Optimizers in Deep Learning: An Imperative Study and Analysis

Ajeet K. Jain[1], PVRD Prasad Rao[2], K. Venkatesh Sharma[3]

[1]Research Scholar, Department of Computer Science and Engineering
Koneru Lakshmaiah Education Foundation, Vaddeswaram, AP, India.
{jainajeet123@gmail.com} {jainajeet1@rediffmail.com}

[2] Professor, CSE, KLEF, Vaddeswaram, AP, India
{pvrdprasad@kluniversity.in}

[3]Professor, CSE, CVR College of Engineering, Hyderabad, India
{venkateshsharma.cse@gmail.com}

ABSTRACT: *Machine learning has enormously contributed to optimization techniques motivating new approaches for optimization algorithms. The optimization approaches in deep learning have broad applicability with the resurgence of novelty starting from Stochastic Gradient Descent to convex and non-convex and derivative-free methods. Selecting an optimizer is a vital choice in deep learning. The optimization algorithm chosen to have convexity principles in its core determines the training speed and final performance predicted by the DL model. The complexity further increases with growing deepness due to hyperparameter tuning, and as the data sets become more significant, they require a better optimizer.*

In this work, we analyze the most popular optimizers algorithms. The augmenting behaviors are tested on MNIST, AutoEncoder data sets, pointing out similarities, differences, and the likelihood of suitability for a given application. The article focuses on the critical role and pinpoints which one would be a better option.

Keywords: Deep Learning, Optimizers, ADAM, Yogi RMS Prop

DOI: 10.6025/stm/2021/3/99-106

Received: 22 September 2021, Revised 20 October 2021, Accepted 4 November 2021

1. Introduction

In ML, the study of text classification has been undertaken by many as it provides the fundamental problem of learning from examples. When the datasets size increases, deep learning (DL) algorithms are indispensable with statistical computations for higher efficiency. Interestingly, one of the pillars of DL is the mathematical tactics of the optimization process, which makes decisions based on previously unseen data. This process is accomplished by carefully chosen parameters (near-optimal solution intuitively) for a given learning problem. *The* hyperparameters *are the parameters of a learning algorithm and not of a given mode*l. The inspiration is to look forward to the optimizing algorithm, which works well and predicts accurately [1, 2, 3, 4].

Similarly, speech and image recognition have been dealt with with great success and accuracy – yet offers the place for new improvements. In achieving higher goals, using various optimizing techniques involving convexity principles is much more cited [5, 6, 7] nowadays, and using logistic and other regression techniques. Moreover, the Stochastic Gradient Descent (SGD) has been very popular over the last many years but also suffers from ill-conditioning and takes more time to compute for larger data sets. At times it also requires hyper-parameter tuning and different learning rates adaptively.

1.1 Background and Related work

Deep Learning (DL) has made a vital pathway in all areas of engineering applications and has generated keen interest due to its closeness to human cognition. Machine Learning (ML) has taken deep roots in addressing human life problems, including healthcare, sociology, economics, logistics, etc. So now we have intelligent products and services, e.g., speech recognition, computer vision, anomaly detection, game playing, and many such areas. The ever-changing tools and tech-

niques of ML have a widening impact on diagnosing diseases, autonomous drive, animated movie-making, innovative recommended systems, and many more. Background work is owing to "optimizer and regularization methodologies"—ranging from Gradient Descent (GD) to Stochastic GD to Momentum based optimizers [8]. Meanwhile, the convex and non-convex theory of optimization is covered briefly, and one can refer more to these topics in cited references [9, 10]. The whole convex optimization methods are relaxed now a day with conditioning, and suitable niche algorithms are investigated to find the most probable accuracy results and thus less execution time and more result-oriented.

The research study seeks a thoughtful process to answer most of the following questions and encompass the gamut of equilibrants to address the issues and challenges. The pertinent ones are:

- How to measure the optimal performance? What are those attributes which compare them using various techniques in deep learning?

- How could non-convex methods be transformed into convex methods?

- How are derivative-free algorithms getting importance while reducing computational time?

- How is hyper-parameter tuning done for optimization techniques?

2. Optimization and Role of Optimizer in DL

To control the output from layers, we assess them against expected. The closeness between them is desired as an expectation. It is the *loss function* of the network- *aka*, objective function [1, 3, 4]. It takes the predicted value and compares it against the valid target, i.e., what we wanted as the actual output, and calculates the difference, depicted in Fig.

