AN INVESTIGATION INTO THE EFFECT OF MOMENTUM ON AN ARTIFICIAL NEURAL NETWORK

Jack Harrison
9905871
Department of Physics and Astronomy
The University of Manchester

Poppy Nikou
9882816
Department of Physics and Astronomy
The University of Manchester

February 7, 2020

ABSTRACT

The effect of the momentum parameter on the convergence time of an artificial neural network (ANN) was investigated using the 'Breast-cancer-wisconsin' and 'Data_banknote_authentication' datasets, referred to as Set A and Set B. This report found that the introduction of classical momentum and Nesterov accelerated gradient reduced both the convergence time and the final error of the model. Momenta between 0 and 0.99 were tested, and the optimal found to be 0.8 for Set A and 0.99 for Set B. Both optimization techniques performed comparably, while both outperformed stochastic gradient descent - reducing the convergence time by up to 60%. The main limitation of this report was in using relatively small data sets, resulting in large errors over repeated tests.

***K*eywords** artificial neural network · stochastic gradient descent · momentum · Nesterov accelerated gradient

1 Introduction

Artificial neural networks (ANNs) are mathematical interpretations of biological neurons, in which an input signal triggers a response via a connected system of inner nodes. ANNs have been shown to be more flexible and yet more effective than many previous methods. Furthermore, it has been shown by Cybenko (1989) that a single layered ANN can approximate any continuous function [1]. As a result, neural networks are now commonplace for many classification tasks such as spam email detection and medical diagnoses.

With the emergence of ever larger datasets, traditional algorithms are often too slow at solving optimization problems on such a large scale and new methods are required. The most widely-used optimization algorithm is gradient descent, proposed by Cauchy (1847), and often used with random input examples to form stochastic gradient descent (SGD) [2]. Improvements to SGD now involve additional terms in the update equations, theoretically improving the algorithm's ability to reach a global minimum. This report aims to investigate the impact of the momentum parameter on a deep (multi-layer) neural network using two first order approaches - classical momentum (CM) and Nesterov accelerated gradient (NAG).

2 Background

A neural network consists of a series of layers made up of processing elements, known as nodes. A schematic diagram of a network is shown in Figure 1. The connections between nodes are given by a weighting, reflecting the importance of a node's output in the subsequent layer. Nodes are stimulated through an activation function, a hyperparameter that controls the response of each layer. Biological neurons are activated by a step function; triggered if a signal is received. In contrast, ANN layers commonly use differentiable activation functions, which is beneficial for assessing the impact of a given weight on the output. The value given to each node, a, is given by

$$a_{i+1} = g(\underline{w}_{i+1} \cdot \underline{x}_i), \tag{1}$$

where i is the layer number, g is the activation function for the layer, $\underline{w}_i$ is a vector of weights connecting to the node and $\underline{x}_i$ are the inputted values (or previous layer outputs).

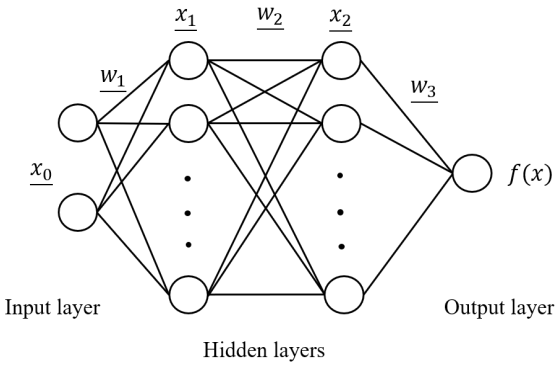

Figure 1: Schematic diagram showing the architecture of a simple neural network. Nodes in adjacent layers are connected via a weighting, determining the significance of a node in the subsequent layer. The values of the nodes $\underline{x}_i$ and weights $\underline{w}_i$ are used to determine the output, $f(\underline{x})$.

Examples are then fed through the network in a process called forward propagation, repeating the calculation in Equation 1 for every node throughout the network. In binary classification, a singular output node is used to give one value that corresponds to the probability of the example being in the upper class.

In order to assess the performance of the ANN, the cross entropy (or log loss), E, between the output and the target value is calculated. This is given by

$$E = -(y \log f(\underline{x}) + (1 - y) \log (1 - f(\underline{x}))), \tag{2}$$

where y is the actual classification and $f(\underline{x})$ is the output of the network for a given example $\underline{x}$. The objective of the network is to minimize the cross entropy, by updating the weights to give the most accurate output.

