resistance to learning introduced by the continual learning framework or method. We define it with respect to a baseline reference point. In this paper, the reference point is the accuracy of the same model on task t when it is trained on all the tasks in the sequence simultaneously. For task t, we refer to this reference point by B_t. Hence, intransigence is given by:

$$I_t = B_t - A_t^t. \tag{3.20}$$

Both forgetting and intransigence can take positive and negative values. Negative forgetting values imply an improvement in performance on a previously learned task as a result of subsequently learning another task (this is sometimes referred to as *positive backward transfer*). Negative intransigence values imply that the continual learning framework (coupled with the algorithm used) improves the performance of the model on the task relative to the baseline reference point (this is sometimes referred to as *positive forward transfer*).

3.4.4 Results

Figures 3.3–3.8 show the average accuracy results for all methods for the k-2, c10-2, c10-k-2, k-c10-2, c100-2, and diverse-2 task sequences, respectively.

For the k-2, c10-,2 c10-k-2, c100-2, and diverse-2 task sequences, the proposed class separation method improves on the activations average method. Only for the k-c10-2 task sequence does class separation perform slightly worse than activation average (by about 0.8%) by the end of the final task, although for most of the training sequence, class separation has a small advantage. We see the largest gaps in performance between activations average and class separation in c10-k-2 (9.5%), c100-2 (5.8%), and diverse-2 (13.9%), with class separation taking the lead in all 3 cases.

The proposed class separation also performs competitively with other recent approaches like EWC and RWalk. In fact, for the longest task sequence, c100-2, class separation outperforms all other methods by a large margin (4.2% higher than EWC, 8.5% higher than RWalk, and 5.8% higher than activations average), suggesting it is the most effective among the examined methods in striking a balance between countering forgetting and efficient use of model capacity.

Similarly in the diverse-2 task sequence, class separation outperforms all other methods by a significant margin (9.2% higher than EWC, 5.1% higher than RWalk, and 13.9% higher than activations average). We note that EWC and RWalk perform comparably to class separation at the beginning of the task sequence. By the 12^{th} task, however, the gap between EWC and class separation starts to grow, which is indicative of intransigence starting to prevent additional learning (*i.e.*, that the capacity of the model is inefficiently used by EWC). Moreover, by the 23^{rd} task, the gap between class separation and RWalk also starts to increase.

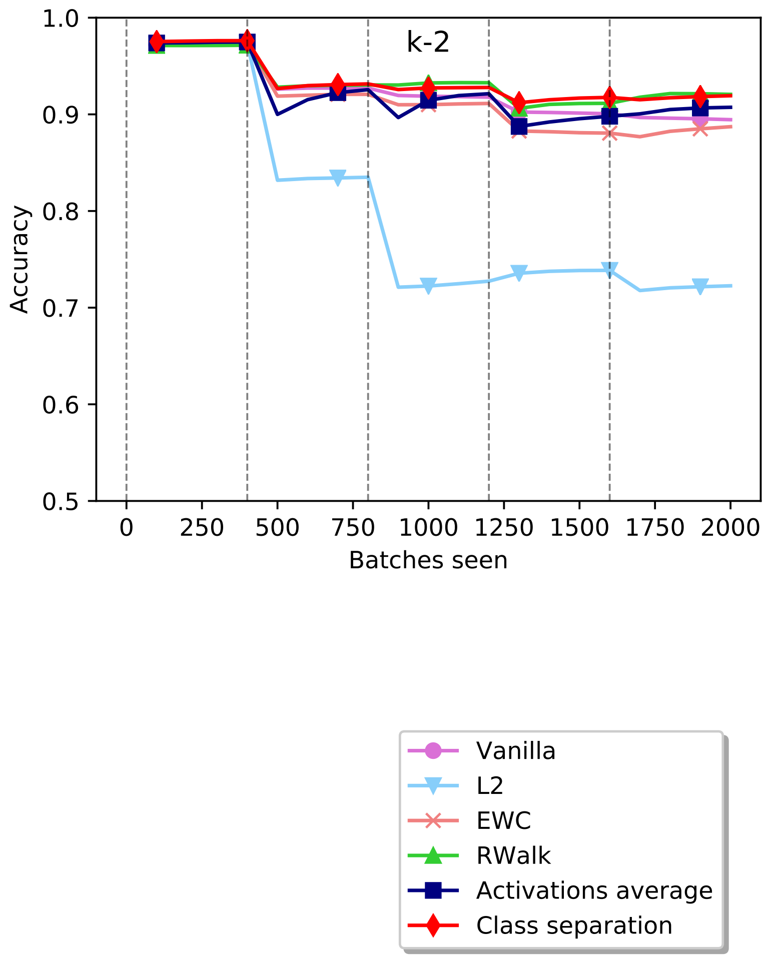

Figure 3.3: Average test accuracy on the k-2 task sequence for all tested methods. Vertical dashed lines indicate transitions between training tasks.

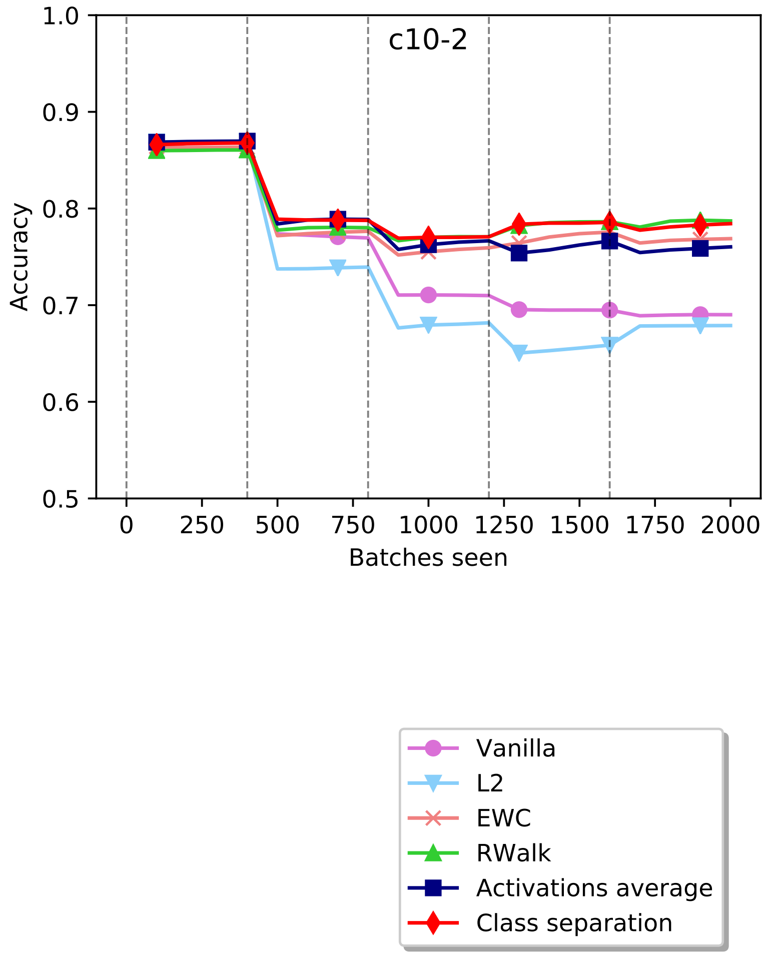

Figure 3.4: Average test accuracy on the c10-2 task sequence for all tested methods. Vertical dashed lines indicate transitions between training tasks.

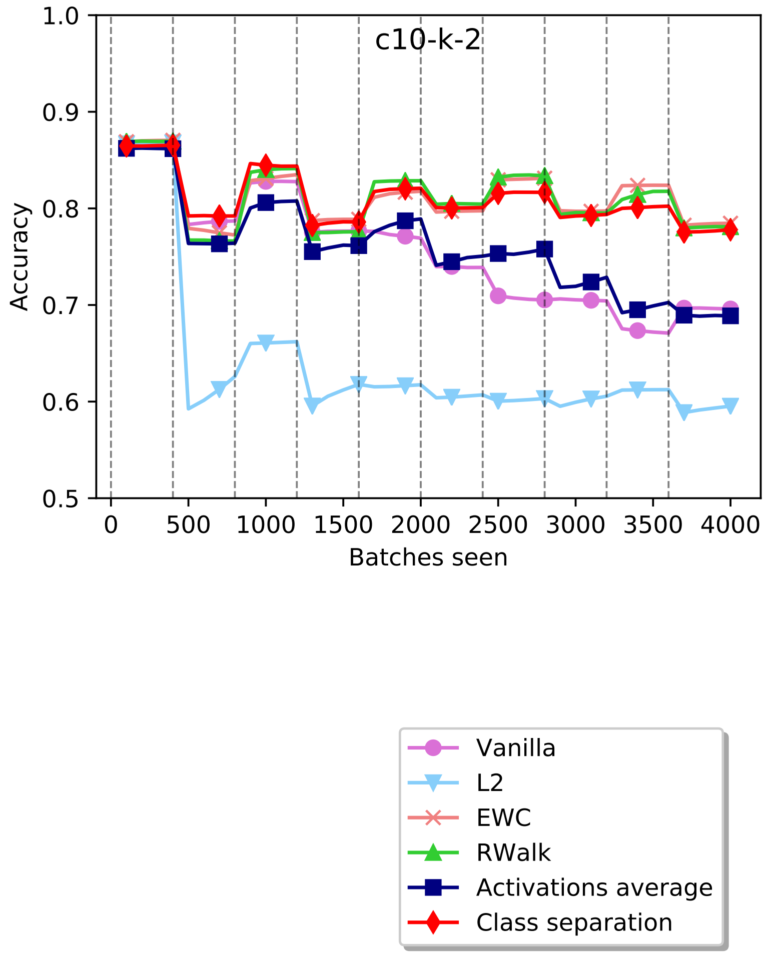

Figure 3.5: Average test accuracy on the c10-k-2 task sequence for all tested methods. Vertical dashed lines indicate transitions between training tasks.

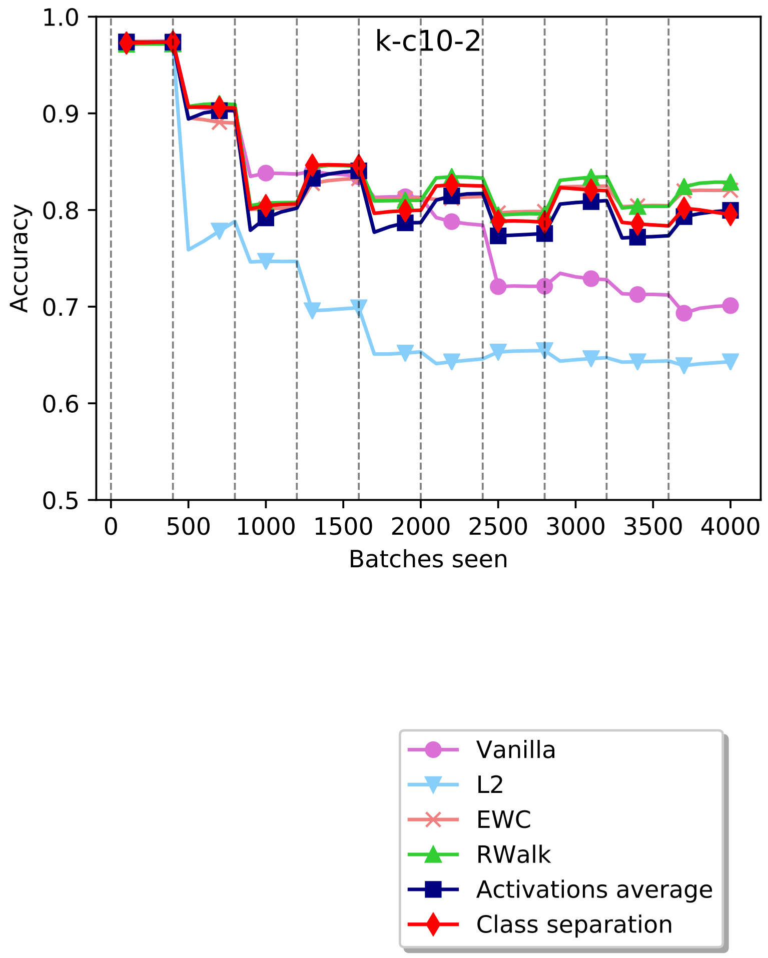

Figure 3.6: Average test accuracy on the k-c10-2 task sequence for all tested methods. Vertical dashed lines indicate transitions between training tasks.

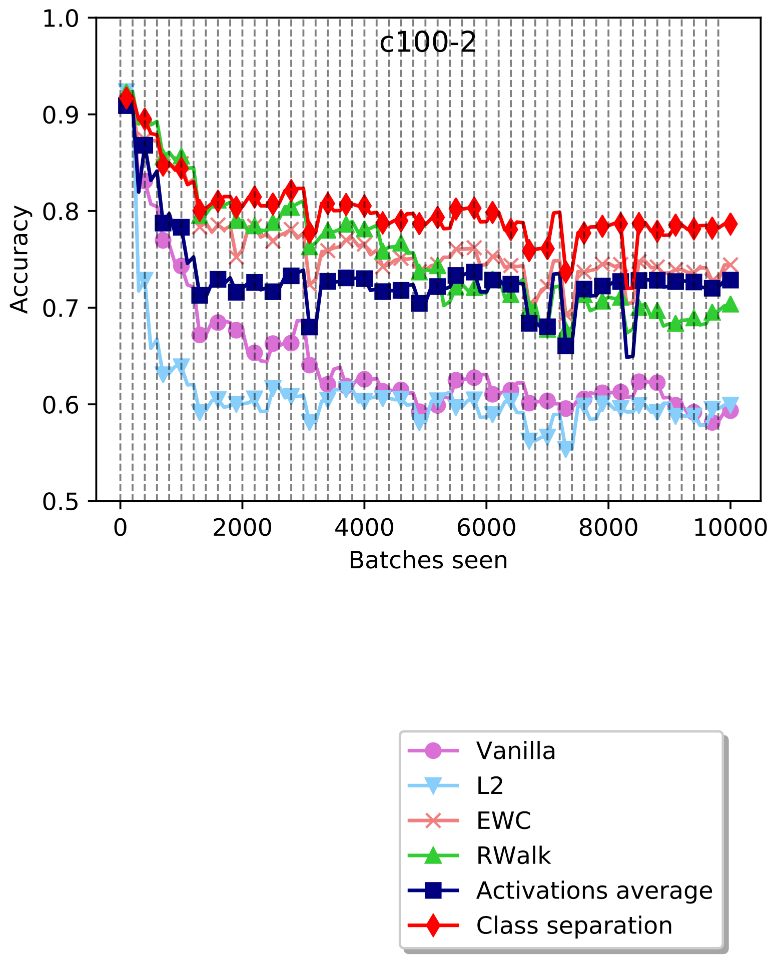

Figure 3.7: Average test accuracy on the c100-2 task sequence for all tested methods. Vertical dashed lines indicate transitions between training tasks.

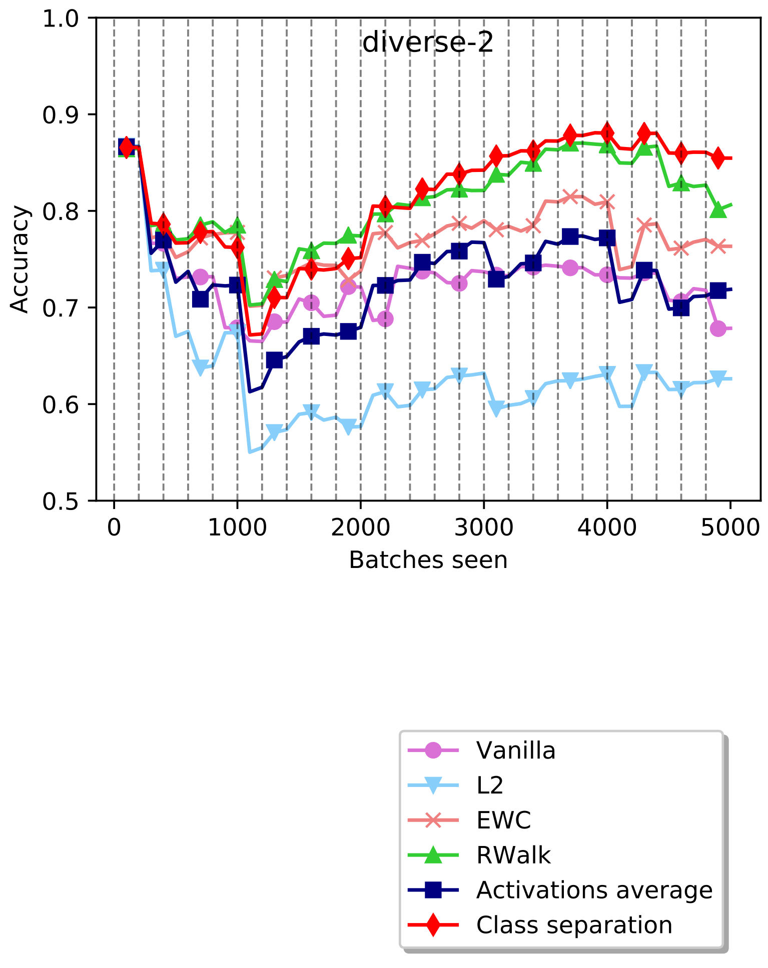

Figure 3.8: Average test accuracy on the diverse-2 task sequence for all tested methods. Vertical dashed lines indicate transitions between training tasks.

To gain more insight into the behaviour of these methods, we consider the forgetting and intransigence values shown in Table 3.1 and Table 3.2, respectively, as defined in Subsection 3.4.3. The values shown are averages across all tasks in the sequences. We observe that the proposed class separation method has lower intransigence values across all task sequences when compared to activations average. The improvement is particularly significant with longer task sequences (c100-2 and diverse-2), which partly explains the accuracy values in Fig. 3.7 and Fig. 3.8. This is also in line with our hypothesis that using class separation is more efficient in terms of capacity usage compared to the activations average method, owing to the latter's assigning high importance values to non-discriminative units.

We can also see that class separation leads to lower forgetting values in some cases (c10-2, c10-k-2, diverse-2) compared to the activations average method.

The high intransigence values of EWC for c100-2 and diverse-2 help explain its poor performance for these task sequences.

Overall, we see that class separation is effective at maintaining both forgetting and intransigence low at the same time. In other words, it is effective at preventing forgetting while still maintaining the model's capacity to learn. It is also effective in cases of longer sequences of tasks (c100-2 and diverse-2), where we observe that otherwise effective methods like RWalk and EWC perform poorly, due to either an inability to counter forgetting over a long sequence or inefficient use of model capacity.

Table 3.1: Forgetting values for all methods and task sequences.

	c10-2	c10-k-2	k-c10-2	k-2	c100-2	diverse-2
Vanilla	16.9	19.5	19.9	5.3	30.6	24.4
L2	0.8	8.7	0.3	-0.1	0.8	6.0
EWC	4.7	8.8	4.7	4.8	3.3	7.0
RWalk	1.8	8.9	3.2	0.0	18.1	10.2
Activations average	4.1	15.6	4.8	0.2	5.9	11.4
Class separation	2.2	9.3	6.9	0.7	8.1	4.4

Table 3.2: Intransigence values for all methods and task sequences.

	c10-2	c10-k-2	k-c10-2	k-2	c100-2	diverse-2
Vanilla	0.7	1.1	0.2	0.4	1.8	-0.5
L2	14.7	20.9	23.6	21.9	30.4	22.4
EWC	2.6	1.9	2.0	1.6	13.5	7.7
RWalk	3.1	2.1	2.5	2.1	3.0	0.4
Activations average	4.0	5.3	4.0	3.2	12.5	8.0
Class separation	3.1	2.1	2.5	1.6	4.5	1.1

3.4.5 Capacity Usage

The intransigence values shown in Table 3.2 support our hypothesis that the proposed class separation method is more efficient in its use of model capacity compared to the activations average method. To explore this further, we report the percentage of units in each layer that remains “free” after learning each task. By “free” we mean that the unit is assigned an importance value below a specific cut-off value. We show results for a 0.1 cut-off and a 0.01 cut-off. Free units can be re-optimized without incurring large penalties. Hence, the more free units remain in the model, the more effectively it is able to learn subsequent tasks. Note that free capacity may increase from one task to the next due to the normalization and averaging in (3.14) and (3.15).

Figures 3.9–3.14 show the free capacity for the two fully connected layers for the k-2, c10-2, c10-k-2, k-c10-2, c100-2, and diverse-2 task sequences, respectively. We compare the capacity usage of the proposed class separation method against activations average. In each figure, we show free capacity plots for a 0.1 and 0.01 cut-off values.

For the 0.1 cut-off, we can see that in most cases the proposed approach performs better (*i.e.*, has higher free capacity values) than activations average. This is more clearly seen with respect to the second fully connected layer.

When the cut-off is lowered to 0.01, we can see that class separation continues to have significant free capacity in most cases, whereas the activations average method leads to almost no units with importance values below the cut-off. For example, by the end of the c100-2 sequence, class separation still leaves around 15% free capacity in the first fully connected layer, while the activations average method is at around 0% at the same point. Once again, these results support the hypothesis that activations average, on its own, is inefficient in its use of model capacity, while class separation significantly reduces capacity usage.

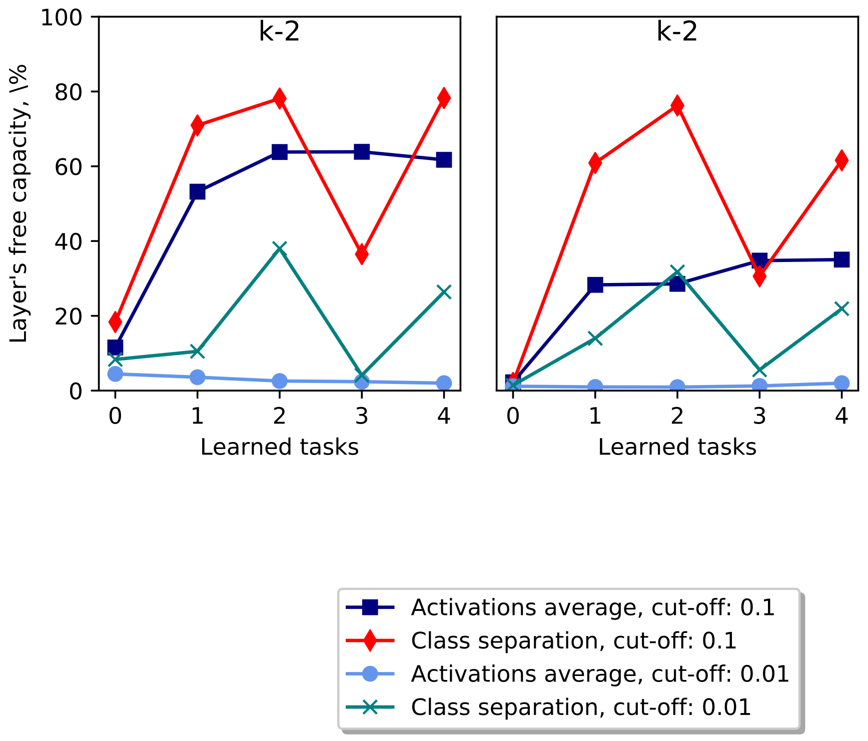

Figure 3.9: Free capacity in the fully connected layers for the k-2 task sequence. Left: free capacity in the first fully connected layer. Right: free capacity in the second fully connected layer.

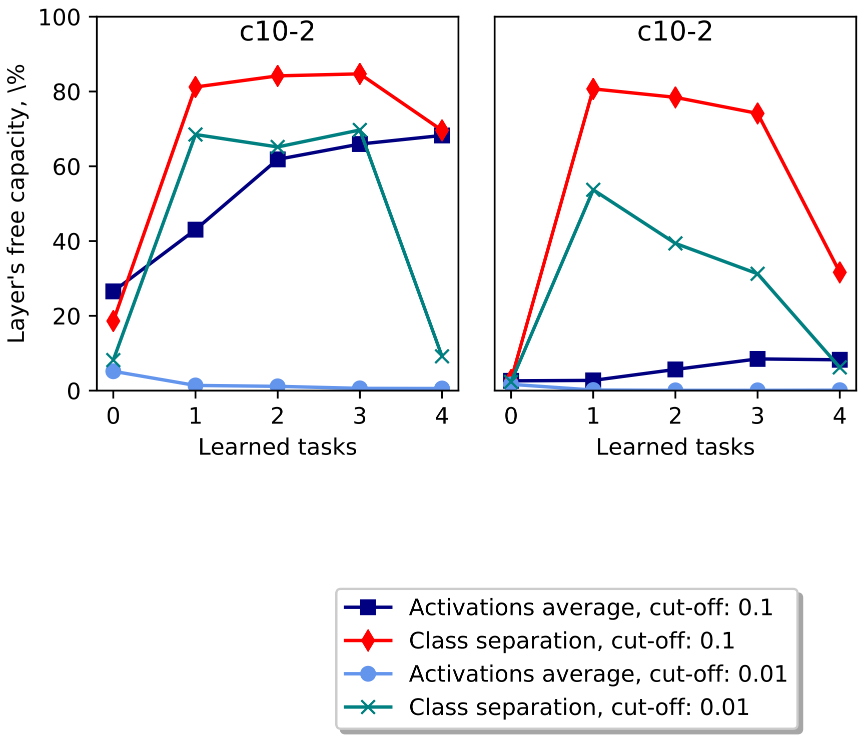

Figure 3.10: Free capacity in the fully connected layers for the c10-2 task sequence. Left: free capacity in the first fully connected layer. Right: free capacity in the second fully connected layer.

