KU LEUVEN

FACULTY OF ENGINEERING SCIENCE

Combining Rehearsal with Linear Discriminant Analysis in Online Continual Learning

Fanourios Mathioulakis

Thesis submitted for the degree of Master of Science in Artificial Intelligence

Thesis supervisor:
Prof. Dr. Marie-Francine Moens

Assessors:
Thierry Deruyttere
Matthias De Lange

Mentor:
Aristotelis Chrysakis

Academic year 2021 – 2022

Preface

Acknowledgments

I would like to thank my mentor Aristotelis Chrysakis for his help and guidance throughout the conduction of the entire thesis. I also wish to thank my thesis supervisor Prof. Dr. Marie-Francine Moens for her valuable feedback. Last but not least, I want to express my gratitude to my family and friends that supported me throughout the completion of my master's degree.

Fanourios Mathioulakis

Contents

Abstract

Current state-of-the-art neural networks fail to learn sequentially from data. When a network is trained on a new task previously learned representations are quickly overwritten by new knowledge, a phenomenon known as catastrophic forgetting. Continual learning is the area of machine learning that tries to tackle this phenomenon. This thesis focuses on online learning, which is a paradigm of continual learning that assumes a possibly infinite stream of not-independent and identically distributed data with no clear task boundaries. This is a realistic but challenging setting that has not been extensively studied. We build on recent work proposing the use of Linear Discriminant Analysis (LDA) to continually train the classification layer of a pre-trained convolutional neural network (CNN) in online continual learning. Specifically, we propose an extension that uses rehearsal to train the entire network instead of relying on a frozen feature extractor. However, the use of rehearsal to train the feature extractor is faced with the problem of drifting features. Namely, the stored mean feature vectors utilized by LDA drift away from the ever-changing feature space. We describe two different approaches for dealing with this problem, one that continually transforms the stored feature vectors to the new feature space and one that avoids the occurrence of drift by using approximations of the feature vectors. We show that the latter approach is an effective way of handling drift without compromising the effectiveness of LDA. Furthermore, we demonstrate that our proposed model outperforms simple rehearsal and SLDA on three different datasets already from low memory sizes (20 samples per class) with higher memory sizes resulting in even greater performance.

List of Figures and Tables

List of Figures

List of Tables

Chapter 1

Introduction

A long-standing challenge of artificial intelligence is building machine learning systems that can learn sequentially from data (Hassabis et al., 2017). Current state-of-the-art neural networks are trained on large collections of independent and identically distributed (i.i.d.) data to learn a specific task. However, when a network is trained on a new task, its performance on the previously learned task quickly degrades. This documented phenomenon is referred to as catastrophic forgetting (Goodfellow et al., 2013; McCloskey and Cohen, 1989; Ratcliff, 1990). Catastrophic forgetting is closely linked to the stability-plasticity dilemma (Abraham and Robins, 2005; Ditzler et al., 2015; Grossberg, 1982, 2013; Mermillod et al., 2013) which refers to the trade-off that a learning system faces between consolidating past knowledge and adapting to new observaions. A learning system, artificial or not, needs to incorporate a level of plasticity in order to integrate new knowledge. At the same time, the integration of new knowledge should not interfere with previously obtained knowledge. This is a task easily handled by humans, who are capable of learning continually throughout their life (McClelland et al., 1995; Murray et al., 2016; Power and Schlaggar, 2017). However, machine learning systems are currently far from accomplishing this objective.

The scientific area that focuses on tackling catastrophic forgetting in machine learning systems is referred to as continual or lifelong learning. Continual learning has been increasingly drawing attention over the last years. Two main settings can be distinguished in the current scientific literature, the task-incremental (De Lange et al., 2021; Kemker et al., 2018; Rebuffi et al., 2017; Wu et al., 2019) and the online setting (Aljundi et al., 2019a,b; Chrysakis and Moens, 2020; Hayes and Kanan, 2020). In the former setting, each task has clearly defined boundaries and can be identified by the learner during training and inference. The training is performed by iterating multiple times over a task's data, similar to how conventional training is performed on a given dataset. Furthermore, all of the data related to the task is available at training time in its entirety. In online continual learning, the learner is trained on small batches of data from a theoretically infinite stream of not-i.i.d. data that once seen can not be revisited. This is a more realistic and challenging setting with many

possible applications (Aljundi et al., 2019a; Hayes et al., 2020) and is the assumed setting for the present thesis.

Continual learning algorithms can be categorized into three main families with respect to their approach to tackling catastrophic forgetting. These constitute the replay methods, regularization methods, and parameter isolation methods (De Lange et al., 2021). We make a detailed presentation of these methods in chapter 3.

In recent work, Hayes and Kanan (2020) presented a novel technique for combating catastrophic forgetting in image classification tasks. They proposed training the classification layer of a frozen pre-trained network using Linear Discriminant Analysis (LDA) (Ghojogh and Crowley, 2019), a method introduced as deep Streaming Linear Discriminant Analysis (SLDA). Updating the output layer through LDA was shown to be resistant to catastrophic forgetting making it suitable for the continual learning setting. The resulting model was able to achieve state-of-the-art performance while being computationally efficient and having low memory requirements. However, SLDA is limited due to its reliance on a frozen feature extractor. This prohibits the model from learning new feature representations and makes its performance heavily dependent on the pre-training phase. The SLDA model introduces some interesting ideas but its static nature makes it a non-viable solution to continual learning.

The present thesis was inspired by the shortcomings of the SLDA model and its potential to improve upon learning new feature representations. Specifically, our goals were to investigate whether SLDA can be successfully combined with a rehearsal-based scheme and if the combined model leads to better performance than the SLDA and rehearsal models individually. The introduction of rehearsal in the SLDA model is essentially an attempt to alleviate the requirement for a frozen feature extractor. However, this task is faced with certain challenges which are described in detail in chapter 5.

The structure of the thesis is designed to slowly familiarize the reader with the main ideas before presenting our contributions. We first make a brief presentation of artificial neural networks, describing certain types of networks as well as the training process. Then in chapter 3 we lay the theoretical groundwork for continual learning followed by a detailed description of the SLDA model in chapter 4. Chapters 5,6, constitute the core of this thesis and are concerned with our work. Finally, in chapter 7 we summarize the main conclusions derived from our investigation and make suggestions for possible directions for further research.

Chapter 2

Artificial Neural Networks

Many of the breakthroughs in machine learning over the last years can be largely attributed to the development of new specialized neural network architectures. Neural networks have demonstrated their ability to perform in a variety of machine learning areas such as computer vision, natural language processing, generative modeling, reinforcement learning, etc. They are the focus of current machine learning research and are evolving at a fast pace (Liu et al., 2017). In this chapter we make a general introduction to neural networks starting with conventional feedforward networks and continuing with convolutional neural networks due to their relevance to image processing tasks. Finally, we briefly describe the training process and also present the paradigm of transfer learning.

2.1 Feedforward Neural Networks

Artificial neural networks (ANNs) borrow their name from biological neural networks found in the human brain. Even though it is not unusual to see claims that ANNs work in a similar way to the biological brain, these claims are rather far-fetched. ANNs are indeed inspired by neuroscience but they are definitely not exact models of the human brain. They should be better thought of as neuro-inspired models guided and evolved through mathematical and engineering principles (Goodfellow et al., 2016).

A simplified depiction of a biological neuron is presented in figure 2.1. As shown, the neuron can be divided into three main parts, the cell body, the dendrites, and the axons (Gerstner and Kistler, 2002). The dendrites receive information from other neurons in the form of chemicals called neurotransmitters . These are the molecules used by the nervous system to transmit messages between neurons (Webster, 2001). They cause electrical changes that are interpreted at the cell body which determines if the signal will be transmitted through the axon. A signal that ends up being transmitted reaches the axon terminals which are connected with dendrites from subsequent neurons. The collection of the neurons and their in-between connections form a neural network.

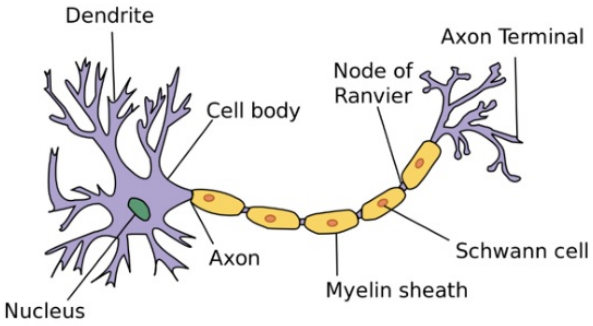

FIGURE 2.1: Simplified depiction of a biological neuron. Source: (Broucke, 2021)

The biological neuron is modeled through a mathematical abstraction called the perceptron, (see figure 2.2), originally proposed by Rosenblatt (1958). Given an input vector $\mathbf{x}$ and the corresponding weight vector $\mathbf{w}$ the output y of the perceptron is calculated as:

$$y = \phi(\mathbf{w}^\top \mathbf{x} + b) \tag{2.1}$$

where b is the *bias* (a constant term) and $\phi(\cdot)$ is a the *activation function* (usually a non-linear function). A more compact form of the equation 2.1 can be derived by adding an extra input term $x_0 = 1$ with corresponding weight w_0, which is now serving as the bias term:

$$y = \phi(\mathbf{w}^\top \mathbf{x}). \tag{2.2}$$

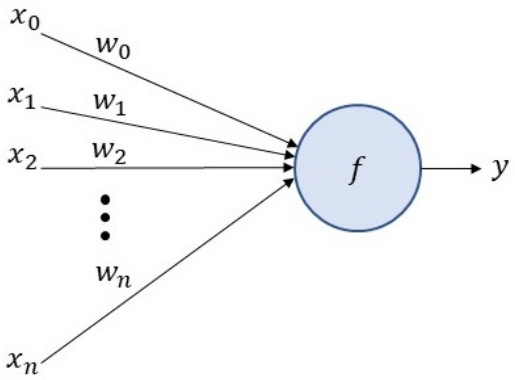

FIGURE 2.2: The perceptron: a mathematical abstraction of the biological neuron.

A *feedforward neural network*, also called *multilayer perceptron* (MLP), is formed when multiple perceptrons are connected in a layer-wise fashion as shown in figure 2.3. Every neuron in a layer is connected to every other neuron in the next layer, with the output of each neuron serving as one of the inputs of the subsequent neurons. This high connectivity is the reason why these networks are also referred to as *fully*

connected neural networks. The name feedforward is attributed to the fact that in this type of architecture information flows in a forward direction. The input vector $\mathbf{x}$ is propagated from the *input layer* (first layer) all the way to the *output layer* (last layer) with no feedback loops. The layers between the input and the output layer are the *hidden layers* and their number along with the number of neurons in each hidden layer determine the network architecture (Goodfellow et al., 2016; Gurney, 2018). Larger numbers of hidden layers result in deeper architectures which is the focus area of *Deep Learning*.

The size of the input/output layer is determined by the task at hand. Specifically, the size of the input layer is equal to the dimensionality of the input data. For a classification problem, where the goal is to predict a class out of n different classes, the output layer has a size equal to n. Each of the n output nodes corresponds to the predicted value for that class. Usually, a *softmax* activation function is used in this classification layer to convert the raw values to probabilities. The conversion of each raw value z_i is calculated from the following equation:

$$\phi(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=i}^{n} e^{z_j}}. \tag{2.3}$$

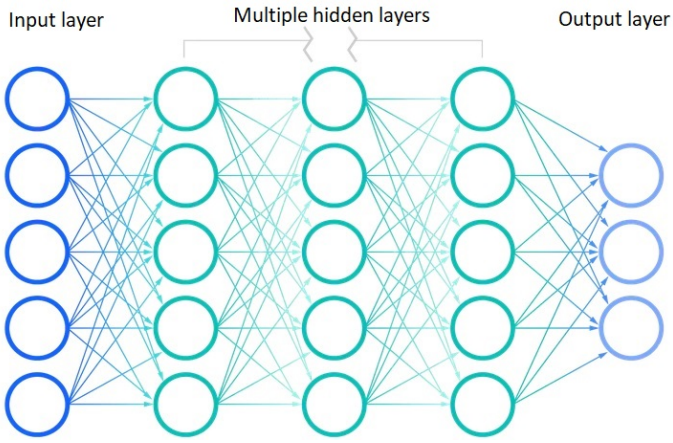

FIGURE 2.3: Depiction of a multilayered feedforward neural network. Source: (IBM Cloud Education, 2020b)

From a mathematical perspective, a neural network is a function approximation model. It is used to approximate the mapping of an unknown function $f(\cdot)$ for a given task. For example in an image classification task, this function could be the mapping of a given matrix representation of an image to its corresponding label. Neural networks are so popular because they have repeatedly succeeded in approximating complex functions at a variety of difficult tasks (Liu et al., 2017). According to the universal approximation theorem, a feedforward network with at least one hidden layer and regardless of the activation function can approximate any continuous function arbitrarily well in a given metric provided it has enough neurons

(Cybenko, 1989; Hornik et al., 1989). This is not a constructive theorem, meaning that it tells us that an MLP with one hidden layer can approximate any function but it does not tell us how to construct such an MLP. For example, it does not specify the number of neurons needed or what training algorithm to use. Nevertheless, it is a powerful theorem that reassures us about the capacity of MLPs as function approximation tools. Unfortunately, it is true that even if an MLP with a single hidden layer can approximate any function, its layer size may need to be too large making its implementation infeasible. However, by increasing the number of hidden layers one can keep the number of neurons relatively low (Goodfellow et al., 2016).

In summary, neural networks are extremely useful function approximation tools and remain the focus of deep learning research. However, the simple architecture of a feedforward network described here is not the one responsible for the grand successes of the last years. Specialized network architectures in areas such as computer vision and natural language processing have shown impressive results which could not be attained with the MLP. In the following section, we present a type of network that has been widely adopted by the field of computer vision due to its superiority in image processing tasks to the classical MLP.

2.2 Convolutional Neural Networks

In the area of computer vision, a type of neural network that has dominated the field due to its success in image processing tasks is the *Convolutional Neural Network* (CNN) (Goodfellow et al., 2016; Gu et al., 2018; O'Shea and Nash, 2015). CNNs are a great example of neuroscience-inspired artificial intelligence. The intuitions for this network's architecture were drawn from knowledge about the visual cortex of the animal brain and turned out to be extremely successful (Goodfellow et al., 2016).

The main operation applied in the layers of a CNN is the *convolution* operation from which the network draws its name. To demonstrate how this operation works in practice, let's first consider a 5×5 matrix representing a small-sized single-channel image as in figure 2.4. The yellow area represents the overlap of the image with a 3×3 *kernel* (also called filter) used for the operation. Each position of the kernel has a fixed weight value shown by the small red numbers in the lower right corner of each square. The operation is performed by sliding the kernel through the image while performing element-wise multiplication between the kernel weights and the pixel values and summing the obtained values at each position. For example, by starting at the upper left corner as shown in the figure, we get the first element of the output by multiplying each kernel weight with its corresponding pixel value and summing the obtained values to get 4 as the result. Shifting the kernel one pixel to the right and performing the operation again we get the output value next to our previous result of 4. By sliding the kernel horizontally and vertically through the entire image we get the output of the operation. In our example, for the given sizes of image and kernel and assuming single-pixel steps at each direction it is easy to see that the result is a 3×3 matrix. The amount of shifting of the kernel at each

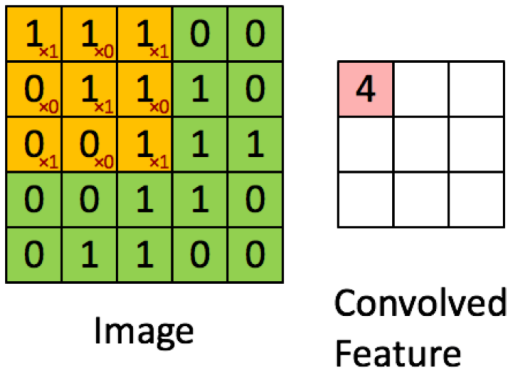

FIGURE 2.4: Visualization of the convolution operation. Source: (Ng et al., 2010)

direction is called the *stride*. For example, using a stride $s = 2$ in our example would result in an output of size 2×2.

