International Journal for Research in Engineering Application & Management (IJREAM)
ISSN : 2454-9150 Vol-05, Issue-02, May 2019

A Survey of Optimization Techniques for Deep Learning Networks

[*]Agnes Lydia, [#]F. Sagayaraj Francis

[*]Research Scholar, [#]Professor, Pondicherry Engineering College, Pondicherry-India,

[*]agneslydia@pec.edu, [#]fsfrancis@pec.edu

Abstract: Deep Learning is a sub-field of Machine Learning, in which a set of training algorithms and optimization techniques work together to perform a particular task. Various Deep Learning Networks are available to perform different kinds of tasks. The state-of-art Neural Network in Deep Learning that addresses Computer Vision tasks is the Convolutional Neural Network (CNN). The Convolutional Neural Network is usually trained with Backpropagation algorithm. With the availability of large scale image datasets and the limitation on computational memory, the existing optimization techniques are not able to perform efficiently. The Gradient Descent being the recurrently used optimization technique for Backpropagation algorithm, several variants have been developed to improvise its performance. In addition, fine tuning the hyperparameters of the training algorithm, further gives the flexibility to enhance the training process as per the requirements of the model. This paper provides an insight about the various optimization techniques and methods to modify hyperparameters to get better results on any image datasets.

Keywords — Neural Network, Deep Learning, Classification of Optimization Techniques, Hyperparameters, Loss Function, Gradient Descent, Backpropagation

I. INTRODUCTION

A Neural Network is an artificial representation of human nervous system, hence the name *Artificial Neural Network* (ANN) [1]. The nervous system consists of millions of nerve cells called the *Neurons*. The Neurons are distributed in hierarchical layers. Each neuron takes inputs from numerous other neurons in the preceding layer and performs the required processing on the input. The output, thus obtained, is passed on to numerous other neurons in the following layer. Deep Learning Networks are the trending network architectures that outperform the existing architectures in different tasks like, Speech Recognition, Object Detection, Pattern Recognition, Image Retrieval, etc. There are different Deep Neural Networks available for different tasks like, Convolutional Neural Nework(CNN)[2] for Computer Vision, Recurrent Neural Network(RNN)[3] for Time Series Analysis, Self-Organizing Maps(SOM)[4] for Feature Detection, Deep-Boltzmann Machines(DBM)[5] and Auto-Encoders(AE)[6] for Recommendation Systems. The goal of any Machine Learning task is to reduce the difference between the expected output and the actual output. This is called as the Loss function or the Cost Function.

II. BACKPROPAGATION

Backpropagation is the commonly used training algorithm for Deep Neural Networks. Though this algorithm was introduced in 1970's, it gained its importance in late 1980's after being implemented by David Rumelhart et al [7] which proves to work much faster than the existing algorithms to train neural networks. The role of Backpropagation is to compute the partial derivative of the cost function with respect to weights and biases of each neuron. The Backpropagation algorithm [8] works in two phases.

- *Forward Propagation:* The algorithm propagates from the input layer to the output layer, and computes the output by feeding the input weights and biases into an activation function at every layer. The value obtained at the output layer is compared with the expect values, and the difference is calculated (Total Error, E).

$$E = \frac{1}{2}(y - f(\sum w_i x_i))^2 \qquad (1)$$

- *Backward Propagation:* This Total Error Value is subtracted from the current weights of every neuron and propagated backward from the output layer to the input layer. The network again trains with the updated weights.

The role of an *Optimizer* is to update the weights of the neurons in such a way so that it minimizes the loss function. The Loss function acts as a guide to direct the optimizer in the right direction towards to the global minimum.

III. CLASSIFICATION OF OPTIMIZATION ALGORITHMS

Optimization algorithms [9] are utilized to minimize or maximize the objective function (cost function), *f(x)*, which is dependent on the model's internal learnable parameters, used in computing the *target values(Y)* from the set of *predictors (X)*. The internal learnable parameters in a neural network are *Weights (W)* and *Biases (b)*. These parameters are learned and updated in the direction of optimal solution. The optimization algorithms are classified into two categories,

 DOI : 10.35291/2454-9150.2019.0100

A. First-Order Optimization Algorithms

First-Order Optimization algorithms [10] minimize or maximize the objective function using its Gradient values with respect to its parameters. The first-order derivative tells us whether the function is increasing or decreasing at a particular point. The most widely used first-order optimization algorithm is the *Gradient Descent*. A Gradient is a multi-variable generalization of a derivative *(dy=dx)*, i.e., rate of change of *y* with respect to *x*. The Gradient points towards the steepest descending direction of the optimal solution. The difference between a regular derivative and a Gradient is that, the derivative is calculated for a function dependent on one variable, where as a Gradient is the partial-derivative calculated for a function dependant on multiple variables. The output of a Gradient is a *vector*. The Gradient is represented by a *Jacobian Matrix*, which consists of first-order partial derivatives.

