Machine Learning Practical - Assignment 1

s1247438

October 27, 2016

Introduction

This coursework is focused on training a multi-layered neural network for digit classification on the MINST dataset. The aim of this coursework is to investigate the effect of learning rates, learning rate schedules and adaptive learning rates on the model training and final performance.

Methodology

The neural network built for the purposes of this assignment is based on the Lab session code provided. After implementing schedules and learning rules the effect of their hyperparameters was investigated. This has been done by creating a parallelisable code allowing to run multiple neural network trainings simultaneously, making the most of computers with a large number of CPUs(50). By running a higher number of experiments the effect of hyperparameters could be observed and the best ones could be chosen for each schedule. The performance of the models trained with these optimal hyperparameters were then compared.

Task 1 - Learning rate schedules

The focus of the first section is to create and compare two different learning rate schedules : constant and exponential. The constant learning rate schedule uses a static learning rate that does not change during the training process and will then be taken as a baseline. The exponential learning rate is time-depended, the value changes as the time progresses.

$$n(t) = n_0 * e^{(-t/r)}$$

The equation contains two parameters n and r which dictate how quickly the learning rate decays. The aim being to start with higher learning rate and progressively decrease over time to make smaller adjustments. This section investigates different hyperparameter variations for both learning rate schedules.

Figure 2 displays effect of different learning rates on error and accuracy. Showing that learning rate of around 0.01 converges the quickest to an accuracy comparable with most other learning rates displayed. The model converges after 20 epochs and from that point it starts being over-fitted to the data.

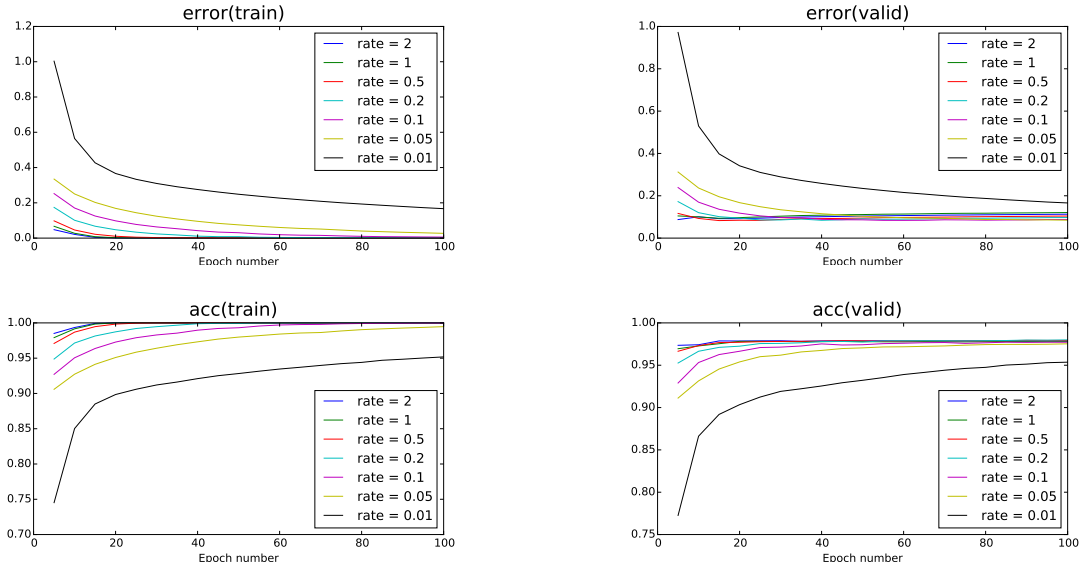

Figure 2: Constant learning rate hyperparameter results.

Figure 4 displays the effect of different hyperparameters on the exponential learning rate. Parameter r dictates how quickly the rate decays and to which final value. Two extremes can happen: one the model with stop learning after a small number of epochs (small r), or the rate does not really decay (large r). Parameter n scales these learning rate decays which r dictates.

Comparing the performance with the constant learning rate, the exponential rate converges as quickly as constant on the optimal parameters and in general faster. It also is a bit less over-fitted thanks to the decay.

