Directed Study Report
CSE4DIR

Ash Hall

17756156

May 28, 2018

Contents

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 a neural network is 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* minimising 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[**extractors**] 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.
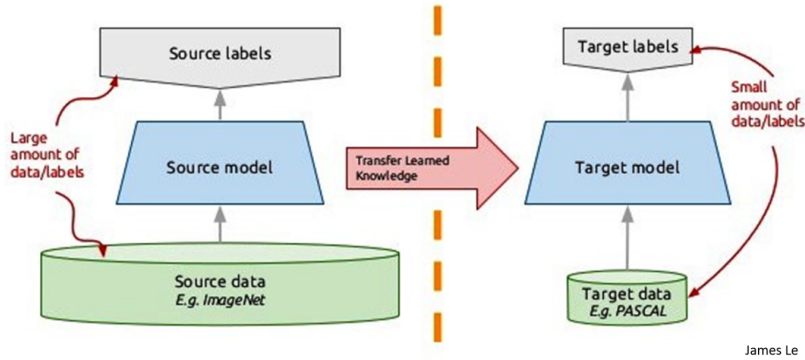
Figure 1.1: Transfer learning overview

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 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 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. [**hebbian**] 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 a larger capacity for knowledge. Although catastrophic forgetting is somewhat reduced, this is not an ideal situation as the models internal capacity remains fixed and as such, is limited to the initial design.

Learning Without Forgetting

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

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).
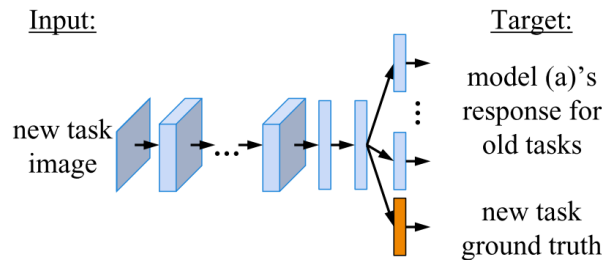

Figure 1.2: Learning Without Forgetting architecture
A new output head (orange) is added for each batch of new classes. The model's predicted class probability distribution over the old classes is recorded and incorporated into the loss function.

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. [**lwf**] 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 incompatible 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[**tensorflow**], Google's open-source computational framework. Furthermore, Deepmind's Sonnet[**sonnet**] 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[**cifar100**], a dataset consisting of 60,000 32x32 images in 100 classes with 600 images per class. While a small – in terms of spatial size – dataset, the 100 classes emulate real-world by having such varied classes as motorcycles, snakes and bottles.

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 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 1.7). 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 performance on the validation set to drop due 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.
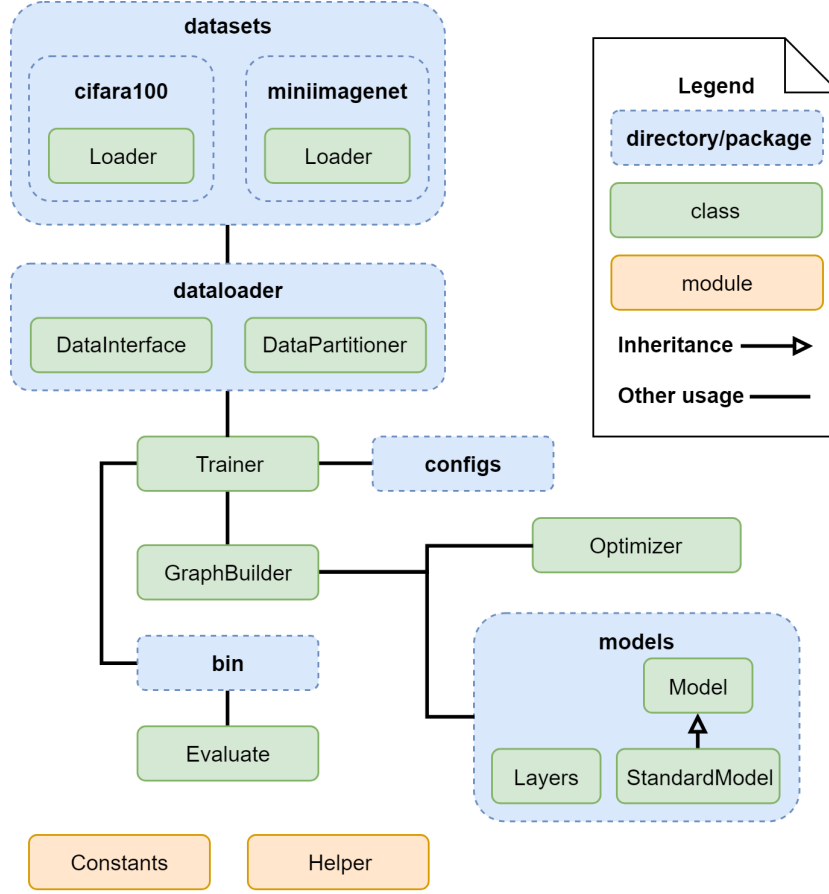
Figure 1.3: Code-base Framework
A high-level view of the code-base structure.

