International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 /doi:10.14445/22315381/IJETT-V69I7P209

Volume 69 Issue 7, 64-68, July, 2021
©2021 Seventh Sense Research Group®

The Performance of Various Optimizers in Machine Learning

Rajendra.P [#1], Pusuluri V.N.H[#2], Gunavardhana Naidu.T[*3]

[#] *Department of Mathematics, CMR Institute of Technology, Bengaluru, Karnataka, India.*

[*] *Department of Physics, Aditya Institute of Technology and Management, Tekkali, Srikakulam, AP, India.*

[1] rajendra.padidhapu@gmail.com, [2] hanuravi@yahoo.com, [3] tgpnaidu@gmail.com

Abstract: *The primary goal of the optimizers is to speed up the training and helps to boost the efficiency of the models. Optimization methods are the engines underlying deep neural networks that enable them to learn from data. When faced with the training of a neural network, the decision of which optimizer to select seems to be shrouded in mystery, since in the general literature around optimizers require a lot of mathematical baggage. To define a practical criterion, the authors carried out a series of experiments to see the performance of different optimizers in canonical problems of machine learning. So we can choose an optimizer easily.*

Keywords: *Optimizers, Machine Learning, Neural Network, Gradient Descent, Adaptive methods.*

I. INTRODUCTION

Deep learning is characterized by using large samples with the help of a single optimizing algorithm. Typical optimization algorithms adjust the parameters of all operations simultaneously and effectively evaluate the influence of each of the parameters of the neural network. The optimization algorithm (Optimizer) is one of the key aspects of training an artificial deep neural network. The loss function or the error function can be called the objective function of the optimization. The standard neural network [1] training objective is given by:

$$\min_{\theta} J(\theta), \text{where } J(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(p(\theta, x_i), y_i) \qquad (1)$$

Where $J(\theta)$ is the loss function for each input x_i, label y_i, and the predicted value $p(\theta, x_i)$?
The basic gradient iteration is given by

$$\theta_{k+1} = \theta_k - \alpha_k \nabla J(\theta_k) \qquad (2)$$

Where α_k is the rate of learning, and $\nabla J(\theta)$ is the standard gradient? The Gradient direction of $\nabla J(\theta)$ gives the greatest reduction in $J(\theta)$. The assumption here is that the function $\nabla J(\theta)$ is continuous and differentiable. Our goal is to investigate the lowest point of the optimization function [2]. However, the actual direction to this valley is not known. We can only observe locally, and therefore the direction of the negative gradient is the best information. Taking a small step in that direction can only bring us closer to the minimum. Once we've taken the small step, we calculate the new gradient and move a small amount in that direction again until we reach the valley. So essentially, all the gradient descent does is following the direction of the steepest descent (negative gradient). The Parameter in the iterative update equation is called step size. We generally do not know the value of the optimal step size; then, we have to try different values. Standard practice is to test a bunch of values on a logarithmic scale and then use the best one. Different scenarios can occur [3], and the image below shows these scenarios for a 1D quadratic. If the value of the learning rate is too low, we will make steady progress towards the minimum. However, this may take longer than ideal. In general, it is very difficult to obtain a step size that takes us directly to the minimum. What we would ideally want is to have a step size slightly larger than optimal. In practice, this provides the fastest convergence. However, if we use too large a learning rate, iterations are moved further and further from the minimum, and the result is divergence. In practice, we would like to use a learning rate that is slightly lower than the divergent rate [4].

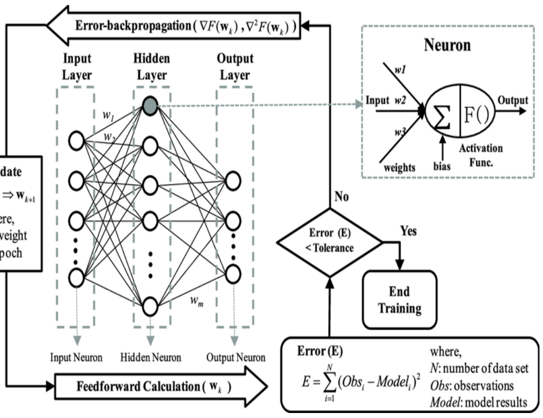


Fig 1: Backpropagation algorithm for neural network training [5].

