Machine Learning Practical Assignment 1 Report

Jin Xu
Student No. 1673820
Exam No. B096012

Contents

1 Introduction

Many successful machine learning algorithms boil down to the optimisation of the error function in the end. Stochastic gradient descent gives us a practical scheme regardless of the error function formula. However, different learning rate schedulers and parameter updating rules tend to have significant effects on the algorithm performance, in terms of convergence speed, generalisation and accuracy. In this report, several widely used learning rate schedulers and parameter updating rules are investigated and compared under different measures.

1.1 Learning Rate Schedules

To optimise the error function $E(w)$, we calculate the partial derivative $\frac{\partial E}{\partial w_i}$ with respect to every parameter w_i, and denote it as D_i. So the basic idea of gradient descent is that, at every updating time t, we calculate all the $D_i(t)$, and the update for each w_i will be:

$$w_i(t) = w_i(t-1) + \Delta w_i(t) \quad where \quad \Delta w_i(t) = -\eta_i(t) D_i(t) \tag{1}$$

Where $\eta_i(t)$ is called learning rate. If you are not using an adaptive learning rule, then $\eta_i(t) = \eta(t)$ for each i. The simplest policy for learning rate is that we use a constant scheduling, fixing the $\eta(t)$ to a constant η. However, sometimes, we want our learning rate to be big at the beginning, in the purpose of broader exploration and faster convergence, and we want our learning rate to be small in the end, tuning the parameters carefully near convergence. We can achieve this by using a time-dependent scheduling. In this report, we use an exponential scheduling to compare to constant scheduling. The learning rate now is defined as:

$$\eta(t) = \eta(0) e^{-\frac{t}{r}} \tag{2}$$

The decay rate r controls the decay speed of the learning rate, and $\eta(0)$ is the initial learning rate.

1.2 Momentum Learning Rule

To encourage the weight change to go in the consistent direction, a momentum term is introduced, and may help overcome shallow local minima and speed up convergence. The update now is defined as:

$$\Delta w_i(t) = -\eta D_i(t) + \alpha \Delta w_i(t-1) \tag{3}$$

where α is the momentum coefficient, which controls how much we trust our former step.

1.3 Adaptive Learning Rule

Adaptive learning rules are a family of algorithms which deal with learning rates on a per-parameter basis, and all the learning rates are not handled by pre-determined schedules, but changed adaptively.

Two kinds of adaptive learning rules, AdaGrad and RMSProp are examined in this report.

AdaGrad is a method that the step size corresponding to each parameter, is normalised by sum squared gradient S, where S can be updated as:

$$S_i(0) = 0 \tag{4}$$

$$S_i(t) = S_i(t-1) + D_i(t)^2 \tag{5}$$

So the learning rate of each parameter will be:

$$\eta_i(t) = \frac{-\eta}{\sqrt{S_i(t)} + \epsilon} \tag{6}$$

And our update for each parameter will be:

$$\Delta w_i(t) = \frac{-\eta}{\sqrt{S_i(t)} + \epsilon} D_i(t) \tag{7}$$

However, the normalisers for AdaGrad are monotonically decreasing, and there is a possibility that our learning rates will flatten out to 0 before training is finished.

So another algorithm called RMSProp is proposed where we will only keep the recent gradients to calculate normalisers. Now, the $S_i(t)$ is updated as:

$$S_i(t) = \beta S_i(t-1) + (1-\beta) D_i(t)^2 \tag{8}$$

Using a moving average of squared gradients as normalisers, the RMSProp does not take the aggressive decreasing learning rate, the rest is just the same as AdaGrad.

2 Learning Architecture

2.1 Data Set

For our experiments, MNIST handwritten digits data set is used for training and evaluation. The data set contains labeled 18×18 images which can be classified as 10 digits. The whole data set is shuffled and divided into training set and validation set. We use 50000 images for training, and the remaining 10000 images for validation. During the training procedure, mini-batch stochastic gradient descent is used and the batch size is fixed to 50.

2.2 Multi-Layer Neural Network

The neural network used in the experiments has one input layer, two hidden layer and the final output layer. Each layer has a typical structure which consists of an affine transform and the non-linear sigmoid function. The output layer is a softmax regression, which is suitable for the one-hot encoding targets. Unsurprisingly, the input layer has 768 units corresponding to the image pixels, while the output layer has 10 units corresponding to 10 digits. Moreover, the hidden layers all have a size of 100 units. It is worthwhile to mention that there is no pre-training procedure to our network, and no pre-processing to the input data.

2.3 Initialisation and Training

To keep the scale of activations at different layers of the network the same at initialisation, a newly proposed initialisation scheme[3] is applied to weights at the beginning. However, biases are all initialised to constant 0.

All the training procedures in this report go through a total of 100 epoch, and each epoch traverses the whole training set using a batch size of 50. We use back-propagation to calculate our gradients efficiently.

2.4 Evaluation

Since we are dealing with one-hot encoding targets classification problem, we will use cross-entropy error function to measure the difference between outputs and targets. The error will be averaged on the whole data set so that we can easily compare performance between training set and validation set.

The error function measures how likely the data set is being observed given the model and parameters. It is a good measure includes our confidence of the model prediction. However, another measure called accuracy can be used to evaluate and compare model performance. It can be easily calculated as:

