Evaluation of Gradient Optimization Techniques for Neural Networks

Aishwarya Rajuladevi Manikyam, Sakshitha Channadi, Divya Bhuvanapalli, and Bhavya Induri
Department of Computer Science, Georgia State University

***Abstract*—Gradient descent technique is the most common and popular method for gradient optimization of neural networks. In this paper, we test and compare the stability, speed and accuracy of the following gradient optimization techniques, viz.: a regular network without momentum, Polyak's classical momentum, RmsProp and Adam. We analyze and discuss the theoretical convergence properties of each of these techniques. We perform tests to analyze how the techniques behave for different learning rates and mini-batch sizes. In our experiments we observe that Adam gives the best results in terms of highest speed of convergence and has an accuracy of 89.608%.**

***Index Terms*—Stochastic Gradient Descent (SGD), Polyak's momentum, RmsProp, ADAM, mini-batch.**

I. Introduction

Neural networks is a run-of-the-mill solution to most classification and prediction problems. Although, a lot of progress is made by fine-tuning the techniques, gradient descent is a very commonplace, but, yielding technique. Having said that, in every learning-based application, we come across manifold implementations of gradient optimization techniques. Stochastic gradient descent and mini-batch gradient descent show better performance when compared to its other variants. In this paper, we focus on studying the different intuitions and behaviors of gradient optimization algorithms. We implement a three-layer fully connected neural network and test the model on the fashion-MNIST dataset. We briefly discuss the various gradient descent optimization algorithms like Polyak's classical momentum, Nesterov's accelerated gradient, Adagrad, Adadelta, RMSprop, Adam, AdaMax, Nadam and AMSGrad. Use of momentum technique would accelerate SGD by progressing in relevant direction, especially, in steeper regions of the curve towards the local optima. We implement a basic no-momentum network to analyze and compare the subsequent techniques. In Nestrov's accelerated gradient, the gradient approximation is not based on current parameters, but, on the future values of the parameters. Hence, this adds an intuition to the approximation based on the position of the curve by slowing down the pace before the curve slopes again. Adagrad algorithm is most suitable for sparse datasets. Instead of updating all parameters at once, Adagrad performs smaller updates for frequent features and significant updates for less occuring features to adapt the learning rate. RMSprop is very similar to Adagrad but overcomes the problem of radically reducing learning rates by changing the dividing factor to an exponentially decreasing function. Adelta is also an extended version of Adagrad which reduces the overhead of storing all previous squared gradient values and accumulates only as many as a fixed window size. Adam is another adaptive moment estimation technique which combines the exponentially decaying average of previous squared gradients of RMSprop and exponentially decaying previous gradients of momentum technique. AdaMax and Nadam are slight variants of Adam technique. For the purpose of study, in this paper, we restrict our analysis to a regular network without momentum, Polyak's classical momentum, RmsProp and Adam an adaptive momentum estimation. We explain the above algorithms in detail in section III and discuss the analysis based on parameters such as speed, robustness and accuracy in section IV and V of this paper.

II. Related Work

Gradient descent is a generic method for continuous optimization, so it can be, and is very commonly, applied to non-convex functions. We review some of the related work on non-convex stochastic optimization. A randomized stochastic gradient (RSG) method was proposed by Ghadimi and Lan (2013) [1] which achieves $O(1/T + \alpha^2/ T)$ convergence rate, where α^2 is an upper bound on the variance of stochastic gradient. Yang et al. (2016) [2] provided a unified convergence analysis for both stochastic heavy-ball method and the stochastic variant of Nesterovs accelerated gradient method, and proved $O(1/ T)$ convergence rate to a stationary point for smooth non-convex functions. Another variant of stochastic gradient called stochastic variance-reduced gradient (SVRG) method was proposed by Reddi et al. (2016) [3], Allen-Zhu and Hazan (2016) [4] that is faster than gradient descent in the non-convex finite-sum setting. Recently, Allen-Zhu (2018) [5] proposed an SGD4 algorithm, which optimizes a series of regularized problems, and is able to achieve a faster convergence rate than Stochastic gradient descent (SGD).

