# Analyzing Gradient Descent Optimization

Akbar Jamal Abbas, Jerold Jacob Thomas, Mohanraam Sethuraman

***Abstract*—The way Neural Networks learn usually involves a huge amount of computation as they are a class of many machine learning algorithms. Due to this complexity, the machine performance is slowed down or the results are not accurate enough. In order to improve efficiency in code and reduce the error rate, there are various types of optimization techniques present and one of the most important is the Gradient Descent. Our main goal is to explore and implement existing gradient descent optimization techniques and compare the performance metrics like speed, accuracy, and stability of each technique over a 3-layer fully-connected neural network. For this analysis, we use the fashion-MNIST dataset.**

***Index terms*—** Gradient Descent Optimization, No Momentum, Polyak's Classical Momentum, Nesterov's Accelerated Gradient, RmsProp, Adam, performance comparison

## I. Introduction

GRADIENT descent is an optimization algorithm which is used to train the machine learning model to minimize the cost function. A gradient measures how much the output of a function changes with respect to the input parameters.

$$x := x - \epsilon \nabla_x f(x)$$

The equation above describes what Gradient Descent does: x describes the position. The minus sign refers to the minimization part of Gradient Descent. The $epsilon(\epsilon)$ in the middle is the learning rate and the gradient term ($\nabla_x f(x)$ ) is the direction of the steepest descent.

Gradient Descent is an optimization algorithm often used for finding the weights or coefficients of machine learning algorithms, such as artificial neural networks and logistic regression. It works by having the model make predictions on training data and using the error on the predictions to update the model in such a way as to reduce the error.

The goal of the algorithm is to find model parameters (e.g. coefficients or weights) that minimize the error of the model on the training dataset. It does this by making changes to the model that move it along a gradient or slope of errors down toward a minimum error value. This gives the algorithm its name of Gradient Descent.

Consider the loss function (J($\theta$)):

$$J(\theta) = (X\theta - Y)^\top (X\theta - Y)$$

where $\nabla J_\theta = X^\top (X\theta - Y)$

There are three main variants of Gradient Descent. They are Batch Gradient Descent, Stochastic Gradient Descent, and Mini Batch Gradient Descent. We choose any one among these three depending upon the size of the dataset.

### A. Batch Gradient Descent

Batch Gradient Descent is a variation of the Gradient Descent algorithm that calculates the error for each example in the training dataset, but only updates the model after all training examples have been evaluated.

One cycle through the entire training dataset is called a training epoch. Therefore, it is often said that batch Gradient Descent performs model updates at the end of each training epoch.

$$\theta_{t+1} := \theta_t - \alpha X^\top (X\theta_t - Y)$$

*1) Pros:*

- Fewer updates to the model mean this variant of Gradient Descent is more computationally efficient than stochastic Gradient Descent.
- The decreased update frequency results in a more stable error gradient and may result in a more stable convergence on some problems.
- The separation of the calculation of prediction errors and the model update lends the algorithm to parallel processing based implementations.

*2) Cons:*

- The more stable error gradient may result in premature convergence of the model to a less optimal set of parameters.
- The updates at the end of the training epoch require the additional complexity of accumulating prediction errors across all training examples.
- Commonly, batch Gradient Descent is implemented in such a way that it requires the entire training dataset in memory and available to the algorithm.
- Model updates, and in turn training speed, may become very slow for large datasets.

### B. Stochastic Gradient Descent

Stochastic Gradient Descent, often abbreviated SGD, is a variation of the Gradient Descent algorithm that calculates the error and updates the model for each example in the training dataset.

The update of the model for each training example means that stochastic Gradient Descent is often called an online machine learning algorithm.

$$\theta_{t+1} := \theta_t - \alpha x^{(i)} ({x^{(i)}}^\top \theta_t - y^{(i)})$$

*1) Pros:*

- The frequent updates immediately give an insight into the performance of the model and the rate of improvement.
- This variant of Gradient Descent may be the simplest to understand and implement, especially for beginners.

- The increased model update frequency can result in faster learning on some problems.
- The noisy update process can allow the model to avoid local minima (e.g. premature convergence).