$$\frac{number\ of\ correct\ predictions}{evaluation\ set\ size} \tag{9}$$

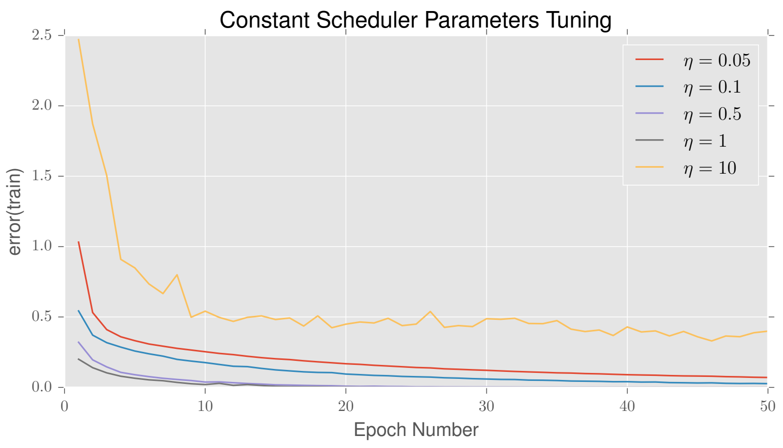

Figure 1: constant learning rates train error

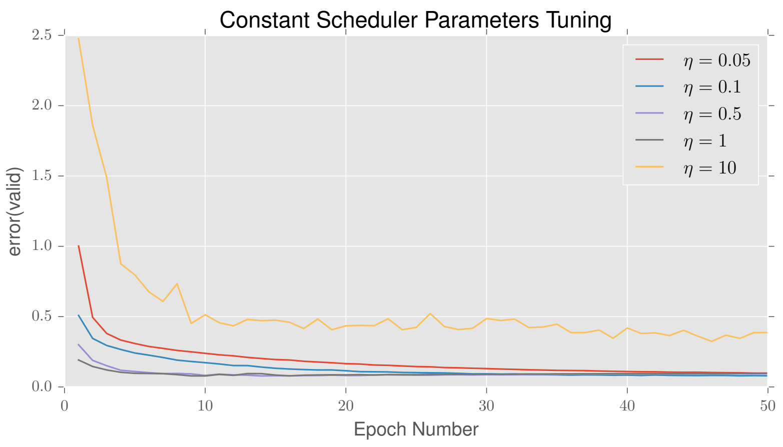

Figure 2: constant learning rates valid error

3 Experiments

3.1 Time-dependent Learning Rate Schedules

Learning rate affects the evolution and final results of our training in many ways. To investigate the potential effects under a simple model, let's use different learning rates in the constant scheduling model as a first step. From figure 1 and figure 2, models with larger learning rates tend to have a faster convergence at the beginning, but may potentially overfit the training data in a short number of epochs. However, that's not always the case. If the learning rate is too large, as η=10 in our experiment, the model will never converge at all. In terms of final performance, the model with extreme large learning rate ($\eta = 10$) behaves poorly, while the model with extreme small learning rate ($\eta = 0.05$) hasn't totally reach the optimal point due to its small step size. Regardless of convergence speed, models with learning rates in a reasonable range all get

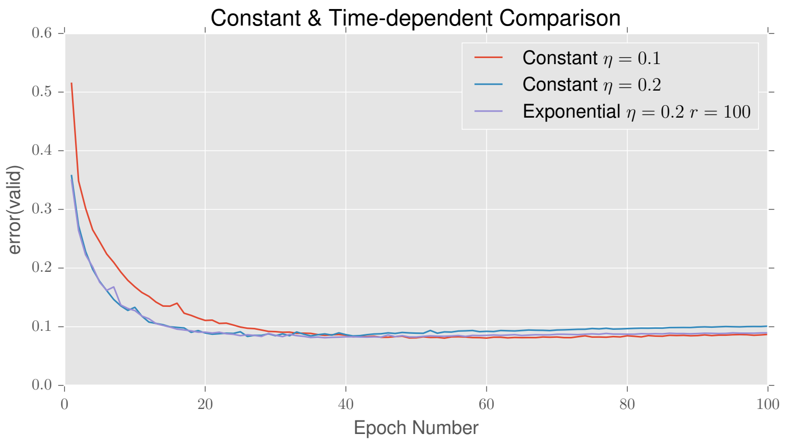

Figure 3: constant and time-dependent schedules

similar final validation errors, and the model with smaller learning rate ($\eta = 0.1$) performs slightly better on the validation set and slightly worse on the training set, showing up a better generalisation.

The problem of constant scheduling is that we are required to have a fixed rate throughout the whole procedure. But in fact, ideally, we want our training to take bolder step at the beginning and be more cautious near the convergence. Bigger learning rate at the beginning lead to faster convergence and broader exploration. At the same time, smaller learning rate in the end avoids vainly going back and forth. So that's the motivation for us to use a scheduling which decrease the learning rate as time pass by. There are many time-dependent scheduling achieving this goal, in this report, we well only examine exponential scheduling. To benefit from learning rate decay of the exponential schedule, we use $\eta = 0.2$ as the initial learning rate, comparing to constant scheduling with $\eta = 0.1$ and $\eta = 0.2$. As shown in figure 3, exponential scheduling converges much faster than constant scheduling with $\eta = 0.1$. But, why not use directly $\eta = 0.2$ if we want to converge faster? From the plot we can see, exponential scheduling first resembles $\eta = 0.2$, achieving faster convergence. But after 40 epochs, it starts to resemble constant scheduling with $\eta = 0.1$, which leads to a slightly better optimal in the end. That is to say, time-dependent scheduling benefits from large initial learning rate, significantly decreasing the error at the beginning, but also benefits from relatively small learning rate in the end, executing a finer tuning. This experiment should clearly show why we prefer a time-dependent scheduling sometimes.

To investigate the effects of free parameters η_0 and r in exponential scheduling, we perform experiments using different initial learning rate η_0, as shown in figure 4 and figure 5, and different decay rate r, which are presented in figure 6 and figure 7. We can see, bigger η_0 contributes to faster decrease of error function, but in the end, performs slightly worse than models with smaller η_0. From the evolution curve on the training set, models with large η_0 fully fit the training set so early, which bring about the not so good local minimum. On the other hand, r controls the decay speed of learning rate. If r is too small, then the learning rate almost flatten out to 0 half way down to the minimum, before

