New York Institute of Technology

Stochastic Methods in Optimization

# Comparative analysis of Stochastic Optimization methods with binary classification using a CNN

DTSC 610/ Fall 2021 - M01

Course Instructor: Dr. Jerry Cheng

Team:

| | | |
|---|---|---|
| Michael Trzaskoma | 1202901 | mtrzasko@nyit.edu |
| Hui (Henry) Chen | 1242445 | hchen60@nyit.edu |
| Bofan He | 0870241 | bhe@nyit.edu |
| Ephraim Hallford | 1303929 | ehalf01@nyit.edu |

# Table of Content

**Abstract**

Optimization methods are used within neural networks to train accurate models. We investigated five optimizers: Adagrad, Adadelta, RMSprop, Adam and Adamax and their relative performance in training a convolutional neural network with the Cats and Dog dataset. After comparative analysis, it was determined that Adam and Adamax exhibited similar training behavior.

## I. Introduction

Optimization methodologies have been implemented in a wide array of machine learning techniques in order to improve performance, increase speed, and reduce memory computational expense [1]. Among the most popular techniques and easily implemented is Gradient Descent [2]. In this paper, we will describe the variants of Gradient Descent and their application in the optimization of Convolutional Neural Networks (CNNs). Each section will discuss in depth the mathematical and conceptual underpinning for these algorithms, and how they can be used in the context of deep learning. We will discuss Adagrad, Adadelta, RMSprop, Adam and AdaMax and expound what potential research can be potentially done in the future to improve these algorithms. We will then discuss how it was applied towards a CNN for the Kaggle Cats and Dogs Dataset and how each optimizer performed.

Gradient descent is concerned with minimizing an objective function J($\theta$). It uses a constant $\eta$, which is the learning rate by modifying a step size based on a gradient w.r.t the parameters [2]. Gradient descent is given by

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Where $\theta$ are the steps that are modified by some increment given by the partial derivatives w.r.t to the objective function [3]. The term alpha is the learning rate, and it is the crucial aspect of what is being modified with the variants we will discuss in detail. For this reason, gradient descent's gradient is dependent upon the entire dataset and therefore it takes time and computational effort to calculate the gradient w.r.t to the whole dataset. Batch gradient descent segments these into batches and calculates the gradient for each batch [4]. Stochastic gradient descent, which serves as the foundation for the following algorithms in this paper, extends gradient descent towards computing the gradient on a batch of a smaller subset of the training data set. Hardt et. al demonstrated that stochastic gradient descent is a stable method for taking randomized iterative steps while converging on a minima [5]. For this reason, we wished to expound upon the differences between emergent variants, and evaluate their performance relative to a CNN model.

## II. Adagrad

Adagrad was first presented by Duchi et al. and seeks to extend stochastic descent methods deeper by taking the regularized sum matrix of all past gradients in order to calculate the current time step [6]. Adagrad has two main versions: diagonal and full matrix. Diagonal matrix updates on learning rate per dimensionality of n matrix whereas full AdaGrad computes learning rates for the entire matrix. AdaGrad is preferred to sparse training sets such as NLP since it performs small updates [7]. The main idea of Adagrad is to propose a different learning rate η for each and every neuron, along with hidden layers, based on different iterations. Hence, practically the default learning rate is 0.001. The aim of that is to overcome the different features that appear on the dataset such as sense and spares. For instance, a dataset may have non-zero or binary observations. The formula for Adagrad is given below:

$$\omega_t = \omega_{t-1} - \eta_t^i \frac{dL}{d\omega_{t-1}}$$

$$\eta_t^i = \frac{\eta}{\sqrt{\alpha_t} + \epsilon}$$

$$\alpha_t = \sum_{i=1}^{t} \left( \frac{dL}{d\omega_i} \right)^2$$

The $\mathcal{W}$ stands for weight, ***t*** for the current iteration, $\eta^i_t$ for the learning rate, and *L* for lost function. Adagrad subtracts the previous weight with a learning rate with a derivative of the loss function with respect to the derivative of previous weights. The learning rate $\eta^i_t$ in Adagrad is crucial where it sums up all of the iterations where it takes a constant learning rate divided by $\alpha_t$ + ε. The epsilon is a small constant integer value to prevent the zero on the denominator. If the denominator of the learning rate becomes zero, it defeats the purpose of Adagrad. The alpha $\alpha_t$ is further derived into where it performs the summation of all iterations of the loss function with respect to the derivative of current iteration weights and then squares it. The $\eta^i_t$ will initially be a big value and as the iteration goes on, it gradually decreases. Because $\eta^i_t$ decreases, the $\mathcal{W}$ gradually decreases on updating.

