On-line Gradient Descent in Layered Neural Networks

P.J. Eilers(s2381575)
Neha Rajendra Bari Tamboli(s3701417)
Group 25

October 2019

Introduction

Neural networks are used for the task of classification and regression. But while doing so, the training can get stuck in plateau states. In these states the order parameters do change or the changes are not significant enough. The most obvious consequence of this is delayed convergence which can lead to computational inefficiencies.In such cases there are a variety of techniques that can be applied to break through these states by the process of hidden-unit specialization. Our primary interest lies in trying some optimisation techniques to overcome these plateau states. We try tuning the learning rates, applying momentum. Meaning the neural network will update the velocity with a beta factor and apply this to the update the weight vectors and push itself out of these plateau states. We also tried Nesterov accelerated gradient technique. We discuss these techniques in detail in section 3.3. We also explain an adaptive learning rates method that we explored. The results are explained per section, followed by conclusion. For these reasons, our research question is: "What is the influence of momentum,learning rate schedules and Nesterov accelerated gradient techniques on the properties of plateau states in shallow neural networks?". The interest lies in making convergence fast, which means more efficient computations.

1 Methodology

1. Generate input vectors $\xi \in \mathbb{R}^N$ with large enough N, since in theory we use infinitely large N where N refers to the number of samples, however this is not possible.

2. We apply Monte-Carlo simulations on the input vectors with a weight vector for the teacher network. This computes a rule which the student networks can use to learn and adapt their own weights to be as close as possible to teacher weight vectors. Each student weight vector should be orthogonal to one teacher weight vector, there is no fixed teacher weight vector to which it is orthogonal as it varies per run.

3. This will produce the new student weight vector for the next timestamp: $J_k^{\mu+1} = J_k^{\mu} - \frac{\eta}{N}.\nabla_{J_k}\epsilon(\{J_i^{\mu}\}, \xi^{\mu})$, where η is the learning rate[1]. We can plot the order parameters and the generalization error to observe on-line gradient descent which are described in further sections.
The order parameters are:

$$R_{in} = J_i \cdot B_n \quad Q_{ik} = J_i \cdot J_k \quad n = 1, \ldots, M \quad i, k = 1, \ldots, K \tag{1}$$

The generalization error, derived for the activation function $g(x)$ =erf($x\sqrt(2)$)

is given by:

$$\epsilon_g = \frac{1}{\pi}\left\{ \sum_{i,j=1}^{K} sin^{-1}\frac{Q_{ij}}{\sqrt{1+Q_{ii}}\sqrt{1+Q_{jj}}} \right.$$
$$+ \sum_{n,m=1}^{M} sin^{-1}\frac{T_{nm}}{\sqrt{1+T_{nn}}\sqrt{1+T_{mm}}}$$
$$- 2\sum_{i=1}^{K}\sum_{j=1}^{M} sin^{-1}\frac{R_{ij}}{\sqrt{1+Q_{ii}}\sqrt{1+T_{jj}}}$$

where $T_{mn} = B_m \cdot B_n$ [2]. In the end result, when the learning is complete, for 2 hidden units, each student weight vector we be orthogonal with one teacher weight vector, and orthogonal with the other student weight vector as well, as seen in the graphs in section 2.

4. Plateau states occur if the order parameters and generalization error do not change (except for small fluctuations) for extended periods of time.

5. We explore different methods as mentioned in the research question to break these states.

2 Common Parameters

- We ran the simulations for μ iterations, it corresponds to 140,000 iterations in the current simulation.
- The number of samples (or dimensions) for each weight vector is 1000
- The default and optimal learning rate is 1.5
- K hidden units are used where K = 2, and the activation function used is sigmoid.
- The teacher weight and student weight vectors are created using randn function which is a Gaussian distribution of random numbers. There are 2 teacher vectors and 2 student vectors for this simulation, each with 140,000 samples.

- We then create isotropic teachers using the Gram-Schmidt orthonormalization process, we apply this technique to the teacher weight vector, we will generally concentrate on isotropic teachers as the examined plateau phenomena come out most clearly when learning a rule given by an isotropic teacher[1]. The Gram–Schmidt process takes a linearly independent set of vectors and generates an orthogonal set.

- For every μ iteration , each hidden unit is updates or calculates the order parameters, generalization error and gradient, following the gradient we update both the student weights.

- The order-parameter $R_in = J_i \cdot B_n$ is the dot product of the student weight vectors(J) and the teacher weight vectors(B). For the sake of simplicity we plot the dot product of first student weight vector with both the teacher weight vectors.

