Directed Study Report
CSE4DIR

Investigation of Catastrophic Forgetting in Neural Networks

Author

Ash Hall

Honours Student

Department of Computer Science

La Trobe University

Supervisor

Zhen He

Professor

Department of Computer Science

La Trobe University

Contents

List of Figures

Directed Study Report

1.1 Introduction

The purpose of this report is to investigate the effects of catastrophic forgetting in artificial neural networks. We will briefly discuss what catastrophic forgetting is and why it occurs, consider some approaches for minimising its effects, then present our experimental findings and discuss the results.

As neural networks are improved by optimising the parameters on a per-example basis, they rapidly forget past learned knowledge to "make way" for new information. This purportedly occurs due to overlap in internal representations of new and old tasks, where a model may "re-purpose" important parameters for a new task, regardless of their importance for old tasks. While catastrophic forgetting is well recognised, its effects are yet to be quantified for comparison. We aim to produce empirical measurements by way of experimentation, and to provide a clear measure of the severity of catastrophic forgetting in artificial neural networks that can be used for comparison moving forward.

1.2 Transfer Learning

The training of a neural network occurs by presenting many examples, and iteratively optimising the weights. A dataset consisting of a small number of examples may be prohibitive to optimisation in this manner, as a model would likely over-fit to the training examples, and not be capable of generalising to novel examples. *Transfer learning* minimises this problem, by allowing a pre-trained model to be re-purposed to a different dataset.

The bulk of an image-classification neural network acts as a *feature-extractor*, with studies[4] showing that a sufficiently-large image dataset leads to a highly-generalisable feature extractor. Transfer learning takes advantage of this property by taking the resultant feature extractor and amending the tail of a neural network to perform classification between different classes. Figure 1.1 shows a model pre-trained on a large image dataset being adapted to another.

There are two fundamental problems with this approach. The first is that although this lessens the effects of over-fitting, it doesn't entirely avoid them; training on a very small dataset is still prone to this problem. The other issue is that if you wished to not replace the old dataset classes, but add the new dataset classes to the model, you would need to store examples and repeatedly present them to the network during re-training. This introduces a whole suite of problems including training-example balancing, which we won't address here.

Simply put, while providing strong results, transfer learning is limited in its usability, and is inherently inapplicable to the task of continuous learning – which we'll address in the following section.
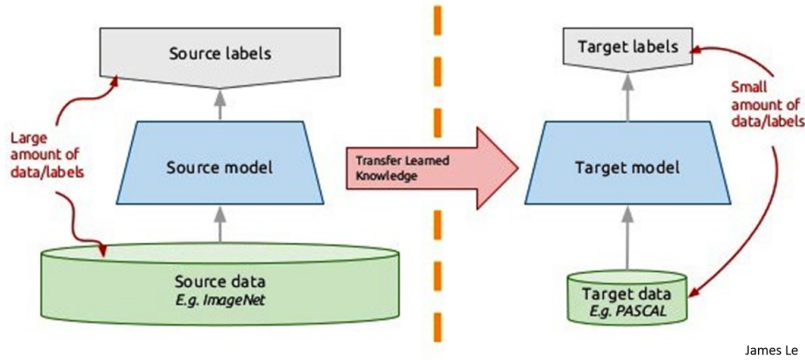
Figure 1.1: Transfer learning overview

1.3 Continuous Learning

The tendency of neural networks to forget older information is known as catastrophic forgetting; the task of learning from a stream of new information with a minimal amount of forgetting is known as continuous learning.

Continuous learning differs from transfer learning in that in order to add classes to an image classifier using transfer learning, examples from the existing classes would need to be *replayed* to the model over time, to ensure that it loses minimal performance on them. A condition of "true" continuous learning is that old examples need not be seen again – this is a reasonable desire, as it would require an increasing amount of training time proportionate to the growing number of images and classes in a dataset.

1.3.1 Related Works

We will now briefly discuss some works related to the task of mitigating catastrophic forgetting – continuous learning. Although this report is to investigate the effects of catastrophic forgetting and not a survey of related work, it is good to consider other approaches. The following works were built to specifically target catastrophic forgetting, so serve as good points of reference.