## III. Adadelta

Mathew Zeiler [10] attempts to tackle one of the largest issues that Adagrad has which is to allow recent gradients to still be relevant, and the methodology he takes in order to achieve this is through utilizing a running mean, with a decaying rate ρ = (0.9~0.99). The calculation of the change in parameters also undergoes a running mean as well.

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)g_t^2$$
$$E[\Delta x^2]_t = \rho E[\Delta x^2]_{t-1} + (1-\rho)\Delta x_t^2$$

When calculating the root of the running mean squared, a really small constant ε = (0.1 ~0.00001) is utilized to prevent zeros from occurring to allow for the model to initiate learning and to learn endlessly, even if it's at a small rate.

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon}$$
$$RMS[\Delta x]_t = \sqrt{E[\Delta x^2]_t + \epsilon}$$
$$\Delta x_t = -\frac{RMS[\Delta x]_{t-1}}{RMS[g]_t} g_t$$

The last key aspect of Adadelta is that the initial conditions (ρ, ε) should not greatly affect the performance of the model which will be highlighted in the results section due to the numerator and denominator of the equation to be experiencing the impact of the parameter.

## IV. RMSProp

RMSprop was first presented by Geoff Hinton in the Coursera course "Neural Networks for deep learning" [11]. RMSprop resolves Adadelta's running mean by taking an exponentially decaying average gamma or beta. Adagrad suffers from rapidly diminishing learning rates due to taking the running sum of the previous gradients[12]. RMSprop strikes a balance by taking the exponentially decay running square gradients of the previous timestep, and thereby overcomes the issue Adagrad presented where it divides the moving gradient by the square root of this mean. It is given by where g is the running mean. In this case, RMSprop attenuates the running mean by the root mean squared so the most recent steps are more impactful[12]. This mathematical methodology allows the algorithm to focus more closely on the most recent learning rate, and chronologically adaptively reduce the importance of the gradient's learning rate/step size for iterations earlier in the process[13]. The update rule for RMSprop is given as:

$$E[g^2](t) = \beta E[g^2](t-1) + (1-\beta)\left(\frac{\partial c}{\partial w}\right)^2$$

$$w_{ij}(t) = w_{ij}(t-1) - \frac{\eta}{\sqrt{E[g^2]}} \frac{\partial c}{\partial w_{ij}}$$

Especially for a large amount of data search. Compared to Gradient Descent, the RMSProp algorithm approaches optima much faster and accurately.

## V. Adam

Kingma et al. presented Adam as a stochastic adaptive learning algorithm in which RMSprop and Stochastic Gradient Descent with Momentum were combined. Adam consists of biased corrected terms that aim to minimize the effect of noise in calculation by correcting the running mean by constant for time t[10]. Adam's update rule is given as where we modify m and v for each step incrementally.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

The terms v and m are bias corrected to attenuate noise as seen earlier:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Where m and v given by

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

The purpose of m is to modify the time step at t by taking the previous running mean at time t-1 modified by the running mean of gradients g, and the second moment, given by v, which is a metric for the uncentered variance. In a sense, Adam seeks to use the exponential decaying mean and modulates this by the variance of previous gradients to update each step [12].

## VI. Adamax

Adamax is a variant of Adam in that it takes the infinity norm. It replaces the root squared mean with the infinity norm in the update step [13]. In the case of Adamax, the second moment is modified as:

$$\begin{aligned} v_t &= \beta_2^p v_{t-1} + (1 - \beta_2^p)|g_t|^p \\ &= (1 - \beta_2^p) \sum_{i=1}^{t} \beta_2^{p(t-i)} \cdot |g_i|^p \end{aligned}$$

Where p is infinity. In this case, it only takes the maximum gradients for variance from the n matrix to converge on a minima. In this case, it is generally stable as it approaches as demonstrated by Kingman in their paper.

