MLP Coursework 1: Learning Algorithms and Regularization

S1875880

Abstract

The purpose of this coursework was to implement and explore RMSProp and Adam learning algorithms. Initially a stochastic gradient descent (SGD) system was set as a baseline. Then, the two mentioned learning algorithms were implemented and compared with the SGD system. A scheduler was implemented which sets the learning rates of SGD and Adam using warm restarts and a cosine function. Finally, the Adam optimizer was extended with L2 regularization and weight decay. For all algorithms the hyperparameters were first optimized. The models were evaluated on the Balanced EMNIST dataset. To compare the results of each algorithm, I used figures and tables. It is shown that Adam with with weight decay and restarts has the highest performance.

1. Introduction

The motivation behind this work was a recent paper by Loshchilov & Hutter (2017), were they used L2 regularization and weight decay on Adam optimizer. The authors propose a modification on the weight decay regularization, which results in better generalization performance for Adam. In addition, they proposed 'AdamWR', a version of Adam with warm restarts which improves its anytime performance.

The structure of the report follows the order of the implementations and experiments. In section 2, the SGD baseline system is described. In section 3 the implementation of RMSProb and Adam is described and then the accuracy and learning curve of RMSProp and Adam are compared with SGD. Section 4 shows the implementation of cosine scheduling algorithm which is then applied to SGD and Adam. Finally, the L2 regularization and weight decay for the Adam is explored in section 5.

The dataset used in all experiments is the EMNIST Balanced dataset Gregory Cohen & van Schaik (2017). The EMNIST dataset extents the MNIST dataset which was derived from a larger dataset called NIST Special Database 19 Grother (1995). The EMNIST dataset not only preserves the digit images, as MNIST does, but also the lower case letters and upper case letters. Each image is 28x28 and there are 62 classes in the EMNIST dataset. For simplicity, 15 letters that are hard to distinguish between lower and uppercase were merged into a single label for both cases. The result is a 47-class dataset, which is more challenging for classification tasks compared to MNIST.

The experiments were executed on a machine with 3.1 GHz i7-6700HQ quad core CPU, 16GB RAM, Windows 10 OS.

2. Baseline system

The baseline system is established with just stochastic gradient descent (SGD) on the EMNIST dataset. The performance of SGD was evaluated with 2-5 hidden layers with 100 ReLU hidden units each.

The purpose of the **first experiment** is to find an optimal learning rate using the validation set. The number of epochs (100), number of hidden units (100) and the number of hidden layers (2) are held constant, while I was tuning the learning rate. The table 1 shows that the higher the learning rate the less accurate is the model and the higher the error.

Learn Rate	Err(valid)	Acc(valid)
0.5	1.03	0.80
0.2	1.00	0.82
0.1	0.84	0.82
0.05	0.65	0.83

Table 1. SGD Baseline - 2 Hidden Layers

For the **second experiment**, the optimal learning rate was found for the number of hidden layers (2-5) using the validation set. The conclusion is that the more layers the network has, the lower the learning rate should be. The same systems were then evaluated on the testing set as shown in table 2 .

Hidden Layers	Learn Rate	Acc(valid)	Acc(test)
2	0.05	0.83	0.81
3	0.02	0.84	0.82
4	0.02	0.83	0.83
5	0.01	0.84	0.83

Table 2. SGD Baseline - 2-5 Layer Models

The purpose of trying out different hyperparameters is to use them to maximize the accuracy of the model. For the next experiments, 3 hidden layers will be used.

3. Learning algorithms – RMSProp and Adam

In this section the RMSProp and Adam with gradient descent algorithms are implemented as shown with the pseudocodes 1 and 2. RMSprop and Adam are both adaptive learning rate methods.