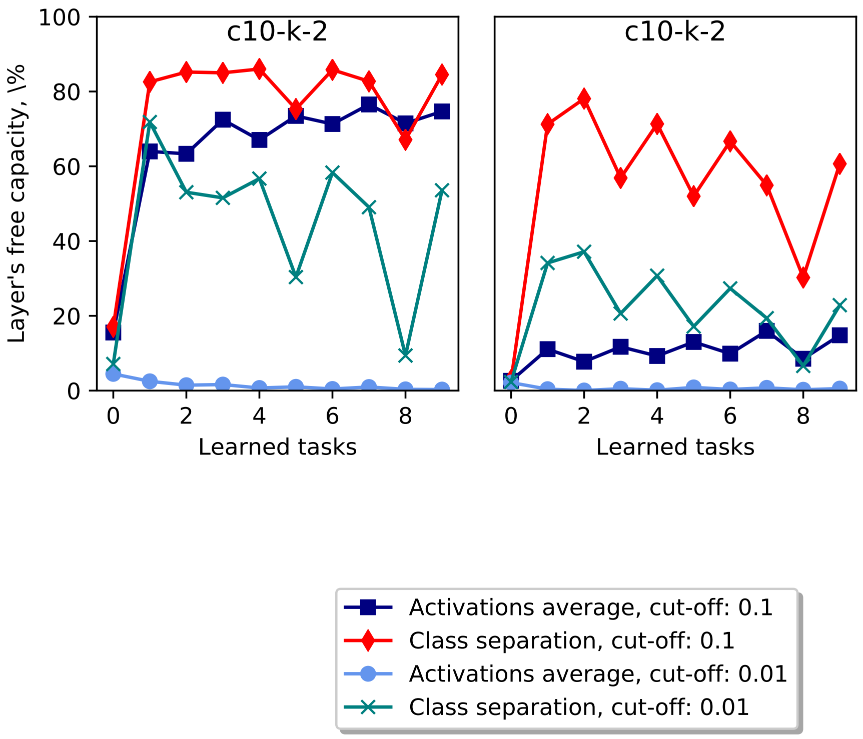

Figure 3.11: Free capacity in the fully connected layers for the c10-k-2 task sequence. Left: free capacity in the first fully connected layer. Right: free capacity in the second fully connected layer.

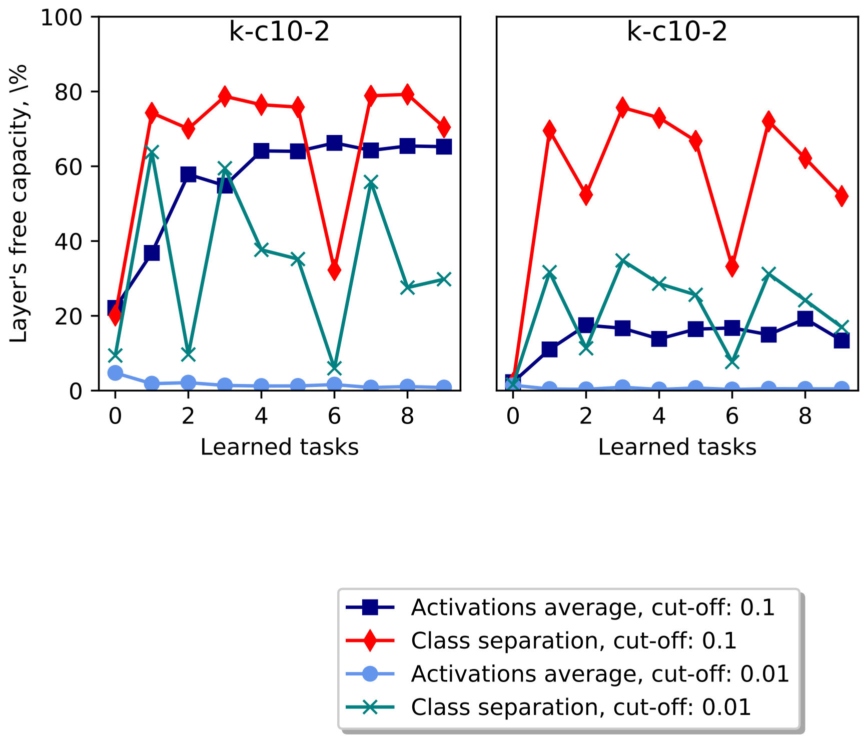

Figure 3.12: Free capacity in the fully connected layers for the k-c10-2 task sequence. Left: free capacity in the first fully connected layer. Right: free capacity in the second fully connected layer.

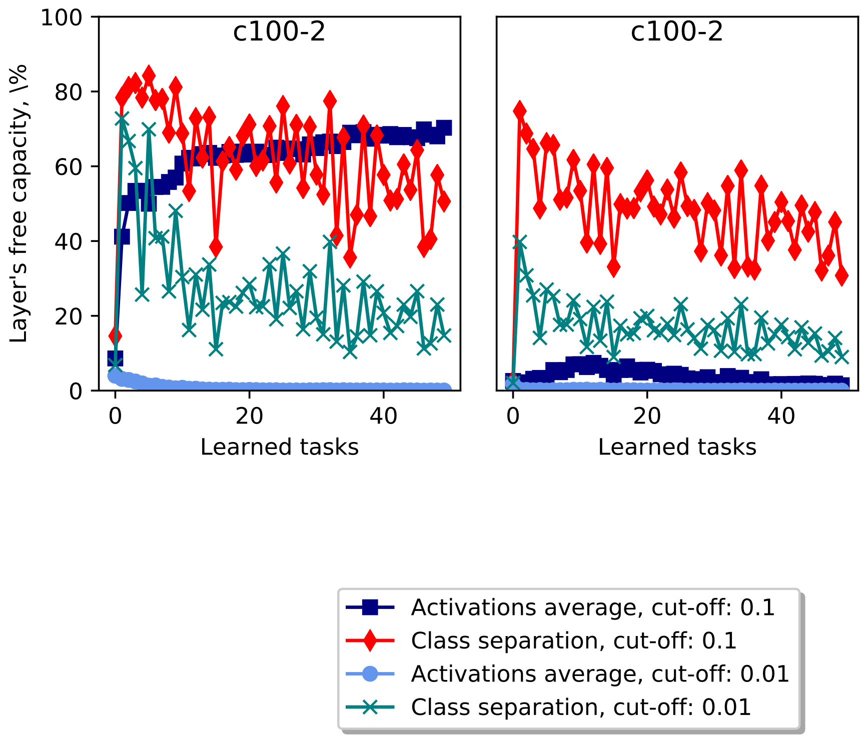

Figure 3.13: Free capacity in the fully connected layers for the c100-2 task sequence. Left: free capacity in the first fully connected layer. Right: free capacity in the second fully connected layer.

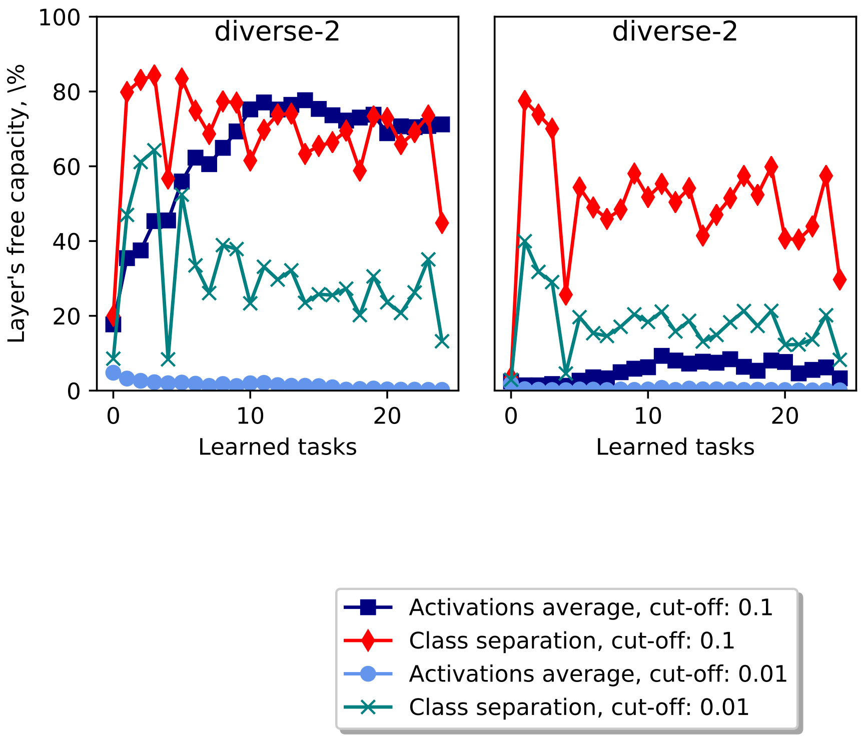

Figure 3.14: Free capacity in the fully connected layers for the diverse-2 task sequence. Left: free capacity in the first fully connected layer. Right: free capacity in the second fully connected layer.

To understand further how the two methods assign importance values to units, we inspect the individual activation values and the corresponding importance estimates after learning 10 tasks of the c100-2 task sequence. Fig. 3.15 and Fig. 3.16 show the activation values for all units[4] in the first and second fully connected layers, respectively, along with the corresponding importance values assigned by activations average and class separation. The units are sorted by class separation importance (the same plots are shown sorted by activations average importance in Fig. 3.1 and Fig. 3.2).

Based on these figures, we observe that most of the units of high activations average are largely not discriminative (on their own). And given these are the last two layers before the classification layer, assigning high importance values to these units is likely a waste of model capacity when there are much more discriminative units in the same layer. Moreover, we note that many of the highly discriminative units are assigned relatively low importance values with the activations average method. This highlights the main advantage of class separation over activations average: the latter assigns importance based on the mean of class-conditional means (for balanced classes), while the former assigns importance based on the separation between class-conditional means (in addition to spread). Hence, we see that in many cases units with low class-conditional means still have a large separation between those means, and thus should be important. While, some units with high mean of class-conditional means provide very small separation between classes, and thus, on their own, are not discriminative.

Fig. 3.17 and Fig. 3.18 show the importance values for these layers at the same point for activations average and class separation side by side. The figures show that class separation importance values taper off toward 0 much faster than the activations average values, reflecting the lower intransigence seen with the former method.

Figures 3.19–3.24 show corresponding activations and importance values in the same c100-2 task sequence, but after learning 20 tasks (*i.e.*, during the 21st task. The plots show importance assignment by activations average and class separation consistent with assignment after learning 10 tasks.

[4]Note: the exactly 0 values are due to "dead" ReLU units and they are excluded from the free capacity plots in Figures 3.9–3.14.

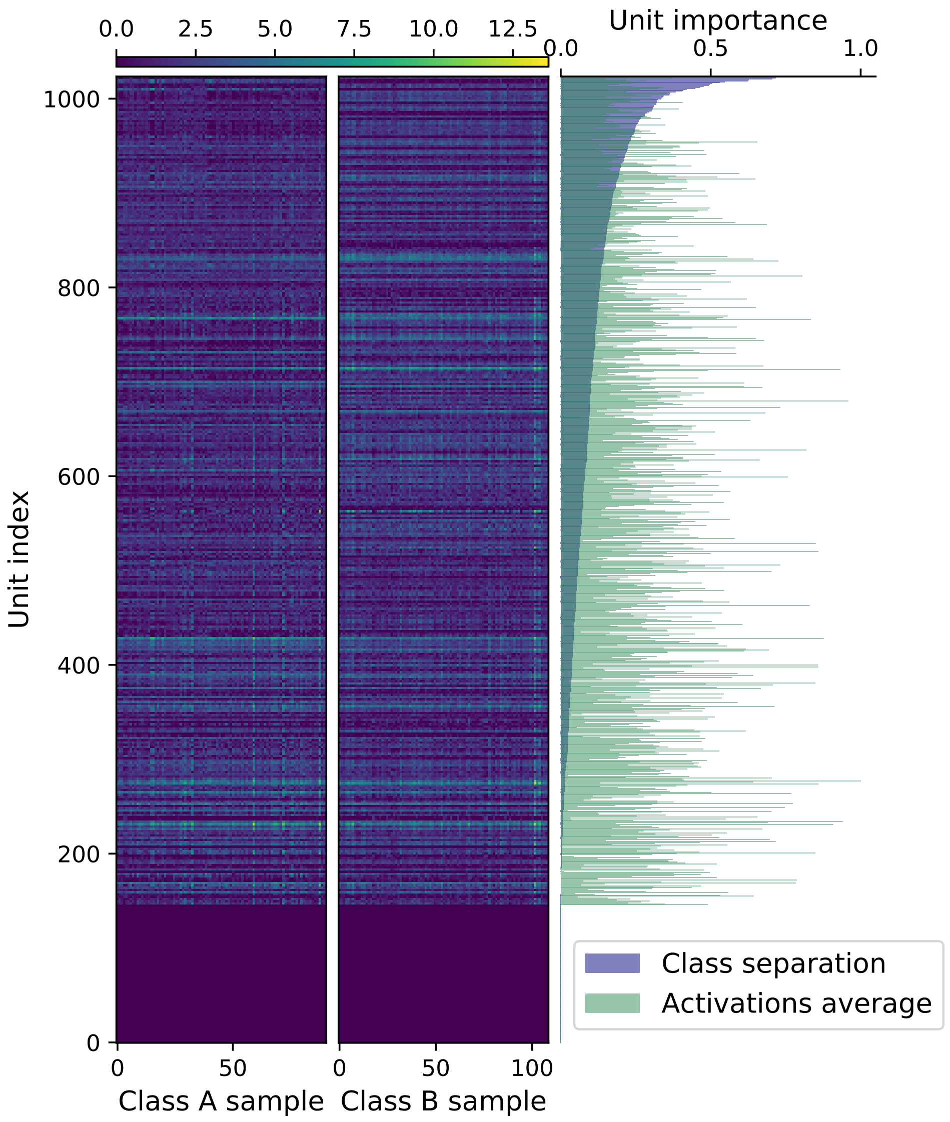

Figure 3.15: Left: activation values for the first fully connected layer for the 11th task in the c100-2 task sequence. The activation values are shown for all units and training samples. The training samples for the 2 classes are shown separately. Right: The importance values assigned by both methods to the corresponding unit. All plots are ordered according to the class separation values.

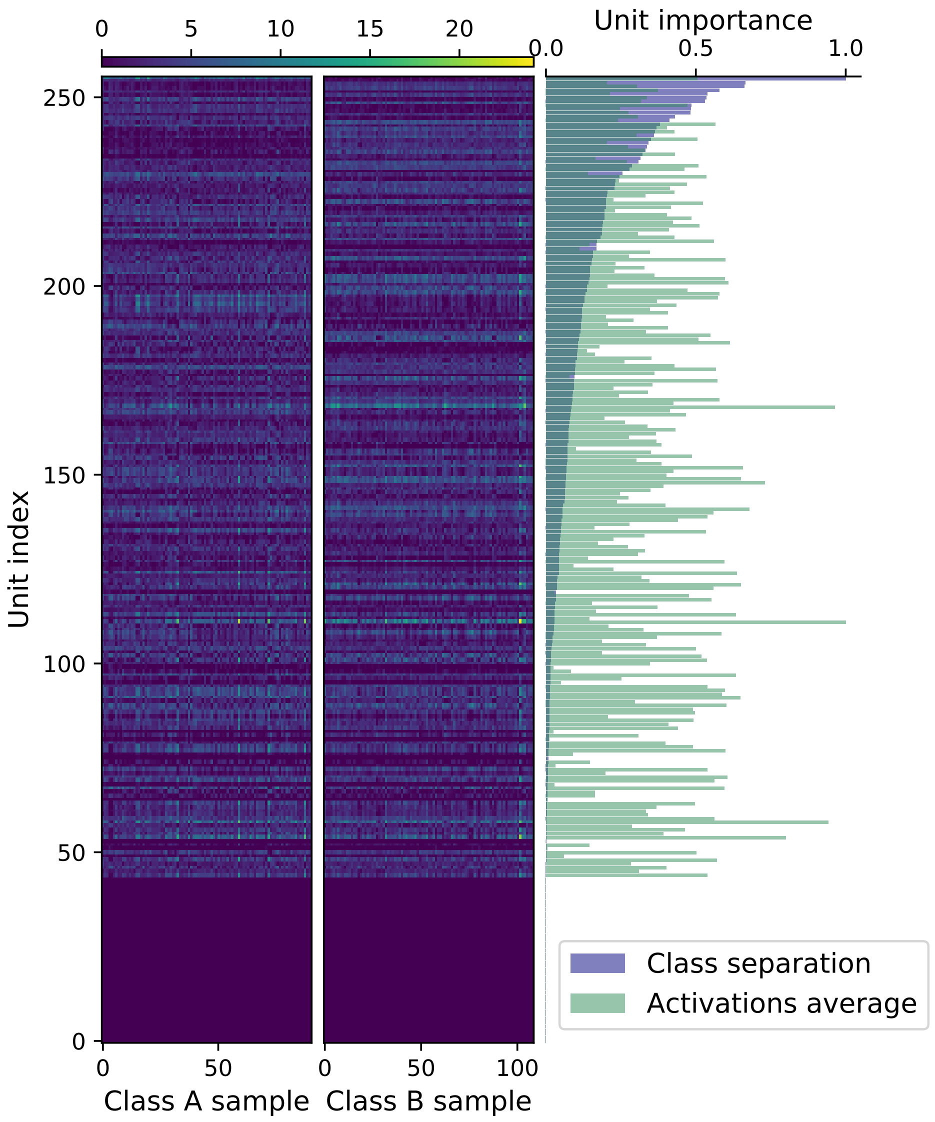

Figure 3.16: Left: activation values for the second fully connected layer for the 11th task in the c100-2 task sequence. The activation values are shown for all units and training samples. The training samples for the 2 classes are shown separately. Right: The importance values assigned by both methods to the corresponding unit. All plots are ordered according to the class separation values.

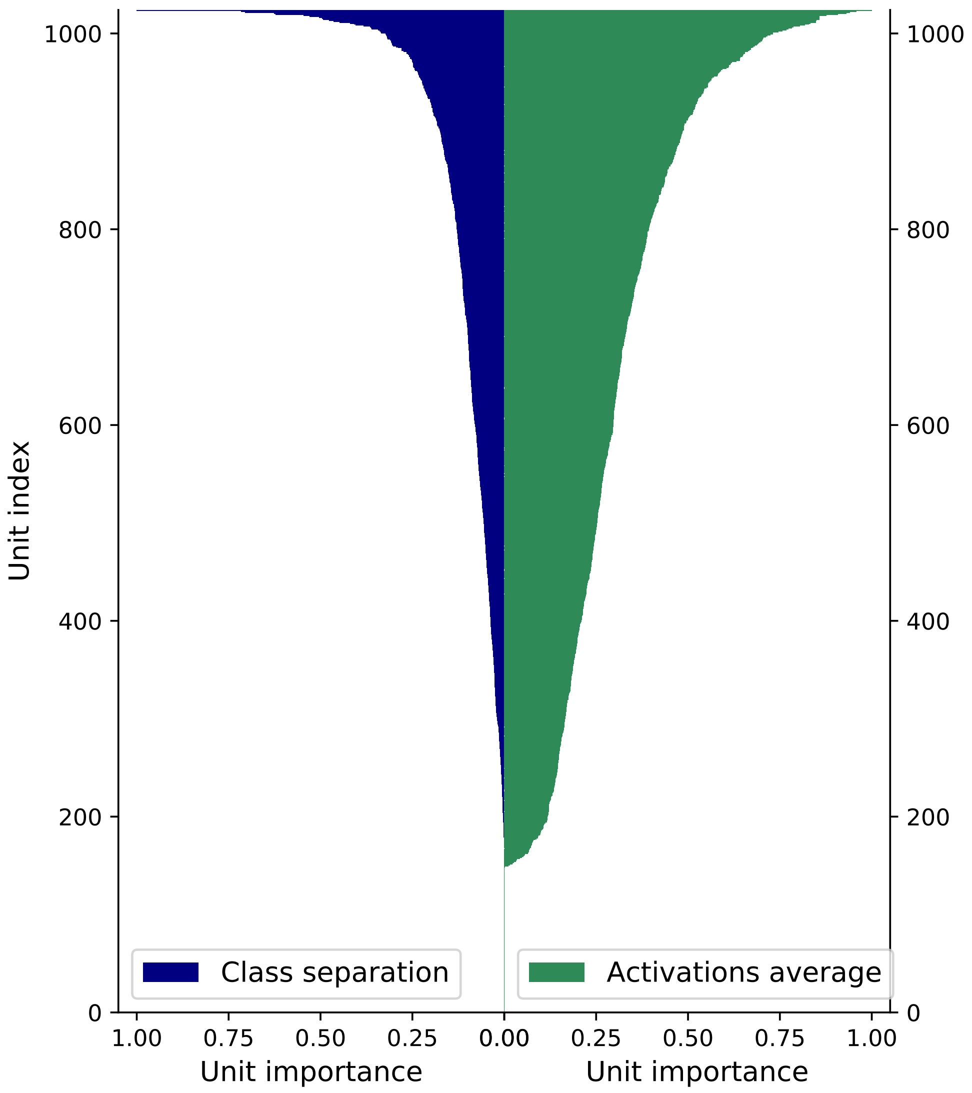

Figure 3.17: Sorted importance values for activations average and class separation. The values are shown at the 11th task in the c100-2 and for the first fully connected layer.

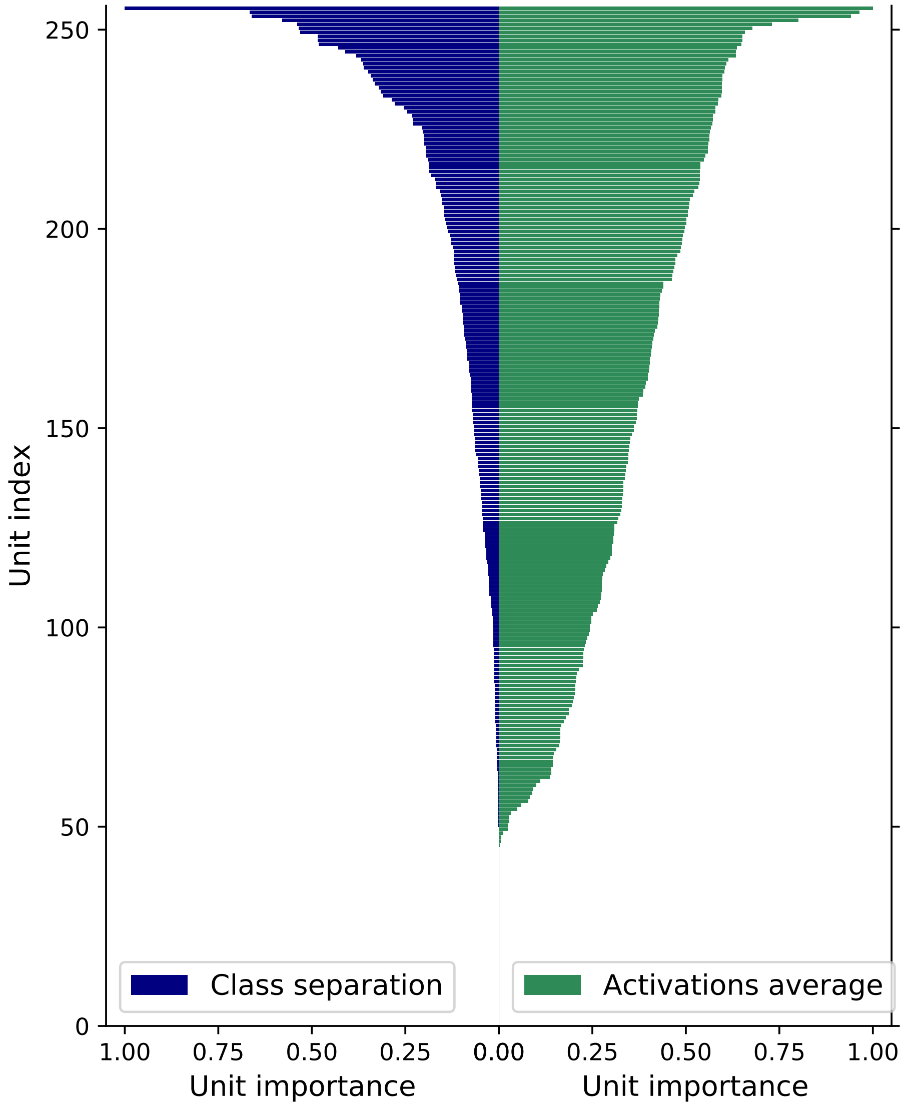

Figure 3.18: Sorted importance values for activations average and class separation. The values are shown at the 11^{th} task in the c100-2 and for the second fully connected layer.