In essence, the goal of neural network training is to minimize the cost function by finding the appropriate weights for the edges of the network. The discovery of these weights is carried out using a numerical algorithm called backpropagation that can be summarized in Fig 1.

II. OVERVIEW OF THE MAIN OPTIMIZERS

The following are the optimizers that we have trained in each network and presented the results in this article.

A. Stochastic Gradient Descent (SGD): The calculation of the partial derivative of the cost function concerning each of the weights of the network for each observation is given the number of different weights and observations. Therefore, a first optimization consists of the introduction of a stochastic (random) behavior. A. S. Nemirovsky & D. B. Yudin studied algorithms to optimize convex functions based on constraints [6]. The authors also studied the method efficiency in the minimization of convex problems. Nicolas Le Roux et al. [7] proposed a new SGD having a fast convergence rate that incorporates the previous gradient rate. Specifically for a neural network, stochastic estimation refers to the error gradient for a single data point (single instance). SGD does something as simple as limiting the derivative calculation to just one observation (per batch). There are some variations based on selecting several observations instead of one (mini-batch SGD).

B. Adaptive Gradient Algorithm (AdaGrad): In the SGD formulation, each weight in the network is updated using an equation with the same learning rate (global γ). Instead, for adaptive methods, we accept a learning rate for each weight individually. For this purpose, the information obtained from the gradients of each weight is used. Eiji Mizutani and S.E. Dreyfus [8] analyzed the Hessian matrix H exploits neural networks "layered symmetry" and making Hessian evaluation for practical problems. The AdaGrad algorithm introduces a very interesting variation in the concept of training factor instead of considering a uniform value for all weights; a specific training factor is maintained for each of them. It would be impractical to calculate this value specifically so, starting from the initial training factor, AdaGrad scales and adapts it for each dimension for the accumulated gradient in each iteration.

C. Adadelta: The Adadelta is a variation of AdaGrad in which instead of calculating the scaling of the training factor of each dimension taking into account the accumulated gradient from the beginning of the execution, it is restricted to a window of the fixed size of the last n gradients. The networks most used in practice have a different structure in different parts. For example, the first parts of a CNN [9] network can be very shallow convolutional layers on large images, and then in the network, we can have convolutions with a large number of channels on small images. The two operations are very different, so a learning rate that works well for the beginning of the network may not work well for the later sections of the network. This means that adaptive learning rates per layer can prove to be useful.

D. Root Mean Square Propagation (RMSprop): The RMSProp is a similar algorithm. It also maintains a different training factor for each dimension, but in this case, the training factor is scaled by dividing it by the mean of the exponential decline of the square of the gradients. The central idea of Root Mean Square Propagation is to normalize the gradient with its mean value of the squares of square root. Yann N. Dauphin et al. [10] proposed a new saddle-free Newton method to 2nd order optimization to escape from saddle points. The authors applied this method for DNN and RNN, and the result gives numerical information for its optimization performance. The original method maintains a non-central second-order moment exponential moving average. The second-order moment is used to normalize all elements of the gradient, which means that each element of the gradient is divided by the square root of the estimated second-order moment. If the expected value of the grand is small, this process is similar to dividing the gradient by the standard deviation.

E. Adaptive moment estimation (Adam): The Adam algorithm combines the benefits of AdaGrad and RMSProp. One training factor per parameter is maintained, and in addition to calculating RMSProp, each training factor is also affected by the mean momentum of the gradient. Deep learning allows very high computational models through numerous handling layers to learn portrayals of information with various degrees of reflection [11]. As has just been verified, the most recent algorithms such as Adam are built based on their predecessors; therefore, we can expect their performance to be superior.

III. THE PROPOSED EXPERIMENTS

It is difficult to have an intuitive vision of the behavior of each one of the optimizers; therefore, it is useful to visualize its performance in different cost functions. From a practical point of view, in a real environment, it is impossible to advance the taxonomy of the cost function. To give the study as much breadth as possible, we have selected the following four classic problems in the field of machine learning. In addition to representing different functional scopes, each one of them exposes specific and representative network architecture:

Table 1: The proposed experiments.