## VII. CNN

Neural networks are used within machine learning to learn features and classify data [13]. A basic neural network consists of an input layer, hidden layer and output layer. Each layer consists of nodes that connect to other nodes within the network and have weights and biases that indicate the degree of connection strength between the nodes. As the net is trained, the weights are updated to reflect their connection and reflect the dataset that they are trained on by specific weights. A CNN is a feed-forward network that flows in one direction from the previous layer and consists of the input layer, convolutional layers, pooling layers, and a dense layer that is fully connected to all layers in the previous layer and is transformed by a nonlinear function that provides probabilities for classes that were fed into the initial layer. Each layer in the network has a threshold action in which nonlinearity assesses the nodes and the importance of the previous layer. We train an objective function such as binary cross-entropy in order to minimize the error between the pdf of the correct input and the observed [15]. Due to the complexity of the mathematical calculations of updating these weights, we can use gradient descent to find the minima of this function over the dataset, and for this reason, we are interested in comparative analysis of optimizers to improve the performance of these models.

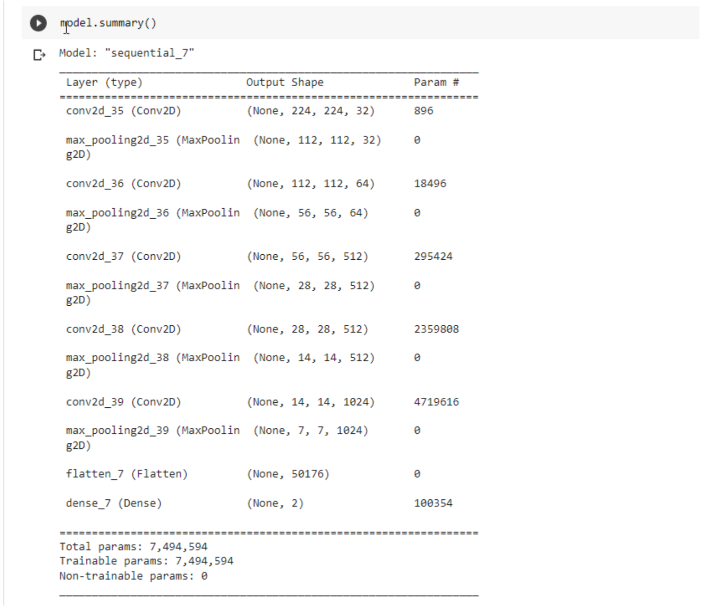
## VIII. Methods

In order to determine the parameters for CNN, we evaluated Adagrad, AdaDelta, RMSprop, Adam and AdaMax over the polynomial function given as:

$$f(x,y) = x^4 + y^2$$

Python Package Matplotlib was used to plot the iterations over time to see their performance with respect to each optimizer. We compared parameters rho for Adagrad, epsilon and rho for Adadelta, rho for RMS prop, and beta1, beta2 and alpha for AdaMax. A CNN was constructed in

Keras in Python with the following architecture as seen in Figure 1.1. All parameters at default for all optimizers with learning rates were set to .0001. The activation function was set to ReLu and fully connected dense layer activation was set to softmax.  5000 images from Kaggle Dogs and Cat Dataset were trained over the model and their performance was visually examined. The objective function was set as binary cross-entropy.



```
model.summary()

Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_35 (Conv2D)          (None, 224, 224, 32)      896

 max_pooling2d_35 (MaxPoolin  (None, 112, 112, 32)     0
 g2D)

 conv2d_36 (Conv2D)          (None, 112, 112, 64)      18496

 max_pooling2d_36 (MaxPoolin  (None, 56, 56, 64)       0
 g2D)

 conv2d_37 (Conv2D)          (None, 56, 56, 512)       295424

 max_pooling2d_37 (MaxPoolin  (None, 28, 28, 512)      0
 g2D)

 conv2d_38 (Conv2D)          (None, 28, 28, 512)       2359808

 max_pooling2d_38 (MaxPoolin  (None, 14, 14, 512)      0
 g2D)

 conv2d_39 (Conv2D)          (None, 14, 14, 1024)      4719616

 max_pooling2d_39 (MaxPoolin  (None, 7, 7, 1024)       0
 g2D)

 flatten_7 (Flatten)         (None, 50176)             0

 dense_7 (Dense)             (None, 2)                 100354

=================================================================
Total params: 7,494,594
Trainable params: 7,494,594
Non-trainable params: 0
_________________________________________________________________
```

Figure 1.1 CNN architecture for Cats and Dogs Dataset

## IX. Results

The parameters for Adagrad, Adadelta, RMSprop and AdaMax were experimentally examined. Evaluation of parameters for Adagrad determined that learning rate affects the speed at which convergence to a minimum occurs.