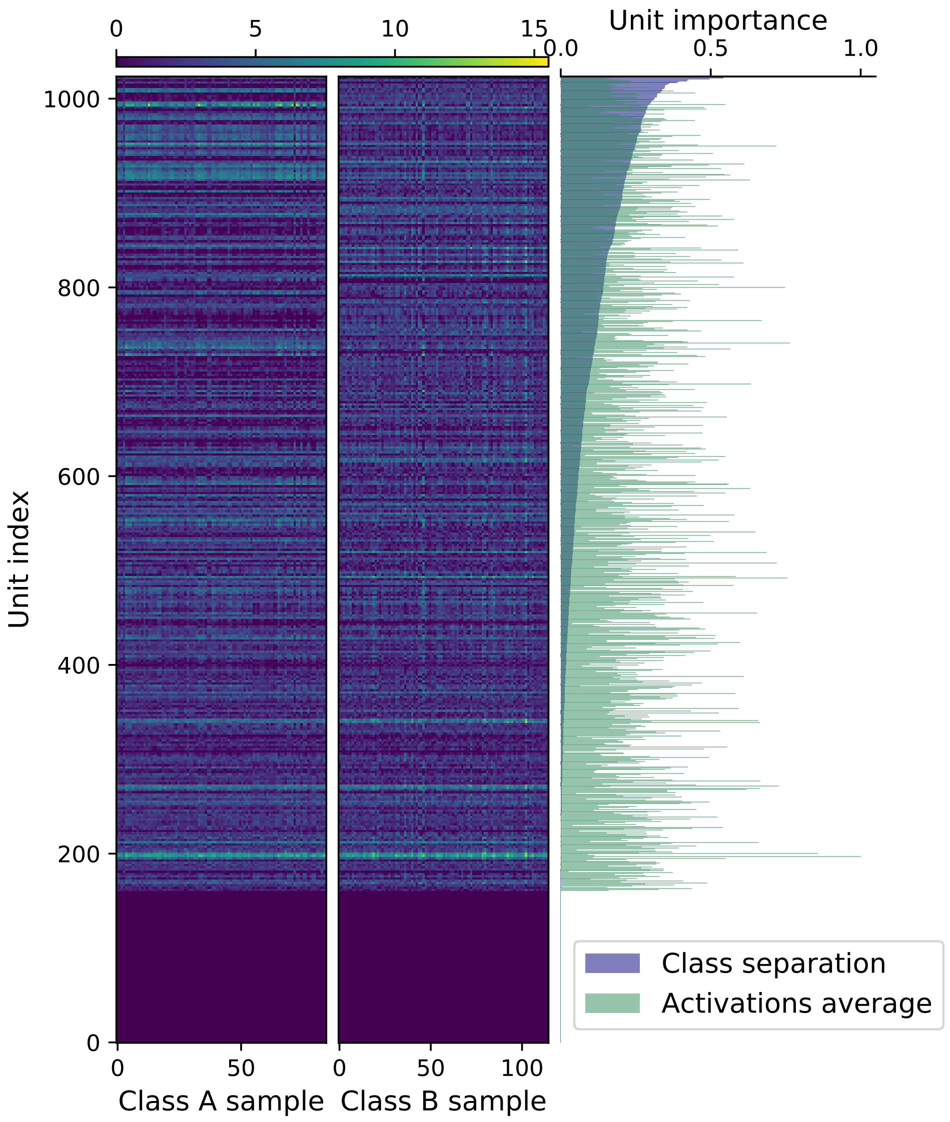

Figure 3.19: Left: activation values for the first fully connected layer for the 21^{th} task in the c100-2 task sequence. The activation values are shown for all units and training samples. The training samples for the 2 classes are shown separately. Right: The importance values assigned by both methods to the corresponding unit. All plots are ordered according to the class separation values.

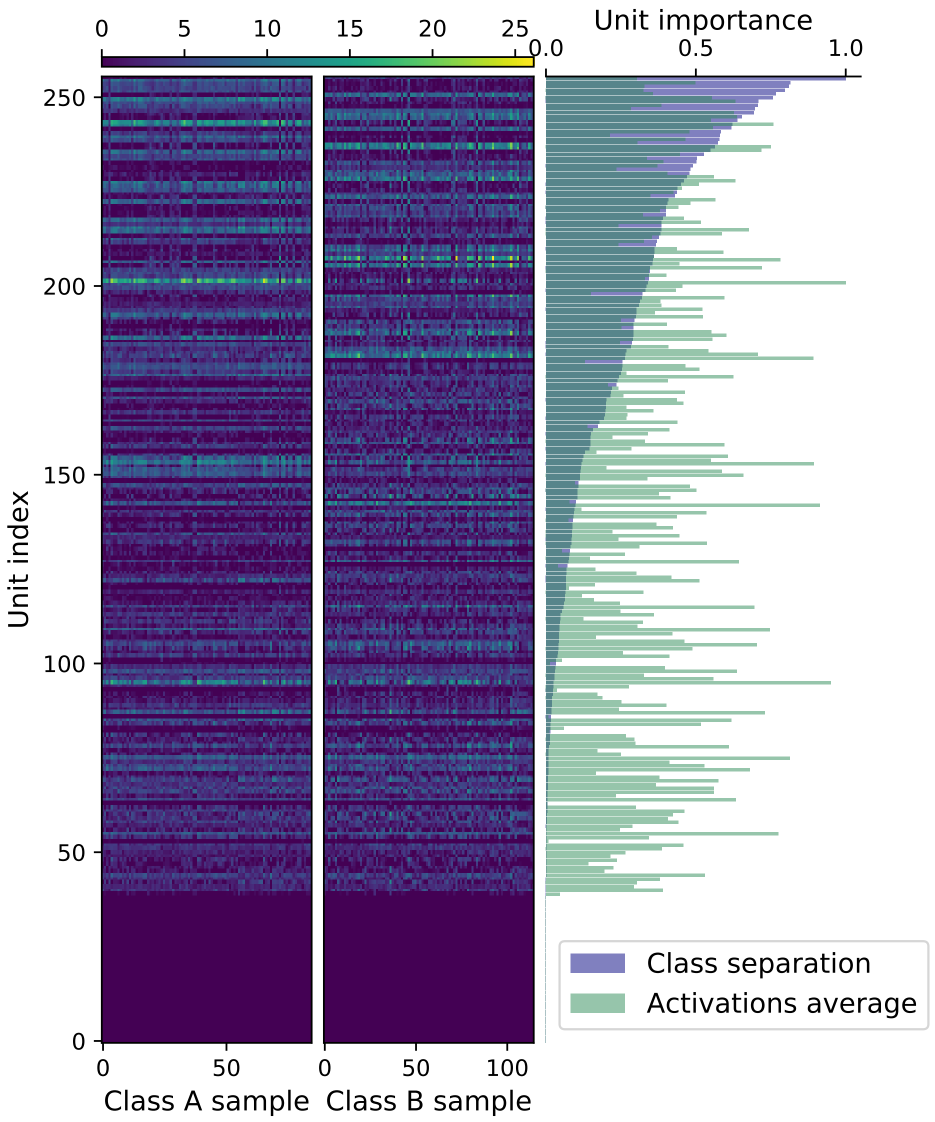

Figure 3.20: Left: activation values for the second fully connected layer for the 21th task in the c100-2 task sequence. The activation values are shown for all units and training samples. The training samples for the 2 classes are shown separately. Right: The importance values assigned by both methods to the corresponding unit. All plots are ordered according to the class separation values.

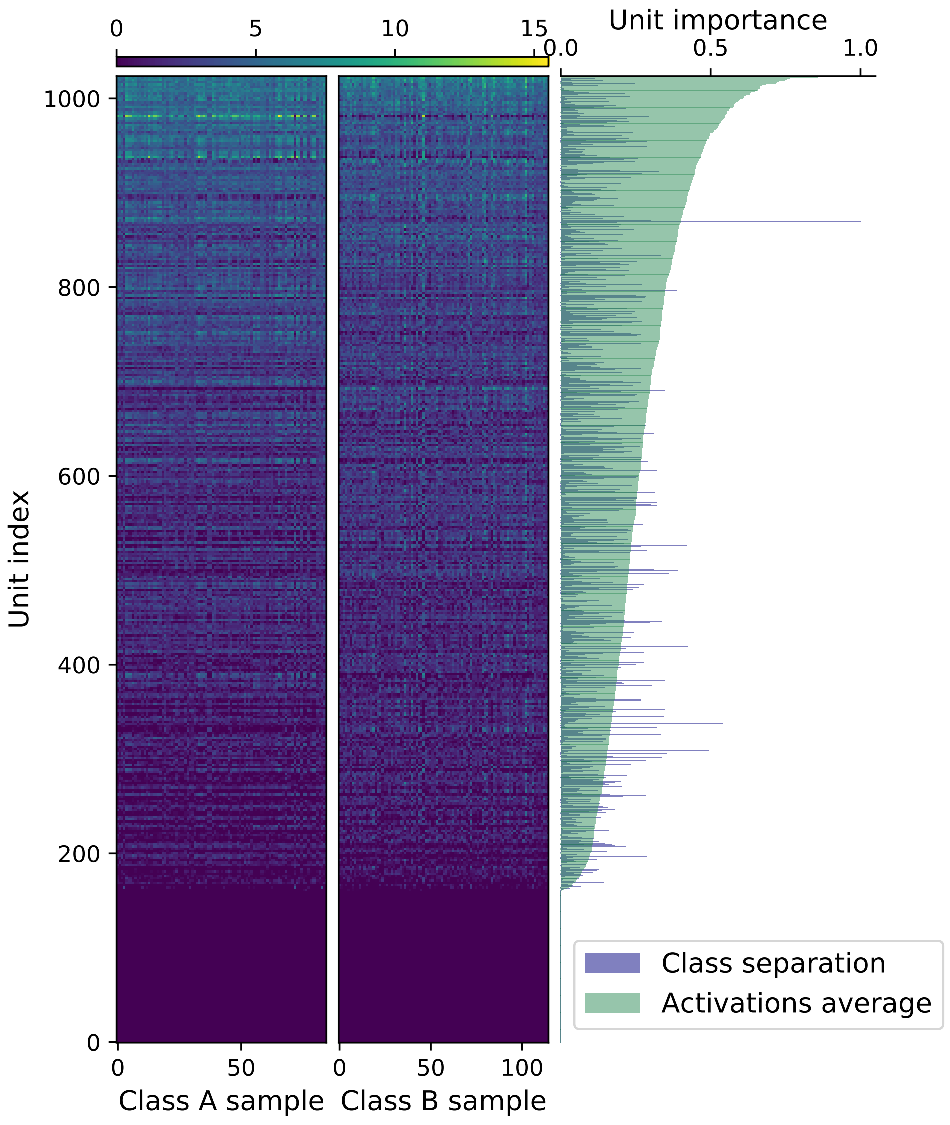

Figure 3.21: Left: activation values for the first fully connected layer for the 21th task in the c100-2 task sequence. The activation values are shown for all units and training samples. The training samples for the 2 classes are shown separately. Right: The importance values assigned by both methods to the corresponding unit. All plots are ordered according to the activation average values.

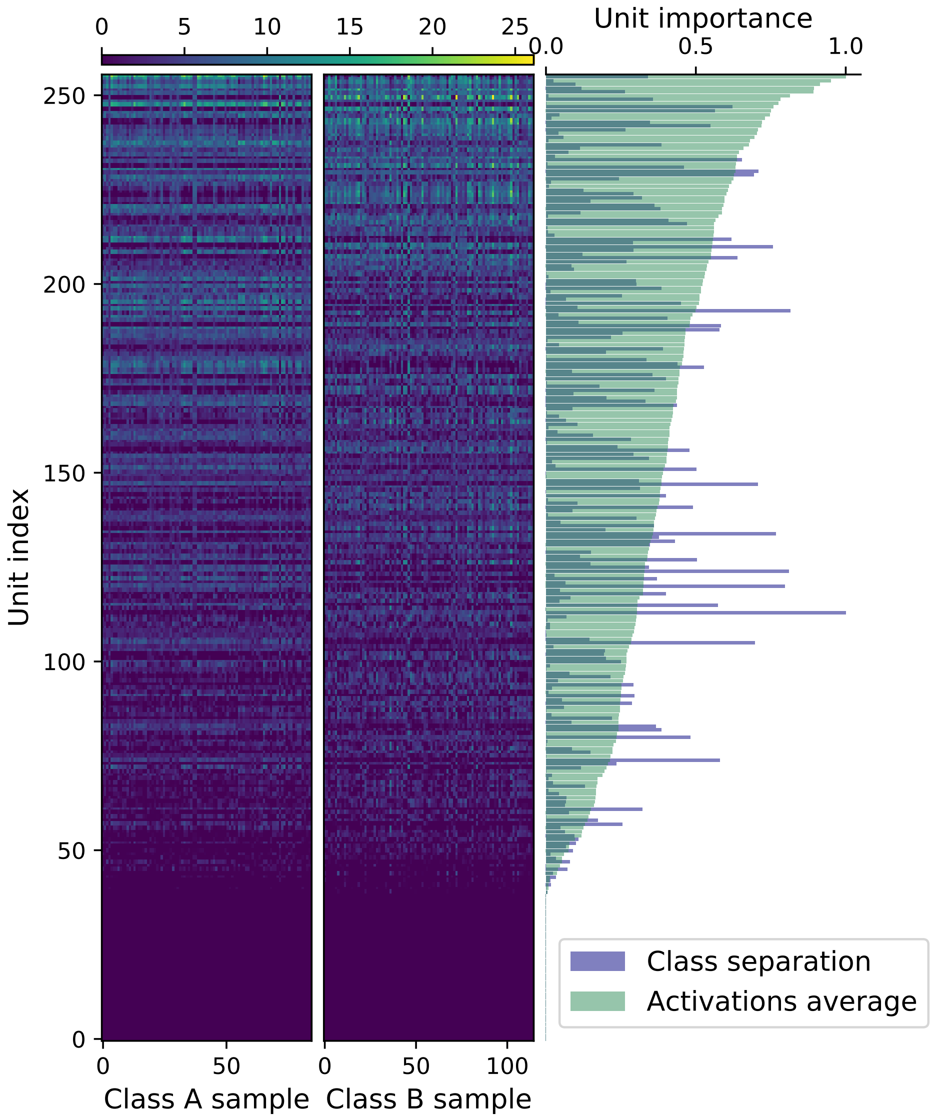

Figure 3.22: Left: activation values for the second fully connected layer for the 21th task in the c100-2 task sequence. The activation values are shown for all units and training samples. The training samples for the 2 classes are shown separately. Right: The importance values assigned by both methods to the corresponding unit. All plots are ordered according to the activation average values.

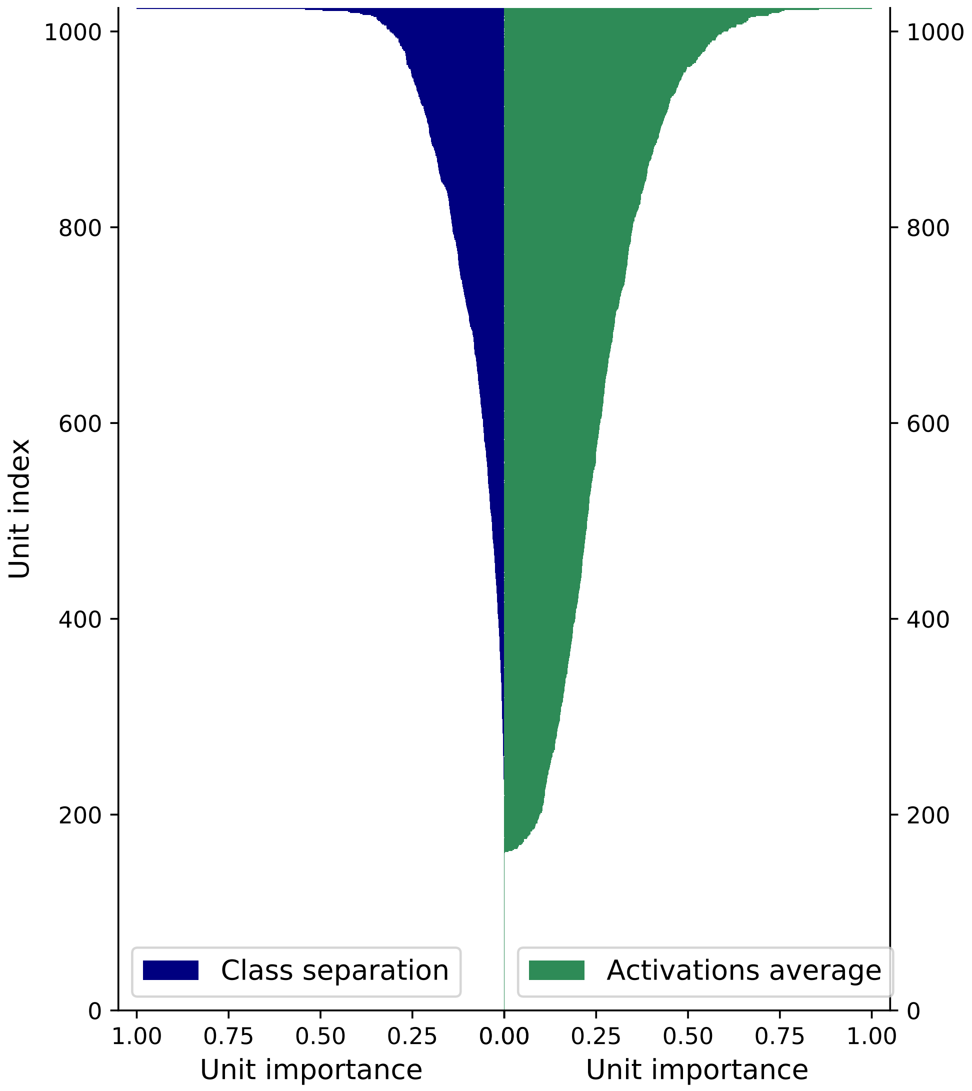

Figure 3.23: Sorted importance values for activations average and class separation. The values are shown at the 21^{th} task in the c100-2 and for the first fully connected layer.

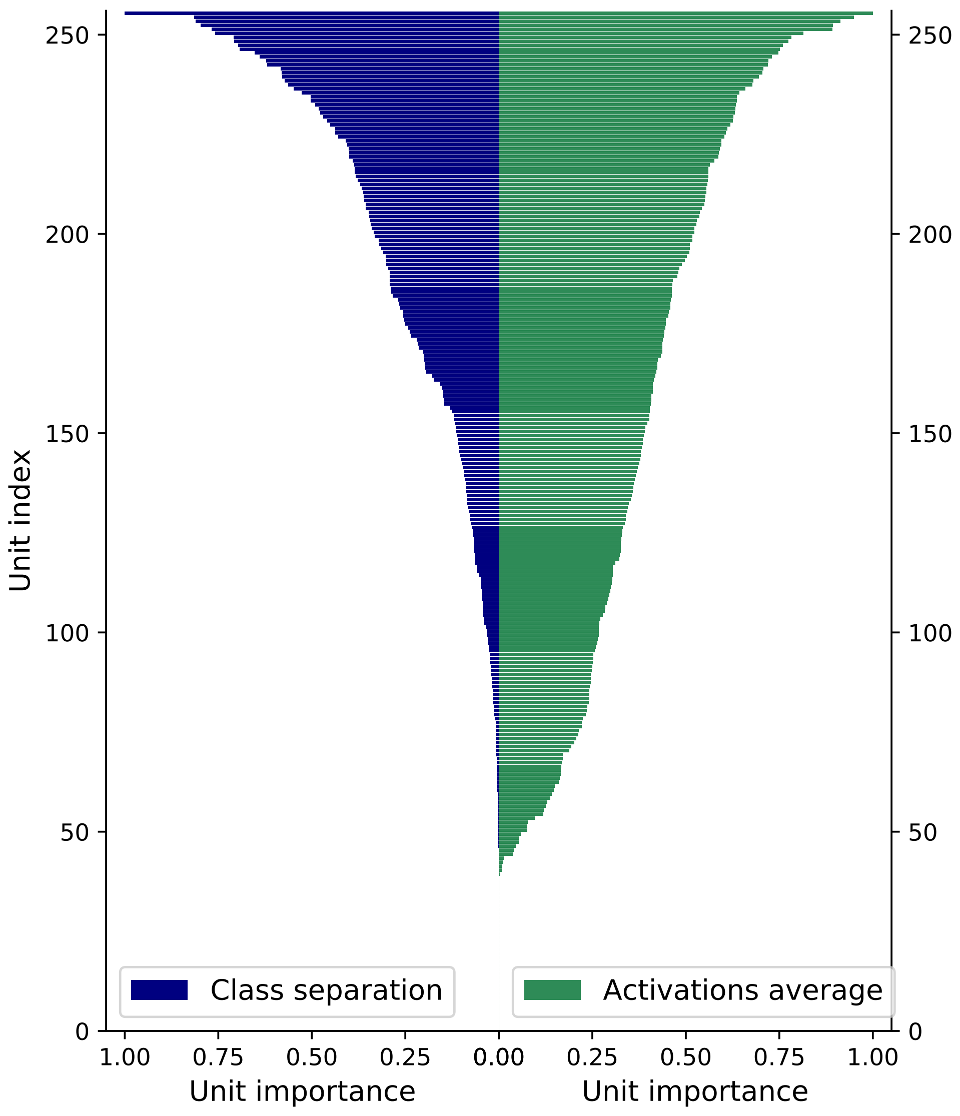

Figure 3.24: Sorted importance values for activations average and class separation. The values are shown at the 21th task in the c100-2 and for the second fully connected layer.

3.4.6 Ablation Study

The method we propose in this chapter has two parts: 1) a class separation-based importance penalty (3.13) and 2) a hidden layer supervision signal (3.16). We want to see the effect of the individual components on the overall performance. Table 3.3 shows the forgetting and intransigence values for the class separation method with and without hidden layer supervision for the 6 task sequences. In 5 out the 6 sequences, we see that the hidden layer supervision reduces both forgetting and intransigence, albeit the effect is small. We note though that this is expected, since we apply the hidden layer supervision penalty only to the first fully connected layer, which is only one layer away from the classification layer.

3.5 Summary

We proposed in this chapter a novel unit importance estimation method for use in fixed-capacity, regularization-based continual learning models. The proposed approach builds on the activations average methods recently proposed by Jung et al. (2020), and addresses some of its limitations in the layers closer to the output classification layer. The underlying idea of the proposed method is that the extent to which a unit is discriminative is an indicator of how important it is, particularly in the final layers. Based on the work on Fisher linear discriminant analysis, we use class-conditional mean separation and within-class spreads as an estimate of how discriminative a unit is. Moreover, we use the idea of companion losses proposed by Lee et al. (2015) to induce units hidden layers to become more discriminative. Through extensive evaluation on diverse image classification tasks, we showed that the proposed approach is competitive with recent methods in the literature, and attains an effective balance between countering forgetting and allowing additional learning.

In the next chapter, we address a different aspect of fixed-capacity continual learning models, namely, the durability of the learned representations absent an explicit penalty on forgetting.