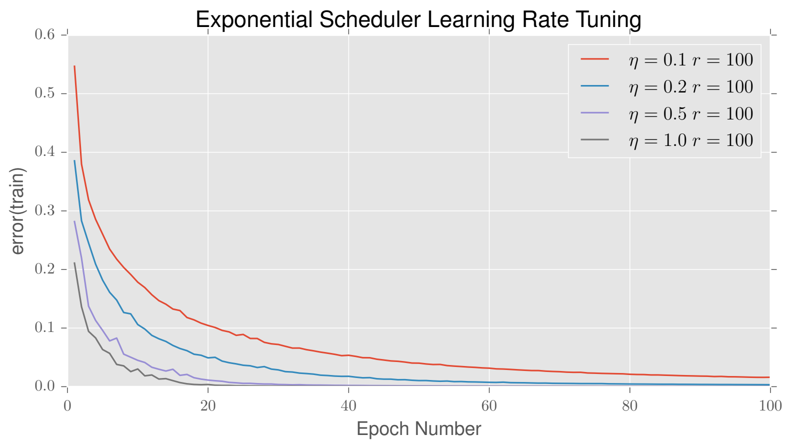

Figure 4: exponential scheduling learning rate tuning on training set

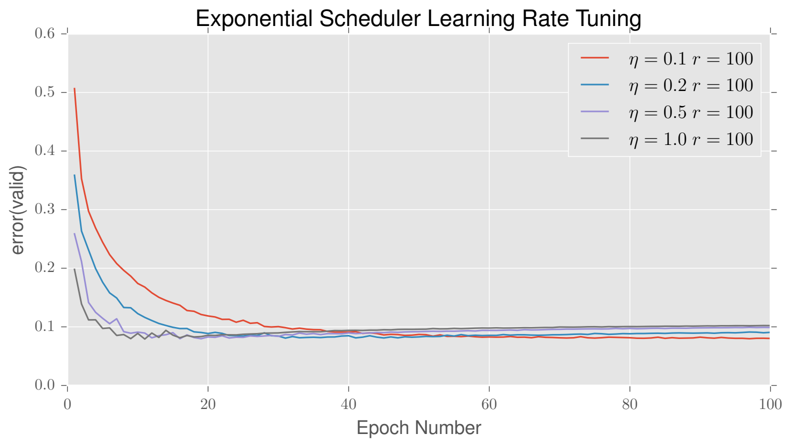

Figure 5: exponential scheduling learning rate tuning on validation set

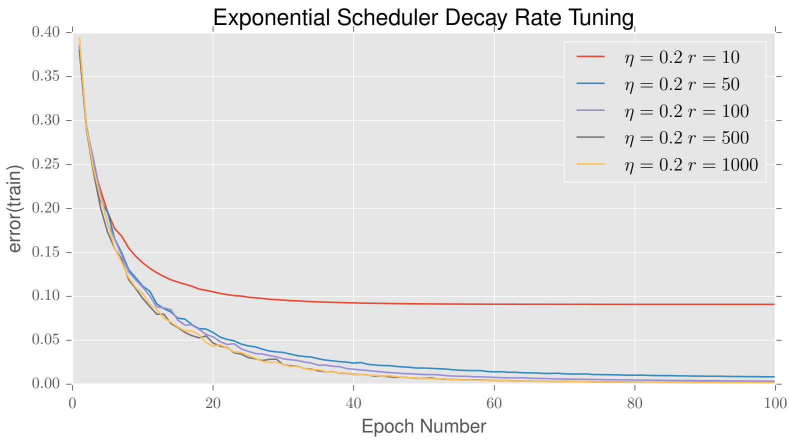

Figure 6: exponential scheduling decay rate tuning on training set

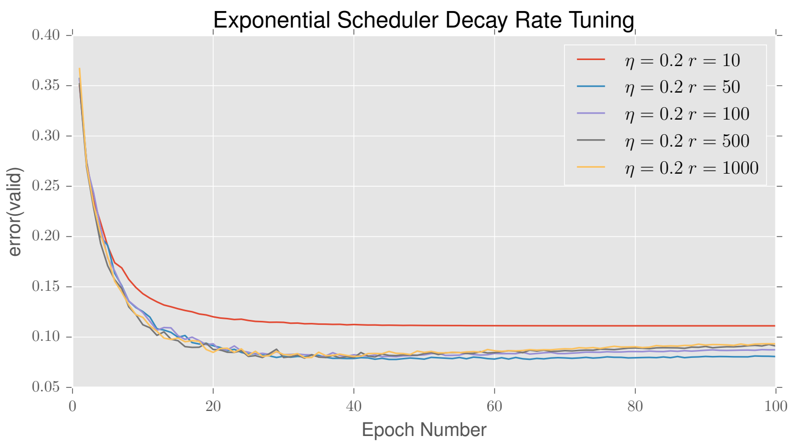

Figure 7: exponential scheduling decay rate tuning on validation set

actual finish of the training, which lead to a much higher final error. Also, the bigger the r is, the closer our model will be to the constant scheduling. This can be easily understood intuitively.

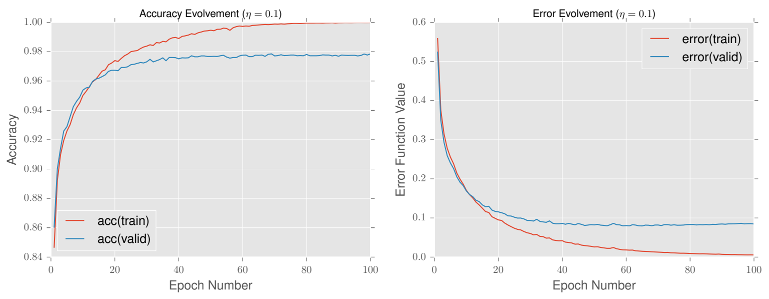

Figure 8: evolution of accuracy and error for constant scheduling

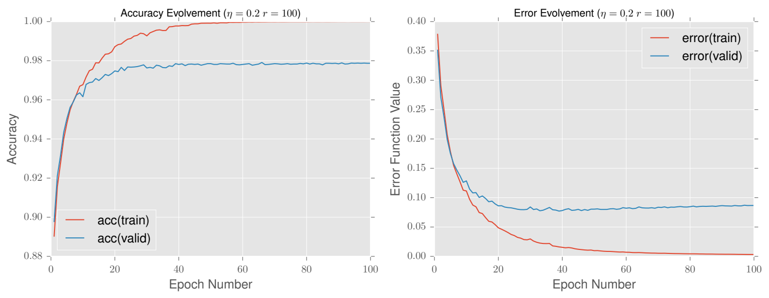

Figure 9: evolution of accuracy and error exponential scheduling

Finally, we apply the error and accuracy evaluation to our models. Both models achieve a final 100% accuracy on the training set. The constant scheduling ($\eta = 0.1$)achieves a final 97.8% accuracy on the validation set, while the exponential scheduling is quite close, 97.9%. In terms of error function, the average error of constant scheduling is 0.00548 on the training set, and 0.0846 on the validation set. In the case of exponential scheduling, the average error is 0.0032 on the training set, and 0.0867 on the validation set. The evolution of error and evaluation for both models are provided in figure 8 and figure 9.

