Machine Learning Practical - Coursework 1

Luca Scimeca

November 22, 2016

Contents

1 Introduction

This report describes the experiments run for *Coursework 1* of the course Machine Learning Practical at the University of Edinburgh. The coursework involves the training of a multi-layer neural network (NN) to perform digit recognition. The experiments focus on the implementation and investigation of various techniques to modify the learning rate parameter η as the network is learning to classify the data, and their subsequent effects on the network and classification task. The techniques reported are generally divided in three categories: Time-dependent learning rates, Momentum learning rules and Adaptive Learning rules.

2 Methods

The experiments were run on MNIST data set containing 28x28 normalized pictures of hand-written digits. At training time, all models were fit with mini-batch stochastic gradient descent, with a batch size of 50 and were run for 100 epochs over the training data. Moreover, all neural networks implemented feature an input of 784 features (pixels in MNIST images), one hidden layer of 100 units and an output of dimension 10, each layer featuring a *sigmoid* transformation, as the NN ultimately classifies each digit in the dataset as one of $\{0..9\}$.
The error function used for gradient descent is:

$$\bar{E} = \frac{1}{2N} \sum_{n=1}^{N} \sum_{k=1}^{10} (y_k^{(n)} - t_k^{(n)})^2 \tag{1}$$

where N is the number of training examples and $y^{(n)}$ and $t^{(n)}$ are the nth model classifier output and target respectively.
As the different learning rules are meant to improve on an initial baseline, the first experiment runs the constant learning rate gradient descent algorithm over the η space, finding a best performance for $\eta = 0.07$ (see Figure 1).

2.1 Time-dependent learning rate

As the name may suggest, time-dependent schedulers focus on changing the η learning rate parameter in the weight update formula proportionally to the time step (epoch number) the algorithm is in. The method is meant to capture the initial need, during training, of *larger* steps to reach the local minimum of the error in the weight space, and the subsequent need of smaller steps as the we get closer to said minimum.
An exponential rate schedule was implemented, and the learning rates at each time step computed as:

$$\eta_(t) = \eta_0 \exp(-\frac{t}{r}) \tag{2}$$

where η_0 is the initial learning rate and k is an hyper-parameter controlling the speed with which $\eta(t)$ falls as t increases. The hyper-parameters were chosen by searching over the each parameter space in turn, and setting the parameter to the value performing best on a validation set.
To set the k hyper-parameter the learning rate η_0 was fixed to 0.07 (previously found to work well for a constant learning rule) and a search over the k space was performed (see Figure 2a), finding the best fit at $k = 100$. A similar procedure was used to search the best validation

fit over the η_0 hyper-parameter (see Figure 2b), now fixing k to 100 and finding the minimum validation error to be for $\eta_0 = 0.12$.

2.2 Momentum learning rule

The second set of experiments explore the insertion of *momentum* or better *velocity* in the learning rule, thus:

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

where the moment coefficient α is here another hyper-parameter which needs setting. A good starting point for the α value is 0.9 (see MLP Lecture 5). To explore how the α parameter effects learning, the momentum learning rule was used to train 4 models, with varying α values. The validation error and accuracy for each model were plotted over 100 epochs, and compared to the training and validation error of a baseline constant Gradient Descent model (see Figure 4). For both models η was fixed at 0.001. As clear from the picture increasing values of α improved performance on the validation set, however, values too close to one were avoided as momentum would drive the *descent* on the gradient dominantly, possibly preventing the gradient step to steer the descent towards the correct direction.
As previously, it is also possible to schedule learning rates; more precisely, it can be useful to get larger momentums when closer to the local minimum and less so when further away, where it is more likely for the direction of the gradient to be less precise in indicating where the local minimum is in the weight space. The alpha scheduling here is:

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

where α, γ and α_∞ are now hyper-parameters which need setting.
The α_∞ parameter sets an asymptotic value for the momentum rate, which intuitively can be a value around 0.9, while the γ and τ parameters control how quickly the rate changes at each epoch. Given the expense of grid searching over the possible parameter values, the search was done over each parameter in turn. The τ parameter was found by searching over a 1 to 100 range, fixing α_∞ to the 0.9 (asymptotically reaching a known good value for α) and $\gamma = \tau$ (see Figure 5a). The γ parameter was thereafter found by searching over a 0 to τ space (see Figure 5b), and finally the α_∞ was searched around 0.9 to find the best parameter settings (see Figure 5c).