Type of the problem	Network architecture	Data set
Simple Regression	Multi-layer perceptron (MLP)	Boston home prices
Multiclass classifier	2 Dimensional Convolutional Neural Network (2D – CNN)	Fashion - MNIST
Binary classifier	1 Dimensional Convolutional Neural Network (1D – CNN) with embeddings	Sentiment analysis - IMDB
Forecasting of time series	Long Short - Term Memory (LSTM) of Recurrent Neural Network (RNN)	Prediction of temperature

The following optimizers are trained in each of the neural networks (all available in Keras).

(i) Adadelta (ii) Adagrad (iii) Adam
(iv) Adamax (v) Ftrl (vi) Nadam
(vii) RMSprop
(viii) SGD (without Nesterov and Momentum)

Regarding the cost functions, we used MAE for the regressions and forecast, binary cross-entropy used for the binary classifier, and categorical cross-entropy used for the multiclass classifier. We kept the default values of the learning factor and the specific parameters of each optimizer (learning rate = 0.001).

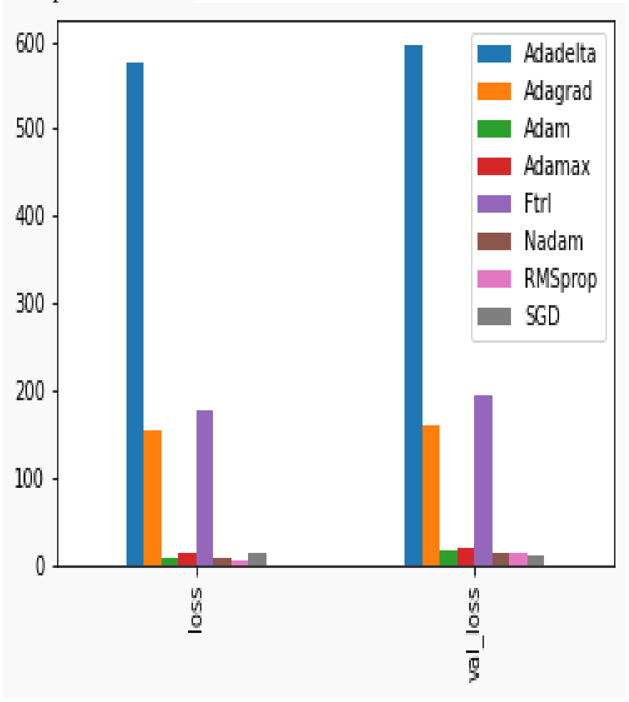
IV. THE RESULTS AND DISCUSSIONS

The following are the results of the executions of the four different experiments.

Experiment 1: We used the Boston housing price dataset [12] that contains the information on prices and features of blocks of houses in the city of Boston. The objective variable of this dataset is the average price of the houses in a block based on the characteristics of the said residential area. In general, we observed three types of behaviors in Simple Numerical Regression.

(i) Adadelta does not converge,
(ii) Adagrad and Ftrl converge in a sub-optimal linear way and
(iii) The remaining algorithms end up finding a minimum and also present good generalization.

It is important to point out that SGD's behavior is quite irregular, with multiple "comings and goings" in the convergence process. In absolute terms, RMSProp has the best performance.