We can see that the convolution operation results in an image of lower size, which may be problematic when multiple convolutions are applied, leading to a progressively smaller size. In practice, to maintain the same size between input and output *padding* is used, which essentially is the addition of extra rows and columns at the edges of the image. Usually, the added pixels have zero value as to not interfere with the result of the convolution. Another property of padding is that it makes better use of the information at the edges of the image. This can be understood by thinking that in the absence of padding information at the inner part of the image is used much more often to calculate output results as opposed to information at the edges (Goodfellow et al., 2016). Taking into consideration all the parameters of the convolution operation like kernel size k, stride s and padding p, and given an input image of size i, the size o of the output image can be calculated from the following equation:

$$o = \frac{i - k + 2p}{s} \tag{2.4}$$

Finally, it should be noted that the convolution operation for a 3-channel image is identical with the only modification of using a 3-channel kernel.

The convolution operation can be seen as a feature detection operation. Each kernel can be thought of as a filter whose purpose is to detect certain features on an image. By letting the kernel weights be trainable parameters the network is able to learn useful feature detectors through the training process. For example a kernel of the form:

$$K = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix} \tag{2.5}$$

would be suited for detecting vertical edges (Ziou et al., 1998). The important thing to consider is that each layer can have many of these feature detectors. As we saw previously the result of the convolution operation is a single-channel image. When

applying convolution with n different filters on an image and combining the resulted outputs on a single matrix we get an n-channel output. So for a $w \times h \times 3$ RGB input image, and n kernels of size $k \times k \times 3$, the result would have a size of $w \times h \times n$ assuming padding was used. In the subsequent layer, n' different kernels with size $k' \times k' \times n$ would be used to give an output of size $w \times h \times n'$. Hence, the number of channels of the output representation at a given layer is always equal to the number of filters used at that layer.

The use of convolution gives rise to the properties of *sparse connectivity* and *parameter sharing* (Goodfellow et al., 2016) of a CNN which are crucial for its success in image processing tasks. As discussed in the previous section, in traditional neural networks the output of neurons at a given layer is derived by an operation that depends on the values of all the neurons of the previous layer. So for a layer with m input neurons and n output neurons there are $m \times n$ connections. Conversely, if we look back to figure 2.4, we see that an output is not dependent on all pixels of the input image but rather only on those covered by the kernel (which typically is orders of magnitude smaller than the entire image). This means that for a kernel that covers k pixels in total and n outputs, there will be $k \times n$ connections. By having sparse connections, a CNN improves computational efficiency since fewer operations are needed to calculate an output. Furthermore, it reduces memory requirements as fewer parameters need to be stored, which also strengthens its generalization performance.

The other important idea is that of parameter sharing. As explained previously, conventional ANNs have a separate parameter dedicated for every input-output pair. This means that there need to be stored $m \times n$ different parameters for matrix multiplication. With convolution, this number is reduced to k, since the same parameters k of a given kernel are used at every part of the image. Since k is negligible compared to $m \times n$, there is a significant decrease in memory requirements. When considered together with parameter sharing, the memory reduction is even greater showing the superiority of the CNN to an MLP for image processing tasks.

So far we have only described the operation of convolution. However, a layer of a CNN usually involves two more steps before producing an output to be passed to the next layer. After the convolution step, a non-linear activation function is applied to the values of the output matrix. This ensures that non-linear representations can be constructed. The final step typically involves the *pooling* operation. This operation reduces the size of the representation by replacing an image region with a locally summarizing statistic. With average pooling, for example, a $k \times k$ kernel is slid through the image and replaces every $k \times k$ region with the average of those values. Another common operation is max pooling which replaces the region with the maximum of those values. Pooling is useful for reducing the size of a representation and accelerating computations while maintaining part of the information in the form of a summarizing statistic. It should be noted that this operation does not affect the number of channels of a representation. The three stages mentioned here, convolution operation, non-linearity application, and pooling can be viewed as a single layer, known as the *convolutional layer*.

A CNN network involves a sequence of convolutional layers (with or without pooling) that transform the input image in a meaningful feature representation. It has been observed that the first layers of a CNN learn simple filters that detect low-level features such as edges and blobs. Layers at deeper parts of the CNN, learn more complex feature detectors capable of detecting high-level features such as eyes or noses in a face recognition task. A typical architecture is shown in figure 2.5. We can see that the CNN consists of a series of convolutional layers followed by some fully connected layers. The part of the network containing the convolutional layers can be thought of as the feature extractor while the fully connected part as the classifier.

An important consideration is that CNNs have automated the feature extraction process. Computer vision scientists used to have to manually extract useful features from images using specialized algorithms. This was a long process that demanded human expertise and domain knowledge. However, with a CNN the feature extraction is performed entirely by the network. During the training process, the network adjusts its weights to find suitable filters that are best for performing the task at hand.

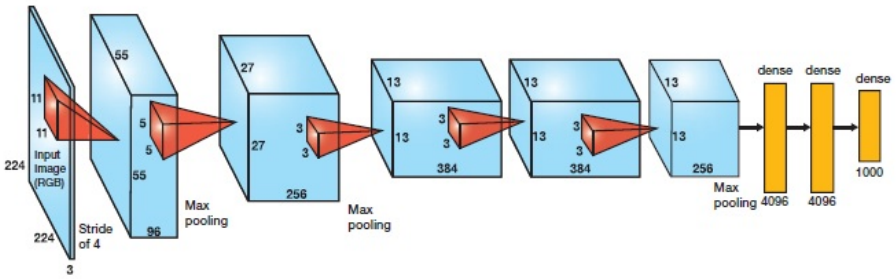

FIGURE 2.5: A typical CNN architecture with multiple convolutional layers (blue boxes) followed by fully connected layers (yellow rectangles). Source: (Broucke, 2021)

In conclusion, CNNs are much better suited for processing image data than conventional MLPs. Using the operations of convolution and pooling, they can extract complex features while keeping memory requirements relatively low and improving computational efficiency and generalization. They are a great example that neuroscience principles can have a real practical impact on the development of artificial intelligence. Understanding the architectural differences between MLPs and CNNs is important for gaining insight as to why CNNs excel at computer vision tasks and appreciating their efficiency as learning models.

2.3 Neural Network Training

Neural networks learn a given task by training on large collections of data. The term training refers to the optimization process by which a neural network adjusts its parameters to optimize a specific objective. Such an objective typically involves the

minimization of an error function, commonly referred to as the *loss function*, *objective function* or *cost function*. This function depends on the network's predictions $\hat{\mathbf{y}}$ about the given data and their corresponding true values $\mathbf{y}$. If $\ell(\hat{y}_i, y_i)$ is the loss for a single instance $\mathbf{x}_i$ of the training set, then the total loss to be minimized is given by the equation:

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{n} \ell(\hat{y}_i, y_i) \tag{2.6}$$

where n is the number of samples in the training set. In this section, we present neural network training by describing the two main algorithms needed for its implementation, *gradient descent* (Cauchy et al., 1847) and *backpropagation* (Rumelhart et al., 1986).

Gradient descent is a first-order optimization algorithm used for minimizing a function by moving in the direction of the negative gradient. Although over the years, more sophisticated optimization algorithms have been proposed (Duchi et al., 2011; Kingma and Ba, 2014; Zeiler, 2012), gradient descent remains one of the most popular algorithms for neural network training. Given a network with parameter vector $\boldsymbol{\theta}$, the loss function is minimized by adjusting the network's parameters through the following update:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{2.7}$$

where η is an adjustable hyperparameter referred to as the learning rate. By taking incremental steps in the direction of the negative gradient, the obtained parameters lead to continuously smaller values of the loss function until a local or global minimum is reached. This clearly demonstrated in figure 2.6. However, in contrast to the simple convex function shown there, neural networks' loss functions are generally complex non-convex functions with many local minima (Li et al., 2017). The size of the optimization steps during training is adjusted by the learning rate η. Large values for η may lead to divergence, making gradient descent unable to reach an optimum. On the other hand, small values require a higher number of iterations for convergence which is not optimal. The learning rate must be tuned to a value that allows for relatively fast convergence while avoiding divergence issues. Besides using a fixed learning rate, it is also possible to use a decaying learning rate which can improve the learning of complex patterns (You et al., 2019).

Training a neural network with gradient descent requires the gradient of the loss function with respect to the network's parameters. This is achieved in a computationally efficient way with the use of the backpropagation algorithm. Discussing in length the details of backpropagation is beyond our scope, thus we will only provide a simplified view mainly for intuition. Backpropagation relies on the ideas of *computational graphs* and *chain rule* to compute the gradient of any function with respect to a variable. The chain rule is a well-known rule of calculus for calculating the derivative of the composition of functions. Given a function $f(y)$ for which $y = y(x)$ is a function of another variable x, the derivative of the composition $(f \circ y)(x)$ with respect to x is calculated using the chain rule:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x}. \tag{2.8}$$

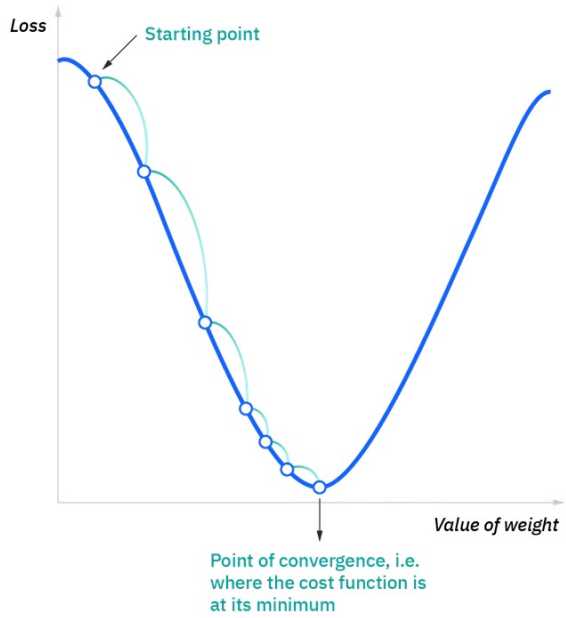

FIGURE 2.6: Optimization trajectory with gradient descent. The loss function is minimized by taking steps in the direction of the steepest descent. Source: (IBM Cloud Education, 2020a)

Equation 2.8 can be generalized for any number of compositions $(f_0 \circ f_1 \circ f_2 \circ ... f_n)(x)$:

$$\frac{\mathrm{d}f_0}{\mathrm{d}x} = \prod_{i=0}^{n-1} \frac{\mathrm{d}f_i}{\mathrm{d}f_{i+1}} \frac{\mathrm{d}f_n}{\mathrm{d}x}. \tag{2.9}$$

Furthermore, in the case of multivariate functions $f(y_1, y_2, ..., y_n)$, with $y_i = y_i(x)$, the chain rule becomes:

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \sum_{i=1}^{n} \frac{\partial f}{\partial y_i} \frac{\mathrm{d}y_i}{\mathrm{d}x}. \tag{2.10}$$

Finally, the second important idea is that of a computational graph. In a computational graph, each node represents a single variable and different variables can be combined through an *operation* to produce an output. An output variable y produced using an input x, is connected with that input through a directed edge. During forward propagation where a neural network propagates an input through its layers to produce a specific output, a computational graph is constructed. Using the relations in this graph and the chain rule it is possible to calculate the derivative of an output with respect to any node of the graph. Thus, the backpropagation algorithm by utilizing these ideas is able to calculate the derivative of the loss function with respect to the network weights, which is subsequently used by gradient descent to optimize the network.

2.4 Transfer Learning

In general, neural networks are trained from scratch in order to learn to perform a specified task on a given domain. A critical concept in machine learning is that of *transfer learning* (Pan and Yang, 2009; Torrey and Shavlik, 2010; Weiss et al., 2016). It refers to the idea of using a pre-trained model on a new task, exploiting the model's knowledge about the previous task on which it was trained. This facilitates the training process by reducing the amount of data needed to learn a new task and by leading to faster convergence. The paradigm of transfer learning has been successfully applied in deep learning to areas such as computer vision and natural language processing (Cao et al., 2013; Chen et al., 2018; Cui et al., 2018; Ruder et al., 2019). Following, we describe transfer learning for the image classification task due to its relevance to the present thesis.

In computer vision, the CNN architecture is responsible for the wide application and success of transfer learning in image processing tasks. As mentioned in section 2.2, CNNs learn low-level features representations in their early layers, which become increasingly complex and task-specific at deeper parts of the network. For example for an animal recognition task, the last layers of the network may learn to detect features such as cat faces or tails. In contrast, early layers learn general features such as edges or blobs. It makes sense then, that since task-specific features are learned only at deeper levels of the network, knowledge about general features should be transferable across tasks (Yosinski et al., 2014). If instead of animal species we trained a CNN to recognize daily objects, then early layers would most likely learn similar features representations. In that sense, it is wasteful to train an entire network from scratch only to learn again similar feature representations in some of its layers. This is where transfer learning can be exploited to improve training efficiency.

In practice, transfer learning is applied by utilizing a *pre-trained* model, a model already trained on a specific dataset to perform some task. Given that the new task to be learned is related to some extent to the already learned task, the acquired knowledge can be transferred in a straightforward way. This is accomplished by allowing the weights of only the last few layers of the network to be trainable while freezing the rest of the network's parameters. This way, the knowledge about feature representations already learned by the first number of layers is transferred during the learning of the new task while the last layers are trained on the new data to extract task-specific features. The specific number of trainable layers depends largely on the similarity of the new and the old task. For example, when the two tasks are closely related it is possible to freeze all the network's parameters except the last classification layer. This method uses the CNN as a frozen feature extractor and fine-tunes the output layer or some the of last fully connected layers to classify the new data without learning new feature representations (see figure 2.7). If the tasks are not closely related then the last n layers should be trained on the new data. The number of layers n, is found through experimentation and there is not a definite rule that can be used to specify it. With this approach, the CNN learns new feature representations but also utilizes those learned from the previous task to do so.

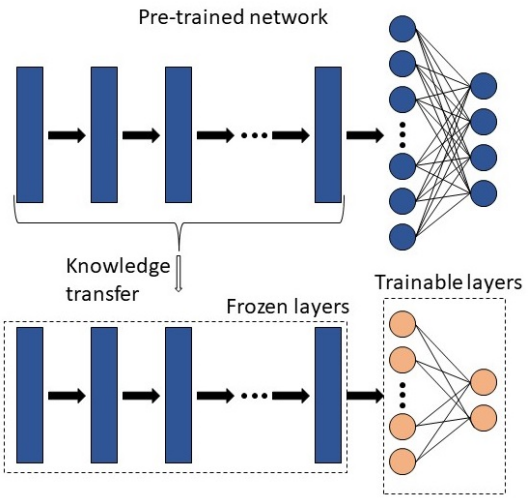

FIGURE 2.7: A typical example of transfer learning. The last layers of the network are replaced and trained on the new task while the rest of the network is kept frozen.

