Pragmatic Assessment of Optimizers in Deep Learning

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

[1]Research Scholar, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, AP, India; (*Association: Asst Prof. , CSE, KMIT, Hyderabad, India*)
[2] Professor, CSE, KLEF, Vaddeswaram, AP, India
[3]Professor, CSE, CVR College of Engineering, Hyderabad, India

Summary
Deep learning has been incorporating various optimization techniques motivated by new pragmatic optimizing algorithm advancements and their usage has a central role in Machine learning. In recent past, new avatars of various optimizers are being put into practice and their suitability and applicability has been reported on various domains. The resurgence of novelty starts from Stochastic Gradient Descent to convex and non-convex and derivative-free approaches. In the contemporary of these horizons of optimizers, choosing a best-fit or appropriate optimizer is an important consideration in deep learning theme as these working–horse engines determines the final performance predicted by the model. Moreover with increasing number of deep layers tantamount higher complexity with hyper–parameter tuning and consequently need to delve for a befitting optimizer. We empirically examine most popular and widely used optimizers on various data sets and networks–like MNIST and GAN plus others. The pragmatic comparison focuses on their similarities, differences and possibilities of their suitability for a given application. Additionally, the recent optimizer variants are highlighted with their subtlety. The article emphasizes on their critical role and pinpoints buttress options while choosing among them.

***Keywords**:*
Deep Learning, Optimizers, Convexity, Lottery Ticket, ADAM, RMS Prop

1. Introduction

Optimizers are indispensable with statistical computations for higher efficiency when the dataset size increases in Deep Learning (DL). Quite interestingly, one on the mainstay of optimization process approach is to make decision based on previously unseen data using statistical methods. This is achieved by carefully chosen parameters for an (sub) optimal solution for a given learning problem. Manifestly, the idea is to look for those optimizing algorithms offering better performance and prediction accuracy [1, 2, 3, 4]. For instance, the text classification provides the fundamental problem of learning from examples. Likewise, speech and image recognition have been studied with paramount performance and accuracy – yet offers improvements avenues. In striving towards these goals, many optimizing techniques involving non–convexity and convexity principles are cited in the ML literature [5, 6, 7]. The Stochastic Gradient Descent (SGD) has been very well accepted optimizing technique past many years, but suffers from ill-conditioning and taking more computation time with bigger data sets. Moreover, it also requires hyper-parameter tuning and different learning rates adaptively. Recently with the advent of new variants of optimizers offering superior performance and sometime outperforming over their counterpart on bigger datasets, it is imperative to seek how these new entrants are providing such a phenomenal performance improvement. We examine the most popularly used optimizers in DL framework using known data sets. We also accentuate the significance of role of optimizers and pragmatically compare them in response to their learning rate, suitability plus other selection criterions.

Manuscript received October 5, 2025
Manuscript revised October 20, 2025
https://doi.org/10.22937/IJCSNS.2025.25.10.10

1.1 Related work

Deep Learning (DL) has paved a strapping path in engineering applications and has generated keen interest in optimizers as working horse solving varied human like problems including healthcare, marketing, finance, sociology, economics, logistics and others. Importantly, we have intelligent products and services like speech recognition, computer vision, anomaly detection, game playing, disease diagnosis, autonomous drive, animated movie making, smart recommended systems and many more. The flogging work is due to optimizers—starting from Gradient Descent (GD) to Stochastic GD to Momentum based optimizers [8-14].

In this work, we are addressing some of the pertinent questions for optimization, those can be delve deep more comprehensively.

What is the measure of optimal performance? What are those attributes which pragmatically compares them in DL?

- How to transform non-convex (functions) techniques into convex ones?
- Are derivative free algorithms more computational efficient?
- What are hyper-parameter optimization methods?
- How modern optimizers are useful to address DL scenarios? How recently known 'Lottery Ticket Algorithm' facilitate for optimization process?

We also highlight the selection process which offers better results while selecting the '*right'* among vivid optimizers. A pragmatic comparison highlights their relative importance, merits and other subtleties.

2. Role of Optimizer in DL

This section highlights about the role of optimizers and performance issues and challenges therein. We need some way to evaluate whether our optimizing algorithm is performing correctly to reduce difference between the current output and the targeted output, and this difference is a response indicator to regulate the difference obtained by the optimizer to make appropriate adjustments. The weights are fundamentally a bunch of numbers and are stored as pattern carrying information what a layer does with its input data. Hence the learning is primarily to update weight values for all the layers such that it accurately maps them to their connected targets. Here is a problem —a deep network may contain hundreds (or even more!) of parameters, so finding their precise values for each layer is a formidable task, particularly knowing that amending value of one parameter influences the rest. To control the output from layers, we calculate them against expected ones and their closeness is desired as expectation and this is accomplished by *loss function* of the network [1, 3, 4] as depicted in Fig. 1.

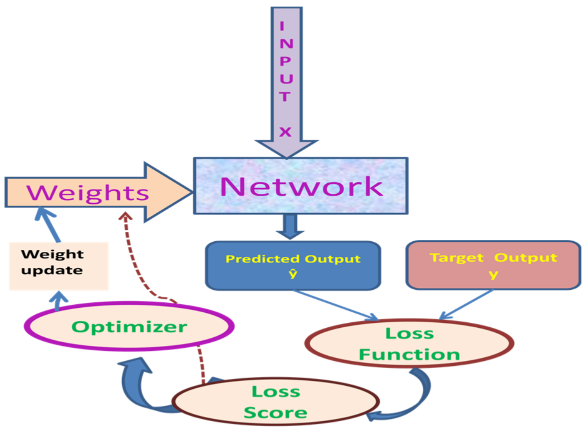


Figure 1 An Optimizer Framework

Basically this difference is a pointer to regulate the value of the weights accordingly leading towards lesser loss score. This modification is performed by '*optimizer*', which augments what is conventionally known as back propagation algorithm.[11-14].