Some of the optimization methods that have a direct relation to the methods discussed in this paper are AdaGrad (Duchi et al., 2011) [14], AdaDelta (Zeiler, 2012) and natural Newton method from Roux & Fitzgibbon (2010). There are also other methods like the Sum-of-Functions Optimizer (SFO) (SohlDickstein et al., 2014) which have memory requirements and is in-feasible on systems without a GPU.

AdaGrad (Adaptive Gradient) [14] is an optimization technique in which the learning rate varies in an adaptive manner based on the parameters. Since it depends primarily on the parameters, it performs significant updates for less occurring parameters and smaller updates for frequent parameters. This works well for sparse data. In AdaGrad, the

learning rates are decreasing because at each iteration, the learning rates are calculated as one divided by the sum of square roots. This causes a huge problem because after a certain number of iterations, the learning rate is so small that the system stops learning and the gradient becomes a constant.

AdaDelta is an optimization technique which was devised to solve the problems of AdaGrad. Instead of adding all the squared roots in the denominator, it uses a sliding window which causes the sum to decrease [15].

III. METHOD

In this paper we build a three-layer fully connected neural network in order to analyze the different gradient techniques on the fashion-MNIST data set [13]. It consists of one input layer, two hidden layers and one output layer. The first step is to group or rather split the data set into training (50,000 samples), validation (10,000 samples) from a total of 60,000 samples. The test sets consists of 10,000 samples. We initialize the weights and biases randomly and multiply it by a factor of 0.01. The next step is to form mini-batches from the train and validation sets. We discuss the results of different mini-batch values in the experiment and results section. For each minibatch we compute the feed forward, the costs for train and validation data and then compute the gradients in the backpropagation step. We implement ReLU and linear activation functions for (L-1) layers and implement a softmax activation function through the network just before the final (output) layer which computes probability for each class. This sums up the gradient descent part of the algorithm. The optimization techniques come into picture when updating the parameters (weights, biases) for the layers. We discuss the updating procedure for each optimization technique in detail below.

A. No Momentum

No Momentum is one of the basic techniques of Gradient Descent which does not include any acceleration component. Each mini batch undergoes one forward propagation, calculation of loss function and one backward propagation. The back-propagation updates the parameters and next mini batch is passed to the network with updated parameters. The update process is very elemental and involves subtracting the parameters by the product of the learning rate and gradients.

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t)$$

$\nabla J(\theta_t)$ is the derivative of loss function with respect to parameters and α is the learning rate, θ are the parameters of the network i.e. weights and bias.

B. Polyak's Classical Momentum

Gradient descent with momentum is a stochastic optimization technique. This came into use because vanilla gradient descent(gradient descent with no momentum) had problems navigating ravines(areas where the surface curves more steeply in one dimension).It oscillates continuously across the slopes of the ravine and made very little progress along the path to the local minima. This led to a much longer convergence time for such surfaces [7]. In the momentum technique, we first define a term momentum, which is a moving average of the gradients. In each iteration,this momentum is used to update the weights of the network [16]. This helps accelerate the gradient vectors in the right direction but at the same time, also suppresses oscillations in the irrelevant directions thus leading to faster convergence in comparison with vanilla gradient descent[7].

$$V_{t+1} = \beta V_t + (1 - \beta)\nabla J(\theta_t)$$

$$\theta_{t+1} = \theta_t - \alpha V_{t+1}$$

V_t - exponentially weighted gradient

C. RMSProp

RMSprop [10] is an adaptive learning rate gradient-based optimization technique proposed by Geoffrey Hinton. This optimizer restricts the oscillations in the vertical direction, thus decreasing the number of iterations and increasing the learning rate for faster convergence [10]. It is derived from Adagrad which was widely used to optimize the gradient descent. This technique adapts the learning rate by performing updates based on the frequency of occurrence of features. The idea is to perform smaller updates on features having high frequency and large updates on features having low frequency. Thus, eliminates the problem of radically reducing gradients which diminishes the learning ability of the network. Another drawback of Adagrad is that, it accumulates the squared gradients in the denominator which consequently makes the denominator directly proportional to the number of iterations. This results in infinitesimally small learning rates as the denominator keeps growing, whereas, RMSprop divides the learning rates by an exponentially decaying average of the squared gradients.

$$G_t \leftarrow \beta_2 . G_{t-1} + (1 - \beta_2) . \nabla J(\theta_t)^2$$

$$\theta_1 \leftarrow \theta_{t-1} - \alpha . \hat{V}_t / (\sqrt{\hat{G}_t} + \epsilon)$$

G_t - exponentially weighted average of squared gradients

D. Adam

Adam is a stochastic optimization method closely related to RMSProp and Adadelta. This method uses only first order gradients with little memory requirement and is well suited for problems that are large in terms of data and parameters. In Adam, we calculate momentum changes for each parameter.