The obvious advantage of transfer learning is reduced training time and efficient learning. However, there are cases where the use of transfer learning is a necessary step to learn a task and is not utilized only as a way to improve efficiency. This concerns tasks for which a limited amount of data is available. A common domain that faces this problem is radiology (Candemir et al., 2021). Using deep learning models for medical imaging classification is usually limited by the few samples available to train the model. Training a model on a limited amount of data leads to poor performance. Luckily, transfer learning can be exploited to alleviate these limitations. Using a model pre-trained on a larger dataset and training only the last few layers of the model on the limited data of the new task one could improve performance. Consequently, the use of transfer learning allows for training models on tasks with limited data, a practice which otherwise would result in possibly poorer performance.

Chapter 3

Continual Learning in Neural Networks

Continual, *incremental*, or *lifelong* learning is the area of machine learning that focuses on developing systems capable of learning from experience in a sequential way. This area has drawn much attention over the last years with significant progress being made. Nevertheless, neural networks are still far from succeeding in incremental learning. In this chapter, we begin by discussing the reasons why neural networks currently fail at this task and present the barriers that learning systems face when trying to accomplish it. Then we make a distinction between the two main settings in continual learning, the task-incremental, and the online setting, with a focus on the latter being the area of interest of this thesis. Finally, we present different families of continual learning algorithms found in the literature and discuss their strengths and limitations.

3.1 Catastrophic Forgetting

Current state-of-the-art neural networks fail to learn in a sequential way. The observed phenomenon that neural networks (and machine learning systems in general) suffer from a sudden decrease in performance on a previously learned task when trained on a new one is known as *catastrophic forgetting* (Goodfellow et al., 2013; McCloskey and Cohen, 1989; Ratcliff, 1990). It is interesting to note, that this phenomenon is not exclusive to artificial learning systems. It is hypothesized that neocortical neurons in the human brain use a learning algorithm that is susceptible to catastrophic forgetting complemented by an experience replay system used to reduce information overwrite (McClelland et al., 1995). As stated by Goodfellow et al. (2013), this means that it is acceptable for machine learning systems to make use of learning algorithms prone to forgetting as long as they use complementary algorithms to mitigate information loss.

In neural networks, catastrophic forgetting occurs due to the large weight changes during training. Given a new task, the parameters of a neural network are updated

with a gradient-based algorithm to minimize the loss function of the new task. By definition, this optimization scheme does not consider the overwrite of previous information. When parameters are updated such as to minimize the loss of the new data, they inevitably disrupt acquired knowledge. This trade-off between acquiring new knowledge while maintaining the existing one is known as the *stability-plasticity dilemma* (Abraham and Robins, 2005; Ditzler et al., 2015; Grossberg, 1982, 2013; Mermillod et al., 2013). Any system that learns incrementally, artificial or not, is ultimately faced with this dilemma. A level of plasticity is important to allow for the learning of new information. At the same time, a level of stability is needed to ensure the previous knowledge is maintained. Conventional neural network training operates on the plasticity side of the trade-off, succeeding in adjusting to new information but at the expense of major information loss. With continual learning research, the goal is to develop learning algorithms that provide a compromise between stability and plasticity in a way that allows for successful lifelong learning similar to how humans and other biological organisms do.

3.2 Continual Learning Settings

Most of the research in continual learning has been focused on the task-incremental setting (De Lange et al., 2021; Kemker et al., 2018; Rebuffi et al., 2017; Wu et al., 2019), mainly because it is a simpler setting to study. In this setting, the data stream can be considered as comprising of a number of clearly separated tasks. The training on each task is performed by iterating multiple times over the task data, which is available in its entirety, similar to how conventional neural network training is performed. To better illustrate this, suppose that we want to sequentially train a neural network on MNIST dataset (LeCun et al., 2010) to classify images of handwritten digits. In the task-incremental setting, each task could be comprised of learning to classify two different digits as shown in figure 3.2. It must be noted

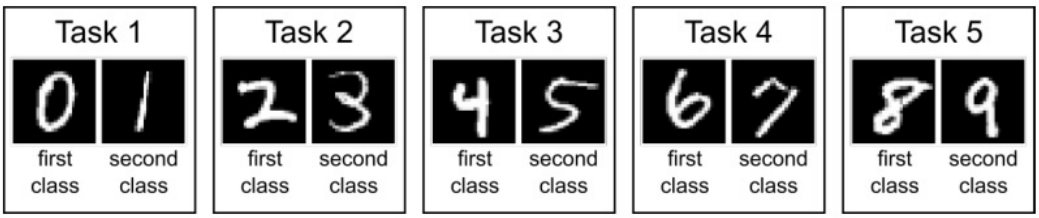

FIGURE 3.1: A typical task sequence in the task-incremental setting. Source: (Van de Ven and Tolias, 2019).

that each separate task can be identified by the model during training and inference. Usually, a multi-head network architecture is used with different output layers devoted to each task (Van de Ven and Tolias, 2019). This greatly simplifies the problem essentially reducing it to a collection of binary classification tasks. The

task-incremental setting is a useful setting for studying continual learning but it makes certain strong assumptions. Training until convergence on a large amount of data for each task is not always a realistic expectation for a continual learner that needs to learn quickly in a non-stationary environment.

A more realistic but also more challenging setting for continual learning that does not require task boundaries or identification is online continual learning (Aljundi et al., 2019a,b; Chrysakis and Moens, 2020; Hayes and Kanan, 2020). In the online setting, the learner is trained on small batches obtained from a theoretically infinite stream of not-i.i.d. data where there are no clear boundaries among tasks. At any given time the learner has only access to the batch obtained from the stream and can iterate through it an arbitrary number of times. However, data from previously seen batches is gone and can not be revisited unless explicitly stored.

3.3 Taxonomy of Continual Learning Approaches

Fighting catastrophic forgetting in neural networks has proven to be a particularly challenging task. There exist a plethora of methods proposing different ways for solving this problem and new ones emerge as research in the field continues. Despite a large number of methods, most of them can be categorized into three main big families: replay methods, regularization methods, and parameter isolation methods. A detailed overview of the different methods can be found in (De Lange et al., 2021; Parisi et al., 2019). In the paragraphs that follow we will elaborate on these methods and discuss their strengths and weaknesses with a greater focus on replay methods, because of their relevance to this thesis. It should be noted, that besides the individual methods described here, hybrid algorithms combining different methods also exist (Lee et al., 2020; Rao et al., 2019)

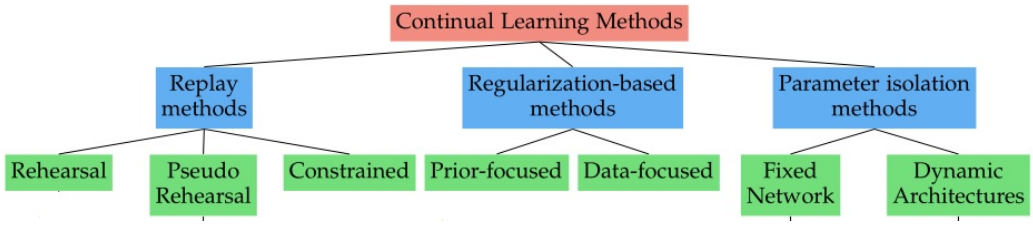

FIGURE 3.2: An overview of the taxonomy of algorithms used in continual learning. Source: (De Lange et al., 2021).

3.3.1 Replay Methods

Replay methods are a family of algorithms that utilize stored samples from previously seen batches of data to mitigate interference of the neural network. A specific and widely used form of replay is *rehearsal* (Chaudhry et al., 2019; Isele and Cosgun, 2018; Rebuffi et al., 2017; Rolnick et al., 2018). It is also an important component

of our proposed model described in chapter 5. In this approach, selected samples from the stream are stored on a dedicated memory buffer and are used to retrain the network along with the new samples. Typically, the memory size is chosen to be much smaller than the size of the stream (1% - 5% of the stream size). This method has proven to be effective in both task-incremental and online settings but it relies on the memory buffer which can result in increased memory requirements for large streams. Furthermore, it runs the risk of overfitting the samples in memory. A detailed investigation on the effectiveness of rehearsal despite the memory overfitting risk can be found in (Verwimp et al., 2021). A variant of rehearsal is *pseudo rehearsal* or *generative replay* (Aljundi et al., 2019a; Robins, 1995; Shin et al., 2017). This approach uses a generative model such as a generative adversarial network (Goodfellow et al., 2014) to replay samples from previous tasks. Even though it alleviates the memory requirements of conventional rehearsal, it comes to the added cost of continually training the generative model further complicating the setting.

The specific choice of algorithms for storing and replaying samples can have an important impact on the performance of the rehearsal-based model. *Reservoir sampling* (Vitter, 1985) is a commonly used family of randomized algorithms for selecting a specified number of samples from a population of unknown size. The idea is to select the samples in a single pass while providing each sample with an equal probability of being selected. We make use of reservoir sampling as a memory population algorithm for our experiments in chapter 6. Rebuffi et al. (2017) used a more sophisticated sampling approach that focuses on accurately approximating the class means in the feature space. Furthermore, for unbalanced streams Chrysakis and Moens (2020) proposed an extension of reservoir sampling that attempts to maintain a balanced subset of the stream in memory. Having an unbalanced memory can quickly lead to forgetting of the minority classes, thus in streams with high class imbalances focusing on obtaining a uniform class distribution in memory is essential for addressing this issue.

For replaying samples, the simplest approach is *random replay* where a number of samples are chosen uniformly at random from memory to augment the obtained batch from the stream. This is also the method we use for implementing rehearsal in chapter 6. A better but more computationally expensive approach is proposed by Aljundi et al. (2019a). This approach selects samples that are going to be maximally interfered when the network is trained on the new batch (see figure 3.3). As the authors mention in the original paper, the motivation for this selection method stems from the observation that the loss of some previous samples may be unaffected, and hence retraining on them does not improve the model.

Another category of replay methods concerns *constrained optimization* methods (Chaudhry et al., 2018; Lopez-Paz and Ranzato, 2017). Similar to rehearsal, these methods require the use of memory to store a subset of the stream. However, the stored instances are used for constraining the weight update of the network instead of explicit retraining. Specifically, they constrain the update induced by new instances such that it does not interfere with previous knowledge.

Finally, there have been some recent attempts in the literature to implement

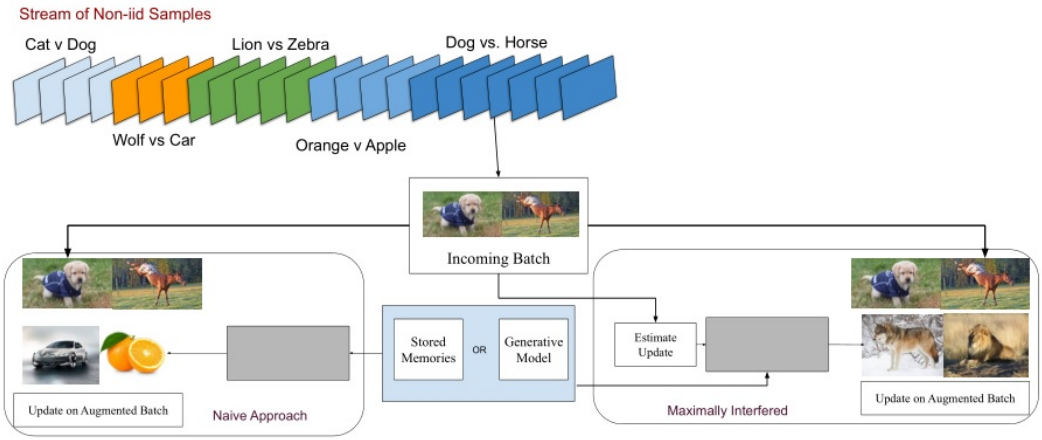

FIGURE 3.3: Diagrammatic representation of maximally interfered retrieval. Naive retrieval (left) randomly samples from memory to augment batch. Maximally interfered retrieval (right) uses information about the estimated network update to sample from memory. Source: (Aljundi et al., 2019a).

feature replay (Hayes et al., 2020; Iscen et al., 2020; Liu et al., 2020; Pellegrini et al., 2020). This is a form of replay that uses latent feature representations instead of original instances. The immediate benefit is that this greatly reduces memory requirements. Furthermore, by not storing raw data, it alleviates possible privacy concerns (Shin et al., 2017; van de Ven et al., 2020). However, it faces a particularly challenging problem. The stored features suffer from a drifting effect that manifests when the weights of the layers before the feature replay layer are updated. When this is allowed to happen the post-update feature space does not correspond to the one where the stored features were obtained from. The result is deterioration in the performance of the model. On the other hand, if the part of the network before the replay layer is chosen to be frozen, or is trained with a very small learning rate, this undermines the learning of new tasks. Consequently, the phenomenon of drift poses a great challenge when trying to apply feature replay in practice.

Replay methods have been effective in combating catastrophic forgetting but they suffer from limited scalability. Namely, as the number of classes increases larger memory buffers are needed to store instances from every class. If the memory is constrained to a fixed size then this would likely lead to deterioration of performance since the stored instances would not represent the original distribution (De Lange et al., 2021).

3.3.2 Regularization Methods

The second major family of continual learning approaches concerns the *regularization-based* methods that constrain the update of the network's parameters as a means to consolidate knowledge. They are implemented in practice through the addition

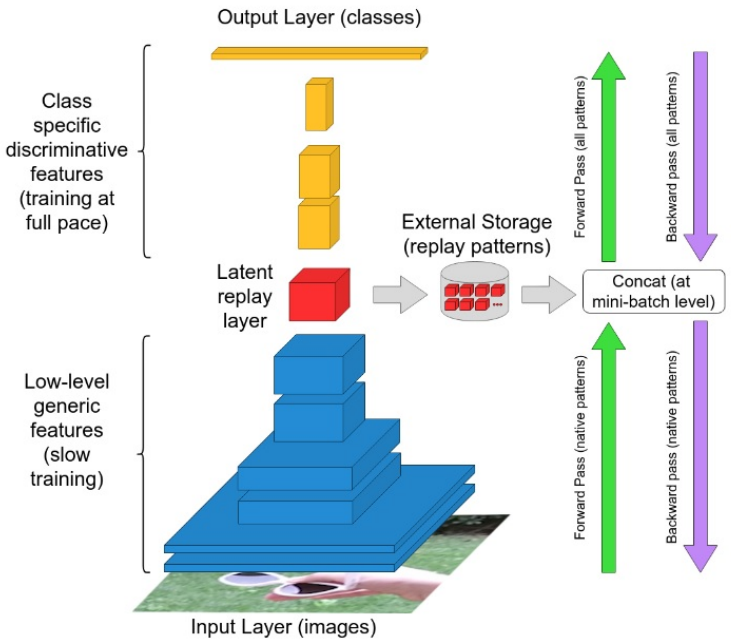

FIGURE 3.4: Diagrammatic presentation of feature replay. Instead of storing raw inputs, feature representations from an intermediate layer (red) are stored and replayed during training. Source: (Pellegrini et al., 2020).

of extra regularization terms in the loss function. The advantage of this family of methods is that they do not require storing raw inputs, which addresses privacy concerns while alleviating memory requirements. It is interesting to note that these approaches are inspired by neuroscience models (Benna and Fusi, 2016; Fusi et al., 2005) which propose that forgetting of consolidated knowledge is combated by the presence of synapses with varying plasticity states.