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 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
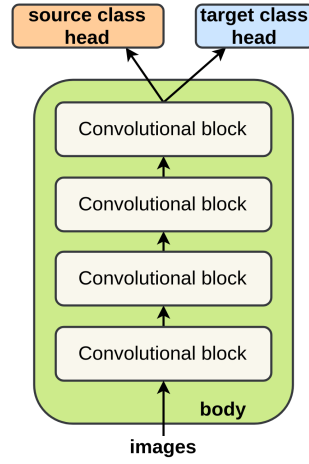
Figure 1.4: Model architecture
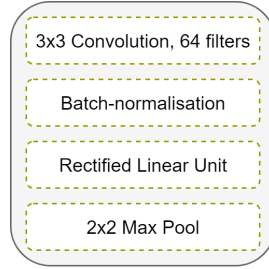A separate fully-connected layer was used for each set of output classes

Figure 1.5: A convolutional block

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 however, as it is in the problem definition that the source model has never seen the target images. In this case, we will consider the source model precision on the source images as a de-facto baseline by which to measure the target model's performance on the target classes. The improvement score equations for source and target classes are shown in equations 1.2 and 1.3, respectively.

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

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

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.

1.10   Experiments and Results

Initial score (source test): 38.8760784
(Put a short introduction to the results here after writing them up)
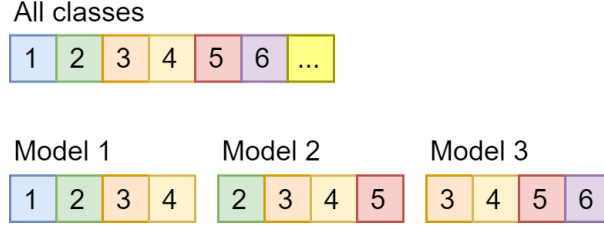
7

Figure 1.6: Class subset selection
A number of class subsets can easily be selected using an overlapping, incremental selection schema. Example using $N_S = 4$.

1.10.1 Retraining with Source and Target Datasets

This experiment explores the performance under the best-possible circumstances. Having access to the source dataset removes the possibility 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.

()

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. Due to the rapid onset of catastrophic forgetting, we will report the score at retraining steps 5, 10, 20 and 50.
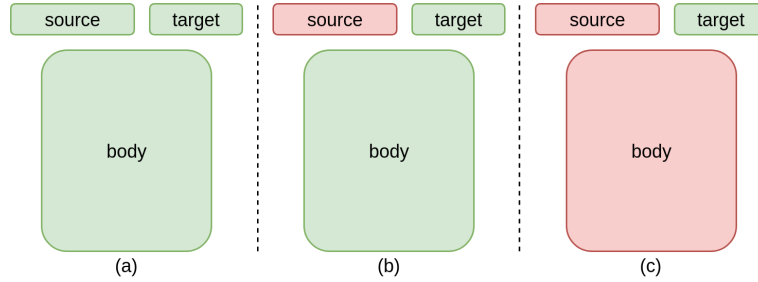()

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

5: S: 0.221976471; T: 0.103137255
10: S: 0.073976471; T: 0.23527451

20: S: 0.00345098; T: 0.38972549
50: S: 0.000070588; T: 0.516686275

Training Body and Target Output Head

5: S: 0.228384314; T: 0.103980392
10: S: 0.080156863; T: 0.231078431
20: S: 0.006039216; T: 0.38354902
50: S: 0.00012549; T: 0.515058824

Training only Target Output Head

5: S: 0.292670588; T: 0.055784314
10: S: 0.280105882; T: 0.062411765
20: S: 0.215713725; T:0.105235294
50: S: 0.118345098; T: 0.185078431
100: S: 0.069341176; T: 0.300588235
200: S: 0.03827451; T: 0.406764706
500: S: 0.016776471; T: 0.475333333
1000: S: 0.007670588; T: 0.511411765
2500: S: ; T:

()

1.10.3   Experiment 3

TODO

1.11   Summary

(Summarise the experiment results and what they indicate)