Stochastic Gradient Descent and Average Gradient in a FeedForward Neural Network

Charaf-ed-dine El Fattahi

Abstract—

This note aims to investigate the performance of three different first order optimization algorithms: the Gradient Descent and two of its variants, the Stochastic Gradient Descent and the Stochastic Average Gradient. We first start by a brief introduction to FeedForward Neural Networks. We then show how the Gradient Descent converges to an optimal solution. Then we argue that such method may not be piratical in the some cases. We briefly introduce its two variants SGD and SAG. Finally we apply these algorithms to the IRIS dataset and compare their behavior.

Keywords—FeedForward Neural Network, Stochastic, Average, Gradient Descent, Optimization.

1 Introduction

Over the past thirty years, the use of artificial neural networks (ANNs) has spread in very diverse areas of industry and services. In finance, ANNs have been used for several methods, for example in bankruptcy prediction [1], forecasting of GDP growth [2], managerial applications [3] and Extracting Structure from Stock Returns [4]. They are particularly used to solve problems of prediction, pattern recognition, classification, categorization and optimization [5]. By training a nonlinear system of multiple variables, ANNs can predict the independent variable of the model. Therefore, ANNs constitute a technique for approximating complex systems, which are almost impossible to model using classical statistical methods.

Artificial Neural Networks offer a window to study problems in multiple fields, without a need to understand the field in depth. Therefore, they are at the forefront of current research. ANNs rely on different optimization algorithms to adapt their variables to given data. In this note we go around one of these algorithms, the gradient descent, and some of its derived versions, the stochastic gradient descent and the stochastic average gradient. In section 2 we introduce in few words the idea behind a feed-forward neural network as well as an illustration of the gradient descent method. In section 3 we present the Stochastic gradient descent method and the stochastic average gradient method. We state the motivation behind such methods. In section 4, we apply these methods of learning to one of the most famous datasets, the Iris dataset.

• *EL FATTAHI Charaf-ed-dine – UdeM.*
E-mail: charaf-ed-dine.el.fattahi@umontreal.ca

2 FeedForward Neural Network and Gradient Descent

In this section we will briefly introduce the mathematics behind a feed-forward neural network (FFNN). We will focus on a one layer model, while the generalization to multi layers is straightforward. Learning methods rely on optimization algorithms to reduce their error per iteration. We will center our discussion on the gradient descent method.

2.1 FeedForward Neural Network

A feed-forward neural network, is a neural network without a recurring connection. These neural networks comprise what we refer to as hidden layers comprising several nodes. We refer to these types, multi-layer perceptron (MLP). These are extension of the first artificial neural network, the perceptron invented by Frank Rosenblatt in the 50's [6].

We start with the simplest of MLPs, a neural network with one hidden layer, which associates a vector x of dimension D to a vector $\hat{y}$ of dimension M,

$$\hat{y} = b + W\phi(b^{(1)} + W^{(1)T}x), \tag{1}$$

where x is a vector of size D (D could be the number of features of the training dataset), $z^{(1)} = \phi(b^{(1)} + W^{(1)}x)$ is a vector of size M, which is the output of the hidden layer, b and W are the parameters of the model commonly referred to as bias and weights, respectively. ϕ is the activation function. There are several choices for the activation function (depending on the situation) [7].

In equation 1, the final output was found by a linear transformation of the hidden layer's output $z^{(1)} = \phi(b^{(1)} + W^{(1)}x)$. It is possible to reapply the same kind of transformation to $z^{(1)}$ itself, with different b and W. This would result in a multi-layered (forward-propagating) neural network with two hidden layers. More generally, we can build a deep neural network by piling up a number of these layers. Each of the layers can have a different size, see figure 1.