Initially, the weights are given random (smaller) values and the network performs a sequence of arbitrary transformations. As expected the discrepancy should be lowered against the larger values. Nevertheless, the network update weights in correct direction for every input and thereby propitiating towards decrease of loss. This process continues as iteration and the training loop is repeated (aka epoch) — and each epoch yields a correct weight update values augmenting to reduce the loss further less. Ideally, the idea is to reduce it towards near zero. A cost function usually just calculates the mean squared error (loss) between an actual output and the desired output when performing supervised learning tasks. So a loss function can be considered synonymously with a cost function [15]. How to train a NN successfully? To accomplish this we need (i) power engine (latest multi-core CPU / GPU) (ii) a suitable network, and (iii) a suitable algorithm with accurate training artifices.

2.1 Deep Network Architecture.

The architectural and activation functions are most important and one can think of a deep network with at least 4-5 layers and sufficient connections. Achieving good performance, one can go to even a deep level of 20 or more and add skip connections. Primarily, **ReLU** (**Re**ctified **L**inear **U**nit) activation function begins a good choice, but depending upon the characteristics of data set, using **tanh** or **swish** activation functions might prove a superior choice while training the algorithm [16].

Additionally another criticality is to use SGD with well-tuned constant step-size, however, momentum and adaptive step size offers additional profit. A prototypical is depicted in Fig.2 interposing 3 aspects with their effects.

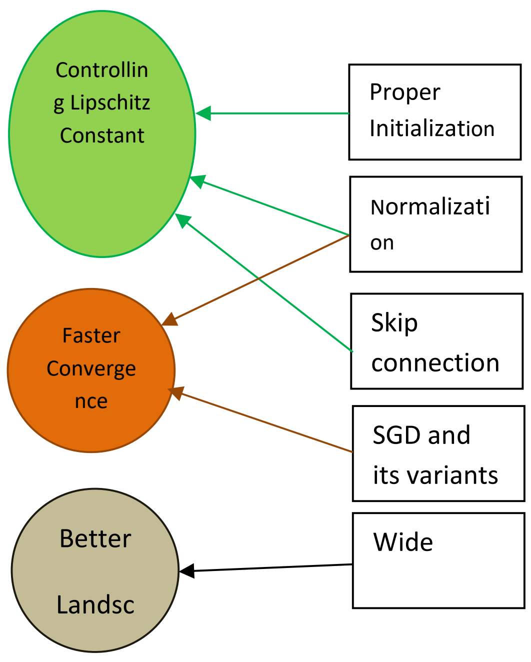


Figure2: preferred choices for training of a NN

They offer 3 characteristics of algorithmic convergence, namely: (i) make convergence possible, (ii) faster convergence and (iii) better global solutions. All three aspects are interrelated.

(i) **Lipschitz** constant is the maximum ratio between variations in the output space and variations in the input space of *f*. This measures the sensitivity of the function with respect to input perturbations.

A function f: □ $^{\mathrm{n}} \rightarrow$ □ $^{\mathrm{m}}$ is called Lipschitz continuous if there exists a constant L such that

$$\forall x, y \in \mathbb{R}^n, \|f(x) - f(y)\|_2 \leq L\|x - y\|_2 \quad (1)$$

For locally Lipschitz functions, it may be computed using the differential operator. For more details, one can refer to [17, 18].

(ii) **Proper Initialization** - is extremely significant to start in order to train a network having many layers and two extra artifices can improvise: adding normalization layers and adding skip connections. Imperatively, which one is a design or a critical choice? Typical these include: initialization strategies, normalization methods, the skip connections, over-parameterization as depicted in Figure 3.

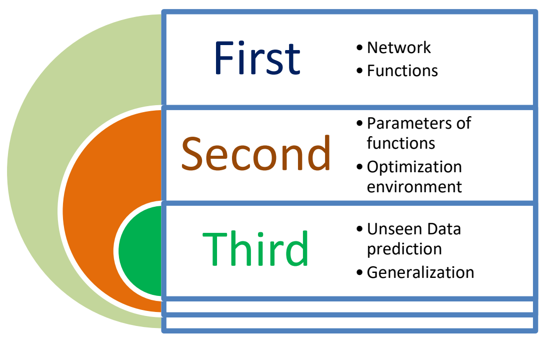


Figure 3. Three steps towards generalization

(iii) **Representation, Optimization and Generalization**. For a supervised learning, we need to find a function that approximates observed samples and identify those parameters for minimizing the loss. Also, we need to use a function from the previous step to make predictions on test data and calculate the resultant test error. These can be further divided into *representation*, *optimization* and *generalization* errors, respectively.

In DL, these errors are often calculated disparately—while noting the representation supremacy of a certain class of functions; we often do not try to look into optimization problem closely. Similarly, while noting the generalization error, we often take for granted that the global optima have been found and in the same way tend to ignore the generalization error while noting optimization properties, assuming the representation error is *zero* [19, 20, 21].

2.2 Optimization Issues:

The challenges for optimization are fairly intricate and are being represented in Fig. 4.

(i) Making the algorithm to converge to a realistic solution.