Table 3.3: Class separation ablation study.

	c10-2	c10-k-2	k-c10-2	k-2	c100-2	diverse-2
			Forgetting			
With hidden layer supervision	2.2	9.3	6.9	0.7	8.1	4.4
Without hidden layer supervision	3.3	10.0	4.2	0.9	8.2	6.4
			Intransigence			
With hidden layer supervision	3.1	2.1	2.5	1.6	4.5	1.1
Without hidden layer supervision	3.3	2.4	2.8	2.1	4.3	1.2

Chapter 4

Learning Durable Representations[1]

We have learned so far that catastrophic forgetting is a major obstacle to building continual learning models. In the previous chapter, we saw how regularization-based approaches that penalize change to optimized parameter values can be used to counter forgetting. The underlying logic of those methods is that, if the parameters are prevented from changing *after* learning a task and while learning a subsequent task, then performance on the former will remain at its previously optimized level. While this clearly serves to prevent forgetting, it has the unintended consequence of preventing future learning. This remains true, to an extent, even with the more sophisticated regularization methods we saw, such as elastic weight consolidation (EWC) (Kirkpatrick et al., 2017).

We refer to methods like EWC as *post hoc* regularization approaches, because they involve optimizing the model in the usual way, as one would optimizing for an isolated task, and only after the act of learning do they attempt to counter forgetting. At that point, the only option is to force the remembering of the learned representations—representations that may not be suitable for a continual learning setting. In other words, with *post hoc* regularization, the fact that we are optimizing a continual learning model does not influence our approach to representation learning (aside from penalizing large changes to parameter values).

In this chapter, we argue that the continual learning setting must be factored in from the start, and should influence our approach to representation learning. A key factor that should drive learning representations in the continual learning setting is that these representations should be as general, as reusable, and as rich as possible. To clarify this, consider a neural network trained on a binary image classification task. It may be enough

[1]This chapter is partly based on (El Khatib and Karray, 2019a).

to solve this task by learning a small set of low-level features that happen to be discriminative for the given data distribution. In fact, recent research by Jo and Bengio (2017) suggests that conventional optimization tends to do just that. That may be acceptable in the isolated learning setting, where our only concern is to learn a narrowly defined, isolated task. We conjecture, however, that this contributes to catastrophic forgetting in the continual learning setting.

In the continual learning setting, the model is expected to continue to learn new tasks and experience new data over time. And so, learning representations that capture only the minimum set of discriminative low-level features for the current training task will necessitate significant re-optimization of the parameters to learn subsequent tasks, which in turn gives rise to forgetting. On the other hand, learning *reusable* representations reduces the extent of re-optimization, and thus reduces forgetting.

This leads to the question that we seek to address here: Given training data for some supervised classification task,[2] can we drive the optimization process such that the representations learned are reusable for a subsequent task of the same nature? We conjecture that there are two factors that lead to reusable representations while learning classification tasks: 1) representations should capture abstract or high-level features, rather than low-level statistics; and 2) representations should capture all the features present in the training data, not only the discriminative ones.

We explore in this chapter various approaches that have the potential to encourage reusable representations. In particular, we explore the effect of pre-training and auxiliary tasks, both supervised and unsupervised, on the performance of continual learning models. In addition, we propose a Kullback–Leibler (KL) divergence metric to track changes in learned representations across training tasks. We show that the use of unsupervised auxiliary tasks is the most effective among the approaches we consider at countering forgetting, rivalling in performance recent continual learning methods, such as EWC, *even without explicitly penalizing forgetting.*

The rest of this chapter is organized as follows. Section 4.1 presents the approaches we explore. Section 4.2 introduces the KL divergence metric. Section 4.3 introduces our experimental setting and results. We conclude the chapter in Section 4.4 with a discussion of future directions.

[2]In this work, we consider image classification tasks.

4.1 Encouraging Durable Representations

As we have seen in Section 2.4, a large number of continual learning approaches in the literature work by penalizing changes to model parameters *after* those parameters have been optimized for a task at an earlier point. With fixed-capacity models having finite representational capacity, there is always a balance to be struck between forcing a model to remember earlier experiences (*i.e.*, preventing forgetting) and leaving room for it to learn new experiences (*i.e.*, preventing intransigence (Chaudhry et al., 2018)). Approaches that penalize changes too severely, such as L2 regularization, often remember the first task learned but fail to learn any new task afterwards. More sophisticated methods, such as EWC (Kirkpatrick et al., 2017) and RWalk (Chaudhry et al., 2018), still have to wrestle with the same issue, but can generally achieve lower intransigence by making use of per-parameter penalties that are scaled by each parameter's importance, as we saw in the previous chapter. We refer to such regularization approaches as *post hoc* methods (El Khatib and Karray, 2019a) because they do not factor in the continual learning requirement from the start (this is evident in the way such models learn the very first task), and only attempt to force remembering after the fact of learning.

We attempt here to tackle the continual learning problem from another angle: driving models toward learning durable representations that are less susceptible to forgetting in the first place. Considering that forgetting is a by-product of the series of re-optimizations that a model goes through when learning tasks in sequence, our hypothesis is that a model that can learn more general, reusable representations will require a lower degree of re-optimization and consequently will be less susceptible to forgetting.

In this section, we describe methods that have the potential to achieve more durable representations. We evaluate the effectiveness of these methods in Section 4.3.

4.1.1 Auxiliary Tasks

One way to drive continual learning models toward learning durable representations is to encourage the learning of parameters that are more general than the training task at hand. For example, for a continual learning model that at some point is learning two classes of images, consider these two optimization strategies:

1. Dedicating the entire capacity of the model toward performing well on this single task; or

2. Dedicating the model's capacity toward capturing as much information as is available in the input data, regardless of whether it is discriminative in the current training task, while simultaneously performing well on the task at hand.

The first strategy is the standard isolated learning framework, which when ported to the continual learning setting results in frequent re-optimization of the model's parameters. And thus, to prevent forgetting, *post hoc* continual learning methods, which adopt this strategy, resort to explicit penalties on forgetting, which in turn hinder future learning, to varying degrees.

We hypothesize that the second strategy is more suited for continual learning models that are expected to accumulate knowledge and tasks over time, and that it reduces forgetting without an explicit penalty on it. We validate this hypothesis in section 4.3.

The question then becomes: how do we drive models to learn more general and reusable representations from the same training data? We explore two types of auxiliary tasks to answer this: unsupervised and supervised. In both cases, we constrain the model to use only the training data of the current training task (*i.e.*, the auxiliary tasks have to be derived in real-time during learning from the primary task). Thus, we explore only approaches that do not require a replay memory or access to external sources of data.

Unsupervised Auxiliary Tasks

Figure 4.1 shows the architecture we use for unsupervised auxiliary tasks. As is standard in image classification models, such as convolutional neural networks (CNN), the model consists of a sequence of layers (the "core network") that encode the input images into a higher-level representation. An output layer subsequently computes class scores from this higher-level representation. We extend this standard architecture with a "reconstruction network". The auxiliary task then is to reconstruct an input image given its higher-level representation at the output of the core network.

Formally, given an input image $\mathbf{x}$ and its one-hot encoded target class $\mathbf{t}$, the output of the core network is given by:

$$\mathbf{z} = f_{\theta_{\mathbf{c}}}(\mathbf{x}),$$

where $\theta_{\mathbf{c}}$ stands for the core network's parameters and $f_{\theta_{\mathbf{c}}}$ is the input-output function corresponding to the core network. The output of the output layer is a probability distribution over classes:

$$\mathbf{s} = {\theta_{\mathbf{o}}}^{\top}\mathbf{z},$$

$$\hat{y}_i = \frac{e^{s_i}}{\sum_j e^{s_j}},$$

where s_i and y_i are class score and class probability for class i, respectively, and $\hat{\mathbf{y}} = [\hat{y}_1, \cdots, \hat{y}_C]$ is the softmax of $\mathbf{s} = [s_1, \cdots, s_C]$, C being the number of classes.

The output of the reconstruction network is given by:

$$\hat{\mathbf{x}} = g_{\theta_\mathbf{r}}(\mathbf{z}),$$

where $\theta_\mathbf{r}$ represents the parameters of the reconstruction network and $g_{\theta_\mathbf{r}}$ is the input-output mapping represented by the reconstruction network.

To simultaneously optimize for the primary and auxiliary tasks, we minimize the following loss function:

$$L_{\text{cls}}(\hat{\mathbf{y}}, \mathbf{t}) + \lambda L_{\text{rec}}(\hat{\mathbf{x}}, \mathbf{x}),$$

where $L_{\text{rec}}(\hat{\mathbf{x}}, \mathbf{x})$ is the auxiliary unsupervised reconstruction loss, λ is a regularization coefficient, and $L_{\text{cls}}(\hat{\mathbf{y}}, \mathbf{t})$ is the standard negative log likelihood loss for the current classification task, given by:

$$-\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{C} t_i^{(j)} \log \hat{y}_i^{(j)}, \tag{4.1}$$

where we have introduced the superscript $^{(j)}$ to denote the j^{th} training sample, and N represents the number of training samples.

For L_{rec}, we experimented with both a mean squared error loss and a binary cross entropy loss. Based on our initial experiments, we found the latter to perform better, and therefore for the rest of this work, we use:

$$L_{\text{rec}} = -\frac{1}{D}\sum_{ijk} [x_{ijk} \log \hat{x}_{ijk} + (1 - x_{ijk})(1 - \log \hat{x}_{ijk})],$$

where i, j, and k index the width, height, and depth of the input, and D is the flattened dimension of the input space. For simplicity, we show this loss here for a single sample. Note that we apply a sigmoid transformation at the end of the reconstruction network.

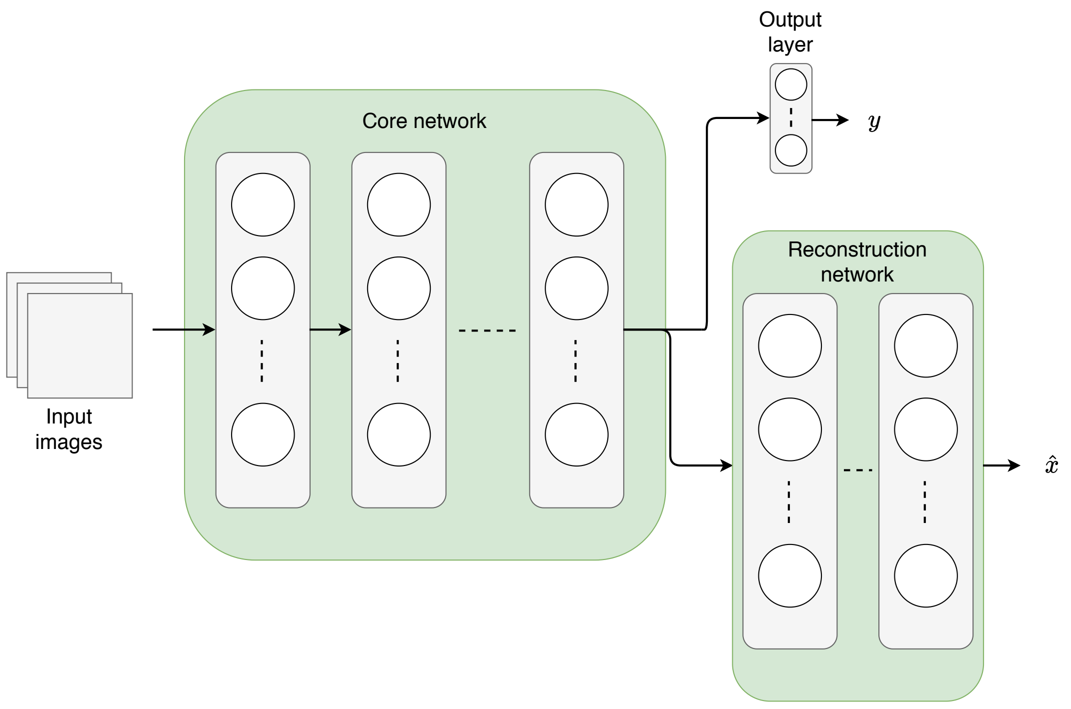

Figure 4.1: The architecture used with unsupervised auxiliary tasks. The core network maps the input images to a higher-level representation. The reconstruction network maps the higher-level network back to the input space to reconstruct the images.

Supervised Auxiliary Tasks

In the same way that we use unsupervised reconstruction auxiliary tasks to encourage more general, reusable representations, we also explore the use of supervised auxiliary tasks. However, we again constrain ourselves to using only the training data available as part of each task, without additional human supervision. In order to create auxiliary supervised tasks automatically, we rely on the clustering-based framework described in Algorithm 1. For each new task and its corresponding training images, we automatically create additional label assignments over the images using multiple clustering algorithms, and then use the new image-label pairs as auxiliary supervised tasks. The premise of this approach is that the clustering algorithms will assign images to clusters based on patterns that are not discriminative in the primary task, thus driving the core network to capture more of the features in the training data.

Fig. 4.2 depicts the architecture used with supervised auxiliary tasks.

During training, we optimize the following loss function (shown for a single training sample j):

$$L_{\text{cls}}(\hat{\mathbf{y}}^{(j)}, \mathbf{t}^{(j)}) + \lambda \sum_k L_{\text{cls}}(\hat{\mathbf{y}}^{(j,k)}, \mathbf{t}^{(j,k)}), \tag{4.2}$$

where k indexes the auxiliary task heads, λ is a regularization coefficient, and $\hat{\mathbf{y}}^{(j,k)}$ is the predicted probability distribution for sample j at the output of auxiliary head k. L_{cls} in both cases is the standard classification loss shown in (4.1).

The K output heads and auxiliary sets of labels are stored temporarily, and are discarded after each training task.

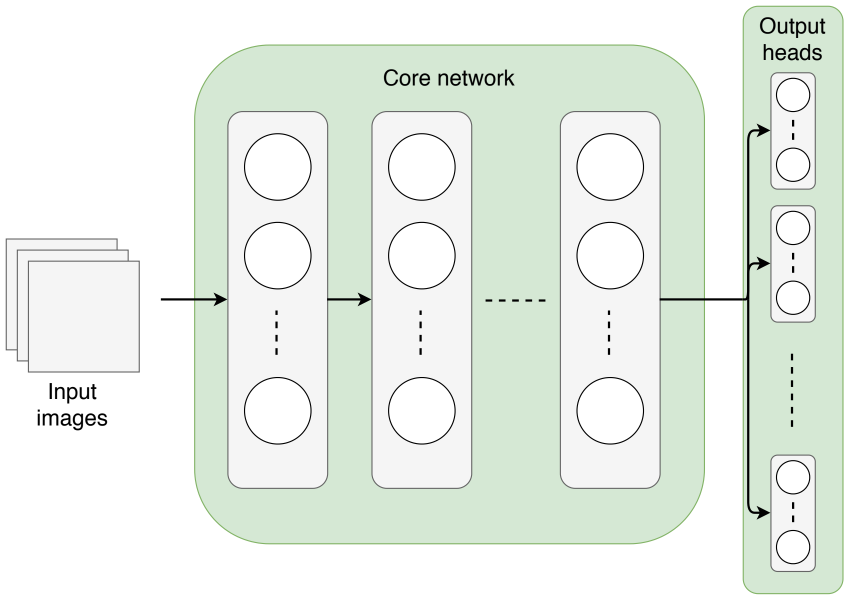

Figure 4.2: The architecture used with supervised auxiliary tasks. The core network maps the input images to a higher-level representation. A set of auxiliary output heads is added to the primary classification head. Each auxiliary head corresponds to a different clustering of the training images.

Algorithm 1 Continual learning process for task $\mathcal{T}$ using supervised auxiliary tasks

```
Require: input samples $\{(\mathbf{x}^{(j)}, \mathbf{t}^{(j)})\}_{j=1}^{N}$ for task $\mathcal{T}$
Require: set of clustering algorithms $\{\mathcal{R}_1, \mathcal{R}_2, ...\mathcal{R}_K\}$
Require: continual learning model, core parameters $\theta_c$
Require: gradient descent algorithm
1: add new head to model corresponding to task $\mathcal{T}$
2: for $k$ in $\{1, 2, ..., K\}$ do
3:     use $\mathcal{R}_k$ to cluster samples $\{\mathbf{x}^{(j)}\}_{j=1}^{N}$
4:     store cluster label assignments $\{\mathbf{t}^{(j,k)}\}_{j=1}^{N}$
5:     add new head to model corresponding to $\{\mathbf{t}^{(j,k)}\}_{j=1}^{N}$
6: end for
7: use gradient descent to minimize cost function (4.2)
8: discard the $K$ temporary heads and label assignments
9: return $\theta_c$, and trained head for task $\mathcal{T}$
```

4.1.2 Pre-training

So far, we have described auxiliary tasks that are learned simultaneously with the primary tasks. We also explore and evaluate the use pre-training in continual learning models. As with auxiliary tasks, we explore both supervised and unsupervised pre-training. The pre-training tasks are exactly the same as the ones described in Subsection 4.1.1, except they are derived from the training data of the first task in the training task sequence. In other words, we continue to constrain the model to the data available with each training task.

Pre-training, supervised or unsupervised, is performed for a certain number of epochs before the very first primary training task. We test whether such a framework leads to a better starting position for the subsequent optimizations, and whether it has a lasting, positive effect on performance.

4.2 Tracking Representation Changes

4.2.1 KL Divergence

Our hypothesis is that learning more general, reusable representations, as opposed to representations that are specific to solving the training task at hand, will reduce the amount of re-optimization per task, and consequently reduce forgetting.

To test this hypothesis, we introduce here a KL divergence-based measure to track internal changes in the model (rather than simply monitoring the performance on specific tasks). This gives us greater insight into representational shifts during optimization that are induced by the methods we test.

The KL divergence between two discrete probability distributions p and q is defined as:

$$D_{\mathrm{KL}}(p||q) = \sum_{y \in \mathcal{Y}} p(y) \log(\frac{p(y)}{q(y)}). \tag{4.3}$$

Or, as an expectation:

$$D_{\mathrm{KL}}(p||q) = \mathrm{E}_{y \sim p(y)}[\log(p(y)) - \log(q(y))]. \tag{4.4}$$

Now, let $p_{\theta_1}(\mathbf{y}|\mathbf{x})$ represent the probability distribution over task $\mathcal{T}_1$ classes, given input $\mathbf{x}$, represented by the model after learning $\mathcal{T}_1$.[3] And let $p_{\theta_2}(\mathbf{y}|\mathbf{x})$ be the corresponding distribution (*i.e.*, over task $\mathcal{T}_1$ classes) after learning task $\mathcal{T}_2$. For the model to continue to perform well on task $\mathcal{T}_1$ after having learned task $\mathcal{T}_2$, $p_{\theta_2}(\mathbf{y}|\mathbf{x})$ should remain "close" to $p_{\theta_1}(\mathbf{y}|\mathbf{x})$. Referring to (4.4), the shift in model distribution over task $\mathcal{T}_1$ classes going from θ_1 to θ_2 can be written as:

$$D_{\mathrm{KL}}(p_{\theta_1}||p_{\theta_2}) = \mathrm{E}_{\mathbf{x}\sim \mathrm{D}_{\mathcal{T}_1}, \mathbf{y}\sim p_{\theta_1}(\mathbf{y}|\mathbf{x})}[\log(p_{\theta_1}(\mathbf{y}|\mathbf{x})) - \log(p_{\theta_2}(\mathbf{y}|\mathbf{x}))], \tag{4.5}$$

where $\mathrm{D}_{\mathcal{T}_1}$ is the input distribution of task $\mathcal{T}_1$.

Therefore, to estimate the representational shift going from any task $\mathcal{T}_i$ to any subsequent task $\mathcal{T}_j$, one can use $D_{\mathrm{KL}}(p_{\theta_i}||p_{\theta_j})$ shown in (4.5). A larger shift in KL divergence is, in general (but not necessarily), associated with a larger degree of forgetting.

In our implementation, we use a slightly simplified version of (4.5). First, we sample $\mathbf{x}$ from the union of all training datasets in the task sequence. Second, instead of monitoring the distribution shift over the classes of each task separately, we use a dummy, fixed output layer, and monitor the distribution over its classes instead. This allows us to monitor a single average representational drift in the core network with respect to all tasks.

4.2.2 Other Measures

While we use the KL divergence-based measure in this work, it is by no means the only measure applicable. One could similarly make use of other measures that quantify distance between distributions, such as the Jensen-Shannon divergence, for example.

We have also explored other measures that could potentially be used to monitor representation changes. Perhaps the simplest approach is to use a direct distance measure (*e.g.*, $||\theta_1 - \theta_2||$) or monitor the angle between the two parameter vectors (El Khatib and Karray, 2019a). However, we find that a KL divergence measure is more predictive of performance than such direct measures. For one, a change in the value of θ does not necessarily translate into a change in $p_\theta(\mathbf{y}|\mathbf{x})$.[4]

[3] Note the change in notation here: we used $\hat{\mathbf{y}}$ earlier to denote the probability distribution. In this case, $\mathbf{y}$ denotes the class variable.

[4] A trivial case of this is a single layer network whose parameter vector is perpendicular to all input samples $\mathbf{x}$. One can scale the parameter vector without affecting $\theta^\top \mathbf{x}$.

4.3 Experiments

In this section, we evaluate the auxiliary tasks and pre-training approaches described in Section 4.1, in their supervised and unsupervised variants. We begin by describing the evaluation metrics that will be used and the experimental setup.

4.3.1 Evaluation Metrics

We report results using three metrics: average accuracy, forgetting, and intransigence—the same metrics used in Chapter 3. Refer to Section 3.4 for the definitions of these metrics. As a reminder, though, note that the overall performance (*i.e.*, the average accuracy) of a continual learning model depends on the sum of forgetting and intransigence. Inspecting forgetting and intransigence separately reveals more insights about the learning dynamics and the strengths and weaknesses of each method.

In addition to these metrics, we will also report the performance of the models on individual tasks, where needed.

Finally, we will use the KL divergence-based measure described in Subsection 4.2.1 to monitor internal representation shifts.

4.3.2 Datasets, Methods, and Testing Framework

Once again, our focus in this work is on image classification tasks. To define the sequences of tasks, we make use of these five data sets: CIFAR10, CIFAR100, MNIST, KMNIST, and SVHN (Netzer et al., 2011).

The testing framework uses sequences of tasks similar to those used in Section 3.4 of the previous chapter. Each task is sampled from the classes of the five data sets. The model is optimized for each task for a certain number of iterations, after which it loses access to the corresponding training data. In addition to the c10-2, c10-k-2, and k-c10-2 task sequences defined in Section 3.4, we define the following task sequences:

- c10-k: Learning CIFAR10 followed by KMNIST. 2 tasks, 10 classes/task.
- sv-m: Learning SVHN followed by learning MNIST. 2 tasks, 10 classes/task.
- c100-20: Learning the classes of CIFAR100, 20 at a time. 5 tasks, 20 classes/task.

- c100-5: Learning the classes of CIFAR100, 5 at a time. 20 tasks, 5 classes/task.

Again, the data sets and sequences are chosen to test performance in diverse conditions: long vs. short task sequences (*e.g.*, c100-5 vs. c10-2), small vs. large tasks (c10-2 vs. c100-20), and similar vs. dissimilar tasks (*e.g.*, c10-2 vs. c10-k).

In addition to the models used in Section 3.4 (vanilla, L2, EWC, and RWalk), we also compare against the Learning without Forgetting (LwF) method by Li and Hoiem (2018). We will use the following notation to denote the approaches described in Section 4.1.

- Aux-us: Unsupervised auxiliary tasks.
- Aux-s: Supervised auxiliary tasks.
- Pre-us: Unsupervised pre-training.
- Pre-s-v1: Supervised pre-training, first version (see below for more details).
- Pre-s-v2: Supervised pre-training, second version.

4.3.3 Implementation Details

We use the same core network for all methods, consisting of 3 convolutional layers with 128 units each, followed by 2 fully connected layers with 1024 and 256 units, respectively. All layers have leaky ReLU activation, with a 0.1 coefficient. All convolutional layers are followed by batch normalization, and the first 2 convolutional layers are also followed by 2×2 max-pooling.

All input images are scaled to have the same $3 \times 32 \times 32$ dimension (gray scale images are expanded across the channels dimension).

We use 100 training samples per class for all tasks, and optimize the model for 400 batches per task, with a batch size of 100 samples. We use 20% of the full training data set for validation and hyper-parameter tuning, and report results on the test set of each data set. We use an Adam optimizer for all experiments, with a learning rate of 10^{-4}. We repeat each experiment 10 times and report average values for all metrics.