3.2 Momentum Learning Rule

As a comparison to basic gradient learning rule, we conduct experiments using momentum learning rule, with different momentum coefficient α. From figure 10 to figure 13, all the models except the one with extreme small momentum fully fit the training set through different number of epochs. Models with big α converge faster than models with small α, but in terms of the optimal value of the error function, models with small α get a slightly better optimum on the validation set. However, when the momentum coefficient is too small ($\alpha = 0.5$), the optimisation can not be finished during the observed time window, which leads to a poor final performance. The accuracy evaluation seems to be contradictory with error function, because bigger momentum does not hurt the performance at all when getting faster convergence. Based on the intuition of our evaluation methods, we can say, even though models with lower α do not enjoy a higher accuracy, they do have a higher confidence of their prediction.

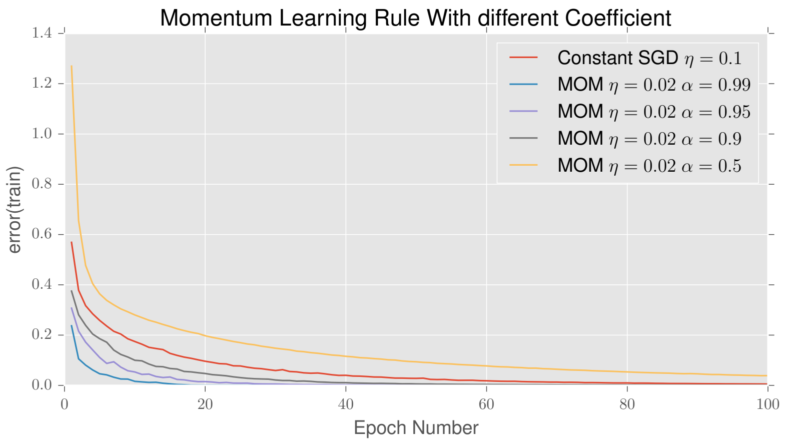

Figure 10: momentum learning rules(error on training set)

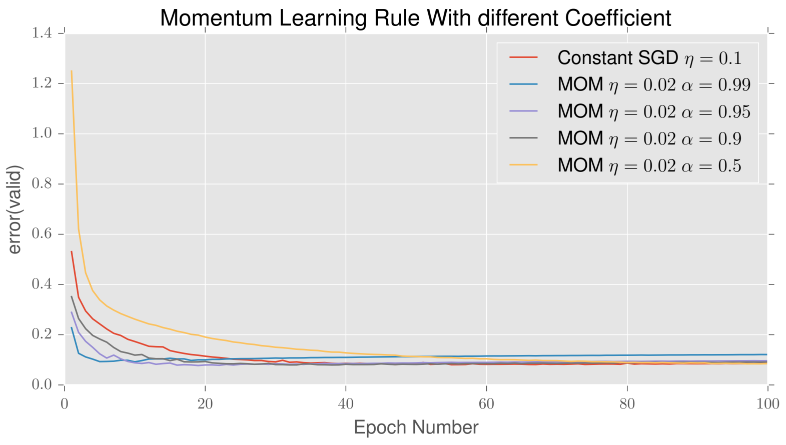

Figure 11: momentum learning rules(error on validation set)

Under the same circumstance, we should trust those models with lower error function values.

To compare with constant scheduling stochastic gradient descent, the final performance on the validation set for both the momentum learning rule and gradient descent baseline are almost the same. However, momentum learning rule does achieve a significantly faster convergence, given the fact that the initial learning rate for momentum learning rule is fixed to only 0.02. Going in the consistent direction builds up its confidence, and consequently increases the magnitude of its step size.

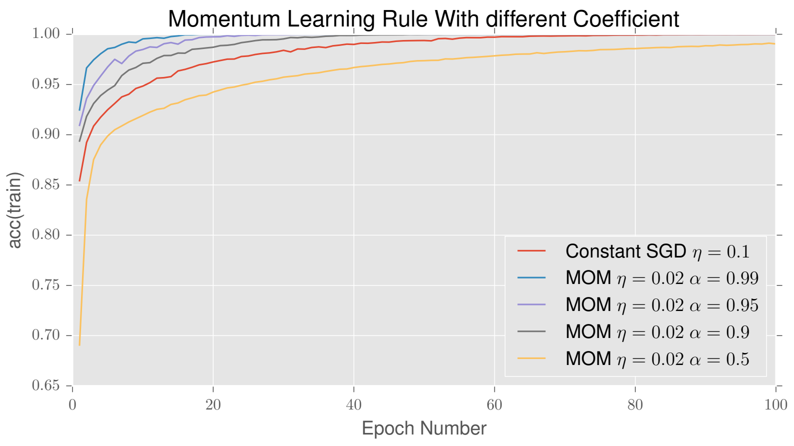

Figure 12: momentum learning rules(accuracy on training set)