- The order-parameter $Q_ik = J_i \cdot J_k$ is the dot product of the student weight vectors with itself(J_i) and the other student weight vector(J_k). For the sake of simplicity we plot the dot product of first student weight vector with itself and the other student weight vector. Else the plot of order parameters gets tedious.

- alpha is $\frac{\mu}{N}$, which represents the time steps in all the graphs, where N is the length of our vectors.

3 Optimization techniques tried

3.1 Tuning Learning rates

We tried different learning rates and found that the most optimal one was 1.5. The learning rate controls how quickly the model is adapted to the problem. Different learning rates tried are $0.5, 1, 1.5, 10$ which are stored in a $Learningrates\_array$.

We run the simulations μ times for all the learning rates for every hidden unit.

3.1.1 Comparing results of various Learning_rates

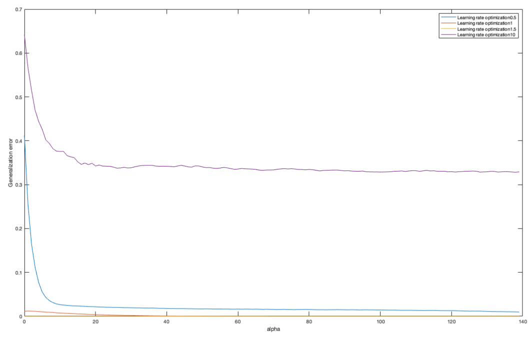


Figure 1: comparison of Generalization error with various learning rates

Figure 1 represents the plot for various learning rates, $learning\_rate = 1.5$ was the most optimal since it converged fastest. A learning rate of 1 converged to 0 but not as fast as 1.5, this can be attributed to the fact that the students took a longer time to learn the rule. The learning rate 0.5 was the slowest and did not converge, or it might converge but takes too long. Learning rate 10 performed the worst, since the model did not remain general anymore and overshootes the minima and hence never converges, due to this it has the highest generalization error.

We compare the order parameters for individual learning rates of $0.5, 1, 1.5, 10$ respectively. Figure 6, shows the order parameter for a $learning\_rate = 0.5$, the plateau states were present for a longer duration of time, in fact until the last time step in our simulation(i.e. 140) therefore not converging, even if the convergence takes place in the near future the number of time steps required are too high.

Figure 7 shows the order parameter for a $learning\_rate = 1$ the plateau states were present approximately till 100th time step before they are escaped. Although for a learning rate of 1 the plateaus states are escaped, but for a learning rate of 1.5 they are broken at around 70th time step which is much better and faster, this can be seen in Figure 8. This comparison of timesteps at which the the order parameters escape these plateau states can also be observed in the generalization error plot for individual learning rates of 1 and 1.5 as in figure 3 and 4, the descent continues once the plateau states are overcome and the alpha represents the time step at which they are overcome.

Figure 9 shows the order parameter for a $learningrate = 10$, the learning rate is too high in order for the gradient to continue descent after a certain period(in this case around 40th time step), and therefore it never converges and the performance diverges.This can be observed from the individual Generalization error plot in Figure 5, to check for the approximate time step at which the learning gets stuck at a plateau state.

