

Experimental Comparison of Stochastic Optimizers in Deep Learning

Emmanuel Okewu[1(✉)], Philip Adewole[2], and Oladipupo Sennaike[2]

[1] Centre for Information Technology and Systems,
University of Lagos, Lagos, Nigeria
eokewu@unilag.edu.ng
[2] Department of Computer Sciences, University of Lagos, Lagos, Nigeria
{padewole,osennaike}@unilag.edu.ng

Abstract. The stochastic optimization problem in deep learning involves finding optimal values of loss function and neural network parameters using a meta-heuristic search algorithm. The fact that these values cannot be reasonably obtained by using a deterministic optimization technique underscores the need for an iterative method that randomly picks data segments, arbitrarily determines initial values of optimization (network) parameters and steadily computes series of error functions until a tolerable error is attained. The typical stochastic optimization algorithm for training deep neural networks as a non-convex optimization problem is gradient descent. It has existing extensions like Stochastic Gradient Descent, Adagrad, Adadelta, RMSProp and Adam. In terms of accuracy, convergence rate and training time, each of these stochastic optimizers represents an improvement. However, there is room for further improvement. This paper presents outcomes of series of experiments conducted with a view to providing empirical evidences of successes made so far. We used Python deep learning libaries (Tensorflow and Keras API) for our experiments. Each algorithm is executed, results collated, and a case made for further research in deep learning to improve training time and convergence rate of deep neural network, as well as accuracy of outcomes. This is in response to the growing demands for deep learning in mission-critical and highly sophisticated decision making processes across industry verticals.

Keywords: Deep learning · Deep neural networks · Error function · Neural network parameters · Stochastic optimization

1 Introduction

The training of deep learning models remains a challenge despite many feats in recent memory [1]. The training is a stochastic optimization problem that is non-convex (i.e. global minimum differs from local minimum), high-dimensional, and complex. Presently, the most solicited approach is the use of stochastic gradient descent and its variants [2]. In a bid to handle specific issues in the training of deep learning models, several extensions to the basic (standard) gradient descent algorithm have been proposed. In Sect. 2, some of these methods are reviewed.

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11623, pp. 704–715, 2019.
https://doi.org/10.1007/978-3-030-24308-1_55

Despite the fact that variants and extensions of stochastic gradient descent perform well empirically, further improvement is possible [3]. This is evidenced by the emergence of several methods that simplify the optimization problem such as backpropagation of error, weight updating, normalization methods and weight initialization techniques.

Pattern recognition (classification) using deep neural networks commences with an activation function that introduces non-linearity into the linear data as follows:

$$\text{Neuron output } (\hat{y}) = x_1.w_1 + x_2.w_2 + x_3.w_3 + \ldots + x_n.w_n + b \quad (1)$$

where x_1, x_2,…, x_n are input variables
w_1, w_2, …w_n are network parameters
b is a network constant called bias.

This neural network learning process continues with the computation of an objective (cost) function as follows:

$$\delta = y_{target} - \hat{y}$$

Since it is unlikely that the network (neuron) output ($\hat{y}$) will equal target output (y_{target}), it means $\delta \neq 0$. It implies an error in the network and attempt is made to exactly match the real-world data. The neural network then starts an iterative stochastic process that progressively reduces the error until it is within acceptable limit. This process is a non-deterministic optimization process that requires a search algorithm otherwise referred to as a stochastic optimization algorithm. At the end of the iterative process, the deep neural network is assumed to have learnt sufficiently the patterns in the historical data.

SGD decides which direction to move each network parameter and by how much to move each parameter [4]. This is achieved by computing the gradient of the loss (error) with respect to each parameter. Each gradient is scaled by the learning rate. Parameter update is performed using the gradient step:

$$w = w - \eta \cdot \nabla F(w;x(i);y(i)) \quad (2)$$

where {x(i), y(i)} are training examples, η is learning rate, ∇F is gradient.