Task 2 - Momentum learning rule

The second section introduces momentum or velocity to the gradient descent. This allows the model to overcome shallow local minima. The model will try to keep a more stable descent direction and will be less thrown off by possibly wrong direction introduced by the batches. The momentum learning rule relies on a coefficient alpha of how much memory it keeps of the past direction. This section investigates how different values of alpha influence the training and performance. The learning rate used is 0.2 as it performed well in the past tests.

Figure 7 displays effect of the alhpa hyperparameter on training and final performance. It shows that the model performs the best with alpha of 0.9 a generally accepted value. The value alhpa affects the speed of convergence. This model achieves similar performance to the baseline model for the final results, but the speed of convergence should be in general improved.

Next, this section investigates how changing the momentum over time changes the performance of our model. The idea being similar to exponential decay in the last section, only from the opposite side. The goal is to start with a small value of alhpa and end with a large

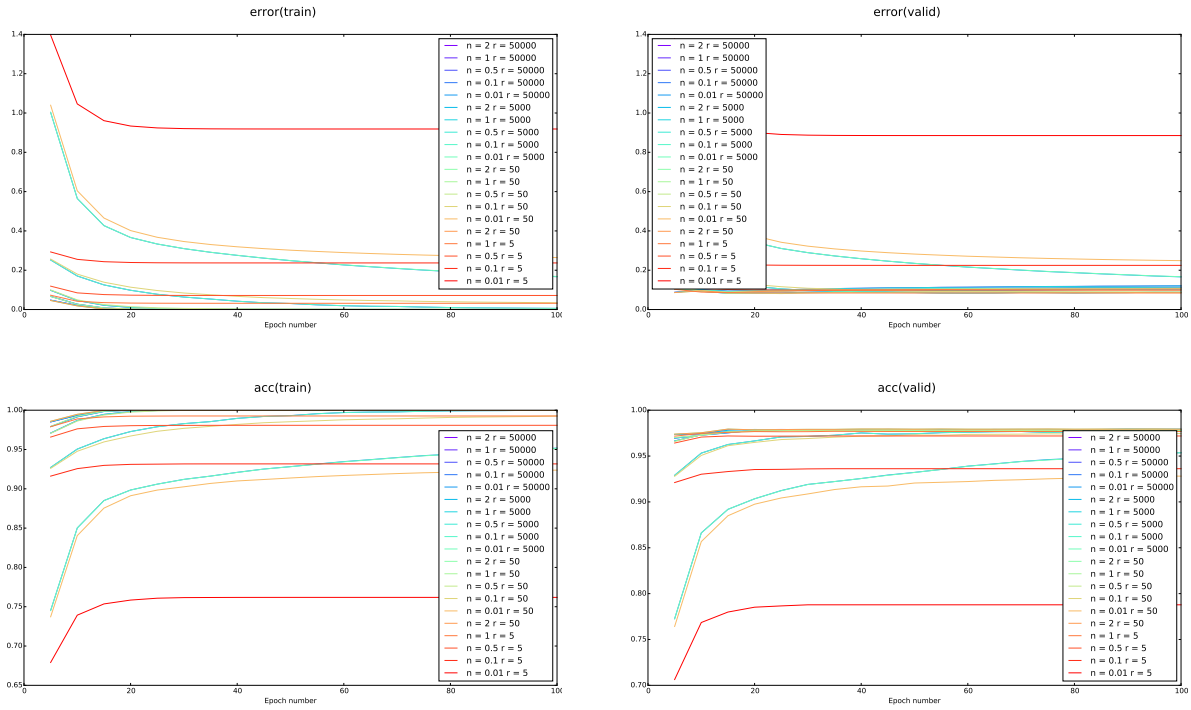

Figure 4: Exponential learning rate hyperparameter results.

one at the end of training, due to larger variances in gradient at the start compared with the end.

$$a(t) = a * (1 - y/(t - tau))$$

Figure 10 compares the time varying momentum to a constant learning schedule. It does not show a significant improvement on this dataset.

Task 3 - Adaptive learning rules

The last section explores adaptive learning rates, namely AdaGrad and RMSProp. Both of these rules apply updates to each weight individually. AdaGrad normalizes each weight parameter, meaning that weights with larger gradient magnitudes will have smaller weight updates and small gradients will result in larger updates.

$$S_i(t) = S_i(t-1) + D_i(t)^2$$

$$cw_i(t) = -n/(\sqrt{}(S_i(t) + e) * D_i(t)$$

In the equation above D is the gradient and e is a small constant (1e-8) to prevent division by 0.