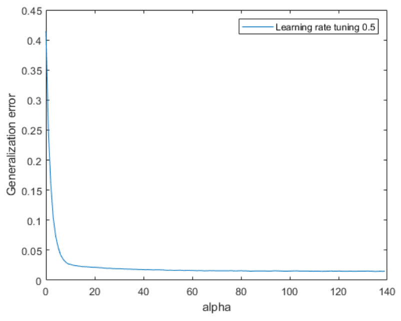


Figure 2: Generalization error for 0.5 learning rate

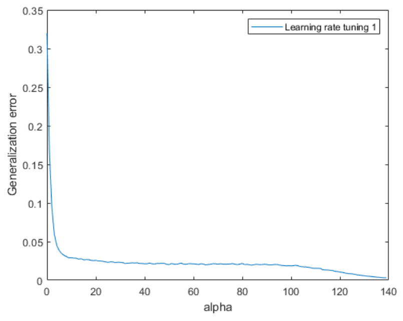


Figure 3: Generalization error for 1 learning rate

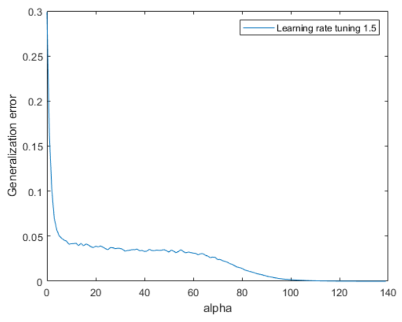


Figure 4: Generalization error for 1.5 learning rate

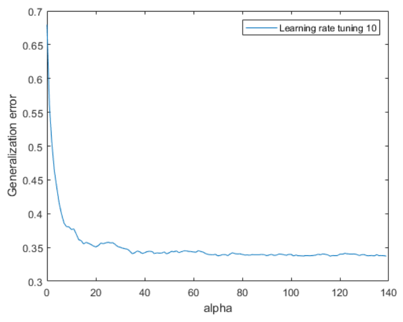


Figure 5: Generalization error for 10 learning rate

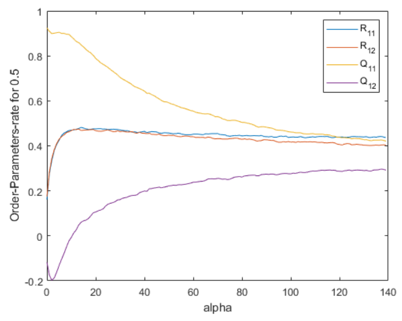


Figure 6: Orderparameter with 0.5 learning rate

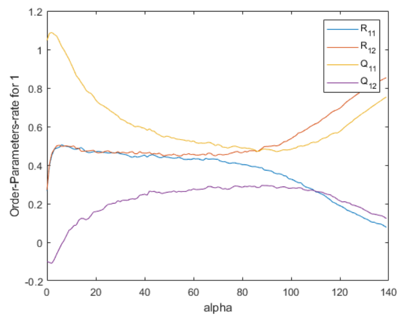


Figure 7: Orderparameter with 1 learning rate

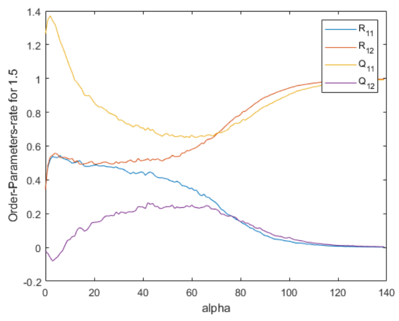


Figure 8: Orderparameter with 1.5 learning rate

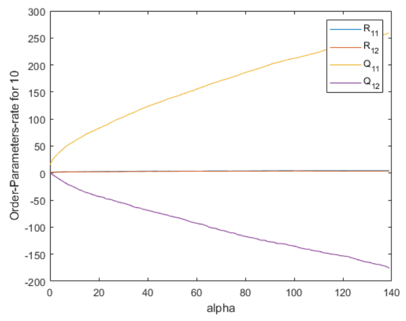


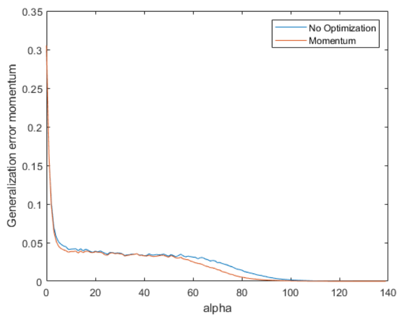
Figure 9: Orderparameter with 10 learning rate

3.2 Momentum

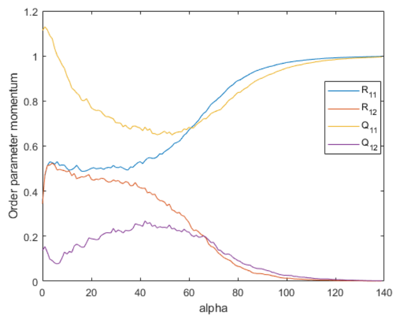
Momentum is a method that can speed up our Gradient Descent, by breaking through a plateau state faster. It uses a kind of moving average which takes into account the previous update steps [4] .

$$\begin{aligned} V_t &= \beta V_{t-1} + (1-\beta)\nabla_{J_k}\epsilon(\{J_i^\mu\}, \xi^\mu), \\ J_i^{\mu+1} &= J_i^\mu - \frac{\eta}{N}V_t \end{aligned} \tag{2}$$

Where V is the velocity, and β is a hyperparameter that represents momentum. The velocity is updated using equation 2, it does this by adding a fraction β of the update velocity of the past time step to the current update and the updated velocity is then used to update the student weight vector. Since generating random data can affect the comparisons between different methods, we use the same random seeds for all the calculations. Momentum helps to accelerate the gradient vectors in the right direction and attaining faster convergence. We set the momentum term β to a value of 0.9. We decrease this β over time, if we do not do this, the process seems to get stuck in a permanent plateau state. In Figure 10a, we can see that using momentum gives slightly better results compared to the normal way without any optimization, the length of the plateau state is slightly shorter.



