1 Part I : Learning Rate Schedules

Training a multi-layer neural network is a difficult optimization task. Generally, the algorithm to train a neural network used is Stochastic Gradient descent, but due to a constant learning rate, the training procedure doesnt stay stable after a certain number of epochs. It has been well established that an increased performance and accuracy can be achieved on implication of a dynamic learning rate that changes during training.

The method of applying learning rate that changes through the training procedure is sometimes called learning rate annealing or adaptive learning rate. Here we consider time dependent learning rate schedules, i.e. learning rates that vary with time. One of the simplest implementation for adaptive learning rates is reducing its value over time, so that the optimization algorithm makes large changes in the beginning and fine tune the weights later when the learning rate decreases.

We consider an exponential time-dependent learning rate schedule given by:

$$\eta(t) = \eta_0 \exp(-t/r) \tag{1}$$

Where, η_0 is the initial learning rate, t the epoch number, $\eta(t)$ the learning rate at epoch t and r a free parameter governing how quickly the learning rate decays.

1.1 Comparision with constant learning rate

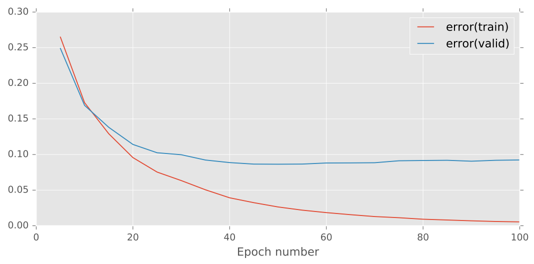
We compare the performance of adaptive learning rate schedule with constant learning rate schedule on training the standard model for MNIST digit classification task. Taking learning rate $\eta_0 = 0.1$ and $r = 50.0$, we compare the performance of algorithms with constant learning rate and adaptive learning rate as shown in the tables.

Epochs	Final Error (Train)	Final Error (Valid)
5	$2.58 * 10^{-1}$	$2.41 * 10^{-1}$
10	$1.69 * 10^{-1}$	$1.65 * 10^{-1}$
20	$9.28 * 10^{-2}$	$1.10 * 10^{-1}$
50	$2.82 * 10^{-2}$	$8.42 * 10^{-2}$
100	$5.59 * 10^{-3}$	$8.62 * 10^{-2}$

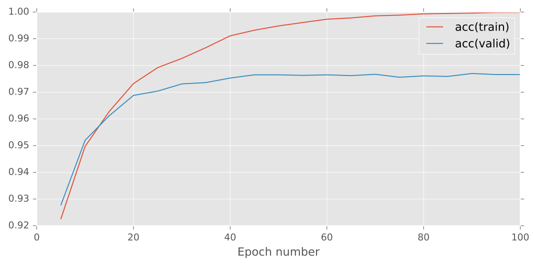
Table 1: Constant Learning Rate, $\eta_0 = 0.1$

Epochs	Final Error (Train)	Final Error (Valid)
5	$2.59 * 10^{-1}$	$2.44 * 10^{-1}$
10	$1.79 * 10^{-1}$	$1.75 * 10^{-1}$
20	$1.13 * 10^{-1}$	$1.27 * 10^{-1}$
50	$5.56 * 10^{-2}$	$9.02 * 10^{-2}$
100	$3.17 * 10^{-2}$	$8.25 * 10^{-2}$

Table 2: Adaptive Learning Rate, $r = 50.0$, $\eta_0 = 0.1$

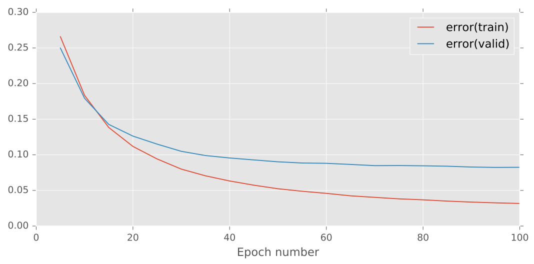


(a) Error Function

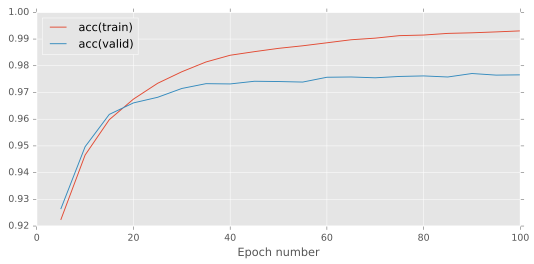


(b) Accuracy Function

Figure 1: Error and Accuracy plots for constant learning rate $\eta = 0.1$



(a) Error Function



(b) Accuracy Function

Figure 2: Error and Accuracy plots for Adaptive learning rate $\eta_0 = 0.1$ and $r = 50.0$