2.1 Backpropagation

Minimising the cross entropy relies on the ability to find the lowest point in a multi-dimensional weight space, known in 2D as the loss surface. This is done using SGD, in which weights are updated after each forward propagation step by

$$w_{i+1} = w_i - \alpha \frac{\partial E}{\partial w_i}, \tag{3}$$

where i is the iteration number, w_i is the weight for a given connection and α is the learning rate of the network. The most common analogy for this process is that of a walker descending to the bottom of a valley from an initial position. The path that the walker takes can be categorized using the size and direction of each step. The direction is determined by the derivative of the loss function with respect to each weight, and the step size given by the learning rate. The choice of learning rate is an important factor; too small a value and the weights will update too slowly, taking a long time to converge to the optimal solution. On the other hand, too large a learning rate and the network risks missing the optimal weights, 'stepping over' the minimum loss.

2.2 Classical momentum

Polyak (1964) proposed an improvement to SGD, using a momentum method in order to retain information about previous updates [3]. Within this, additional terms are added to Equation 3 giving

$$w_{i+1} = w_i - z_{i+1}, \tag{4}$$

and z_{i+1} given by

$$z_{i+1} = \alpha \frac{\partial E}{\partial w_i} + \beta z_i, \tag{5}$$

where β is the momentum parameter and z_i is the velocity term [4].

Qian (1999) showed that the inclusion of the momentum parameter extends the walker's descent analogy [5]. In this case, instead of a walker descending down a hill, a particle with mass, m, rolls down a hill. Using Newton's equation of

motion, Qian showed that m is equivalent to β. The larger the particle's mass, the less affected the particle's descent is by small valleys – allowing the particle to be carried over them to the bottom. Momentum has a similar effect on the descent down a loss surface, reducing the impact of small variations in height and allowing the parameter values to converge to the global minimum faster.

2.3 Nesterov accelerated gradient

Nesterov accelerated gradient (NAG) is a slight variation in the classical momentum (CM) method. Both are first order optimization algorithms, but instead of updating the weights with the momentum at the same time as the gradient update, the algorithm 'looks ahead' and finds an improved gradient at the momentum-updated weight position. This alters Equation 5 to

$$z_{i+1} = \beta z_i + \alpha \frac{\partial E(w_i - \beta z_i)}{\partial w_i}, \tag{6}$$

where $E(w_i - \beta z_i)$ is the new loss using momentum-updated weights [6]. This 'looking ahead' helps correct for mistakes caused by the momentum update after it has been carried out. This is particularly useful in the high momentum case, when the network is prone to oscillations while in a 'valley' at the base of the loss surface. NAG helps to dampen such oscillations through the corrections, reducing the convergence time.

3 Experiments

A neural network was constructed with an input layer, two hidden layers and an output layer. The number of nodes in the input layer was equal to the number of features in the dataset and one output node was used in order to carry out binary classification. The number of neurons in the hidden layers was chosen to avoid overfitting by following a suggested rule of thumb: the number of nodes in the hidden layers should be between the total number of input and output nodes [7]. The learning rate was kept constant between layers in order to keep tests consistent.

Hyperbolic tangent, or tanh, functions were chosen for the hidden layer's activation functions, in order to increase the sensitivity of the network over small input values, as compared with sigmoid functions. The added sensitivity is due to the tanh function's sharply peaked derivative, making it more effective during backpropagation. Sigmoid functions inside neural networks risk causing nodes to 'get stuck', since very negative values are mapped close to zero, effectively removing any updates in the nodes' weights. A sigmoid function was chosen for the output layer in order to produce outputs between zero and one, corresponding to the probability of predicting the upper class. This was then used alongside a threshold value of 0.5 to decide which class has been predicted.

This report used the 'Data_banknote_authentication' and 'Breast-cancer-wisconsin' datasets from the UCI ML repository, referred to as Set A and Set B respectively [8]. Both datasets were chosen as they are to be used for binary classification, with multiple features and a large number of examples, 1372 and 683 respectively.

Before experiments could be carried out, both datasets had to be preprocessed. Firstly, Set B contained examples with missing features. In total, these examples made up less than 2% of the dataset and therefore it was deemed acceptable to remove them. Secondly, since ANNs are not scale invariant both datasets were scaled using scikit-learn's MinMaxScaler. This subtracted the minimum value of each feature and divided by the range, giving values between 0 and 1. The MinMaxScaler was chosen over the StandardScaler, which results in zero mean and unit variance. Since the range of the features given by the StandardScaler is dependent on the magnitude of the outliers, one feature might be scaled in the range [-2, 2] and the other in the range [-0.5, 0.5]. Keeping all features within the range [0, 1] was found to reduce the errors throughout the experiments. The data was then randomly split 70:15:15 between the training, cross validation and testing sets using scikit-learn's TrainTestSplit. Since both datasets had approximately even class proportions, stratified data selection was not required and only a random selection method was used.