(ii) Speeding convergence rate.

(iii) Ensuring convergence like a global minimum.

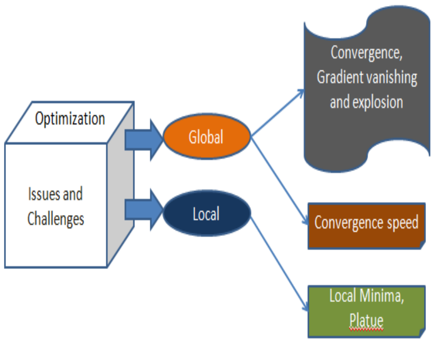


Figure 4. Optimization: Issues and Challenges

All these challenges are chiefly inspired by optimization techniques although the vertical divide is never clear-cut due to blur boundaries. For instance, some algorithms provide better convergence rate—citing it as a global challenge. Others visualize these as sub-areas of DL optimization techniques as an important artefact.

2.3 Stochastic GD Optimization

Paradoxically SGD follows the gradient of a mini-batch 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:

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

(2)

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

where'η' (eta) is the 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, then evidently more number of iteration updates will be required for convergence. On the contrary, if η = 0.5 or greater, then in this case the recent updates shall be highly dependent on the recent instance. Eventually, an obvious wise decision is to 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.

This intrigue technique is known as '***line search***'- a very popular scheme for DL fraternity to tuning. Yet another intuitive way is to monitor for first few beginning epochs and make a prudent choice of 'η' offering best or near-to-best performance [22-26. In later part, a more interesting technique **MAS** (**M**ixing **A**dam and **S**GD) is discussed to elaborate.

2.4 Stochastic Gradient Descent with Momentum

The SGD algorithms have a trouble to get towards global optima, and have a tendency to get stuck into local minima (Figure 5).

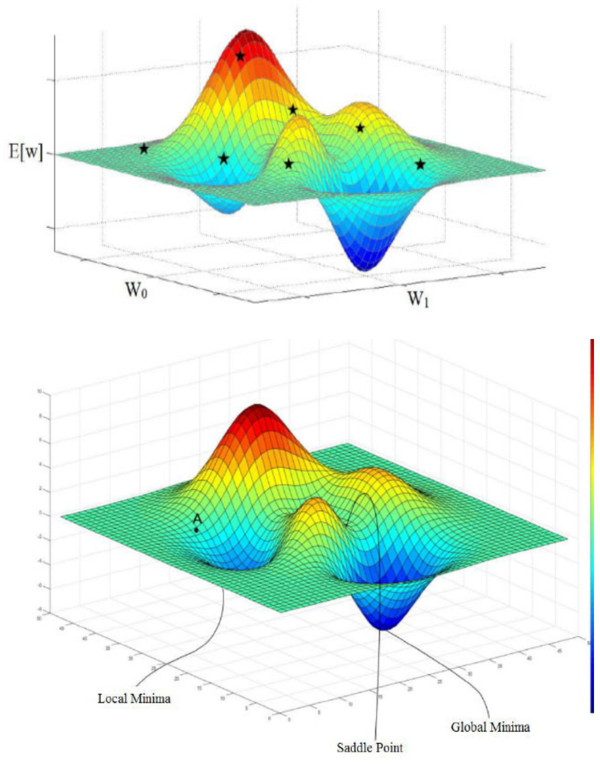


Figure. 5 A 3-D Representation with Local and Global Minima (Maxima)

Furthermore, smaller values of gradient can create problem of vanishing gradient! To overcome this, momentum based principle is adopted to accelerate the process of learning. The method takes 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 converges faster with reducing oscillations– in order to achieve this, we use another hyper-parameter **'v'** known as velocity. 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): α ϵ [0 , 1], known as momentum parameter and its contribution is to find how fast the previous gradient exponentially decays. The updated values are computes as:

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

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

(3)

From equation (3) it is obvious that the velocity vector **'v'** keeps on adding the gradient values. Additionally, for a bigger value of α (alpha) relative to ϵ, the gradient affects the current direction more from previous iteration. This in fact the founding principle on which ADAM optimizer works. 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

The DL practice is fundamentally aimed at optimization and regularization techniques. The contemporary optimizers with their process framework are briefly described with their relative merits and limitations.

3.1 AdaGrad

The simplest optimizing algorithm is **AdaGrad**, where dynamically changing learning rates are model parameters. Here, for parameters whose partial derivatives are higher, for them decrease their corresponding learning rate substantially, otherwise, the algorithm takes inversely to where derivatives are lower instead. *Why one needs different learning rates*? We need this adaptive feature of learning rate for following reasons:

(i) *Learning Rate for Sparse Features*—where quite a large number of zero value features are present–like in a bag of words. This necessitates that we should have a mechanism to perform larger updates on those features.

(ii) *Learning Rate for Dense Features*—where quite a large number of non-zero values are present. This kind of dense features when present in the data set necessitates for the mechanism to perform smaller updates on those ones.

To accomplish these , **AdaGrad** employs square value of the gradient vector using a variable '**r**' for gradient accumulation , as in equation(4).

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

(4)

Using this equation (6) the square of gradient is collected and subsequently the update of parameters is computed by a scaling factor ' **δ** + √**r** ' , where **δ** is a very low value constant for numeric stability. The update applied as per the following equation (5) now:

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

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

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

(5)

Here ⊙ 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 δ helps to avoid this to happen. Also, the term '**ϵ** 'stands for global learning rate.