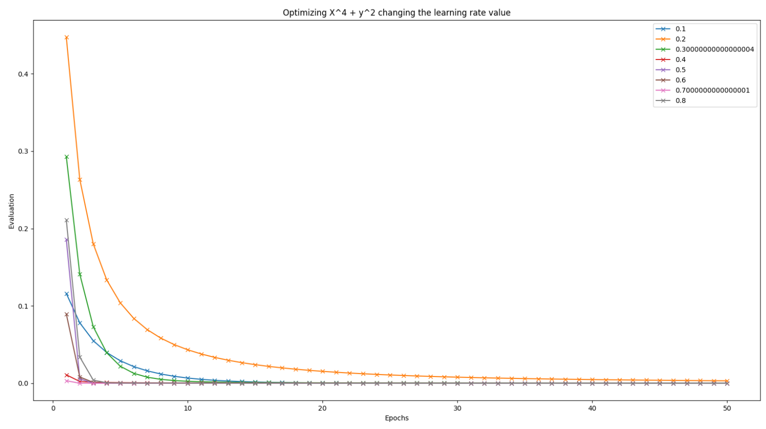


Figure 1.2 Adjusted Learning Rate for Adagrad

Adadelta's performance was evaluated for parameters epsilon and rho. The below figure indicates that performance varies depending upon modulating rho with a constant epsilon however very slightly if at all.

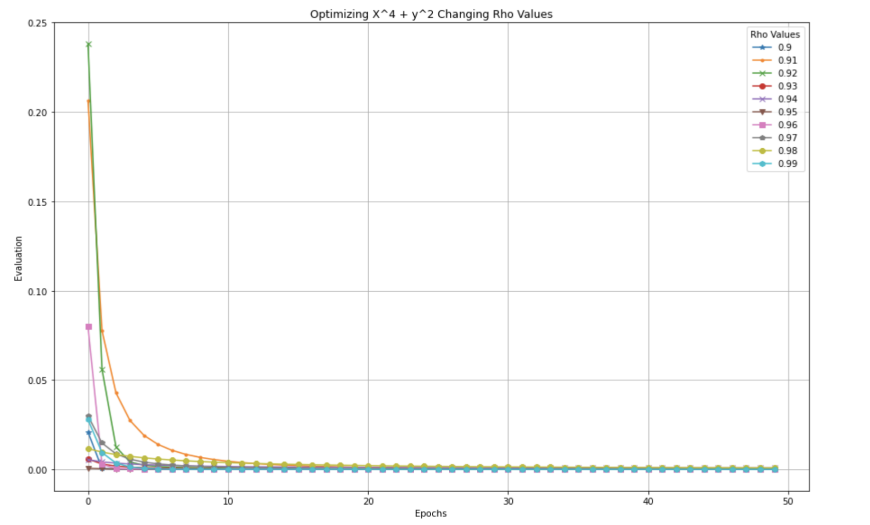


Figure 1.3 Modifying Rho values for Adadelta

In addition, Adadelta's parameter epsilon was experimentally evaluated with a constant rho and indicated slightly improved performance with smaller epsilon constants however overall lead to only slight difference.