Algorithm:

Require: α : Stepsize

Require: $\beta_1, \beta_2 \in [0,1)$: Exponential decay rates

Require: $f(\theta)$: Stochastic objective function
Require: θ_0: Initial parameter vector
$V_0 \leftarrow 0$ (Initialize first moment vector)
$G_0 \leftarrow 0$ (Initialize second moment vector)
$t_0 \leftarrow 0$ (Initialize timestep)
while θ_t not converged do
t ← t + 1
$\nabla J(\theta_t) \leftarrow \nabla_\theta f_t(\theta_t - 1)$ (Get Gradients)
$V_t \leftarrow \beta_1.V_t - 1 + (1 - \beta_1).\nabla J(\theta_t)$ (Update biased first estimate)
$G_t \leftarrow \beta_2.G_t - 1 + (1 - \beta_2).\nabla J(\theta_t)^2$ (Update biased second raw moment estimate)
$\hat{V}_t \leftarrow V_t/(1-\beta_1^t)$ (Compute bias-corrected first moment estimate)
$\hat{G}_t \leftarrow G_t/(1-\beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_1 \leftarrow \theta_{t-1} - \alpha.\hat{V}_t/(\sqrt{\hat{G}_t} + \epsilon)$ (Update parameters)
end while
return θ_t

The algorithm [11] updates exponentially moving averages of the gradient V_t and the squared gradient G_t where the hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages. The moving averages themselves are estimates of the 1st moment (the mean) and the second raw moment (the non-centered variance) of the gradient. During the initial iterations, when the decay rates are small, the moment estimates that are biased tend towards zero. This is due to the initialization of moving averages to vectors of zeros. However,this can be counteracted by the bias-corrected estimates $\hat{V}_t$ and $\hat{G}_t$.

IV. Experiments

In this section, we discuss the various tests conducted for different learning rates and mini-batch sizes and we analyze the train and validation costs. Also, for every test we report the train, validation and test accuracies. First, we analyze the train and validation costs for different values of learning rates and the plots below depict the cost curves for the specific learning rates. These learning rates are chosen based on the best reported accuracies from our tests. We observe that No-Momentum, Momentum and Adam methods perform very well for learning rate 0.2, whereas, in case of RMSProp the cost shoots up and then decreases for the same learning rate. RMSProp reports good accuracy for learning rate 0.008.

Figures 1 to 4 show these train and validation curves. Next, we analyze the behavior of different methods by keeping the learning rate constant. We plot the validation costs for each technique by varying the learning rate as 0.01, 0.1 and 0.2.

Figure 5 indicates the validation costs of all techniques against iterations for learning rate 0.01.Similarly figure 6 and 7 depict the same for learning rates 0.1 and 0.2 respectively. Lastly, we analyze the behavior of each technique for different sizes of mini-batches when the learning rate is fixed at 0.1.

Figure 8 depicts test accuracies of all the four techniques for various mini batch sizes ranging from 50 to 1000. Figure 9 depicts validation accuracies of all the four techniques for various mini batch sizes ranging from 50 to 1000.