2.3 Adaptive learning rule

The last set of experiments explore two adaptive learning rate rules, *RMSProp* and *Adam*.
In RMSProp the weight gradient is computed as:

$$S_i(t) = \beta S_i(t-1) + (1-\beta) \frac{\partial E}{\partial w_i(t)}^2 \tag{5}$$

$$\Delta w_i(t) = \frac{-\eta}{\sqrt{S_i(t)} + \epsilon} \frac{\partial E}{\partial w_i(t)} \tag{6}$$

where $D_i(t)$ is the gradient of the error function with respect to w_i at time t, $S_i(t)$ a weighted average of the square sum of the gradients up to t, and ϵ a value to prevent division by 0, set to

$1e-7$ throughout the rest of the experiments.
The best hyper-parameter settings in RMSProp were found similarly to other experiments. Initially β was set to 0.9, a good initial value for the hyper-parameter (see MLP Lecture 4, Slide 9), and a search on the η space was performed, yielding the best performance for $\eta = 0.0001$ (see Figure 7a). Afterwards, η was fixed at 0.001 and the β hyper-parameter searched around 0.9 to find the setting which would minimize validation error (see Figure 7b), i.e. $\beta = 0.9999$. The last learning rate rule, Adam, can be considered an extension of RMSProp, where a *momentum* biased parameter is substituted to the previous derivative of the error with respect to the weight, thus:

$$M_i(t) = \alpha M_i(t-1) + (1-\alpha)\frac{\partial E}{\partial w_i(t)} \tag{7}$$

$$\Delta w_i(t) = \frac{-\eta}{\sqrt{S_i(t)}+\epsilon} M_i(t) \tag{8}$$

The hyper-parameters are now α, β and η, which needs setting. η is again chosen to be 0.001, so to be able to compare the other hyper-parameter influences across experiments, while α and β are searched around the values 0.9 and 0.999 respectively (see MLP Lecture 4, slide 10).
Figure 8 shows the change in minimum validation error with different values of *alpha* (see Figure 8a) and *beta* (see Figure 8b)

3 Results

3.1 Time-dependent learning rate

The time-dependent learning schedule was compared to a constant learning rate baseline. To be able to make comparisons based only to the the influence of the exponential learning rate scheduling to the constant model, both models were given the same randomly initialized weight matrix. Both NN were trained on the model 5 times with a different random weight initializations each time, their performance was then averaged and compared.
In all trials, the performance on the validation set is consistently better for the constant learning rule in the first ≈ 40 epochs, however, the constant learning rate eventually starts worsening as the model starts over-fitting on the training examples. The exponentially scheduled learning rate, start decaying the η parameter with the number of epochs, effectively taking smaller steps towards the error minimum in weight space, and therefor asymptotically reaching a better performance, eventually, to that of a constant learning rate scheduler. Figure 3 compares the two models in one such trial.

3.2 Momentum learning rule

The momentum learning rule was compared to a constant learning rate baseline. Both momentum and constant rate models had a fixed η of 0.001. The α parameter controls the effect of the *velocity* in the current direction, while descending the gradient. Figure 4 shows how changing the α parameter affects learning, as compared to a constant rate baseline. As alpha increases the momentum drives the steps more confidently towards the previously computed direction, effectively reaching better performance faster than otherwise. Momentum coefficient scheduling was thereafter implemented. Figure 6 shows the performance of a NN trained with momentum scheduling against that of a constant momentum coefficient. The hyper-parameter values for

both models were the best found when searching over each hyper-parameter space in turn.
Momentum scheduling performs consistently better over the 100 epochs effectively making the velocity contribute less initially, and increasingly more as the error gets closer to a minimum. The minimum validation error on momentum scheduling is 0.0855 while for constant momentum it is of 0.1703. Although the run time of both models is similar ($\approx 170sec$), the hyper-parameter search for the scheduling is considerably more expensive. However, the increase in performance over the validation set is non trivial (validation Error ≈ 0.0848 smaller).

