MLP Coursework 1: Learning Algorithms and Regularization

s1879797

Abstract

This paper presents details about the implementation of a few well-known Gradient Descent optimizations, the results of the experiments made and some opinions about these results.

1. Introduction

The paper will start with the presentation of Stochastic Gradient Descent algorithm and some experiments which were made using this algorithm in order to determine the best model. Two other algorithms (RMSprop and Adam) will be presented and the relation between these two and Stochastic Gradient Descent will be highlighted. How these two algorithms can bring an improvement and what limitations they have, how cosine annealing method can make the curve of points to converge faster and how to keep the weights in check using regulation are other subjects that will be discussed bellow.

2. Baseline systems

In order to establish a baseline system on EMINIST I had to carry experiments using Stochastic Gradient Descent (SGD) with the following configuration: 100 ReLU hidden units per layer and 2-5 hidden layers.

I chose to split the models in two categories: models tested over 100 data points (batch size) and 100 epochs, and models tested over 200 data points (batch size) and 150 epochs. The range of the learning rates used in these experiments was wide enough to provide underfitting and also overfitting: $\{0.0001, 0.001, 0.01, 0.015, 0.1, 0.2, 0.5\}$. I chose not to use learning rates bigger than the ones already selected because in that case the model will have a very high change to overfit (this statement is provided based on my previous experiments training stochastic gradient models). After the experiments finished I realized that 0.2, 0.5 are also very big values for the learning rate. Even 0.1 was a bad learning rate value, but compared with the other two was still acceptable.

In total, I trained 56 models (4 different categories based on the number of hidden layers, 2 different categories based on the batch size and number of epochs, and 7 categories based on the learning rate values).

The models were trained using the training data set provided and then the best 8 models (one for each number of hidden layers and data sizes) based on the validation set. After that

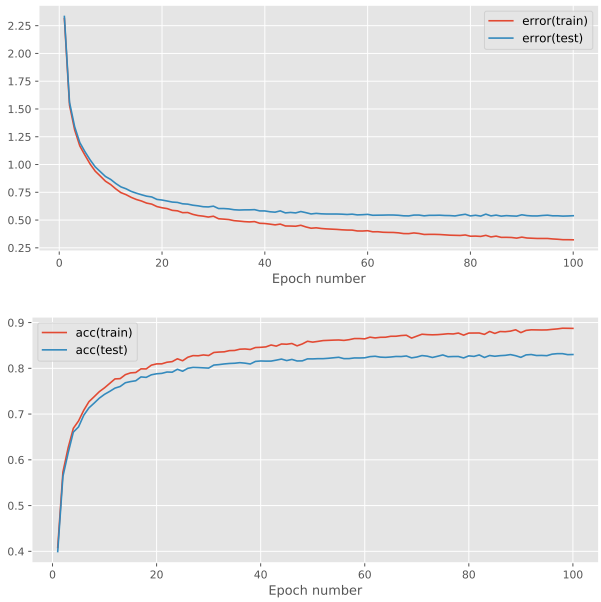

Figure 1. Error and accuracy results for the best Stochastic Gradient Descent model

I tested these 8 models in order to find the best one using the test set. The results are presented in table 1.

As we can see from the results the best model is number 3 (figure 1) because has the best accuracy rate and pretty good (not the best) runtime. When I chose the best model I tried to do this keeping a good ratio between accuracy value and the average runtime for an epoch because when you want to choose a model you cannot spend huge time to train it, even if it has the best accuracy value. Another example for a good model is number 6. Some people might consider it better than number 3 because they have almost the same accuracy, but the runtime is better for number 6. Still, I considered number 3 better as a personal choice, because for me the accuracy was more important than that difference in the runtime.

I could have chosen to find the best model using "early stopping", but I chose to spend more time on training the models in order to see the representations for each one of them and to better understand the error and accuracy values.