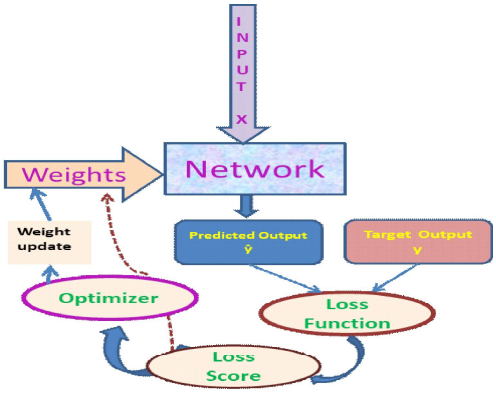

Figure 1. An Optimizer Framework

The basic principle utilizes this scored difference as a pointer to regulate the value of the weights accordingly, leading toward a lesser loss score. This adjustment is the function performed by the '*optimizer*,' which augments what is conventionally known as the back propagation algorithm [11, 12, 13, 14, 15].

2.1 Optimization Issues

The criticalities of optimization issues in DL are fairly intricate, and a pictorial representation is in Fig. 2 with enumeration as:

(i) Making the algorithm starts running and converging to a realistic solution.

(ii) Making the algorithm converge fast and speed up the convergence rate.

(iii) Ensuring convergence with a stumpy value–like global minimum.

2.2. Stochastic GD Optimization

Ironically, SGD closely follow the gradient of a mini-batch selected at random. While training a network, we estimate the gradient using a suitable loss function. At an iteration '**k**', the gradient will be updated accordingly. Hence, the calculation for '**m**' examples input from the training set having *y*(*i*) as target, is:

Figure 2. Optimization: Issues and Challenges

$$\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)}) \qquad (1)$$

$$\theta \leftarrow \theta - \eta\,\hat{g}$$

here 'η' (eta) is learning rate. Further, the learning rate is of paramount importance as the magnitude of an update at ' k^{th} ' iteration is dictated by this. For instance, if η = 0.01 (very small to small), then evidently more number of iteration updates will be required for convergence. On the contrary, if η = 0.5 or more, then in this case the recent updates shall be highly dependent on the recent instance. Eventually, an obvious wise decision is to select (choose) it arbitrary by trial —this is one very important hyper-parameter tuning in DL systems. On the parallel side of it, yet another way could be 'choose *one among several learning rates*' which provide smallest loss value.

2.3. Stochastic Gradient Descent with Momentum

From the preceding section and paragraphs, it is obvious that SGD has a trouble to get towards global optima, and has a tendency to get stuck into local minima, as depicted in Figure 3.

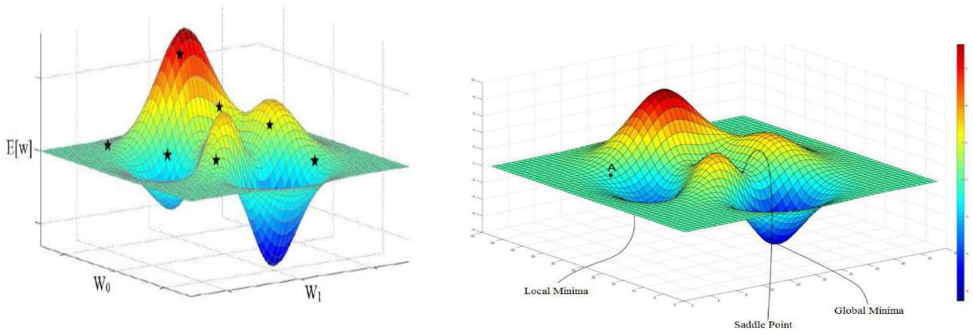

Figure 3. A 3-D Representation with Local and Global Minima (Maxima)

Moreover, smaller values of gradient or noisy ones can create another problem of vanishing gradient issue! To overcome this problem, the method involving momentum (a principle borrowed from physics) is adopted to accelerate the process of learning. The momentum method aims to resolve 2 very important issues:

(i) variance in SGD

(ii) variance when solving Hessian Matrix for poor conditioning

The method takes the brevity of running moving average by incorporating previous update in the recent change as if there is

A momentum due to preceding updates.

