International Journal of Latest Engineering Research and Applications (IJLERA) ISSN: 2455-7137

Volume – 03, Issue – 04, April 2018, PP – 69-72

Optimization Algorithms for Deep learning

Madhuri Yadav
(USIC&T, GGSIPU, India)

Abstract: Deep learning is a machine learning technique that enables computers to solve complicated problems by dividing them into simpler ones using experience and data. Deep learning has solved various complex problems with increasing accuracy over time. It has become more useful as amount of data has increased and hardware cost has decreased. Although, deep learning concept is a great success, but there are challenges which are yet to overcome. There are millions of parameters and hyper-parameters which are needed in deep learning. All these parameters are needed to be tuned and optimized for better and faster results. This paper explains various optimization algorithms for learning in deep networks.

Keywords: ADAM, Convolutional neural network, Deep learning, Gradient descent, Optimization.

I. INTRODUCTION

Deep learning has been proved useful in many pattern recognition problems including computer vision, speech recognition, natural language processing, robotics, video games, and finance. In recent years deep convolutional networks have shown their strength in numerous pattern recognition problems. Some remarkable achievements have been recently obtained in object detection [1], facial expression recognition [2], and scene classification [3]. This technology is used in various commercial applications such as content filtering, e-commerce websites and image classifiers. Convolutional neural networks (CNN) are the important type of networks based on deep learning, feed forward artificial neural networks. A convolutional network comprises of convolutional layers and pooling layers followed by one or more fully connected layers, like in artificial neural networks. The architecture of CNN takes 2D image as input and extracts features using different filters. The CNN uses back propagation algorithm to compute gradient with respect to parameters used in the model to use gradient based optimization. A CNN consists of a number of convolutional and sub-sampling layers followed by fully connected layers. The input to a convolutional layer is an image of mxmxr dimensions where mxm is the height and width of the image and r is the number of channels (an RGB image has r=3 and grayscale image has r=1). The convolutional layer will have kxk filters (or kernels) of size n x n x q where nxn is smaller than the dimension of the image and q is the depth of image. The filters are convolved with the image to produce k feature maps or activation maps. Each map is then sub-sampled typically with mean or max pooling over p x p contiguous regions where p is generally chosen to be small such as 2. This is called pooling layer or sub-sampling layer. It reduces the spatial resolution of the feature map. This operation aims to reduce the precision with which the position of the features extracted by the previous layer is encoded in the new map.

In this way, the convolutional network is obtained by repetition of convolutional and pooling layer. Each convolutional layer can have different number of filters and of different sizes. The initial layers extract low level features, whereas the layers towards the output layers recognize contextual elements instead of simple shapes, edges and corners. The concatenation of sets of convolution and sub-sampling layers achieve a high degree of in-variance to geometric transformation of the input. The last fully connected layer receives the set of learned features and outputs the class of given image. The convolutional network learns using back-propagation which uses gradient descent optimization. It tunes parameters such as filters of convolutional layers and other hyper-parameters. To achieve good recognition results, this network experiments with gradient descent methods to calculate optimum synaptic weights between the neurons. The initial value of these weights for the convolutions and for the fully connected layer can be randomly chosen or by using weight initialization techniques given by researchers such as Xavier filler, proposed by Glorot et al [4]. The loss is calculated using a logistic function of the soft-max output (known as SoftmaxWithLoss). The activation function of the neurons is a ReLu (rectified linear unit), defined as f(z)=max(z,0). The ReLu function generally learns much faster in deep architectures [5].

II. OPTIMIZATION ALGORITHMS

In neural networks, the research of the best hyper-parameters is extremely time consuming. One of the most important parameters in gradient descent learning is learning rate. When the learning rate is too high the optimization can diverge, on the contrary, if it is too low the optimization can be slow. To solve these problems some gradient methods have been proposed which are discussed further.