AdaGrad's performance is displayed in Figure 12. The constant e should stay very small and the depended parameter is the learning rate showing similar results to constant learning rate. Both perform their best at learning rate 0.03. AdaGrad is more computationally

Learning rate	Accuracy(valid)
2	0.978
1	0.977
0.5	0.979
0.2	0.98
0.1	0.977
0.05	0.975
0.01	0.953

n	r	Accuracy(valid)
2	50000	0.979
1	50000	0.9779
0.5	50000	0.9793
0.1	50000	0.9772
0.01	50000	0.9536
2	5000	0.9785
1	5000	0.978
0.5	5000	0.9792
0.1	5000	0.9772
0.01	5000	0.9536
2	50	0.9793
1	50	0.9777
0.5	50	0.9785
0.1	50	0.9745
0.01	50	0.9281
2	50	0.9793
1	5	0.9768
0.5	5	0.9719
0.1	5	0.9362
0.01	5	0.7878

Figure 5: Comparison of Constant and Exponential learning rates accuracies on the validation set.

intensive and learns quicker in terms of convergence but does not display enough of a boost to argue it's case over for example the exponential learning rate decay with gradient descent.

RMSProp is a stochastic gradient descent of RProp. It is also very similar to AdaGrad where it introduces decay rate using the hyperparameter Beta. The effects of Beta are explored in Figure 14, showing the best performance with Beta equal to 0.9 which is the commonly accepted value. RMS Prop shows similar performance to AdaGrad and introduces a lot of computational overhead as well for just a small performance boost.

$$S_i(t) = B * S_i(t-1) + (1-B) * D_i(t)^2$$

$$cw_i(t) = -n/(\sqrt{(S_i(t) + e)} * D_i(t)$$

Discussion

Various learning rates, schedules and adaptive learning rates were investigated. Both AdaGrad and RMSProp are the most sophisticated models but they do not provide enough of a value for the computational overhead with the given dataset. Experimenting with different hyperparameters showed that all models can perform very similarly if the right hyperparameters are chosen. This resulted in all models not varying too much from the Baseline as it

