MLP Coursework 1: Learning Algorithms and Regularization

s1874193

Abstract

This report discusses and evaluates the RMSProp and Adam learning algorithms in the context of the EMNIST dataset. The impact of learning rate schedulers and the introduction of weight decay to the Adam algorithm are discussed and evaluated. Differences between $L2$ regularisation and weight decay for Adam and SGD are shown, and the introduction of Adam with weight decay presented a way to resolve these differences.

1. Introduction

This report explores the RMSProp and Adam learning algorithms and the effects of $L2$ regularisation. Models are trained on the EMNIST dataset (Cohen et al., 2017) and experiments are performed to compare different approaches. The primary questions explored are the impact of learning rate schedulers and weight decay on Adam, as well as the differences between Adam, RMSProp and SGD. The majority of the discussion focuses on the impact of cosine annealing and Adam with weight decay, as introduced by Loshchilov & Hutter (2017).

The EMNIST dataset is an extension upon the common MNIST dataset that adds images of handwritten letters to the digits already present. There are 62 possible classes in the complete EMNIST dataset. However, some characters were removed due to an inability to distinguish between upper and lowercase characters. For use in this report, the dataset was split into training, validation and testing sets. All models were trained on the test set and hyperparameters selected using the validation set. Model performance could then be compared on the test set.

Chapter 2 discusses the development of baseline systems, Chapter 3 compares the RMSProp and Adam learning algorithms, Chapter 4 discusses cosine annealing and it's effect on Adam and SGD, and Chapter 5 compares weight decay and $L2$ regularisation on the Adam algorithm. The work is then concluded in Chapter 6.

2. Baseline systems

It is important to develop an initial baseline system to be used as a point of comparison for any experiments conducted. In order to find a suitable baseline, some experimentation must be performed to ascertain hyperparameter settings. The system used was a simple multilayer perceptron (MLP) using Stochastic Gradient Descent (SGD) and ReLu activation functions. This system offers a very standard and easy to implement neural network. Experiments were then conducted to find a suitable learning rate and depth of the network, all using 100 epochs.

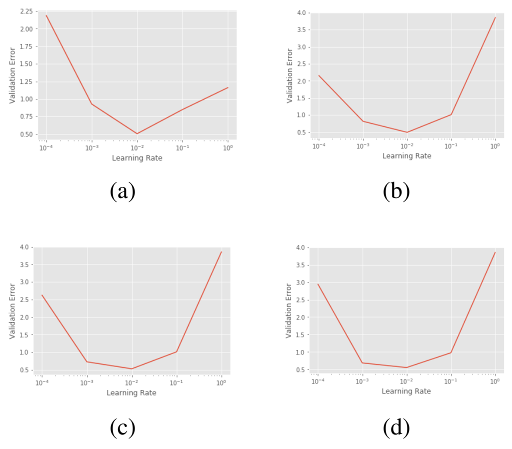


Figure 1. Validation error against change in learning rate. a = 2 layers b = 3 layers c = 4 layers d = 5 layers

To find a suitable value for the learning rate a grid search was performed across a range of learning rates (10^{-4} – 10^{0}) on the above model for a range of network depths (Figure 1). It is clear that a learning rate of 10^{-2} gives the best validation set performance on all of the tested models, and will, therefore, be used for the baseline model going forward. As the number of layers increases, there is less extreme variation of the learning rate. Consequently, if training time was an issue with the deeper models, the learning rate could be increased with only a small increase in error.

Using the learning rates for each network depth obtained above, these models were then compared on the test set to evaluate which offered the best performance. Table 1 shows the performance of each model on the train and test set. While performance on the training set continues to improve as the network depth is increased, the test performance begins to decrease after three layers. These results suggest that as the model complexity increases, it starts to overfit to the training data. Unnecessary increases in model complexity can often lead to overfitting, so we want to select the simplest model that performs best on the dataset (Goodfellow et al., 2016). In this case, the three-layered model will be used for the baseline model going forwards.