Each update is referred to as a gradient step and the process is known as gradient descent. As indicated in Eq. **2**, step size (learning rate) is set by estimating curvature (gradient) from first order information. Given m passes over sample data and n neural network parameters (w_1, w_2, …, w_n) with cost functions (f_1, f_2, … f_m), the gradient is a matrix of first order partial derivatives ($\partial f/\partial w$) given as:

$$\frac{\partial f}{\partial w} = \begin{bmatrix} \frac{\partial f1}{\partial w1} \frac{\partial f1}{\partial w2} \cdots \frac{\partial f1}{\partial wn} \\ \frac{\partial f2}{\partial w1} \frac{\partial f2}{\partial w2} \cdots \frac{\partial f2}{\partial wn} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\partial fn}{\partial w1} \frac{\partial fn}{\partial w2} \cdots \frac{\partial fn}{\partial wn} \end{bmatrix}$$

At the point where $\partial f/\partial w = 0$, the deep neural network has sufficiently learnt patterns in the data.

The manner in which data is fetched from the training dataset and learning conducted has given rise to three variants of gradient descent – standard (batch) gradient descent, stochastic gradient descent, and mini-batch gradient descent as discussed in Sect. 2. The generic version is stochastic gradient descent (SGD) [5].

The term stochastic refers to the process of random selection of training instances {x(i); y(i)} and random initialization of network parameters [4]. Since the error is computed based on randomly selected data and randomly initialized network parameters, the error function is a stochastic objective function.

Stochastic optimization involves taking a number of gradient steps such that the model secures very low loss on training data as a condition for generalization [6].

2 Background and Related Work

2.1 Deep Learning

Deep learning is a subset of machine learning that makes use of deep neural networks. It is based on a set of algorithms used to model high level abstractions in data [7]. It can also be seen as a computational software that imitates the network of neurons in a brain. Deep learning models are constructed with connected layers which include input layer, output layer and hidden layers [1]. Each hidden layer is composed of neurons. The connected neurons process and propagate the input signal they receive to the next layer. The determinants of the strength of the signal are the weight, bias and activation function. The network takes in large amounts of input data, process them through multiple layers as it learns increasingly complex features of the data at each layer. From object detection to speech recognition, deep neural networks offer accuracy as they learn from data directly without coding of any predefined knowledge by programmers. The machine (deep) learning process involves understanding the problem, identifying data, selecting appropriate deep learning model, training of model, and finally, testing of the model.

Each layer in a neural network represents a deeper level of knowledge which suggests the presence of a hierarchy of knowledge. Hence, a neural network with four layers learns more complex and complicated features than one with two layers. The learning process takes place in two phases: the first phase involves the application of a nonlinear transformation of the input so as to create a statistical model as output. The second phase targets the improvement of the model using a mathematical method called derivative. These two phases are repeated many times until a tolerable level of accuracy is reached. This repetition is called an iteration. Typical deep neural networks are convolutional neural networks and recurrent neural networks [8].

The popular algorithm for training deep neural networks is the first order method called gradient descent [9]. The use of second order methods like Newton's method for deep neural networks (DNN) is discouraged in that they are computationally intensive. Moreover, DNN problems are non-convex high dimensional problems for which second order methods are unsuitable. Non-convex problems are problems whose local minimum differs from global minimum [10].

The optimization focus of DNN is to minimize the objective function with respect to (w.r.t) the network parameters and SGD algorithm is used. SGD is a stochastic approximation of gradient descent optimization [11]. It is an iterative method that optimizes the differentiable objective function. It is called stochastic for two reasons: training samples are chosen randomly or in a shuffled manner and network parameters are randomly determined or initialized. In a nutshell, it means involving chance or probability.