Algorithm 1 RMSProp

given: learning rate 1e-3, beta 0.9, epsilon 1e-8
initialize: mean of squared gradients <- 0
repeat
  for w, mean sq grad, grad **to** weights, mean of sq grads, grads wrt params **do**
    mean sq grad <- mean sq grad * beta
    mean sq grad <- mean sq grad+(1-beta)*(grad**2)
    w <- w-(learning rate * grad)/((mean sq grad + epsilon)**0.5)
  end for
until stopping criterion is met
return optimized parameters w

Algorithm 2 Adam

given: learning rate 1e-3, beta1 0.9, beta2=0.999 epsilon 1e-8
initialize: first moment vector moms1 <- 0, second moment vector moms2 <- 0, step count t <- 0
repeat
  for w, mom1, mom2, grad **to** weights, moms1, moms2, grads wrt params **do**
    t <- t + 1
    mom1 <- mom1 * beta1
    mom1 <- mom1 + (1 - beta1) * grad
    mom1hat <- (mom1)/(1 - beta1**t)
    mom2 = mom2 *beta2
    mom2 = mom2 + (1 - beta2) * (grad**2)
    mom2hat = (mom2)/(1 - beta2**t)
    w<- w-(learning rate * mom1hat) / ((mom2hat + epsilon)**0.5)
  end for
until stopping criterion is met
return optimized parameters w

RMSprop was proposed by Tieleman & Hinton (2012) in Hinton's Coursera lecture and it has not been published. RMSprop is an extension of Rprop that works with mini-batches. For RMSProp a moving average of the squared gradient for each weight is kept. The gradient is divided by this moving average.

Adam was proposed by (Kingma & Ba, 2015). It is similar with RMSprop but with momentum. Adam keeps decaying averages of past gradient and past squared gradient. During the first steps these averages are biased to zero. To correct this, the "hat" of first and second moment are calculated, as shown in the psudocode.

For RMSProp and Adam the chosen learning rate is 0.0001 and for SGD 0.02. RMSProp was overfitting with 0.001 learning rate. The learning curves for RMSprop, Adam and SGD for 100 epochs on the validation set are shown in figures 1 , 2, 3 .

Figure 1. RMSprop Error-Epoch Curve. Learning rate: 0.0001

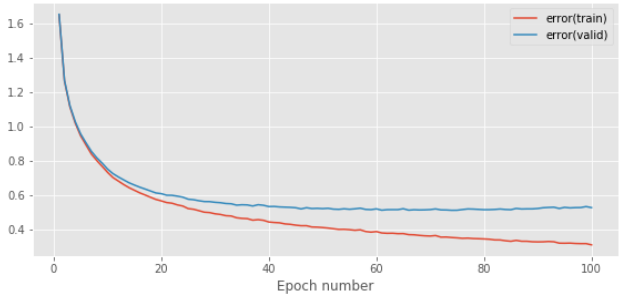


Figure 2. Adam Error-Epoch Curve. Learning rate: 0.0001

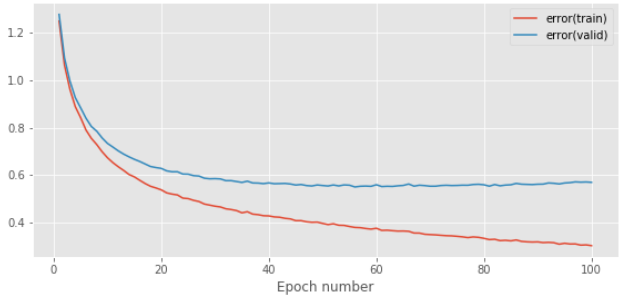
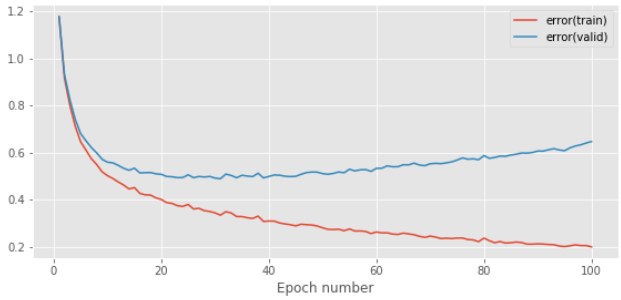


Figure 3. SGD Error-Epoch Curve. Learning rate: 0.02



	SGD	RMSprop	Adam
Acc(test):	0.823	0.824	0.824

Table 3. SGD, RMSprop, Adam test set accuracy

Table 3 shows the accuracy of SGD, RMSpop and Adam on the test set.