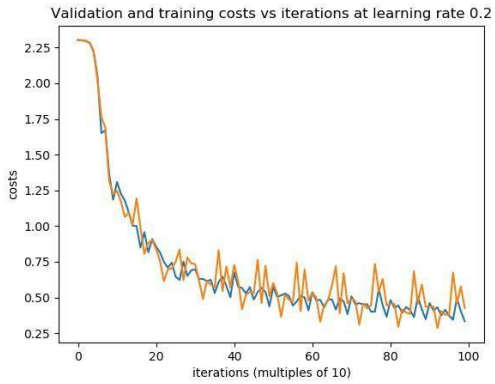

Fig. 1: Train, Validation and test curves for the best case of No Momentum

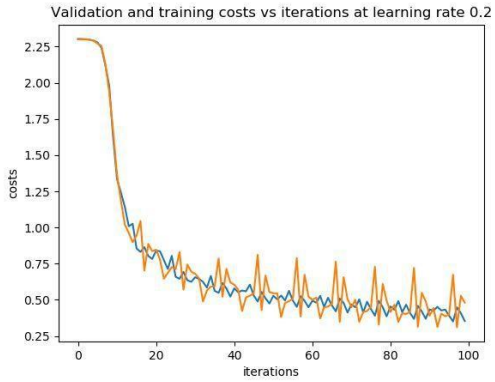

Fig. 2: Train, Validation and test curves for the best case of Momentum

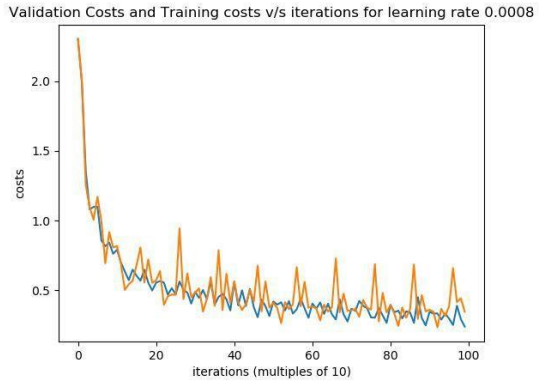

Fig. 3: Train, Validation and test curves for the best case of

RMSProp

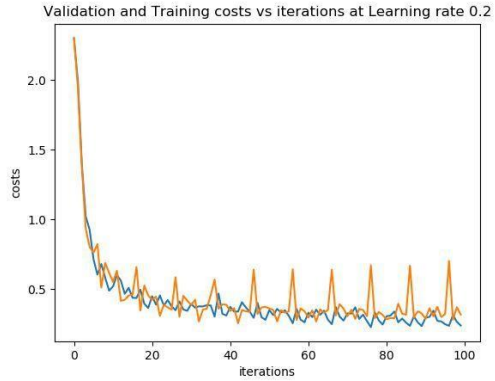

Fig. 4: Train, Validation and test curves for the best case of Adam

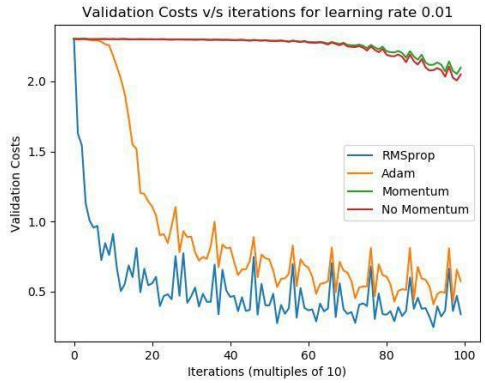

Fig. 5: Learning rate 0.01 plot of validation costs of all techniques

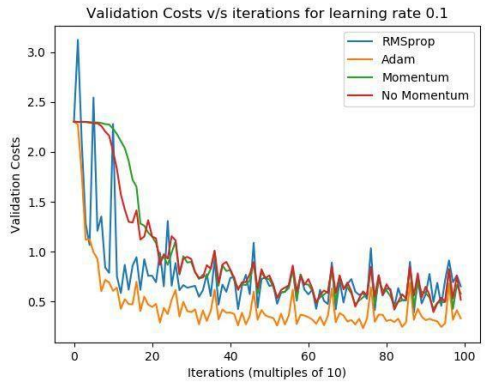

Fig. 6: Learning rate 0.1 plot of validation costs of all techniques

In the above graphs we can see that Momentum and NoMomentum techniques perform poorly for learning rate 0.01.