2.2 Gradient Descent Algorithm and Optimization in Deep Neural Networks

Gradient descent algorithm is an optimization algorithm that helps to refine machine learning operations. In artificial neural networks generally and deep learning in particular, its focus is to adjust the input weights of neurons towards finding local minima or global minima in order to optimize a problem. In deep neural networks, the algorithm minimizes the objective function parameterized by the network parameters. It searches for the values of the parameters (coefficients) of a function (f) that minimize the cost function (cost). This search algorithm is best used when it is difficult to calculate the parameters using analytical tools like linear algebra. The strategy of the search and optimization algorithm is to try different coefficient (parameter) values, evaluate their cost, and select new coefficients with better (lower) cost. The repetition of this process for a number of times will lead to the local minimum as shown in Fig. 1 and the values of the coefficients that culminated in this minimum cost would be ascertained. Deep learning problems are non-convex in that local minimum differs from global minimum [10, 11].

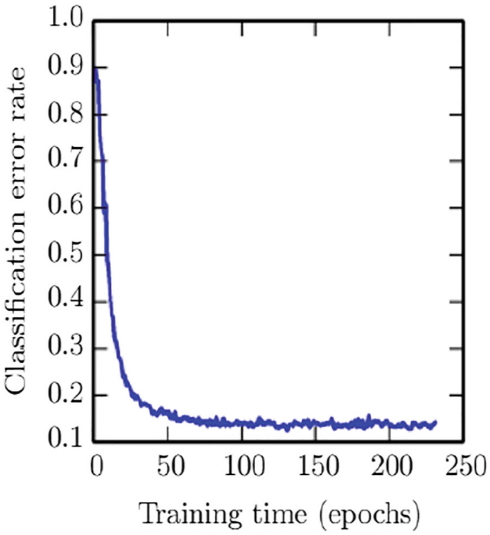

Fig. 1. Length of time to reach local minimum measured in epochs (Source: Dabura 2017)

The length of time it takes an algorithm to reach the local minimum in deep neural networks is a measure of the efficacy of the gradient descent algorithm. As a result, various improvements on the basic gradient descent algorithm had focused on tackling one impediment or the other that tends to impede on convergence. In this section, related works covered are optimizers, all of which modify learning rates by estimating curvature (gradient) from first order information. The works underscore efforts by various researchers to make efficient and effective use of gradient descent in deep neural networks and are analyzed under four titles: parameter updates using simple decay rate; adapting parameter updates to slope error function; per-parameter update using individual learning rate; and per-parameter updates using individual momentum changes.

2.3 Related Works

Previous works on stochastic optimizers in deep learning are as follows.

In the work of [5], first-order iterative methods were discussed, with particular focus on standard (batch) gradient descent. Though first order methods can evade saddle points by following directions of negative curvature (gradient), these saddle points are surrounded by plateaus otherwise referred to as regions of small curvature. This accounts for the slowness of first order methods near saddle points which creates illusion of a local minimum.

First order gradient descent is an algorithm suitable for non-convex optimization models like deep neural networks [13, 14]. In training and optimizing such intelligent systems, gradient descent is the most popular algorithm [15]. However, the limitation of the standard (batch) gradient descent is that it computes the gradient of the entire dataset before performing an update. This makes it to be very slow as well as difficult to control for very large datasets and don't fit in the memory. The size of an update (big or small) is determined by the learning rate (η). For convex error surfaces, it is guaranteed to converge to the global minimum while it converges to local minimum for non-convex problems like deep neural networks. Another drawback of standard batch gradient descent is that it calculates redundant updates for huge datasets.

[16] credited [17] for developing stochastic gradient descent (SGD) to tackle the problem of prolonged training time associated with standard gradient descent. Standard gradient descent calculates gradient for all training instances before performing parameter update. SGD is also called incremental gradient descent and it is an iterative method that optimizes differentiable objective function which is seen as stochastic approximation of gradient descent optimization. It is called stochastic in that data instances are selected randomly or shuffled unlike in standard gradient descent where samples are chosen as a single group or better still, in the order they appear in the training set. It performs parameter update for every training example. It is a faster technique than standard gradient descent as it executes one update at a time.