	Batch Size	Num Layers	Learning Rate	Acc Train	Acc Test	Runtime ($\frac{s}{epoch}$)
1	100	2	10^{-2}	0.873	0.828	4.17
2	100	3	10^{-2}	0.887	0.830	4.43
3	100	4	10^{-3}	0.783	0.761	4.51
4	100	5	10^{-3}	0.797	0.778	4.83
5	200	2	10^{-2}	0.863	0.821	3.13
6	200	3	10^{-2}	0.878	0.829	3.46
7	200	4	10^{-3}	0.755	0.734	3.82
8	200	5	10^{-3}	0.770	0.754	6.98

Table 1. The best 8 models using Stochastic Gradient Descent

Learning Rate	Decay Rate	Err Train	Err Test	Acc Train	Acc Test
10^{-5}	0.9	0.588	0.670	0.82	0.80

Table 2. The best model using RMSprop

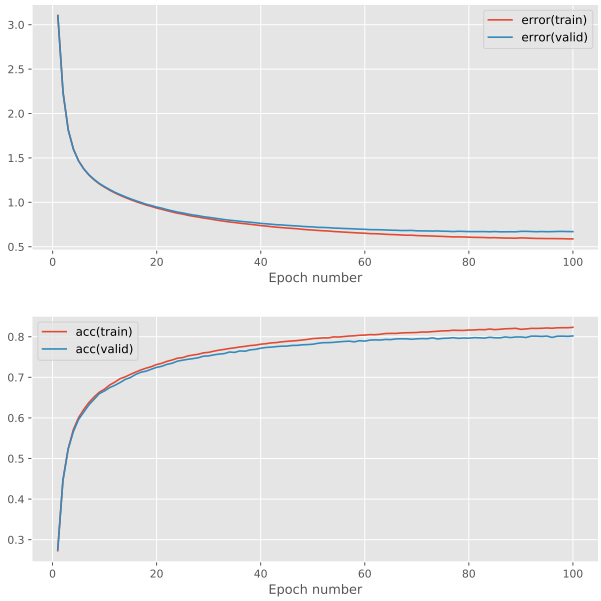

Figure 2. Error and accuracy results for the best RMSprop model

3. Learning algorithms – RMSProp and Adam

I have completed the rules for RMSprop and Adam algorithms in the **mlp.learning_rules** module using the information provided in lecture 5 and the two additional documents provided in the documentation (Tieleman & Hinton (2012), Kingma & Ba (2015)). Besides these, I read other documents because I also wanted to find more information about how momentum influences the learning rate. I checked Ruder (2016) where I found enough information in order to understand the update of the learning rate for both RMSProp and Adam. Furthermore, I could see the evolution from Stochastic Gradient Descent to Adagrad, and then to RMSprop, and finally, to Adam. The best explanation that I found about the two algorithms to help understand and implement them was in Karpathy (2018).

Here, the experiments were performed using 100 data points in order to match the model for Stochastic Gradient Descent.

For RMSprop I did experiments on 30 models choosing values for the learning rate from $\{0.000001, 0.0000015, 0.00001, 0.000015, 0.0001, 0.00015\}$ and for decay rate from $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. The number of hidden layers was 3 (as specified). After training the models using the training dataset from EMINIST, I selected the best model (according to the accuracy-error relation) using the validation set from EMINIST.

The characteristics of the best model are presented in table 2 and the representation in figure 2.

For Adam I had to do more experiments than for RMSprop because Adam uses two decay rates in order to calculate the momentum and average of past squared gradients. This time, because of the bad results obtained for RMSprop using the last two learning rates from the set, I assumed that for Adam I will also obtain bad results using those learning rates. Therefore, the experiments were made using learning rates from $\{0.000001, 0.0000015, 0.00001, 0.000015\}$, values for the first decay rate from $\{0.85, 0.9, 0.95\}$ and values for the second decay rate from $\{0.95, 0.975, 0.999\}$. I was right assuming that the results for Adam using the learning rates $\{0.0001, 0.00015\}$ will be bad because after I finished these experiments I noticed that the bigger learning rate used produced the worst results and with increasing learning rate, error increases and accuracy decreases. I was not satisfied with the experiments I did, therefore, I did four more experiments keeping the values used for learning rate and using 0.99 for the first decay rate and 0.999 for the second decay rate.

Learning Rate	Decay Rate 1	Decay Rate 2	Err Train	Err Test	Acc Train	Acc Test
$1.5*10^{-5}$	0.9	0.95	0.629	0.666	0.81	0.80

Table 3. The best model using Adam

After I finished my 40 experiments using the Adam algorithm, based on the accuracy-error ratio, I was able to choose the best model. And the results are presented in table 3 and figure 3.