The momentum based SGD will converge faster with reducing oscillations. To achieve this, we use another hyper-parameter '**v**' known as velocity. This hyper-parameter tells the speed and of course the direction by which its moves in the given parameter space. Usually '**v**' is set as negative of gradient value of exponential decaying average. Moving further on, we would require yet one more hyper-parameter α (alpha): $\alpha \in [0 , 1]$, known as momentum parameter and its contribution is to find how fast the previous gradient exponentially decays. The new (updated) values are computes as

$$v \leftarrow \alpha v - \epsilon \frac{1}{m} \nabla_\theta \Big(\sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)}) \Big) \qquad (2)$$

$$\theta \leftarrow \theta + v$$

From equation (2) it is obvious that the velocity vector '**v**' keeps on adding the gradient values. Moreover, for a bigger value of α (alpha) relative to ε, the gradient affects the current direction more from previous iteration. The commonly used values of α: from 0.5 to 0.99. Despite being so intuitive and nice technique, the limitation of this algorithm is additional parameter inclusion and extra calculations involved.

3. Various Optimizers in DL Practitioner Scenario

The DL practice is more of the optimization and regularization process. The currently available optimizers with their process framework are briefly described with their relative merits and limitations. Each one has some peculiarities or the other and an insight of those will an exemplary thought progression.

3.1. AdaGrad Optimizer

The simplest of optimizing algorithms to begin with is **AdaGrad**, where the algorithm's names itself suggest, the algorithm adapts, i.e., dynamically changes the learning rate with model's parameters Here, for parameters whose partial derivative are higher (larger) for them decrease their corresponding learning rate substantially. Contrary to this intuition, the algorithm takes inversely to where derivatives are lower. A natural question to ask is '*why one needs different learning rates'?* So to accomplish these characteristics, **AdaGrad** employs square value of the gradient vector using a variable '**r**' for gradient accumulation, as stated in following equation(3).

$$g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)}) \qquad (3)$$

Using this equation (6) the square of gradient is collected and subsequently the update of parameters is computed by a scaling factor $\boldsymbol{\delta} + \sqrt{\mathbf{r}}$, where δ is a very low value constant for numeric stability. The update applied as per the following equation (4) now:

$$r \leftarrow r + g \odot g$$

$$\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g \qquad (4)$$

$$\theta \leftarrow \theta + \Delta\theta$$

Here $\odot$ operator implies element-wise multiplication of vectors. As can be inferred from above equations, when '**r**' is close to a 'near-zero' value, the term in the denominator should not be evaluated as 'NaN = Not A Number' and thus the term ä helps to avoid this to happen. Also, the term '**e**'stands for global learning rate.

3.2. RMSProp

The modified version of **AdaGrad** is **RMSProp** – **R**oot **M**ean **S**quare **Prop**ortional [16]. In order to alleviate the problems of **AdaGrad,** here we recursively define a decaying average of all past gradients. By doing so, the running exponential moving average at each time step depends only on the average of previous and current gradients. It performs better in the non-convex setting as well with same characteristics features. Comparison wise, **AdaGrad** contracts the learning rate according to the entire

history of the squared gradient whereas **RMSProp** exploits an exponentially decaying average to discard history from the extreme past such that it can converge quickly after finding a convex bowl. The equation to implement is:

$$r \leftarrow \rho r + (1-\rho) g \odot g \tag{5}$$

here ρ is the decay rate. Then parameter update is computed and applied as follows:

$$\Delta\theta = -\frac{\epsilon}{\sqrt{\delta + r}} \odot g$$
$$\theta \leftarrow \theta + \Delta\theta \tag{6}$$

3.3. Adam

Adam (**Ada**ptive **M**omentum) one majorly used optimization algorithms in DL and joins the heuristic of both the momentum and **RMSProp** and interestingly been designed for deep neural nets [17]. This algorithmic technique has the squared gradient feature of **AdaGrad** and to scale the learning rate analogous to RMSProp and feature of momentum using moving average. The fine algorithm calculates individual learning rate for each parameter using a term called '*first moment*' (similar to velocity vector) and '*second moment*' (similar to acceleration vector). A few salient features are :

— Momentum term is in-built as an estimate of first-order moment

— In-built bias correction while estimating for first and second order moments - sometime called as initialization at origin (start point)

— Update moving exponential averages of gradient '$\mathbf{m}_t$' and square gradient '$\mathbf{u}_t$' with hyper-parameters ρ1 and ρ2 (in original paper by the authors, they are denoted by b1 and b2, respectively) as these control the cited decaying rates.