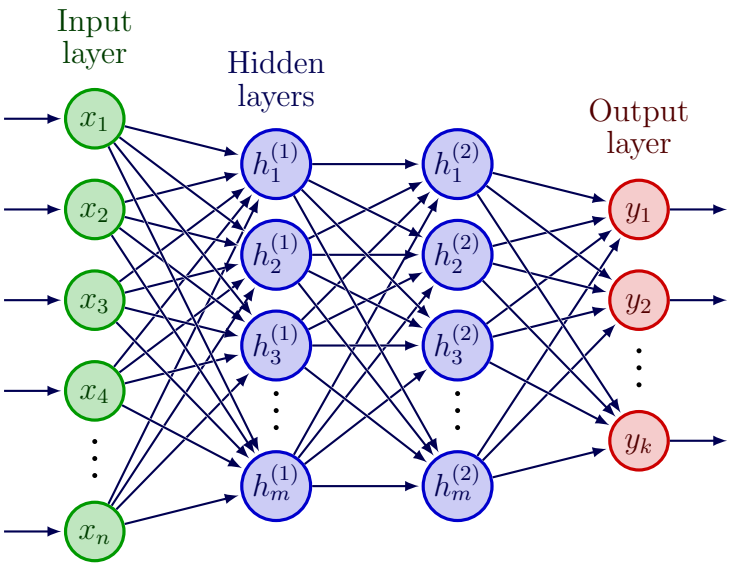

Figure 1: FeedForward Neural Network, with n inputs, 2 hidden layers of size m and k outputs. The arrows refer to the multiple connections between different nodes. The term FeedForward is what makes the arrows to point forward (from left to right).

2.2 Gradient descent

The gradient descent is a first order optimization algorithm. It is intended to minimize a real differentiable function, called the objective function or the cost or loss, defined on a Hilbert space. This cost function is a function of the model's parameters, $W^{(i)}$ and $b^{(i)}$. Two of the most common cost functions are:

- Mean squared Error (MSE): $\frac{1}{N}\sum_1^N (y - \hat{y})^2$,
- Cross Entropy Loss (CEL) (here, for binary classification): $(y \log(\hat{y}) + (1-y)\log(\hat{y}))$.

The algorithm is iterative and therefore proceeds by successive improvements of the bias and weights in the sense of cost reduction. The displacement is carried out in a direction opposite to the gradient, so as to decrease the objective function.

The idea is to start with random values for the bias and weights, for example $b_i^{(l)} = 0$ and $W_{jk}^{(l)} = 1$ and compute the gradient of the cost function. The next step is to update the bias and weights in a direction opposite to the gradient,

$$b^{k+1} = b^k - \alpha\nabla_b Q(b^k, W^k) \tag{2}$$
$$W^{k+1} = W^k - \alpha\nabla_W Q(b^k, W^k) \tag{3}$$

where k is the previous iteration number, α is the learning rate, usually takes values bellow 1, and Q is the objective function. We repeat these steps until we almost converge to a satisfactory solution (error being less than some predefined threshold value). The Gradient descent algorithm is summarized in figure 2.

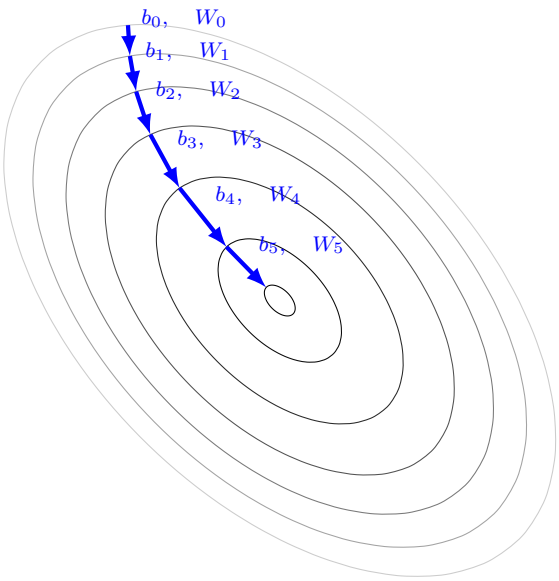

Figure 2: Gradient descent algorithm converging to the minimal cost solution.

3 Stochastic Gradient Descent and Average Gradient

In this section we will introduce alternatives to the Gradient Descent method discussed in section 2.2 in case of big data. These are the Stochastic Gradient Descent and Stochastic Average Gradient.

3.1 Motivation

One of the main advantages of using Stochastic Gradient Descent is that calculations are done faster than the ordinary gradient descent algorithm. This comes handy when dealing with large datasets, as stochastic gradient descent can converge faster since it performs updates more frequently.