Learning to Learn with Backpropagation of Hebbian Plasticity

The work by Miconi et. al. [6] considers an observed biological neurological pattern, where synaptic efficacy (neuron firing strength) increases through persistent stimulation. In this approach, they define a "Hebbian trace", which acts as a moving-average of pair-wise synaptic activities. They train a plasticity parameter which determines how much the Hebbian trace affects a pair's connection.

A network gains additional learning capacity by using this technique, as the network learns a sparser internal representation, effectively endowing it with a larger capacity for knowledge. Although catastrophic forgetting is somewhat reduced, this is not an ideal situation as the model's internal capacity remains fixed and as such, is limited to the initial design.

Learning Without Forgetting

The technique proposed by Li et. al. [5] seeks to preserve acquired knowledge by recording the existing model's response to new classes examples. They augment the model by adding an

4

additional set of outputs for the new classes, and producing a prediction probability distribution on the old heads using new class examples (figure 1.2).
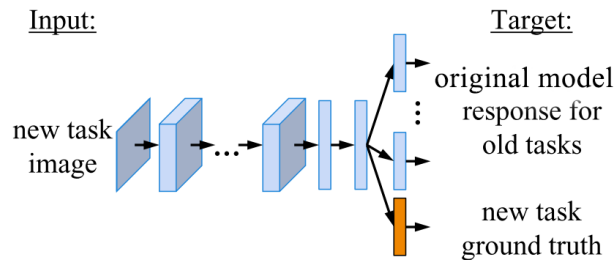

Figure 1.2: Learning Without Forgetting architecture
A new output head (orange) is added for each batch of new classes. The model's predicted class probability distribution over the old classes is recorded and incorporated into the loss function.

This is a very literal and largely successful approach to mitigating catastrophic forgetting, but comes with a price – the user of the model must know prior to performing inference which set of classes the image belongs to. This effectively makes it quite similar to having multiple models, the only difference being that the class groups share a feature extractor.

Overcoming Catastrophic Forgetting

The approach presented by Kirkpatrick et. al. [5] introduces a specialised loss function named *elastic weight consolidation* (EWC), with the sole purpose of preserving old information. They compute a Fischer information matrix – which is essentially a representation of each weight's importance for the pre-existing classes – and use it to add an extra term to the loss calculation. With this term included, the loss function has the effect of restraining important parameters to a location in weight-space which provide good performance on the pre-existing classes.

Similarly to *Learning Without Forgetting*, this is an effective technique due to the fact that the retention of knowledge is explicitly embedded in the network's training.

Summary

The aforementioned techniques are designed to minimise the effects of catastrophic forgetting, and each does so in an effective and reasonable manner. That being said, the results yielded by each are incomparable as there is no consistent method for measuring catastrophic forgetting when extending neural networks to more classes. We seek to perform experiments which provide a consistent, coherent basis by which to measure these effects.

1.4   Environment

The following experiments were performed on a computer running the UNIX operating system, with two 8GB Nvidia GTX 1080 graphics cards, a quad-core processor and 64GB of memory. The implementation of the experiments was written using the Python interface for Tensorflow[1], Google's open-source computational framework. Furthermore, Deepmind's Sonnet[3] was used to provide a simple interface for Tensorflow. The graphing and result logging was done using Tensorboard, a visualisation tool included with Tensorflow.

1.5 Dataset

The experiments described in this report were performed using CIFAR-100[2], a dataset consisting of 60,000 32x32 images in 100 classes with 600 images per class. While a small – in terms of spatial dimensions – dataset, the 100 classes emulate real-world by having such varied classes as motorcycles, snakes and bottles.

CIFAR-10 is a variant of CIFAR-100, with more images but only 10 classes. CIFAR-10 is a regularly used model for measuring performance of techniques, but isn't applicable to our problem domain as we require a larger number of classes. Use of CIFAR-100 is regularly used in tasks where a larger number of classes is required, such as few-shot learning and continuous learning.