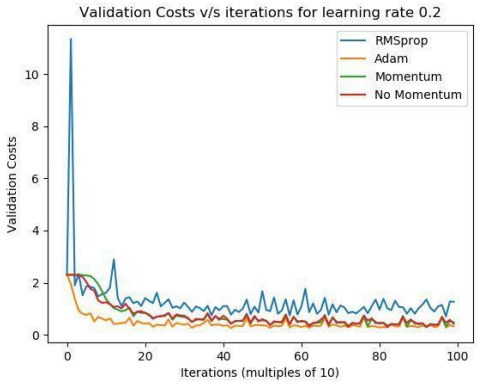

Fig. 7: Learning rate 0.2 plot of validation costs of all techniques

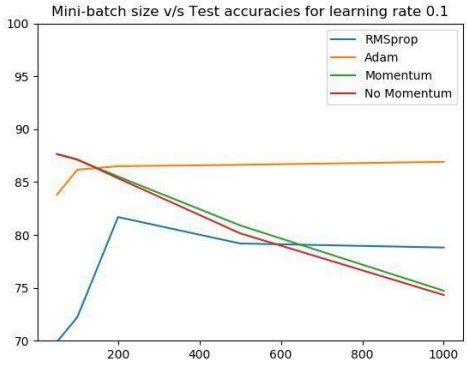

Fig. 8: Plot of test accuracies of all the techniques for various mini-batch sizes

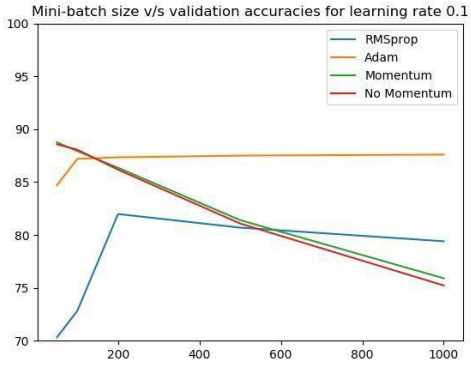

Fig. 9: Plot of validation accuracies of all the techniques for various mini-batch size

V. Results

In this section we report the train, validation and test accuracies for different experiments. Tables I and II indicate the

accuracy values for different techniques when mini-batch size is varied while keeping the learning rate constant. Table III, IV and V indicate accuracy values for different methods while keeping the mini batch values constant (500). From the tables below, we observe that, when the learning rate is low such as 0.01, Adam and RMSProp show high accuracies, whereas, No Momentum and Momentum techniques show very low accuracies. When the value of the learning rate is high, such as 0.1 and 0.2, we observe that the accuracies for Momentum and No Momentum techniques drastically improve. While Adam technique shows optimal accuracies, we observe a decline in accuracies of RMSProp for higher learning rates. We fix the learning rate to be 0.1 for all cases and vary the mini-batch size from 50 to 1000. We observe that the accuracy of RMSProp increases with the increase in size of mini-batch (from 50 to 500). It performs best at mini-batch value of 500 and its accuracy reduces when the mini-batch size is further increased. For Adam, the accuracy increases with the increase in size of mini-batch from 50 to 500. It is observed to be optimal at mini-batch value of 500 and it starts reducing when the mini-batch size is increased to 1000. Considering the momentum and no momentum techniques,the accuracies are very high at smaller mini batch sizes and they gradually decrease when the number of mini-batch size is increased.

Tables I and II depict the validation and training accuracies for different mini batch sizes varying from 50 to 500.

TABLE I: Testing Accuracies for different mini-batch sizes

Method	50	100	200	500	1000
No Momentum	88.570%	88.060%	86.170%	81.080%	75.220%
Momentum	88.750%	87.930%	86.350%	81.460%	75.940%
RMSProp	70.320%	72.820%	81.980%	80.690%	79.400%
Adam	84.700%	87.190%	87.340%	87.600%	87.610%
Method	50	100	200	500	1000
No Momentum	87.460%	87.140%	85.360%	80.150%	74.320%
Momentum	87.660%	87.110%	85.520%	80.940%	74.720%
RMSProp	69.890%	72.220%	81.690%	79.190%	78.810%
Adam	83.870%	86.170%	86.500%	86.630%	86.910%

TABLE II: Validation Accuracies for different mini-batch sizes

Tables III,IV and V depict the training,testing and validation accuracies for all the techniques with learning rates 0.01, 0.1 and 0.2 respectively.