The frequent updates have high variance and stimulate the loss function to fluctuate to different intensities. This is a positive development on one hand, as it aids the discovery of new and possibly better local minima, whereas standard gradient descent only converges to the minimum of the basin. Nevertheless, the setback of SGD is that it complicates convergence to exact minimum and can overshoot the local minimum owing to frequent updates and fluctuations (oscillations). Studies, have however, shown that as the learning rate (η) slowly decreases, SGD exhibits the same convergence paradigm as standard gradient descent.

Mini-batch gradient descent was developed to address the limitations of SGD [5]. When using SGD, the updates for each training dataset instance can give rise to a noisy plot of cost over time. In a bid to rectify these challenges of high variance parameter updates and unstable convergence associated with SGD, mini-batch gradient descent (MGD) emerged as a response and an improvement. It circumvents the issues with standard batch gradient descent and SGD by combining the best of both techniques. MGD breaks down dataset into mini-batches and executes an update for each mini-batch with n training instances. MGD is credited with ability to reduce the variance in the parameter updates which results in much better and stable convergence. It also uses highly optimized matrix optimizations commonly found in state-of-the-art deep learning libraries which makes computation of the gradient w.r.t. a mini-batch very efficient. The flexibility of MGD as mini-batch sizes can range from 50 to 256. Additional credit is that these batch sizes vary as per the application and problem being tackled. It is therefore a good algorithm for training a neural network.

In [18] and [19], proposals for the use of momentum as an optimization algorithm were made. Momentum was designed to tackle the problem posed by high variance oscillations in SGD which mini-batch gradient descent could not handle comprehensively. These oscillations impede SGD from converging on time. The Momentum technique accelerates SGD by navigating along the relevant direction while softening oscillations in irrelevant directions.

[20] developed Nesterov accelerated gradient (NAG) to solve a problem associated with the Momentum technique: as the minima (the lowest point on the curve) is reached, the momentum is very high and the convergence may not slow down at that point owing to high momentum. This potentially could cause the loss function to miss the minima completely and rather continue to move up. To address this problem, NAG algorithm computes the gradient after making a big jump premised on previous momentum. Then after, it makes a correction which culminates in a parameter update. This strategy of anticipatory update prevents the loss function from decreasing too fast so that it does not miss the minima. It equally makes the convergence process to be more responsive to changes.

In [4], a new SGD algorithm called Adagrad that improved on Momentum and Nesterov Accelerated Gradient (NAG) was proposed. Adagrad means adaptive gradient. Sparse gradients constitute a key issue in training deep learning models. To solve this problem, Adagrad algorithm was developed. The algorithm adaptively tunes learning rate for each parameter and in the process performing larger updates for rarely updated parameters. Nevertheless, Adagrad's update rule results in a vanishing (decaying) rate which forces the learning rate to monotonically decrease to a very small amount. A very small learning rate stalls the algorithm.

[21] introduced Adadelta to address the problem of decaying learning rate associated with Adagrad. The algorithm is an extension of Adagrad aimed at eliminating its decaying learning rate problem. Rather than accumulate all previous squared gradients, Adadelta limits the number of accumulated previous gradients to certain constant size w. Since the storage of w previously squared gradients could be inefficiently done, there is a recursive definition of the sum of gradients as a decaying mean of past squared gradients. Hence, the running average at time step t only depends on previous average and current gradient. A further enhancement in Adadelta is the fact that setting a default learning rate is not necessary.

In furtherance of efforts to solve the problem of vanishing learning rates, [22] developed Root Mean Square Propagation (RMSProp). The technique maintains per-parameter learning rates and it is an extension of stochastic gradient descent algorithm that attempts to fix the issue of vanishing (decaying) learning rate. A version of RMSProp optimization algorithm uses momentum [23]. RMSProp with momentum computes parameter updates using momentum on rescaled gradient Bias-correction term is missing in RMSProp.