Network Depth	Training Error	Test Error
Two Layers	0.392	0.548
Three Layers	0.324	0.540
Four Layers	0.305	0.578
Five Layers	0.250	0.581

Table 1. Train and test error for a range of network depths with a learning rate of 0.01

3. Learning algorithms – RMSProp and Adam

RMSProp and Adam are two gradient descent based optimisation algorithms that attempt to scale their parameter updates over time automatically. RMSProp can be described as normalising the weight updates by the average of the squared gradient for a moving window of previous gradients. This approach differs from a similar, well-known algorithm, AdaGrad, which normalises the weight updates by the sum of the squared gradients for all previously seen gradient values. The introduction of a moving window with RMSProp allows the learning algorithm to take into account the previous gradient values, without being forced in directions that are no longer relevant. Importantly, this change no longer limits the effective learning rate to always decrease, as is the case for AdaGrad. By removing this restriction, the learning rate can be increased in cases where the gradient is small.

$$S_i(t) = \beta S_i(t-1) + (1-\beta)d_i(t)^2 \tag{1}$$

$$\Delta_{wi}(t) = \frac{-\eta}{\sqrt{S_i(t)}+\epsilon}d_i(t)^2 \tag{2}$$

Equation 1 details the moving window of the gradients, where β is the decay rate. Typically set to 0.9, β is used to indicate the extent to which previous gradients are used in the normalisation. Equation 2 shows how the gradient is updated, where η is the learning rate and ϵ is set to a small value ($\sim 10^{-8}$) to avoid division by zero.

Adam extends on the approach used for RMSProp by introducing a momentum term. Hinton, who proposed RMSProp, noted that momentum did not appear to have any significant effect on the RMSProp algorithm (Hinton et al., 2012). With Adam, Kingma & Ba (2014) attempted to address this issue of momentum by introducing a momentum-smoothed gradient to perform parameter updates.

$$M_i(t) = \alpha M_i(t-1) + (1-\alpha)d_i(t) \tag{3}$$

$$S_i(t) = \beta S_i(t-1) + (1-\beta)d_i(t)^2 \tag{4}$$

$$\Delta_{wi}(t) = \frac{-\eta}{\sqrt{S_i(t)}+\epsilon}M_i(t) \tag{5}$$

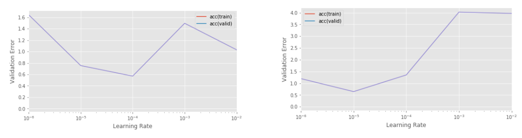


Figure 2. Validation error against change in learning rate. a = Adam b = RMSProp

Here, the gradient used previously for RMSProp is replaced by a momentum smoothed gradient, $M_i(t)$. The hyperparameters, α and β, are typically set to 0.9 and 0.999 respectively.

The default three-layer model described in Chapter 2 was tested with both RMSProp and Adam. Initially, a grid search was performed across both learning rules to find suitable learning rate values for each.

Figure 2 shows a plot of the validation error as the learning rate was adjusted. The plot shows optimal learning rate values are 10^{-4} and 10^{-5} for Adam and RMSProp respectively. The range of learning rate values used was determined by running a few short experiments to get an idea of some viable values. Values outside of the range used did not show promising results early into training. Some experimentation was also performed on the additional hyperparameters for Adam and RMSProp (β and α in equations 1, 6, and 4). However, it was clear that adjusting these from the default values had little to no impact on performance.

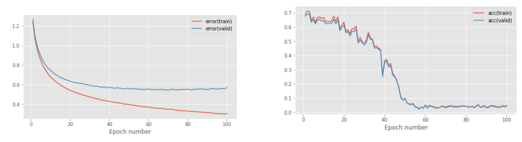


Figure 3. Learning curves for Adam (a) and RMSProp (b)

Figure 3 shows the learning curves for Adam and RMSProp using the learning rate values found in Figure 2, and the default values for all other hyperparameters. Looking at the curves, the difference between the two learning rules is clear. Adam has a very smooth learning curve, which notably contrasts with that of RMSProp. The difference between the two curves is most likely due to the additional momentum parameter used in Adam. The momentum parameter pushes the gradient in the direction of previous updates, which can have a smoothing effect.