### C. Mini Batch Gradient Descent

Mini-batch Gradient Descent is a variation of the Gradient Descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients.

Implementations may choose to sum the gradient over the mini-batch or take the average of the gradient which further reduces the variance of the gradient.

Mini-batch Gradient Descent seeks to find a balance between the robustness of Stochastic Gradient Descent and the efficiency of batch Gradient Descent. It is the most common implementation of Gradient Descent used in the field of deep learning.

$$\theta_{t+1} := \theta_t - \alpha X_k^\top (X_k \theta_t - Y_k)$$

*1) Pros:*

- The model update frequency is higher than batch Gradient Descent which allows for a more robust convergence, avoiding local minima.
- The batched updates provide a computationally more efficient process than stochastic Gradient Descent.
- The batching allows both the efficiency of not having all training data in memory and algorithm implementations.

*2) Cons:*

- Mini-batch requires the configuration of an additional mini-batch size hyperparameter for the learning algorithm.
- Error information must be accumulated across mini-batches of training examples like batch Gradient Descent.

## II. Related Work

Gradient Descent has been a common method for optimization of neural networks for a long time. Even though in the recent times, many new Deep learning libraries use different algorithms to perform Gradient Descent optimizations but the way these algorithms work are hidden thus making it difficult to evaluate the algorithms for different parameters. To overcome this we implemented the neural network and the different Gradient Descent methods from scratch. Before we go further into the implementation work, we will explore what other related works[8] have been done related to our work.

Finding local minima can be performed by combining Gradient Descent with the linear search but the cost associated with it can be very high. Using Newton's method can be a better method to do this but for datasets with higher dimensions, it can be difficult to implement. Another method through the convergence at local minima can be achieved faster through Momentum method[2]. This method was improvised by Nesterov Accelerateed Gradient Descent[3] where the future position was predetermined before applying momentum. Next, Adagrad[4] is an algorithm that modifies the learning rate for every parameter, thus making it a very efficient Gradient Descent algorithm. However, it has its own disadvantage as it aggregates squared gradient in the denominator of its equation. Adagrad was further improved by the introduction of the algorithm called Adadelta[5]. Through this algorithm, the limitation of Adagrad can be overcome by defining the aggregate of squared gradients as decaying average of previously squared gradients. There are many other Gradient Descent optimization algorithms but discussion of all such algorithm is beyond the scope of this paper. In the following section, the optimization algorithms used will be discussed in depth.

## III. Method

### A. Regular Network with No Momentum

For a regular neural network with no momentum, we are using simple Gradient Descent using mini batch with no momentum factors. This is just a regular neural network without taking into consideration any momentum. Considering no momentum takes away any force or bias involved in the neural network. Its simple to evaluate but it is not efficient as it ignores any force acting on the network. The equation to calculate with no moment is given below,

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

### B. Polyak's Classical Momentum

In Machine Learning applications, we use Gradient Descent to navigate the loss surface. The problem with vanilla Gradient Descent is that the convergence time is long for certain loss surface, and one of the tricks is to use momentum to reach faster convergence. The momentum method can accumulate velocity in the direction where the gradient is pointing towards the same direction across iterations. It achieves this by adding a portion of the previous weight update to the current one. We first show the math of momentum and then run some experiments to see if momentum can really help us to attain faster convergence. The following equation is the Gradient Descent with momentum update.  is the portion of the previous weight update you want to add to the current one ranges from [0, 1]. When  = 0, it reduces to vanilla Gradient Descent.

$$V = \beta V + \nabla L(w)$$

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

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

### C. Nesterov Accelerated Gradient Descent

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

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

### D. RMSProp

Root Mean Square prop[6] (RMSProp) is an optimizer which uses the recent gradients to normalize the current gradient. This is a mini batch version of RProp. For each weight it keeps the average of the root mean squared

gradients. RMSProp is given by,

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla J(\theta_t)$$

where

$$G_t = \beta G_{t-1} + (1 - \beta) \nabla J(\theta_t)^2$$

The RMSProp optimizer is similar to the Gradient Descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster.