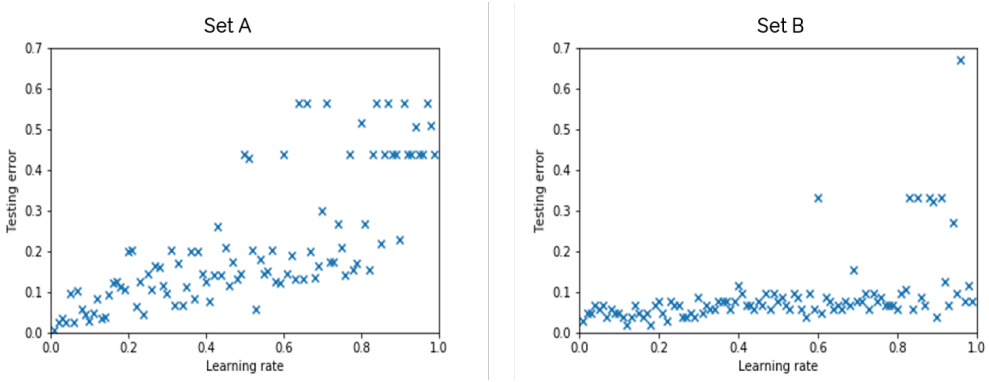

Figure 2: Scatter plots showing the increase in testing error (% of wrong classifications) with learning rate in an artificial neural network, for two datasets. The results are shown for two datasets: A and B. Both show fluctuations in the testing error at high learning rates, indicating that the network hasn't converged.

In order to benchmark the impact of momentum, a learning rate had to be chosen to keep constant throughout experiments. To do this, the network was trained using SGD over 100 epochs, shuffling the training data after each epoch. Learning rates from 0.001 to 1 were tested and the results shown in Figure 2.

For both datasets, learning rates below 0.1 allow the network to converge after 100 epochs. Above 0.2, the testing error begins to fluctuate, indicating that the algorithm is struggling to converge due to too high a learning rate. As a result, learning rates of 0.005 and 0.001 were chosen for Set A and Set B respectively. The values chosen were slightly lower than suggested in Figure 2, in order to emphasise the effect of momentum on the convergence time in subsequent experiments.