The Error-Epoch graphs show how the validation error (blue) and train error (red) change over each epoch. When the validation error starts to increase while the train error continues to decrease, the model starts to overfit. It is much more noticeable in the SGD. This could be avoided with learning rate schedulers, discussed in the next section.

4. Cosine annealing learning rate scheduler

Learning rate schedulers are used to set the learning rate and are responsible for updating it every batch or epoch. The scheduler implemented in this coursework is called cosine annealing. The learning rate is decreased periodically and then restarts to an increased value. This scheduler is proposed by (Loshchilov & Hutter, 2017). The pseudocode for the cosine annealing learning rate scheduler is shown in pseudocode 3.

Algorithm 3 Cosine Annealing Learning Rate Scheduler

given: learning rate, epoch number, min learning rate, max learning rate, total epochs per period, discount, expansion
initialize: next restart tNext <- total epochs per period, period expansion tPeriod<- total epochs per period, epoch since last restart tCur <- 0, batch iteration t <- 0
tCur <- epoch number % total epochs per period
t <- int(epoch number/total epochs per period)
effective max learning rate = max learning rate
for i **to** range(t) **do**
  effective max learning rate <- effective max learning rate * discount
end for
if epoch number == tNext **then**
  tPeriod <- tPeriod * expansion
  tNext <- tNext + tPeriod
  t <- t + 1
  tCur <- 0
  restart learning rule
end if
h <- min learning rate + 0.5*(effective max learning rate - min learning rate)*(1-cos(pi*(tCur/total epochs per period)))
tCur <- tCur + 1
return effective learning rate coefficient h

This approach works for any given starting epoch. First, to find the epoch since last restart, the given epoch, modulo, the total epoch per period is calculated. To find the current batch iteration number, we use the integer of the epoch number divided by the total epochs per period. The effective max learning rate is calculated by iterating over the each batch up to that point, and keeping the product of the discount factor. To implement the "warm restart" we keep a variable tNext which indicates the next epoch to restart. tNext is calculated by expanding the period and adding it to the previous tNext. When the epoch number is equal to tNext, the learning rate restarts. Finally, the learning rate coefficient h is calculated and returned as shown in the pseudocode.

The scheduler was tested on both Adam and SGD. For each, a baseline system was evaluated with constant learning rate 0.01. The scheduler was tested on both algorithms with and without restarts. All tests had a 'budget' of 100 epochs, minimum learning rate 0.0001, maximum learning rate 0.01 and learning rate discount factor 0.9. The restart-test had

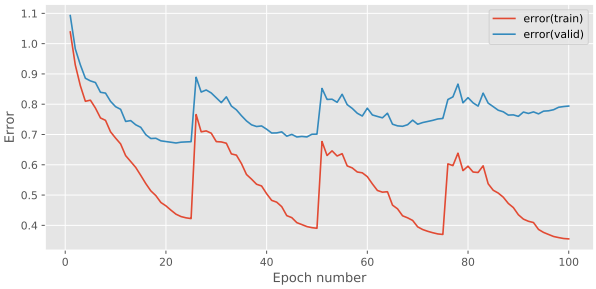


Figure 4. Adam learning curve with warm restarts every 25 epochs.

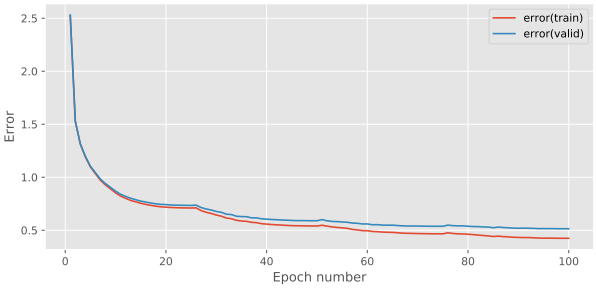


Figure 5. SGD learning curve with warm restarts every 25 epochs.

25 epochs per period and expansion factor of 3. Table 4 shows the results for all tests. The Adam results were as expected, it performs best with the scheduler and warm restarts. The results of SGD are similar for all tests. This may be because of the chosen learning rate (0.01) for the baseline was the optimal.