The difference between RMSprop and Gradient Descent is on how the gradients are calculated. The following equations show how the gradients are calculated for the RMSprop and Gradient Descent with momentum. The value of momentum is denoted by beta and is usually set to 0.9

### *E. ADAM*

Adaptive Moment Estimation[7] (ADAM) is an algorithm for efficient stochastic optimization that only requires first-order gradients with low memory requirements. This method computes adaptive learning rates for every different parameter individually, from the first and second gradient moment estimates. ADAM is well suited for problems that have a large amount of data or parameters and also for problems with sparse gradients. The ADAM update rule is given by,

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \epsilon}} V_{t+1}$$

where

$$V_{t+1} = \beta_1 V_t + (1 - \beta_1) \nabla J(\theta_t)$$

$$G_{t,ii} = \beta_2 G_{t-1,ii} + (1 - \beta_2) \nabla J(\theta_{t,i})^2$$

ADAM is an optimization algorithm that can be used instead of the classical stochastic Gradient Descent procedure to update network weights iterative based on training data. Stochastic Gradient Descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training.

A learning rate is maintained for each network weight (parameter) and separately adopted as learning unfolds. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, ADAM also makes use of the average of the second moments of the gradients (the uncentered variance).

Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters $\beta_1$ and $\beta_2$ control the decay rates of these moving averages.

## IV. Experiments

Evaluation of Gradient Descent can be done using two methods. They are

- By keeping the size of the Batch as constant while varying the learning rate of the neural network.
- By keeping the learning rate of the neural network as constant while varying the size of the batch

### *A. Varying the Learning rate*

**Learning Rate = 0.0001**

| Accuracy | No Momentum | Polyak's | RMSProp | ADAM |
|---|---|---|---|---|
| Train | 49.88 | 63.96 | 92.08 | 89.12 |
| Test | 50.20 | 63.50 | 85.90 | 85.50 |
| Validation | 47.70 | 62.10 | 84.00 | 84.60 |

The above table is computed by keeping the batch size as constant (i.e) 10 items in the dataset, while the learning rate is 0.001

**Learning Rate = 0.001**

| Accuracy | No Momentum | Polyak's | RMSProp | ADAM |
|---|---|---|---|---|
| Train | 74.86 | 81.36 | 100 | 99.08 |
| Test | 74.50 | 80.10 | 86.70 | 84.80 |
| Validation | 74.10 | 79.20 | 84.20 | 84.50 |

The above table is computed by keeping the batch size as constant (i.e) 10 items in the dataset, while the learning rate is 0.01

**Learning Rate = 0.003**

| Accuracy | No Momentum | Polyak's | RMSProp | ADAM |
|---|---|---|---|---|
| Train | 81.24 | 86.40 | 96.52 | 100 |
| Test | 80.70 | 83.80 | 80.60 | 85.50 |
| Validation | 79.60 | 81.70 | 81.10 | 83.20 |

The above table is computed by keeping the batch size as constant (i.e) 10 items in the dataset, while the learning rate is 0.003

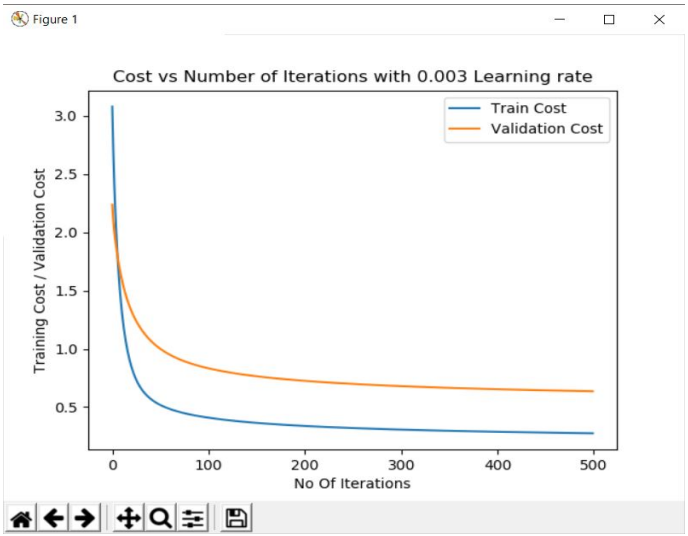


Fig. 1. The best accuracy of the neural network with no momentum in the gradient descent, when the learning rate is 0.003