Both models were then compared to the baseline model on the test set (Table 2). Using RMSProp resulted in a worse performance than the Baseline. However, Adam gave a slight improvement. This improvement suggests that gradient normalisation used in RMSProp is not enough to outperform SGD on this dataset, and the addition of a momentum term is necessary for improved results.

	Baseline	RMSProp	Adam
Test error:	0.54	0.605	0.531

Table 2. Comparison of test error for Adam and RMSProp

4. Cosine annealing learning rate scheduler

As seen in the previous chapters, selecting a suitable learning rate can be challenging, particularly if it is to be adjusted over time. One alternative to this is to use a learning rate scheduler. Learning rate schedulers define a procedure in which the learning rate is set and adjusted over time. When well defined, such an approach can be very effective. Loshchilov & Hutter (2017) introduce a scheduling algorithm, cosine annealing, in which the learning rate is decreased along a cosine curve. The learning rate may then be periodically increased before being decreased by the cosine curve again. This increasing is known as a warm restart. More formally, cosine annealing is given by

$$\eta(t) = \eta^{(i)}_{min} + 0.5(\eta^{(i)}_{max} - \eta^{(i)}_{min})(1 + cos(\pi T_{cur}/T_i)) \quad (6)$$

where η_{min} and η_{max} represent the minimum and maximum learning rate values. T_{cur} is the number of epochs performed since the last restart, and T_i is the number of epochs between restarts. Both T_i and η_{max} may be adjusted over time by multiplying each by a constant factor. Loshchilov & Hutter (2017) suggest that by doing so results in a "good anytime performance".

To test the performance of cosine annealing an initial hyperparameter search was performed. For both Adam and SGD, with and without warm restarts, the minimum and maximum learning rate and expansion factors were tested. The best performance for Adam was found with the following hyperparameters: $\eta_{max} = 10^{-4}, \eta_{min} = 10^{-6}, \eta_{maxmult} = 0.3$ using warm restarts. The warm restart hyperparameters were set to: $T_i = 25, T_{mult} = 3$. These hyperparameters gave an error value of 0.51.

The best performance for SGD was found with the following hyperparameters: $\eta_{max} = 10^{-2}, \eta_{min} = 10^{-3}, \eta_{maxmult} = 0.3$ using warm restarts. The warm restart hyperparameters were the same as those used for Adam. These hyperparemeters gave an error value of 0.46.

Both optimisation algorithms performed best with warm restarts; however, SGD slightly outperformed Adam. Both of the algorithms gave better performance when compared to the baseline and learning rules without a learning scheduler. However, Adam would be expected to outperform SGD. As discussed by Loshchilov & Hutter (2017), $L2$ regularisation is equivalent to weight decay for SGD, but this is not the case for Adam. Consequently, cosine annealing does not work equivalently for both algorithms, giving better performance for SGD. To correct for the disparity between algorithms a separate implementation of Adam with weight decay must be used.

5. Regularization and weight decay with Adam

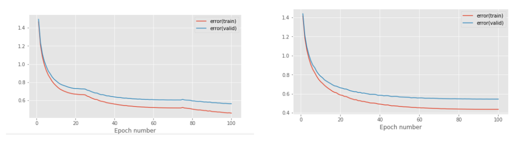
As briefly discussed above, $L2$ regularisation is equivalent to weight decay for SGD, however, for adaptive gradient algorithms (e.g. Adam), this is not the case. Adaptive gradient algorithms adapt both the loss function and regulariser for $L2$ regularisation, while they only adapt the loss function for weight decay. This disparity between $L2$ regularisation and weight decay can lead to weights with large gradients being regularised more when using weight decay than $L2$ regularisation. Therefore, the two approaches are not comparable in their standard forms.