As can be seen from the given tables and graphs, Algorithms with Adaptive learning rates achieve a better value of Final Error and Accuracy on the given problem of MNIST Handwritten digit classification on the standard network architecture. If we change the initial learning rate to some higher value (As we shall see in the following sections), we see that the performance degrades dramatically for algorithm using constant learning rate as compared to that using an adaptive learning rate. This can be accounted to the face that decaying learning rate produces a smoother training error as the optimizer approaches the minimum value due to the decreasing step size with each epoch.

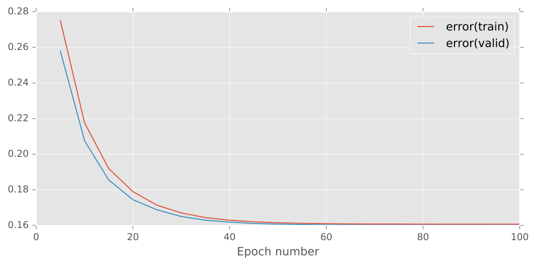
1.2 Effect of η_0 and r

We analyse the effect of changing the initial learning rate η_0 and the decay parameter r on the training procedure.

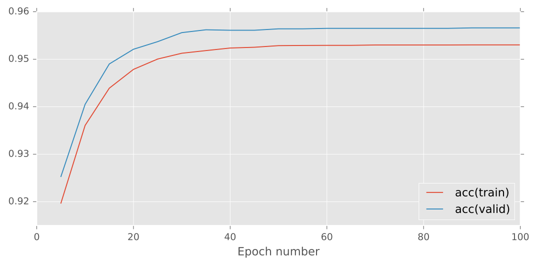
From the given tables and graphs showing relation between different values of r, the decay parameter and learning error achieved, it can be said as we increase r, we see the rate of change of learning rate decreases, i.e. a higher value of r signifies learning rate recays slowly and a lower value signifies quick decay in learning rate. From the table, for the given problem, we see that there is a sweet spot for r between $r = 50.0$ and $r = 75.0$ depicting an optimal value for this network configuration.

Epochs	Final Error (Train)	Final Error (Valid)
5	$2.76 * 10^{-1}$	$2.59 * 10^{-1}$
10	$2.20 * 10^{-1}$	$2.10 * 10^{-1}$
20	$1.81 * 10^{-1}$	$1.78 * 10^{-1}$
50	$1.64 * 10^{-1}$	$1.63 * 10^{-1}$
100	$1.63 * 10^{-1}$	$1.63 * 10^{-1}$

Table 3: Adaptive Learning Rate, $r = 10.0$, $\eta_0 = 0.1$

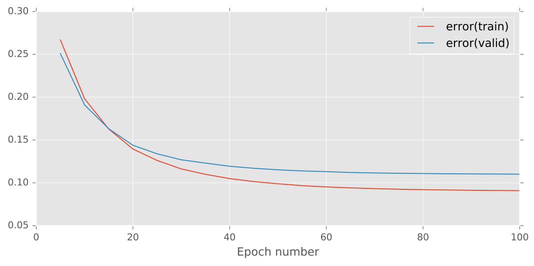


(a) Error Function

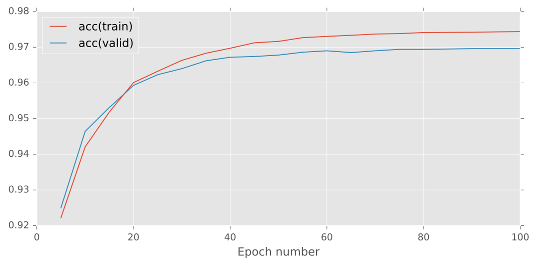


(b) Accuracy Function

Figure 3: Error and Accuracy plots for Adaptive learning rate $\eta_0 = 0.1$ and $r = 10.0$



(a) Error Function



(b) Accuracy Function

Figure 4: Error and Accuracy plots for Adaptive learning rate $\eta_0 = 0.1$ and $r = 20.0$

2 Momentum Learning Rule

Momentum learning rule extends the idea of basic gradient learning rule by introducing additional momentum state variables for the parameters. This technique helps the optimizer get out of local minima such that when the gradient keeps pointing in the same direction, it increases the step size towards the minimum, and when the gradients keep changing directions, momentum will smooth out the step size.