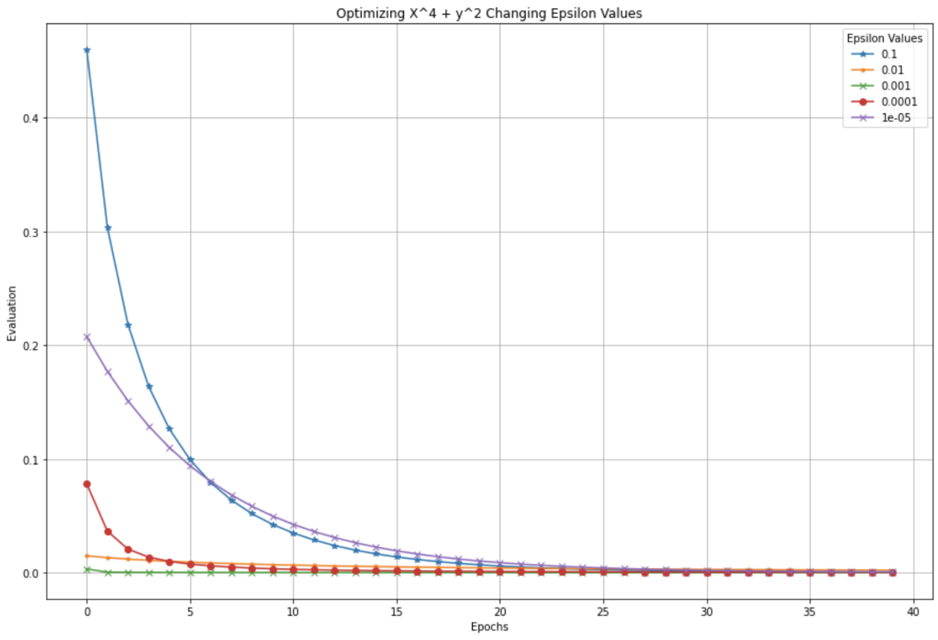


Figure 1.4 Epsilon Parameters with Adadelta

RMsprop parameter rho was evaluated and indicated improved performance by modulating Rho.

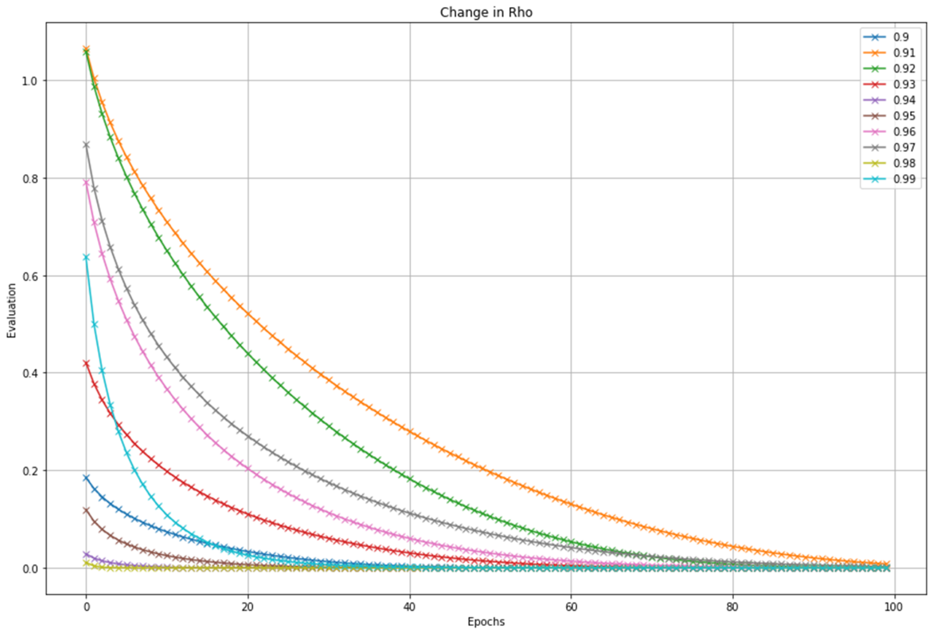
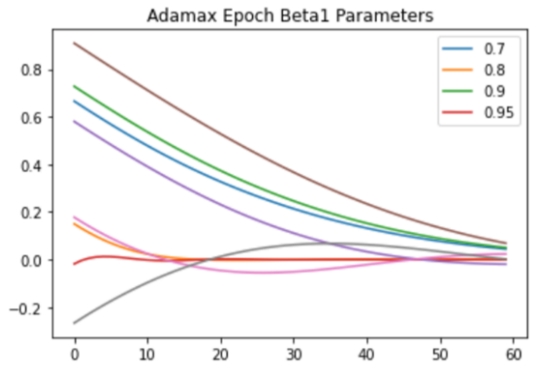


Figure 1.5 Rho Parameters for RMSprop

The parameters beta1, beta2 and alpha for AdaMax were modulated. Performance for the polynomial function indicated performance was optimal around .95-.99 and beta2 close to .99 as demonstrated by Figure 1.6, Figure 1.7 and Figure 1.8.