Model	Scheduler	Acc(valid)	Error(valid)
Adam	No (Baseline)	0.747	0.947
Adam	Yes w/o restarts	0.795	0.959
Adam	Yes w/ restarts	0.802	0.794
SGD	No (Baseline)	0.837	0.502
SGD	Yes w/o restarts	0.837	0.506
SGD	Yes w/ restarts	0.835	0.514

Table 4. Performance of Adam and SGD with Learning Rate Scheduler

5. Regularization and weight decay with Adam

In this section of the coursework the L2 regularization and weight decay were implemented in the Adam algorithm. The implementation separates the regularization hyperparamter and the learning rate hyperparameter(s) as proposed by (Loshchilov & Hutter, 2017). The weight decay hyperparmater is independent of the learning rate hyperparameter(s).

The implementation of equation in line 12 of algorithm 2 in the paper of (Loshchilov & Hutter, 2017) is shown in psudocode 4. The Adam with Weight Decay Learning Rule Functionality Test was passed.

Algorithm 4 Line 12 of algorithm 2 of (Loshchilov & Hutter, 2017)

given: learning rate <- lr, initial learning rate <- ilr, epsilon <- e, weight decay <- wd
param -= (lr/ilr) * (ilr*mom1hat/sqrt(mom2hat + e) + wd*param)
return optimized parameters

Three different experiments were carried out on regularization and weight decay with Adam.

- L2 regularization vs. weight decay
- constant learning rate vs. cosine annealing schedule
- no restarts in the scheduler vs. use of a warm restart

L2 regularization vs. weight decay.

For both of these test the learning rate was 0.01 and the model was trained for 100 epochs. As shown in the algorithm 4, the weight decay is implemented with the '+wd*param'. To test the L2 regularization, that part of the equation was removed and L2Penalty was applied to the input AffineLayer of the model, with coefficient 0.0001. The model with L2 regularization took 45 minutes to train, 5 times more than the one with weight decay.

Model	Acc(test)	Error(test)
Adam with Weight Decay	0.761	0.921
Adam with L2 regularization	0.742	0.873

Table 5. Performance of Adam with weight decay and L2 regularization.

Adam with weight decay has higher accuracy than L2 regularization. Figures 6 and 7 show the lurning curves on the validation set for Adam with weight decay and L2 Regularization respectively.

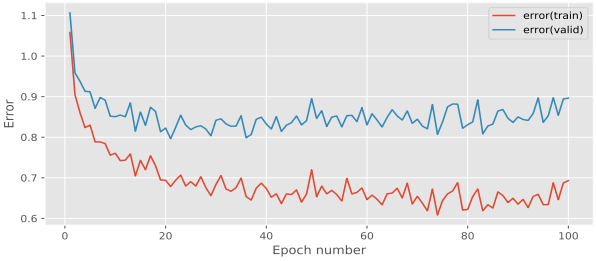


Figure 6. Adam learning curve with Weight Decay.

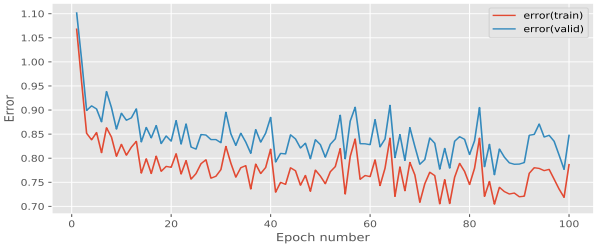


Figure 7. Adam learning curve with L2 Regularization.

Constant Learning Rate vs. Cosine Annealing Schedule. In this test, Adam algorithm is compared with constant learning rate and cosine annealing scheduler. The constant learning rate is set to 0.001. The initial learning rate for Cosine Annealing Schedule is again 0.001 but the scheduler can decrease it down to 0.0001. The table 6 shows the results on the test set.

Figures 8 and 9 show the learning curves.

Learning Rate	Acc(test)	Error(test)
Constant Learning Rate	0.816	0.916
Cosine Annealing Schedule	0.817	1.12

Table 6. Performance of Adam with weight decay: Constant Learning Rate vs Cosine Annealing Schedule.