3.3 Adaptive learning rule

RMSProp, though slower to perform 100 epochs over the data set (≈248sec), converges much sooner, and reaches similar performance to other methods after only 30-40 epochs. At its minimum validation error, RMSProp, in fact, reaches the highest accuracy (see Table 1).
Adam was trained and compared to the RMSProp. Figure FIGURE shows how the validation error, for two models trained with RMSProp and Adam respectively, changes over 100 epochs. Adam appears to be more stable in its validation change, a feature given by the tendency of the momentum additional parameter to drive the direction of the gradient step towards the current direction (effectively being more *reluctant* to abrupt changes). RMSProp, however, seems to be reaching a lower minimum.

4 Discussion

The performance on hand-written digit recognition for a three-layer Neural Network trained with different learning rules was shown to be somewhat variable. The increase in complexity of the algorithms did not always translate in considerable gains in accuracy, if not perhaps convergence time.
Time-scheduling the learning rule hyper-parameter improves on the constant learning rule baseline, though not by much (≈ 0.004). Scheduling however is not expensive and the improvement, depending on the application, might be worth it.
Scheduling momentum improves much on a constant momentum learning rule, achieving better performance on a validation set throughout the 100 epochs in which it is trained. The momentum drives the gradient in the previously found direction down the weight space, effectively making the error reach lower values sooner than otherwise, and keeping a better asymptotic behavior once reached. Scheduling the momentum allows initially for the steps to be barely driven by it, and to be increasingly more as the error gets closer and closer to a minimum. Training with momentum, however, is considerably more expensive than a constant learning rule scheduler ($\approx 20sec$), and the improvement in validation error or accuracy might not be worth the additional computing time.
The adaptive learning rates converge much sooner than the previously tried algorithms, although at the expense of a non-trivial increase in training time ($\approx 100sec$). RMSProp achieves top accuracy performance at ≈ 20 epochs. Arguably the momentum here contributes less to the step taken by the network down the gradient at time t. Adam accounts for momentum by introducing a new weighted average of the gradients, effectively reaching top performance yet faster than the previously implemented RMSProp. Even here, the expense a training time is non trivial, performing 100 epochs in twice the time needed for a constant rate base rule, but potentially only needing to do $\approx = 20$ epochs to reach similar performance on a validation set.
It is fundamental to point out that the best set of hyper-parameter, throughout the experiments,

was achieved through a systematic search over their values. A more effective approach would be to perform a *grid search* over all combination of hyper-parameters, a search which would otherwise be too expensive, when the hyper-parameter space becomes grater than 2 dimensional. Figure 10 shows the error and accuracy change on a validation set over 100 epochs for each of the models described in the report, with the best found hyper-parameter settings.

APPENDIX 1: Figures

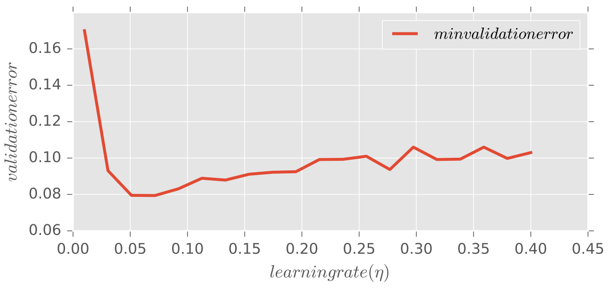
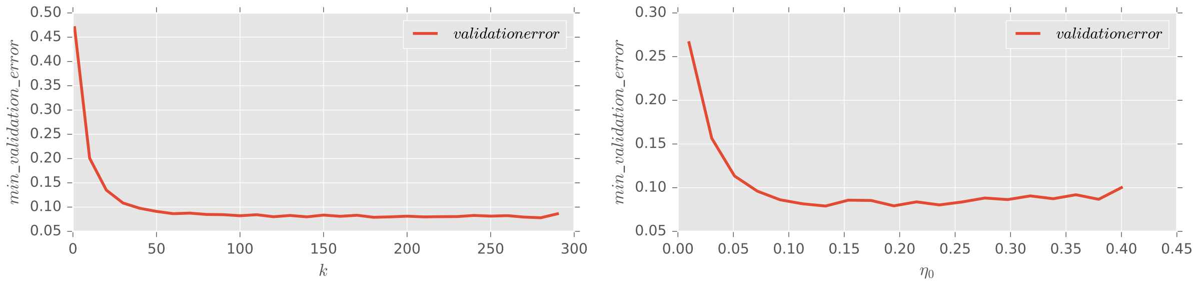