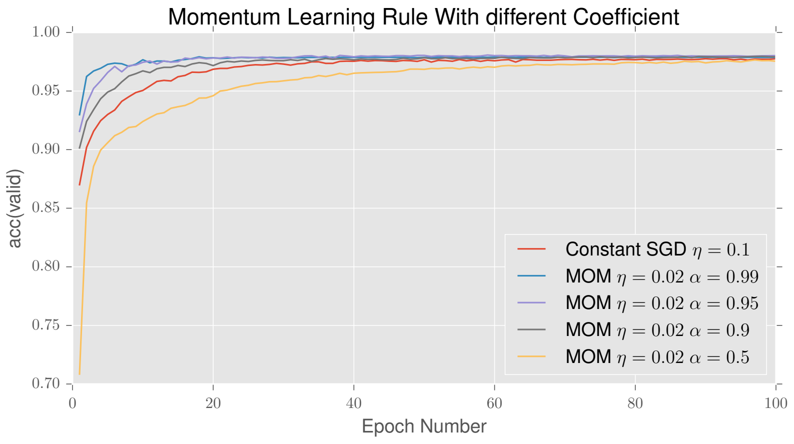

Figure 13: momentum learning rules(accuracy on validation set)

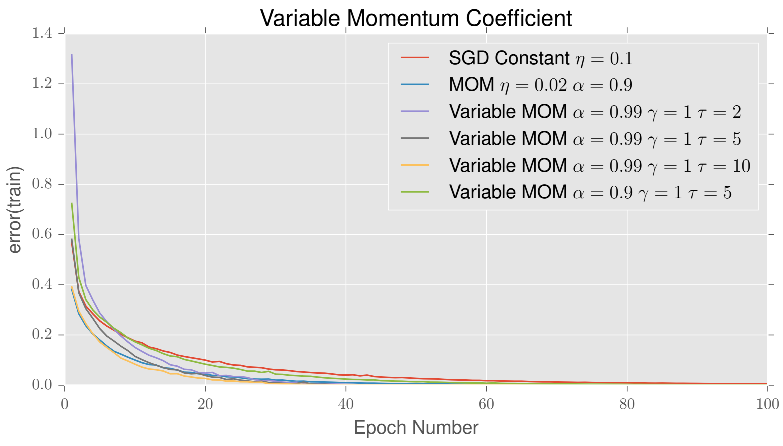

Figure 14: variable momentum learning rules(error on training set)

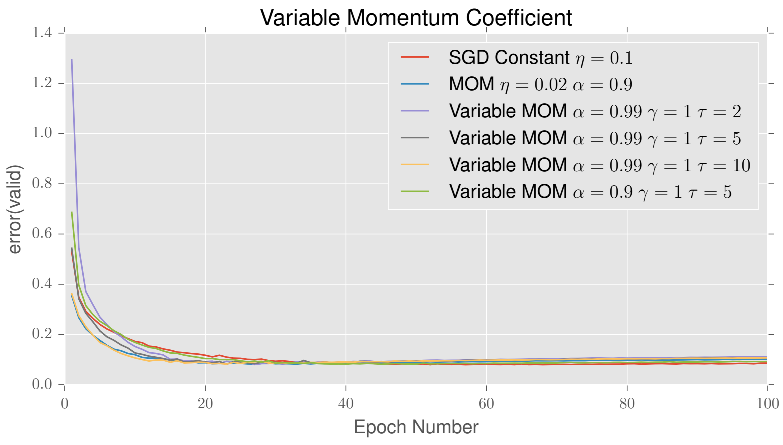

Figure 15: variable momentum learning rules(error on validation set)

Sometimes, we don't want to stick to a direction at the beginning, as we want our optimiser to do more exploration, avoiding shallow local minimum. The training starts at a random state, so the direction of the first few steps may not make sense at all. In figure 14 to figure 17, we compare constant scheduling, basic momentum learning rule with our variable momentum schemes. Several values for α^{∞}, λ and τ are given and no model with a variable momentum scheme seems to over-perform the constant momentum scheme, in terms of no matter final error or convergence speed. However, we can not draw the conclusion that variable momentum scheme does not help the neural network learning, since all the experiments are conducted under a relatively shallow neural network, and are trained using a relatively simple task. The exploration at the beginning may not help much in this specific task.

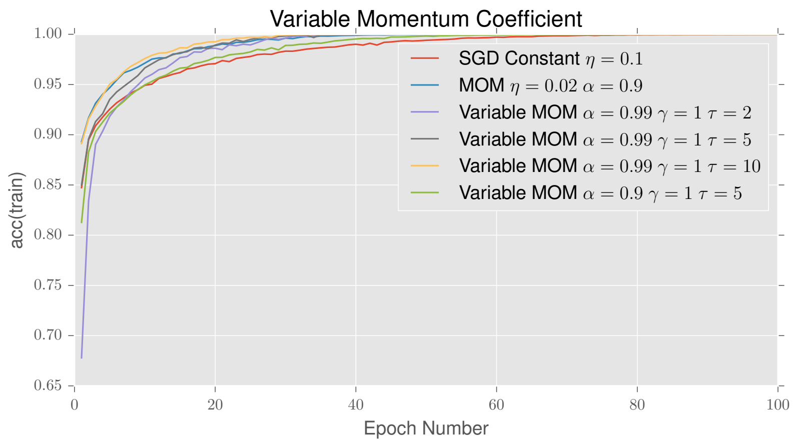

Figure 16: variable momentum learning rules(accuracy on training set)

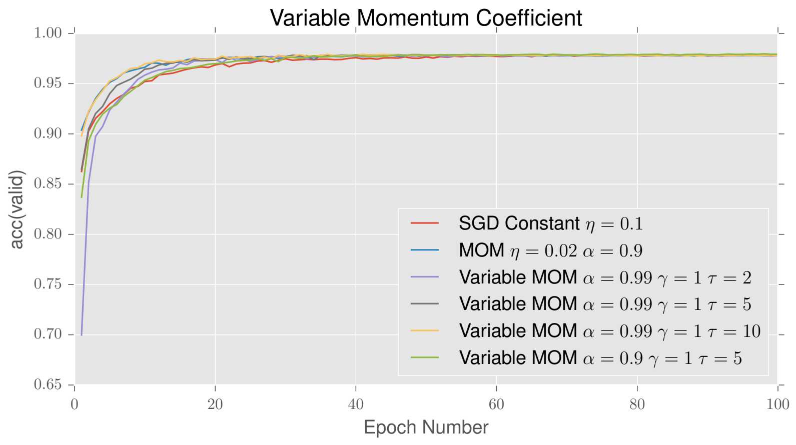

Figure 17: variable momentum learning rules(accuracy on validation set)