The most obvious difference between these two algorithms and Stochastic Gradient Descent is how the weights are updated. RMSprop computes an exponential average of the square of the gradient in order for the most recent gradient updates to weight more than the less recent

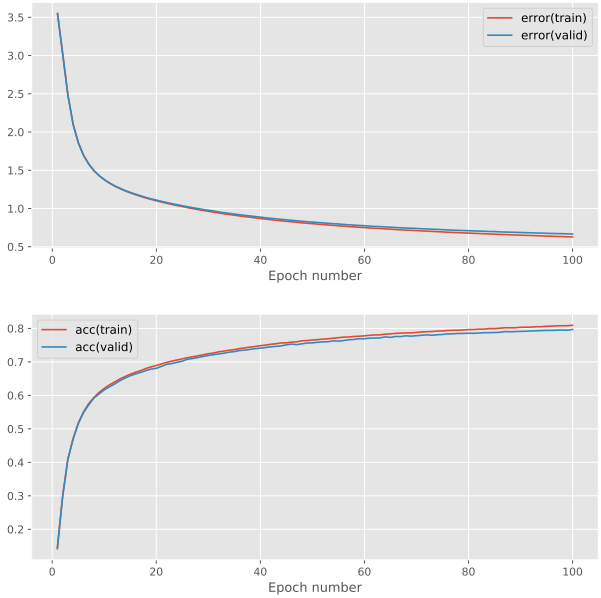

Figure 3. Error and accuracy results for the best Adam model

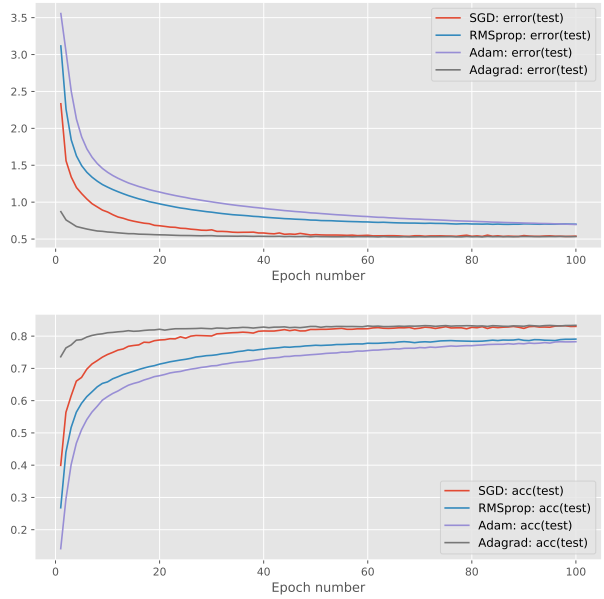

Figure 4. Comparison: SGD vs. AdaGrad vs. RMSprop vs. Adam

Model	Learning Rate	Err Test	Acc Test	Runtime ($\frac{s}{epoch}$)
SGD	0.01	0.539	0.830	4.97
AdaGrad	0.01	0.534	0.833	6.45
RMSprop	10^{-5}	0.701	0.791	6.34
Adam	$1.5*10^{-5}$	0.698	0.783	8.53

Table 4. Comparison results

ones in the update operation. I can think of Adam like an improvement of both Stochastic Gradient Descent and RMSprop (RMSprop it is an improvement of SGD). Adam computes adaptive learning rates for each parameter exponentially decaying average of past squared gradients v_t and average of past gradients m_t (Ruder, 2016).

Using the test dataset I compared the best model of SGD, the best model of RMSprop and the best model of Adam (I also added the default model of AdaGrad). Adam should produce the best results because, after all, it is an improvement of RMSprop which is suppose to be better than Stochastic Gradient Descent because of the adaptative learning rate. The results are presented in table 4 and figure 4.

The results were quite surprising because Adam and RMSprop, which were supposed to provide the best results, produced the worst results. I am confused because I do not know what the problem is. I rechecked the algorithms and the implementation is done correctly. I think that for some tasks the algorithms could produce bad results as it is stated in Mack (2018).

4. Cosine annealing learning rate scheduler

I have implemented the cosine annealing learning rate scheduler in **mlp.scheduler** according to the mathematical formula for updating learning rate presented in Loshchilov & Hutter (2017a). The cosine annealing method is used to quickly decrease the learning rate (to increase the probability of finding the minimum) and periodically increase it (to keep a fast convergence rate). In order to implement the method I had to follow algorithm 1 and to look over Loshchilov & Hutter (2017b) because there is shown how cosine annealing was applied for the first time over Stochastic Gradient Descent which helped me understand why and how it should be apply it over Adam.

I already had the results for Adam and Stochastic Gradient Descent using a fixed learning rate and no scheduler because I calculated them before. In order to obtain the results for cosine annealing with no restarts and cosine annealing with restarts ($T_i = 25$ and $T_{mul} = 3$) for both Adam and Stochastic Gradient Descent I used the training set to train the models and the validation set to choose the best ones. I have trained 36 models because I variate min_learning_rate with values: $\{0, 0.0001, 0.001\}$, and max_learning_rate with values: $\{0.01, 015, 0, 1\}$.