Figure 1: The figure shows the minimum validation error achieved by the NN, trained with a constant learning rule over 100 epochs, as the η parameter varies. The optimal performance on the validation set is achieved at $\eta = 0.07$, with a validation error $Err = 0.0794$ and an accuracy of $Acc = 0.9776$.

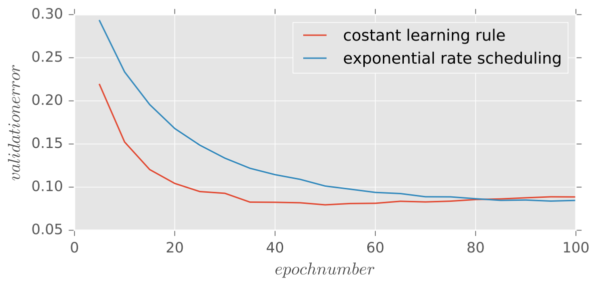


(a)

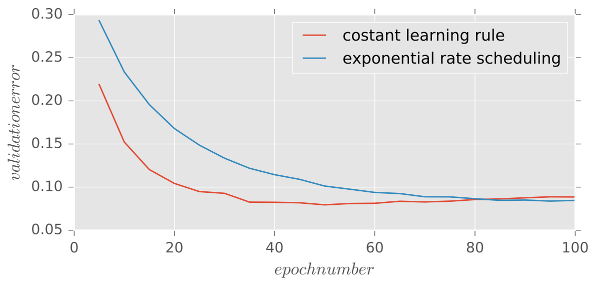
(b)

Figure 2: Figure a the minimum validation error achieved by the NN in 100 epochs, over different values of k, initially setting η_0 at 0.07. The value which minimizes validation error is $k = 100$ with an error $Err = 0.0822$ and an accuracy $Acc = 0.975$

Figure b shows the minimum validation error achieved by the NN in 100 epochs, over different values of η_0, setting k to the previously found optimal setting 100. The value which minimizes validation error is $\eta_0 = 0.13$ with an error $Err = 0.0790$ and an accuracy $Acc = 0.977$

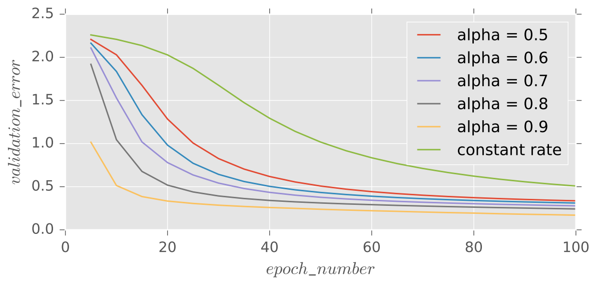


(a)

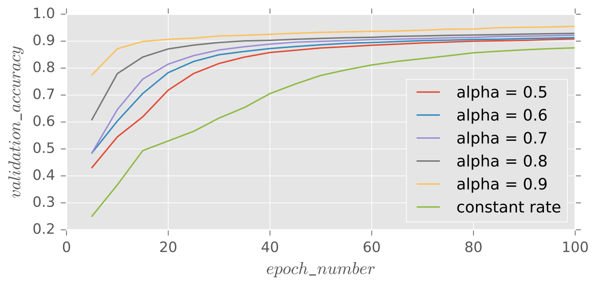


(b)

Figure 3: The figure shows the performance of a NN trained with a constant learning rate and a second trained with an exponential time scheduling learning rate, with the k parameter set to 100. Both models are initialized with the same randomly generated weight matrix and have a set learning rate of 0.12. Though initially the performance of the constant learning rule seems to be besting the exponential rate scheduler, it eventually starts over-fitting whilst the exponential rate scheduler reduces the weight adjustment steps enough to keep improving its performance on the validation set.