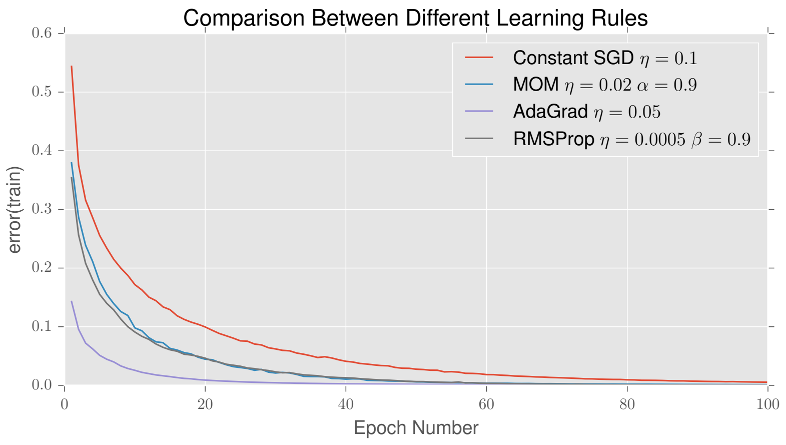

Figure 18: learning algorithm comparison(error on training set)

3.3 Adaptive Learning Rule

The experiments in this section are conducted using AdaGrad and RMSProp algorithm, and compared to previous basic gradient descent and momentum learning rule. All hyper-parameters are set to carefully chosen values for performance comparison. The final error and accuracy are reported in the table below. AdaGrad and Momentum have the best performance on the validation set, and RMSProp seems to overfit the training data heavily, resulting in a very large validation error in the end. However, the performance difference is not significant, and it is affected by specific settings. So we cannot draw further conclusions on this.

algorithm	error(train)	error(valid)	acc(train)	acc(valid)
SGD	5.40e-03	8.52e-02	100%	97.8%
Momentum	1.36e-03	9.84e-02	100%	97.9%
AdaGrad	6.49e-04	9.61e-02	100%	97.9%
RMSProp	6.45e-04	1.93e-01	100%	97.5%

Observing the evolution of accuracy and error during the training, all the adaptive algorithms achieve a faster convergence than basic gradient descent in terms of number of epochs. However, each epoch takes 2.08s in RMSProp, 1.92s in AdaGrad, 1.32s in momentum update rule and 1.15s in basic gradient descent. Adaptive learning rules apparently have greater computational cost. Taking these into consideration, the order of convergence speed is that AdaGrad $>$ Mom $>$ RMSProp $>$ Basic GD. Why AdaGrad is the fastest one? The normalisers in AdaGrad are monotonically decreasing, so AdaGrad will have the same effect as exponential scheduling. Given this fact, we will not be wrong about too large learning rates in the end, so we dare to try much larger initial learning rate at the beginning. RMSProp, however, do not have this property, so will need to give a very small initial learning rate.

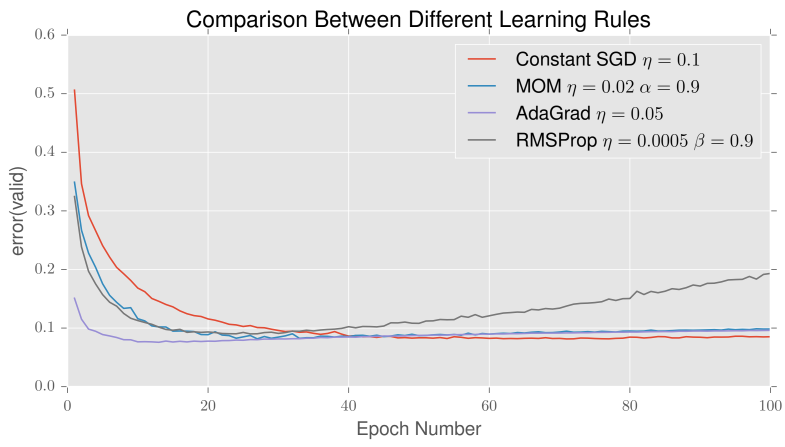

Figure 19: learning algorithm comparison(error on validation set)

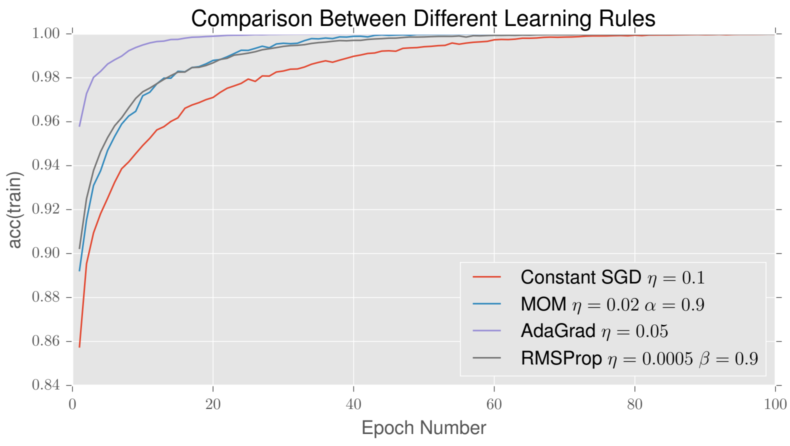

Figure 20: learning algorithm comparison(accuracy on training set)

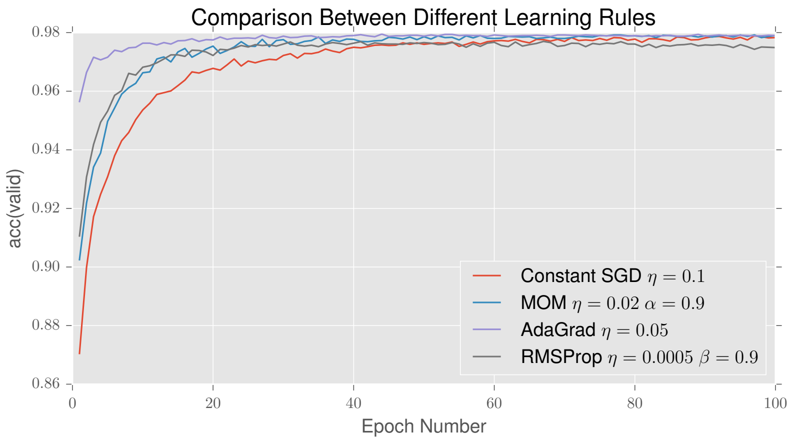

Figure 21: learning algorithm comparison(accuracy on validation set)