We give an example with a Logistic regression that uses $20,000$ genes (features) to determine whether a patient have a disease. In this case, computing one gradient will amount to computing $20,000$ derivatives,

$$\frac{\mathrm{d}}{\mathrm{d}\ \mathrm{gene}_i}\text{Loss function}, \quad \text{for } i \in \{0, 1, \ldots, 19,999\}\,. \tag{4}$$

What if the training dataset contains 10^6 rows. In this case we will have to compute $20,000$ derivatives for each of the 10^6 rows. Adding the number of iterations to our calculations ($1,000$ iterations is a common number), we end up with,

$$20,000 \cdot 10^6 \cdot 10^3 = 2 \cdot 10^{13}$$

computations to train our model. This number can explode rapidly and can cost serious computation time. So for Big Data, the gradient descent is a slow method.

In the next subsection we discuss the Stochastic Gradient descent, an algorithm to overcome this problem.

3.2 Stochastic Gradient Descent

Instead of computing the gradient of each data row, Stochastic Gradient Descent randomly picks one sample for each step and use it to compute the gradient. Thus the SGD reduced the number of computed terms by the number of available training data.

It is intuitive to think that this method will probably result in bigger errors compared to the the Gradient Descent. The SGD is more useful in the case of redundant data.

One interesting feature of the SGD method is that when we get additional data, we can easily use it to take another step for the parameter estimates without having to start form scratch. We just pick up where we left of (with the latest update version of b and W) and take next steps with the new data.

There are many extensions and variants of SGD. One straight forward extension is to consider small batches of the given dataset instead of picking one random sample. This comes in handy when dealing with redundant data, as mentioned before, where the batches can contain one sample from each cluster.

Another variant is the Stochastic Average Gradient.

3.3 Stochastic Average Gradient

The Stochastic Average Gradient algorithm is a Stochastic Gradient method that keeps in track of the previously computed gradients. The new gradient of the current iteration is an average over

all previous gradients and the updated parameters are:

$$b^{k+1} = b^k - \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla_b Q_i \left(b^k, W^k\right) \tag{5}$$

$$W^{k+1} = W^k - \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla_W Q_i \left(b^k, W^k\right). \tag{6}$$

The weights and biases are therefore modified as a function of the average of all the gradients calculated previously. The cost of iterations is constant and independent of the size of the training set, since at each iteration we only compute one gradient, as in SGD. The rate of convergence is however better.

4 IRIS Dataset

In this section we will apply the methods discussed above on a famous dataset, the Iris Dataset. This data set contains 150 rows with 4 features (the length and the width of the sepals and petals in cm). The target has 3 possible values ('setosa', 'versicolor', 'virginica').

4.1 Results: GD vs SGD vs SAG

We run our FFNN, with two hidden layers of size 3 each, on the IRIS dataset using three different optimization algorithms, where we recorded the training computation time as well as loss and accuracy updates on training and validation sets. We present plots of our results. For more detailed results, we put our .ipynb file in the following GitHub repository [8].

In figure 3, we have plotted the loss per iteration as well as the accuracy per iteration on a training set of 112 samples in the case of Gradient Descent. We used an iteration number of $10,000$. The final accuracy is set to 0.9 while the overall training time is 164.72s.

While the Gradient Descent can achieve meaning-full results at $2,000$ iterations, we used $10,000$ instead to compare it with the SGD and SAG since the IRIS dataset is a small one containing only 150 samples.

In figure 4, we have plotted the loss per iteration as well as the accuracy per iteration on a training set of 112 samples in the case of Stochastic Gradient Descent. We used an iteration number of $10,000$. The final accuracy is oscillating bellow 0.97 while the overall training time is 69.17s.

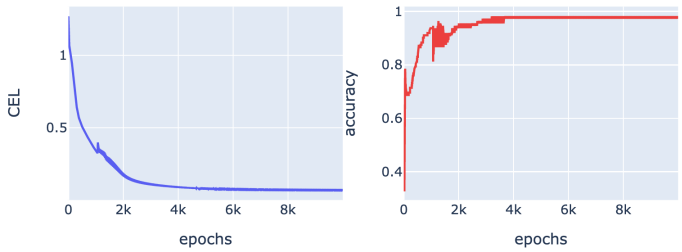

Figure 3: Plot of the loss and accuracy of the Gradient Descent performance per iteration on a training set of 112 samples.