Note that in most cases, hyper-parameter tuning was performed on the C10-2 task and the parameter values were carried over to the other tasks. We believe this to be a reasonable process in that a method should not need extensive tuning for every different task sequence. However, we note that some methods may be more susceptible to this process than others.

We use a regularization coefficient of $1e3$ for L2, $1e7$ for EWC, 1.0 for RWalk and LwF, $1e3$ for Aux-us, and $1e-3$ for Aux-s. Moreover, the regularization coefficient in the case of Aux-s is reduced by a factor of 6 after each training task.

For Aux-s, we use KMeans to find clusters in the training images (refer to Algorithm 1), with 2, 3, 4, 5, 6, 7, 8, and 10 clusters (that gives a total of 8 auxiliary tasks).

The two versions of supervised pre-training differ in how the auxiliary tasks are presented to the model. For pre-s-v1, the 8 tasks are presented simultaneously, and the model is pre-trained for 400 epochs. For pre-s-v2, we use a random curriculum, where a single auxiliary task is randomly selected and learned for 2 epochs. This process is repeated 200 times (for a total of 400 epochs, as well).

4.3.4 Results

Table 4.1 and Table 4.2 show forgetting and intransigence values, respectively, for all models and experiments. We make the following observations:

- Unsupervised auxiliary tasks (Aux-us) generally lead to significant reduction in forgetting, with performance comparable to explicit forgetting penalties such as EWC and RWalk.
- The use of supervised auxiliary tasks (Aux-s) shows mixed results. In some cases (c10-2), it does improve performance, albeit not as much as unsupervised tasks. However, for most experiments, Aux-s did not show significant improvement.
- Unsupervised pre-training (Pre-us) shows a small reduction in forgetting in most cases. However the improvement is not significant, and there are cases where it leads to poorer performance (sv-m).
- Supervised pre-training (Pre-s-v1 and Pre-s-v2) shows mixed effects. In some cases (c10-2, c10-k), it shows a reduction in forgetting. In those cases, pre-training with a random curriculum (Pre-s-v2) induces a larger reduction in forgetting compared to parallel pre-training (Pre-s-v1) (11.4% vs. 14.9% for c10-2; 24.3% vs. 28.7% for c10-k). In other cases, however, supervised pre-training did not show significant effects, and in one case (sv-m) led to increased forgetting.

- Unsupervised auxiliary tasks (Aux-us) is relatively more effective with tasks with a small number of classes (c10-2, c10-k-2, k-c10-2, c100-5). With larger tasks (c100-20, sv-m, c10-k), Aux-us tends to induce higher intransigence. (See Subection 4.3.4 for more on the forgetting-intransigence trade-off.)

- All models perform relatively worse when the tasks involved are dissimilar. Dissimilar tasks include, for example, learning MNIST after SVHN (sv-m). While both are tasks to recognize digits from "0" to "9", their input distributions are significantly different (*e.g.*, images in the former are gray scale and handwritten, unlike those in the latter task).

- With regard to the baseline models, we note that EWC does address the weakness of the isotropic L2 penalty. This is reflected in the lower intransigence values for EWC, reflecting the fact that, while it penalizes parameter changes (like L2), it does so in a more intelligent manner, allowing less critical parameters more leeway to learn additional tasks. Nonetheless, we still see that with a long or large task sequence (c100-5, c100-20), EWC still suffers from high intransigence. We believe this is a consequence of the purely *post hoc* regularization strategy, which leads to inefficient use of model capacity (not as inefficient as L2, but still with limitations).

- We note that RWalk suffers from a higher degree of forgetting than EWC, especially with long task sequences (c100-5) and large tasks (c100-20). We believe this is a consequence of its use of a running Fisher information matrix estimate (Chaudhry et al., 2018), which, as the model sees more training batches, de-emphasizes contributions from older training batches. This again highlights the important trade-off between forgetting and intransigence that continual learning models must balance (Subsection 4.3.4).

- Finally, we note that LwF is a strong contender in cases of high inter-task similarity (c100-5, c10-2, c100-20). However, because it relies on the current training data as a proxy for older training data (Li and Hoiem, 2018), it shows poor performance when the the task sequence contains highly dissimilar tasks (c10-k, c10-k-2, k-c10-2, sv-m). (We will discuss LwF and the effect of inter-task similarity in more detail in Section 5.1.)

In the remainder of this section, we discuss how different methods are impacted by representation changes during training, as measured by the KL divergence-based metric proposed in Subsection 4.2.1. Moreover, we highlight noteworthy observations about individual task accuracy over the training sequence, and draw insights about the different

Table 4.1: Forgetting for the tested methods on diverse task sequences.

	c10-2	c10-k	c10-k-2	k-c10-2	sv-m	c100-20	c100-5
Vanilla	15.3	30.3	17.5	16.6	36.5	33.5	44.1
L2	0.7	13.1	8.3	0.4	24.1	0.3	2.1
EWC	2.8	18.6	8.0	3.3	32.0	0.2	2.2
RWalk	2.7	18.2	9.2	3.6	29.2	11.0	22.2
LwF	0.4	16.1	12.1	9.3	22.5	6.5	4.1
Aux-us	2.6	10.6	7.4	5.5	4.0	4.3	14.7
Aux-s	8.8	27.1	18.9	15.0	36.1	34.3	42.8
Pre-us	14.3	26.9	16.4	16.4	42.7	31.7	41.2
Pre-s-v1	14.9	28.7	19.7	15.6	43.1	30.8	44.7
Pre-s-v2	11.4	24.3	18.2	16.9	40.6	28.4	39.7

Table 4.2: Intransigence for the studied methods on diverse task sequences.

	c10-2	c10-k	c10-k-2	k-c10-2	sv-m	c100-20	c100-5
Vanilla	0.7	-3.1	-0.6	-0.1	-3.2	-5.7	-4.5
L2	14.3	20.4	19.2	22.1	5.6	12.4	25.7
EWC	2.2	-1.3	0.4	0.9	-2.6	11.9	13.2
RWalk	2.8	1.1	0.4	0.7	-2.0	1.6	-0.1
LwF	1.2	-2.7	-0.7	-0.2	-2.8	-5.7	-4.4
Aux-us	3.7	18.5	1.8	2.3	25.1	19.4	5.3
Aux-s	1.2	-2.8	-0.7	-0.6	-2.7	-5.9	-4.6
Pre-us	0.8	-2.1	-0.1	-0.2	-3.1	-5.9	-4.6
Pre-s-v1	0.7	-1.9	-0.1	-0.6	-0.3	-3.1	-3.7
Pre-s-v2	1.2	-0.3	0.1	0.1	1.0	-2.5	-3.0

methods from them. We also discuss further the trade-off between forgetting and intransigence, and how it affects the different models.

Representation Changes

We attempt here to understand the internal dynamics of the proposed approaches, in particular Aux-us, in terms of the KL divergence measure in (4.5), and relative to the other methods tested.

Fig. 4.3 shows the KL divergence values for the c10-2 task. The first observation we make is that without any intervention ("Vanilla"), the model's distribution exhibits a relatively large KL divergence with every new training task, and a large accumulated change across the full task sequence ("$1-5$"). We can also see that the L2 model almost entirely prevents any change to the model's distribution across tasks. This is due to the explicit and strong penalty it imposes on changes to any model parameter. As a consequence, the L2 model exhibits high intransigence, as we saw in Table 4.2. Comparing L2 to EWC, we can see the advantage of the latter's more sophisticated approach to penalizing parameter changes. EWC allows parameters to change in an inverse-proportionate manner to their importance, as we saw in Chapter 3. Thus, it does allow for larger changes in the model's distribution than L2 (but still smaller than the Vanilla model), which again is reflected in its lower intransigence values. We also note that EWC allows larger changes early on in the training sequence (we talk more about this below). We note as well that RWalk and LwF result in KL divergence reductions from task to task. In the case of LwF, however, the accumulated changes across the entire task sequence can be larger than the Vanilla model in some cases (*e.g.*, Fig. 4.4). This is possibly explained by the way LwF prevents forgetting, which does not involve an explicit penalty on changes to parameters.[5]

Considering the behaviour of the proposed approaches, Aux-us and Aux-s, we can see that both reduce changes in the model's distribution; however, Aux-us does so to a significant extent. Aside from the L2 model which suffers from significant intransigence, Aux-us has the lowest accumulated change across the task sequence ("$1-5$"). This indicates 1) that Aux-us learns representations that are better suited to the continual learning setting, and 2) that those representations are durable (*i.e.*, less susceptible to significant re-optimization). Note that unlike the other methods, Aux-us does not impose any explicit penalty on forgetting and that the training data it uses are only the training data for

[5]This observation merits further investigation. We conjecture that our use of the simplified KL divergence measure described in Subsection 4.2.1 may be obscuring some of the details of the behaviour of LwF. One may explore this issue further in future work, using a per-task KL divergence measure.

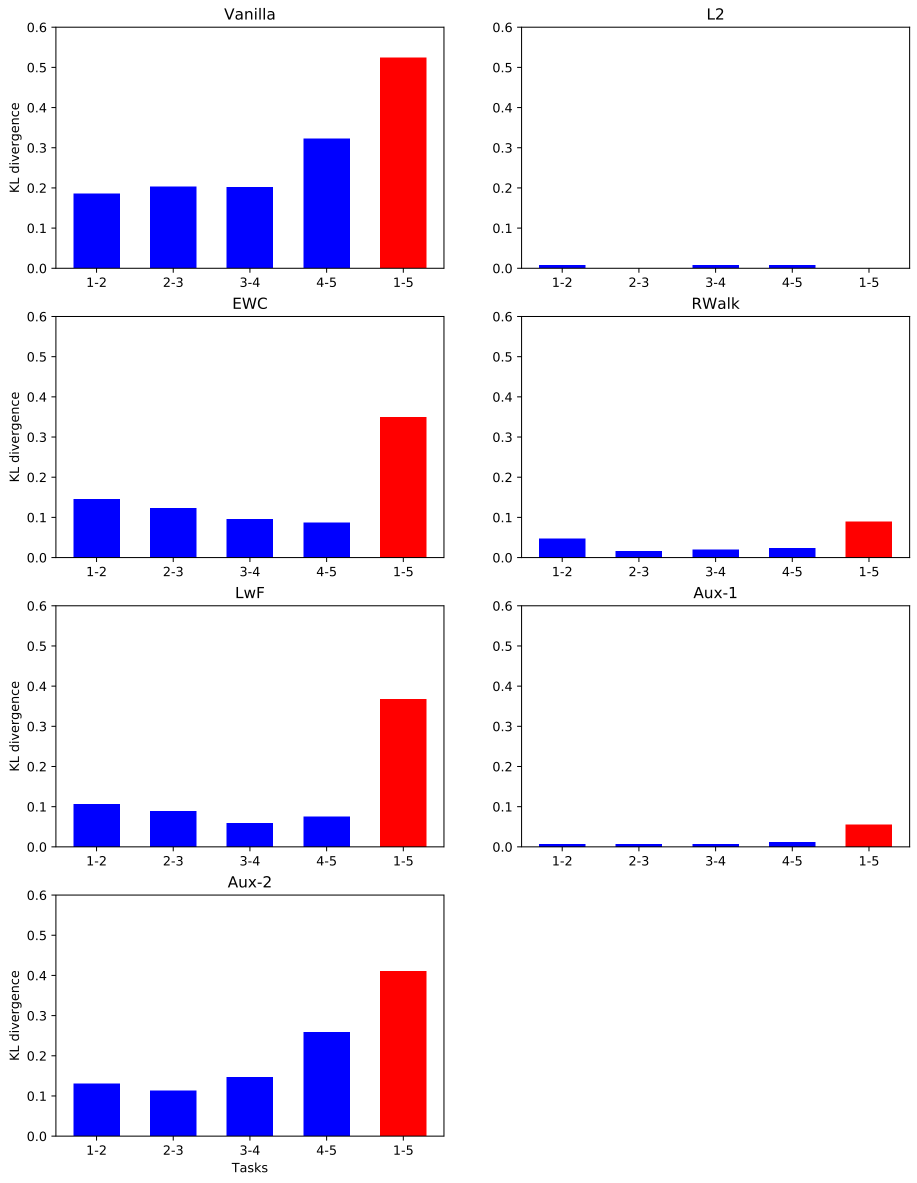

Figure 4.3: KL divergence values for the c10-2 task sequence. The last bar (in red) in each plot shows the overall accumulated change after the first task. Note: Aux-1 refers to Aux-us and Aux-2 refers to Aux-s.

each new task. In other words, Aux-us is not explicitly penalizing changes to the model's parameters or distribution. Instead, the model is able to learn each new task without significant changes to the distribution. We emphasize this point because it is critical to the distinction between *post hoc* regularization methods (*e.g.*, EWC) and what we are proposing. *Post hoc* methods penalize changes after the fact; whereas our proposal is to make changes unnecessary in the first place by learning reusable representations.

Fig. 4.4 shows the KL divergence values for the c10-k-2 task. We see again the same consistent behaviour of the methods. Of note here is that the c10-k-2 alternates between 2 sets of highly dissimilar tasks (CIFAR10 classes and KMNIST classes). Hence, we see that for most methods, the 1-step KL divergence is larger than the 2-step KL divergence shown in Fig. 4.5 (*e.g.*, "$1-2$" tends to be larger than "$1-3$"). In other words, there is a roughly cyclical nature to the way the model's distribution changes, where there is a large change learning a KMNIST task after a CIFAR10 task, which is partially reversed after learning another CIFAR10 task. For all but the LwF method, this results in an accumulated change ("$1-10$") that is smaller than the sum of all 1-step changes.

The last task sequence we consider here is c100-5, which demonstrates the behaviour for longer task sequences. The KL divergence values are shown in Fig. 4.6. We see that EWC, RWalk, and Aux-us result in lower KL divergence values compared to the Vanilla model. Fig. 4.7 shows a zoomed-in version of the same plots, and demonstrates noteworthy differences in the 3 methods. First, we note that EWC allows larger changes in the model's distribution toward the beginning of the task sequence. This means that initially a large potion of the model's parameters are "unimportant" to previous performance and thus are allowed to change significantly, which in turn allows the model's distribution to change. However, as the model learns more tasks, EWC starts to behave much like L2, in that most of its parameters become "important", and thus the model's distribution does not change significantly after. This again is the weakness of post hoc regularization: the model's capacity is used inefficiently and is used up quickly. By comparison, we can see that RWalk continues to allow larger changes in the model's distribution, due to its use of a running Fisher information matrix estimate (Chaudhry et al., 2018), and this is reflected in its higher forgetting and lower intransigence for this sequence (Table 4.1 and Table 4.2). As for Aux-us, we see that again it reduces changes in model distribution at each step. However, because the task sequence is long and Aux-us does not penalize change explicitly, we see that the overall accumulated change ("$1-20$") is relatively large, which is reflected in the forgetting value in Table 4.1.

As we end this treatment of KL divergence, we emphasize that the metric is not meant as a general predictor of performance. In fact, as we saw in the results, some successful methods allow significant change to the model's distribution without suffering significant

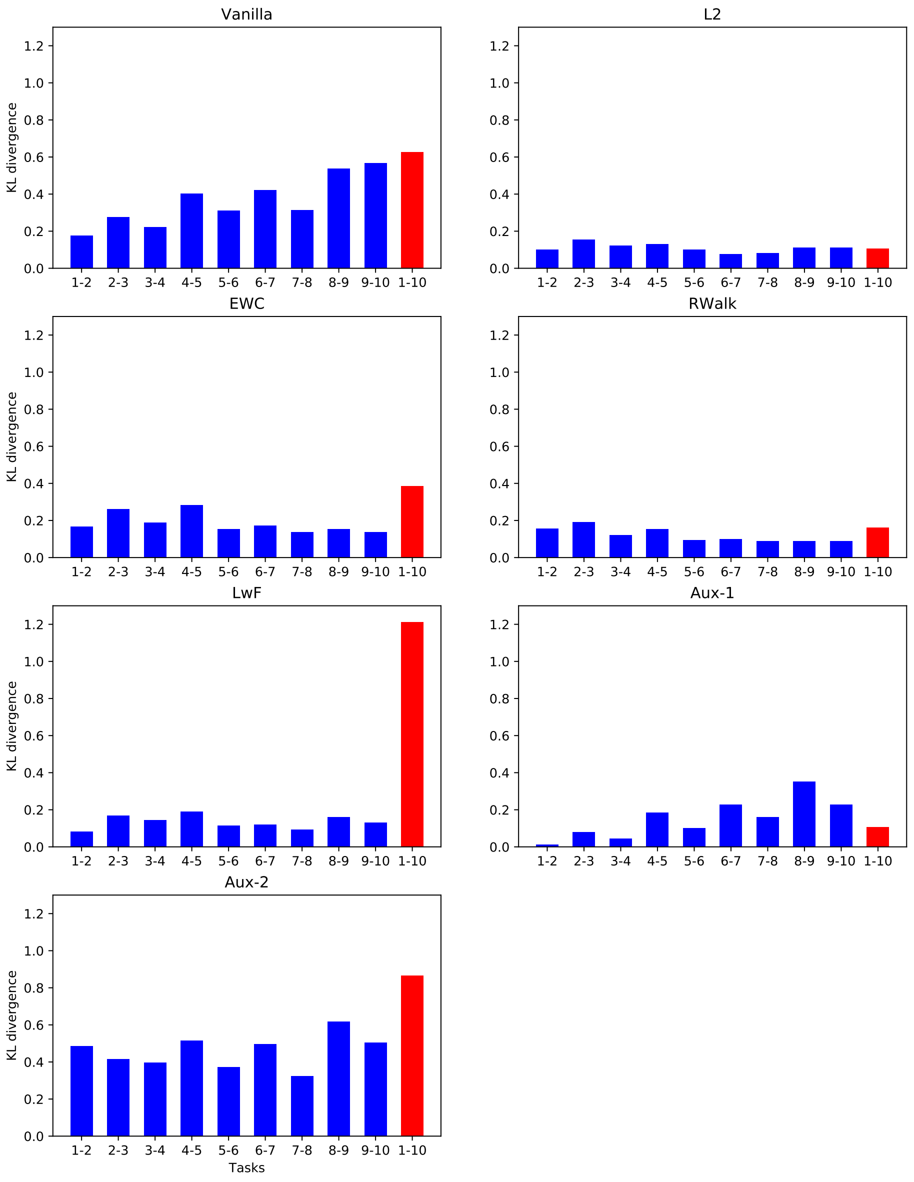

Figure 4.4: KL divergence values for the c10-k-2 task sequence. The last bar (in red) in each plot shows the overall accumulated change after the first task. Note: Aux-1 refers to Aux-us and Aux-2 refers to Aux-s.

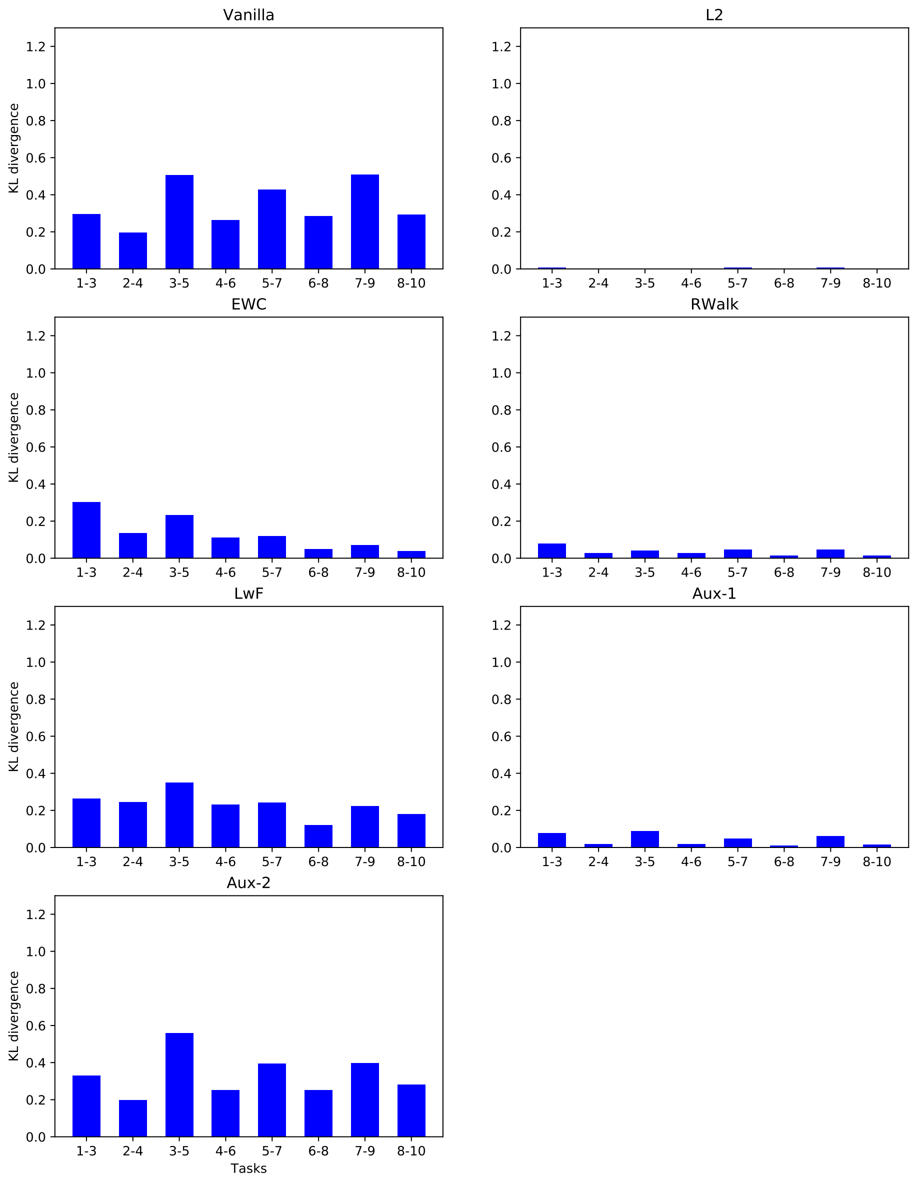

Figure 4.5: Two-step KL divergence values for the c10-k-2 task sequence. The values represent the change in distribution after learning 2 tasks in the sequence. Because the task sequence alternates between highly dissimilar sets of tasks (from CIFAR10 and KMNIST), we see a roughly cyclical change in the model's distribution for many methods.

forgetting. However, for methods that do rely on the durability of the learned representations, such as Aux-us and Aux-s, the KL divergence metric is a good indicator of performance. And for the remaining methods, we use the metric to gain insight into their internal workings.

Single task trends

The preceding discussion is perhaps a good segue to Fig. 4.8, which shows the performance of EWC, RWalk, and Aux-us on individual tasks in the c100-5 task sequence. These plots reaffirm the points we have made so far. 1) We see that EWC is able to maintain accuracy on earlier tasks in the sequence (Tasks 1–2) better than RWalk and Aux-us, however at the expense of not being able to adequately learn later tasks (Tasks 18–20). EWC's lower performance on later tasks is what drives its average intransigence higher than RWalk's. On the flip side, RWalk does a better job at learning later tasks, because of its less restrictive regularization approach that de-emphasizes earlier tasks. As for Aux-us, we first note that the learning curve for each task tend to be slower.[6] Moreover, aside from the higher intransigence early on, Aux-us follows a similar trend to RWalk, performing on later tasks better than EWC, but seeing higher forgetting for early tasks.

Forgetting-Intransigence Trade-off

We have noted earlier that there is almost always a trade-off with continual learning models between forgetting and intransigence. Methods that restrict the model too much (*e.g.*, L2) have low forgetting at the expense of high intransigence. Not restricting the model at all (*e.g.*, the Vanilla model) causes high forgetting with low intransigence.

Each method tends to have its own forgetting-intransigence curve. Fig. 4.9 shows this curve for L2, EWC, and Aux-us. The ideal continual learning method should have low forgetting and low intransigence, and hence its curve should be closer to the origin. We can see that L2, for example, is a weak approach, as its forgetting-intransigence curve always lies in the high intransigence region. Our proposed approach is more similar to the EWC curve. Both can exhibit low forgetting at a small cost in intransigence.

Of course, the forgetting-intransigence curve is task-specific. And we have seen in Table 4.2 that for certain tasks, both EWC and Aux-us will exhibit high intransigence.

[6] Allowing Aux-us to train for more epochs per task reduces its intransigence values. However, for a fair comparison, and due to limited computational resources, we limit all experiments to 400 iterations per task.