1.6 Framework

The code-base for the experiments is provided in figure 1.3, which details a high-level overview of the Python class, module and directory structure. The `Trainer` trains models, and takes command-line flags to switch the training mode (used when re-training models in different manners). It is also responsible for session management, and writing variables and Tensorboard data to the `bin` directory. The `GraphBuilder` abstracts the graph-building operations away from the `Trainer`, to allow for a cleaner training script. It reads from a configuration `YAML` file stored in `configs`, and builds the appropriate `Model` and `Optimizer` as required. The data loading, partitioning and sampling is presented via a consistent interface in the `dataloader` package, with the dataset loaded being specified in the configuration file mentioned.

The described modular design provides the ability to quickly perform experiments with different datasets/model configurations. Automatic file-naming in the `bin` directory allows for experiment results to be easily reviewed and compared in Tensorboard with little effort.

1.7 Key Terminology

Source classes refer to a set of classes on which a *source model* has been pre-trained. Similarly, a *target model* is produced by extending a source model's classification ability to also include a set of *target classes* – a disjoint set of classes to those that the source model was trained on.

1.8 Pre-Training Procedure

Each of the experiments were performed in a two-stage training procedure. The first stage of training was consistent for each experiment, which will now be described.

The model architecture (figure 1.4) used for training was consistent with other works of the same nature, with a body consisting of four convolutional blocks (figure 1.5) followed by two parallel fully-connected output layers – one each for the source and target datasets.

For CIFAR-100, the number of total classes $N = 100$, allowing us the ability to choose a smaller number of *source* and *target* classes and compare results. For a number of per-model source classes N_S, we can easily build $N - N_S - 1$ overlapping incremental class subsets (figure 1.6).

We have chosen to work with the number of source classes $N_S = 50$, and the number of target classes $N_T = 10$, for a total of 60-way classifier in the end. This number of classes was chosen as the number of examples in each class is quite low, so training on 50 classes allows for
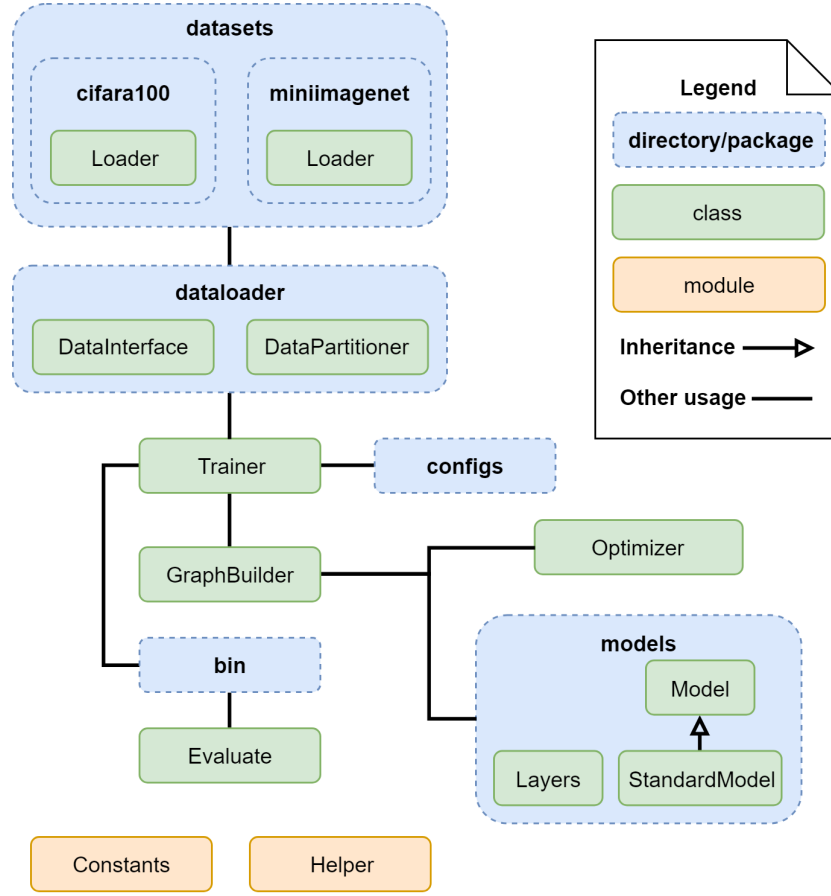
Figure 1.3: Code-base Framework
A high-level view of the code-base structure.

the learning of somewhat general features. This is also to better simulate a real-world use-case, where classifiers in practice often have a large number of classes between which to predict.