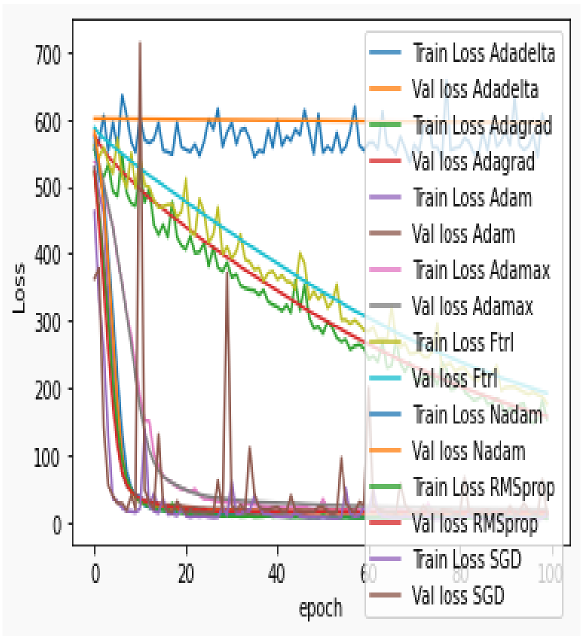
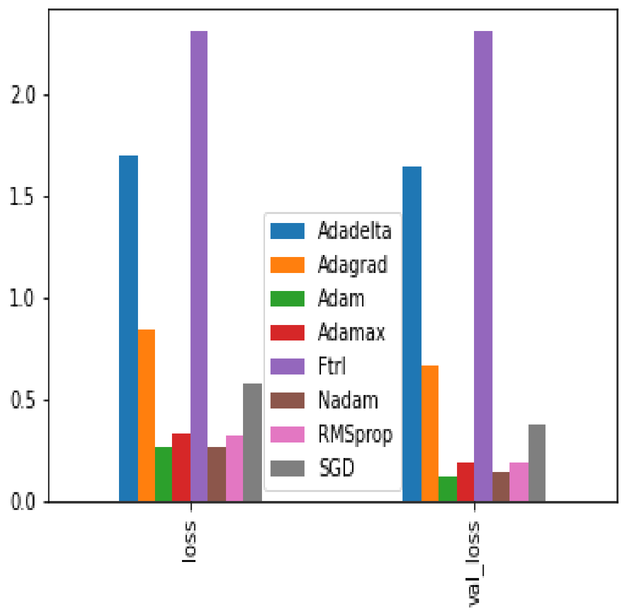


Fig 2: Convergence of the different optimizers in a simple numerical regression.

Experiment 2: In this section, we are implementing the 2D-CNN in Keras. Since our dataset is the Fashion MNIST [13], it contains images of size (28X 28) of 1 color channel instead of (64X 64) with 3 color channels. In Classifier (multi-class) of images, neither Adadelta nor Ftrl showed good behavior. The rest of the algorithms follow a similar error minimization pattern, each one at a different speed, increasing the number of epochs is probably all end up in a similar situation. Adam has the best overall performance and RMSProp the fastest convergence.



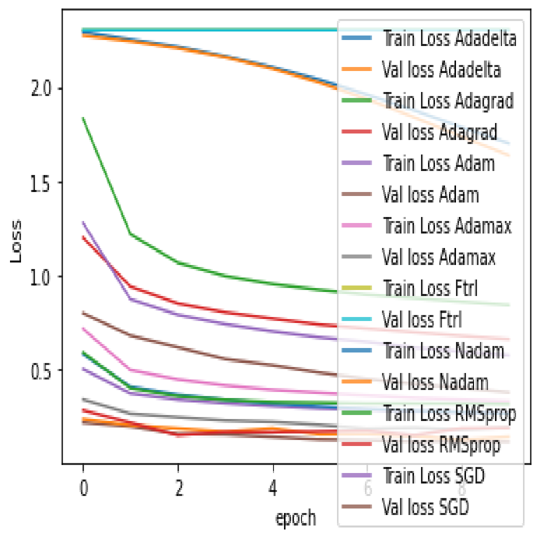
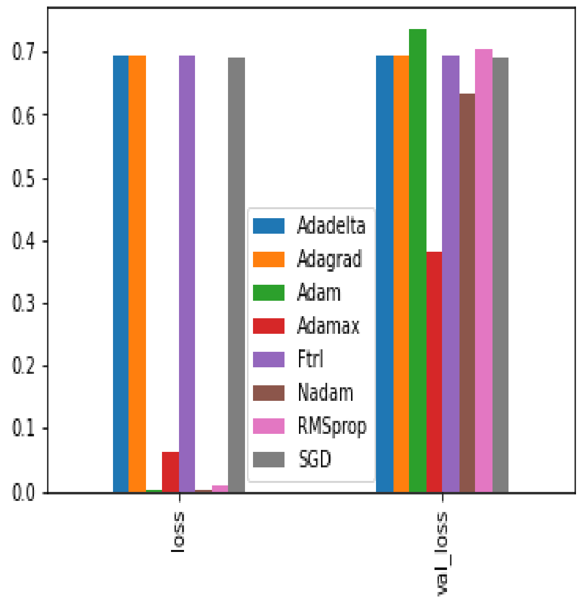


Fig 3: Convergence of the different optimizers in the multiclass CNN classifier.