[11] proposed Adam stochastic optimization algorithm. The method rides on the philosophy that in as much as it is useful to calculate learning rate for each parameter as demonstrated in previous algorithms, then computing individual momentum changes for each parameter and storing them separately would likewise be beneficial. Hence, the emergence of a new modified technique and improvement called Adam. Adam means Adaptive Moment Estimation. It is a popular algorithm for training deep neural

networks and integrates the benefits of Adagrad and RMSProp. It leverages on the moving average of past gradients to ascertain the direction of descent just as it uses the running average of past squared gradients to scale (modify) the learning rate. Adam offers a remarkable improvement on stochastic gradient descent in that it performs well in practice. It converges fast and enhances the learning speed of neural network models. Generally, it favourably compares with other adaptive learning-method algorithms. Nevertheless, there is room for further improvement [3].

3 Methodology

Overtime, research efforts towards improving deep neural networks via stochastic optimization algorithms have yielded some results that are well documented. This study attempts to validate the reported improvement by experimentally evaluating established optimizers like Adam, RMSProp, Adadelta, Adagrad and SGD using Python deep learning environment. High dimensional data suitable for deep learning as well as a suitable deep neural network model were carefully chosen for experiments. The details of the series of experiments and analysis of outcomes are outlined as follows:

To empirically validate the performances of the deep learning optimizers and justify research efforts committed to improving stochastic optimization in deep neural network with respect to accuracy and training time, select optimizers were executed using Python deep learning libraries [12]. The libraries include Tensorflow and Keras API. The data used is the MNIST (Modified National Institute of Standards and Technology) database, a dataset of handwritten digits images [24]. Convolutional Neural Network was used as the deep neural network model [25].

MNIST dataset is partitioned into training data and testing data. The training data comprises 60,000 images while the testing data is made up of 10,000 data. The linear structure of the database is further analyzed as follows:

Training Data
Total examples (images) = 60,000 (images of handwritten digits)
Pixels per image = 28 x 28 = 784
Size (shape) of training (image) data = [60000,784]

Testing Data
Total instances (images) = 10,000
Pixels per image = 28 x 28 = 784
Size (shape) of training data = [10000, 784]

Label Data
Total examples (images) = 60,000
Total values in label = 60,000
Size (shape) of label data = [60,000]

The Python code implemented in turn for each of the select optimizers (Adam, Adagrad, Adadelta, RMSProp, and SGD) is outlined as follows:

```
# Program for image classification using deep learning (convolutional neural
# network) and MNIST database
import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train.shape
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print('Number of images in x_train', x_train.shape[0])
print('Number of images in x_test', x_test.shape[0])
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(10,activation=tf.nn.softmax))
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(x=x_train,y=y_train, epochs=20)
model.evaluate(x_test, y_test)
image_index = 4444
plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')
pred = model.predict(x_test[image_index].reshape(1, img_rows, img_cols, 1))
print(pred.argmax())
```

4 Results and Interpretations

The above Python code was implemented for each of the five deep learning optimizers (Adam, RMProp, Adadelta, Adagrad and Stochastic Gradient Decent), one after the other using 20 iterations. However, due to space constraint in this report, we show the output for only 15 iterations. The three categories of outcome are training time, loss function, and accuracy as shown in Tables 1, 2 and 3 below.

Table 1. Training time

Optimizer	1st iteration	2nd iteration	3rd iteration	4th iteration	5th iteration	6th iteration	7th iteration	8th iteration	9th iteration	10th iteration	11th iteration	12th iteration	13th iteration	14th iteration	15th iteration
SGD	83 s	78 s	95 s	95 s	94 s	94 s	94 s	87 s	87 s	86 s	88 s	103 s	98 s	91 s	88 s
Adagrad	86 s	82 s	103 s	114 s	104 s	104 s	104 s	102 s	113 s	109 s	109 s	110 s	106 s	105 s	106 s
Adadelta	127 s	121 s	145 s	145 s	152 s	138 s	131 s	139 s	135 s	129 s	130 s	133 s	135 s	145 s	139 s
RMSProp	95 s	88 s	109 s	107 s	108 s	107 s	106 s	116 s	115 s	109 s	106 s	108 s	107 s	108 s	110 s
Adam	110 s	106 s	125 s	125 s	125 s	127 s	125 s	125 s	127 s	126 s	123 s	121 s	121 s	115 s	119 s