A specific type of regularization uses *knowledge distillation* (Hinton et al., 2015) from a model trained on previous tasks to the model trained on the new one. Learning without Forgetting (LwF) (Li and Hoiem, 2017) uses the outputs of the previous model on the new task's instances and trains the new model to mimic those outputs while also being trained on the new data. The added constrain to copy the previous model's outputs ensures that the parameters of the new model will be updated in such a way as to minimize loss of previous knowledge. However, with this approach, the computational time needed to train for a given task scales linearly with the number of tasks already learned. On top of that, its success depends heavily on the relevance of the tasks which is a significant limiting factor.

A different approach to regularization-based methods estimates a distribution over the parameters of the network and uses that distribution as a prior when training on a new task. Parameters that are considered important are being protected by an added penalizing term in the loss function. Elastic weight consolidation (EWC)

(Kirkpatrick et al., 2017), introduces a quadratic penalty in the loss function using the difference of the parameters of the old and new tasks. However, it has been demonstrated that EWC is not able to learn new classes in an incremental fashion (Kemker et al., 2018). Similar to EWC, Zenke et al. (2017) propose a method that penalizes the change of influential parameters by allowing individual synapses to estimate their importance for a specific task. This approach however can be applied incrementally in an online fashion throughout learning.

Regularization methods offer an intuitive way to minimize catastrophic forgetting, inspired by neuroscientific principles. Nevertheless, over long sequences, the constraints imposed by these approaches might not be enough to successfully preserve knowledge from old tasks (Farquhar and Gal, 2018). Given the limited amount of parameters in a neural network, the added constraints inevitably lead to a trade-off between the performance of the new versus the old tasks as the number of learned tasks increases (Parisi et al., 2019).

3.3.3 Parameter Isolation Methods

The last family of continual learning approaches constitutes *parameter isolation* methods which dedicate different sets of parameters for each task. By isolating the network's parameters this way, catastrophic forgetting is directly avoided since training on a given task can no longer interfere with previous knowledge. The isolation of parameters can be achieved with both static and dynamic architectures.

When considering static architectures, parameter isolation is implemented by deactivating certain parts of the network. While training for a new task, the corresponding parameters are activated while the rest are masked out (Mallya and Lazebnik, 2018). This way the parameters of the network are distributed over different tasks.

In contrast to static architectures, dynamic architectures are not restrained by a limited number of resources. They can be expanded to adjust for the learning of new tasks by increasing the number of neurons or layers. In (Rusu et al., 2016), the authors propose a method that builds a different sub-network each time a new task needs to be learned. When a sub-network is created, it is also connected via lateral connections to the pre-existing sub-networks which allows the transfer of knowledge. During training on the new task, the trained sub-networks are kept frozen while only the parameters of the added network are updated, hence avoiding catastrophic forgetting. Xiao et al. (2014) proposed a way of dynamically increasing a network's parameters and forming a hierarchical structure by adding new branches at the topmost layers of the network. In contrast, Yoon et al. (2017) allowed their dynamically expanding network (DEN) to add parameters at any layer using selective retraining and group sparse regularization.

In summary, parameter isolation methods can directly avoid catastrophic forgetting by dedicating different parameters for every task. For long sequences, these approaches are able to perform when the neural resources are not limited by a fixed architecture. The obvious disadvantage is that the model's size continues to

grow with the number of tasks learned. Finally, models in this family are generally restricted to the task incremental setting since task identification is needed during training and inference (De Lange et al., 2021).

Chapter 4

Deep Streaming LDA

In this chapter, we present the theory on *deep Streaming Linear Discriminant Analysis* (SLDA) (Hayes and Kanan, 2020) which served as motivation for the present thesis. SLDA is a recently proposed online continual learning algorithm that does not fit any of the general families described in the previous chapter. The idea behind SLDA is to use a frozen pre-trained CNN and update its output layer through *Linear Discriminant Analysis* (LDA) (Ghojogh and Crowley, 2019). Updating the output layer in this way is shown to be resistant to catastrophic forgetting which makes the model suitable for the continual learning setting. We begin by setting the theoretical background for *Fisher's Discriminant Analysis* (FDA) (Fisher, 1936) and LDA. In the scientific literature, FDA is often referred to as LDA which creates confusion when trying to separate between the two methods. Even though FDA and LDA have different formulations, they have been shown to be equivalent (Ghojogh and Crowley, 2019). Finally, we present the SLDA model from a technical standpoint.

4.1 Fisher's Discriminant Analysis

For presenting FDA, we follow the derivation as in (Bishop, 2006). FDA, is a dimensionality reduction technique that (in the case of two classes) reduces a d-dimensional vector $\mathbf{x} \in \mathbb{R}^d$ to a single dimension $y \in \mathbb{R}$. It projects the data onto a line using the transformation:

$$y = \mathbf{w}^\top \mathbf{x}. \tag{4.1}$$

The intuition behind FDA is to find a one dimensional projection that maximizes class separation, hence, a measure of separability on one dimension is needed. For a class C_k with n_k data points, the class mean $\boldsymbol{\mu}_k$ and its corresponding projection μ_k are given by the equations:

$$\boldsymbol{\mu}_k = \frac{1}{n_k} \sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i \tag{4.2}$$

$$\mu_k = \mathbf{w}^\top \boldsymbol{\mu}_k. \tag{4.3}$$

For the two-class problem, a way of measuring class separation is to measure the distance between the projected means of the two classes. Consequently, if C_1, C_2 are the two classes, their distance on the projected space can be calculated from the equation:

$$|\mu_2 - \mu_1| = |\mathbf{w}^\top(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)|. \tag{4.4}$$

This may seem a good measure of separability at first, however, it is not optimal since it does not take into account the variance of the data. This can be better illustrated by figure 4.1. As we can see from the picture on the left, when only maximizing

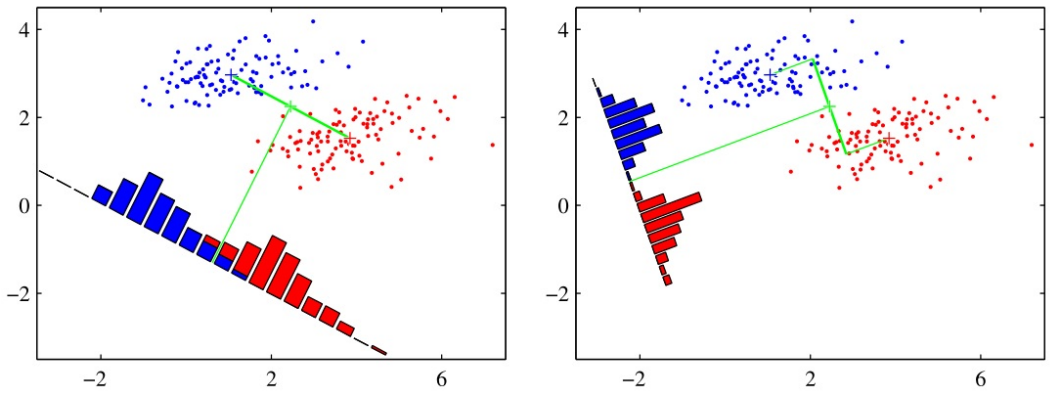

FIGURE 4.1: Projected data distribution when maximizing for the distance between the projected means of the two classes (left) versus when also minimizing for projected data variance (right). Source: (Bishop, 2006).

for class distance, there is a significant overlap of the projected data. In contrast, the image on the right shows a better solution that takes into account projected data variance where the projected data is completely separated even if the projected means are closer.

The solution to this problem came from Fisher, where he proposed a different optimization measure that accounts for both class distance and class variance. The *within-class variance* of the projected data for class C_k, is a measure of the variance that quantifies how much is the data distributed away from the class mean and is given by:

$$s_k^2 = \sum_{y_i \in C_k} (y_i - \mu_k)^2. \tag{4.5}$$

The total within-class variance when considering both classes, is defined to be the sum $s_1^2 + s_2^2$. Eventually, the optimization criterion as proposed by Fisher is defined as follows:

$$J(\mathbf{w}) = \frac{(\mu_1 - \mu_2)^2}{(s_1^2 + s_2^2)}. \tag{4.6}$$

By utilizing the concept of *between-class variance*, which expresses how much are the classes distributed away from each other, we can explicitly express J in terms of

$\mathbf{w}$ by defining the between-class covariance matrix $\mathbf{S}_B$:

$$\mathbf{S}_B = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^\top \tag{4.7}$$

and the within-class covariance matrix $\mathbf{S}_W$:

$$\mathbf{S}_W = \sum_{\mathbf{x}_i \in C_1} (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^\top + \sum_{\mathbf{x}_i \in C_2} (\mathbf{x}_i - \boldsymbol{\mu}_2)(\mathbf{x}_i - \boldsymbol{\mu}_2)^\top. \tag{4.8}$$

Using equations (4.1),(4.3),(4.5) we can rewrite Fisher's criterion (4.9) as:

$$J(\mathbf{w}) = \frac{\mathbf{w}^\top \mathbf{S}_B \mathbf{w}}{\mathbf{w}^\top \mathbf{S}_W \mathbf{w}}. \tag{4.9}$$

To find the optimal $\mathbf{w}^*$ we solve the equation:

$$\frac{\mathrm{d}J}{\mathrm{d}\mathbf{w}} = 0. \tag{4.10}$$

We should note that we only care about the direction of $\mathbf{w}^*$ and not its magnitude since we are looking for a projection direction. This can also be verified by equation 4.9 where any scaling of the weight vector $\mathbf{w}$ would be canceled out yielding the same result. After some algebraic manipulation and given the fact that we only care about the direction of $\mathbf{w}^*$ the optimal solution is found to be:

$$\mathbf{w}^* \propto \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1). \tag{4.11}$$

Equation (4.11) provides the direction of the projection axis such that the Fisher criterion (4.9) is maximized. Even though a dimensionality reduction method, FDA can also be used as a classifier when a threshold ($y \geq y_0$) is specified for the projected data. FDA can be generalized for the multi-class problem, the interested reader can refer to the detailed generalization derived by Bishop (2006).

4.2 Linear Discriminant Analysis

LDA is a supervised classification method that optimizes for the class posterior distribution. We will follow the formulation as presented in the tutorial of Ghojogh and Crowley (2019). Given a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, with $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}$, we would like to produce a classification method based on these instances. First, we consider the binary case where only two classes are presented. We make the assumption that the two classes follow a normal distribution with identical covariance:

$$\begin{aligned} \mathbf{x} \in C_1 &\sim N(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) \\ \mathbf{x} \in C_2 &\sim N(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}). \end{aligned} \tag{4.12}$$

We can find the classification boundary by equating the posterior probabilities:

$$P(\mathbf{x} \in C_1 | \mathbf{x}) = P(\mathbf{x} \in C_2 | \mathbf{x}). \tag{4.13}$$

Using Bayes theorem, equation (4.13) becomes:

$$\frac{P(\mathbf{x}|\mathbf{x} \in C_1)}{P(\mathbf{x})} = \frac{P(\mathbf{x}|\mathbf{x} \in C_2)}{P(\mathbf{x})} \Rightarrow$$
$$\frac{f_1(\mathbf{x})\pi_1}{\sum_{i=1}^{|C|} P(\mathbf{x}|\mathbf{x} \in C_i)\pi_i} = \frac{f_2(\mathbf{x})\pi_2}{\sum_{i=1}^{|C|} P(\mathbf{x}|\mathbf{x} \in C_i)\pi_i} \Rightarrow$$
$$f_1(\mathbf{x})\pi_1 = f_2(\mathbf{x})\pi_2 \tag{4.14}$$

where π_i the class prior probability and $f_i(\mathbf{x})$ the probability density function. Following, by using the the multivariate Gaussian probability density function:

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d|\boldsymbol{\Sigma}|}} exp\left(-\frac{(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}{2}\right) \tag{4.15}$$

and doing some algebraic manipulations omitted here for brevity, we arrive at the equation for the classes boundary:

$$2(\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1))^\top \mathbf{x} + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) + 2\ln(\frac{\pi_2}{\pi_1}) = 0. \tag{4.16}$$

The boundary equation is of the form $\mathbf{a}\mathbf{x} + b = 0$ which represents the equation of a line, hence LDA is a linear classifier. We can use the right side of equation (4.16) to construct a classifier $\delta(\mathbf{x})$ such that $\mathbf{x} \in C_1$ if $\delta(\mathbf{x}) < 0$ and $\mathbf{x} \in C_2$ if $\delta(\mathbf{x}) > 0$.

The generalization for multiple classes is straightforward but the derivation will be omitted here for brevity. A detailed derivation is presented in (Ghojogh and Crowley, 2019). It can be shown that the classifier function in this case can be written as:

$$\delta_k(\mathbf{x}) = \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\mathbf{x} - \frac{1}{2}\boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \ln(\pi_k). \tag{4.17}$$

Consequently, equation (4.17) can be used to classify an instance by maximizing the posterior of a class, when assuming a normal distribution of the data and a single shared covariance matrix among the classes. The covariance matrix and the prior class probability can be estimated as:

$$\boldsymbol{\Sigma} = \frac{\sum_{k=1}^{|C|} n_k \boldsymbol{\Sigma}_k}{n} \tag{4.18}$$

$$\pi_k = \frac{n_k}{n}. \tag{4.19}$$

Finally, in order to classify an instance $\boldsymbol{x}$ we use equation (4.17) and choose the class that gives the maximum value:

$$C = \arg\max_k \delta_k(\mathbf{x}). \tag{4.20}$$

4.3 Deep Streaming LDA

Hayes and Kanan (2020) recently proposed *deep Streaming LDA* (SLDA), a novel method for resisting catastrophic forgetting by updating the output layer of a pre-trained CNN through LDA. More specifically, they used Incremental Linear Discriminant Analysis (ILDA) (Pang et al., 2005), a type of LDA adjusted for the online learning setting. ILDA differs from LDA in the sense that it can be applied to a non-static environment by updating the classes' means and the common covariance matrix every time new data is obtained.

The CNN used for the SLDA model can be thought of as being composed of two main parts, the pre-trained feature extractor $G(\cdot)$, and the classification layer $F(\cdot)$ which consists of the last fully connected layer. Given an input image $\mathbf{x}$ the output is obtained as $y = F(G(\mathbf{x}))$. Due to the fact that early layers of a CNN learn generic transferable features (Yosinski et al., 2014), the SLDA model keeps the feature extractor part of the network frozen and trains only the output layer $F(\cdot)$. The feature extractor part is trained during an initial base initialization phase, a common approach in incremental learning literature (Castro et al., 2018; Rebuffi et al., 2017). Given a feature representation $\mathbf{z} = G(\mathbf{x}) \in \mathbb{R}^d$ extracted from the frozen part $G(\cdot)$ of the CNN, the output is calculated as:

$$F(G(\mathbf{x})) = \mathbf{W}\mathbf{z} + \mathbf{b} \tag{4.21}$$

where weight matrix $\mathbf{W} \in \mathbb{R}^{K \times d}$ and bias vector $\mathbf{b} \in \mathbb{R}^K$ are being updated online, with K the total number of classes.