For the current example, the equation for parameter updates using momentum is given as:

$$M_{t+1} = \alpha M_t - \eta D_i(t) \tag{2}$$

Epochs	Final Error (Train)	Final Error (Valid)
5	$2.71 * 10^{-1}$	$2.56 * 10^{-1}$
10	$2.01 * 10^{-1}$	$1.95 * 10^{-1}$
20	$1.43 * 10^{-1}$	$1.47 * 10^{-1}$
50	$1.01 * 10^{-1}$	$1.17 * 10^{-1}$
100	$9.26 * 10^{-2}$	$1.12 * 10^{-1}$

Table 4: Adaptive Learning Rate, $r = 20.0$, $\eta_0 = 0.1$

Epochs	Final Error (Train)	Final Error (Valid)
5	$2.56 * 10^{-1}$	$2.43 * 10^{-1}$
10	$1.77 * 10^{-1}$	$1.76 * 10^{-1}$
20	$1.02 * 10^{-1}$	$1.17 * 10^{-1}$
50	$4.27 * 10^{-2}$	$8.48 * 10^{-2}$
100	$2.02 * 10^{-2}$	$7.96 * 10^{-2}$

Table 5: Adaptive Learning Rate, $r = 75.0$, $\eta_0 = 0.1$

$$\theta_{t+1} = \theta_t + M_{t+1} \tag{3}$$

Where, M_{t+1} is the updated value of momentum, α is the momentum coefficient, η is the learning rate, $D_i(t)$ is the gradient w.r.t. parameters at epoch t. We use these updates to optimize the parameters such that θ_{t+1} is the updated value of parameters, θ_t is the previous value of parameters.

2.1 Comparision with Gradient Descent

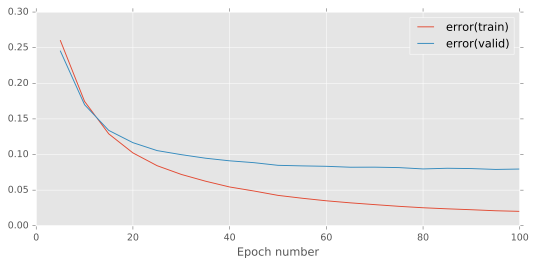
As can be seen from the table and graphs, the rate of change in accuracy as well as error for algorithms using the momentum rule us much higher than standard gradient descent. The learning rate $\eta = 0.002$ has been taken in the current example to show the effectiveness of the Momentum learning rule.

Epochs	Final Error (Train)	Final Error (Valid)
5	$5.34 * 10^{-1}$	$4.97 * 10^{-1}$
10	$3.63 * 10^{-1}$	$3.37 * 10^{-1}$
20	$2.76 * 10^{-1}$	$2.36 * 10^{-1}$
50	$1.67 * 10^{-1}$	$1.66 * 10^{-1}$
100	$8.97 * 10^{-2}$	$1.11 * 10^{-1}$

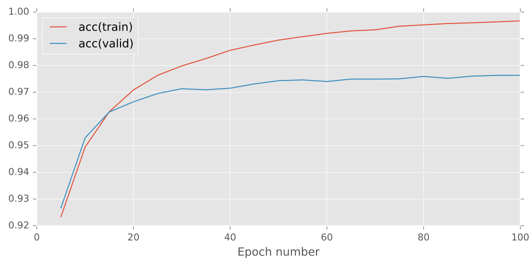
Table 6: Momentum learning rule, $\alpha = 0.9$, $\eta_0 = 0.002$

2.2 Role of α

In this section, we change the values of the momentum coefficient α to attain values 0.95, 0.90 and 0.80. As observed from the graphs and tables, keeping the value of α high ensures a better