For each of these subsets, a model was trained using an Adam optimizer with a learning rate of 10^{-4}. Due to the relatively small number of examples in the training/validation sets (500/100), training was stopped after 3000 iterations. Stopping at this many training iterations was determined empirically, as training for longer caused the model's accuracy on the validation set to drop due to over-fitting to the training set.

The weights of each model were saved to disk and not modified any further, such that each experiment began with the exact same models.

1.9 Metrics

A critical aspect is that the metrics reported are truly representative of the amount of catastrophic forgetting occurring. Due to this, each reported value is the average value over $N - N_S - 1$ models.

The quality of a model is measured by two components: the accuracy of the target model
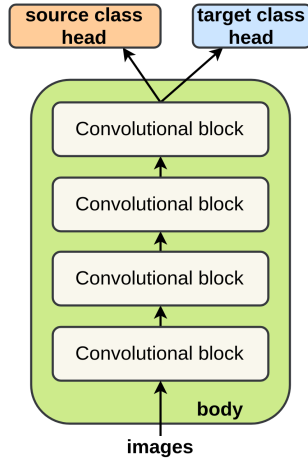
Figure 1.4: Model architecture
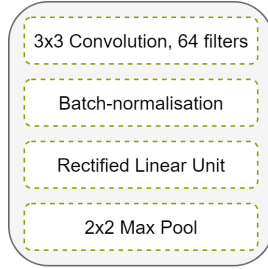A separate fully-connected layer was used for each set of output classes

Figure 1.5: A convolutional block

on the target classes, and the accuracy of the target model on the source classes. The former is a measure of how much the model learns about the target classes; the latter a measure of how much the model retains accuracy on the source classes. Instead of considering the precision in terms of their absolute value, we will instead compute the difference between target model and source model precision. Concretely, for a collection of images, a model will produce a number of correct (C) and incorrect (I) predictions. We can therefore compute the precision for a given model $\boldsymbol{\theta}$ on images $\boldsymbol{x}$ as in equation 1.1.

$$\text{Precision}(\boldsymbol{\theta}, \boldsymbol{x}) = \frac{C}{C + I} \tag{1.1}$$

To compute the relative improvement or deterioration of performance of a model, we simply need to compute the difference in precision between a source model and corresponding target model. It is not possible to measure the source model's performance on the target classes directly however, as it is in the problem definition that the source model has never seen the target images. In this case, we will train a model of equivalent capacity from scratch to predict between the target classes. This model will be used to compute the source model's accuracy on the target data-set, but it should be noted that this model is discarded once pre-training is complete, and only the true source model used. The improvement score equations for source and target classes
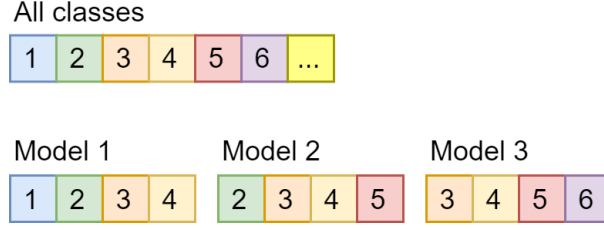
Figure 1.6: Class subset selection
A number of class subsets can easily be selected using an overlapping, incremental selection schema. Example using number of source classes $N_S = 4$.

are shown in equations 1.2 and 1.3, respectively.

$$\text{Score}_S = \text{Precision}(\boldsymbol{\theta}_T, \boldsymbol{x}_S) - \text{Precision}(\boldsymbol{\theta}_S, \boldsymbol{x}_S) \tag{1.2}$$

$$\text{Score}_T = \text{Precision}(\boldsymbol{\theta}_T, \boldsymbol{x}_T) - \text{Precision}(\boldsymbol{\theta}_S, \boldsymbol{x}_T) \tag{1.3}$$

By only considering the relative improvement of a model's performance, we don't allow the difficulty of the particular classes to distort our metrics, nor the quality of the source models.

Score_S represents reverse-transfer – a positive score indicates that the model's performance has improved on the source classes by seeing the target classes; a negative score indicates forgetting of knowledge pertinent to the source classes.