Experiment 3: The data used in this project is the so-called Data Set Review Movie (Movie Review Large Dataset), which is often referred to as the IMDB data set[14]. The IMDB dataset was having 50000 extremely polar movie reviews (either good or bad), of which we used 50% for training and 50% for testing. The problem is determining whether criticism or review of a movie has either positive or negative sentiment. The training of the CNN network presents a peculiar behavior in text classifiers (binary). We see how the overfitting phenomenon appears with almost all optimizers. Therefore we explored the training with more epochs, applying regularization or drop-outs. RMSProp has the best absolute value and Adam the most efficient convergence.



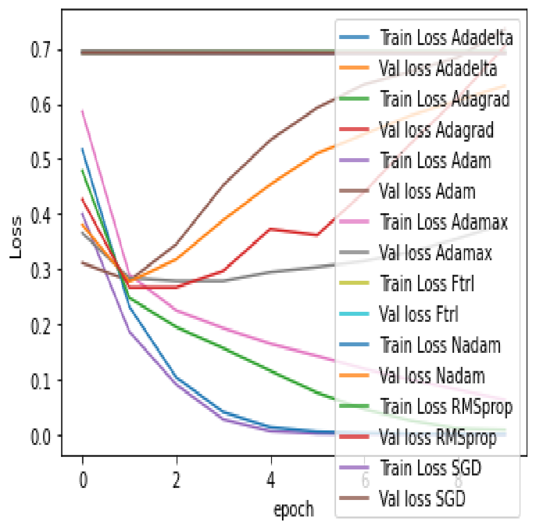
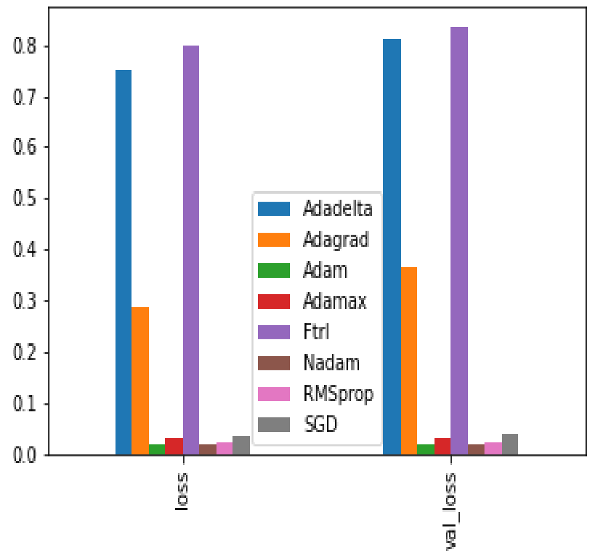


Fig 4: Convergence of the different optimizers in a binary CNN classifier with Embeddings.

Experiment 4: A time series is a collection of observations of a variable taken sequentially and ordered in time. The series can have annual, semi-annual, quarterly, monthly, etc., according to the periods in which the data that compose it is collected. Time series can be defined as a particular case of stochastic processes since a stochastic process is a sequence of random, ordered and chronologically equidistant variables referred to as observable characteristics at different times. In the forecasting of the time series scenario, we again observe three different convergence patterns. (i) Frtl and Nadam do not find the minimum (Ftrl is expected, given its particular applicability). (ii) Adagrad shows a suboptimal convergence, and (iii) the rest of the algorithm has similar behavior, with Adam and RMSProp being the ones with the best behavior.