Advantages

- Easier to use for simple learning tasks
- Manual tuning of learning rate is eliminated–set to a value of 0.01 as default

Disadvantages

- Suitable for simple problems having quadratic space
- Stops early while training even medium complexity networks, as scaling factor affects convergence and also stops before trying to get towards better solutions

3.2 RMS Prop

Root **M**ean **S**quare **Prop**ortional [27] is modified version of **AdaGrad** where it recursively defines a decaying average of all past gradients. In so doing, the running exponential moving average at each time step depends only on the average of previous and current gradients. Moreover, **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 faster after finding a convex bowl. The implementing equation is:

$$r \leftarrow \rho r + (1-\rho) g \odot g$$

(6)

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

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

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

(7)

Characteristic features

- Adapting in increase or decrease of learning rate with each epoch , i.e.,

it chooses different learning rate for each parameter
- New input data does not dramatically changes the gradient and hence convergence to local minima faster

3.3 Adam

Adam (**Ada**ptive **M**omentum) is majorly used optimization algorithms in DL and combines the heuristic of both the momentum and **RMS Prop** and interestingly been designed for DNN [28]. 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*' (like a velocity) and '*second moment*' (like acceleration). The algorism combines the characteristics of AdaGrad having sparse gradient and RMSProp having the mechanism for on-line and non-stationary data sets.

__Salient features:__

- Momentum term is in-built as an estimate of first-order moment
- In-built bias correction while estimating for first and second order moments
- Update moving exponential averages of gradient '$\mathbf{m_t}$' and square gradient '$\mathbf{u_t}$' with hyper-parameters ρ1 and ρ2 (originally these are denoted by β1 and β2 , respectively by authors) as these control the cited decaying rates.

These moving averages are estimation of mean (first moment) and uncentered variance (second moment) of gradient. In this process, at time step 't', the various estimates are :

$$m_t \leftarrow \rho_1 m_{t-1} + (1-\rho_1) g_t$$

$$u_t \leftarrow \rho_2 u_{t-1} + (1-\rho_2) g \odot g$$

(8)

Subsequently, the bias is corrected in first and second moments and thus using the corrected moment estimates parameter updates are calculated and applied as:

$$\hat{m}_t \leftarrow \frac{m_t}{1-\rho_1^t}$$

$$\hat{u}_t \leftarrow \frac{u_t}{1-\rho_2^t}$$

$$\triangle\theta = -\epsilon \frac{\hat{m}_t}{\sqrt{\hat{u}_t} + \delta}$$

$$\theta_t \leftarrow \theta_{t-1} + \triangle\theta$$

(9)

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

__Advantages__ :

(i) Requires no tuning for learning rate as no modifications needed to rescale gradients
(ii) Less memory requirements and hence the algorithm is proficient
(iii) Fitting for gradients having noisy and sparsity characteristics

3.4 AdaMax

Adamax is an algorithm with infinity norm as another variant of Adam [28]. The idea borrowed here is that, as Adam algorithm updates individual weights with respect to inversely proportional to L_2 norm, this technique is more generalized with Lp norm taken for updating.

Here, at time step '**t**', we calculate gradient with respect to stochastic way for biased first moment via training infinity norm. Subsequently, the parameters are updated according to Equation (12):

$$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$$

$$m_t \leftarrow \rho_1 m_{t-1} + (1 - \rho_1) g_t$$

$$\gamma_t \leftarrow max(\rho_2 \gamma_{t-1}, |g_t|)$$

$$\theta_t \leftarrow \theta_{t-1} - (\epsilon/(1 - \rho_1^t)) m_t/\gamma_t$$

(10)

The basic advantage of using **AdaMax** is that we need not to make correcting the values of initialization bias and parameter updates a much easier bound in comparison to Adam [28, 29].

3.5 AMSGrad

Can we have a technique where the exponential moving average guarantees convergence as one of prominent features? Obviously '*yes*' and this is what **AMSGrad** technique provides by combining the features of Adam and RMS Prop together [30]. As noticed from the previous discussions, the first and second order moments variables are modified as per Equation as: .

$$m_t \leftarrow \rho_1^t m_{t-1} + (1 - \rho_1^t) g_t$$

$$u_t \leftarrow \rho_2 u_{t-1} + (1 - \rho_2) g_t^2$$

$$\hat{u}_t \leftarrow max(\hat{u}_{t-1}, u_t)$$

(11)

The fundamental differentiation from Adam algorithm is as per Equation (14):

$$\theta_{t+1} \leftarrow \prod_{F,\sqrt{\hat{U}_t}} \left(\theta_t - \epsilon_t m_t / \sqrt{\hat{u}_t}\right)$$

(12)

Noting that, it keeps the maximum of all $\mathbf{u_t}$ until present time step and takes this value for normalizing averages successively instead of $\mathbf{u_t}$ as their counterpart in Adam. By accomplishing this way, **AMSGrad** achieves a non-increasing step size and the parameters are revised as per (14). The most noticeable feature is that **AMSGrad** neither way decreases nor increases the learning rate. In addition to this, it reduces $\mathbf{u_t}$ - which guides to non-decreasing rate in the event when gradient is huge in future expected iterations.

4. Recent Optimizers in the Fray

Recently, many things in the optimization techniques are in forefront of research community in ML and DL are striving for more with better parameter tuning and exponential decay rates and using state-of-the-art technology to improve the performance on CNN, GAN, RNN and other data sets. Here, we highlight a few recent optimizers with different blends which they exhibit close relationships with the optimizers presented so far [31].