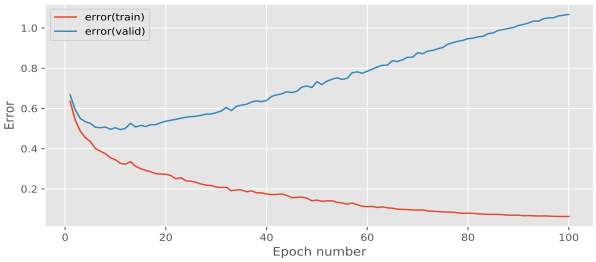


Figure 9. Adam learning with weight decay and Cosine Annealing Scheduler.

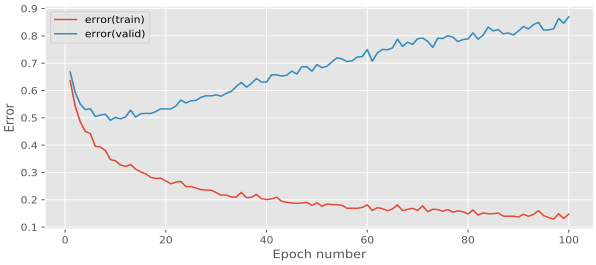


Figure 8. Adam with weight decay and constant learning rate.

No restarts in the scheduler vs. use of a warm restart. The purpose of this test is to see if a learning rate 'warm restart' on the Cosine Annealing Scheduler for Adam with Weight Decay impacts the output. Table 7 shows that a greater accuracy and lower error was achieved on the model with a restart.

Scheduler	Acc(test)	Error(test)
No Restart	0.817	1.12
Restart on 50th Epoch	0.822	0.997

Table 7. Performance of Adam with weight decay and Cosine Annealing Schedule: No Restarts vs Restart on the 50th epoch.

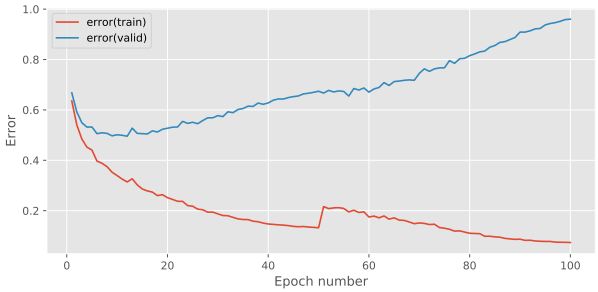


Figure 10. Adam with weight decay and Cosine Annealing Scheduler with restart at 50th epoch.

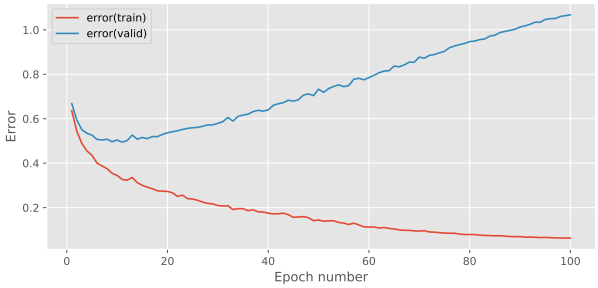


Figure 11. Adam with weight decay and Cosine Annealing Scheduler without restarts.

6. Conclusions

We initially implemented a baseline system with Stochastic Gradient Descent (SGD), which achieved an accuracy of 0.823 on the test set. Then, we implemented the RMSprop and Adam algorithms with gradient decent. RMSprop achieved 0.824 and Adam 0.824 on the test set. Later we implemented a function that adjusts the learning rate with a cosine function and 'warm restarts'. This function was tested on Adam and SGD models. Adam showed improved performance with this function (from 0.747 to 0.802 accuracy), but SGD did not improve as much. Finally, we implemented weight decay and L2 regularization for the Adam optimizer. Adam performed better with weight decay and cosine annealing schedule with warm restarts, achieving accuracy of 0.822 on the test set.

References

Gregory Cohen, Saeed Afshar, Jonathan Tapson and van Schaik, AndrÃl'. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017. URL https://arxiv.org/abs/1702.05373.

Grother, P. Nist special database 19 handprinted forms and characters database. *National Institute of Standards and Technology, Tech*, 1995.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *ICML*, 2015. URL https://arxiv.org/abs/1412.6980.

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

Tieleman, Tijmen and Hinton, Geoffrey E. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.