In order to update the weight matrix and the bias, SLDA stores a mean feature vector $\boldsymbol{\mu}_k$ with an associated count c_k for each class k an a single shared covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$. The means and associated counts are updated in a streaming manner when a new feature instance $(\mathbf{z}_t, y_k)$ arrives at time t:

$$\boldsymbol{\mu}_{k,t+1} = \frac{c_{k,t}\boldsymbol{\mu}_{k,t} + \mathbf{z}_t}{c_{k,t} + 1} \tag{4.22}$$

$$c_{k,t+1} = c_{k,t} + 1. \tag{4.23}$$

To make predictions using equation (4.21), the weight matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$ are updated by the following equations:

$$\mathbf{w}_k = \Lambda\boldsymbol{\mu}_k \tag{4.24}$$

$$b_k = -\frac{1}{2}(\boldsymbol{\mu}_k\Lambda\boldsymbol{\mu}_k) \tag{4.25}$$

where $\mathbf{w}_k$ the k^{th} row of the weight matrix, b_k the k^{th} element of $\mathbf{b}$, and Λ the precision matrix (inverse of covariance matrix). In the experiments by Hayes and Kanan (2020) both cases for plastic and non-plastic covariance matrix were tested,

where the update for the former case was based on (Dasgupta and Hsu, 2007) and is given by the equation:

$$\boldsymbol{\Sigma}_{t+1} = \frac{t\boldsymbol{\Sigma}_t + \Delta_t}{t+1} \tag{4.26}$$

with Δ_t

$$\Delta_t = \frac{t(\mathbf{z}_t - \boldsymbol{\mu}_{k,t})(\mathbf{z}_t - \boldsymbol{\mu}_{k,t})^\top}{t+1}. \tag{4.27}$$

It is easy to see why SLDA is resistant to catastrophic forgetting. The classes' means used for the prediction are independent, hence learning to classify instances from new classes does not interfere with previously obtained knowledge. Furthermore, even if the covariance matrix changes with time, in the worst case this can result in gradual forgetting (Hayes and Kanan, 2020).

Chapter 5

Combining SLDA with Rehearsal

The SLDA model (Hayes and Kanan, 2020) constitutes a novel approach to fighting catastrophic forgetting but it relies on a frozen feature extractor making it a non-viable solution to continual learning. In this chapter we describe our proposed model which is an extension of SLDA that uses rehearsal to train its feature extractor. However, training the feature extractor is ultimately faced with the problem of drifting features. The stored mean feature vectors used to update the output layer through LDA suffer an aging effect as the weights of the feature extractor are updated during training. We discuss this issue in detail and present different ways of addressing it.

5.1 Shortcomings of SLDA

The reliance of SLDA on a frozen feature extractor limits the performance of the model to the feature representations learned during the pre-training phase. This restriction does not allow the SLDA algorithm to scale on tasks whose features differ significantly from the learned features. Furthermore, even when dealing with classes with similar features to those in the pre-training phase, the model could still improve on performance by learning new class-specific features.

Evidence for improvement of the SLDA model by learning better representations is presented in the original paper itself. For example, in figure 5.1 obtained from the original paper, we can see how the accuracy of the SLDA model is affected by increasing the number of base initialization classes. The figure clearly demonstrates how the SLDA model benefits from better representations learned during the base initialization phase. It is interesting to note that already from 100 classes the model outperforms End-to-End (Castro et al., 2018) and iCaRL (Rebuffi et al., 2017) models.

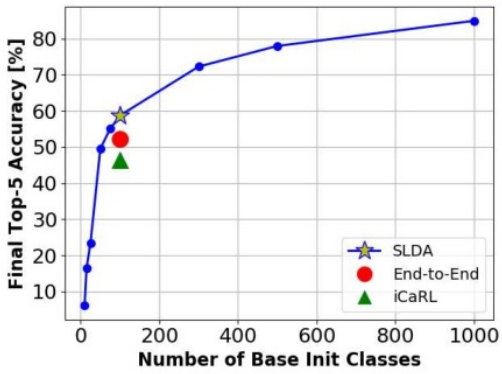

FIGURE 5.1: SLDA model final top-5 accuracy on ImageNet as a function of the number of base initialization classes. The model is compared with two other continual learning models at 100 classes. Source: (Hayes and Kanan, 2020).

5.2 Replay-SLDA

Based on the ability of the SLDA model to improve upon learning better feature representations, we chose to investigate whether it could benefit from incorporating a rehearsal algorithm. Making such a combination work would alleviate the SLDA model's limitation of having a static feature extractor while maintaining its resistance to catastrophic forgetting by updating the output layer through LDA as previously described in section 4.3. Our goal was to determine whether the two algorithms (rehearsal and SLDA) could be combined and how the resulting model would compare to each one of them individually. We refer to our proposed combined model as *Replay-SLDA*.

5.2.1 Problem Formulation

Here we take a moment to make a compact description of our entire setting for clarity. We consider the online continual learning setting, which assumes a possibly infinite stream of non-i.i.d. data with no clear task boundaries. At each given time the learner has access only to the data of the batch obtained from the stream and can not access previously seen instances unless explicitly stored. We use the data to train the network through rehearsal-based training while updating the classification layer through LDA.

5.2.2 Main Challenges

One challenge that arises when trying to combine rehearsal with SLDA is to decide whether to use the network's original classification layer or the SLDA model's output layer during training. We found that using the SLDA model's layer results in numerical instability during training. This can be attributed to the large weight

values in the output layer compared to the values of the weights of the pre-trained layers. We found that the SLDA model's output layer weights as calculated from (4.24) can have values of two to three orders of magnitude greater than the rest of the layers. This can result in gradient updates of large magnitude which explains numerical instability. To overcome this, we used the network's original fully connected layer to train the network through gradient descent while simultaneously training a separate classification layer through LDA. This layer is the final layer used to make predictions for the model.

The main difficulty concerns the *drift* of stored mean feature vectors utilized for the update of the output layer. Suppose that at a specific time the SLDA model has seen a number of samples from the stream and stored the corresponding mean feature vectors $\boldsymbol{\mu}_k$ for each seen class k. Now, if we were to update the weights of the feature extractor using a replay method the new feature space would be different from the one where the mean feature vectors were originally obtained. For a given instance $\mathbf{x}$ and a feature extractor $G(\mathbf{W}, \mathbf{x})$ we would have:

$$G(\mathbf{W}, \mathbf{x}) \neq G(\mathbf{W}', \mathbf{x}) \tag{5.1}$$

where $\mathbf{W}, \mathbf{W}'$ is the feature extractor's weight matrix before and after the update respectively. For a single update, the difference might be negligible but as the number of updates increases, the extracted features at a given time might correspond to a completely different feature space than the original one. This way, the stored mean feature vectors, which are continuously updated, drift further from the real mean feature representations at a specific time. This phenomenon is detrimental to the accuracy of the model as it interferes with the performance of LDA.

It becomes apparent that in order for SLDA to be successfully combined with a rehearsal algorithm, the problem of drift needs to be addressed. This is the main obstacle that undermines the joint use of the two algorithms. In the subsections that follow we discuss different ways for approaching this problem.

5.2.3 Correcting for Drift

A way of resolving drift is to try to transform the stored mean feature vectors to their corresponding representations in the new feature space after every update of the network. Approaches that fall under this category essentially focus on correcting for drift. This is not an easy task and it becomes increasingly difficult when dealing with large streams containing a high number of classes even if the transformation method is highly accurate.

An approach for correcting for drift is presented in (Iscen et al., 2020) where the authors faced the same problem but in a different setting. In their task, drift was introduced by implementing rehearsal as feature replay, a method that stores and replays feature representations from a specific layer instead of original instances. However, the features stored for replay also suffer from drift as the network updates its weights during training. To resolve this, they used a feature adaptation method by training a separate network after each weight update of the original network to transform

the feature vectors to their representations in the new feature space. To illustrate this, suppose that at a specific time a batch of instances $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)^\top$ is obtained. Before the network's update, these instances correspond to the feature representations $\mathbf{Z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n)^\top$ while after the update the new representations are $\mathbf{Z}' = (\mathbf{z}'_1, \mathbf{z}'_2, \ldots, \mathbf{z}'_n)^\top$. To obtain the feature mapping between the two spaces, the authors propose to train a separate network using $\mathbf{Z}$ as input and $\mathbf{Z}'$ as the target. This process should be repeated every time the main network updates its weights. For a more detailed presentation of this method, the reader can look at the original paper.

The feature adaptation method by training a mapping network is a powerful technique but it dramatically increases computational cost. For this reason, we chose not to adopt it for managing drift in our setting. A simpler approach (but not as accurate) is to calculate the differences between the corresponding vectors in $\mathbf{Z}, \mathbf{Z}'$ and use the mean difference to update the mean feature vectors in a way similar to the one described by Yu et al. (2020). The idea is straightforward, if $\boldsymbol{\mu}_k$ is the mean feature vector of class k, before the update and $\boldsymbol{\mu}'_k$ after the update, then the new value of the mean vector can be calculated as:

$$\boldsymbol{\mu}'_k = \boldsymbol{\mu}_k + \Delta\boldsymbol{\mu}_k. \tag{5.2}$$

Since we don't know the value of $\Delta\boldsymbol{\mu}_k$, we approximate it from the values of the obtained batch at a given time. Let the obtained batch be $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)^\top$, then the average change of the feature vectors in the batch can be calculated as follows:

$$\Delta\mathbf{z} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{z}'_i - \mathbf{z}_i). \tag{5.3}$$

By making the assumption that $\Delta\boldsymbol{\mu}_k \approx \Delta\mathbf{z}$ for $k = 1, \ldots, K$, where K the total number of observed classes, the equation (5.2) becomes:

$$\boldsymbol{\mu}'_k = \boldsymbol{\mu}_k + \Delta\mathbf{z}. \tag{5.4}$$

This is not an exact method, but it maintains a low computational cost while handling drift to an extent. Our implementation of Replay-SLDA with drift correction is described in algorithm 1.

Finally, we would like to mention another approach that if implemented could completely correct for feature drift. We abandoned this method because we found it impossible to develop in practice as will be explained. Nevertheless, we will briefly present here the mathematical idea behind it as it may serve as inspiration for future work. The idea is to calculate the exact differential $\mathrm{d}\boldsymbol{\mu}_k$ for each mean feature vector $\boldsymbol{\mu}_k$, provided that changes of the network's weights are sufficiently small ($\Delta w_i \approx \mathrm{d}w_i$), and use it to correct for drift:

$$\mathrm{d}\boldsymbol{\mu}_k = \frac{\partial\boldsymbol{\mu}_k}{\partial w_1}\mathrm{d}w_1 + \frac{\partial\boldsymbol{\mu}_k}{\partial w_2}\mathrm{d}w_2 + \cdots + \frac{\partial\boldsymbol{\mu}_k}{\partial w_N}\mathrm{d}w_N \tag{5.5}$$

$$\boldsymbol{\mu}'_k = \boldsymbol{\mu}_k + \mathrm{d}\boldsymbol{\mu}_k. \tag{5.6}$$

The terms $\mathrm{d}w_i$ can be easily calculated since we have access to the values of the network weights both before and after the update. The difficulty arises when trying to calculate the partial derivatives $\frac{\partial \boldsymbol{\mu}_k}{\partial w_i}$. In order to do so using automatic differentiation, the computational graphs of all previously seen instances are needed which can quickly explode memory requirements. Consequently, despite being an almost exact correction for drift, the technical requirements that arise constitute the use of this method prohibiting.

Algorithm 1 Replay-SLDA with drift correction

Input: (model with feature extractor $G(\cdot)$, fc classification layer $F(\cdot)$ and LDA classification layer $L(\cdot)$), replay batch size b, loss $\ell(\cdot)$, covariance matrix $\boldsymbol{\Sigma}$

while stream not empty **do**
  receive batch $(\mathbf{X}_t, \mathbf{y}_t)$ from stream
  sample replay batch $(\mathbf{X}_r, \mathbf{y}_r)$ of size b from memory
  combine batches to a single augmented batch $(\mathbf{X}_a, \mathbf{y}_a) \leftarrow \left(\begin{pmatrix}\mathbf{X}_t \\ \mathbf{X}_r\end{pmatrix}, \begin{pmatrix}\mathbf{y}_t \\ \mathbf{y}_r\end{pmatrix}\right)$
  get feature vectors prior to network update: $\mathbf{Z}_t = G(\mathbf{X}_t)$
  update mean feature vectors: $\boldsymbol{\mu}_k$
  update classification layer $L(\cdot)$ according to $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}$
  predict outputs: $\hat{\mathbf{y}}_a = F(G(\mathbf{X}_a))$
  update $G(\cdot), F(\cdot)$ according to loss $\ell(\hat{\mathbf{y}}_a, \mathbf{y}_a)$
  get new feature vectors: $\mathbf{Z}'_t = G'(\mathbf{X}_t)$
  calculate mean batch drift: $\Delta\mathbf{z}_t = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{z}'_{ti} - \mathbf{z}_{ti})$
  transform mean feature vectors: $\boldsymbol{\mu}'_k = \boldsymbol{\mu}_k + \Delta\mathbf{z}_t$
  sample instances from $(\mathbf{X}_t, \mathbf{y}_t)$ and store to memory

5.2.4 Avoiding Drift

In the previous subsection, all the different methods presented focused on correcting for drift by transforming the stored mean feature vectors. Here, we will describe an approach that tackles the problem from a completely different perspective. When combining rehearsal with SLDA, there is a way to directly avoid dealing with drifting features. The trade-off is to use approximations of the classes' mean feature vectors instead of the real ones. However, as we will see in the next chapter, these approximations can successfully represent the real means in most practical cases.

Since Replay-SLDA already uses a memory buffer to store instances for replay, we found that a better way of dealing with drift is to exploit the buffer and use the stored samples to calculate the mean feature vectors at the end of training. It turns out that this is a very effective approach that directly avoids drift. This is implemented in practice by training a network on the stream using only rehearsal-based training. At any point where the combined model is needed, an SLDA model is created with

the trained network as a feature extractor base. The output layer is updated through LDA by training it on the samples of the memory buffer. This is done by forwarding the stored instances of the buffer through the feature extractor part of the network, calculating the approximations to the mean feature vectors of each class $\tilde{\boldsymbol{\mu}}_k$, and using them to update the output layer of the SLDA model as described by equations (4.24, 4.25). The result is an SLDA model with a feature extractor that has been trained on the stream and an output layer trained on a subset of the stream. This method has an almost identical computational cost to the chosen rehearsal scheme, in contrast to the corrective methods described previously. The only added cost comes from forwarding the memory through the network, calculating the mean feature vectors, and updating the output layer, which is negligible for typical memory sizes used by rehearsal schemes. It is clear that the mean feature vectors approximations $\tilde{\boldsymbol{\mu}}_k$, and hence the performance of the model, depends on the size of the memory buffer. If m is the memory size and s the stream size, then it is true that:

$$\tilde{\boldsymbol{\mu}}_k \to \boldsymbol{\mu}_k \text{ as } m \to s. \tag{5.7}$$

It turns out, that for typical memory sizes used by rehearsal models, the mean feature vectors are successfully approximated. This means that Replay-SLDA does not require more memory than a rehearsal model, considering that the stored mean feature vectors are of negligible size compared to the memory buffer.