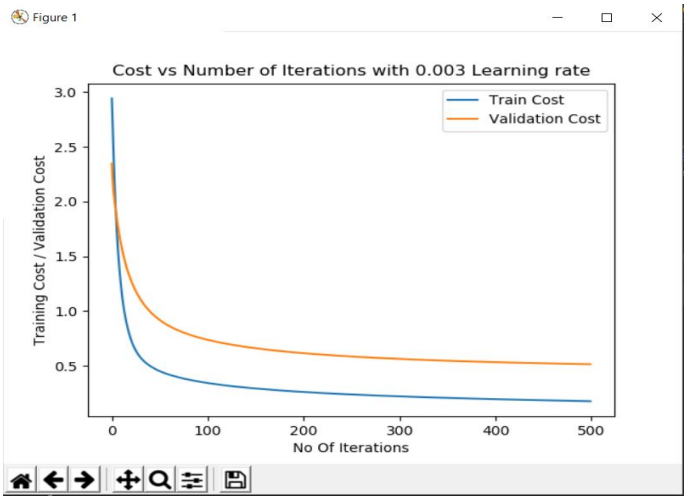


Fig. 2. The best accuracy of the neural network with momentum in the gradient descent, when the learning rate is 0.003

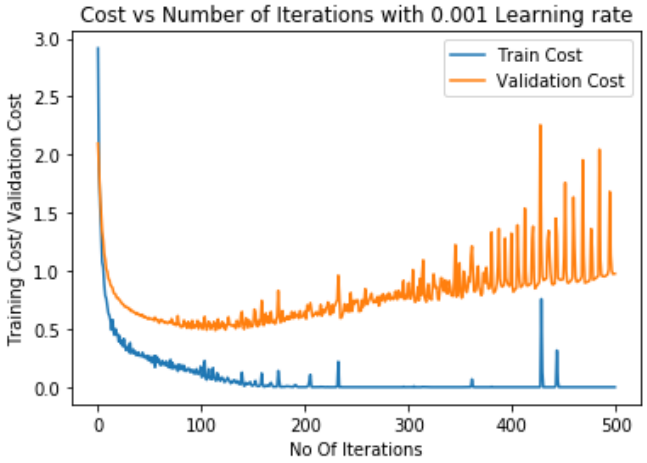


Fig. 3. The best accuracy of the neural network using gradient descent optimization technique - RMSProp, when the learning rate is 0.001

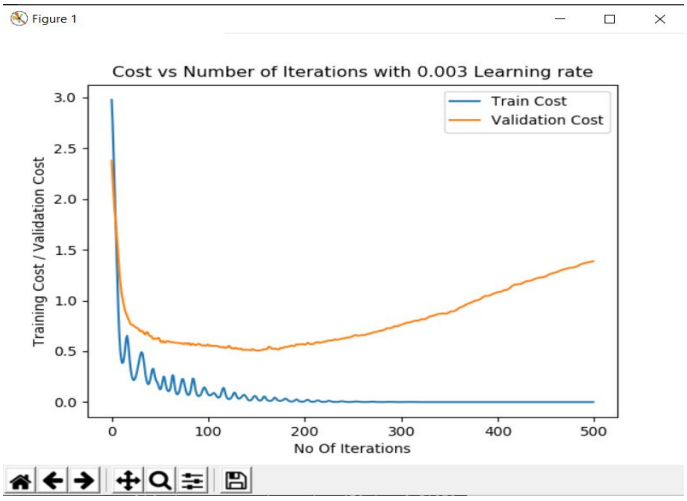


Fig. 4. The best accuracy of the neural network using gradient descent optimization technique - ADAM, when the learning rate is 0.003

## B. *Varying the size of the Batch*

**Learning Rate = 0.0001 Batch Size = 5**

| Accuracy | No Momentum | Polyak's | RMSProp | ADAM |
|---|---|---|---|---|
| Train | 35.62 | 54.00 | 91.26 | 85.20 |
| Test | 35.10 | 54.80 | 85.90 | 83.40 |
| Validation | 34.70 | 52.70 | 83.90 | 81.40 |

The above table is computed by keeping the learning rate as constant (i.e) 0.0001 , while the batch size is 5 items in the dataset.