2.1 STOCHASTIC GRADIENT DESCENT (SGD)

These are the most widely used optimization algorithms in convolutional networks proposed by Button [6]. The algorithm implementing SGD is given below:

Algorithm 1: Stochastic Gradient Descent at training iteration *k*.

```
Input: Learning rate $\lambda_k$ , initial parameter Θ.
While stopping criteria do not met
Take a minibatch of m examples from the training set $\{x^{(1)},\ldots..x^{(m)}\}$ with corresponding targets $y^{(i)}$.
Compute gradient estimate:
        $\square \leftarrow \frac{1}{m} \Delta_{\Theta} \sum_i L(f(x(i);\Theta), y(i))$
Apply update: $\Theta \leftarrow \Theta - \lambda g$
End
```

In gradient descent optimization, cost gradient is calculated for entire dataset or training samples for every iteration, whereas in SGD only few training samples are used for parameter updation. If gradient descent optimization is used and the numbers of training samples are large it takes too long for parameter updation, and thus longer time to finally converge to obtain global minima.

Thus, SGD is used which converges much faster. The main property of SGD is that the computation time per update does not grow with number of training samples. So, it converges even if the number of training samples is high, this is the reason it is mostly used in convolutional networks.

Although, it is a popular optimization strategy, learning of SGD can be sometimes slow. SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum. Therefore, a variant of SGD called SGD with momentum was introduced to accelerate learning. Momentum [7] is a method that helps accelerate SGD in the relevant direction and dampens oscillations

Algorithm 2: Stochastic Gradient Descent (SGD) with momentum

```
Input: Learning rate λ , initial parameter Θ, momentum parameter α, initial velocity υ.
While stopping criteria do not met
Take a minibatch of m examples from the training set $\{x^{(1)},\ldots..x^{(m)}\}$ with corresponding targets $y^{(i)}$.
Compute gradient estimate:
        $\square \leftarrow \frac{1}{m} \Delta_{\Theta} \sum_i L(f(x(i);\Theta), y(i))$
Compute velocity update: $\upsilon \leftarrow \alpha\upsilon - \lambda g$.
Apply update: $\Theta \leftarrow \Theta + \upsilon$.
End
```

2.2 ADAPTIVE GRADIENT (ADAGRAD)

Adaptive gradient optimization algorithm [8] (AdaGrad), dynamically incorporate knowledge of the geometry of the data observed in earlier iterations to perform more informative gradient-based learning.

As shown in Algorithm 3, it individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all the historical squared values of the gradient. The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate. The net effect is greater progress in more gently sloped directions of parameter space.

For training deep learning models, the accumulation of squared gradients from the beginning of training can result in excessive decrease in effective learning rate.

Algorithm 3: AdaGrad algorithm

```
Input: Global learning rate λ , initial parameter Θ, small constant δ, gradient accumulation variable κ=0.
While stopping criteria do not met
Take a minibatch of m examples from the training set $\{x^{(1)},\ldots..x^{(m)}\}$ with corresponding targets $y^{(i)}$.
```

```
Compute gradient :
        □ $\leftarrow \frac{1}{m} \Delta_{\Theta} \sum_i L(f(x(i);\theta), y(i))$
Accumulate squared gradient: $\kappa \leftarrow \kappa + g \odot g$
Compute update: $\Delta\Theta \leftarrow -\frac{\lambda}{\delta+\sqrt{\kappa}} \odot g$
Apply update: $\Theta \leftarrow \Theta + \Delta\Theta$
End
```

2.3 RMSPROP

The RMSProp optimization algorithm given by Hinton [9] modifies AdaGrad to perform better in the nonconvex setting by changing the gradient accumulation into an exponentially weighted moving average. AdaGrad is designed to converge rapidly when applied to a convex function. When applied to a nonconvex fnction to train a neural network, the learning trajectory may pass through many different structures and arrive that region that is locally convex. RMSProp uses an exponentially decaying average to discard history from the extreme past so that it can converge rapidly after finding a convex.
It has been shown to be effective and practical optimization algorithm for deep neural networks.