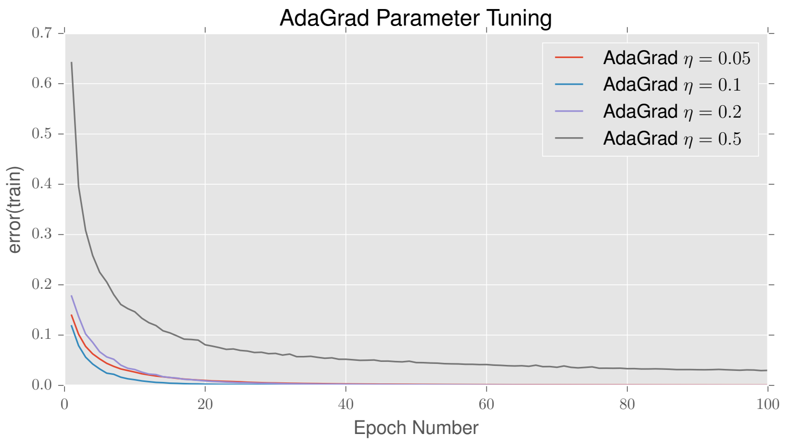

Figure 22: AdaGrad(error on training set)

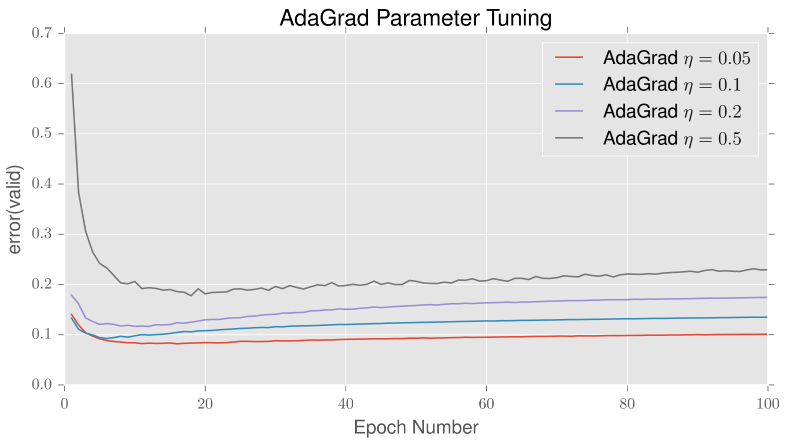

Figure 23: AdaGrad(error on validation set)

To investigate the effects of free parameters, we conduct experiments under different settings. As shown in figure 19 to figure 22, relatively high η in AdaGrad will end up in a local minimum, while lower η tends to converge at a faster speed. Surprisingly, this is totally the opposite in the situation of our previous model, where bigger learning rate tends to converge faster at the beginning, and models with too small value cannot converge during the time window. This phenomenon can be explained as the effect of normalisers. Since the normalisers are the square sum of gradients in each direction, so at first, this value should be value small if the learning rate is small. After normalisation, the actual learning rate turns out to be very big, which is consistent with our previous belief. However, except for some extreme value for η, the evolution seems to be quite close to each other. That is to say, the AdaGrad is not sensitive to η, given that η is in a reasonable range.

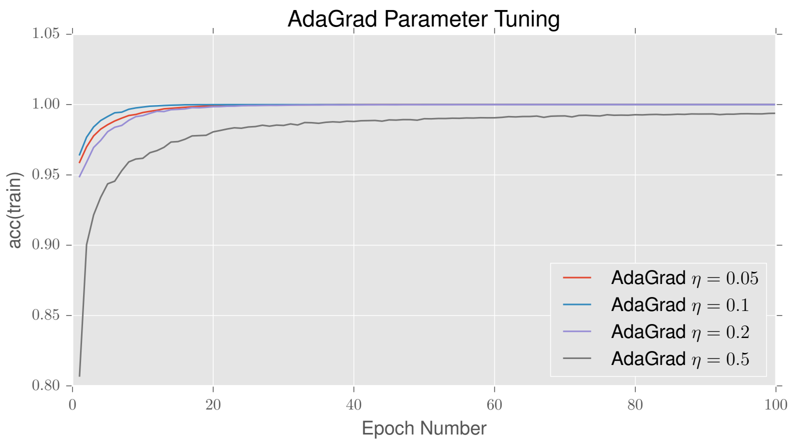

Figure 24: AdaGrad(accuracy on training set)

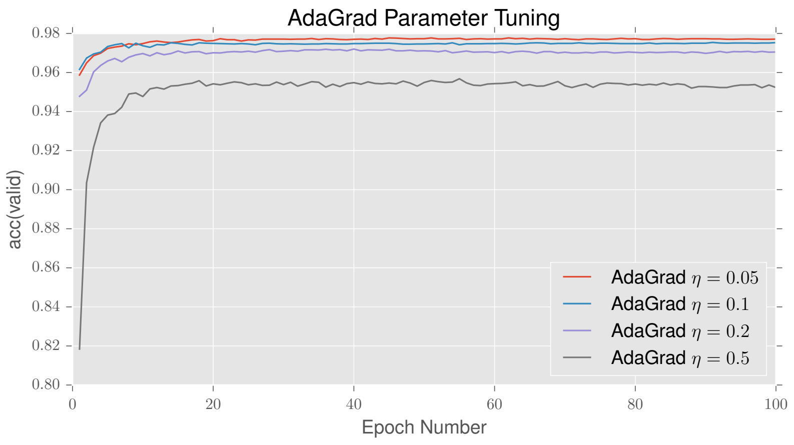

Figure 25: AdaGrad(accuracy on validation set)