B. Second-Order Optimization Algorithms

Second-Order Optimization algorithms [11] use second-order derivatives also known as *Hessian Matrix* to minimize or maximize the objective function [12]. Hessian is a matrix of second-order partial derivatives. The second-order derivatives tell us whether the first derivative is increasing or decreasing, which in turn helps to find the function's curvature. Since the computation of Second-order derivatives is expensive, it is not widely used. Though it's expensive, it has its own advantages. Second-order derivatives provide better performances in step-wise terms, and this algorithm does not neglect the curvature of the surface, which plays a major role in optimization.

IV. Gradient Descent

A. Batch-Gradient Descent

It is the most important technique used to train and optimize intelligent systems. It is majorly used to update the weights of neurons in a neural network. The network thus trained, is further optimized using Backpropagation techniques. The input fed into the network, first propagates forward calculating the dot product of input signals and their corresponding weights. Then apply activation function to those sum products, which transforms the input signal to an output signal. This process introduces non-linearities to the model which enables the model to learn non-arbitrary functional mappings. The model then propagates backwards through the networks carrying the error terms, while updating the weights using Gradient Descent in the opposite direction of the curvature.

$$\theta = \theta - \eta \bullet \nabla J(\theta) \qquad (2)$$

is the formula for parameter updates, where η is the learning rate, $\nabla J(\theta)$ is the gradient of the loss function $J(\theta)$ with respect to the parameter θ. The parameter, learning rate decides the value of the weight updates. The Learning rate converges to a global minimum for convex surfaces and to a local minimum for non-convex surfaces. The Batch-Gradient Descent[13] also known as the Vanilla Gradient-Descent, calculates gradient of the cost function with respect to the parameters weights and biases for the entire training dataset. This makes it complicated to handle very huge datasets, slower computations and cannot fit in allocated memory. It also calculates redundant updates for large datasets.

B. Stochastic Gradient Decent.

Stochastic-Gradient Descent (SGD) [14] in contrast performs a parameter update for each training example. To overcome the drawbacks in Batch-Gradient Descent, Stochastic-Gradient Descent has evolved. The challenge in using Stochastic-Gradient Descent is choosing a proper learning rate, to avoid fluctuations while converging to the optimum point.

$$\theta = \theta - \eta \bullet \nabla J(\theta; x(i); y(i)) \qquad (3)$$

where $x(i)$ and $y(i)$ are training examples. This algorithm does not perform well when the curve has saddle points i.e., points where one dimension slopes up and the other dimension slopes down.

V. Optimizing the Gradient Descent

There are several optimization techniques [15] that are incorporated to overcome the challenges in implementing the Gradient-Descent, such as

A. Momentum

Ravines are areas in a slope, where the surface curvature is steeper in one dimension than another. These ravines are common around local optima. While oscillating in these ravines, SGD makes very slow progress to obtain the local optimum position. To handle this slow progress, SGD adds up a parameter called Momentum (usually = 0:9), which accelerates the convergence in the relevant direction.

$$V(t) = \gamma \bullet V(t-1) + \eta \bullet \nabla J(\theta) \qquad (4)$$

where γ is momentum, $V(t-1)$ is the previous step update vector, $\eta \bullet \nabla J(\theta)$ is the current step update vector. Then finally the parameter is updated by,

$$\theta = \theta - V(t) \qquad (5)$$

The Momentum value increases for the dimensions whose gradients are in same direction and decreases for gradients in different directions.

B. Nesterov Accelerated Gradient (NAG)

In Nesterov Accelerated Gradient [16], first the gradient with momentum is calculated for the current position, and then makes a leap in the direction of the accumulated gradient, and calculates the value for the new position. Gradient is calculated for smaller steps either beyond the current position, or reduced to lesser values than the current position. This anticipatory updates prevents SGD from converging too fast, and results in better and results.

$$V(t) = \gamma \bullet V(t-1) + \eta \bullet \nabla J(\theta - \gamma \bullet V(t-1)) \qquad (6)$$

and then update the parameters using

$$\theta = \theta - V(t) \qquad (7)$$

C. Adagrad

The Adagrad optimization technique [17] adapts smaller learning rates for features occurring frequently and adapts higher learning rates for infrequent features. This unique capacity of this technique makes it suitable to handle sparse data.

 DOI : 10.35291/2454-9150.2019.0100

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i}} + \varepsilon} \bullet g_{t,i} \qquad (8)$$

where $g_{t,i}$ is the gradient of the loss function with respect to the parameter $\theta(i)$ at time step t. The main advantage of using Adagrad is the learning rate need not be manually changed, setting the initial learning rate to 0.01 makes the algorithm adapt on its own. The disadvantage in Adagrad is the accumulation of the squared gradients in the denominator, which in turn makes the learning rate infinitesimally small.

D. *Adadelta*

Adadelta[18] is an extension of Adagrad, that seeks to rectify the weakness in Adagrad. Instead accumulating all the past squared gradients, Adadelta restricts window of accumulating the gradients to a fixed value, *w*. The sum of gradients is recursively defined as a decaying average of all past squared gradients.

$$E[g^2](t) = \gamma \bullet E[g^2](t-1) + (1-\gamma) \bullet g^2(t) \qquad (9)$$

where $E[g^2](t)$ is the running average at time step t that depends only on the previous average and the current gradient. The parameter is updated as

$$\Delta\theta(t) = -\eta \bullet g(t,i) \qquad (10)$$

$$\theta(t+1) = \theta(t) + \Delta\theta(t) \qquad (11)$$

With Adadelta it is not necessary to set the learning rate.

