Going Deeper with Neural Networks

Sridhar Adhikarla

18/09/2018

Abstract

Going Deeper with Neural networks is one of the best way to get better knowledge from the data, but that cannot be done with the traditional network architecture and optimization algorithms. In this paper I am going to discuss some of the most important research papers that introduce us to the algorithms that would allow us to train our networks efficiently and correctly even for very deep neural network architectures.

Introduction

In today’s world we have abundance of data, and we are looking for good ways to gain intelligence from this data. There are a lot of machine learning algorithms to do this but one of the most efficient algorithms is Neural networks. Neural Networks algorithm was inspired by the brain. A neural network is made up of neurons like the brain, that is how it got the name, but it is not even close to how a brain functions.

Neural networks learn to perform tasks from examples without being explicitly programmed with a set of rules. The neural network has layers of neurons interconnected. The first layer is the input layer and the last layer is called the output layer. We don’t see the values for the layers in between, which are called hidden layers. In most of the applications, one of the ways to get better prediction accuracy is to increase the number of hidden layers. This can cause some problems like overfitting if correct optimization algorithm is not used.

The most basic optimization algorithm is Gradient Descent, which cannot be used for deep neural networks as it converges very slowly and can overfit the data. Other algorithms are just a variance of gradient descent, some modifications made to the basic algorithm to allow it to converge faster. In this review I am going to discuss three such papers that were published to help the optimization algorithms to converge faster without overfitting the data even for very deep neural networks. The papers discussed are-

1 – Dropout: a simple way to prevent neural networks from overfitting [1]

2 – Batch Normalization: Accelerating deep network training [2]

3 – ADAM: a method for stochastic optimization [3]

Dropout: Simple way to prevent neural networks from overfitting

Deep neural networks with large number of parameters are very powerful machine learning systems that can learn very complicated relationships between their inputs and outputs. Overfitting is a major problem in such large networks and it is also slow to use, so trying out different network architectures would take a lot of time. Dropout is the technique for dealing with such problems. Dropout prevents the network from being dependent on a specific value too much, making the weights distributed uniformly across the layers of the network.

Dropout is a technique that prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. [4] In this technique during the training of the network a random set of nodes from each layer with their connections are dropped for each training case, this prevents the network from relying on some nodes to get the output and distributes the weights across the layers uniformly. Since random layers are dropped out this gives rise to different network architectures every time. This is the reason the final weights are called an approximation of exponentially many different network architectures. During the test time for the network the units are not dropped instead the weights are scaled down. If a layer had probability p of retaining its units then it is multiplied by p and this scaled down weight is used for predictions during test time. By doing this scaling exponential networks with shared weights can be combined to a single network at test time.

According to the author this technique has lower generalization error on a wide variety of classification problems compared to other regularization methods. Dropping out 20% of the input units and 50% percent of the hidden units was found to be optimal. Although dropout itself gives significant improvements but using dropout with other regularization methods provide a significant boost over just using dropout. The author goes on to talk about a lot of experiments done with this technique and compares the error rate of this technique with the previous ones. According to the data Dropout had significant improvements over the old methods in a wide variety of application domains like object detection, digit recognition, speech recognition, document classification, analysis of biological data.

In this paper the results of applying Dropout on one of the famous datasets “MNIST” was given and according to the summary Neural network without dropout had approximately 1.6% error rate on training set and using dropout with Neural network the error rate dropped down to 0.94%. This is significant improvement on the error rate on just using Dropout with Neural Networks.

Batch Normalization: Accelerating deep network training

Deep Neural networks are trained on very large datasets. This dataset may include various distributions of data and as the distribution of data changes the inputs to the next layers change and this causes a lot of deviations which forces us to choose a smaller learning rate slowing down the algorithm. The change in distribution of layers inputs presents a problem because the layers need to continuously adapt to the new distribution. Change in the distribution of internal nodes in a deep network during training is called “Internal Covariate Shift” [5], and this problem can be handled by normalizing the inputs to internal layers. Reducing “Internal covariate shift” dramatically accelerates training deep neural networks without the risk of divergence.