4.1 EVE

There are two learning rates of paramount interest as depicted in Figure 4 (sec.4) and these influence the convergence and other virtues of optimizer. Recapturing the thinking towards this, we can have the adaptive gradient feature of SGD where the intended these is well suited too. So, this one is cited as modification of Adam optimizer with coefficient confining two themes—firstly adapting the learning rate locally for each parameter and secondly combining all parameters together and performing an update globally. This sounds great! And in fact it has shown to be outperforming Adam and others while training deep neural networks (DNN)-like CNN for classification of pictures (photographs) and RNN for language translation task [32].

4.2. RAdam

The problem of adaptive learning rate and its big variance is usually noticed in the early stage of training the network and the idea of '*warmup*' works as a supplement in order to reduce the variance. The term '*warmup*' step is just a parameter used to lower the learning rate to reduce impact of model deviation from learning on new data suddenly. This implies that we can use a very low learning rate for a set number of training steps, i.e., '*warmup steps*'. Afterwards we use our "*regular*" learning rate or better known as '*learning rate scheduler'*. We can also gradually increase your learning rate over the number of warmup steps. The proposed **R**ectified **Adam** (**RAdam**) is a modification by introducing a term to rectify the variance of the adaptive learning rate. The experimental results on image classification, language modeling, and neural machine translation verify our intuition and demonstrate the efficacy of this new variant [33,34] .

4.3. MAS[1] (**M**ixing **A**DAM and **S**GD)

Another variant that join together Adam and SGD by weighing the contributions collectively with the task of regular (constant) weights and exploit them at the same time by taking the best of them. This intuitive blending idea is illustrated in Figure 5 with a side caption therein. Recently various demonstrations are cited with CNN DNN incorporating different images and the conventional classification of text data with MAS optimizer and cited that it produced better results than their single SGD or ADAM counterpart optimizers individually [35].

4.4 Lottery Ticket Hypothesis

From the passion of gambling world, the inspiration is to select (get) a ticket whose chance of winning is highest! In the same way, the training of DL models is often compared with lottery to buy every possible ticket. However, if we know the winning process, it seems we can make a prudence choice about selection process. By the same token, in DL models, the training processes produce large structures of inter-connections of NN equivalent to a large container of lottery tickets. In this hypothesis, the model undergoes optimization techniques-like pruning that remove unnecessary weights from NN in order reduce model size without compromising the performance.

Thus this is in turn equivalent to searching of winning tickets from the container leaving the rest. Such pruning process produces structures which are almost 90% smaller than the original NN structure [34]. This idea leads to the hypothesis that a large NN contains a smaller sub-network that if trained properly, will attain a similar accuracy in counterpart.

Thus looking at the concision of this hypothesis, it opens more fronts for understanding and research to become one of the most important DL fields as it challenges the conventional wisdom in DL network training. For more insightfulness, one can refer to [33, 34, 35, 36].

[1] https://gitlab.com/nicolalandro/multi optimizer

5. Experimental Results

A. Web Resource:

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html

The various optimizers were tested using classification from the **sklearn** data sets by generating random classification. Initially, this creates a cluster of points normally distributed with standard deviation as 1 about the vertices of an **n_informative** hyper cube with sides of length **2*class_sep** and assigns an equal number of clusters to each class. Further, it also introduces interdependence between these features and adds various types of noise to the data. The numbers of samples were taken as 100(default) and number of features as 20.

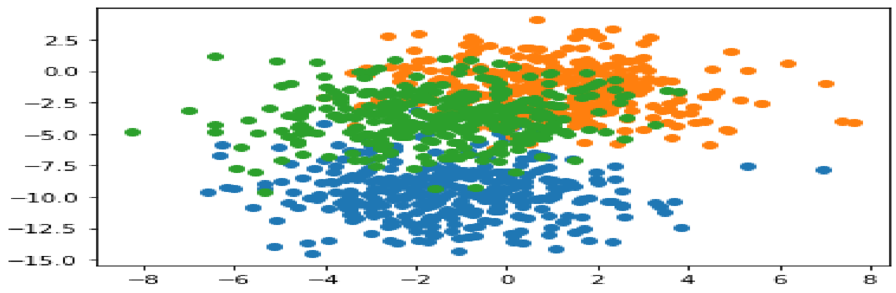


Figure. 6 Sample output showing 3 different classes

```
pyplot.savefig('sgd.png')
pyplot.show()
```

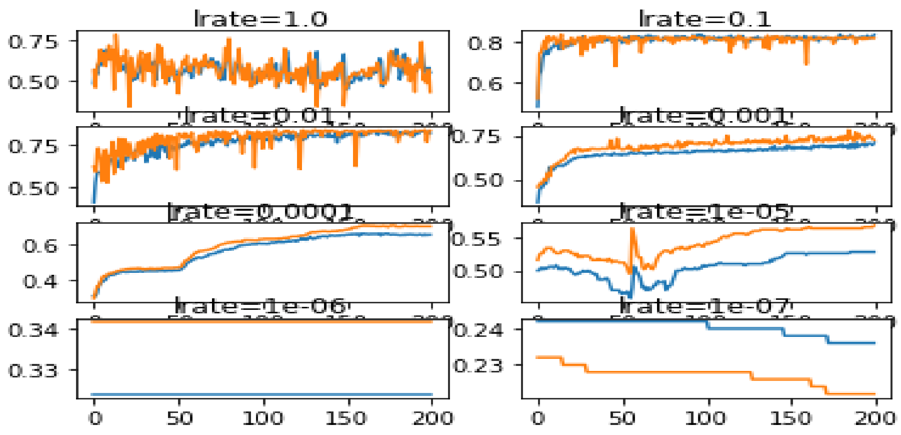


Figure. 7(a) SGD with '***lrate***' and accuracy with number of epochs

```
    pyplot.savefig('adam.png')
    pyplot.show()
```