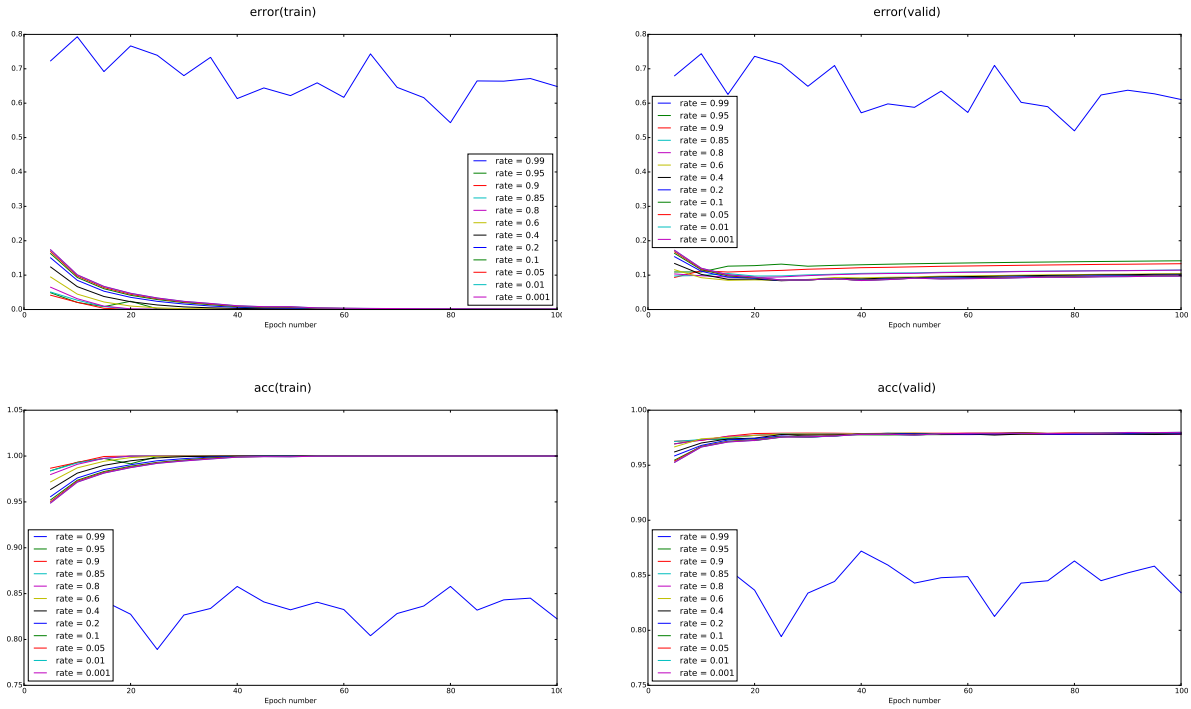

Figure 7: Momentum learning rule hyperparameter results.

was performing quite optimally already. The classification misses occurred often at really unclear data points which humans would not really know how to classify themselves. Of all the models explored I would suggest the exponential learning rate decay as it exhibited a convergence speed-up with very little overhead.

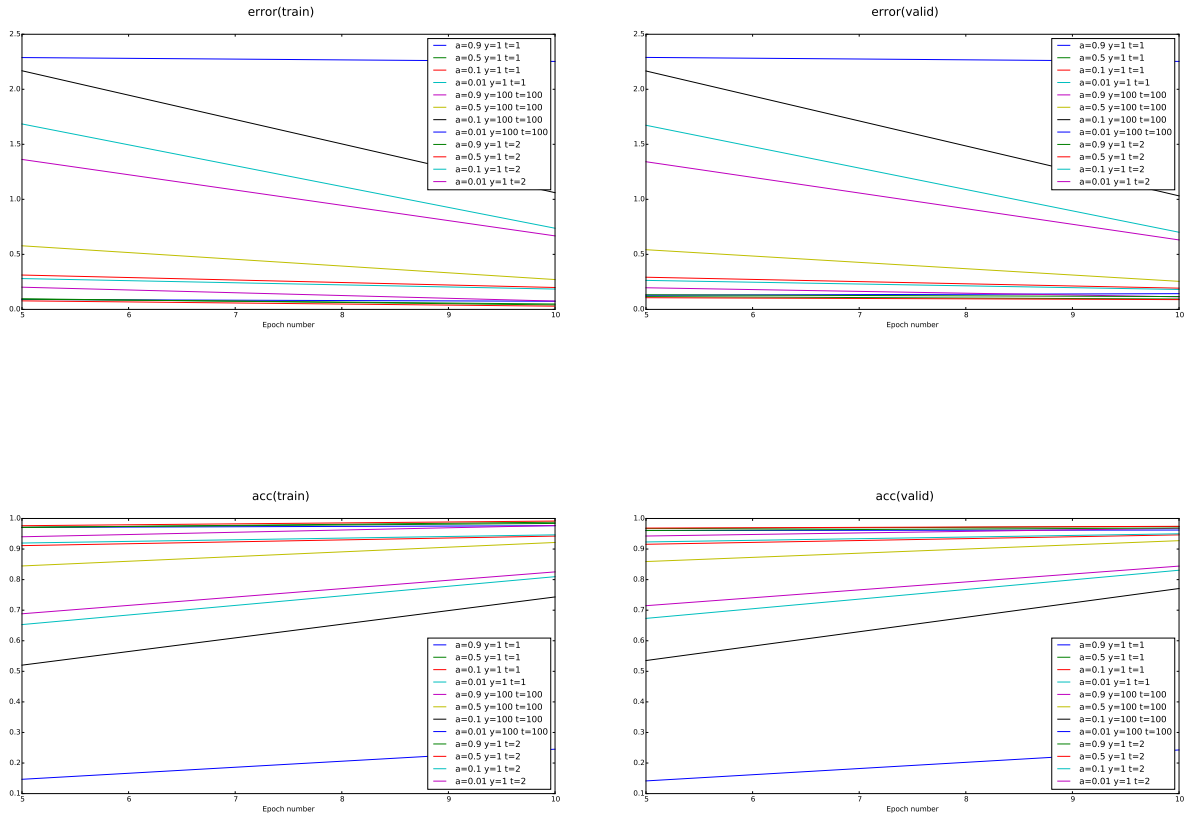

Figure 9: Decaying momentum learning rule hyperparameter results.