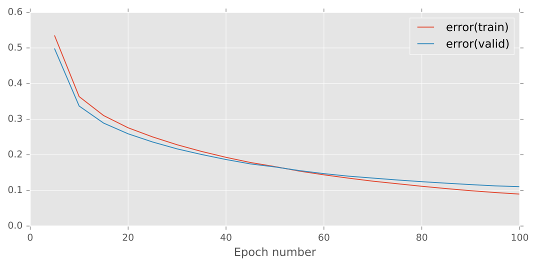


(a) Error Function

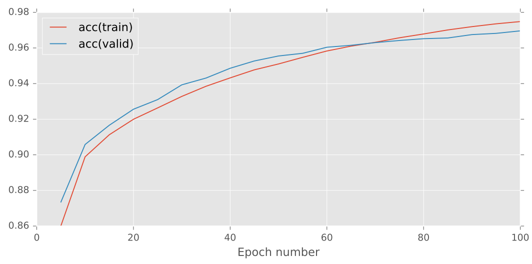


(b) Accuracy Function

Figure 5: Error and Accuracy plots for Adaptive learning rate $\eta_0 = 0.1$ and $r = 75.0$



(a) Error Function



(b) Accuracy Function

Figure 6: Error and Accuracy plots for Moment learning rule $\eta_0 = 0.002$ and $\alpha = 0.9$

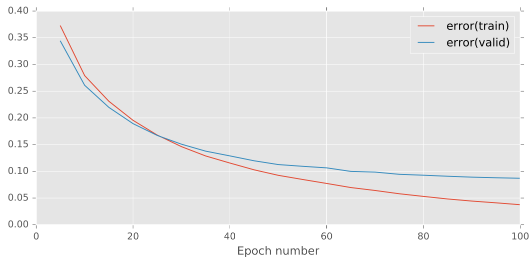
Epochs	Final Error (Train)	Final Error (Valid)
5	$3.72 * 10^{-1}$	$3.44 * 10^{-1}$
10	$2.79 * 10^{-1}$	$2.61 * 10^{-1}$
20	$1.95 * 10^{-1}$	$1.89 * 10^{-1}$
50	$9.25 * 10^{-2}$	$1.13 * 10^{-1}$
100	$3.75 * 10^{-2}$	$8.70 * 10^{-2}$

Table 7: Momentum learning rule, $\alpha = 0.95$, $\eta_0 = 0.002$

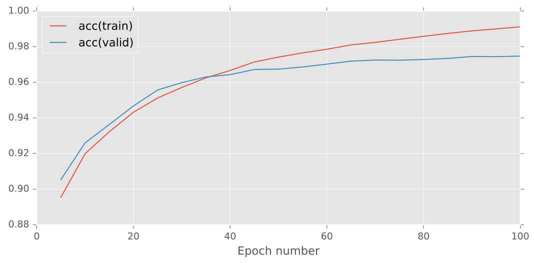
Epochs	Final Error (Train)	Final Error (Valid)
5	$9.94 * 10^{-1}$	$9.63 * 10^{-1}$
10	$5.43 * 10^{-1}$	$5.07 * 10^{-1}$
20	$3.62 * 10^{-1}$	$3.37 * 10^{-1}$
50	$2.53 * 10^{-1}$	$2.39 * 10^{-1}$
100	$1.70 * 10^{-1}$	$1.68 * 10^{-1}$

Table 8: Momentum learning rule, $\alpha = 0.80$, $\eta_0 = 0.002$

momentum and hence is able to produce better results in terms of error (loss) and accuracy of the model. keeping $\alpha = 0$ will transform the process to simple gradient descent.