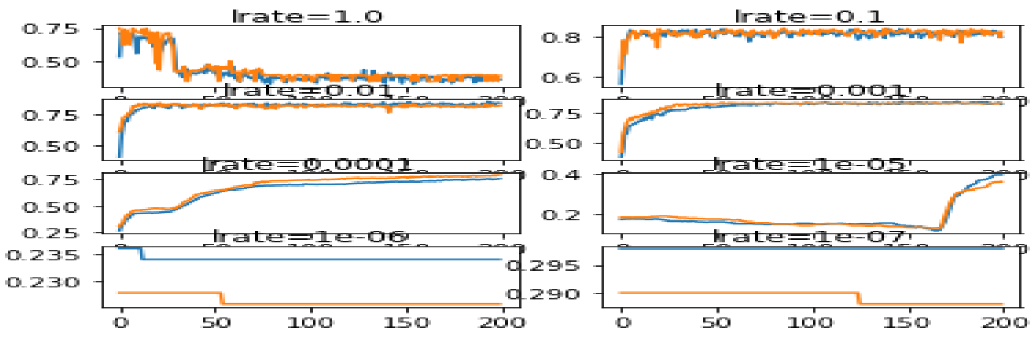


Figure.7(b) ADAM with '*lrate*' and accuracy with number of epochs

```
  pyplot.savefig('adagrad.png')
  pyplot.show()
```

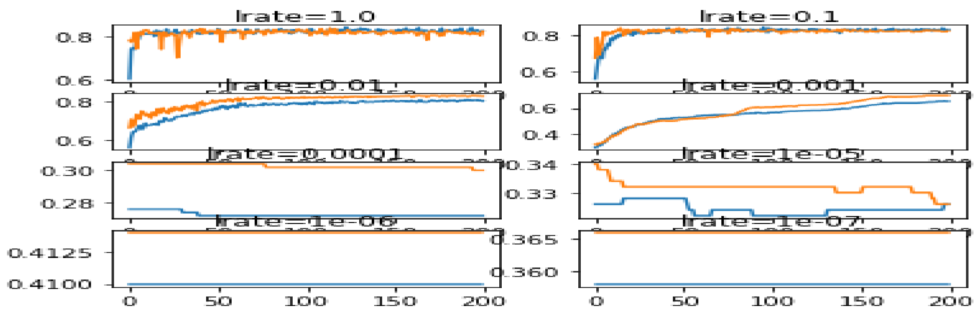
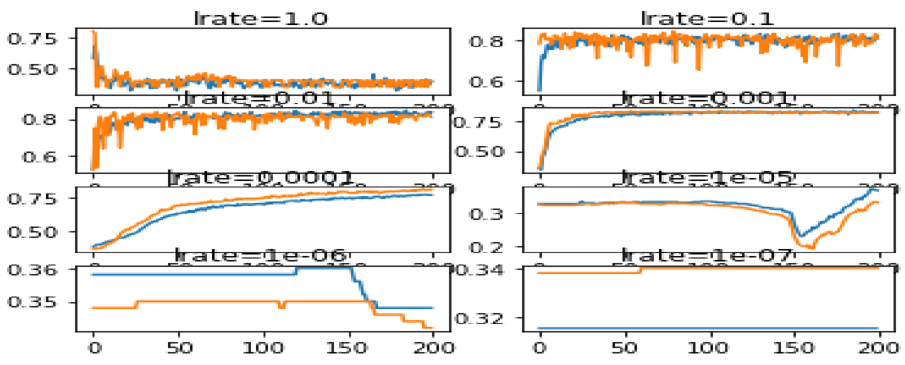


Figure. 7(c) AdaGrad with '*lrate*' and accuracy with number of epochs

```
  pyplot.savefig('rmsprop.png')
  pyplot.show()
```



.

Figure 7(d) RMSProp with '***lrate***' and a ccuracy with number of epochs

```
pyplot.savefig('amsgrad.png')
pyplot.show()
```

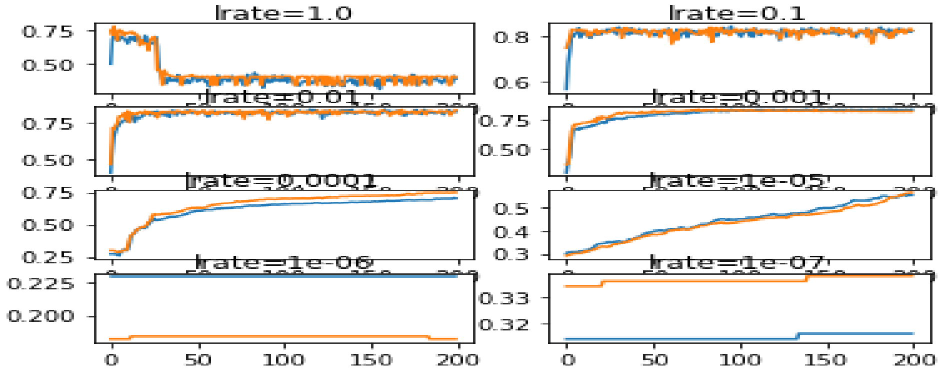


Figure. 7(e) AMSGrad with '*lrate*' and accuracy with number of epochs

B. Resource:

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.