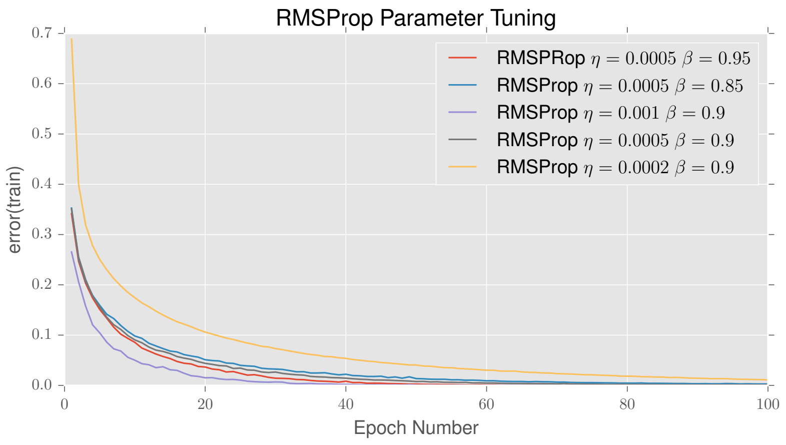

Figure 26: RMSProp(error on training set)

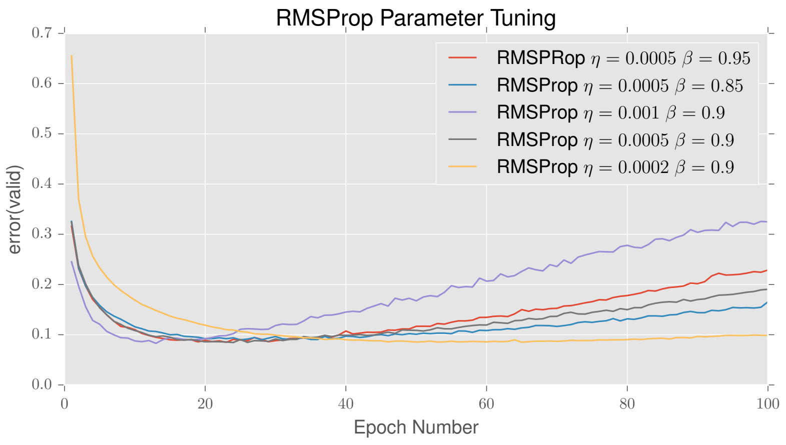

Figure 27: RMSProp(error on validation set)

RMSProp is in a similar case actually, the evolution curve is quite similar except for same extreme parameter settings. The β and η here seem to have very limited influence. In a reasonable range, lower β and η will lead to slightly slower convergence and better generalisation. (see figure 23 to figure 26)

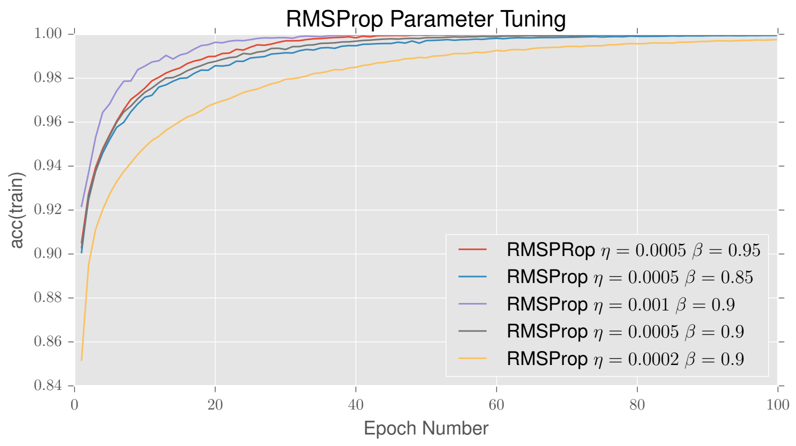

Figure 28: RMSProp(accuracy on training set)

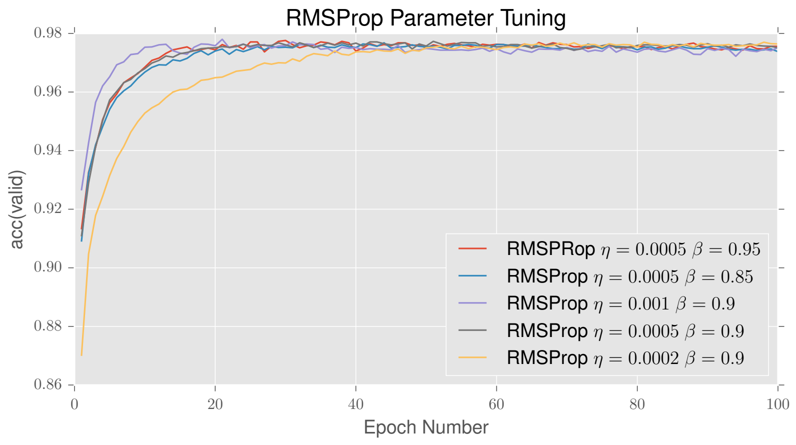

Figure 29: RMSProp(accuracy on validation set)

4 Conclusions

Several learning rate schedules and parameter updating rules are investigated and compared with each other in this report. As stated in the experiments and discussion, they have different performance and behaviours in terms of convergence speed, generalisation, and evolution of accuracy and error during training. However, we should not draw general conclusions about these algorithms, as all the results are acquired under a single learning architecture, and on a single data set. Anyway, these experiments do give us a clear picture of all these algorithms, in terms of both their properties and constraints.

References

[1] A.Senior, G.Heigold, M.Ranzato et al., An empirical study of learning rates in deep neural networks for speech recognition, Speech and Signal Processing, ICASSP 2013

[2] Andrej Karpathy, http://cs231n.github.io/neural-networks-3

[3] X.Glorot, Y.Bengio, 2010, Understanding the difficulty of training deep feed-forward neural networks, Proceedings of the 13th International Conference on Artificial Intelligence and Statistics