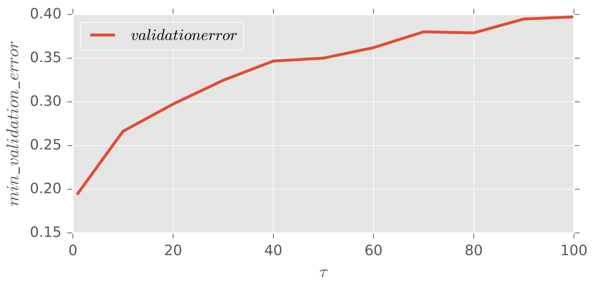


(a)

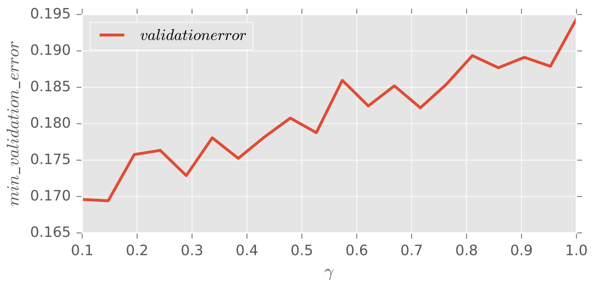


(b)

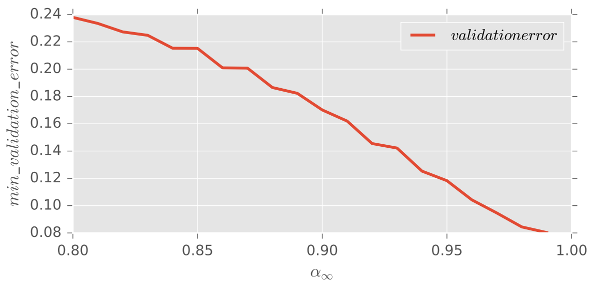
Figure 4: The figure shows the performance of a NN trained with momentum learning rule, at various alpha values, against the performance of one trained with a constant rate learning rule, both with a learning rate set to 0.001. From the graphs it is clear how increasing the alpha value, for a momentum learning rule, drives the error more *steeply* towards the minimum, effectively reaching it sooner than otherwise. The performance seems to be best at $\alpha = 0.9$. Values too close too one might render the gradient descent unstable as the momentum would dominate and the *descent* would keep the previously, possibly wrong, direction indeterminately.



(a)

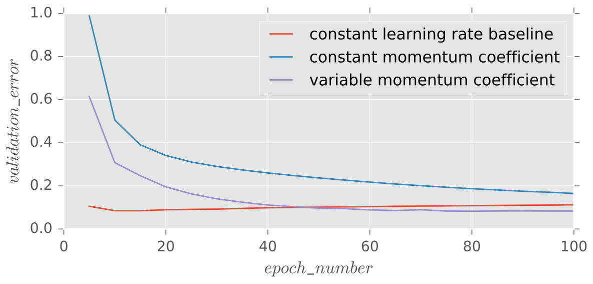


(b)

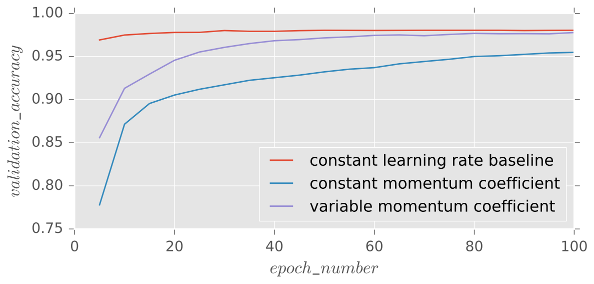


(c)

Figure 5: The figure shows the best performance achieved on a validation set, with a NN trained with scheduled momentum, at various τ (a), γ (b) and α_∞ (c) hyper-parameter values. The minimum were reached at $\tau = 1$,$\gamma = 0.15$ and $\alpha_\infty = 0.98$ reaching an overall validation error $Err = 0.0831$ and accuracy $Acc = 0.9782$.