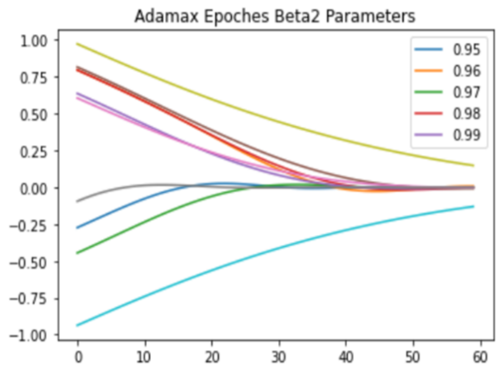


Figure 1.6 Beta1

Figure 1.7 beta2

The parameter alpha was determined to be best when larger than .02 in the case of the polynomial objective function.

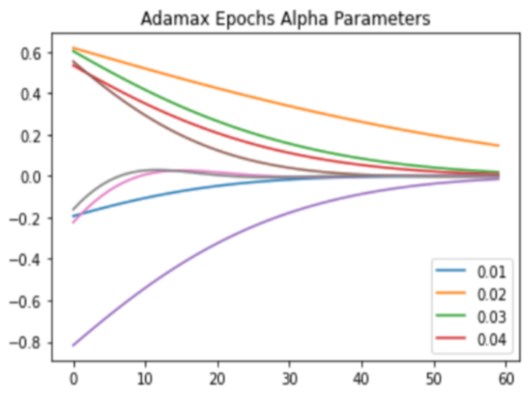


Figure 1.8 Alpha parameter

The relative performance of the CNN was experimentally evaluated and determined RMSprop, Adamax, and Adam performed best with the model. Adadelta exhibited the weakest performance and Adagrad was around the same performance as Adadelta.