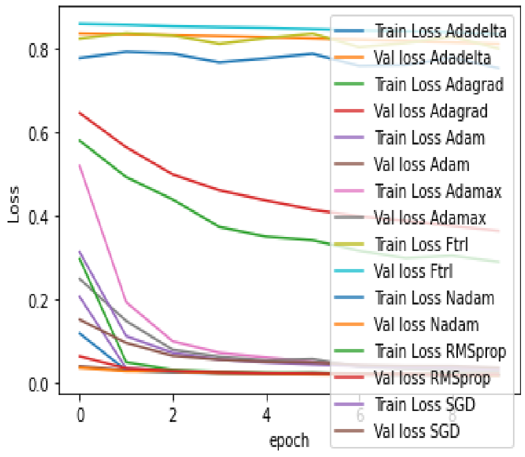


Fig 5: Convergence of the different optimizers with RNN LSTM

V. CONCLUSIONS

The Neural network optimizers usually use some type of GD algorithms to drive the backpropagation, frequently with a system that helps avoid getting stuck at a local minimum, for instance, optimizing the selected mini-jobs randomly, and applying moment corrections to the slope. Some GD algorithms adapt the rate of learning to the model parameters by look into the history of gradients [15]. In conclusion, we have observed in an empirical way how perhaps the Adam algorithm presents an adequate behavior in different problems; therefore, it can be a good point to start testing in our models. An extremely important factor (perhaps the most important hyper-parameter) that we cannot ignore and that we have assumed constant is the training factor. In successive notes, we explored how, once an optimizer is selected, we can adjust its training factor and understand the trade-off between this factor, execution times, and convergence.

REFERENCES

[1] Rajendra, P., Subbarao, A., Ramu, G. et al., Prediction of drug solubility on parallel computing architecture by support vector machines, Netw Model Anal Health Inform Bioinforma 7 (13) (2018).https://doi.org/10.1007/s13721-018-0174-0

[2] Murthy N, Saravana R, Rajendra P., Critical comparison of northeast monsoon rainfall for different regions through analysis of means technique, Mausam, 69 (2018) 411–418

[3] Narasimha Murthy, K.V., Saravana, R. & Rajendra, P., Unobserved component modeling for seasonal rainfall patterns in Rayalaseema region, India 1951–2015. Meteorol Atmos Phys 131 (2019) 1387–1399. https://doi.org/10.1007/s00703-018-0645-y

[4] Rao AS, Sainath S, Rajendra P, Ramu G., Mathematical modeling of hydromagnetic Casson non-Newtonian nanofluid convection slip flows from an isothermal sphere. Nonlinear Eng, 8(1) 645–660. https://doi.org/10.1515/nleng-2018-0016

[5] Kim, Sung Eun & Seo, Il.Artificial Neural Network ensemble modeling with conjunctive data clustering for water quality prediction in rivers, Journal of Hydro-environment Research, (2015). Doi: 9. 10.1016/j.jher.2014.09.006.

[6] A. S. Nemirovsky and D. B. Yudin., Problem Complexity and Method Efficiency in Optimization, John Wiley & Sons,(1983).

[7] N. L. Roux, M. Schmidt, and F. R. Bach., A stochastic gradient method with an exponential convergence rate for finite training sets, in Advances in Neural Information Processing Systems, (2012) 2663–2671.

[8] Mizutani E, Dreyfus SE.. Second-order stagewise backpropagation for Hessian-matrix analyses and investigation of negative curvature, Neural Netw, 21(2-3) (2008) 193-203.Doi: 10.1016/j.neunet.2007.12.038.

[9] Defazio, A., Bach, F. R. & Lacoste-Julien, S., SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives, In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence & K. Q. Weinberger (eds.), (2014)1646-1654. NIPS.

[10] Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., & Bengio, Y., Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, Advances in Neural Information Processing Systems, (2014) 2933-2941.

[11] LeCun, Y., Bengio, Y. & Hinton, G., Deep learning, Nature, 521(2015) 436–444. https://doi.org/10.1038/nature14539

[12] Boston housing price dataset: https://www.kaggle.com/c/boston-housing (Accessed on 24/03/2021)

[13] Fashion MNIST dataset: https://www.kaggle.com/zalando-research/fashionmnist (Accessed on 24/03/2021)

[14] Sentiment analysis IMDB dataset: https://datasets.imdbws.com (Accessed on 24/03/2021)

[15] Rajendra, P., Brahmajirao, V., Modeling of dynamical systems through deep learning, Biophys Rev,12 (2020) 1311–1320.https://doi.org/10.1007/s12551-020-00776-4