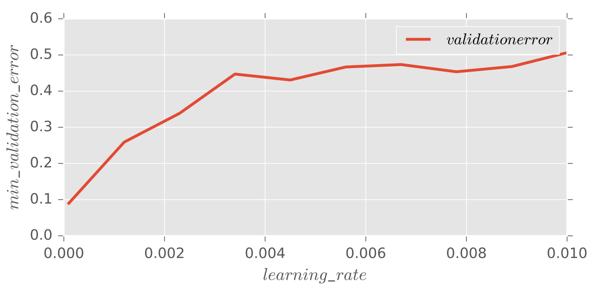


(a)

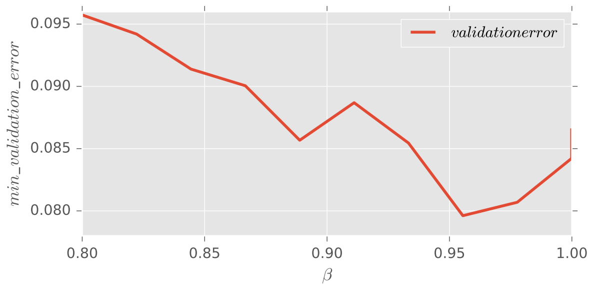


(b)

Figure 6: The figure shows compares the performance of a NN trained with constant momentum at $\alpha = 0.9$ and momentum scheduling with $\tau = 1$,$\gamma = 0.15$ and $\alpha_\infty = 0.98$. From the figures it is clear how the variable momentum learning rule makes the model perform consistently better throughout the 100 epochs.

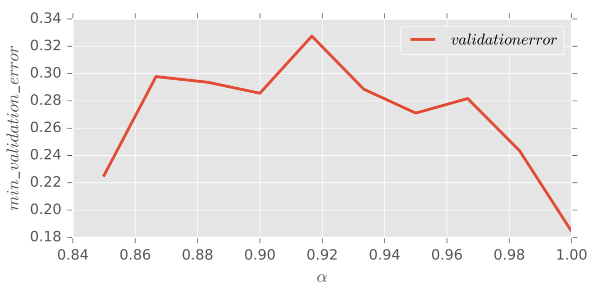


(a)

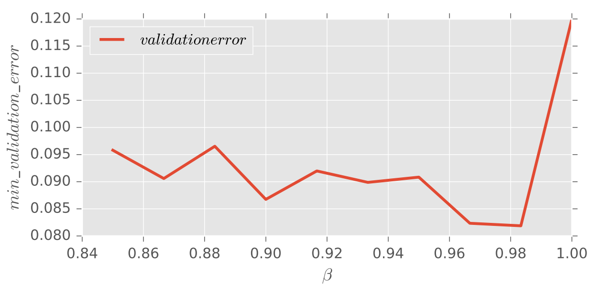


(b)

Figure 7: Figure a shows the minimum validation error achieved from a NN trained with RMSProp, at different values of η, setting $\beta = 0.9$. The model achieves optimal performance at $\eta = 0.0001$, with validation error $Err = 0.090$ and the accuracy $Acc = 0.974$. Figure b shows the minimum validation error achieved from a NN trained with RMSProp, at different values of the β, setting $\eta = 0.0001$. The model achieves optimal performance at $\beta = 0.9555$, with validation error $Err = 0.0796$ and the accuracy $Acc = 0.977$.