Score_T represents forward-transfer – a positive score indicates that the model yielded better target-class performance having been pre-trained on the source classes; a negative score indicates that the pre-training (or restricted training procedure) worsened results on the target classes

1.10   Experiments and Results

After training a batch of 51 models as according to section 1.8, the accuracies obtained on the validation set of the source dataset $\boldsymbol{x}_S$ and target dataset $\boldsymbol{x}_T$ were 38.87% and 64.88%, respectively .

For the source dataset, it may seem like a low validation accuracy, but it is worth mentioning that this 50-way classification task is already quite difficult, given that there are only $500 + 100 = 600$ images available for each image. The task of generalising from 500 known to 100 previously unseen examples is difficult, with other works typically employing image-augmentation techniques such as rotations, flips, crops, etc. No image augmentation techniques were used for this result, so that we may consider only the "true" classification accuracy.

With regards to the target dataset, it must be recognised that the task of 10-way classification is significantly easier than 50-way due to the reduced number of classes between which to classify.

1.10.1   Retraining with Source and Target Datasets

This experiment explores the performance under ideal circumstances – that is, where training can occur using images from both the source and target dataset. Having access to the source dataset minimises the effects of catastrophic forgetting, as models are consistently exposed to examples from the source dataset. It is unfair to compare the results from this experiment with those from other experiments, so this simply provides the "best possible" results.

After retraining for 2500 iterations, the classification accuracy on the source and target datasets were 35.73% and 28.95%, respectively – resulting in scores:

$$\text{Score}_S = -0.032$$
$$\text{Score}_T = -0.360$$

It is reasonable to expect a negative score for the source dataset, as simply by increasing the number of classes from 50 to 60 there is an inherent drop in accuracy. While the model did learn to classify between the target classes, its score is quite low. This is largely due to the same effects of an increased number of classes, but also that the model capacity is stretched rather thin compared to a situation in which only target class predictions are desired.

1.10.2 Retraining with only Target Dataset

The following experiments actually investigate the effects of catastrophic forgetting, as models aren't exposed to the source dataset after pre-training is complete. The technique employed in this section is referred to as "parameter freezing", which is performed by simply locking parameters for certain parts of the model, now allowing them to change when re-training.

We will explore the effects of different freezing strategies by re-training the models with different sections of the models frozen (figure 1.7), and will discuss the purpose behind the different freezing strategies and their consequences.

Catastrophic forgetting has a rapid onset, which can be seen in figure 1.8, reporting the score at retraining steps 5, 10, 20, 50, where we see that the third freezing strategy produces the best results, but that no techniques satisfactorily mitigate the effects of catastrophic forgetting. The final scores are compared in figure 1.9(a) and (b) after training for 50 and 2500 iterations respectively, demonstrating the benefits of transfer learning.
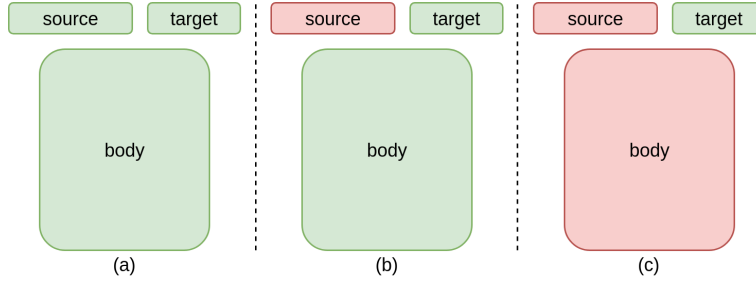

Figure 1.7: Retraining freezing strategies
The strategies used for weight-freezing when re-training. Green indicates trainable parameters, red indicates frozen parameters.

Training All Weights

In this experiment we consider the most naive of re-training strategies – re-training the entire model with no consideration for frozen weights (figure 1.7a). We should expect that the source output head's predictions quickly go to zero, as the "correct answer" is never one of those classes.