(a)



(b)

Figure 10: (a) Comparison between the generalization error of gradient descent with and without momentum (b) Order parameters for momentum

3.3 Nesterov accelerated gradient

Nesterov accelerated gradient is a slightly different way to calculate momentum. When we use this version we're first looking at a point where the current momentum is pointing to and computing gradients from that point. See Figure 11 for a visual representation.

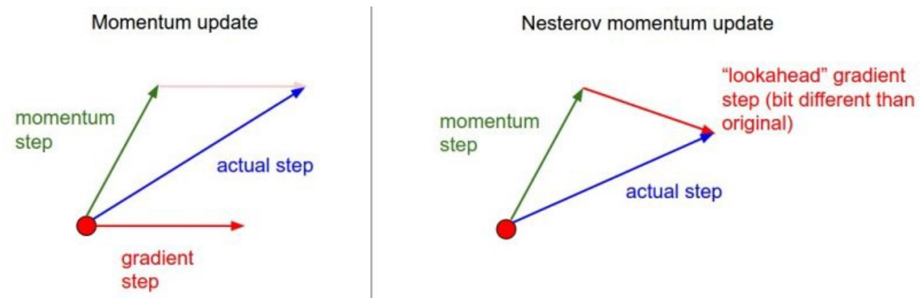
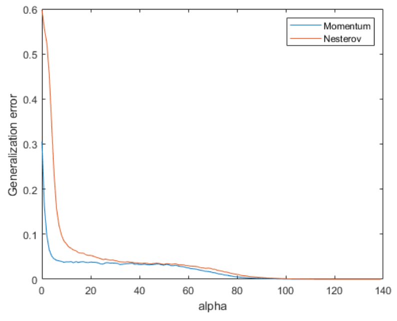


Figure 11: Visual representation of Nesterov Accelerated Gradient calculation, original from http://cs231n.github.io/neural-networks-3/

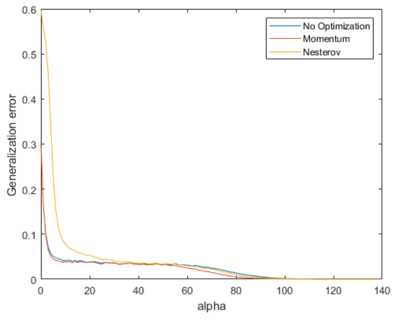
We can define Nesterov momentum using the follwing formulas:

$$\begin{aligned} V_t &= \beta V_{t-1} + (1-\beta)\nabla_{J_k}\epsilon(\{J_i^\mu - \beta V_{t-1}\}, \xi^\mu) \\ J_i^{\mu+1} &= J_i^\mu - \frac{\eta}{N} v_t \end{aligned} \tag{3}$$

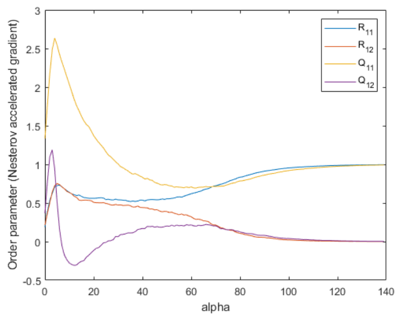
It gives us an approximation of the next position of the parameters. Thus the gradient calculated is w.r.t. to the future positions of the parameters and not the current positions. The above update formula updates the velocity. The updated velocity is then subtracted from the student weight vector as with our previous momentum calculation. This method seemed to have no improvement over general momentum, however, as seen in Figure 12a, standard momentum is the best performing optimization of the two. There does seem to be very slight improvement over using no momentum.



(a)



(b)



(c)

Figure 12: (a) Comparison between the generalization error of gradient descent with momentum and using Nesterov momentum (b) Comparison between no momentum(no optimization, LR = 1.5), normal momentum and Nesterov momentum (c) Order parameters for Nesterov momentum

3.4 Adaptive Learning Rates

We can use a dynamic learning rate to hopefully break plateau states faster, one method for this is using restarts. We can decrease our learning rate over time and reset it to its original value to hopefully jump from one local minimum to another [3]. Stochastic Gradient Descent with Restarts uses cosine annealing, which decreases learning rate in the form of half a cosine curve, as seen in Figure 13. As we get closer to our desired result, we probably want to restart less often, one solution for this is to increase cycle length over time.