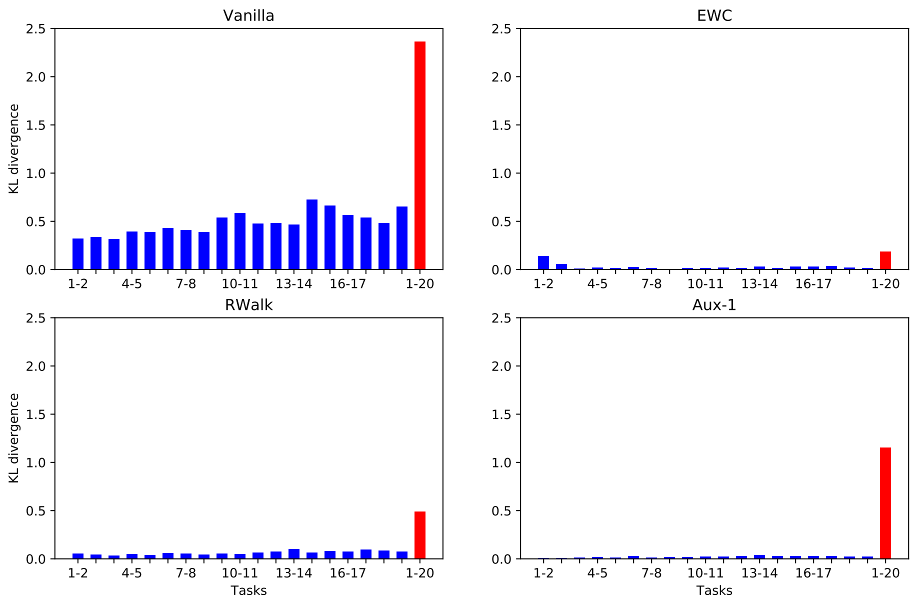

Figure 4.6: KL divergence values for the c100-5 task sequence. The last bar (in red) in each plot shows the overall accumulated change after the first task. Note: Aux-1 refers to Aux-us.

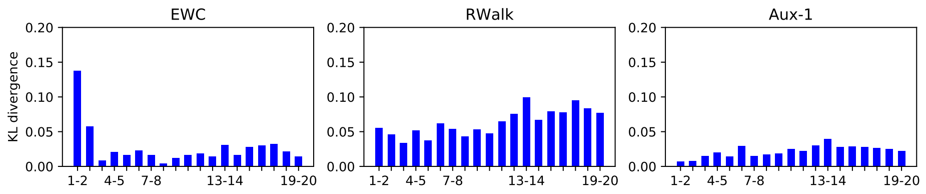

Figure 4.7: KL divergence values for the c100-5 task sequence, zoomed in. Note: Aux-1 refers to Aux-us.

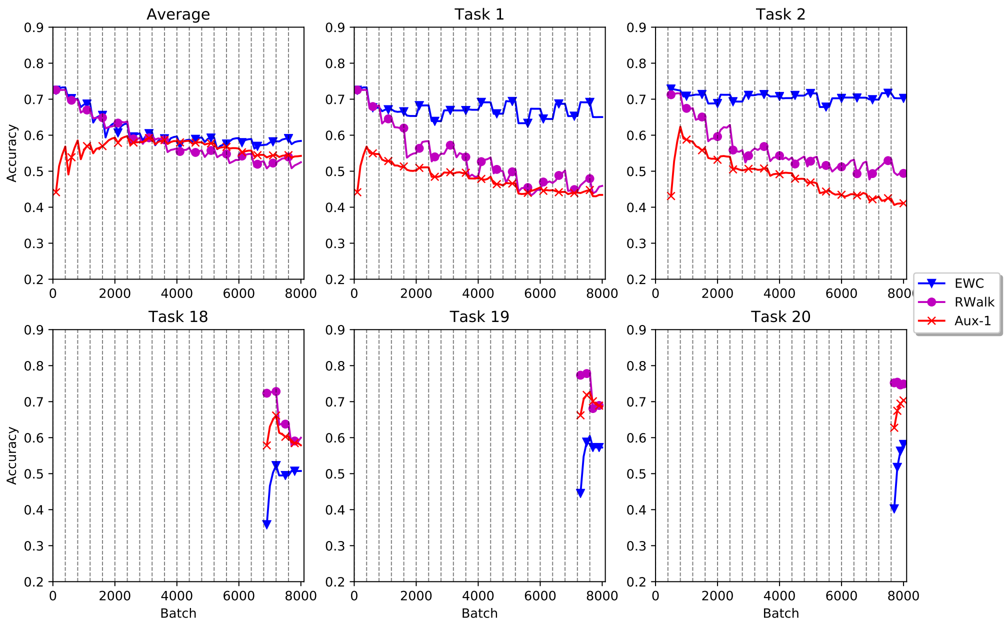

Figure 4.8: Single task behaviour for the c100-5 task sequence.

The goal of a successful regularization-based continual learning method is to attain a favourable forgetting-intransigence curve for all possible task sequences.

4.4 Summary

We presented in this chapter an alternative approach to *post hoc* regularization methods that relies on learning durable representations that are more suited to continual learning. We explored multiple methods, including supervised and unsupervised auxiliary tasks, and supervised and unsupervised pre-training. Our experiments suggest that unsupervised auxiliary tasks (Aux-us) is the most promising direction to pursue.

Using the KL divergence-based metric we proposed, we were able to show that Aux-us was able to reduce changes to the model's distribution across tasks, and consequently reduce forgetting, without explicitly penalizing forgetting. Moreover, we presented an extensive evaluation of the proposed method and multiple strong contenders from the literature, and used the proposed metric to gain insight into the internal dynamics of each method. There remain interesting questions about the behaviour of some of the models (*e.g.*, the ability of LwF to allow significant changes to the model's distribution without affecting performance).

As far as computational and memory consideration is concerned, we note that the complexity of Aux-us is not a function of the number of tasks learned. The method only requires additional fixed memory to store the decoder layers, and additional computations to forward-pass and back-propagate through those layers.

As for possible extensions to this work, we believe there is potential for coupling durable representations with explicit penalties on forgetting. We saw that explicit *post hoc* penalties, like EWC, are inefficient with longer task sequences. We believe with better representation learning from the start, we could improve on such methods.

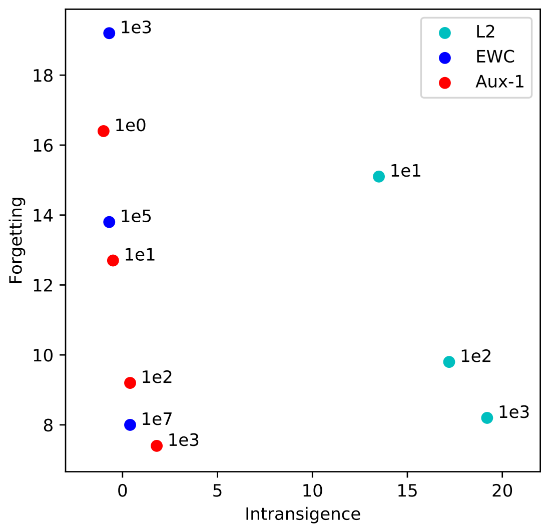

Figure 4.9: Forgetting-intransigence trade-off curves for the c10-k-2 task sequence.

Chapter 5

Other Challenges to Continual Learning

In the previous chapters, we reviewed and proposed various methods to counter forgetting, through regularization and other mechanisms. As we stated earlier, catastrophic forgetting is seen in the literature as a major obstacle to continual learning. In this chapter, we address other challenges facing continual learning research. In particular, in Section 5.1, we explore the effect of task properties, specifically inter-task similarity, on continual learning performance and propose a mechanism to increase the resiliency of continual learning models to changes in inter-task similarity. In Section 5.2, we examine the performance gap between single-head and multi-head continual learning models and propose a strategy to improve performance in the single-head setting.

5.1 On the Effect of Inter-Task Similarity

Our discussion of continual learning and catastrophic forgetting so far, however, can be described as task sequence-agnostic. In most of these efforts, the tasks to be learned are treated merely as replaceable evaluation tools, and do not factor in in the design decisions of the algorithms put forward. We argue in this chapter that some characteristics of the tasks to be learned, specifically inter-task similarity, bear a significant impact on forgetting, as well as on the success or failure of certain continual learning methods. We seek to explore this relationship and examine its effect on continual learning methods, focusing mainly on the learning without forgetting (LwF) model (Li and Hoiem, 2016; Li and Hoiem, 2018), which is particularly susceptible to it. But we also show that the effect of inter-task similarity on performance extends to continual learning models more generally, including the EWC model (Kirkpatrick et al., 2017) we saw in previous chapters. From the insights gained, we propose a rehearsal-based modification to LwF, EWC, and potentially other methods, that addresses their vulnerability to sequences of low inter-task similarity. We show that the proposed extension improves performance in various testing scenarios, at a modest memory and computational cost.

5.1.1 Related Work

Inter-task similarity and the relation between tasks has been studied before in the context of multi-task learning (Zhang and Yeung, 2013; Luo et al., 2017; Shui et al., 2019; Zhang et al., 2018). The goal there is often to exploit the similarities between tasks to the model's advantage. To our knowledge, the effect of inter-task similarity has not been explored before in the context of continual learning.

The use of replay was on of the earliest approaches explored in the literature to counter forgetting (Ratcliff, 1990; Robins, 1993; Aljundi et al., 2019a). However, this to our knowledge is the first attempt to use replay mechanisms to alleviate forgetting induced by low similarity in regularization-based models. Replay has also been used recently to narrow the gap between single-head and multi-head continual learning performance (Chaudhry et al., 2018; El Khatib and Karray, 2019b).

Shin et al. (2017) demonstrated recently that their deep generative replay model can be effectively used in conjunction with LwF. Our work differs from (Shin et al., 2017) in that we focus on the effect of inter-task similarity on performance and use the LwF model only to demonstrate the effectiveness of memory replay. Moreover, while Shin et al. (2017) use a generative adversarial (Goodfellow et al., 2014) model to generate replay samples, we show that a far less costly—computationally and in terms of memory—replay mechanism, namely a small replay memory, is enough to reduce forgetting significantly when the tasks learned are dissimilar.

5.1.2 Characteristics of Forgetting

In this section, we explore some of the characteristics of catastrophic forgetting, in particular, how inter-task similarity affects the extent of forgetting. As a start, we look at these characteristics in the context of a generic neural network. Later, however, we narrow down on a specific continual learning model that is particularly affected by these characteristics and that can benefit from taking inter-task similarity into account.

Effect of Inter-Task Similarity

Perhaps not surprisingly, when a neural network is trained on a sequence of tasks, one after another, the similarity of each task to the previously learned and yet-to-be-learned tasks has an impact on the overall performance of the model and the severity of forgetting it experiences. Learning a sequence of tasks that are relatively more similar induces lower forgetting than learning tasks that share no common features. An intuitive explanation of this phenomenon is that inter-task similarity affects the amount of re-optimization of model parameters with each new task—the more dissimilar the tasks are, the more significantly the parameters drift, and that translates into more severe forgetting.

We focus again on forgetting in the context of image classification tasks. In this context, inter-task similarity refers to the similarity between the images from different tasks.[1] Two tasks with significant overlap between their data distributions or content features, or where the average image-to-image distance (with images taken from different tasks) is small, have high inter-task similarity. For example, two binary image classification tasks where all four classes are drawn from, say, CIFAR10 (Krizhevsky, 2009) have a higher inter-task similarity, on the average, than two tasks drawn, respectively, from CIFAR10 and, say, MNIST (LeCun et al., 1998).

To demonstrate this point, consider the performance of a baseline neural network trained on the different sequences of tasks shown in Table 5.1 (values to the left of the arrows). We can see that the most severe cases of forgetting occur when learning sequences with low inter-task similarity (bottom two rows).

The effect of inter-task similarity on forgetting can be stated in terms of the so-called *backward transfer* (Lopez-Paz and Ranzato, 2017) phenomenon: the effect of learning a task on the performance of a previously learned task. Forgetting occurs in cases where, due to different factors, the sequence of training tasks results in *negative* backward transfer.

Interestingly in some cases, a training sequence can result in *positive* backward transfer. Fig. 5.1 shows such a case. We can see that every time the model transitions to learning a task from CIFAR10, there is an increase in the accuracy on the original CIFAR10 task (even

[1]We note that this does not account for the effect introduced by the labelling over the image samples in each task. However, this effect is tempered by the availability of task-specific parameters in the output layer (*i.e.*, the multi-head setting).

Table 5.1: Average forgetting values for a baseline model and LwF. (Refer to Fig. 5.2 for sample images from these data sets.)

Tasks	Forgetting
	Baseline → LwF
CIFAR10, 2 classes at a time	12.5% → 0.6%
STL-10 → CIFAR10	10.4% → 0.5%
CIFAR10 → KMNIST	29% → 18.3%
SVHN → MNIST	39.1% → 19.0%

though the model is never re-trained on that original task). This means that inter-task similarity can even induce a reversal of forgetting after the fact.

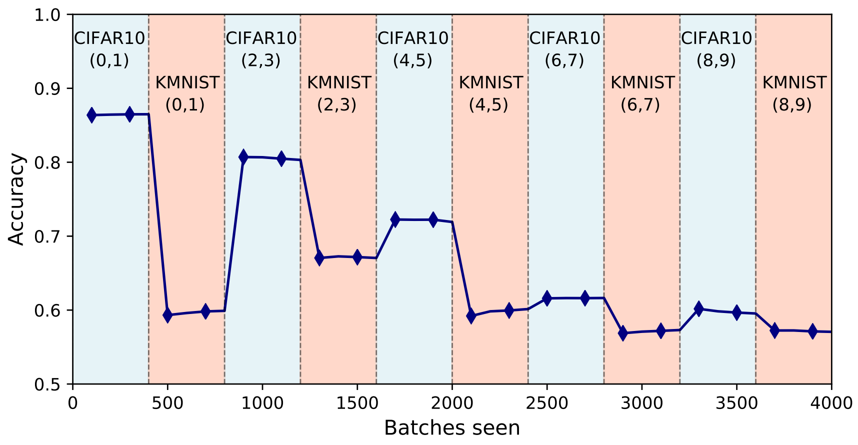

Figure 5.1: Accuracy on CIFAR10 classes (0,1) as the model is trained on the c10-k-2 task sequence described in Section 3.4 (2-class tasks sampled from CIFAR10 and KMNIST in alternating fashion). Shaded regions indicate the current training task (labelled above).

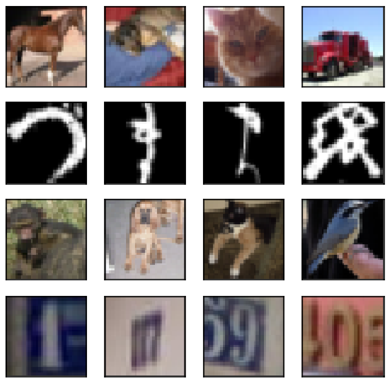

Figure 5.2: From top to bottom: samples from CIFAR10 (Krizhevsky, 2009), KMNIST (Clanuwat et al., 2018), STL-10 (Coates et al., 2011), and SVHN (Netzer et al., 2011)

Impact of Inter-Task Similarity on Continual Learning Approaches

While positive and negative backward (and forward) transfer in continual learning have been noted before (Lopez-Paz and Ranzato, 2017; Chaudhry et al., 2018), in almost all cases, they are not used to influence algorithmic design decisions or how forgetting is approached, and are merely used as evaluation tools. In this work, we seek to use these insights about the effect of inter-task similarity on backward transfer in improving on continual learning approaches.

One of the more susceptible continual learning models to changes in inter-task similarity is the Learning without Forgetting (LwF) model introduced by Li and Hoiem (2016). We delve into the details of this model and what makes it susceptible to inter-task similarity in the next section. Consider for now the results in Table 5.1. We can see again that dissimilar tasks (the bottom two rows) induce a higher degree of forgetting in the first place. We also see that LwF is relatively unsuccessful in countering forgetting in those same cases.

5.1.3 Accounting for Dissimilar Tasks

In this section, we describe the rehearsal-based mechanism we propose to counter the adverse effect of low inter-task similarity. We describe the proposed mechanism in the context of the learning without forgetting model (LwF), and thus we begin with an overview of LwF.

Learning Without Forgetting

We saw in Chapter 3 how fixed-capacity models counter forgetting through a regularization mechanism (Kirkpatrick et al., 2017; Chaudhry et al., 2018; El Khatib and Karray, 2019a). These methods work by imposing an explicit penalty on parameter drift, scaled by an estimate of the importance of each parameter. By contrast, the learning without forgetting (LwF) (Li and Hoiem, 2016; Li and Hoiem, 2018) model imposes a penalty on changes to the input-output mapping represented by the model. This means that there is only an implicit penalty on parameter drift. Moreover, in principle, model parameters are allowed to change without penalty, provided the overall input-output mapping does not change. In this respect, it is less restrictive than the methods we saw in Chapter 3 that impose explicit penalties on changes to model parameters.

The cost function used by LwF to prevent forgetting a previously learned task T_{k-1} while being optimized for task T_k is of the form

$$\mathcal{L}_{\text{cls}}^{k} + \lambda \mathcal{L}_{\text{lwf}}^{k-1} \tag{5.1}$$

where $\mathcal{L}_{\text{cls}}^{k}$ is the standard cross entropy loss for the current training task:

$$\mathcal{L}_{\text{cls}}^{k} = -\frac{1}{N_k} \sum_{i=0}^{N_k-1} \sum_{j=0}^{C_k-1} p_{i,j}^{k} \log(\hat{p}_{i,j}^{k}), \tag{5.2}$$

and $\mathcal{L}_{\text{lwf}}^{k-1}$ is the LwF penalty designed to penalize changes to the previously learned input-output mapping. C_k is the number of classes in task T_k, $p_{i,j}^{k}$ is the target probability that sample $\mathbf{x}_i^k$ belongs to class j, and $\hat{p}_{i,j}^{k}$ is the corresponding probability predicted by the model.

Li and Hoiem (2018) use a distillation loss (Hinton et al., 2015) for $\mathcal{L}_{\text{lwf}}^{k-1}$:

$$\mathcal{L}_{\text{lwf}}^{k-1} = -\frac{1}{N_k} \sum_{i=0}^{N_k-1} \sum_{j=0}^{C_k-1} \tilde{p}_{i,j}^{*k-1} \log(\hat{p}_{i,j}^{*k-1}), \tag{5.3}$$

where

$$\hat{p}_{i,j}^{*k-1} = \frac{(\hat{p}_{i,j}^{k-1})^{\frac{1}{r}}}{\sum_m (\hat{p}_{i,m}^{k-1})^{\frac{1}{r}}}, \tag{5.4}$$

and

$$\tilde{p}_{i,j}^{*k-1} = \frac{(\tilde{p}_{i,j}^{k-1})^{\frac{1}{r}}}{\sum_m (\tilde{p}_{i,m}^{k-1})^{\frac{1}{r}}}. \tag{5.5}$$

These two equations perform the transformations proposed by Hinton et al. (2015) on the model's output probability distribution .

$\tilde{p}_{i,j}^{k-1}$ represents the output on task T_{k-1} just before training on task T_k. $\hat{p}_{i,j}^{k-1}$ here denotes the output on task T_{k-1} while learning task T_k. Hence, the penalty drives the outputs for task T_{k-1} to remain close to their initial values prior to learning task T_k, which in turn preserves the input-output mapping on task T_{k-1}.

A missing detail from the description above is the input images that are used to produce responses $\tilde{p}_{i,j}^{k-1}$ and $\hat{p}_{i,j}^{k-1}$. Ideally, in order to counter forgetting of task T_{k-1}, one would want to maintain the input-output mapping for inputs sampled from the data distribution of T_{k-1}. This, however, would imply that the model still has access to this data set, which is not consistent with the continual learning framework (with the exception of rehearsal approaches). Instead—and this is where inter-task similarity becomes important—the authors use the training data set for task T_k as a proxy for the training data set of task T_{k-1}.

The use of the current training data set as a proxy for earlier data explains why LwF is relatively unsuccessful in cases of low inter-task similarity: the current data are not a representative sampling of the data over which LwF seeks to preserve the input-output mapping.

Extending LwF With A Memory Component

In order to remedy this weakness in the original LwF approach (as well as in other continual learning models), we propose extending LwF with a modest replay memory component. As we show in the next section, even a memory budget of 1% the size of the training data can be an effective remedy in cases of low inter-task similarity. Algorithm 2 summarizes the proposed approach.[2] Note that we show the steps for 2 tasks and without going over batching details, but the algorithm is extensible to the general case.

As can be seen, we use a basic replay approach, where, after learning each task, we store b samples from the associated training set in the replay memory. When learning subsequent tasks, the model is trained on memory samples periodically, according to a pre-defined schedule. In this implementation, the model rehearses memory samples after every r iterations, and for s rehearsal epochs.

We argue in this work that many otherwise successful continual learning methods perform poorly when the learning sequence includes highly dissimilar tasks. As we show in Subsection 5.1.4, the performance gains achieved by augmenting such methods with a small replay memory outweigh the memory and computational costs incurred as a result.

Applicability to Other Methods

We developed Algorithm 2 using LwF as a base model. The approach, however, is applicable to other regularization-based continual learning methods. As we show in Subsection 5.1.4, the performance of the EWC model, for example, on dissimilar tasks improves significantly when augmented with a replay memory.

[2]Note that we deviate from the original LwF approach slightly, in that we do not re-optimize θ^{k-1} and we do not use weight decay with any model.

Algorithm 2 LWF-REPLAY for continual learning of 2 tasks

```
Require: sequence of tasks $T_0, T_1$ and corresponding data sets $(X_0, Y_0), (X_1, Y_1)$
Require: continual learning model, core parameters $\theta^g$
Require: memory $M$, memory budget per task $b$, replay interval $r$, replay steps $s$
 1: for $k$ in $[0, 1]$ do
 2:     if $k > 0$ then
 3:         compute starting old task responses $\tilde{p}_{i,j}^{k-1}$ using $X_k$ and $\theta^{k-1}$
 4:     end if
 5:     initialize task-specific output head, $\theta^k$
 6:     for step in training steps do
 7:         compute current task responses $\hat{p}_{i,j}^{k}$ using $X_k$ and $\theta^k$
 8:         compute $\mathcal{L}_{\text{cls}}^k$ using (5.2)
 9:         if $k > 0$ then
10:             compute current old task responses $\hat{p}_{i,j}^{k-1}$ using $X_k$ and $\theta^{k-1}$
11:             compute penalty $\mathcal{L}_{\text{lwf}}^{k-1}$ using (5.3)
12:             $\mathcal{L} = \mathcal{L}_{\text{cls}}^k + \lambda \mathcal{L}_{\text{lwf}}^{k-1}$                    // Eq. (5.1)
13:         end if
14:         update model parameters $\theta^g$ and $\theta^k$ using gradient descent on $\mathcal{L}$
15:         if $k > 0$ and replay interval $r$ passed then
16:             for $h$ in $[0, \cdots, s]$ do
17:                 compute $\mathcal{L}_{\text{cls}}$ over memory samples
18:                 update core parameters $\theta^g$ using gradient descent     // rehearsal updates
19:             end for
20:         end if
21:     end for
22:     Store $b$ samples from $(X_k, Y_k)$ in memory $M$                   // update memory
23: end for
```

5.1.4 Experiments

As before, our experimental setup consists of a set of continual learning problems, in each of which a model is trained on a sequence of image classification tasks. The problems are designed to evaluate the effect of inter-task similarity on catastrophic forgetting and the sensitivity of different continual learning approaches to changes in this similarity. We make use of the c10-2, c10-k, and sv-m task sequences from Section 3.4 and Section 4.3. In addition, we define the following task sequence:

- k-m-f: KMNIST (Clanuwat et al., 2018) followed by MNIST (LeCun et al., 1998) followed by FashionMNIST (Xiao et al., 2017).

We use c10-2 and k-m-f to test performance on sequences with high inter-task similarity, and use c10-k and sv-m to test performance on sequences with low inter-task similarity.

Implementation Details

For all tasks and data sets, we use 100 training samples per class. This makes each task relatively more challenging, as well as reduces the training time required for executing the set of experiments. All images are resized to 32×32 pixels and normalized to the range $[0,\ 1]$. Gray scale images are replicated across the channels dimension, so that they have the same shape as the RGB images and can be processed by the same model. All models share the same base architecture. With the exception of task-specific output units, all the parameters of each model are shared across tasks (which gives rise to forgetting). We report the average of 10 runs for all experiments. We use $\lambda = 1.0$ for the LwF coefficient (Eq. (5.1)). For the 1% budget cases, we use a replay interval (r in Algorithm 2) of 1 (memory samples are rehearsed with every batch) and 1 replay step (s in Algorithm 2). For the 5% budget cases, we use a replay interval of 10 and 5 replay steps. We use a batch size of 100. We use a $1e7$ coefficient for the EWC model.