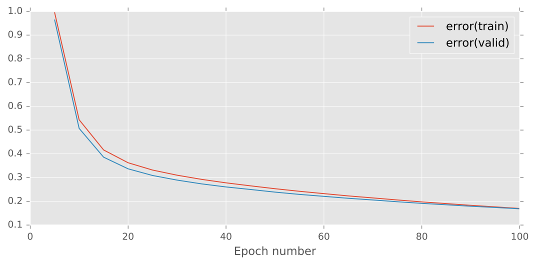


(a) Error Function

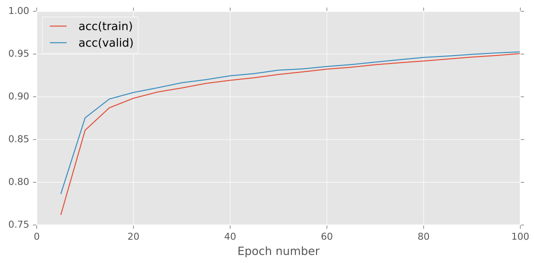


(b) Accuracy Function

Figure 7: Error and Accuracy plots for Moment learning rule $\eta_0 = 0.002$ and $\alpha = 0.95$



(a) Error Function



(b) Accuracy Function

Figure 8: Error and Accuracy plots for Moment learning rule $\eta_0 = 0.002$ and $\alpha = 0.80$

2.3 Variable Momentum Coefficient

Similar to scheduling of learning rate w.r.t. time, it is also possible to vary the momentum coefficient over a training run. In this particular example, we start with an initial value of α and keep increasing it to an asymptotic value of α_∞ as shown by the equation:

$$\alpha(t) = \alpha_\infty(1 - \frac{\gamma}{t + \tau}) \tag{4}$$

Where, $\alpha(t)$ is the time-dependent momentum coefficient, $\tau \geq 1$ and $0 \leq \gamma \leq \tau$ determine the initial momentum coefficient and how quickly the coefficient tends to α_∞.

3 Adaptive Learning Rules

Gradient Descent algorithm is one of the most popular algorithms used for optimization. But it has been shown that SGD (Stochastic Gradient Descent) has some trouble optimizing objective functions and may get stuck in local minima sometimes. To overcome this issue, several other variants of SGD can be used to improve performance and accuracy. Two such algorithms considered for this experiment are: RMSProp and Adam optimizers. Both of these algorithms use adaptive learning rates and have been proven to perform very well.

3.1 RMSProp

RMSProp is an adaptive learning rate variant of Stochastic Gradient Descent normalized by a moving average of the squared gradient as shown in the following equation:

$$S(t+1) = \beta S(t) + (1-\beta)D(t)^2 \tag{5}$$

$$\theta_{t+1} = \theta_t - \frac{\eta D(t)}{\sqrt{S(t+1)} + \epsilon} \tag{6}$$

Where, $S(t)$ is the previous value of the moving average momentum, $S(t+1)$ is it's next value, β is the decay rate of the moving average, $D(t)$ is the gradient w.r.t. parameters, ϵ is a free parameter (ex. $\epsilon = 1e-08$) to avoid division by zero error, θ_{t+1} and θ_t are the updated and current values of the model parameters.

3.2 Adam

Adaptive Moment Estimation (Adam) is a method similar to RMSProp with an additional momentum term that stores a moving average of past gradients along with the moving average of past squared gradients as in RMSProp. The Adam parametric update procedure follows the given equations:

$$M(t+1) = \alpha M_(t) + (1-\alpha)D(t) \tag{7}$$

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

$$\theta_{t+1} = \theta_t - \frac{\eta M(t+1)}{\sqrt{S(t+1)} + \epsilon} \tag{9}$$

Where, $M(t+1)$ and $M(t)$ are the updated and current values of the momentum of moving average of gradient, $S(t+1)$ and $S(t)$ are the updated and current values of the momentum of moving average of squared gradient, α and β are momentum coefficients, η is the initial learning rate, θ_{t+1} and θ_t are the updated and current model parameters, ϵ is a free parameter to avoid division by zero error while training.

3.3 Comparision with previous methods

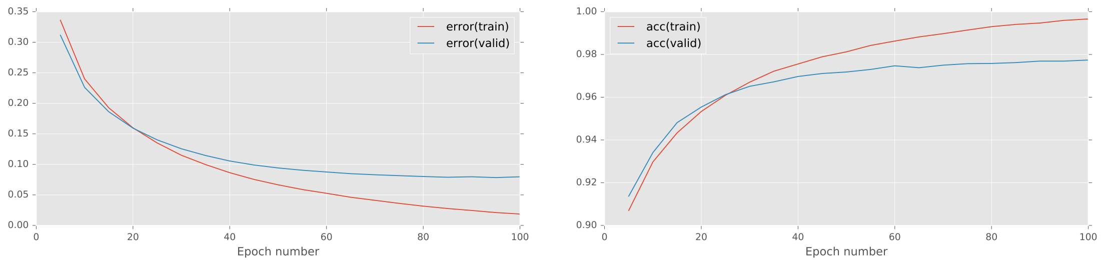
This section comprises of a brief comparision of RMSProp, Adam and the previous methods discussed, i.e. Gradient descent and Gradient descent with momentum learning rules.