Training for 50 and 2500 iterations yielded respective scores of:

$$\text{Score}_S = -0.389$$
$$\text{Score}_T = -0.132$$

$$\text{Score}_S = -0.389$$
$$\text{Score}_T = +0.033$$

Catastrophic forgetting occurred very quickly and with great effect, with the model's performance on the source classes dropping rapidly to zero. This is to be expected and is a typical demonstration of catastrophic forgetting, as there is no attempt made to mitigate its effects. The model's body – which is supposed to act as a general feature-extractor – quickly discards information learnt relating to the source classes to learn quickly about the new classes. A positive score for the target classes implies that the model performed better due to pre-training. This is an expected result, as the pre-training serves as a good initialisation of weights, resulting in a model with slightly more generalised feature representations - this is a typical example of transfer-learning's efficacy.

Training Body and Target Output Head

This experiment investigates only freezing the source class output head (figure 1.7b). We shouldn't expect this to perform very well on the source classes, as although we aren't *explicitly* training the source output head to produce all zeros, the target output head may learn to produce very high confidence in its predictions to "outweigh" the source output head. Training for 50 and 2500 iterations yielded respective scores of:

$$\text{Score}_S = -0.389$$
$$\text{Score}_T = -0.134$$

$$\text{Score}_S = -0.389$$
$$\text{Score}_T = +0.037$$

The results were very similar to those of the previous experiment, as should be expected. The small difference in Score_T between this and the last experiment is largely superficial, as the network simply has to apply few updates per training-step in this experiment, resulting in a slightly faster training procedure. The reason for this model's success is that not only is the target head outweighing the source head's predictions, the features in the body are still being changed. The output head of a neural network classifier can be considered a representation of the intensity to which a particular feature appears in a given class. We are now modifying what each feature actually is, but not allowing the source head to update its representation to match them.

11

Training only Target Output Head

What we now explore is the method regularly used in transfer learning, where only the target output head has its parameters changed (figure 1.7c). This does differ slightly to transfer learning, in that transfer learning implies that the original head is removed entirely. We can expect that this technique will minimise catastrophic forgetting the most, as neither the feature extractor nor the source output head have their weights shifted. Training for 50 and 2500 iterations yielded respective scores of:

$$\text{Score}_S = -0.270$$
$$\text{Score}_T = -0.464$$

$$\text{Score}_S = -0.385$$
$$\text{Score}_T = -0.096$$

As expected, the catastrophic forgetting took much longer to occur to such an extent as previously encountered. This is as the source output head and the feature representations in the body of the model aren't allowed to change, so the source output head always aligns with the same features developed during pre-training. After some time, the target output head learns to produce outputs with artificially high raw prediction values, effectively outweighing the predictions from the other head.

It is interesting to note that this is the method with the lowest score, as the model has no opportunity to adapt the internal feature representation to suit the target classes. This effect is relatively small on datasets with similar features such as CIFAR, but would have a much greater effect on more diverse datasets where the source and target class features differ.

The difference in accuracy between the target model *training* and *validation* sets was quite small, which means that there was practically no over-fitting on the training set after so many iterations. We theorise that this trend would continue, and would occur due to a very limited amount of changes that the network could apply in order to improve. The body could be modified during training in other experiments, resulting in a feature extractor that is tightly-bound to the images it has seen. In contrast, this training schema only allows for adjustments to the output head, meaning that the model has to find the best-possible combination of features to represent the classes. In only having access to modify the output head, the model is very limited in its fitting capacity, resulting in a generalised representation of the target classes.