**Learning Rate = 0.0001 Batch Size = 10**

| Accuracy | No Momentum | Polyak's | RMSProp | ADAM |
|---|---|---|---|---|
| Train | 49.88 | 63.96 | 92.08 | 89.12 |
| Test | 50.20 | 63.50 | 85.90 | 85.50 |
| Validation | 47.70 | 62.10 | 84.00 | 84.60 |

The above table is computed by keeping the learning rate as constant (i.e) 0.0001 , while the batch size is 10 items in the dataset.

**Learning Rate = 0.0001 Batch Size = 20**

| Accuracy | No Momentum | Polyak's | RMSProp | ADAM |
|---|---|---|---|---|
| Train | 59.74 | 69.56 | 88.68 | 94.88 |
| Test | 59.70 | 68.20 | 79.60 | 80.80 |
| Validation | 57.50 | 69.20 | 80.80 | 85.00 |

The above table is computed by keeping the learning rate as constant (i.e) 0.0001 , while the batch size is 20 items in the dataset.

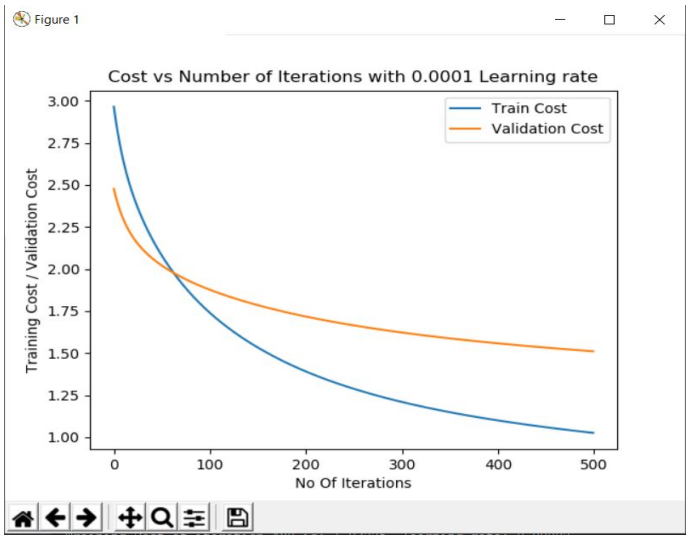


Fig. 5. The best accuracy of the neural network with no momentum in the gradient descent, when the learning rate is 0.0001 and batch size is 20.

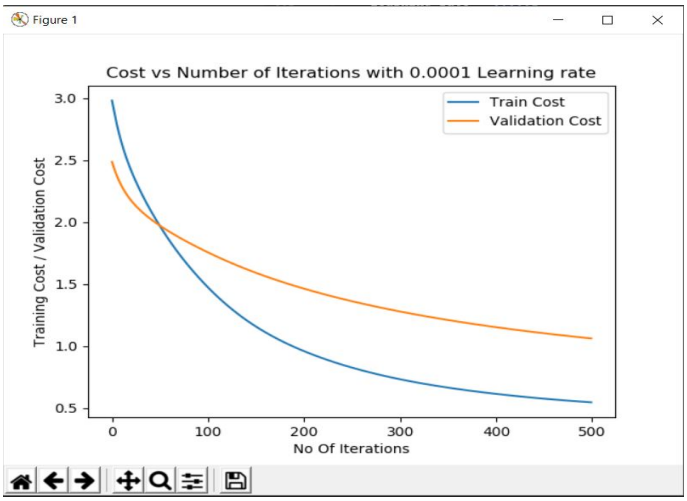


Fig. 6. The best accuracy of the neural network with momentum in the gradient descent, when the learning rate is 0.0001 and batch size is 20.