TABLE III: Learning Rate - 0.01

Accuracy	No Momentum	Momentum	RMSProp	Adam
Training	37.750%	25.234%	85.916%	82.358%
Testing	37.610%	25.040%	83.890%	80.940%
Validation	38.200%	24.570%	84.810%	81.660%

TABLE IV:LearningRate=0.1

Accuracy	No Momentum	Momentum	RMSProp	Adam
Training	81.464%	82.462%	80.898%	89.584%
Testing	80.150%	80.940%	79.190%	86.630%
Validation	81.080%	81.460%	80.690%	87.600%

TABLE V: Learning Rate - 0.2

Accuracy	No Momentum	Momentum	RMSProp	Adam
Training	85.406%	86.192%	71.414%	89.608%
Testing	83.610%	84.640%	70.880%	86.610%
Validation	85.020%	85.530%	71.010%	87.400%

VI. Conclusion

From our experiments we conclude that Adam optimizer is the best optimization technique with learning rate equal to 0.2. RMSProp also performs almost as well as Adam in terms of speed of convergence and accuracy. No Momentum and Momentum techniques show similar behavior except that Momentum method converges faster than No-Momentum method. Also, low mini-batch size results in better model performance, whereas, a larger mini-batch size implodes the ability of the model on new data.

VII. Division of Work

We began with building the three-layer fully connected neural network. This was the base line for our project where each one of us worked together. Further, we divided the implementation of the four gradient descent algorithms between us, where, Divya Bhuvanapalli implemented the mini-batch gradient descent, along with No-Momentum technique. Aishwarya Manikyam implemented the Polyaks Classical Momentum technique, Sakshitha channadi and Bhavya Induri implemented the Adam algorithm and Bhavya Induri implemented the RMSprop technique.

VIII. Self-peer evaluation table

Aishwarya Manikyam	Divya Bhuvanapalli	Sakshitha channadi	Bhavya Induri
20	20	20	20

References

[1] S. Ghadimi and G. Lan, "Stochastic first-and zeroth-order methods for Nonconvex stochastic programming," *SIAM Journal on Optimization 23 23412368*, 2013.

[2] T.Yang, Q. Lin and Z. Li, "Unified convergence analysis of stochastic momentum methods for convex and non-convex optimization," *arXiv preprint arXiv:1604.03257*, 2016.

[3] S. J. Reddi, A. Hefny, S. Sra, B. Poczos and A. Smola, "Unified convergence analysis of stochastic momentum methods for convex and non-convex optimization," *314323*, 2016.

[4] Z. Allen-Zhu and E. Hazan, "Variance reduction for faster non-convex optimization," *International Conference on Machine Learning*, 2016.

[5] Z. Allen-Zhu, ”How to make the gradients small stochastically,” *arXiv preprint arXiv:1801.02982*, 2018.

[6] A. S. Walia, ”Types of Optimization Algorithms used in Neural Networks,” 2017. [Online]. Available: https://towardsdatascience.com/typesof-optimization-algorithms-used-in-neural-networks-and-ways-tooptimize-gradient-95ae5d39529f

[7] S. Ruder, ”An overview of gradient descent optimization algorithms” *CoRR*, 2016.

[8] Loizou, Nicolas and P. Richtrik, ”Momentum and Stochastic Momentum for Stochastic Gradient, Newton, Proximal Point and Subspace Descent Methods,” *CoRR*, 2017.

[9] Y. Bengio, N. Boulanger-Lewandowski and R. Pascanu, ”Advances in optimizing recurrent networks,” *IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC*, 2017, pp. 8624-8628.

[10] R. Gandhi, ”A look at Gradient Descent RMSProp Optimizers,” 2018(accessed October 20, 2018). [Online]. Available: https://towardsdatascience.com/a-look-at-gradient-descent-and-rmspropoptimizers-f77d483ef08b

[11] D. P. Kingma and J. Ba, ”Adam: A method for stochastic optimization,” *arXiv preprintarXiv:1412.6980*, 2014.

[12] Y. E. Nesterov, ”A method for solving the convex programming problem with convergence rate o(1/k 2),” *in Dokl. Akad. Nauk SSSR*, vol. 269, pp. 543547, 1983.

[13] H. Xiao, K. Rasul, and R. Vollgraf, (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[14] J. Duchi, E. Hazan, and Y. Singer, ”Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,” *Journal of Machine Learning Research*, 12:21212159, 2011.

[15] M. D. Zeiler, ”An Adaptive Learning Rate Method,” *arXiv preprint*, arXiv: 1212.5701, 2012.

[16] B. Vitaly. Stochastic Gradient Descent with Momentum Towards Data Science. Towards Data Science, 4 Dec. 2017, [Online]. Available: towardsdatascience.com/stochastic-gradient-descent-with-momentuma84097641a5d.