We normalize the data by subtracting each value of the data by the mean(mu) and then dividing it by the standard deviation(sd). The new normalized dataset has mean 0 and variance 1. This method of normalization is applied on the input layer so that the distribution of the input data remains same. Batch normalization extends this method and applies it on the hidden layers and sets them to have a new mean and variance using the learnable parameters Gamma(G) and Beta(B). During the training we compute the mean and variance for each mini batch and use that, but at test time we use the exponentially weighted average of all the mean and variance of all the mini batches and use that for computing the normalized values.

Formulas for batch normalization

Using Batch Normalization, we can use larger learning rates without any side effects like exploding gradients, vanishing gradients or getting stuck at a local optimum. In addition to this Batch Normalization also has a slight regularization effect, thereby not requiring Dropout to take care of the overfitting, it solves both the problems.

Some steps suggested by the author to accelerate batch normalization are Increase Learning Rate, Remove Dropout, Reduce L2 weight regularization, accelerate learning rate decay, Shuffle training examples thoroughly and Remove local response normalization [6]. This algorithm can be applied with tradition neural networks, convolutional neural networks and the authors are working towards the application of this algorithm in Recurrent Neural Networks.

Batch Normalization when applied to the Inception Model and trained over the ImageNet dataset was able to beat the best algorithm of the compaction by a margin. The algorithm got an error rate of 4.9% on the test set consisting of 100,000 images.

ADAM: a method for stochastic optimization

Adam (derived from Adaptive moment estimation) is an algorithm for first order gradient-based optimization of stochastic objective functions [7]. For problems with large amount of data and lots of parameters, Adam algorithm would do well. The method is also appropriate for non-stationary objectives and problems with very noisy or sparse gradients [7]. It is computationally efficient and has very little memory requirement. This algorithm was designed to combine two famous algorithms AdaGrad (which works well with sparse gradients) [8] and RMSProp (which works well in online and non-stationary settings) [9].

Let F(theta) be a noisy objective function that is differentiable. The Adam algorithm calculates the first moment V of the gradient of the function F using the AdaGrad algorithm and the second moment S of the gradient of function F using RMSProp. It then does bias correction on the calculated values as they start at zero and then it updates the parameter theta as -

Theta = theta – learning\_rate \* (V/root(S+epsilon))

The ideal setting for Adam algorithm according to the author is learning-rate(0.001), beta1(0.9) used to calculate the first moment V, beta2(0.999) used to calculate the second moment S, epsilon(10^-8) added for mathematical stability. The values V and S start at 0, that is why the bias correction is added to the algorithm so that it can get to the correct values for the moment quicker.

Adam brings the best of both the algorithms together. It makes the gradient descent converge faster by not allowing it to deviate much and stay focused towards reaching the optimum. Adam algorithm when applied with Dropout works best according to the experiments performed by the author. Dropout adds a little noise, but it also decreases the iteration cost and helps the algorithm adapt faster and better. The experiments show that Adam combined with Dropout when applied on the MNIST data set can get the error rate on the train set down to 0.2% which is a much greater improvement on just using Dropout algorithm. The author shows some experiments in the paper that confirm the analysis on the rate of convergence in convex problems.

Conclusion

In this paper I have introduced three algorithms that are computationally efficient and if are used as suggested by the authors of the papers will work correctly and will allow you to train your Deep Neural Network efficiently.

References

[1] **Dropout: a simple way to prevent neural networks from overfitting** (Submitted 11/2013; published 06/2014; Journal of Machine Learning research 15; Authors: Nitish Srivastava, Geo Rey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov; Department of Computer Science, University of Toronto)

[2] Batch Normalization: Accelerating deep network training

[3] **ADAM: a method for stochastic optimization** (Published as a conference paper at ICLR 2015; Authors: Diederik P. Kingma, University of Amsterdam; Jimmy Lei Ba, University of Toronto)

[4]

[5]

[6]

[7]

[8]