Our model benefits from larger buffer sizes through two mechanisms. First, a larger buffer means that the feature extractor will be trained on more samples through replay, thus learning better representations. Second, with a larger buffer, the final approximations for the mean feature vectors will be closer to the real ones resulting in better training of the classification layer through LDA. An overview of the algorithm is shown below:

Algorithm 2 Replay-SLDA by avoiding drift

Input: model $f(\cdot)$, replay batch size b, loss $\ell(\cdot)$, covariance matrix $\boldsymbol{\Sigma}$

while stream not empty **do**
  receive batch $(\mathbf{X}_t, \mathbf{y}_t)$ from stream
  sample replay batch $(\mathbf{X}_r, \mathbf{y}_r)$ of size b from memory
  combine batches to a single augmented batch: $(\mathbf{X}_a, \mathbf{y}_a) \leftarrow \left(\begin{pmatrix}\mathbf{X}_t \\ \mathbf{X}_r\end{pmatrix}, \begin{pmatrix}\mathbf{y}_t \\ \mathbf{y}_r\end{pmatrix}\right)$
  predict outputs: $\hat{\mathbf{y}}_a = f(\mathbf{X}_a)$
  update model according to loss: $\ell(\hat{\mathbf{y}}_a, \mathbf{y}_a)$
  sample instances from $(\mathbf{X}_t, \mathbf{y}_t)$ and store to memory
create SLDA model with feature extractor $G(\cdot)$ from $f(\cdot)$
get feature vectors from memory instances: $\mathbf{Z}_m = G(\mathbf{X}_m)$
calculate mean feature vectors $\tilde{\boldsymbol{\mu}}_k$ from $\mathbf{Z}_m$
update the SLDA model's output layer according to $\tilde{\boldsymbol{\mu}}_k, \boldsymbol{\Sigma}$

Chapter 6

Experimental Work

This chapter contains the evaluation of our model for three different datasets. In particular, we present how Replay-SLDA compares to its components (SLDA and rehearsal), which was the ultimate goal of the thesis. Furthermore, we show how our final implementation Replay-SLDA which avoids drift compares to our initial one which corrects for drift as well as to the ideal Replay-SLDA model utilizing the real mean features of the stream. In the paragraphs that follow we will refer to our first implementation of Replay-SLDA as Replay-SLDA (v1) while to our final implementation as Replay-SLDA (v2).

6.1 Datasets

Here we make a brief description of all relevant datasets. For the experiments, we use a CNN model pre-trained on the ImageNet ILSVRC-2012 dataset (Russakovsky et al., 2015) which is the most popular subset of the ImageNet database (Deng et al., 2009). Its training set consists of about 1.28 million annotated images of 1000 classes. It is common to refer to ImageNet ILSVRC-2012 dataset as "ImageNet", even though this term strictly refers to the ImageNet database. For brevity, in the rest of the text, we will also refer to this dataset simply as ImageNet.

For the training and evaluation phases of our experiments, we use the CIFAR-10, CIFAR-100, (Krizhevsky and Hinton, 2009) and the colorectal histology (Kather et al., 2016) datasets. CIFAR-10 consists of 60000 32×32 RGB images belonging to 10 different classes, with the training set reserving 50000 of those images and the test set the rest. CIFAR-100 is similar to CIFAR-10 except that it contains 100 classes. This means that it reserves 500 images per class for its training set in contrast to CIFAR-10 which uses 5000 images. The fewer instances per class make CIFAR-100 a more challenging dataset. Furthermore, from the perspective of continual learning algorithms, CIFAR-100 poses an even greater challenge due to its high number of classes that must be learned sequentially. This means that the performance of a continual learning algorithm on CIFAR-100 demonstrates to what extent the algorithm can handle catastrophic forgetting over a long sequence of tasks.

Finally, the colorectal histology dataset, which we will refer to as CRH, contains 5000 150×150 RGB histological images from 8 different types of colorectal cancer. We split it into a training set and test set using an 80-20 split, resulting in a training set of 4000 images. This corresponds to 500 images per class, similar to CIFAR-100, though with a much smaller number of classes.

6.2 Implementation Details

For all the experiments, we use a ResNet-18 (He et al., 2016) pre-trained on ImageNet. The choice of CNN architecture was not of particular importance for the investigation of this thesis hence we chose ResNet-18 to comply with the model used in the original SLDA paper (Hayes and Kanan, 2020). Furthermore, we use reservoir sampling as a memory population algorithm due to its simplicity and effectiveness in populating the memory with a representative subset of the stream. For replaying samples from memory we use random replay. The size of the incoming batch is set to $b = 10$ while the same size is chosen for the replay batch. The training of the network is performed through gradient descent with a learning rate of 0.001. Finally, we use accuracy as the evaluation metric which represents the fraction of the total predictions that are correct.

It should be noted that for all experiments we keep the running statistics of the batch normalization layers to the values of the pre-trained model and do not allow them to be updated during training using batch statistics. However, we allow for the parameters gamma and beta (Ioffe and Szegedy, 2015) to be updated through gradient descent. This choice is based on the fact that in continual learning the classes are presented in a sequential way to the learner and the running statistics of the batch-norm layers at the end of training correspond only to the last seen class. This leads to deterioration of performance which we validated experimentally.

For the SLDA model, we use a covariance matrix initialized to ones like in some of the experiments of Hayes and Kanan (2020). We keep the covariance matrix static since in the original paper continually updating it offered only a marginal improvement on performance. Furthermore, for every experiment, we sort the given dataset by grouping instances of the same class together in order to create a non-i.i.d stream of data. Before each run, we permute the order of classes to be presented to the model as well as the distribution of samples within each class. This ensures that our experiments do not involve the same ordered stream. Regarding the conventional non-sequential training of the CNN, this was conducted for a total of three epochs. Finally, we note that the presented accuracy for all experiments refers to the averaged values over five runs.

6.3 Experimental Results

6.3.1 Results on CIFAR-10

Here we present the performance of our final implementation of Replay-SLDA (Replay-SLDA (v2)) on CIFAR-10. In figure 6.1 we compare the performance of our model to SLDA, Rehearsal, and conventional non-sequential training (Offline). The final

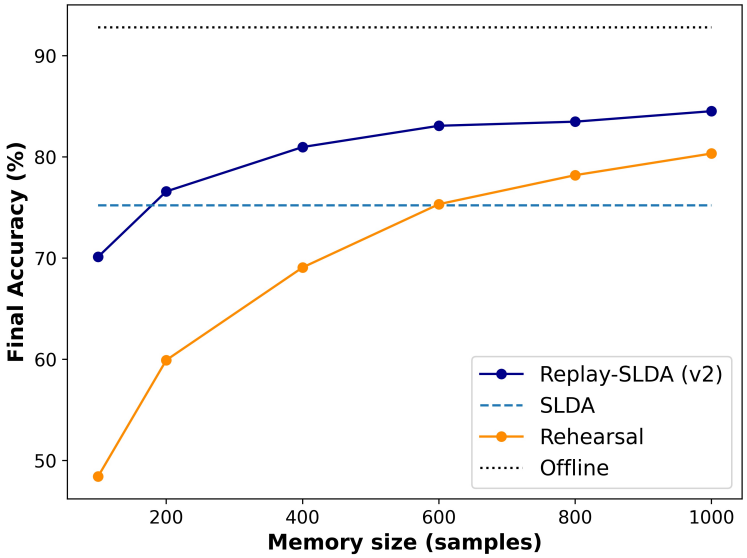

FIGURE 6.1: Comparing the performance of Replay-SLDA to Rehearsal, SLDA, and Offline models on CIFAR-10 for different memory sizes. Each point corresponds to the average value over 5 runs.

accuracy on CIFAR-10 is presented as a function of the memory size measured in number of stored instances. Since SLDA and offline training do not use a memory buffer, they are represented as horizontal lines. Replay-SLDA (v2) outperforms SLDA already from a memory size of 200 samples, corresponding roughly to 20 samples per class, with its accuracy increasing with memory size. When compared to Rehearsal we can see that our model consistently outperforms it for all chosen memory sizes with the accuracy gap between them being inversely associated with the memory size. This is not the case for the SLDA model which has constant performance and hence the accuracy gap increases with memory size. While SLDA has a constant accuracy of 75.2%, Replay-SLDA (v2) reaches an accuracy of 84.5% at 1000 samples. This is almost an extra 10% gain in final accuracy. Furthermore, we see that while Rehearsal outperforms the SLDA model when the memory size gets over 600 samples Replay-SLDA (v2) needs less than a third of that memory to achieve this. Another interesting observation is at the intersection point of the curves of the SLDA and the Rehearsal models at 600 samples. At that memory size, the two models have about the same accuracy of 75.2%. At the same memory, the Replay-SLDA (v2) model has

an accuracy of 83% which is an 8% gain in final accuracy. It's worth noting that the peak accuracy of Replay-SLDA (v2) at 1000 samples is only 8% below the accuracy of 92.8% obtained through offline training.

Besides the final accuracy, we also investigated how the accuracy of our model compares with Rehearsal and SLDA after having seen different percentages of the stream during training. These results are presented in figure 6.2 for three different chosen memory sizes. Replay-SLDA (v2) consistently outperforms the other two

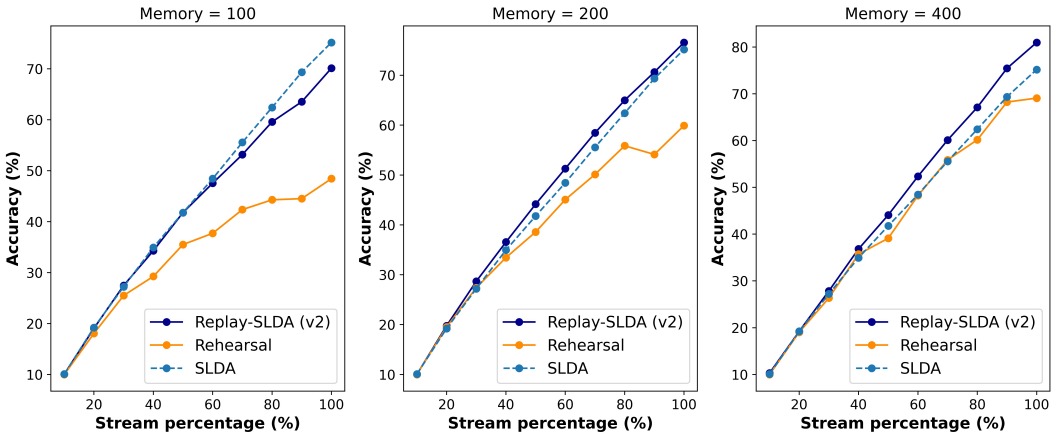

FIGURE 6.2: Evaluation at different percentages of the CIFAR-10 stream for different memory sizes. Each point corresponds to the average value over 5 runs.

models at different percentages of the stream with the exception when the memory has a size of 100 samples. An interesting remark is that for the memory sizes of 200 and 400 samples, Replay-SLDA (v2) outperforms the other models at a percentage of about 30% of the stream and above. This can be understood by considering two concepts. First, at the very early stages of the stream (10% or less), only one class has been presented to the models, hence all of them learn to predict this class resulting in the same final accuracy. Second, Replay-SLDA (v2) improves on SLDA by learning better feature representations. At small stream percentages, the model has not seen enough classes yet and therefore the feature extractor has not learned any new representations, making its performance similar to SLDA. The difference in accuracy becomes more evident as the model sees a larger percentage of the stream at which point the feature extractor has improved over the frozen extractor of the SLDA model. Similar reasoning can be followed when comparing Replay-SLDA (v2) to Rehearsal. Replay-SLDA (v2) improves over Rehearsal by updating the output layer through LDA which in turn depends on the mean feature representations of each class. Therefore, for Replay-SLDA (v2) to outperform Rehearsal it needs to have seen a certain number of classes first.

We also investigated how Replay-SLDA (v2) compares to our original implementation of the model (Replay-SLDA (v1)) as well as to the ideal model utilizing the real mean feature vectors of the stream. We repeat here that while Replay-SLDA

(v2) avoids the occurrence drift, Replay-SLDA (v1) utilizes a transformation with the goal of correcting it. We did this by transforming the stored feature representations (equation 5.4) to the new feature space after every update of the network's weights using the approximation $\Delta\boldsymbol{\mu}_k \approx \Delta\mathbf{z}$. Regarding the performance of the ideal model, we estimate it by following two separate training stages. During the first stage, we train a rehearsal model on the entire stream. In the second stage, we create an SLDA model using the trained CNN as a feature extractor and loop again over the stream to train the output layer of the SLDA model. This gives the performance of a Replay-SLDA model that has access to the true mean feature vectors of the stream at the end of training. The ideal model can not be implemented in practice because it requires two passes over the entire stream which is unrealistic for a continual learning setting. The observant reader might have noticed that the implementation of Replay-SLDA (v2) is similar to the ideal model with the difference that during the second training stage we train the model on the buffer samples instead of the entire stream. Finally, as Naive model, we refer to the combined model in which no action is taken to address the drift of stored features. The algorithm for this model would be similar to algorithm 1 with the drift correction steps removed.

The results are presented in figure 6.3. As can be seen, the Naive model is far

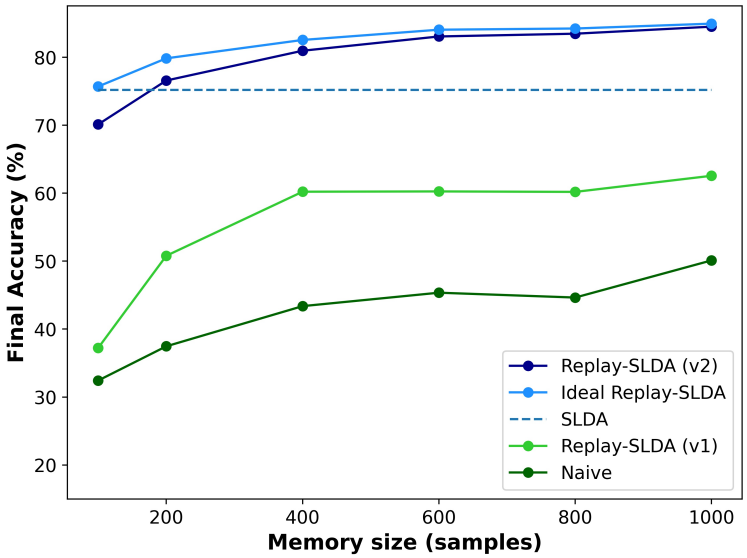

FIGURE 6.3: Comparison of different implementations of Replay-SLDA on CIFAR-10. Each point corresponds to the average value over 5 runs.

from optimal and maintains poor performance, significantly lower than the original SLDA model. This result was expected considering that after multiple updates of the network the stored mean feature vectors drift away from the current representational space undermining the performance of LDA. Regarding Replay-SLDA (v1), we can see that even though it greatly improves over the Naive model, it is still not able to outperform the SLDA model. The most promising result is represented by the two

top curves. It is encouraging to observe that the accuracy the Replay-SLDA (v2) model closely approximates the accuracy of the ideal model for memory sizes of 400 samples and above. This implies that already from 400 samples, which corresponds on average to 40 samples per class for CIFAR-10, the mean feature vectors of the stream are successfully approximated. As the memory size increases it is expected that the final accuracy of the Replay-SLDA (v2) model will be almost identical to the ideal case but the equality holds only for a memory of the size of the stream. However, a memory of this size would be redundant since with only 1000 samples (or about 100 samples per class), the difference in final accuracy between our model and the ideal model is only 0.45%.

6.3.2 Results on CIFAR-100

Here we present the evaluation of our model on CIFAR-100. It is considered a challenging dataset for continual learning algorithms due to the high number of classes needed to be learned sequentially and also due to the limited number of instances per class compared to CIFAR-10. Again, we compare the performance of Replay-SLDA (v2) with SLDA and Rehearsal for different memory sizes. The sizes chosen are greater than the ones for CIFAR-10 due to the higher number of classes but they remain constrained to typical memory requirements for such tasks. The results are shown in figure 6.4.