Observing the plots for accuracy and error, Adam is accounted better than RMSProp due to a better rate of convergence and achieved minimum value. Adam can be seen as much better than rest of the methods considered in this work as it is attaining the lowest error score.

The free parameters α and β play a very important role in the performance of the considered algorithms. α is the momentum coefficient for moving average of gradient in Adam optimizer and β is the momentum coefficient of moving average of squared gradients in both Adam and RMSProp. Updates in the values of these parameters can affect the performance and achieved accuracy of the algorithms. It has been observed that values for β close to 0.90 perform optimally and values for α close to 1.0 have shown to perform very well. Decreasing the values of α slows down the moving average of gradient in Adam optimizer and hence degrades the performance.

Epochs	Final Error (Train)	Final Error (Valid)
5	$3.36 * 10^{-1}$	$3.12 * 10^{-1}$
10	$2.40 * 10^{-1}$	$2.26 * 10^{-1}$
20	$1.60 * 10^{-1}$	$1.59 * 10^{-1}$
50	$6.65 * 10^{-2}$	$9.42 * 10^{-2}$
100	$1.87 * 10^{-2}$	$7.84 * 10^{-2}$

Table 9: Adam learning rule, $\alpha = 0.9$, $\beta = 0.99$, $\eta = 0.001$, $\epsilon = 1e - 08$



(a) Error Function

(b) Accuracy Function

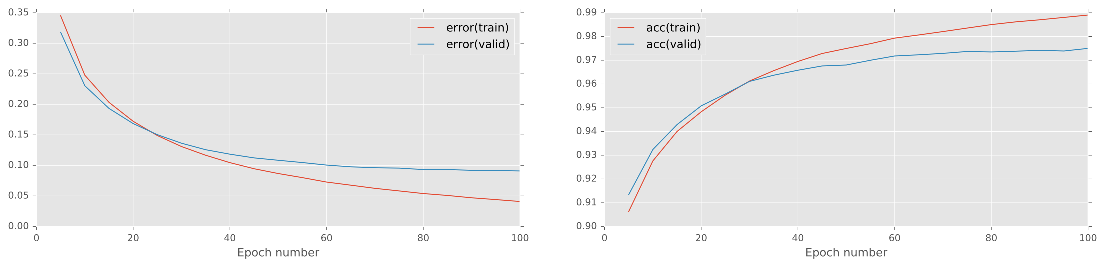
Figure 9: Error and Accuracy plots for Adam learning rule

Epochs	Final Error (Train)	Final Error (Valid)
5	$3.45 * 10^{-1}$	$3.18 * 10^{-1}$
10	$2.48 * 10^{-1}$	$2.30 * 10^{-1}$
20	$1.72 * 10^{-1}$	$1.68 * 10^{-1}$
50	$8.67 * 10^{-2}$	$1.08 * 10^{-1}$
100	$4.09 * 10^{-2}$	$9.09 * 10^{-2}$

Table 10: Adam learning rule, $\alpha = 0.9$, $\beta = 0.99$, $\eta = 0.001$, $\epsilon = 1e - 08$

4 Conclusions

From this work, focus was put on various methods for optimizing parametric models and how different learning methods affect the training procedure for various algorithms. Analysis of all the experiments shows Adam Optimizer to be one of the best performing optimizers for the given problem. We also saw that decaying/dynamic learning rates can produce impressive results as compared to using standard gradient descent, and introducing momentum in our training procedure seems to enhance the results even more. For a trade-off between computation time and state memory, RMSProp is slower than Adam optimizer but still performs very well as compared to Gradient Descent and Moment based learning rules.



(a) Error Function (b) Accuracy Function

Figure 10: Error and Accuracy plots for RMSProp learning rule