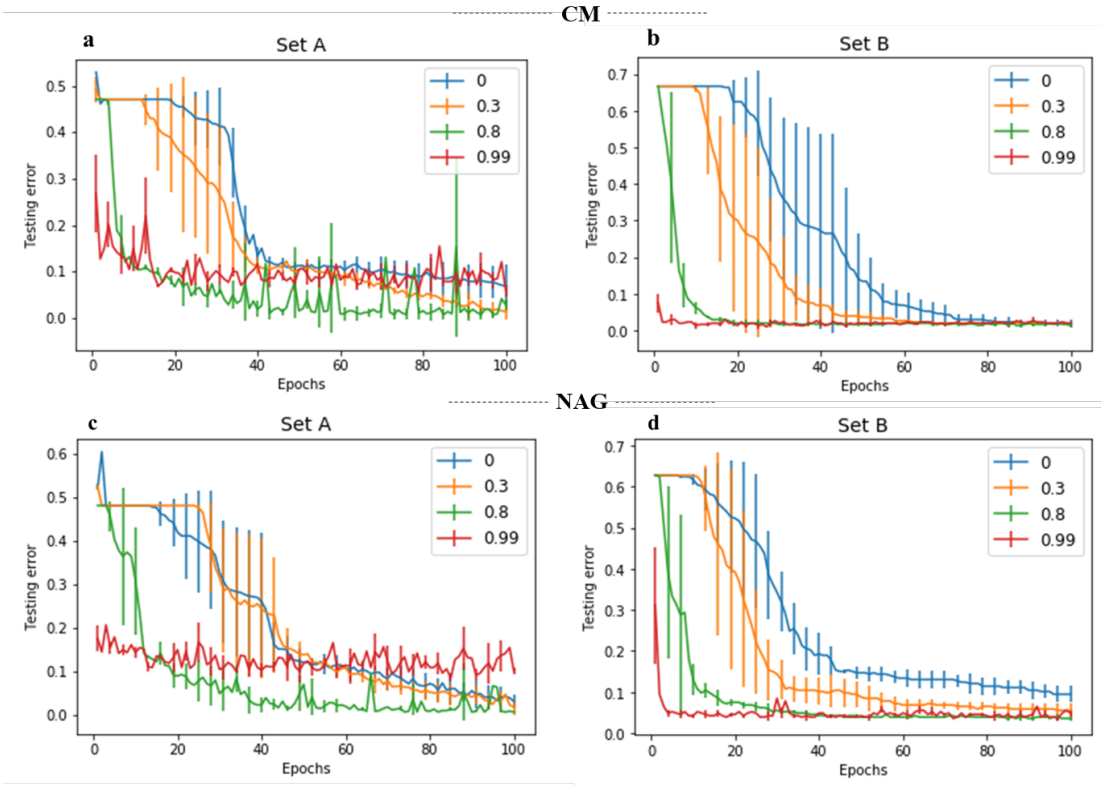

Figure 3: Graphs showing the effect of increasing the momentum parameter on the testing error (% of wrong classifications) of a neural network using: **a, b)** classical momentum (CM) and **c, d)** Nesterov accelerated gradient (NAG). The results are shown for two datasets: A and B.

CM and NAG were then separately added to the backpropagation algorithm. Momentum values in the range [0, 1] at regular intervals were used to observe the impact of each method. Figure 3 shows the effect of the different methods on the testing error of the two datasets for a range of values. Each datapoint is the average over three tests, with the standard deviation between tests assigned as the error. It is noted that with a β of 0, both algorithms become the original SGD.

In order to quantify the time to convergence, it was decided that the network was deemed to have converged when five epochs were observed at the same testing error. Since this was subjective, an error of five epochs was assigned to each reading. The time to converge was then given as the number of epochs until this limit was reached. Figure 4 shows the effect of momentum using this measure. Due to the random nature of the neural networks and the subjective readings, values are not intended to be exact and instead be used as an indication of trends over the range of momentum values.

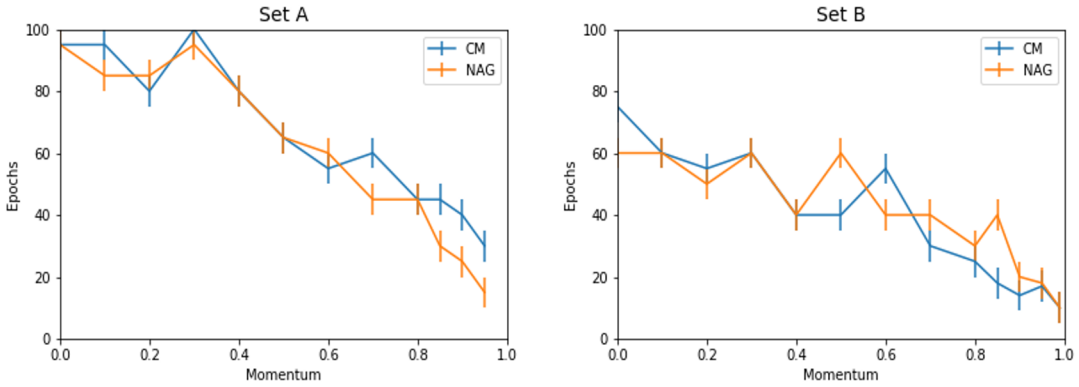

Figure 4: Graphs showing the number of epochs needed to converge to a minimum against momentum for classical momentum (CM) and Nesterov accelerated gradient (NAG) optimizers. The results are shown for two datasets: A and B.

Figure 4 clearly shows that both NAG and CM improve the convergence time of the network as compared to SGD. In both cases, the addition of momentum improves the convergence time by up to 60%. It is noted that momentum values that did not converge have been omitted from the graph.

4 Discussion

When deciding the learning rate in Figure 2, an approximate quadratic shape was expected with the testing error showing a visible minimum and remaining high over extreme values of learning rates. However, Figure 2 only shows an increasing testing error with learning rate. When using SGD over n epochs with m examples, the network will have updated $n \times m$ times by the end. For our datasets, the network is able to converge after this many updates for even the smallest learning rate tested. Using smaller learning rates would have shown the expected behaviour.