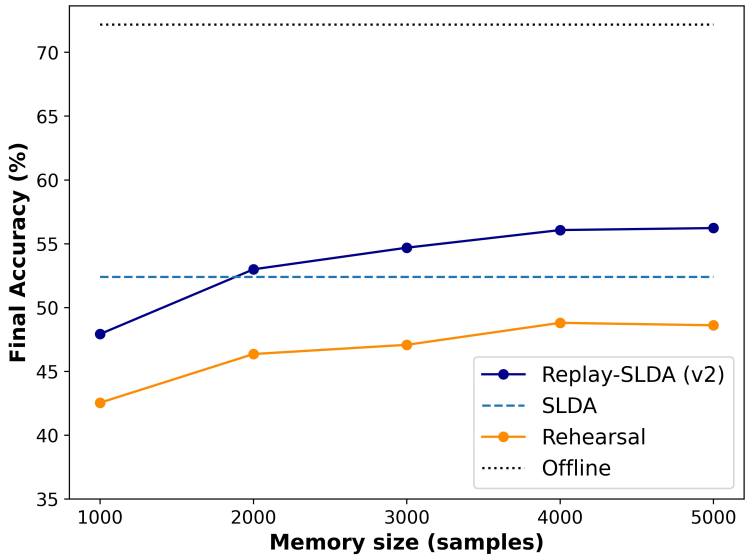

FIGURE 6.4: Comparing the performance of Replay-SLDA to Rehearsal, SLDA, and Offline models on CIFAR-100 for different memory sizes. Each point corresponds to the average value over 5 runs.

Replay-SLDA (v2) outperforms Rehearsal on all chosen memory sizes and SLDA for sizes of 2000 samples (about 20 samples per class) and above. Furthermore, at 5000

samples it is able to obtain a final accuracy of 56.2% which is 3.8% above the SLDA model. This difference is not as large as observed in CIFAR-10 but given that this is a more challenging dataset, an improvement of about 4% is significant. Learning sequentially on a stream of 100 different classes is particularly challenging. This is why we observe a larger gap here between Replay-SLDA and offline training than in CIFAR-10. We could possibly have obtained higher final accuracy by increasing the size of the replay batch used for training. Here we set it to $b = 10$ similarly to CIFAR-10 experiments. While in CIFAR-10 a replay batch of this size could mean that on average samples from all classes are seen during replay this is not true for CIFAR-100.

When comparing the final performance of the models after having seen different percentages of the stream (figure 6.5) we make similar observations as in CIFAR-10. At higher memory sizes (larger than or equal to 2000) Replay-SLDA starts to

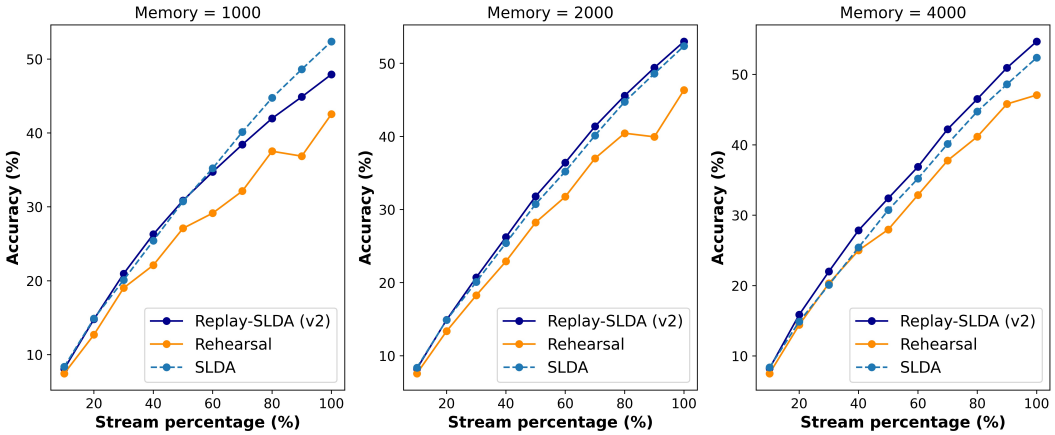

FIGURE 6.5: Evaluation at different percentages of the CIFAR-100 stream for different memory sizes. Each point corresponds to the average value over 5 runs.

outperform SLDA after having seen about 20% - 30% of the stream for reasons already explained. Perhaps an unexpected observation is that for a memory of 1000 samples our model has slightly higher accuracy than SLDA at the initial stages of the stream before it is outperformed. We can understand this if we think that in the initial stages of the stream for a memory of 1000 samples there are enough samples to represent the classes seen so far. This means that the network can learn better representations and have a classification layer comparable to SLDA. As a larger part of the stream is seen the number of observed classes increases leading to a lower number of instances per class in memory. This limits the performance of both rehearsal and LDA which uses the mean feature vectors obtained from the memory instances. As a result SLDA eventually outperforms our model even without learning new representations by being able to train a better classification layer.

Finally, the comparison of different implementations of Replay-SLDA is presented in figure 6.6. As we can see, Replay-SLDA (v1) performs poorly and has comparable

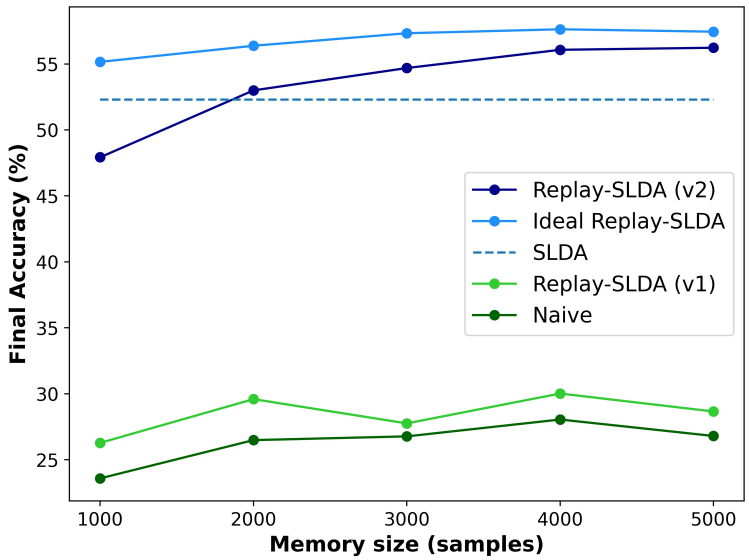

FIGURE 6.6: Comparing different implementations of Replay-SLDA on CIFAR-100. Each point corresponds to the average value over 5 runs.

accuracy to the naive implementation. Our correction for drift provides an improvement over the Naive model but this is significantly lower than the improvement observed in CIFAR-10. Given the large number of classes in CIFAR-100, it was expected that our method for transforming all mean feature vectors using the same approximation from the current batch would be even less successful in addressing drift. Finally, looking at the performance of the ideal model we notice that Replay-SLDA (v2) closely approximates it for memory sizes $m \geq 4000$. The ideal model outperforms SLDA already from a memory size of 1000 samples, showing that even with only 10 samples per class it has learned better feature representations. This is contrary to CIFAR-10 where the ideal model had the same accuracy as the SLDA model at the corresponding memory size (10 samples per class). However, even with the improved feature representations at this initial size, our model does not outperform the SLDA model due to the inadequately approximated mean feature vectors.

6.3.3 Results on CRH

In this subsection, we conclude the presentation of our experimental results by presenting the final evaluation of our model on the medical colorectal histology dataset (CRH) containing images from 8 different classes. Due to the qualitative similarity of this dataset's results to the results on CIFAR-10 and CIFAR-100 we make a more brief description of the results.

In figure 6.7, we can see that already from a memory size of 100 samples (about 12-13 samples per class), Replay-SLDA (v2) reaches a final accuracy of 80.5% which is higher than the accuracy of 79.3% achieved by the SLDA. The final accuracy of

our model continues to increase with memory size but seems to quickly stagnate to an accuracy of 83.0% at a memory size of 400 samples and beyond. This is close to a 4% higher final accuracy than SLDA. We hypothesize that the reason for this quick stagnation in performance is attributed to the fact that the pre-trained CNN is already able to extract accurate feature representations from this dataset. This can be verified by looking at the accuracy of the SLDA model (79.3%), which indicates that the representations learned by the CNN in its pre-training phase can also be used to achieve high discrimination of this dataset's classes. Our model's peak performance is 7.3% below the accuracy of the offline model, which is encouraging given that it is a sequential learning algorithm. Furthermore, it also consistently outperforms the Rehearsal model which itself fails to outperform SLDA for any of the tested memory sizes. Finally, we note that in this dataset Rehearsal showed a higher variance in final performance than in the previous datasets. Moreover, there were certain outlier runs that resulted in lower mean accuracy. For example, the sudden drop in mean final accuracy from 100 to 200 samples is attributed to a single run and does not represent the trend of the rest of the runs. We hypothesize that in this dataset Rehearsal was more sensitive to the particular composition of samples stored in memory. Nevertheless, as seen in the plot, Replay-SLDA (v2) was less affected by this which could be attributed to the use of the LDA layer.

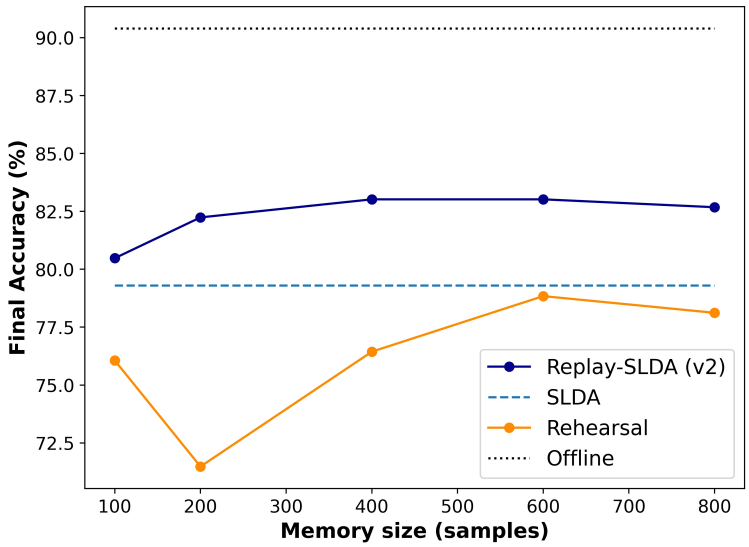

FIGURE 6.7: Comparing the performance of Replay-SLDA to Rehearsal, SLDA, and Offline models on CRH for different memory sizes. Each point corresponds to the average value over 5 runs.

Once again, the results in 6.8 show the evolution of the final accuracy at different points of the stream. Contrary to the previous datasets, Replay-SLDA (v2) significantly outperforms SLDA at the initial stages of the stream (less than 30%) for memory sizes of 100 and 200 samples. We hypothesize that this observation is made

because in these cases the feature extractor has learned new feature representations that can be utilized by the LDA layer to achieve better final accuracy than SLDA. This is heavily dependent on each dataset used. As to why this is not observed for the case of memory size 400, we have to consider that in each experiment the class ordering of the stream is chosen at random. Thus if instances from a certain class affect final performance more than others, class ordering would also affect performance in the initial stages of the stream. We theorize that this is the case in this dataset, which also justifies the higher variance in the results as previously mentioned.

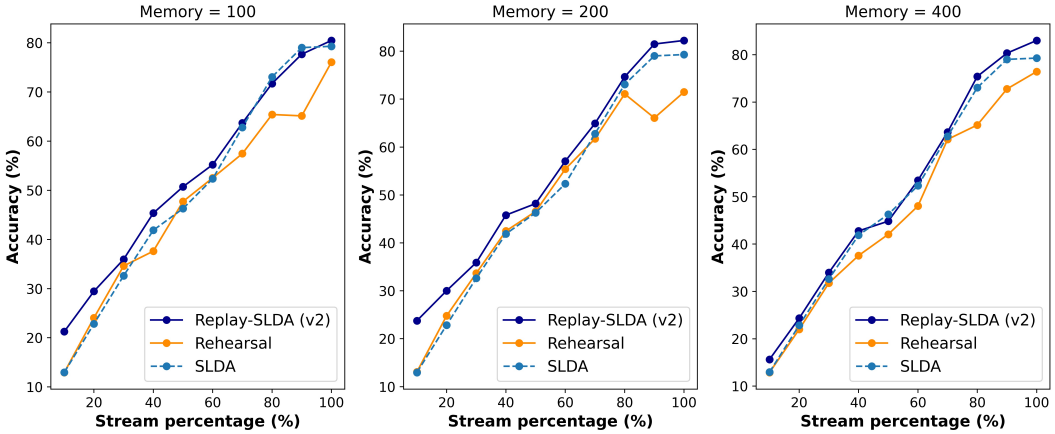

FIGURE 6.8: Evaluation at different percentages of the CRH stream for different memory sizes. Each point corresponds to the average value over 5 runs.

The final plot (see figure 6.9) concerns again the comparison of different implementations of the Replay-SLDA model. Similar observations to the previous datasets are made. An interesting observation is that Replay-SLDA (v2) has similar performance to the ideal model already from a memory size of 100 samples. At that memory size, their difference in final accuracy between the two models is 1.3% while from memory sizes of 400 samples and above their accuracies are almost identical. This demonstrates that in this dataset with only about 13 samples per class the approximations of the mean feature vectors are almost identical to the real ones. Perhaps in CRH dataset the instances in each class have similar feature representations hence they are easily approximated with only a few samples.

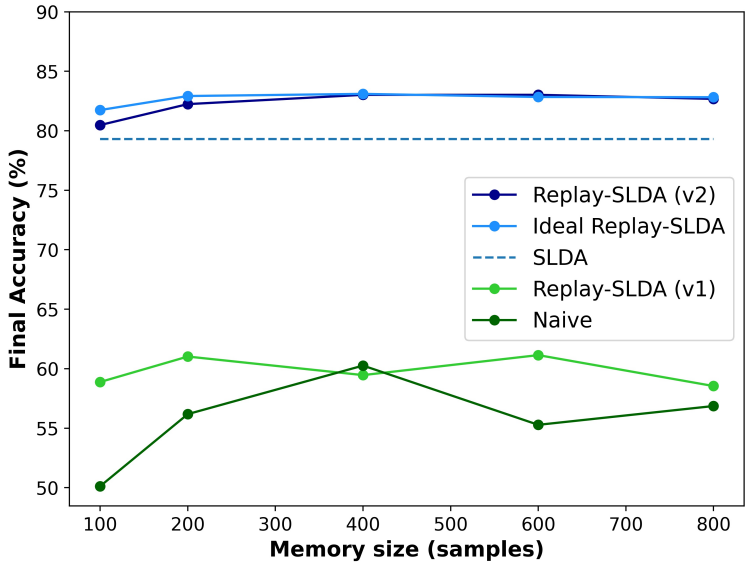

FIGURE 6.9: Comparing different implementations of Replay-SLDA on CRH. Each point corresponds to the average value over 5 runs.