These moving averages are estimation of mean (first moment) and uncentered variance (second moment) of gradient. Moving ahead in the pipeline process, at time step '*t*', the various estimates are as follows:

$$m_t \leftarrow \rho_1 m_{t-1} + (1-\rho_1) g_t$$
$$u_t \leftarrow \rho_2 u_{t-1} + (1-\rho_2) g \odot g \tag{7}$$

Then, bias is corrected in first and second moments. By using the corrected moment estimates parameter updates are calculated and applied:

$$\hat{m}_t \leftarrow \frac{m_t}{1-\rho_1^t} \qquad \hat{u}_t \leftarrow \frac{u_t}{1-\rho_2^t}$$
$$\Delta\theta = -\epsilon \frac{\hat{m}_t}{\sqrt{\hat{u}_t} + \delta} \tag{8}$$
$$\theta_t \leftarrow \theta_{t-1} + \Delta\theta$$

From [28], the default values are: **ρ1** (**β1**) = **0.9** and **ρ2** (**β2**) = **0.999** and **δ** = **10** $^{-8}$.. Adam works quite well in deep learning scenarios and is one of the most favored adaptive learning-method algorithms.

3.4. Yogi Optimizer

One of the problems of Adam is that it can fail to converge even in convex settings when the second moment estimate blows up. As a fix [xxx] proposed a refined update optimizer called "Yogi" whose analysis is shown with respect to other optimizers [18].

.

4. Experimental Analysis

Simple MNIST ConvNet (CONVolutional neural NETwork) Handwritten Digit Classification (keras.io)

The MNIST dataset contains 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9 and to classify them into one of 10 classes representing integer values from 0 to 9.