We use the same core architecture for all the tested models, with 3 convolution layers, each with 128 units of 3×3 kernel size. Each convolution layer is followed by batch normalization. The first 2 convolution layers are also followed by 2×2 max-pooling. The convolution layers are then followed by 2 fully-connected layers, with 1024 and 256 units, respectively. All layers use a leaky ReLU activation, with a negative slope of 0.1.

All models are trained for 400 batches of size 100 per task and evaluated in 100-batch intervals.

All of the data sets used have a pre-defined train/test split. We use 20% of the original training data (*i.e.*, not the reduced-size version used in training) for hyper-parameter tuning. The reported results are on the test set.

For all experiments, we use the Adam optimizer (Kingma and Ba, 2015), with a learning rate of $1e-4$, and 0.9 and 0.999 for β_1 and β_2, respectively.

Evaluation Metrics

As we did in the last two chapters, we will make use of forgetting and intransigence (defined in Subsection 3.4.3) to monitor performance.

Results

We evaluate the following models: a vanilla baseline model; replay with 1% memory budget per task (Replay-1); replay with 5% memory budget per task (Replay-5); LwF; the proposed extension to LwF (LwF-Replay-1 and LwF-Replay-5); EWC; and a memory-augmented version of EWC with 1% memory budget (EWC-Replay-1) and with 5% memory budget (EWC-Replay-5).

Table 5.2 and Table 5.3 show forgetting and intransigence values, respectively, for all models. We note the following observations.

First, both baseline EWC and LwF perform relatively poorly in the low inter-task similarity cases, with EWC resulting in 16.5% forgetting for c10-k and 29.5% for sv-m, and LwF resulting in 18.3% and 19.0% for the same tasks, respectively. By contrast, both EWC and LwF reduce forgetting to a very low degree when dealing with sequences of high inter-task similarity, with EWC reducing forgetting to 4.4% for k-m-f and 3.7% for c10-2, and LwF resulting in 2.3% for k-m-f and 0.6% for c10-2.

In the two experiments with low inter-task similarity (c10-k, sv-m), the addition of the replay memory to EWC and LwF significantly improves performance. EWC-Replay-1 reduces forgetting to 6.6% in c10-k (from 16.5% with EWC) and to 11.7% in sv-m (from 29.5% with EWC). We note that increasing the size of the replay memory from 1% to 5% does not improve the performance of EWC-Replay. LwF-Replay-5 reduces forgetting to 7.8% in c10-k (from 18.3% with LwF) and to 8.4% in sv-m (from 19.0% with LwF).

Overall, when dealing with dissimilar tasks, the memory-augmented models outperform the other variants, and outperform replay-only models with similar memory budgets, by a significant margin.

Table 5.2: Average forgetting percentage values.

Model	k-m-f	c10-k	c10-2	sv-m
Vanilla	21.9	29.1	12.5	39.1
Replay-1	14.6	17.8	10.7	23.7
Replay-5	6.7	12.9	5.2	17.5
EWC	4.4	16.5	3.7	29.5
EWC-Replay-1	**1.7**	**6.6**	2.2	11.7
EWC-Replay-5	1.9	7.6	1.5	12.8
LwF	2.3	18.3	**0.6**	19.0
LwF-Replay-1	2.7	11.9	2.1	15.0
LwF-Replay-5	2.5	7.8	0.7	**8.4**

Table 5.3: Average intransigence percentage values.

Model	k-m-f	c10-k	c10-2	sv-m
Vanilla	-1.5	-2.3	0.4	-2.5
Replay-1	-1.0	0.3	1.2	-1.9
Replay-5	-1.0	-0.6	0.7	-1.8
EWC	0.9	-0.6	1.9	-2.0
EWC-Replay-1	1.2	0.0	2.7	-1.7
EWC-Replay-5	6.1	0.7	2.9	-1.8
LwF	-1.3	-1.7	0.5	-2.1
LwF-Replay-1	-1.0	-0.7	1.9	-1.8
LwF-Replay-5	-0.8	-0.5	1.4	-2.0

Complexity Considerations

The results show that a memory budget as low as 1% of training data can be sufficient to improve performance. For example, for the c10-k task sequence, the memory budget per task is only 10 samples. For the $3 \times 32 \times 32$ images we are using, that's equivalent to $30,720$ parameters to store—a small fraction of the model's size. That is a small cost to pay given the improvement in performance. In our implementation, memory size scales linearly with the number of tasks. The added computational cost is also small (optimizing for the samples in memory), but also scales linearly with the number of tasks in our implementation.

5.2 Single-Head Continual Learning[3]

So far, we have seen various aspects of the continual learning problem. In Chapter 3, we discussed the role of parameter importance in countering catastrophic forgetting. In Chapter 4, we discussed representation learning and emphasized the role of durable, reusable representations in preempting forgetting. We explored the effect of inter-task similarity on the performance of certain continual learning methods in the previous section. In this section, we turn to another aspect of the continual learning problem, namely, performance in the single-head setting.

Most of the research effort in continual learning has focused primarily on ways to counter catastrophic forgetting and, in so doing, retain knowledge in a model's parameters over multiple optimizations with different objectives. Various approaches have been proposed over the years, including parameter regularization strategies, rehearsal and pseudo-rehearsal, and network growing.

We argue in this section that, in being mainly focused on countering forgetting, continual learning research has neglected what we see as an equally important issue facing continual learning—the performance gap between single-head and multi-head models.

Continual learning models are usually presented in one of two settings: multi-head or single-head, with the latter's performance usually being much worse. (Chaudhry et al., 2018). We look more closely at the source of performance degradation in single-head models and propose a strategy to remedy it. We show that the commonly used softmax cross entropy loss is not well aligned with the single-head framework. Instead we propose

[3]Parts of this section are adapted by permission from Springer Nature: Springer Lecture Notes in Computer Science, vol 11662 (El Khatib and Karray, 2019b), © Springer Nature Switzerland AG 2019.

using multiple binary cross entropy losses. We argue that, coupled with auxiliary unlabelled data, this leads to single-head models that are more robust to the addition of new classes during future learning.

5.2.1 Background

The literature on continual learning does not dedicate much attention to the single-head vs. multi-head question. In most cases, models are evaluated in either setting without explicit reasoning. In other cases, researchers argue that reporting results in the multi-head setting is justified by the fact that it is often significantly more feasible to predict the task from which an input sample is drawn than to classify that sample into one of the classes of that task. Some researchers, on the other hand, have argued, correctly in our view, that reporting results in the multi-head setting paints an overly optimistic picture for the performance of continual learning models, compared to single-head performance, and that it is not always straightforward to distinguish between tasks without external input (Chaudhry et al., 2018).

Single-Head vs. Multi-Head

As we now discuss continual learning in two settings—multi-head and single-head—it is fitting to refine some of the terminology we have been using in the previous chapters, to simplify the presentation of this chapter. We define the following concepts:

- A *learning experience* is the process of presenting a set of training data to a model and the corresponding optimization of the model's parameters. Each learning experience derives from a different, possibly mutually exclusive, data set. The subset of classes present in the training data of one experience is also different from, and possibly mutually exclusive with, the subset present in another.
- An *episode*[4] is a sequence of learning experiences (*i.e.*, what we have been referring to as a *task sequence*).
- A *task* encapsulates a set of classes among which a model should learn to discriminate. When evaluating a model on a task, no external information is given to the model that would allow it to narrow down the subset of classes from which a test image

[4]Note that our use of the term *episode* differs from its usage in the reinforcement learning literature.

comes. On the other hand, when evaluating a model's performance on a set of tasks, the model is told the task (and hence the subset of classes) from which each test image is drawn. In terms of network architecture, each task corresponds to a different output *head* (a subset of output units grouped together). The subset of classes corresponding to each task (or head) can be expanded over time. We provide this definition to remove any ambiguities when discussing single-head and multi-head settings.

To give a concrete example, consider a model that learns the 10 classes of MNIST (LeCun et al., 1998) in a continual learning framework, 2 classes at a time. In both single-head and multi-head settings, this corresponds to an *episode* of 5 *learning experiences*. In the single-head setting, all 5 learning experiences are over the same task (call it, for example, *mnist-0*), hence there is only one output *head*. The output units in this head however are expanded with each new learning experience: with the first experience, the *mnist-0* head contains 2 units, corresponding to classes 0 and 1; by the 5^{th} experience, it contains 10 units, corresponding to the 10 classes. In each learning experience, the model is presented with data from 2 classes only, but is evaluated on data drawn from all the classes in the *mnist-0* head. By contrast, in the multi-head setting, the model eventually contains 5 heads, each with 2 output units. With each presented image, the model is told which head to use (by the end, this increases the probability of correctly guessing the class of an image randomly from $\frac{1}{10}$ in the single-head setting to $\frac{1}{2}$ in the multi-head setting). Fig. 5.3 and Fig. 5.4 show illustrations of the classification process in the multi-head and single-head settings, respectively.

Sources of Performance Decay

We have seen so far two main sources of performance decay in continual learning models: forgetting and intransigence. We defined these concepts formally in Subsection 3.4.3. Briefly though, forgetting refers to the difference between the peak or final performance of a model on a task while optimizing for that task and the subsequent performance on it while optimizing for other tasks. Intransigence refers to the resistance to learning a task introduced by the continual learning setting. Both of these concepts are defined in the multi-head setting. Fig. 5.5 shows a typical continual learning trend on a task sequence, and Fig. 5.6 illustrates forgetting and intransigence.

In the single-head setting, we face a third source of performance decay, resulting from having the model make classification decisions over a larger set of classes (the union of the classes in all learning experiences). We refer to this source of decay as *confusion* and

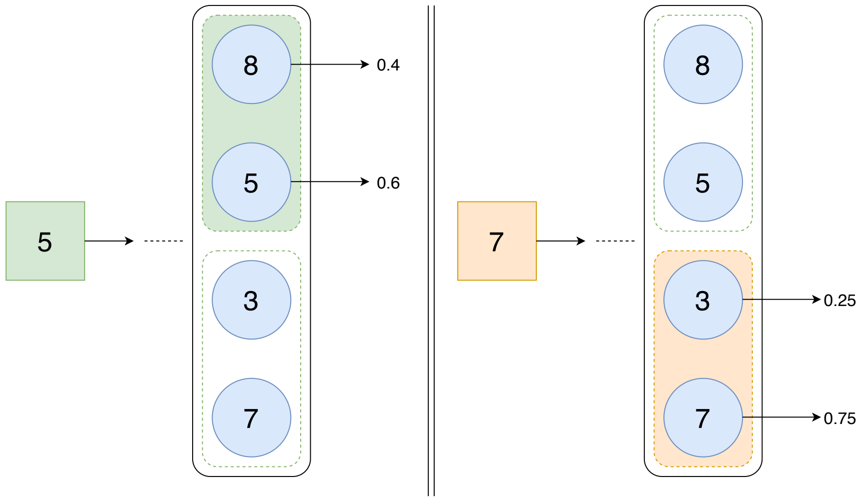

Figure 5.3: Illustration of making classification decisions in the multi-head setting. The model at this point uses two output heads (corresponding to two tasks). For each test input, the model selects one output head.

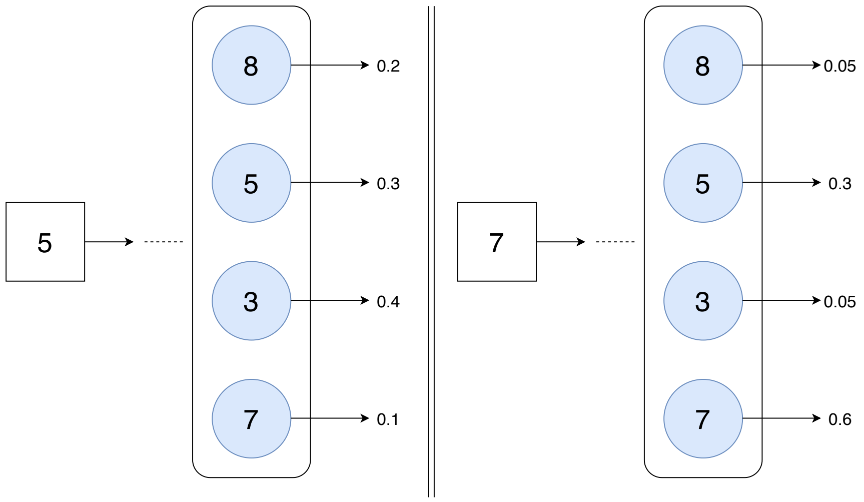

Figure 5.4: Illustration of making classification decisions in the single-head setting. All learned classes from all learning experiences are added to a single output head. For each test image, the model makes a classification decision (*i.e.*, outputs a probability distribution) over all classes in the output head.

we define it as the gap between multi-head accuracy and single-head accuracy. Fig. 5.6 illustrates confusion in a typical continual learning problem.

Understanding Confusion

Before discussing ways to alleviate confusion in the single-head setting, it is helpful to further discuss how the phenomenon arises.

Consider the 2-class classification problem shown in Fig. 5.7. The plot on the left shows the feature space for two classes, "red" and "blue"; the schematic on the right shows the two units in the output layer corresponding to the two classes. When the "red" and "blue" units are optimized to solve this classification problem, the "blue" unit learns to output high values for samples in the blue region of the feature space and low values for samples in the red region. The "red" unit, on the other hand, learns to output high values in the red region and low values in the blue region.

Now consider what happens after the same model learns an additional "green" class in the absence of training data for "red" and "blue" (shown in Fig. 5.8). Since the "red" and "blue" units where never optimized using "green" training samples, the units do not learn to output low values for "green" samples. In practice, some "green" samples will output high values for the "red" and/or "blue" units, and if those values are higher than the output of the "green" unit, those samples will be misclassified.

These misclassified samples are behind the confusion seen in the single-head setting.

5.2.2 Proposed Approach

Binary vs. Multi-Class Classification Loss

We focus in this work on image classification tasks, where the objective is to train a model to learn to predict class $y \in \mathcal{C}$ given an image $\mathbf{x}$. The cost function often used here is the cross entropy or the multi-class negative log-likelihood:

$$L(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(y_i|\mathbf{x}_i), \tag{5.6}$$

where N is the number of training samples and y_i is the true class of $\mathbf{x}_i$. Note that here the model probability distribution p_θ is a softmax distribution over all the classes present in the current learning experience:

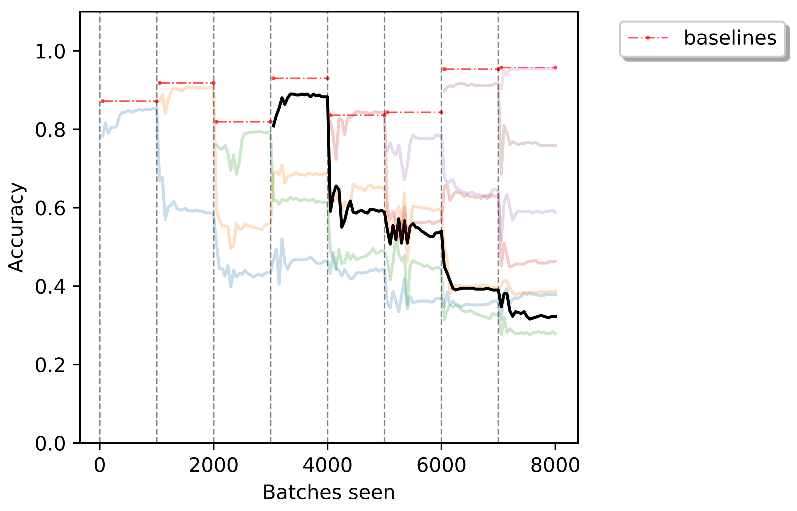

Figure 5.5: Typical multi-head performance on a sequence of tasks. Vertical dashed lines indicate training task transitions. Horizontal dashed lines show baseline performance on each task. Performance on each task is shown separately. The highlighted task is illustrated further in Fig. 5.6.

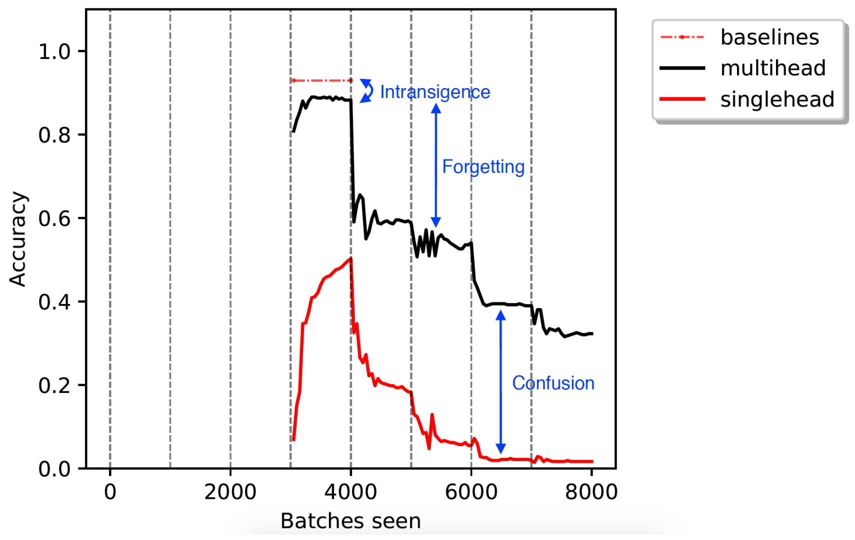

Figure 5.6: Illustration of forgetting, intransigence, and confusion in a typical continual learning problem.

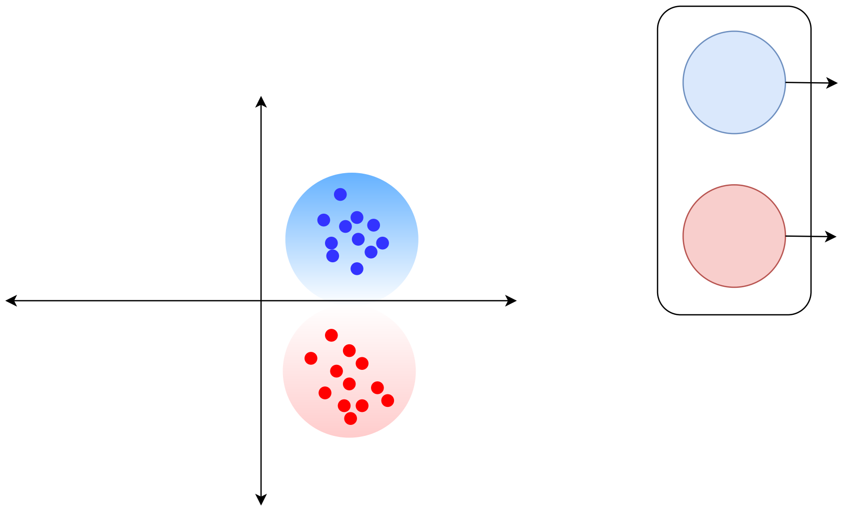

Figure 5.7: Understanding confusion in the single-head setting. Left: feature space for a two-class classification problem. Right: corresponding units in the output layer.

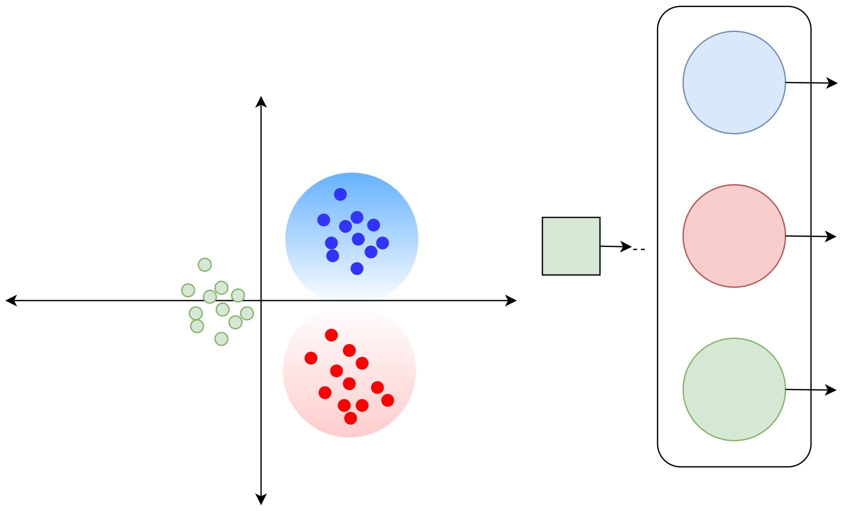

Figure 5.8: The introduction of a third “green” class in a subsequent learning experience.

$$p_\theta(y|\mathbf{x}) = \frac{e^{z_y}}{\sum_{j\in\mathcal{C}} e^{z_j}}. \tag{5.7}$$

Based on our discussion of confusion, we argue here that this cost function is not an appropriate choice for the loss in the single-head setting. This function drives the output of the unit corresponding to the correct class to be higher than the output of the other units in the learning experience. Ideally, the optimization should drive the model distribution toward the one-hot encoded ground truth. In practice, however, the resulting distribution has higher entropy, even for correctly classified inputs. For example, a high-accuracy model on a 4-class classification problem may correctly predict a probability distribution $[0.25, 0.25, 0.3, 0.2]$ for a sample from class 2 (out of classes $0-3$). While this may not affect performance in a multi-head setting, it does have the potential to degrade performance in the single-head setting.

Consider, for example, that in a subsequent learning experience this same model learns another 4 classes, $4-7$, using the same cost function. When evaluating this model with samples drawn from classes $0-7$, the predicted class of the model is the maximum output across all output units in the output head. This would be a reasonable prediction strategy had the model been trained to minimize a single negative log-likelihood for all 8 classes. However, with the model being trained with 2 learning experiences, one for classes $0-3$ and another for $4-7$, this is no longer the case. This is because units $0-3$ have been optimized to output values that make sense *relative to outputs from other units in the same learning experience.* The same is true for units $4-7$. Now, presented with a test sample, say from class 2, the output of the units from the first learning experience may be $[0.2, 0.2, 0.32, 0.28]$ and the output of the units from the second learning experience may be $[0.35, 0.25, 0.25, 0.15]$. The maximum taken across the units of the first learning experience corresponds to the correct class. Taken across all classes, however, it results in a misclassification.

This example illustrates the weakness of optimizing the output of units only relative to other units in the same learning experience. What is the alternative, though? Optimization relative to all units in the head is not possible in the continual learning framework (assuming one does not use a replay memory).

We conjecture that optimizing separate binary classifiers for each class in a learning experience leads to predictions better suited to an expanding single-head that can be extended with additional classes over time. Taken on its own, a binary classifier is optimized to output a high probability for a correct sample and a low probability for any other sample. Of course, the samples seen in the training set of a learning experience are still limited to a subset of classes from the total in the head. And so, one could argue that the binary classifier is still making relative predictions just as the softmax classifier. Our experiments, however, suggest this is not the case, especially when coupled with unlabelled auxiliary data, as we discuss in the next section.

To train a binary classifier for a unit, we binarize the labels of the samples in the corresponding learning experience. During optimization, we jointly minimize multiple binary cross entropy cost functions, one for each unit in the learning experience:

$$L(\theta) = \sum_{u=1}^{R} L_u, \tag{5.8}$$

where R is the number of units in the learning experience and

$$L_u = -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(y_i^{(u)}|\mathbf{x}_i). \tag{5.9}$$

p_θ here is a binary distribution over $y^{(u)}$, where $y^{(u)} = 1$ for a sample from class u and 0 otherwise.

Using Auxiliary Unlabelled Data

In order to arrive at more general binary classifiers for each class (*i.e.*, classifiers that will not significantly deteriorate when faced with samples drawn from outside the classes they were trained with), we propose augmenting the training sets for all learning experiences with additional data. Ideally, one would want to have training data for all classes that will eventually be added to an output head present for all learning experiences. But this is not feasible in the continual learning framework. We instead propose to use unlabelled data.

Unlabelled data are relatively cheap to obtain. For a robot navigating an environment, for example, unlabelled images can be randomly sampled from its surroundings. In web-connected applications, random data can be scraped from the internet.

We augment the training set for each learning experience with random unlabelled data. To train a binary classifier, these data samples are given negative labels. This assumption may not always hold. However, for many applications, such as image classification, the probability of a randomly sampled image to be positive for any class is so low that the resulting data contamination, if any, ends up being negligible.