Algorithm 4: RMSProp algorithm

```
Input: Global learning rate $\lambda$ , initial parameter $\Theta$, small constant $\delta$, gradient accumulation variable $\kappa$=0, decay rate $\rho$.
While stopping criteria do not met
Take a minibatch of m examples from the training set $\{x^{(1)},\ldots x^{(m)}\}$ with corresponding targets $y^{(i)}$.
Compute gradient :
        □ $\leftarrow \frac{1}{m} \Delta_{\Theta} \sum_i L(f(x(i);\theta), y(i))$
Accumulate squared gradient: $\kappa \leftarrow \rho\kappa + (1-\rho) g \odot g$
Compute update: $\Delta\Theta \leftarrow -\frac{\lambda}{\delta+\sqrt{\kappa}} \odot g$
Apply update: $\Theta \leftarrow \Theta + \Delta\Theta$
End
```

2.4 ADAM

Adam means adaptive moments. It is also an adaptive learning rate optimization algorithm. In this, momentum is incorporated directly as an estimate of first-order moment of the gradient. It also includes bias correction to the estimates of both first-order and second-order moments to account for their initialization.

Algorithm 5: Adam algorithm

```
Input: Step size $\epsilon$, exponential decay rates $\rho_1$ and $\rho_2$ (default 0.9 and 0.999 respectively), initial parameter $\Theta$, small constant $\delta$, 1st and 2nd moment variables S=0, $\kappa$=0, time step t=0.
While stopping criteria do not met
Take a minibatch of m examples from the training set $\{x^{(1)},\ldots x^{(m)}\}$ with corresponding targets $y^{(i)}$.
Compute gradient :
        □ $\leftarrow \frac{1}{m} \Delta_{\Theta} \sum_i L(f(x(i);\theta), y(i))$
t=t+1

Update biased first moment estimate: $S \leftarrow \rho_1 S + (1-\rho_1) g$
Update biased second moment estimate: $\kappa \leftarrow \rho_2 \kappa + (1-\rho_2) g \odot g$
Correct bias in first moment: $S \leftarrow \frac{S}{1-\rho 1}$
Correct bias in second moment: $\kappa \leftarrow \frac{\kappa}{1-\rho 2}$
Compute update: $\Delta\Theta \leftarrow -\epsilon \frac{S}{\delta+\sqrt{\kappa}} \odot g$
Apply update: $\Theta \leftarrow \Theta + \Delta\Theta$
End
```

III. CONCLUSION

The algorithm for optimizing deep models by adaptive learning rates is presented. No algorithm is best as the choice of algorithm depends upon the user's familiarity with the algorithm and the problem for which algorithm is being used.

REFERENCES

[1] Krizhevsky, A., Sutskever, I., Hinton, G.E, ImageNet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems*, *2*, 2012, 1097-1105.

[2] A.T. Lopes, E. de Aguiar, A.F. De Souza, T. Oliveira-Santos, Facial expression recognition with convolutional neural networks: coping with few data and the training sample order , *Pattern Recognit.*, *61*, 2017, 610-628.

[3] K. Nogueira, O.A. Penatti, J.A.d. Santos, Towards better exploiting convolutional neural networks for remote sensing scene classification, arXiv preprint arXiv:1602.01517, 2016.

[4] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, *Proc. of the International Conference on Artificial Intelligence and Statistics (AISTATS10), Society for Artificial Intelligence and Statistics*, Sardinia, Italy, 2010, 249-246.

[5] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, *Proc. 14th International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, 2011, 315–323.

[6] L. Bottou, Stochastic Gradient Descent Tricks, in Montavon(Ed.), *Neural Networks, Tricks of the Trade, Reloaded* (New York: Springer,2012) 430-445.

[7] Qian, N., On the momentum term in gradient descent learning algorithms, *Neural Networks : The Official Journal of the International Neural Network Society*, 12(1), 1999, 145–151.

[8] Duchi, J., Hazan, E., & Singer, Y, Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, *Journal of Machine Learning Research, 12*, 2011, 2121–2159.

[9] Tieleman T., Hinton G. *Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude COURSERA: Neural Networks for Machine Learning*, 4, 2012.