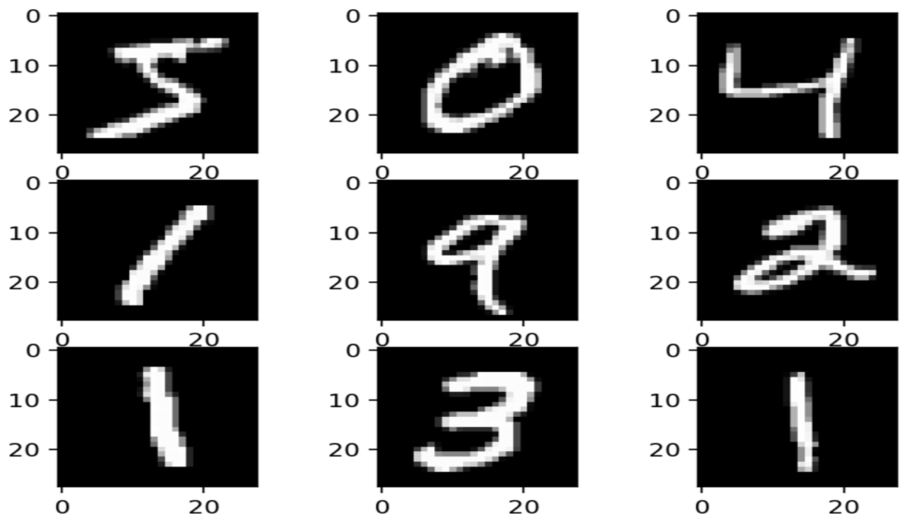


Figure. 8 MNIST Hand-writing data

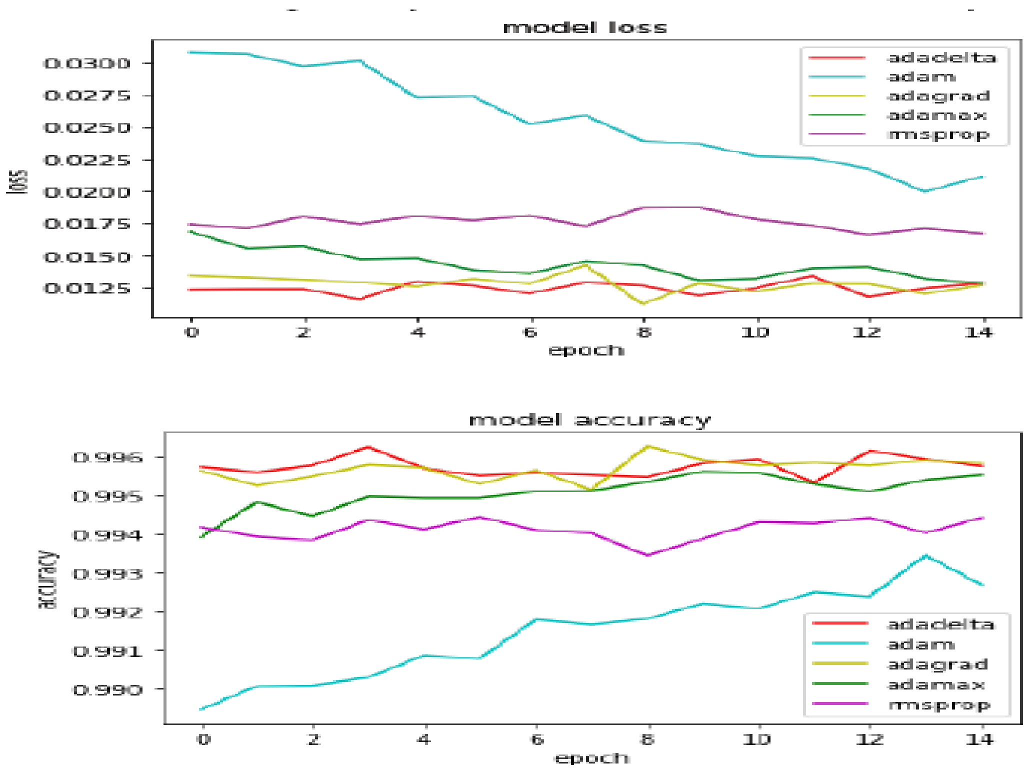


Fig. 9 (a, b) Different Optimizers with Loss and Accuracy

6. Conclusion

The preceding narrations and the pertinent graphs with different optimizers provide their comparison with respect to number of epochs and accuracy and loss as intrinsic parameters. Some emerging points with the experimental analysis are:

— A starting point that usually works well is **ReLU** as activation function
— Decreasing loss provides a way to know the faster convergence rate and it also reflect the learning goals providing intuitive learning idea with approximations

—Begin with a simple dataset having a few samples with a fitting network and after getting good results, increase the complexity of data and network with tuning parameters subsequently

—Start with a well-known optimizer to seek the results and then try with another to get a reconnaissance and it might lead to improvement–not always! The correctly chosen optimizer with adequate learning rate and other parameters makes a good deal. However, there no single *optimizer that solves for all types of date set.*

— *Choice of a Particular Optimizer*: Choose a well known and already experimented optimizer with default learning rates and other parameters settings. Try to run with iterations (epochs) and notice the loss (accuracy) results. Then, shift towards other similar-featured optimizer and observe the changes. *Indeed an exhaustive process*!