Alhpa	Accuracy(valid)
0.99	0.8342
0.95	0.9781
0.9	0.9788
0.85	0.9784
0.8	0.9779
0.6	0.9788
0.4	0.9784
0.2	0.9791
0.1	0.9797
0.05	0.9797
0.01	0.9799
0.001	0.9799

Alhpa	Gamma	Tau	Accuracy(valid)
0.9	1	1	0.9622
0.5	1	1	0.9739
0.1	1	1	0.9464
0.01	1	1	0.8307
0.9	100	100	0.9679
0.5	100	100	0.9274
0.1	100	100	0.7709
0.01	100	100	0.2428
0.9	1	2	0.9688
0.5	1	2	0.9743
0.1	1	2	0.9499
0.01	1	2	0.8442

Figure 10: Comparison of momentum hyperparameters on accuracy/

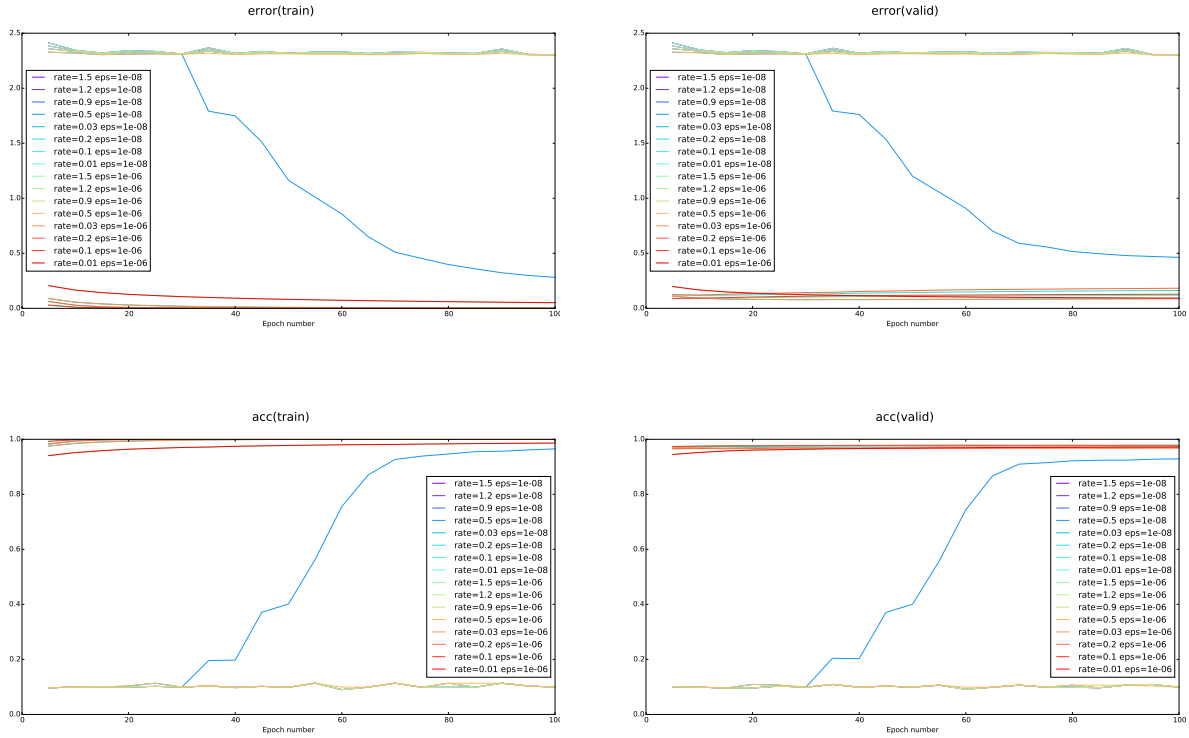

Figure 12: AdaGrad learning rule hyperparameter results.