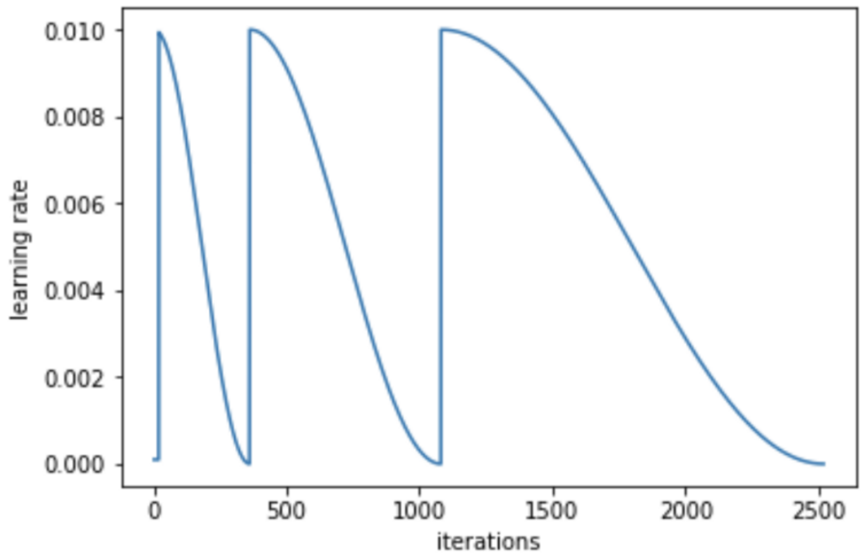
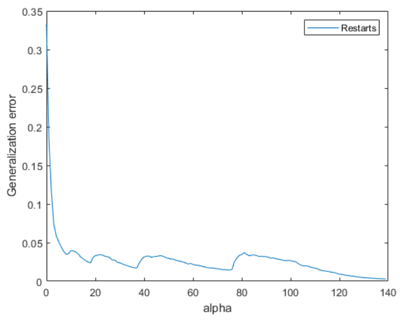
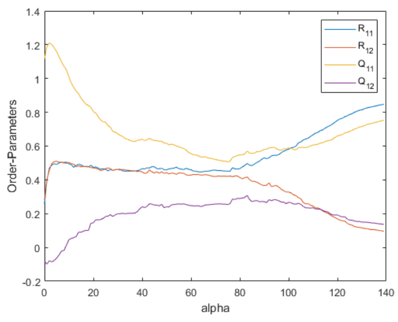


Figure 13: Represantation of our adaptive learning rate with restarts, where we increase cycle length over time. Original from https://course.fast.ai/

However, when we implemented this method, it had no positive effects, it only increased the length of the plateau states, as seen in Figure 14a. The generalization error increases again with each restart, and the order parameters stay in plateau states until there are no more restarts.



(a)



(b)

Figure 14: (a) Generalization error for SGDR, (b) Order parameters for SGDR

4 Conclusion and summary

We investigated the occurrences of the plateau states using 2 hidden units, in which the students learned a rule defined by a teacher. We analysed how, the learning process can get stuck in barren plateau states and discussed a few techniques to overcome it. We analyse the importance of an optimal learning rate. We try various techniques, out of which our implementation of momentum without using any other optimizations(like,Nesterov) seemed to give to best results with the shortest plateau states. Overall, if there was any improvement using techniques like momentum and Nesterov, over the methods without any optimization(i.e. when only a learning rate is used),it was very slight (Figure 12b). One reason for this could be that we used only 2 hidden units, so any improvement would seem small. However, the problem with using more hidden units is that the computational time is increased, especially when we are using the same random seeds for each method, which seems to be computationally intensive in MATLAB. Furthermore, when using more hidden units, more timesteps are needed, which also increase computation time. Stochastic Gradient Descent with Restarts uses a method to jump from one local minima to another, however, when we use 2 hidden units there is usually only one plateau state and therefore only one local minimum, so it makes sense that this method is ineffective with our initialization.

References

[1] Michael Biehl, Peter Riegler, and Christian Wöhler. "Transient dynamics of on-line learning in two-layered neural networks". In: *Journal of Physics A: Mathematical and General* 29 (Aug. 1996), pp. 4769–4780. DOI: 10.1088/0305-4470/29/16/005.

[2] Michael Biehl Elisa Oostwal Michiel Straat. "Hidden Unit Specialization in Layered Neural Networks: ReLU vs. Sigmoidal Activation". In: (Oct. 2019).

[3] Gao Huang et al. "Snapshot Ensembles: Train 1, get M for free". In: *CoRR* abs/1704.00109 (2017). arXiv: 1704.00109. URL: http://arxiv.org/abs/1704.00109.

[4] Ning Qian. "On the Momentum Term in Gradient Descent Learning Algorithms". In: *Neural Netw.* 12.1 (Jan. 1999), pp. 145–151. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(98)00116-6. URL: http://dx.doi.org/10.1016/S0893-6080(98)00116-6.