Table 2. Loss function

Optimizer	1st iteration	2nd iteration	3rd iteration	4th iteration	5th iteration	6th iteration	7th iteration	8th iteration	9th iteration	10th iteration	11th iteration	12th iteration	13th iteration	14th iteration	15th iteration
SGD	0.5094	0.2330	0.1814	0.1534	0.1380	0.1231	0.1144	0.1058	0.0979	0.0929	0.0891	0.0823	0.0799	0.0766	0.0736
Adagrad	0.1858	0.0906	0.0696	0.0576	0.0499	0.0451	0.0403	0.0364	0.0338	0.0308	0.0281	0.0261	0.0249	0.0232	0.0220
Adadelta	0.1998	0.0806	0.0575	0.0436	0.0368	0.0302	0.0253	0.0232	0.0185	0.0171	0.0137	0.0125	0.0119	0.0104	0.0095
RMSProp	0.2057	0.0882	0.0664	0.0586	0.0507	0.0455	0.0426	0.0349	0.0340	0.0302	0.0267	0.0259	0.0247	0.0217	0.0223
Adam	0.2183	0.0891	0.0635	0.0504	0.0389	0.0328	0.0278	0.0248	0.0216	0.0190	0.0181	0.0178	0.0155	0.0141	0.0152

Table 3. Accuracy

Optimizer	1st iteration	2nd iteration	3rd iteration	4th iteration	5th iteration	6th iteration	7th iteration	8th iteration	9th iteration	10th iteration	11th iteration	12th iteration	13th iteration	14th iteration	15th iteration
SGD	0.8565	0.9305	0.9464	0.9544	0.9586	0.9625	0.9658	0.9689	0.9700	0.9713	0.9721	0.9746	0.9756	0.9762	0.9772
Adagrad	0.9448	0.9738	0.9793	0.9830	0.9853	0.9863	0.9881	0.9896	0.9901	0.9912	0.9921	0.9924	0.9930	0.9935	0.9938
Adadelta	0.9408	0.9760	0.9828	0.9863	0.9890	0.9909	0.9924	0.9928	0.9940	0.9948	0.9958	0.9958	0.9964	0.9963	0.9970
RMSProp	0.9378	0.9739	0.9806	0.9835	0.9860	0.9870	0.9887	0.9898	0.9902	0.9915	0.9925	0.9933	0.9936	0.9943	0.9946
Adam	0.9353	0.9726	0.9803	0.9842	0.9875	0.9894	0.9906	0.9913	0.9928	0.9937	0.9940	0.9939	0.9949	0.9952	0.9947

In terms of training time, Table 1 shows that SGD has the best time (with the lowest training time) followed by Adagrad, RMSProp, Adam and Adadelta in that order. However, Table 2 indicates that the lowest value of the loss function at the 15th iteration was obtained by Adadelta (0.0095) followed by Adam (0.0152), Adagrad (0.0220), RMSProp (0.0223), and SGD (0.0736). With respect to accuracy, Table 3 shows that at the 15th iteration, Adadelta has the best accuracy Fig. (0.9970) followed by Adam (0.9947), RMSProp (0.9946), Adagrad (0.9938), and SGD (0.9772).

These experimental outcomes confirm that research efforts to improve on gradient descent algorithms for training deep neural networks have paid off in terms of convergence rate and accuracy. The results also show why Adam is considered in the literature as one of the best optimizers for training deep neural networks [3, 11]. Overall, the figures indicate that Adam is the most consistent in terms of performance across the three metrics: training time, loss function, and accuracy.