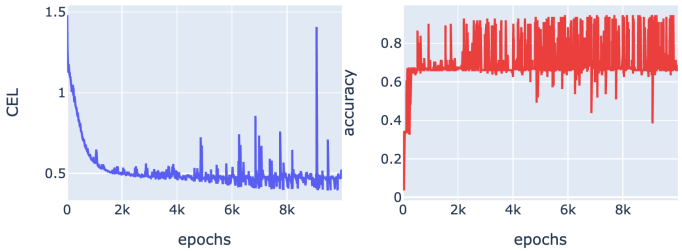

Figure 4: Plot of the loss and accuracy of the Stochastic Gradient Descent performance per iteration on a training set of 112 samples.

In figure 5, we have plotted the loss per iteration as well as the accuracy per iteration on a training set of 112 samples in the case of Stochastic Average Gradient. We used an iteration number of $10,000$. The final accuracy is set to 0.92.

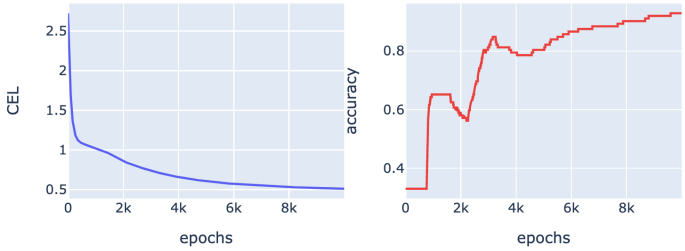

Figure 5: Plot of the loss and accuracy of the Stochastic Average Gradient performance per iteration on a training set of 112 samples.

4.2 Discussion

Comparing the 3 algorithms, we see that the SGD has the smallest training time of 69.17s compared to 164.72s for the Gradient Descent. However, GD scores a 0.96 accuracy compared to an oscillating value bellow 0.97 for the SGD.

One more characteristic behavior we found for the SGD is the oscillation of the accuracy and loss both on the training and validation data. In addition, we can see that the SAG obtains the same overall shape as the SGD but does not oscillate as much in the convergence since the functions taken into account in the displacement gradient are more numerous and therefore more general, which makes it possible to have a better estimate of the optimal descent.

5 Conclusion

In this note, we applied three different optimization algorithms to a FeedForward Neural Network on a small dataset of 150 samples.

We studied the behavior of the loss and accuracy per iteration for each of these variants. The computation time is the highest for the GD while is the lowest for the SGD and SAG. However, SGD has an oscillating behavior which may suggest being non reliable in certain situations or maybe needs improvement. Its SAG variant is more reliable and faster than the GD. The fact the the gradient is computed over averaged samples, growing per iteration, dumps the oscillation behavior of the SGD.

On larger data, SAG may be a good solution to solve the time complexity, but other variants may be interesting to investigate, such as batch GD or second order optimization algorithms.

References

[1] *Rick L. Wilson, Ramesh Sharda. "Bankruptcy prediction using neural networks". Decision Support Systems, Volume 11, Issue 5, 1994, Pages 545-557, ISSN 0167-9236, https://doi.org/10.1016/0167-9236(94)90024-8*

[2] *Greg Tkacz, "Neural network forecasting of Canadian GDP growth", International Journal of Forecasting, Volume 17, Issue 1, 2001, Pages 57-69, ISSN 0169-2070, https://doi.org/10.1016/S0169-2070(00)00063-7.*

[3] *Tam, Kar Yan, and Melody Y. Kiang. "Managerial Applications of Neural Networks: The Case of Bank Failure Predictions." Management Science, vol. 38, no. 7, INFORMS, 1992, pp. 926–47, http://www.jstor.org/stable/2632376.*

[4] *Guoqiang Zhang, B. Eddy Patuwo, Michael Y. Hu, "Forecasting with artificial neural networks: The state of the art", International Journal of Forecasting, Volume 14, Issue 1, 1998, Pages 35-62, ISSN 0169-2070, https://doi.org/10.1016/S0169-2070(97)00044-7.*

[5] *Philip J. Drew, and John R. T. Monson, "Artificial neural networks". (2000) Surgery. https://www.sciencedirect.com/science/article/pii/S0039606000361542*

[6] *Rosenblatt, Frank (1957), "The Perceptron–a perceiving and recognizing automaton". Report 85-460-1, Cornell Aeronautical Laboratory.*

[7] *Chigozie Nwankpa and Winifred Ijomah and Anthony Gachagan and Stephen Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning". (2018) arXiv1811.03378.*

[8] *EL FATTAHI CHaraf-Ed-Dine, "IFT 6512 - Project" Github Repository. (2022) https://github.com/devCharaf/IFT-6512---Project.*