Loshchilov & Hutter (2017)'s Adam with weight decay is equivalent to SGD with weight decay, allowing Adam with warm restarts to outperform SGD with warm restarts. Adam with $L2$ regularisation and weight decay were implemented and compared, showing the difference between the two empirically. After some hyperparameter searching, $L2$ regularisation reached an error value of 0.5 on the validation set with the regularisation parameter set to 10^{-4}. In contrast, weight decay reached an error value of 0.507, with the weight decay value set to 10^{-5}. The values are similar, however, when looking at the accuracy percentage, there is more of a distinction (84% vs 83%). This distinction is not as significant as that suggested in the original paper, indicating that further testing should be done to generate more conclusive results.

The cosine annealing and warm restarts previously discussed were then compared using Adam with weight decay. A comparison between a constant learning rate scheduler and the cosine annealing, both with warm restarts and using Adam with weight decay, found that the constant scheduler slightly outperformed the cosine annealing. The constant learning rate scheduler reached an error of 0.51 on the test set, while the cosine annealing reached an error of 0.563. Comparing these results to those using Adam without weight decay, weight decay has had little impact on performance. Perhaps simpler models are more effective on the EMNIST dataset.

Experiments were also run to evaluate the use of warm restarts. With warm restart parameters set to $T_i = 25, T_{mult} = 3$ and a weight decay value of 10^{-5}, an error of 0.563 was achieved on the test set. In comparison, the same model without warm restarts (i.e. $T_i = 100, T_{mult} = 1$) resulted in an error of 0.544. Again, these results are quite similar. However, they do suggest that cosine annealing without warm restarts offers slightly better performance. Figure 4 shows a comparison of learning curves with and without warm restarts. While the curves are similar, the warm restarts create a much less smooth curve, which may impact training performance.

Table 3 shows a comparison of the train and test error for the above experiments. As discussed, the test performance for Adam with $L2$ regularisation is better than weight decay. Table 3 gives a more detailed view, demonstrating that weight decay performs significantly better on the training



(a) (b)

Figure 4. Learning curves for Adam with warm restarts (a) and no warm restarts (b)

	Train Error	Test Error
Adam L2	0.312	0.5
AdamW	0.263	0.507
AdamW (const scheduler)	0.378	0.51
AdamW (cos anneal no restart)	0.437	0.544
AdamWR	0.46	0.563

Table 3. Comparison of performance for Adam variants

data. However, it is not able to generalise as effectively. These results indicate that $L2$ regularisation may offer better generalisation performance than weight decay, yet further experimentation is necessary.

6. Conclusions

This report discussed and compared multiple learning algorithms, with an emphasis on the impacts of cosine annealing and weight decay on the Adam algorithm. The differences between the RMSProp and Adam algorithm algorithms were discussed, and it was shown that Adam outperformed RMSProp and the baseline SGD model. The effect of cosine annealing was also examined, and it was found that SGD outperformed Adam, empirically showing that weight decay and $L2$ regularisation are not equivalent for Adam, while they are for SGD. This effect was then examined throughout Chapter 5, and am adapted Adam algorithm using weight decay was discussed that is equivalent to SGD with weight decay.

The best performance on the EMNIST dataset proved to be SGD using cosine annealing and warm restarts. This model achieved an error value of 0.46, with $\eta_{max} = 10^{-2}, \eta_{min} = 10^{-3} and \eta_{maxmult} = 0.3$. Given the discussed addition of weight decay to Adam, it should be able to outperform SGD's performance here.

Further work may include confirming the improvement of Adam with weight decay over SGD and running additional experiments to improve understanding of the impacts of weight decay and $L2$ regularisation on Adam. Additionally, this work could be expanded to different test cases, offering a greater understanding of its implications.

References

Cohen, Gregory, Afshar, Saeed, Tapson, Jonathan, and van Schaik, André. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

Goodfellow, Ian, Bengio, Yoshua, Courville, Aaron, and Bengio, Yoshua. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Hinton, Geoffrey, Srivastava, Nitish, and Swersky, Kevin. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, pp. 14, 2012.

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

Loshchilov, Ilya and Hutter, Frank. Fixing weight decay regularization in Adam. *arXiv preprint arXiv:1711.05101*, 2017. URL https://arxiv.org/abs/1711.05101.