After the experiments I could choose the best model for each category, but although Stochastic Gradient Descent algorithm had a very good accuracy and a nice plot, on the other side, Adam produced very bad results. The models selected for testing are displayed in table 5. Even after I used the test set to plot final results for the models previously chosen the results displayed by the Adam algorithm when using cosine annealing method remain very bad (as we can see in figure 5).

Algorithm 1 Cosine Annealing Update Function

multiplier = 0
if *epoch* − *previous_epoch* > 1 **then**
  if *last_restart* + T_i <= *epoch* **then**
    while *last_restart* + T_i <= *epoch* **do**
      $\eta_{max} = \eta_{max} * discount\_factor$
      $T_i = T_i * T_{mul}$
      $last\_restart = last\_restart + T_i$
    end while
    $T_{cur} = epoch - last\_restart$
    $multiplier = \eta_{min} + 0.5(\eta_{max} - \eta_{min})(1 + \cos(\pi \frac{T_{cur}}{T_i}))$
  end if
else
  if $T_{cur} == (T_i - 1)$ **then**
    $T_{cur} = 0$
    $\eta_{max} = \eta_{max} * discount\_factor$
    $T_i = T_i * T_{mul}$
    $last\_restart = last\_restart + T_i$
  else
    $T_{cur} = epoch - last_restart$
  end if
  $multiplier = \eta_{min} + 0.5(\eta_{max} - \eta_{min})(1 + \cos(\pi \frac{T_{cur}}{T_i}))$
end if
previous_epoch = *epoch*
learning_rule.learning_rate = *multiplier*
return *learning_rule.learning_rate*

Model	Min Learning Rate	Max Learning Rate
SGD (no restarts)	0.0001	0.01
Adam (no restarts)	0.0001	0.01
SGD (with restarts)	0.0001	0.01
Adam (with restarts)	0.0001	0.1

Table 5. Comparison results using cosine annealing

5. Regularization and weight decay with Adam

Although, for Stochastic Gradient Descent L2 Regularization and Weight Decay have the same meaning, for Adam they are two different regularization: for L2 regularization all the weights are regularized with the same amount, while for weight decay the regularization is performed using the same factor which gives a uniform adjustment as stated in Loshchilov & Hutter (2017a).

I performed multiple experiments in order to display the opposition between Adam with Weight Decay and Adam using L2 Regularization. The experiments were made using weight decay hyperparameter with values from $\{0.00001, 0.0001\}$. The results of these experiments can be seen in table 6 and the graphic representation in figure 6

As we can see L2 Regularization is slightly better that Weight Decay, but the difference in runtime is very big. In opinion, I will choose the Weight Decay model over the L2 Regularization.

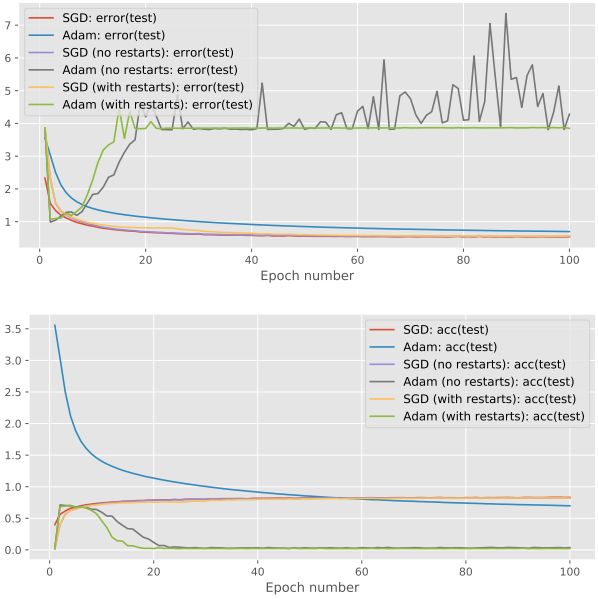

Figure 5. Cosine annealing method: results

Model	Decay Rate	ERR TEST	ACC TEST	Runtime ($\frac{s}{epoch}$)
Adam Weight Decay	0.00001	0.844	0.755	7.07
Adam L2 Regularization	0.0001	0.685	0.785	17.05