E. *RMSProp*

RMSprop[19] is adaptive learning rate method, proposed in the similar time of Adadelta. This was also developed to overcome the radically diminishing problem of Adagrad.

F. *Adam*

Adaptive Moment Estimation (ADAM)[20], computes adaptive learning rates for each parameter. Adam stores the decaying average of past squared gradients, like Adadelta and RMSprop, and also stores the decaying average of past gradients, like Momentum.

$$m_t = \frac{m_t}{1-\beta_1^t} \qquad (12)$$

$$v_t = \frac{v_t}{1-\beta_2^t} \qquad (13)$$

m_t, is the value of the first momentum i.e., the Mean and v_t, is the value of the second momentum i.e., the Variance. Then the parameter is updated by

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_t} + \varepsilon} m_t \qquad (14)$$

The preferable values for β_1 is 0.9, β_2 is 0.999 and $\varepsilon = 10^{-8}$.

G. *Nadam*

Adam is combination of RMSprop and Momentum. Nesterov-Accelerated Gradient (NAG) is superior to Momentum. Nesterov-Accelerated Adaptive Momentum Estimation (NADAM) [21] combines the capabilities of both Adam and NAG.

VI. CONCLUSION

The rapidly growing data in the present, leads to the need for the rise of efficient algorithms to handle it. Existing algorithms are enhanced to cope up with these voluminous data with various optimization techniques. A brief introduction about the most used Gradient Descent algorithm and how the algorithm is enhanced with the addition of new parameters is discussed in Section IV and Section V. This survey intends to motivate several researches by providing knowledge on the details about neural networks, its architecture, training algorithms and the optimization techniques.

REFERENCES

[1] Mcculloch, Warren S., and Walter Pitts. "A Logical Calculus of the Ideas Immanent In Nervous Activity." The bulletin of mathematical biophysics 5.4 (1943): 115-133.

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[3] Connor, Jerome T., R. Douglas Martin, and Les E. Atlas. "Recurrent neural networks and robust time series prediction." IEEE transactions on neural networks 5.2 (1994): 240-254.

[4] Ultsch, Alfred. "Self-organizing neural networks for visualisation and classification." Information and classification. Springer, Berlin, Heidelberg, 1993. 307-313.

[5] Wang, Hao, Naiyan Wang, and Dit-Yan Yeung. "Collaborative deep learning for recommender systems." Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, 2015.

[6] Wu, Yao, et al. "Collaborative denoising auto-encoders for top-n recommender systems." Proceedings of the Ninth ACM International Conference on Web Search and Data Mining. ACM, 2016.

[7] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." Cognitive modeling 5.3 (1988): 1.

[8] Leung, Henry, and Simon Haykin. "The complex backpropagation algorithm." IEEE Transactions on signal processing 39.9 (1991): 2101-2104.

[9] Cochocki, A., and Rolf Unbehauen. Neural networks for optimization and signal processing. John Wiley & Sons, Inc., 1993.

[10] Sra, Suvrit, Sebastian Nowozin, and Stephen J. Wright, eds. Optimization for machine learning. Mit Press, 2012.

 DOI : 10.35291/2454-9150.2019.0100

[11] Ruffio, E., et al. "Tutorial 2: Zero-order optimization algorithms." Eurotherm School METTI (2011).

[12]Hagan, Martin T., and Mohammad B. Menhaj. "Training feedforward networks with the Marquardt algorithm." IEEE transactions on Neural Networks 5.6 (1994): 989-993.

[13] Cotter, Andrew, et al. "Better mini-batch algorithms via accelerated gradient methods." Advances in neural information processing systems. 2011.

[14] Bottou, Léon. "Large-scale machine learning with stochastic gradient descent." Proceedings of COMPSTAT' 2010. Physica-Verlag HD, 2010. 177-186.

[15] Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv : 1609. 04747 (2016).

[16] Dozat, Timothy. "Incorporating nesterov momentum into Adam." (2016).

[17] Ward, Rachel, Xiaoxia Wu, and Leon Bottou. "Adagrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization." arXiv preprint arXiv:1806.01811 (2018).

[18] Zeiler, Matthew D. "ADADELTA: An Adaptive learning rate method." arXiv preprint arXiv : 1212. 5701 (2012).

[19] Basu, Amitabh, et al. "Convergence guarantees for RMSprop and Adam in non-convex optimization and their comparison to Nesterov Acceleration on Autoencoders." arXiv preprint arXiv:1807.06766 (2018).

[20] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980(2014).

[21] Kim, Jihyun, and Howon Kim. "An effective intrusion detection classifier using long short-term memory with gradient descent optimization." 2017 International Conference on Platform Technology and Service (PlatCon). IEEE, 2017.

 DOI : 10.35291/2454-9150.2019.0100