5.2.3 Experimental Findings

In this section, we present our experimental results. We begin with a comparison of the multi-head and single-head continual learning settings. In both settings, we train a convolutional neural network (CNN) on CIFAR100 (Krizhevsky, 2009) incrementally. We present results for different episode configurations. In the baseline episode, there is a single learning experience with all 100 classes from CIFAR100. We also use episodes with 2 50-class experiences, 5 20-class experiences, 10 10-class experiences, 20 5-class experiences, and 50 2-class experiences. These episode configurations are designed to show how the multi-head and single-head settings affect performance in scenarios ranging from a large number of small learning experiences to a single large learning experience.

Fig. 5.9 and Fig. 5.10 show results in the multi-head and single-head settings, respectively. We note a number of observations from these two figures. First, generally, multi-head performance is significantly better than single-head performance. Second, using multi-head performance as a measure of forgetting can be misleading: as Fig. 5.9 shows, performance for the baseline episode (where there is no continual learning or forgetting) is worse than all other episodes. In fact, as the number of learning experiences increases, the final average accuracy on all tasks in the episode increases. Which is counter-intuitive, as one would expect the larger number of learning experiences to bring about more forgetting. The reason behind this, of course, is that a as the number of learning experiences in an episode increases, the difficulty of each individual task decreases, which raises the average performance overall. Table 5.4 illustrates this point more clearly. The table shows the final average accuracy in each case vs. the accuracy of a model that makes random guesses.

Another observation we note from these results is that the degradation in performance that can be ascribed to catastrophic forgetting is significantly less than that which can be ascribed to confusion in the single-head setting. This highlights the importance of learning to recognize classes in a way that is robust to the addition of classes over time. And this is what we try to address with the proposed strategies.

Table 5.4: Final average accuracy on CIFAR100 for different multi-head and single-head episode configurations. Reprinted by permission from Springer Nature: Springer Lecture Notes in Computer Science, vol 11662 (El Khatib and Karray, 2019b), © Springer Nature Switzerland AG 2019.

	Multi-head			**Single-head**		
Episode	Trained	Random	Δ	Trained	Random	Δ
Two 50-class	34.8	2.0	32.8	20.9	1.0	19.9
Five 20-class	43.0	5.0	38.0	18.1	1.0	17.1
Ten 10-class	45.8	10.0	35.8	12.9	1.0	11.9
Twenty 5-class	53.0	20.0	33.0	8.6	1.0	7.6
Fifty 2-class	61.4	50.0	11.4	3.5	1.0	3.0
One 100-class	-	-	-	32.5	1.0	31.5

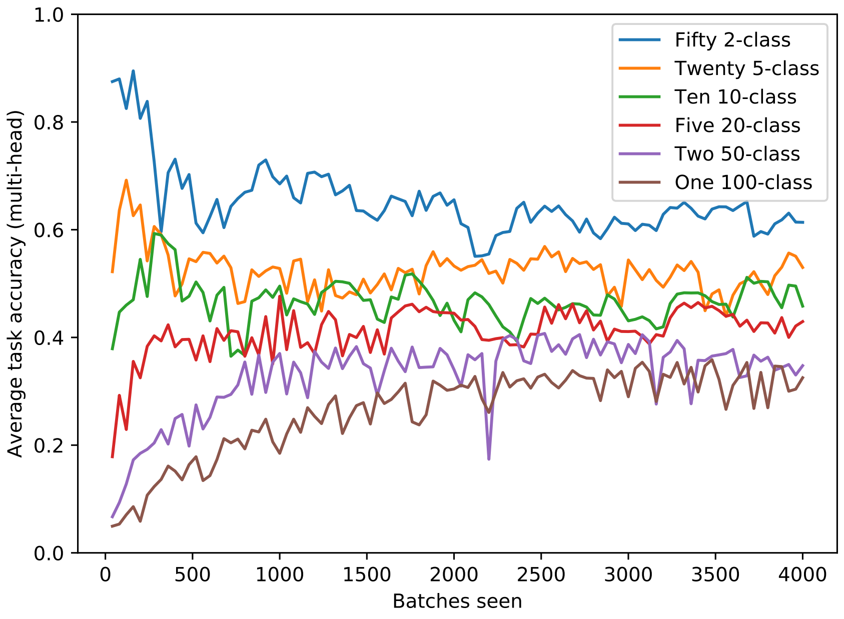

Figure 5.9: Average accuracy on CIFAR100 for different multi-head episode configurations. Reprinted by permission from Springer Nature: Springer Lecture Notes in Computer Science, vol 11662 (El Khatib and Karray, 2019b), © Springer Nature Switzerland AG 2019.

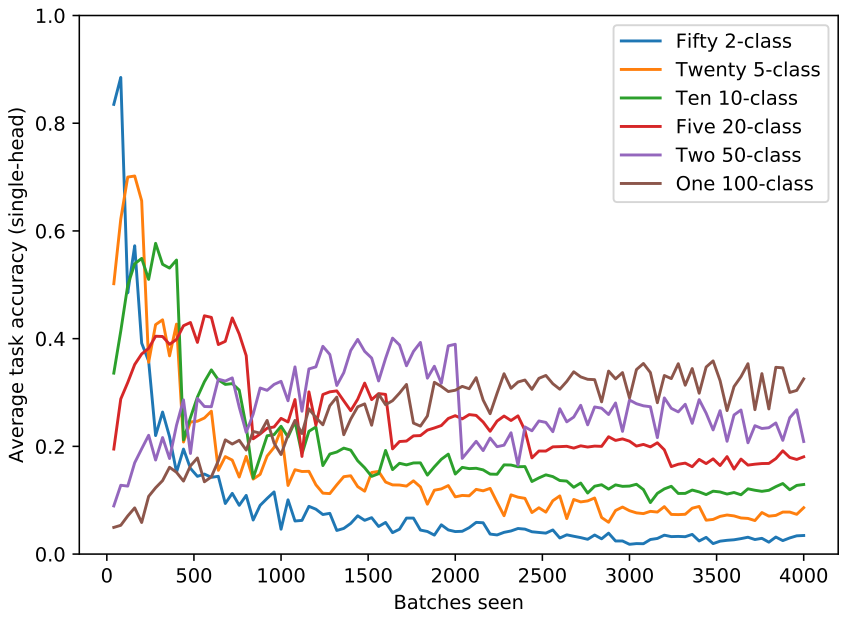

Figure 5.10: Average accuracy on CIFAR100 for different single-head episode configurations. Reprinted by permission from Springer Nature: Springer Lecture Notes in Computer Science, vol 11662 (El Khatib and Karray, 2019b), © Springer Nature Switzerland AG 2019.

Countering Confusion

We demonstrate now the effect of using our proposed approach on narrowing the gap between single-head and multi-head performance. We begin by describing the testing framework and implementation details.

Implementation Details

We use the following architecture for the core network: 3 convolution layers (128 units each) followed by 2 fully connected layers (1024 and 128 units, respectively). Convolution layers are followed by batch normalization, and the first 2 layers are also followed by 2×2 max-pooling. All layers use a leaky ReLU activation, with a 0.1 coefficient. We use the Adam optimizer with $1e-4$ learning rate for all experiments.

We use an episode with 8 learning experiences. Each learning experience corresponds to a 4-class classification problem, where the classes are randomly drawn from CIFAR100. By the end of the episode, the model learns a total of 32 classes. We use 100 training samples per class. For the proposed approach, we also make use of 5000 unlabelled samples from the STL-10 (Coates et al., 2011) data set. In addition, for the binary cross entropy loss, we use a weighting coefficient of 10 for the positive class (to counter the class imbalance).

The model is trained for 700 batches per learning experiences, with a batch size of 100. For the first 100 iterations of each learning experience, only the output units are updated, while the core network is frozen.

We repeat all experiments 10 times and report average values.

Results

Fig. 5.11 shows the result for the proposed approach vs. a vanilla baseline model in the multi-head and single-head settings. We can see that the proposed approach has a negligible effect on the multi-head performance (which is expected, since it is not designed to counter forgetting) and a significant positive effect in the single-head setting. By the end of the episode, the proposed approach results in an 8% improvement in the single-head setting. These gains are brought about by countering confusion (rather than forgetting). Table 5.5 shows a breakdown of performance in terms of forgetting and confusion. As we can see, the proposed approach reduces confusion by more than 10%.

While the proposed approach reduces confusion, it has no effect on forgetting. Hence, it stands to result in further improvement in the single-head performance if used in conjunction with methods that counter forgetting. Fig. 5.12 shows the result of combining the proposed approach with EWC (Kirkpatrick et al., 2017). (We use a $10e7$ regularization coefficient for EWC, and a $10e4$ coefficient for the combined model.) Again, we can see that while EWC improves forgetting (and thus the multi-head performance), the addition of the proposed approach does not affect multi-head performance significantly. On the other hand, EWC does not result in a significant improvement in the single-head setting. When the proposed approach is used with EWC, we reap the benefits of both approaches, reducing forgetting *and* confusion. As we can see in Table 5.5, the overall improvement in the single-head setting is more than 15% (25.6% vs. 10.5%). For reference, a baseline model with the same architecture and trained on all 32 classes simultaneously achieves a 48% accuracy.

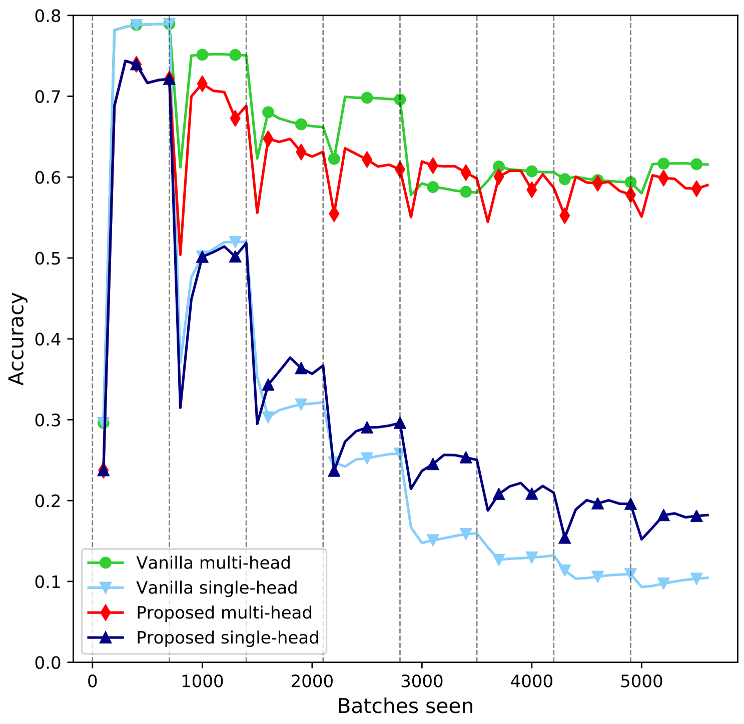

Figure 5.11: Average accuracy in the multi-head and single-head settings. Vertical dashed lines indicate transitions in learning experiences.

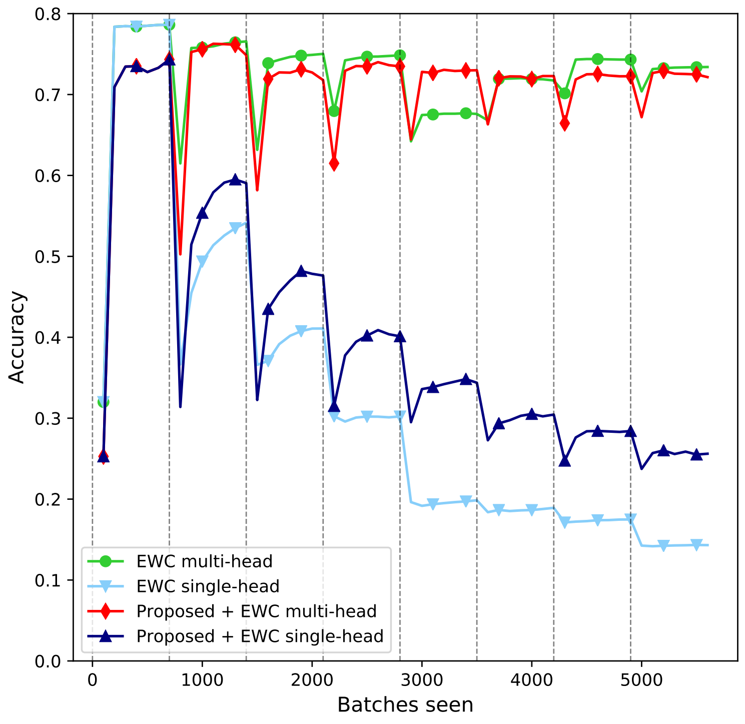

Figure 5.12: Average accuracy in the multi-head and single-head settings, with EWC. Vertical dashed lines indicate transitions in learning experiences.

Table 5.5: Multi-head and single-head performance of the proposed approach.

	Forgetting	Confusion	Multi-head	Single-head
Vanilla	23.1	51.1	61.6	10.5
Proposed	22.6	40.8	59.0	18.2
EWC	5.9	59.1	73.4	14.3
Proposed + EWC	6.2	46.5	72.2	25.6

5.3 Summary

We explored in this chapter the role played by inter-task similarity in continual learning models. We showed that inter-task similarity has a significant effect on the severity of forgetting. Moreover, we showed that continual learning models, such as LwF and EWC, are sensitive to changes in inter-task similarity. Building on those insights, we proposed a rehearsal-based extension to LwF and EWC, and showed that a 1% memory budget is enough to improve performance when learning sequences with low inter-task similarity.

In the second part of this chapter, we proposed a strategy to improve the performance of continual learning models in the single-head setting. We showed that through a combination of a loss function more suited to this learning framework and the use of auxiliary unlabelled data, we are able to achieve a significant improvement in the single-head average accuracy. While the proposed approach addresses confusion and not forgetting, we showed that combining it with a method that counters forgetting, such as EWC, results in further improvement to single-head performance.

Chapter 6

Conclusion

6.1 Summary

The last decade has seen significant advances in the field of deep learning. Whether in computer vision, natural language processing, or speech recognition applications, deep models have shown remarkable capabilities.

Nonetheless, and as success inevitably attracts scrutiny, we are becoming more aware of the limitations of current deep learning technologies. This dissertation focused on one of those limitations: that, while they tend to excel on isolated tasks, deep models perform poorly in a continual learning setting.

We saw that catastrophic forgetting is often seen as the primary culprit behind this poor performance in the continual learning setting, leading continual learning researchers to focus on devising various methods to counter forgetting. We attempted to build on some these methods in this dissertation. We also tackled other aspects of the continual learning problem, including the effect of task properties on performance and the gap between performance in the multi-head and single-head settings.

We saw in Chapter 3 that importance estimation plays a critical role in fixed-capacity continual learning models, where it is important to balance countering forgetting with freeing up model capacity to allow additional learning. We proposed a novel unit importance estimation method based on class separation. The approach builds on recent work by Jung et al. (2020), and draws on the work on Fisher linear discriminant analysis (Murphy, 2012, p. 274). We showed that by using a unit's ability to discriminate between classes as a measure of its importance, we are able to mitigate forgetting while keeping intransigence in check.

We took a different approach to countering forgetting in Chapter 4, arguing that encouraging durable representations—representations that are in and of themselves less susceptible to forgetting—can be as effective at countering forgetting as using explicit *post hoc* penalties (Kirkpatrick et al., 2017; Chaudhry et al., 2018; El Khatib and Karray, 2019a). We showed that using auxiliary unsupervised tasks, we are able to reduce the amount of re-optimization of model parameters from task to task, thus improving continual learning performance and reducing forgetting, without an explicit forgetting penalty.

In Section 5.1, we explored the effect of task properties on the performance of continual learning models and showed that lower inter-task similarity leads to higher forgetting. We demonstrated this effect on recent continual learning methods, and showed that the performance of methods such as learning without forgetting (LwF) (Li and Hoiem, 2016; Li and Hoiem, 2018) and elastic weight consolidation (EWC) (Kirkpatrick et al., 2017) varies significantly with inter-task similarity. We showed that using a small replay memory can alleviate some of the performance degradation when dealing with dissimilar tasks.

Finally, in Section 5.2, we addressed the performance of continual learning models in the single-head setting. We explored the sources of performance decay in this setting relative to the multi-head setting and showed that improving performance in the single-head setting requires not only countering forgetting, but also countering confusion. We proposed a simple approach to reduce confusion in the single-head setting, using auxiliary unlabelled data and a modified cost function, and demonstrated that it can be used effectively in conjunction with EWC to simultaneously reduce forgetting and confusion.

6.2 Future Work

As we look toward the future, we note potential directions to extend the work presented in this dissertation.

First, we demonstrated the effectiveness of the importance estimate presented in Chapter 3 for sequences of binary tasks. We would expect similar performance gains to extend to sequences of multi-class tasks. The extension to a multi-class setting is possible using multi-class versions of Fisher linear discriminant analysis (Murphy, 2012, p. 274).

Second, while we presented learning durable representations in Chapter 4 as an alternative to *post hoc* regularization methods, we see the two approaches as complementary. Depending on the setting, durable representations are still subject to forgetting, and stand to benefit from being used in conjunction with explicit penalties on forgetting, such as those used in elastic weight consolidation (Kirkpatrick et al., 2017) or other regularization-based approaches (Chaudhry et al., 2018; Lopez-Paz and Ranzato, 2017).

Third, we saw in Section 5.1 how task properties such as inter-task similarity influence the behaviour and performance of continual learning models. An interesting potential extension to the work we presented is to use estimates of inter-task similarity to adapt continual learning methods in real-time. While it is clearly not feasible to calculate similarity between two task distributions that are not available concurrently, one could make use of replay memory samples, limited as they may be, to calculate an estimate of inter-task similarity.

Finally, while we focused in this dissertation on sequences of image classification tasks, we note that the continual learning framework is more general, and applicable to other types of tasks. There is potential to apply some of the ideas proposed here to other domains, such as reinforcement learning tasks, for example.

References

R. Aljundi, E. Belilovsky, T. Tuytelaars, L. Charlin, M. Caccia, M. Lin, and L. Page-Caccia. Online continual learning with maximal interfered retrieval. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 11849–11860. Curran Associates, Inc., 2019a.

R. Aljundi, M. Rohrbach, and T. Tuytelaars. Selfless sequential learning. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019b.

D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. Hannun, T. Han, L. Johannes, B. Jiang, C. Ju, B. Jun, P. LeGresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, Y. Xie, D. Yogatama, B. Yuan, J. Zhan, and Z. Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 173–182, New York, New York, USA, 20–22 Jun 2016. PMLR.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In Y. Bengio and Y. LeCun, editors, *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proc. European Conference on Computer Vision (ECCV)*, 2018.

L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4): 834–848, 2018.

Z. Chen and B. Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(3):1–145, 2016.

J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio. Attention-based models for speech recognition. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 577–585. Curran Associates, Inc., 2015.

A. Chrysakis and M.-F. Moens. Online continual learning from imbalanced data. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1952–1961, Virtual, 13–18 Jul 2020. PMLR.

T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha. Deep learning for classical japanese literature. *ArXiv*, abs/1812.01718, 2018.

A. Coates, H. Lee, and A. Y. Ng. An analysis of single layer networks in unsupervised feature learning. In *AISTATS*, 2011.

V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *ArXiv*, abs/1603.07285, 2016. URL https://arxiv.org/pdf/1603.07285.pdf.

V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. C. Courville. Adversarially learned inference. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

A. El Khatib and F. Karray. Preempting catastrophic forgetting in continual learning models by anticipatory regularization. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2019a.

A. El Khatib and F. Karray. Strategies for improving single-head continual learning performance. In F. Karray, A. Campilho, and A. Yu, editors, *Image Analysis and Recognition*, pages 452–460, Cham, 2019b. Springer International Publishing. ISBN 978-3-030-27202-9.

M. Farajtabar, N. Azizan, A. Mott, and A. Li. Orthogonal gradient descent for continual learning. In S. Chiappa and R. Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3762–3773, Online, 26–28 Aug 2020. PMLR.

M. Frean and A. Robins. Catastrophic forgetting in simple networks: an analysis of the pseudorehearsal solution. *Network: Computation in Neural Systems*, 10(3):227–236, 1999.

R. French. Dynamically constraining connectionist networks to produce distributed, orthogonal representations to reduce catastrophic interference. In *In Proceedings of the 16th Annual Cognitive Science Society Conference*, 1994.

R. M. French. Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. In *In Proceedings of the 13th Annual Cognitive Science Society Conference*, pages 173–178. Erlbaum, 1991.

R. M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999. ISSN 1364-6613.

R. M. French and N. Chater. Using noise to compute error surfaces in connectionist networks: A novel means of reducing catastrophic forgetting. *Neural Computation*, 14: 1–15, 2002.

T. Furlanello, J. Zhao, A. M. Saxe, L. Itti, and B. S. Tjan. Active long term memory networks. *ArXiv*, abs/1606.02355, 2016. URL http://arxiv.org/abs/1606.02355.

S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

S. Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11(1):23–63, 1987.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. URL http://arxiv.org/abs/1503.02531.

F. Huszar. Note on the quadratic penalties in elastic weight consolidation. *Proceedings of the National Academy of Sciences*, 115(11):E2496–E2497, 2018. ISSN 0027-8424.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

J. Jo and Y. Bengio. Measuring the tendency of cnns to learn surface statistical regularities. *ArXiv*, abs/1711.11561, 2017. URL http://arxiv.org/abs/1711.11561.

H. Jung, J. Ju, M. Jung, and J. Kim. Less-forgetful learning for domain expansion in deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

S. Jung, H. Ahn, S. Cha, and T. Moon. Adaptive group sparse regularization for continual learning. *ArXiv*, abs/2003.13726, 2020. URL https://arxiv.org/abs/2003.13726.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

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

J. Kolen and J. Pollack. Backpropagation is sensitive to initial conditions. *Complex Systems*, 4:269–280, 1990.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, Nov 1998.

C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-Supervised Nets. In G. Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 562–570, San Diego, California, USA, 09–12 May 2015. PMLR.

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

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

D. Lopez-Paz and M. A. Ranzato. Gradient episodic memory for continual learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6467–6476. Curran Associates, Inc., 2017.

Y. Luo, D. Tao, and Y. Wen. Exploiting high-order information in heterogeneous multi-task feature learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 2443–2449, 2017.

A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, pages 72–88, Cham, 2018. Springer International Publishing.

M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165, 1989.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective.* The MIT Press, 2012.

J. Murre. The effects of pattern presentation on interference in backpropagation networks. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society*, pages 54–59, 1992.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, Nov 2011.

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

A. Robins. Catastrophic forgetting in neural networks: the role of rehearsal mechanisms. In *Proceedings of the 1993 the First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, pages 65–68, Nov 1993.

A. Robins. Catastrophic forgetting, rehearsal, and pseudorehearsal. *Connection Science: Journal of Neural Computing, Artificial Intelligence and Cognitive Research*, 7:123–146, 1995.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back propagating errors. *Nature*, 323:533–536, 1986.

A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *ArXiv*, abs/1606.04671, 2016. URL http://arxiv.org/abs/1606.04671.

J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. Progress & compress: A scalable framework for continual learning. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4528–4537, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

J. Serra, D. Suris, M. Miron, and A. Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 4548–4557. PMLR, 10–15 Jul 2018.

H. Shin, J. K. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2990–2999. Curran Associates, Inc., 2017.

K. Shmelkov, C. Schmid, and K. Alahari. Incremental learning of object detectors without catastrophic forgetting. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

C. Shui, M. Abbasi, L.-E. Robitaille, B. Wang, and C. Gagne. A principled approach for learning task similarity in multitask learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 3446–3452. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

A. V. Terekhov, G. Montone, and J. K. O'Regan. Knowledge transfer in deep block-modular neural networks. In S. P. Wilson, P. F. Verschure, A. Mura, and T. J. Prescott, editors, *Biomimetic and Biohybrid Systems*, pages 268–279. Springer International Publishing, 2015.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *ArXiv*, abs/1708.07747, 2017.

J. Yoon, E. Yang, J. Lee, and S. J. Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018.

F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3987–3995, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

J. Zhang, J. Zhang, S. Ghosh, D. Li, J. Zhu, H. Zhang, and Y. Wang. Regularize, expand and compress: Nonexpansive continual learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

Y. Zhang and D.-Y. Yeung. Learning high-order task relationships in multi-task learning. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI-13*, pages 1917–1924, 2013.

Y. Zhang, Y. Wei, and Q. Yang. Learning to multitask. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 5771–5782. Curran Associates, Inc., 2018.