Nonetheless, keep experimenting by increasing towards momentum based and

other current blends of optimizers(like Adam, AMSGrad or RADam, etc.).

We have provided with an understanding of reasons for a particular optimizer for a given data set and its give s foundation for the pros and cons of their suitability. 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].

Furthermore, there are several promising results one can obtain from different set(s) of optimizer(s) and this might require more investigations and deeper delving into. The upcoming and already announced optimisers – like YOGI has to be integrated into the DL framework, so that it can be empirical tested against others. The Lottery Ticket technique is already gaining momentum in this area and sooner hopefully more insight shall we be able to explore with.

References

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

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

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

[4] Ajeet K. Jain, Dr. PVRD Prasad Rao and Dr. K Venkatesh Sharma;"*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 2021

[5] John Paul Mueller and Luca Massaron, Deep Learning for Dummies, John Wiley, 2019

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

[7] Ajeet K. Jain, Dr.PVRD Prasad Rao , Dr. K. Venkatesh Sharma, Deep Learning with Recursive Neural Network for Temporal Logic Implementation, International Journal of Advanced Trends in Computer Science and Engineering, Volume 9, No.4, July – August 2020, pp 6829-6833

[8] Srivasatava et al.

http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf

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

[10] Stephen Boyd and Lieven Vandenberghe, Convex Optimization, Cambridge University Press, USA 2004

[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. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580, 2012

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

[14] Glorot, X., Bordes, A., and Bengio, Y, Deep sparse rectifier neural networks.

In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS), pages 315–323. 2011.

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

[16] Prajit Ramachandran, Barret Zoph, Quoc V. Le, SWISH: A Self-Gated Activation Function, arXiv:1710.05941v1 [cs.NE] 16 Oct 2017

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

[18] Kavosh Asadi , Dipendra Misra and Michael L. Littman, Lipschitz Continuity in Model-based Reinforcement Learning, Proceedings of the 35 th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018

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

[20] J. Duchi, E. Hazan and Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research, pp 2121-2159, 2011

[21] Prabhu CSR, Gandhi R, Jain A K, Lalka VS, Thottempudi SG, Prasada Rao PVRD; "A Novel Approach to Extend KM Models with Object Knowledge Model (OKM) and Kafka for Big Data and Semantic Web with Greater Semantics", Advances in Intelligent Systems and Computing 993, pp.544, 2020

[22] Bottou, L. Online algorithms and stochastic approximations. In Saad, D., editor, Online Learning and Neural Networks. Cambridge University Press, Cambridge, 1998

[23] I. Sutskever, J. Martens, G. Dahl and G. Hinton, On importance of initialization and momentum in deep learning, International Conference on Machine Learning, Atlanta, USA, pp. 1139-1147, 2013

[24] Y. Nesterov, A method of solving a convex programming problem with convergence rate O(1/k^2), Soviet Mathematics Doklady, 27, pp 372-376, 1983

[25] J. Duchi, E. Hazan and Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research, pp 2121-2159, 2011

[26] Ajeet K Jain, Dr.PVRD Prasad Rao and Dr.K Venkatesh Sharma;"Extending Description Logics for Semantic Web Ontology Implementation Domains", Test Engineering and Management 83, pp.7385, 2020

[27] G. Hinton, Neural networks for machine learning, Coursera, video lectures, 2018

[28] D. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980, 2014.

[29] S.J. Reddi, S. Kale and S. Kumar, On the convergence of Adam and beyond, International Conference on Learning Representations, Vancouver, Canada, 2018.

[30] Zaheer, M., Reddi, S., Sachan, D., Kale, S., & Kumar, S. Adaptive methods for nonconvex optimization. Advances in Neural Information Processing Systems (pp. 9793–9803), 2018

[31] Londhe, A., Prasada Rao, P.V.R.D. "*Platforms for big data analytics: Trend towards hybrid era" International Conference on Energy, Communication, Data Analytics and Soft Computing*, ICECDS 2017 DOI: 10.1109/ICECDS.2017.8390056

[32] Hiroaki Hayashi, Jayanth Koushik and Graham Neubig ; Eve: A Gradient Based Optimization Method with Locally and Globally Adaptive Learning Rates , arXiv:1611.01505v3 [cs.LG] 11 Jun 2018

[33] Liyuan Liu , et al., On The Variance Of The Adaptive Learning Rate And Beyond, arXiv:1908.03265v3 [cs.LG] 17 Apr 2020

[34] https://d2l.ai/chapter_optimization/lr-scheduler.html

[35] Nicola Landro, Ignazio Gallo, Riccardo La Grassa, Mixing ADAM and SGD: a Combined Optimization Method, arXiv:2011.08042v1 [cs.LG] 16 Nov 2020

[36] Jonathan Frankle and Michael Carbin, The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, arXiv:1803.03635v5 [cs.LG] 4 Mar 2019

[37] Yadla, H.K., Rao, P.V.R.D.P. "Machine learning based text classifier centered on TF-IDF vectoriser, International Journal of Scientific and Technology Research, 2020

[38]Varakumari, S., Prasad Rao, P.V.R.D., Sirisha, M., Mohan Rao, K.R.R. MANOVA- A multivariate statistical variance analysis for WSN using PCA 2018 International Journal of Engineering and Technology(UAE) 7, 2018

[39]Phani Madhuri, N., Meghana, A., Prasada Rao, P.V.R.D., Prem Kumar, P."Ailment prognosis and propose antidote for skin using deep learning", International Journal of Innovative Technology and Exploring Engineering, 2019