Table 6. Weight Decay vs. L2 Regularization

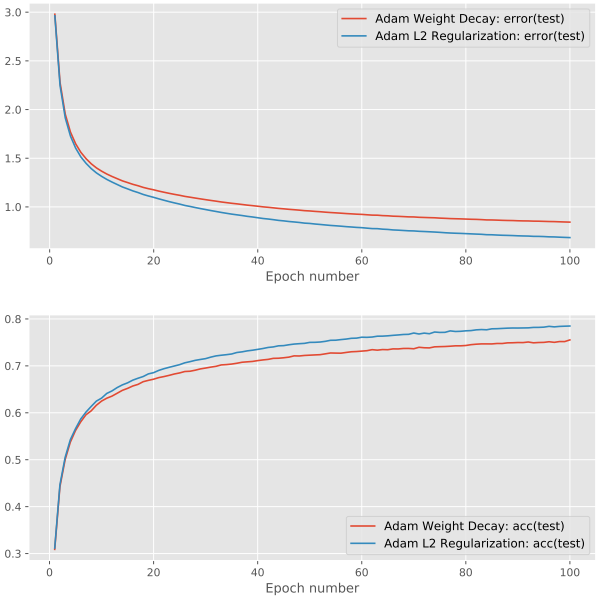

Figure 6. Weight Decay vs. L2 Regularization

I also did another comparison involving three models: Adam with Weight Decay, Adam with Weight Decay

Model	Decay Rate	ERR TEST	ACC TEST	Runtime ($\frac{s}{epoch}$)
Adam Weight Decay	0.00001	0.844	0.755	7.07
Adam No Restart	0.00001	4.52	0.022	18.58
Adam Warm Restart	0.00001	4.39	0.022	18.10

Table 7. Adam Weight Decay vs. Adam Weight Decay No Restarts vs. Adam Weight Decay Warm Restart

and Scheduler with no restarts and Adam with Weight Decay and Scheduler with warm restart (I used the hyperparameters from a previous example: $T_i = 25$ and $T_{mul} = 3$). The comparation is displayed in figure 7 and the result are in table 7.

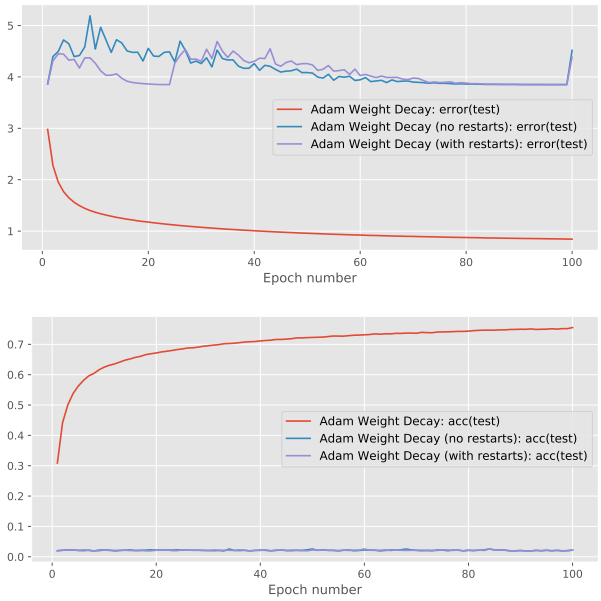

Figure 7. Adam Weight Decay vs. Adam Weight Decay No Restarts vs. Adam Weight Decay Warm Restart

6. Conclusions

Some clarification that maybe I did not put in the paper is that every time when I had to find the ideal hyperparameters for the best model I used grid search method and I assume values for testing based on my own assumptions or similar to some example from documentation.

In conclusion, after I finished all the experiments I can say that the results were not quite what I was expecting and I am not sure it something was wrong with my implementation or this is the best of what these algorithms can come up with (maybe I need to use other optimization algorithms/methods).

References

Karpathy, Andrej. Cs231n: Convolutional neural networks for visual recognition (stamford university), 2018. URL http://cs231n.github.io/neural-networks-3/.

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

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

Loshchilov, Ilya and Hutter, Frank. Sgdr: Stochastic gradient descent with warm restarts. 2017b. URL https://arxiv.org/pdf/1608.03983.pdf.

Mack, David. How to pick the best learning rate for your machine learning project, 2018. URL https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2.

Ruder, Sebastian. An overview of gradient descent optimization algorithms, 2016. URL http://ruder.io/optimizing-gradient-descent/.

Tieleman, Tijmen and Hinton, Geoffrey E. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.