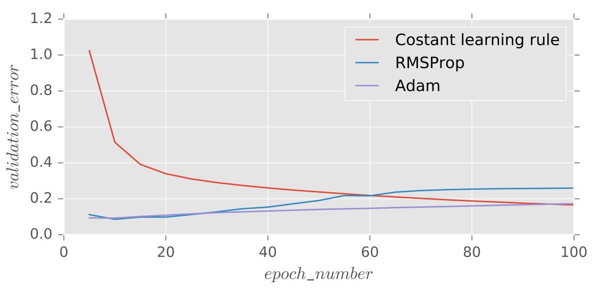


(a)

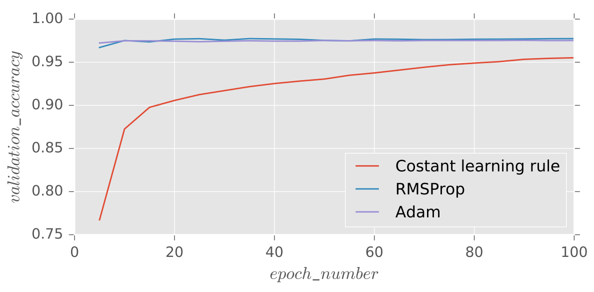


(b)

Figure 8: Figure a shows the minimum validation error achieved from a NN trained with Adam, at different values of α, setting initially $\beta = 0.999$. The model achieves optimal performance at $\alpha = 0.9578$, with validation error $Err = 0.1569$ and the accuracy $Acc = 0.9783$. Figure b shows the minimum validation error achieved from a NN trained with Adam, at different values of β, setting $\alpha = 0.9578$. The model achieves optimal performance at $\beta = 0.9899$, with validation error $Err = 0.0818$ and the accuracy $Acc = 0.9753$.

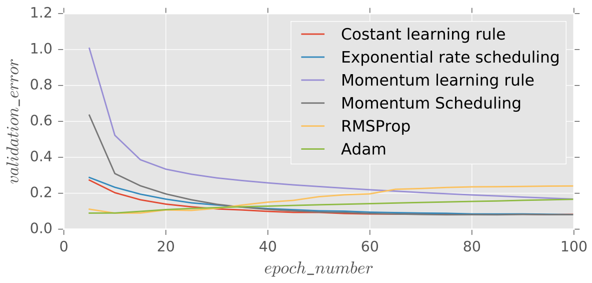


(a)

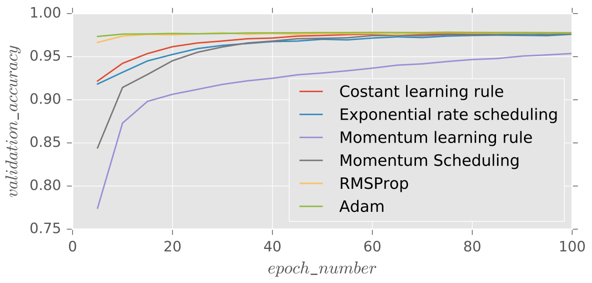


(b)

Figure 9: The performance of two NN trained with RMSProp and Adam, as compared to a constant learning rate baseline. All models were trained with the optimal found parameters, i.e.:
RMSProp $\eta = 0.0001$ and $\beta = 0.9555$; Adam $\eta = 0.0001$, $\beta = 0.9999$ and $\alpha = 0.9578$



(a)



(b)

Figure 10: The figure shows the performance of all the implemented learning algorithms, when starting from a common randomly initialized weight matrix.

APPENDIX 2: Tables

Table 1: The table shows the performance of the algorithms implemented in this report against the validation set, and the average training time for each model on the dataset. From the data it is clear how increasing the complexity of the learning rule introduces delays in the algorithm. The much slower performance of the adaptive learning rules, however, needs to be compared to the need, for other methods to do extensive hyper-parameter searches in order to reach similar performance. Moreover, both RMSProp and Adam reach similar performance to other methods much faster, i.e. if early stopping were implemented, the algorithms would need only run for but a fraction of the number of epochs needed by other methods to achieve similar performance.

Gradient Descent with:	**Validation Error (Minimum)**	**Accuracy (Maximum)**	**Average Run time**
Constant learning rate	0.0811	0.9775	143.3
Exponential learning rate scheduling	0.0812	0.9760	149.9
Momentum	0.1675	0.9537	147.8
Momentum scheduling	0.0802	0.9766	153.7
RMSProp	0.0893	0.9779	236.1
Adam	0.0895	0.9784	259.4