1.11   Summary

We have explored different weight-freezing strategies to minimise the effects of catastrophic forgetting, and have explored the severity to which catastrophic forgetting occurs when re-training a neural network. We found no satisfactory method for the mitigation of catastrophic forgetting. The effect is slowed if only retraining the new output head, but forgetting is still encountered with no mid-point resulting in good classification results on both sets of classes. Although the resultant model did have good generalisation ability, there exists a limit. The feature extractor already parametrised during the pre-training procedure is not guaranteed to represent the target classes, especially if the source and target classes differ by a large amount.
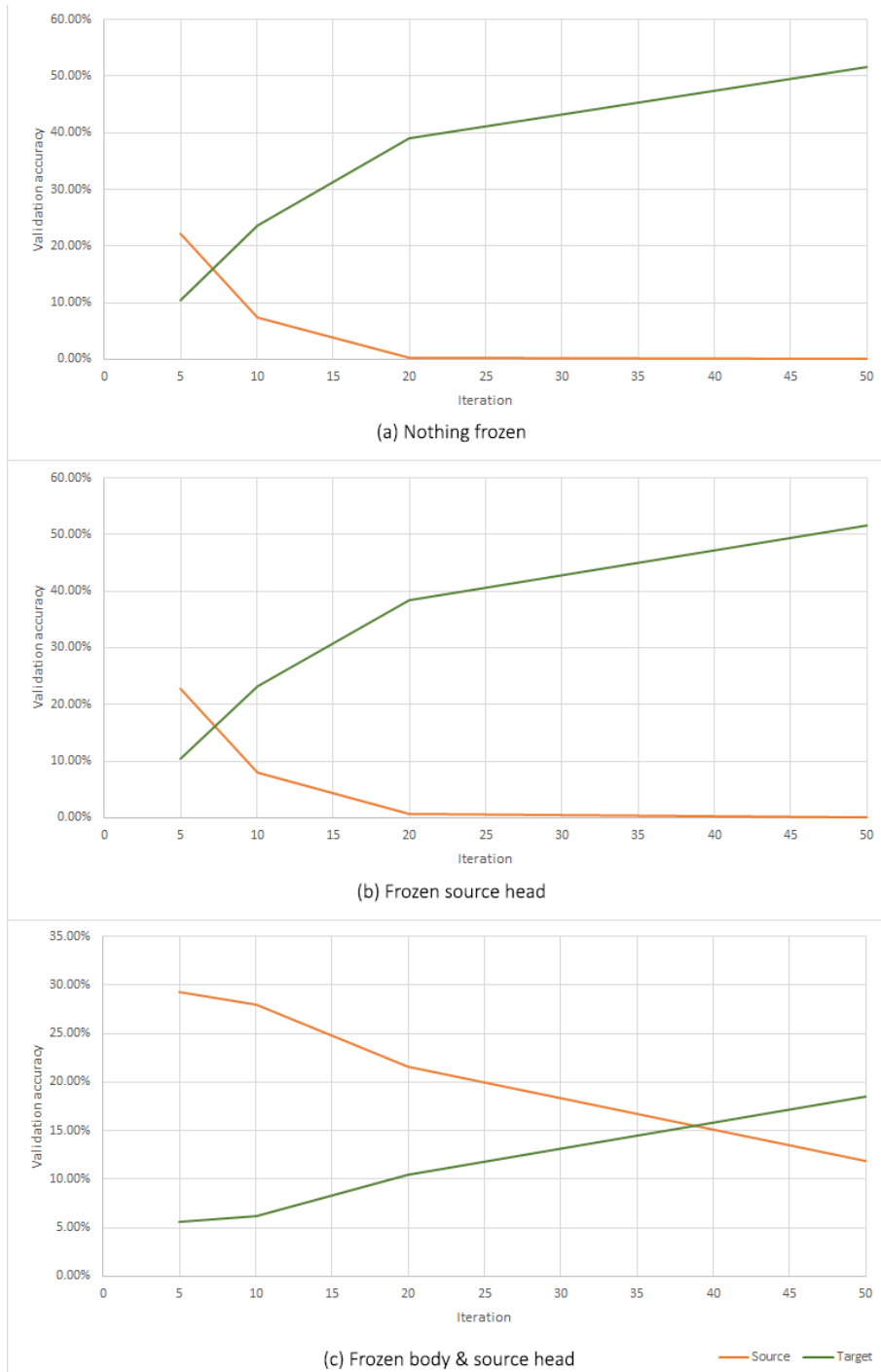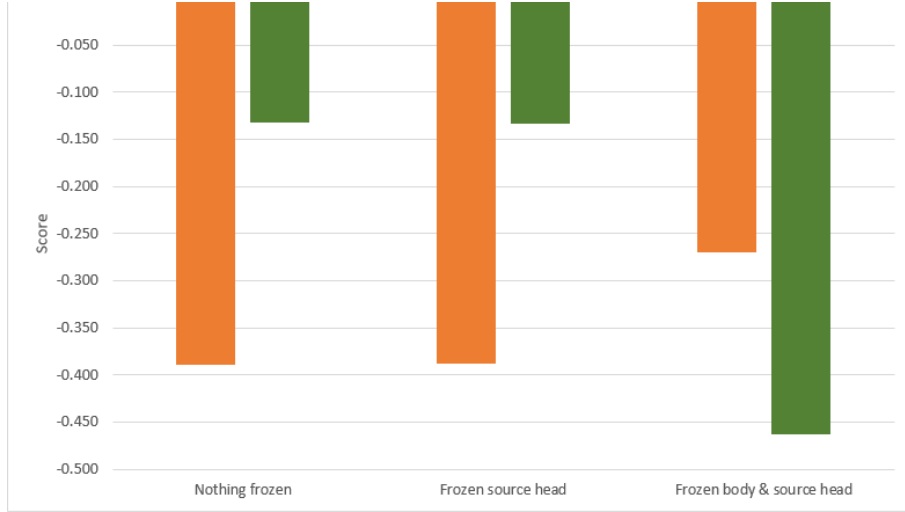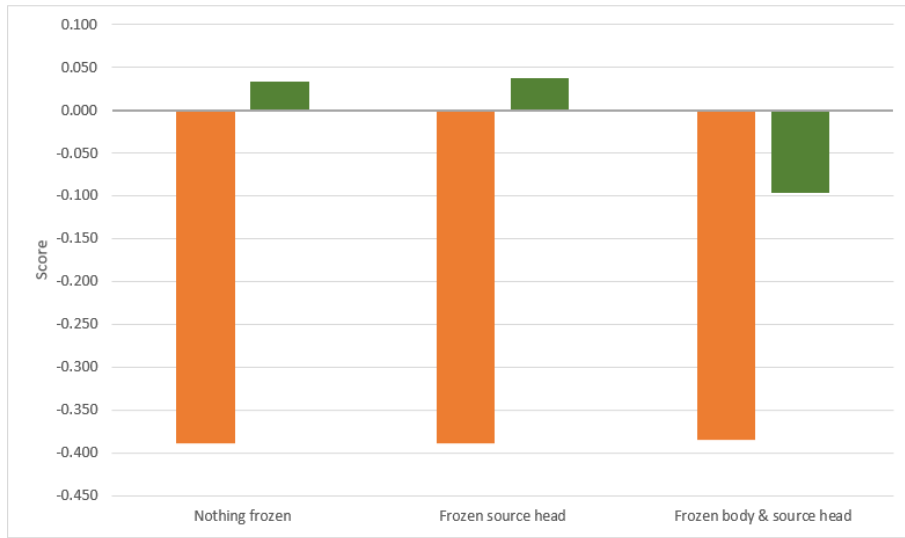
Figure 1.8: Freezing strategy learning/forgetting rates