Figure 3 shows how the introduction of CM and NAG changes the convergence of the network after each epoch. Setting β to 0, shown by the blue line, is equivalent to only using SGD during backpropagation. The blue lines reinforce the idea that the chosen learning rate was smaller than optimal, since in each subplot the line doesn't converge to the global minimum. Figure 3d) highlights this with the network only reducing to a final error of 0.1 and getting stuck for several epochs at a local minimum, an issue SGD is prone to.

All the subplots in Figure 3 follow a general trend: increasing β decreases the convergence time. Furthermore, increasing β appears to decrease the final testing error. This is a result of the higher momentum 'carrying' the descent over local minima and allowing it to converge to the global minima faster. An anomaly to this trend are the $\beta = 0.99$ tests on Set A. The momentum here is too high, causing the descent to oscillate around a separate local minimum. It was expected that the introduction of NAG would account for this but, despite visually reducing the oscillation size, NAG still cannot escape the local minimum due to the high momentum 'pulling' it back [4]. In Figure 3b) however, $\beta = 0.99$ appears to cause a slight overfitting. Instead of missing the global minimum, the network appears to find the global minimum within 15 epochs. Over the subsequent epochs the network continues to learn on the training data, causing overfitting and a gradual rise in testing error. This difference between Set A and Set B highlights that the optimal momentum value is dependent on the data.

Figure 4 shows the number of epochs to convergence for the full range of momenta tested, allowing for a clearer comparison of the optimizers. For Set A, neither optimizer reduces the number of epochs until the momentum is higher than $\beta = 0.3$. Although β close to one converges the fastest, it converges to a local minimum resulting in a higher final test error, as shown in Figure 3a) and 3c). The optimal β for Set A was therefore determined to be 0.8. For Set B, the number of epochs decreased steadily with β, as well as the final testing error of the model. Therefore the optimal value was chosen to be 0.99. When comparing CM and NAG, neither outperformed the other and as such the optimal momentum value was taken as the same for both optimizers. Both optimizers gave large improvements on the results obtained from SGD, lowering the final error by up to 5% and decreasing the convergence time by up to 60%.

Both CM and NAG are first order optimization algorithms; only containing first order differential equations. In this report, both are shown to improve the convergence time and the final error of the model as compared with SGD. A further improvement to the network could be the use of a second order optimizer such as Newton, an algorithm which exploits the Newton Raphson method of obtaining the roots of a function. The second order differentials provide additional curvature information which help to adaptively change the step length of the optimization trajectory during backpropagation.

5 Conclusion

An artificial neural network was successfully trained on the 'Breast-cancer-wisconsin' and 'Data_banknote_authentication' datasets, Set A and Set B respectively, from the UCI ML repository [8]. Classical momentum (CM) and Nesterov accelerated gradient (NAG) were added to the network's stochastic gradient descent algorithm, and the effect on the convergence time investigated. It was found that both CM and NAG significantly improve the convergence time, with a reduction of up to 60% over tests. However, no significant difference was found between CM and NAG. The optimal momentum parameter was therefore found to be 0.8 for Set A and 0.99 for Set B. It was concluded that the inclusion of both CM and NAG improves the convergence time for the majority of momentum values, but tuning of the hyperparameter is still required for best improvements. A further extension would have been the use of second order optimizers within the network such as Newton, in order to further improve backpropagation.

References

[1] Cybenko G. (1989) 'Approximation by superpositions of a sigmoidal function', *Mathematics of Control, Signals and Systems*, 2(4), pp. 303-314.

[2] Robbins H, Monro S. (1951) 'A Stochastic Approximation Method', *The Annals of Mathematical Statistics*, 22(3), pp. 400-407.

[3] Polyak B. (1964) 'Some methods of speeding up the convergence of iteration methods' *Ussr Computational Mathematics and Mathematical Physics*. 4(5).

[4] Sutskever I., Martens J., Dahl G., Hinton G. (2013) 'On the importance of initialization and momentum in deep learning', *Journal of Machine Learning Research*, 28, pp. 1139-1147.

[5] Qian N. (1999) 'On the momentum term in gradient descent learning algorithms', *Neural Networks*, 12(1), pp. 145-151.

[6] Bengio Y., Boulanger-Lewandowski N., Pascanu R. (2012). Advances in Optimizing Recurrent Networks. *Acoustics, Speech, and Signal Processing*

[7] Heaton J. *Introduction to Neural Networks with Java*, 2nd ed. : Heaton Research, Inc.; 2008

[8] Dua D. , Graff C. (2019) *UCI Machine Learning Repository*, Available at: http://archive.ics.uci.edu/ml (Accessed: 1st November 2019).