Though it is difficult to say that any particular algorithm outperformed others across board (across the three metrics), the outcome gives a ray of hope that future research aimed at improving performance of deep learning models could make deep learning more attractive to various industries as a tool for generating useful and timely information for informed decision making.

5 Conclusion and Future Work

The series of experiments in this study were conducted using select deep neural network optimizers like stochastic gradient descent, Adagrad, Adadelta, RMSProp and Adam. We used the MNIST database of images of handwritten digits with a total of 60,000 training data and 10,000 testing data on a Python platform, using Python deep learning libraries (Tensorflow and Keras API). The outcome showed remarkable improvement in both accuracy of output and convergence rate of loss function. In any case, as observed by [3], there is room for further improvement. Researchers are therefore encouraged to invest more time and efforts in deep learning. This is particularly so given the increasing relevance of deep learning in various industry verticals. Therefore, in future research work, further efforts should be made towards resolving open issues of deep learning. These challenges include sparse gradient, variance (oscillations) in parameter updates, vanishing (decaying) learning rate, and presence of poor local optima.

References

1. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. DIRO, Université de Montréal, Montréal (2010)
2. Bengio, Y., LeCun, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539. Bibcode:2015Natur.521..436L. PMID 26017442
3. Koushik, J., Hayashi, H.: Improving stochastic gradient descent with feedback. In: Conference Paper at ICLR 2017

4. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**(Jul), 2121–2159 (2011)
5. Kim, D., Fessler, J.A.: Optimized first-order methods for smooth convex minimization. Math. Prog. **151**, 8–107 (2016)
6. Walia, A.S.: Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent (2017)
7. Shridhar, K.: A Beginners Guide to Deep Learning (2017)
8. Tran, D.T., Iosifidis, A., Gabbouj, M.: Improving efficiency in convolutional neural networks with multilinear filters. Neural Netw. **105**, 328–339 (2018)
9. Li, J., Zhou, T., Wang, C.: On global convergence of gradient descent algorithms for generalized phase retrieval problem. J. Comput. Appl. Math. **329**, 202–222 (2018)
10. Anandkumar, A.: Nonconvex optimization: challenges and recent successes. In: ICML 2016 Tutorial
11. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. In: Published as a Conference Paper at ICLR 2015
12. Brownlee, J.: How to setup a python environment for machine learning and deep learning with anaconda. In: Python Machine Learning (2017)
13. Li, G., et al.: Training deep neural networks with discrete state transition. Neurocomputing **272**, 154–162 (2018)
14. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015). arXiv:1404.7828 . https://doi.org/10.1016/j.neunet.2014.09.003. PMID 25462637
15. Sutskever, I.: Training recurrent neural networks (PDF). Ph.D., University of Toronto, p. 74 (2013)
16. Mei, S.: A mean field view of the landscape of two-layer neural networks. In: Proceedings of the National Academy of Sciences (2018)
17. Robbins, H., Monro, S.: For developing SGD in their 1951 article titled "A Stochastic Approximation Method" (1951)
18. Sutskever, I., Martens, J., Dahl, G., Hinton, G.E.: On the importance of initialization and momentum in deep learning'(PDF). In: Dasgupta, S., Mcallester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning (ICML-13), Atlanta, GA, vol. 28, pp. 1139–1147. Accessed 14 Jan 2016
19. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**(6088), 533–536 (1986)
20. Nesterov (1983)
21. Zeiler, M.D.: Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)
22. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. COURSERA: Neural Netw. Mach. Learn. **4**(2), 26–31 (2012)
23. Graves, A.: Generating Sequences with Recurrent Neural Networks (2014)
24. Yalçın, O.G.: Image Classification in 10 Minutes with MNIST Dataset (2018)
25. Torres, J.: Convolutional Neural Networks for Beginners. Practical Guide with Python and Keras (2018)