(a) After 50 re-training iterations

(b) After 2500 re-training iterations

Figure 1.9: Freezing strategy learning/forgetting scores
A comparison of results showing the scores for each freezing strategy after 50 (a) and 2500 (b) iterations.

Bibliography

[1] Martín Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: *CoRR* abs/1603.04467 (2016). arXiv: `1603.04467`. URL: `http://arxiv.org/abs/1603.04467`.

[2] Geoffrey Hinton Alex Krizhevsky Vinod Nair. *CIFAR-10 and CIFAR-100 datasets*. 2009. URL: `https://www.cs.toronto.edu/~kriz/cifar.html` (visited on 05/14/2018).

[3] Deepmind. *Sonnet - Tensorflow-based neural network library*. 2017. URL: `https://github.com/deepmind/sonnet` (visited on 05/14/2018).

[4] Lars Hertel et al. "Deep Convolutional Neural Networks as Generic Feature Extractors". In: *CoRR* abs/1710.02286 (2017). arXiv: `1710.02286`. URL: `http://arxiv.org/abs/1710.02286`.

[5] Zhizhong Li and Derek Hoiem. "Learning without Forgetting". In: *CoRR* abs/1606.09282 (2016). arXiv: `1606.09282`. URL: `http://arxiv.org/abs/1606.09282`.

[6] Thomas Miconi. "Backpropagation of Hebbian plasticity for lifelong learning". In: *CoRR* abs/1609.02228 (2016). arXiv: `1609.02228`. URL: `http://arxiv.org/abs/1609.02228`.