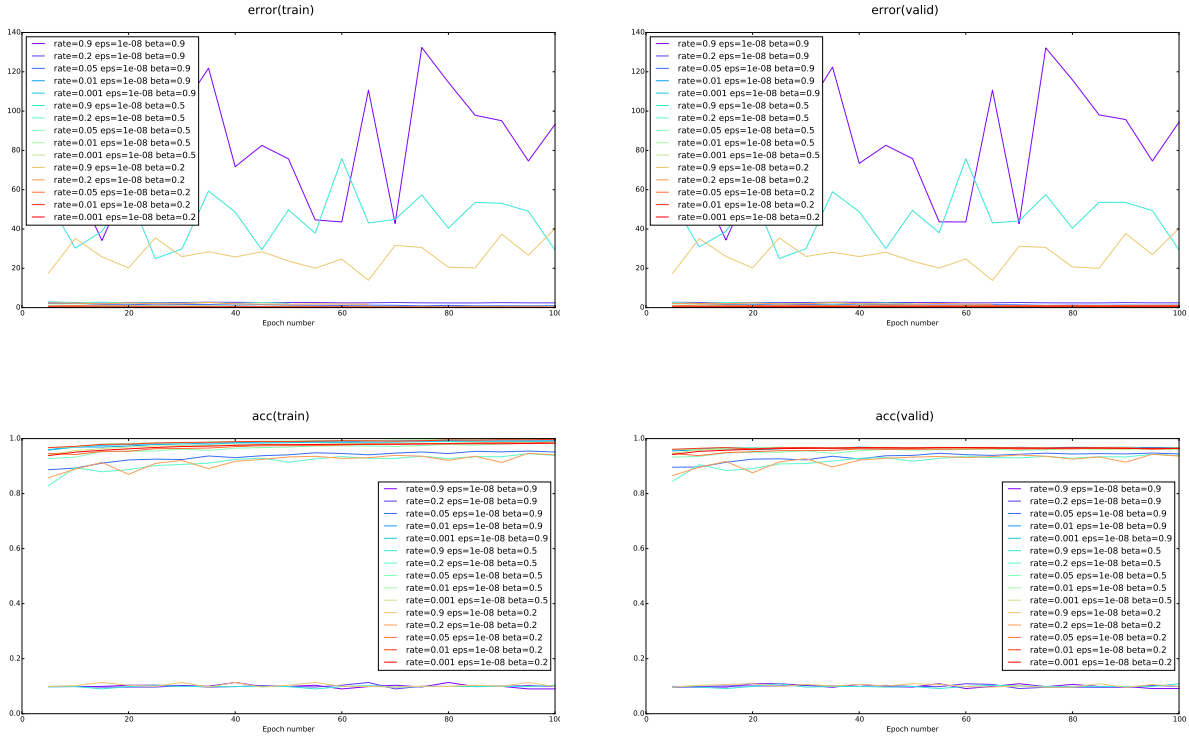

Figure 14: RmsProp learning rule hyperparameter results.

Learning rate	Eps	Accuracy(valid)
1.5	1e-08	0.099
1.2	1e-08	0.099
0.9	1e-08	0.099
0.5	1e-08	0.929
0.03	1e-08	0.9785
0.2	1e-08	0.9708
0.1	1e-08	0.978
0.01	1e-08	0.9718
1.5	1e-06	0.099
1.2	1e-06	0.099
0.9	1e-06	0.099
0.5	1e-06	0.099
0.03	1e-06	0.9783
0.2	1e-06	0.968
0.1	1e-06	0.9759
0.01	1e-06	0.9719

Learning rate	Eps	Beta	Accuracy(valid)
0.9	1e-08	0.9	0.0915
0.2	1e-08	0.9	0.099
0.05	1e-08	0.9	0.9443
0.01	1e-08	0.9	0.9676
0.001	1e-08	0.9	0.9618
0.9	1e-08	0.5	0.109
0.2	1e-08	0.5	0.9376
0.05	1e-08	0.5	0.9625
0.01	1e-08	0.5	0.9668
0.001	1e-08	0.5	0.9605
0.9	1e-08	0.2	0.0991
0.2	1e-08	0.2	0.9364
0.05	1e-08	0.2	0.9645
0.01	1e-08	0.2	0.9645
0.001	1e-08	0.2	0.9645

Figure 15: Comparison of AdaGrad and RMSGrad hyperparameters on accuracy.