6.4 Discussion

The experimental results demonstrate that our final implementation of Replay-SLDA successfully combines rehearsal-based training with SLDA, addressing the SLDA model's limitation of relying on a frozen feature extractor. On all three tested datasets, our model outperformed the SLDA model already from relatively low memory sizes. Specifically for CIFAR-10, CIFAR-100, Replay-SLDA needed at least 200 and 2000 samples respectively (corresponding on average to 20 samples per class) while for CRH it needed at least 100 samples in total (12-13 samples per class on average). SLDA seems to be superior to Replay-SLDA only under low memory conditions (10 samples per class or less), thus in such scenarios perhaps SLDA should be preferred.

The comparison of the ideal model, which uses the real mean feature vectors of each class, to Replay-SLDA (v2) showed that the use of approximations of the mean feature vectors is a successful way to address drift without significantly compromising the performance of LDA in the output layer. In CIFAR-10 and CIFAR-100, for memory sizes corresponding roughly to 40 samples per class, Replay-SLDA had only 1.6% lower final accuracy than the ideal model while for the same samples per class this difference was only 0.08% in the CRH dataset. Consequently, for typical memory sizes, using the memory to train the output layer with LDA shows similar performance to the ideal model which uses the entire stream, thus making it an effective way to avoid drift.

Regarding the results Replay-SLDA (v1), it is clear that it did not succeed in managing drift. Even though, the transformation of the mean feature vectors (see

equation 5.2) corrected for part of the drift, the performance of the model was consistently lower than SLDA and closer to the Naive model than the ideal one. This is even more prominent in CIFAR-100, where there is a high number of classes and the use of a common transformation for all of them is far from optimal for correcting for drift.

Finally, in table 6.1 we present the mean final accuracies together with their estimated standard deviation for the different tested continual learning algorithms on all three datasets. For each dataset, we chose to present the results for a memory size

TABLE 6.1: Mean final accuracy and standard deviation over 5 runs of different tested continual learning algorithms on CIFAR-10, CIFAR-100, and CRH. The memory size m is chosen to correspond on average to 40 samples per class in each dataset.

Methods	CIFAR-10 (m=400)	CIFAR-100 (m=4000)	CRH (m=400)
Naive	43.4 ± 4.2	28.0 ± 1.6	60.3 ± 6.1
Rehearsal	69.1 ± 3.4	48.8 ± 3.1	76.4 ± 6.8
SLDA	75.2 ± 0.0	52.4 ± 0.0	79.3 ± 0.0
Replay-SLDA (v1)	60.2 ± 4.6	30.0 ± 1.9	59.5 ± 5.9
Replay-SLDA (v2)	$\mathbf{81.0 \pm 0.8}$	$\mathbf{56.1 \pm 0.7}$	$\mathbf{83.0 \pm 1.4}$

m corresponding roughly to 40 samples per class. Replay-SLDA achieves the highest final accuracy (highlighted in bold) on all three datasets. The SLDA model has zero standard deviation since it is a deterministic model and its final accuracy depends only on the mean feature vectors of the stream (given a static covariance matrix). It is interesting to note that the algorithm with the lowest standard deviation after SLDA is Replay-SLDA (v2). In all of the tested algorithms involving a rehearsal scheme, the component of rehearsal adds some randomness to the experiments. However, in Replay-SLDA (v2) this element of randomness is moderated by the use of the LDA layer which tries to maximize class separation. Consequently, besides having a higher final accuracy than all tested models, Replay-SLDA (v2) is also more robust compared to the stochastic ones showing less variance in its predictive performance.

Chapter 7

Conclusions and Future Work

The goal of the thesis was to investigate whether the SLDA model could be combined with a rehearsal-based scheme and how the resulting model would perform compared to the SLDA and rehearsal models individually. The SLDA model introduces a novel approach to tackling catastrophic forgetting in image classification tasks by using LDA to train the classification layer (Hayes and Kanan, 2020). This type of training is resistant to catastrophic forgetting making it suitable for the continual learning setting. However, the model relies on a pre-trained frozen feature extractor which prevents it from learning new feature representations. This is a major limitation since it renders the model static, defeating the continual learning purpose, and also makes its performance heavily dependent on the pre-training phase of the feature extractor.

To overcome the limitations of the SLDA model, we proposed to let the feature extractor learn new feature representations through rehearsal-based training while updating the classification layer with LDA. This idea however is not easily implemented due to a major difficulty that arises when trying to combine these two methods. Namely, by updating the weights of the feature extractor, the stored mean feature vectors utilized by the LDA layer start to drift away from the ever-changing feature space. In a way, the feature extractor can learn new features at the cost of the LDA performance. This turned out to be quite a challenging problem to solve.

Initially, we tried to tackle feature drift by transforming the stored mean feature vectors after every update of the feature extractor. To avoid other proposed computationally intensive transformations we used a simplistic transformation described by equation 5.4. However, the use of this transformation was not enough to correct for drift. Our next approach focused on avoiding the occurrence of drift overall and turned out to be a successful solution to the problem. In this approach, we used conventional rehearsal-based training to train a CNN on the incoming stream. At inference time we created an SLDA model by using the trained CNN as a feature extractor and replaced the output layer with an LDA layer trained on the stored instances from the memory. By using the stored samples at the end of the training to train the output layer, this implementation totally avoids the drift of stored feature

vectors. However, the cost for this is to use approximations for the real mean feature vectors of each class since they are calculated from a small subset of the total stream.

Our experimental results on three different datasets demonstrated that our final implementation Replay-SLDA had better performance than the rehearsal and SLDA models individually. Specifically, Replay-SLDA attained higher accuracy than the SLDA model already from relatively low memory sizes, while it consistently outperformed simple rehearsal for all tested memory sizes. Furthermore, our implementation of Replay-SLDA showed similar performance at typical memory sizes to the ideal Replay-SLDA model which uses the real mean feature vectors of the stream. This means that using approximations for the mean feature vectors is not a major concern as long as the memory contains enough samples per class. However, we acknowledge that this is, to a certain degree, dependent on the dataset used.

We can conclude that all of the goals set in this thesis were met. We demonstrated that rehearsal can be successfully combined with SLDA and provided an effective way to deal with feature drift which constituted the main challenge compromising the joint use of the two algorithms. Furthermore, we showed that already from low memory sizes the combined model outperformed the SLDA and rehearsal models individually by exploiting the advantages of each method. Consequently, we can conclude that Replay-SLDA can be considered as a possible improvement over SLDA that addresses its main limitation of relying on a frozen feature extractor.

In this final paragraph, we would like to share some ideas for further improving our model that could serve as inspiration for future work. An interesting direction would be to replace reservoir sampling with another memory population algorithm that focuses on selecting samples such as to better approximate the real mean feature vectors of the stream. This could enhance the performance of the model making it similar to the performance of the ideal model even at low memory sizes. Furthermore, reservoir sampling is sot suited for imbalanced datasets. Sampling methods that try to balance the memory population as in (Chrysakis and Moens, 2020), while also trying to optimize for approximating the stream's mean feature vectors could also be a possible direction of research. Finally, the effect of a covariance matrix constructed from the instances of the memory buffer could also be investigated. Similar to how Replay-SLDA uses the memory to approximate the mean feature vectors at the end of the rehearsal training, one could also use the memory to construct an approximation of the covariance matrix.

Bibliography

Abraham, W. C. and Robins, A. (2005). Memory retention–the synaptic stability versus plasticity dilemma. *Trends in Neurosciences*, 28(2):73–78.

Aljundi, R., Caccia, L., Belilovsky, E., Caccia, M., Lin, M., Charlin, L., and Tuytelaars, T. (2019a). Online continual learning with maximally interfered retrieval. *arXiv preprint arXiv:1908.04742*.

Aljundi, R., Kelchtermans, K., and Tuytelaars, T. (2019b). Task-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11254–11263.

Benna, M. K. and Fusi, S. (2016). Computational principles of synaptic memory consolidation. *Nature Neuroscience*, 19(12):1697–1706.

Bishop, C. M. (2006). Pattern recognition. *Machine Learning*, 128(9).

Broucke, S. v. (2021). Advanced analytics in business lecture 8: Deep learning.

Candemir, S., Nguyen, X. V., Folio, L. R., and Prevedello, L. M. (2021). Training strategies for radiology deep learning models in data-limited scenarios. *Radiology: Artificial Intelligence*, 3(6):e210014.

Cao, X., Wipf, D., Wen, F., Duan, G., and Sun, J. (2013). A practical transfer learning algorithm for face verification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3208–3215.

Castro, F. M., Marín-Jiménez, M. J., Guil, N., Schmid, C., and Alahari, K. (2018). End-to-end incremental learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 233–248.

Cauchy, A. et al. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. (2018). Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*.

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H., and Ranzato, M. (2019). On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*.

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., and Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 801–818.

Chrysakis, A. and Moens, M.-F. (2020). Online continual learning from imbalanced data. In *International Conference on Machine Learning*, pages 1952–1961. PMLR.

Cui, Y., Song, Y., Sun, C., Howard, A., and Belongie, S. (2018). Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 4109–4118.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Dasgupta, S. and Hsu, D. (2007). On-line estimation with the multivariate gaussian distribution. In *International Conference on Computational Learning Theory*, pages 278–292. Springer.

De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on Computer Vision and Pattern recognition*, pages 248–255. Ieee.

Ditzler, G., Roveri, M., Alippi, C., and Polikar, R. (2015). Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7).

Farquhar, S. and Gal, Y. (2018). Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188.

Fusi, S., Drew, P. J., and Abbott, L. F. (2005). Cascade models of synaptically stored memories. *Neuron*, 45(4):599–611.

Gerstner, W. and Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press.

Ghojogh, B. and Crowley, M. (2019). Linear and quadratic discriminant analysis: Tutorial. *arXiv preprint arXiv:1906.02590*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in Neural Information Processing Systems*, 27.

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

Grossberg, S. (1982). How does a brain build a cognitive code? *Studies of mind and brain*, pages 1–52.

Grossberg, S. (2013). Adaptive resonance theory: How a brain learns to consciously attend, learn, and recognize a changing world. *Neural Networks*, 37:1–47.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377.

Gurney, K. (2018). *An introduction to neural networks*. CRC press.

Hassabis, D., Kumaran, D., Summerfield, C., and Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258.

Hayes, T. L., Kafle, K., Shrestha, R., Acharya, M., and Kanan, C. (2020). Remind your neural network to prevent catastrophic forgetting. In *European Conference on Computer Vision*, pages 466–483. Springer.

Hayes, T. L. and Kanan, C. (2020). Lifelong machine learning with deep streaming linear discriminant analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 220–221.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 770–778.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

IBM Cloud Education (2020a). Gradient descent. https://www.ibm.com/cloud/learn/gradient-descent Last accessed on 2021-12-14.

IBM Cloud Education (2020b). Neural networks. https://www.ibm.com/cloud/learn/neural-networks Last accessed on 2021-12-14.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456. PMLR.

Iscen, A., Zhang, J., Lazebnik, S., and Schmid, C. (2020). Memory-efficient incremental learning through feature adaptation. In *European Conference on Computer Vision*, pages 699–715. Springer.

Isele, D. and Cosgun, A. (2018). Selective experience replay for lifelong learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Kather, J. N., Weis, C.-A., Bianconi, F., Melchers, S. M., Schad, L. R., Gaiser, T., Marx, A., and Z"ollner, F. G. (2016). Multi-class texture analysis in colorectal cancer histology. *Scientific Reports*, 6:27988.

Kemker, R., McClure, M., Abitino, A., Hayes, T., and Kanan, C. (2018). Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images.

LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2.

Lee, S., Ha, J., Zhang, D., and Kim, G. (2020). A neural dirichlet process mixture model for task-free continual learning. *arXiv preprint arXiv:2001.00689*.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2017). Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*.

Li, Z. and Hoiem, D. (2017). Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947.

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., and Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26.

Liu, X., Wu, C., Menta, M., Herranz, L., Raducanu, B., Bagdanov, A. D., Jui, S., and de Weijer, J. v. (2020). Generative feature replay for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 226–227.

Lopez-Paz, D. and Ranzato, M. (2017). Gradient episodic memory for continual learning. *Advances in Neural Information Processing Systems*, 30:6467–6476.

Mallya, A. and Lazebnik, S. (2018). Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773.

McClelland, J. L., McNaughton, B. L., and O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102(3):419.

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

Mermillod, M., Bugaiska, A., and Bonin, P. (2013). The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology*, 4:504.

Murray, M. M., Lewkowicz, D. J., Amedi, A., and Wallace, M. T. (2016). Multisensory processes: a balancing act across the lifespan. *Trends in Neurosciences*, 39(8):567–579.

Ng, A., Ngiam, J., Foo, C. Y., Mai, Y., and Suen, C. (2010). Ufldl tutorial. *Computer Science Department, Stanford University. http://deeplearning. stanford. edu/tutorial*.

O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

Pang, S., Ozawa, S., and Kasabov, N. (2005). Incremental linear discriminant analysis for classification of data streams. *IEEE Transactions on Systems, Man, and Cybernetics, part B (Cybernetics)*, 35(5):905–914.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71.

Pellegrini, L., Graffieti, G., Lomonaco, V., and Maltoni, D. (2020). Latent replay for real-time continual learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10203–10209. IEEE.

Power, J. D. and Schlaggar, B. L. (2017). Neural plasticity across the lifespan. *Wiley Interdisciplinary Reviews: Developmental Biology*, 6(1):e216.

Rao, D., Visin, F., Rusu, A. A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2019). Continual unsupervised representation learning. *arXiv preprint arXiv:1910.14481*.

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

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. (2017). icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010.

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

Rolnick, D., Ahuja, A., Schwarz, J., Lillicrap, T. P., and Wayne, G. (2018). Experience replay for continual learning. *arXiv preprint arXiv:1811.11682*.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386.

Ruder, S., Peters, M. E., Swayamdipta, S., and Wolf, T. (2019). Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18.

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

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

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

Shin, H., Lee, J. K., Kim, J., and Kim, J. (2017). Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*.

Torrey, L. and Shavlik, J. (2010). Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global.

van de Ven, G. M., Siegelmann, H. T., and Tolias, A. S. (2020). Brain-inspired replay for continual learning with artificial neural networks. *Nature Communications*, 11(1):1–14.

Van de Ven, G. M. and Tolias, A. S. (2019). Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*.

Verwimp, E., De Lange, M., and Tuytelaars, T. (2021). Rehearsal revealed: The limits and merits of revisiting samples in continual learning. *arXiv preprint arXiv:2104.07446*.

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57.

Webster, R. (2001). *Neurotransmitters, drugs and brain function*. John Wiley & Sons.

Weiss, K., Khoshgoftaar, T. M., and Wang, D. (2016). A survey of transfer learning. *Journal of Big data*, 3(1):1–40.

Wu, Y., Chen, Y., Wang, L., Ye, Y., Liu, Z., Guo, Y., and Fu, Y. (2019). Large scale incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 374–382.

Xiao, T., Zhang, J., Yang, K., Peng, Y., and Zhang, Z. (2014). Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 177–186.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. (2017). Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? *arXiv preprint arXiv:1411.1792*.

You, K., Long, M., Wang, J., and Jordan, M. I. (2019). How does learning rate decay help modern neural networks? *arXiv preprint arXiv:1908.01878*.

Yu, L., Twardowski, B., Liu, X., Herranz, L., Wang, K., Cheng, Y., Jui, S., and Weijer, J. v. d. (2020). Semantic drift compensation for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6982–6991.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR.

Ziou, D., Tabbone, S., et al. (1998). Edge detection techniques-an overview. *Pattern Recognition and Image Analysis C/C of Raspoznavaniye Obrazov I Analiz Izobrazhenii*, 8:537–559.