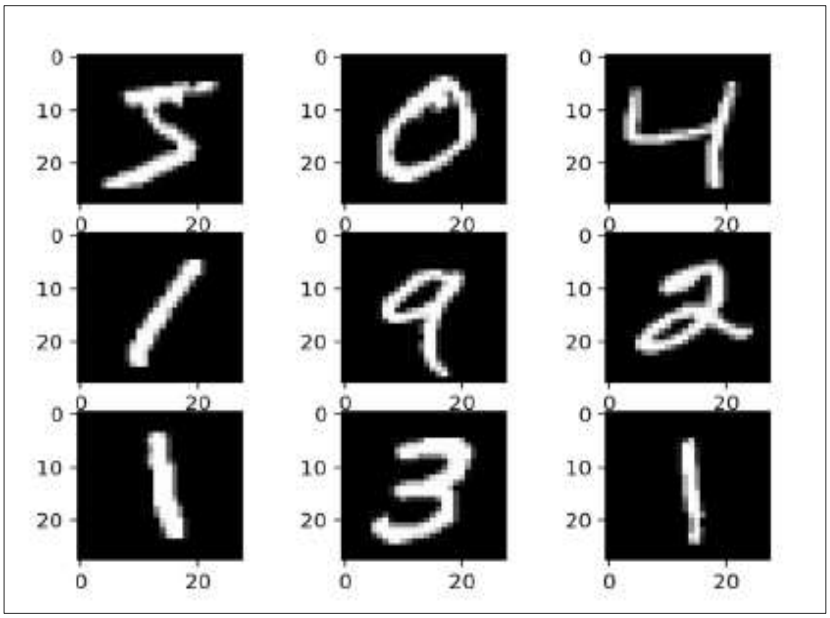

Figure 4. MNIST Hand-writing data

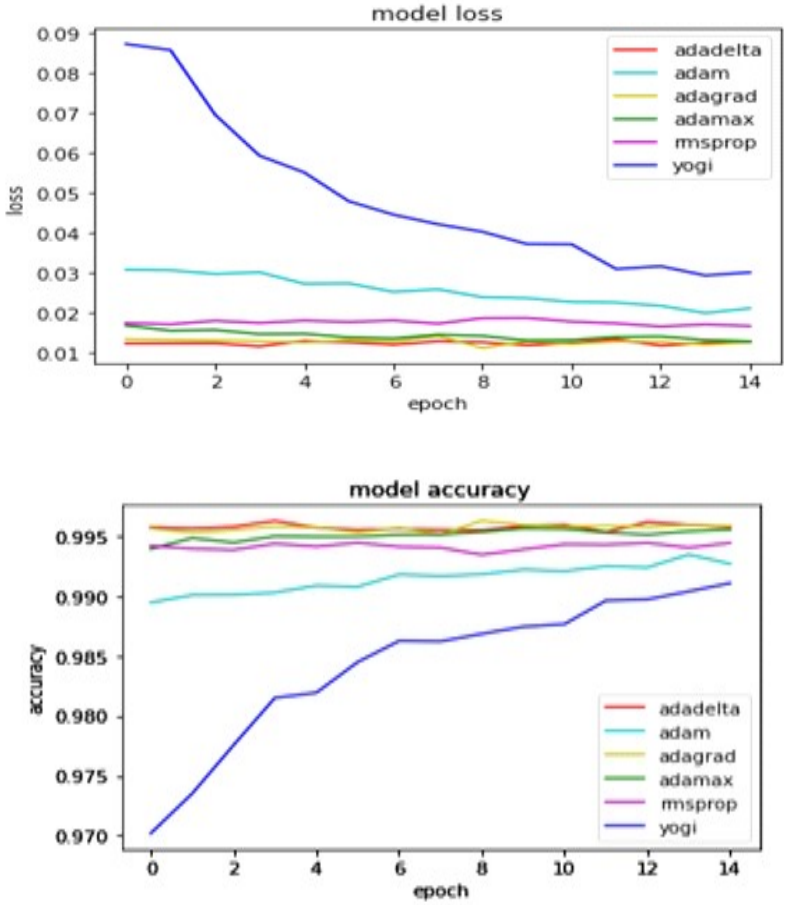

Figure 5. A comparative graph depicting various optimizers

5. Discussion and Conclusion

As can be visually analysed from various graphs from the experimental analysis, some salient points emerges:

— A good start is to have technique that usually works well with ReLU and tuning of a few hyperparameters

— Start with a simple dataset having fewer (easier) samples with a simple network, and after obtaining promising results gradually increase the complexity of both data and network while tuning hyperparameters and trying regularization methods.

— Training with random initialization is first choice and later one can go with pre-trained model to speed up

— Begin with well-known optimizer to get results and trying out with each will provide a reconnaissance and might lead to improvement. The correctly chosen optimizer with learning rate and other parameters makes a good deal.

— *Selecting a Particular Optimizer*: Choose a well-understood optimizer with default learning rates and other parameters settings. Try to change these parameters with iterations (epochs) and see the loss (accuracy) results. Subsequently, shift towards other similar-featured optimizer and observe the changes. *This is indeed an exhaustive process*! Nevertheless, keep experimenting by increasing towards momentum based and other current blends of optimizers (like Adam, AMSGrad or RADam, etc.).

The most popular in DL research community are Adam and RMSProp and the results were shown as promising ones. However, this imperativeness provides an insightful for making a visionary choice of optimizers. Also, getting an overview of their criticalities and understanding the reasons for choice makes a footing platform in DL [37,38,39].

References

[1] Goodfellow, Ian., Bengio, Yoshua., Courville, Aaron. (2016). Deep Learning MIT Press, USA 2016

[2] Bishop, C.M. (1995). Neural Network for Pattern Recognition, Clarendon Press, USA 1995

[3] Chollet, François. (2018). Deep Learning with Python, Manning Pub., 1st Ed, NY, USA, 2018

[4] Jain, Ajeet K., PVRD Prasad Rao., Venkatesh Sharma, K. (2021). *A Perspective Analysis of Regularization and Optimization Techniques in Machine Learning*, Computational Analysis and Understanding of Deep Learning or Medical Care: Principles, Methods and Applications. CUDLMC 2020, Wiley-Scrivener, April/May.

[5] Mueller, John Paul., Massaron, Luca. (2019). Deep Learning for Dummies, John Wiley, 2019

[6] Patterson, Josh., Gibson, Adam. (2017). Deep Learning: A Practitioner's Approach, O'Reilly Pub. Indian Edition, 2017

[7] Jain, Ajeet, K., PVRD Prasad Rao., Sharma, Venkatesh. (2020). Deep Learning with Recursive Neural Network for Temporal Logic Implementation, *International Journal of Advanced Trends in Computer Science and Engineering*, 9 (4) July – August 2020, 6829-6833.

[8] Srivasatava, et al. http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf

[9] Bertsekas, Dimitri P. (2009). Convex Optimization Theory, Athena Scientific Pub., MIT Press, USA.

[10] Boyd, Stephen., Vandenberghe, Lieven. Convex Optimization, Cambridge University Press, USA.

[11] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1 (4) 541–551.

[12] Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580.

[13] Glorot, X., Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), p. 249–256.

[14] Glorot, X., Bordes, A., Bengio, Y. (2011). Deep sparse rectifier neural networks. *In*: Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), p. 315–323.

[15] Zeiler, M., Fergus, R., Stochastic pooling for regularization of deep convolutional neural networks. In Proceedings of the International Conference on Learning Representations, ICLR, 2013.

[16] Fabian Latorre, Paul Rolland and Volkan Cevher, Lipschitz Constant Estimation Of Neural Networks Via Sparse Polynomial Optimization, ICLR 2020

[17] Kingma, D., Ba, J. (2014). Adam: A method for stochastic optimization, arXiv:1412.6980.

[18] Manzil Zaheer, et al (2018). Adaptive Methods for Nonconvex Optimization, 32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.