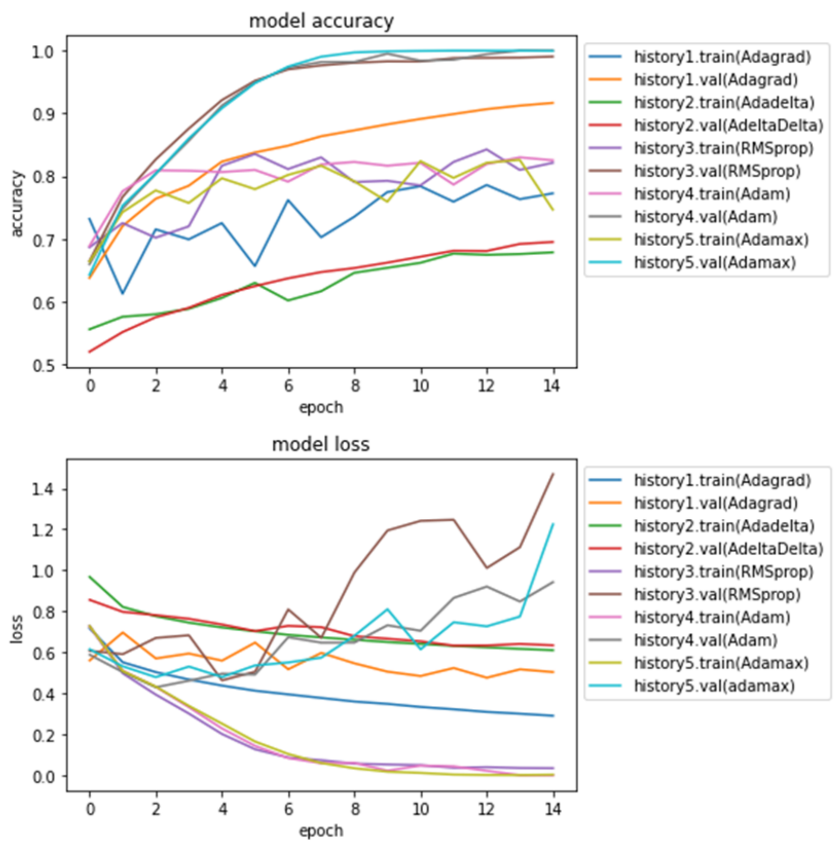


Figure 1.9 Optimizers Performance on Kaggle Data Set

## X. Discussion

Comparative analysis of optimization algorithms is an area of research within machine learning. Dogo et al. in 2018 investigated the performance of first-order stochastic gradient descent algorithms and evaluated how they performed over a convolutional neural network in which they found that NADAM, a variant of Adam, excelled against RMSprop, Adam, Adadelta, Adagrad and Adamax [16]. In a similar vein, we sought to investigate the performance of five optimizers in the performance of training a CNN. Based on our evaluation, our results indicate that Adamax, Adam and RMSProp performed better than Adagrad and Adadelta for this data set in question. Aligned with the theoretical foundation of Adagrad, in that the summation of previous gradients leads to smaller iterative steps, we can conclude that taking the exponential decay running mean is superior when dealing with dense datasets. As noted by Duchi and reinforced by Lydia, Adagrad is preferred when dealing with sparse data sets such as text processing within NLP [17]. Little is known about the theoretical foundation of how neural networks learn [18]. Future consideration can be taken into how these algorithms perform while improving the speed and accuracy of the models. We should consider the training curves. As we can note in Figure 1.9, the smoothness of Adagrad differs from Adadelta since it has slower steps sizes.

Possible improvements in our design would be to isolate the training and validation sets and render them separately. With that in mind, a larger dataset could be used in order to improve results. Future research could be done to incorporate comparisons of Nadam. We can test the algorithms over a different CNN architecture to determine if it has similar results. In conclusion, optimizers modify a learning rate as it progresses through a data set.

Acknowledgements

We would like to thank Jerry Cheng for course instruction and inspiration for our work.

## References

1. Ruder, S., 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

2. Hochreiter, S., Younger, A.S. and Conwell, P.R., 2001, August. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks* (pp. 87-94). Springer, Berlin, Heidelberg.

3. Mason, L., Baxter, J., Bartlett, P. and Frean, M., 1999, May. Boosting algorithms as gradient descent in function space. In *Proc. NIPS* (Vol. 12, pp. 512-518).

4. Dong, X. and Zhou, D.X., 2008. Learning gradients by a gradient descent algorithm. *Journal of Mathematical Analysis and Applications*, *341*(2), pp.1018-1027.

5. Hardt, M., Recht, B. and Singer, Y., 2016, June. Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning* (pp. 1225-1234). PMLR.

6. Duchi, J., Hazan, E. and Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(7).

7. Lydia, A. and Francis, S., 2019. Adagrad—an optimizer for stochastic gradient descent. *Int. J. Inf. Comput. Sci*, *6*(5).

8. Ward, R., Wu, X. and Bottou, L., 2018. Adagrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization. *arXiv preprint arXiv:1806.01811*, *2*.

9. Tieleman, T. and Hinton, G. RMSPROP: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, Lecture 6.5, 2012

10. Zeiler, M.D., 2012. Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

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

12. Balles, L. and Hennig, P., 2018, July. Dissecting adam: The sign, magnitude and variance of stochastic gradients. In International Conference on Machine Learning (pp. 404-413).

PMLR.

13. SUVRIT, S. R. A. (2015). *Optimization for machine learning*. PHI LEARNING.

14. Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.698*

15. Kasar, M.M., Bhattacharyya, D. and Kim, T.H., 2016. Face recognition using neural network: a review. International Journal of Security and Its Applications, 10(3), pp.81-100.

16. Han, S. H., Kim, K. W., Kim, S., & Youn, Y. C. (2018). Artificial Neural Network: Understanding the Basic Concepts without Mathematics. *Dementia and neurocognitive disorders*, *17*(3), 83–89. https://doi.org/10.12779/dnd.2018.17.3.83

17. Dogo, E.M., Afolabi, O.J., Nwulu, N.I., Twala, B. and Aigbavboa, C.O., 2018, December. A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks. In *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)* (pp. 92-99). IEEE.

18. Lydia, A. and Francis, S., 2019. Adagrad—an optimizer for stochastic gradient descent. *Int. J. Inf. Comput. Sci*, *6*(5).

19. Kornblith, S., Norouzi, M., Lee, H. and Hinton, G., 2019, May. Similarity of neural network representations revisited. In *International Conference on Machine Learning* (pp. 3519-3529). PMLR.

## Appendix A

Source code

Link : https://drive.google.com/drive/folders/1c3wajq-5bPOwhpaJtpS670LsHTYxl8ha?usp=sharing