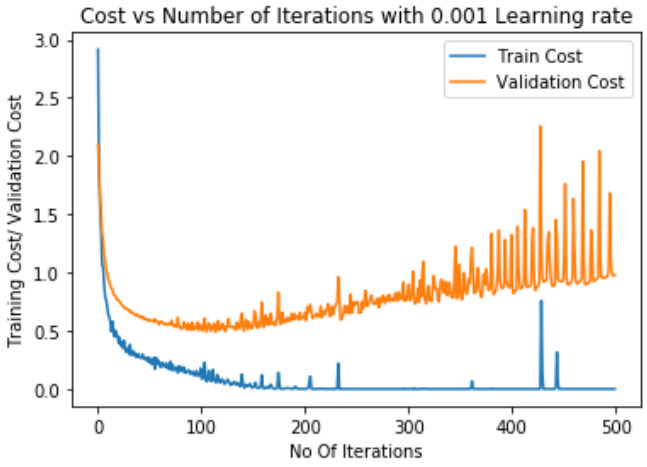


Fig. 7. The best accuracy of the neural network using gradient descent optimization technique - ADAM, when the learning rate is 0.0001 and the batch size is 10.

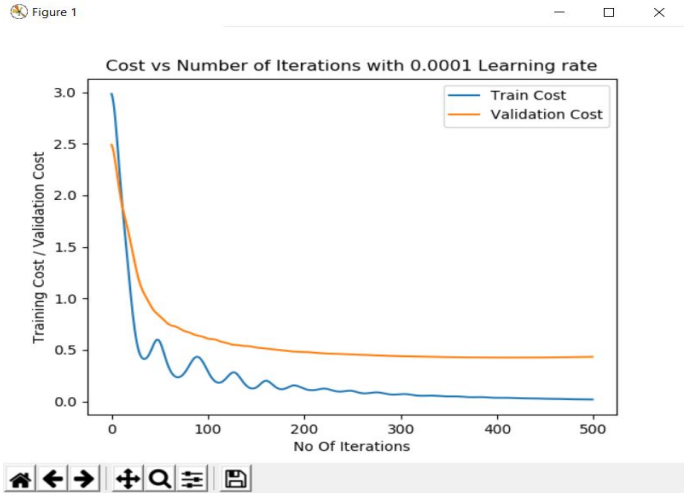


Fig. 8. The best accuracy of the neural network using gradient descent optimization technique - ADAM, when the learning rate is 0.0001 and the batch size is 20.

## V. Conclusions

From observation of graphs and tables, where we first varied the learning rate without changing other parameters and next we kept learning rate constant and varied batch sizes, we can conlude both ADAM and RMSProp performs the best but ADAM performs slightly better than RMSProp.

## VI. Division of Work

All members of the team worked together and contributed equally with code implementation

Akbar Jamal Abbas: Implemented no momentum and momentum. Helped with writing report

Myself: Implemented ADAM and worked in evaluation of different gradient descent optimization algorithms.

Mohanraam Sethuraman: Implemented RMSProp and worked in evaluation of different Gradient Descent optimization algortihms.

## VII. Self-Peer Evaluation Table

| Akbar Jamal Abbas | Myself | Mohanraam Sethuraman |
|---|---|---|
| 20 | 20 | 20 |

## References

[1] Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms by Han Xiao, Kashif Rasul, Roland Vollgraf.

[2] Polyaks classical momentum: B. T. Polyak, Some methods of speeding up the convergence of iteration methods, USSR Computational Mathematics and Mathematical Physics, vol. 4, no. 5, pp. 117, 1964.

[3] Nesterovs Accelerated Gradient: 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, 1983, pp. 543547.

[4] Duchi, J., Hazan, E., Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 21212159.

[5] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method.

[6] RmsProp: T. Tieleman and G. Hinton, Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural networks for machine learning, vol. 4, no. 2, pp. 2631, 2012.

[7] ADAM: "Adam: A Method for Stochastic Optimization" by Diederik P. Kingma, Jimmy Ba.

[8] Additional resources: An overview of Gradient Descent optimization algorithms by S. Ruder