Gradient Descent and Stochastic Gradient Descent

Kanumuri Bheemeswara Vijay Varma
BTech Artificial Intelligence 2020
AI20BTECH11012
IIT Hyderabad

Sachin Karumanchi
BTech Artificial Intelligence 2020
AI20BTECH11013
IIT Hyderabad

Vallepu Manikanta
BTech Artificial Intelligence 2020
AI20BTECH11014
IIT Hyderabad

Abhiroop Chintalapudi
BTech Artificial Intelligence 2020
AI20BTECH11005
IIT Hyderabad

Guguloth Hruday
Btech Mathematics and Computing 2020
MA20BTECH11020
IIT Hyderabad

***Abstract*—Gradient Descent and Stochastic Gradient Descent to find the optimal hyperparameters (or weights of the Artificial Neural Networks). We use Gradient Descent because Gradient Descent converges into a Local Minimum whether the Optimizing function is Convex or Non-Convex. So, by using Gradient Descent and its variants(SGD, Mini Batch), we can find the optimal weights of an Artificial Neural Network. We verify the correctness of the above by designing a working Artificial Neural Network and training it with MNIST data with different loss functions and activation functions and observe the error between each iteration and it indeed converges to a Local Minimum.**

I. Introduction

Artificial Neural Networks or shortly **ANN**'s are a method of Supervised Learning. These ANN's are trained with data and predict the output of unseen data. ANNs are based on inter-connection of nodes (or neurons) called **Artificial Neurons**. These neurons connect with each other and pass the signal(or input). A neuron receives a real number as input and the output is calculated by some non-linear function of the weighted sum of its inputs. Every neuron has some weights attached to it. These weights are first randomly generated. In the training phase of ANN's, a **Loss Function** is used to calculate the error between predicted and true value. By the value of the error, the weights are updated. We have to find the Optimal weights of the ANN by minimizing the Loss Function Value(Error). Now, this transforms into an **Unconstrained Optimization** Problem. So we employ Gradient Descent and Stochastic Gradient Descent to optimize the above problem.

Gradient Descent algorithm is one of the most used algorithms that solves Optimization problems using first-order iterations. There are some situations where we cannot find the Local minimum of a given function analytically (by using Linear Algebra). In such cases, **Optimization Algorithms** are used and Gradient Descent is the most used Optimization Algorithm in Machine Learning, to find the **optimal parameters** of the cost function and **model parameters**. Gradient Descent and Stochastic Gradient Descent (a form of Gradient Descent) are used to optimize millions of hyper parameters (or) weights of the Artificial Neural Networks. Gradient Descent (GD) is an optimization method to find a local (preferably global) minimum of a function. In **Backpropagation**, it is used to iteratively update the weights in order to minimize the error function. In a GD optimization, all the training samples are used for each update of the weights whereas in Stochastic Gradient Descent (SGD), only one or a small batch of training samples are used for each step. With large training sets (common to ANN implementations), the computation time of the GD optimization can become extremely long as one must compute the outputs, errors, and Gradients of all the samples at each iteration. SGD is therefore almost always preferred over GD in neural networks.

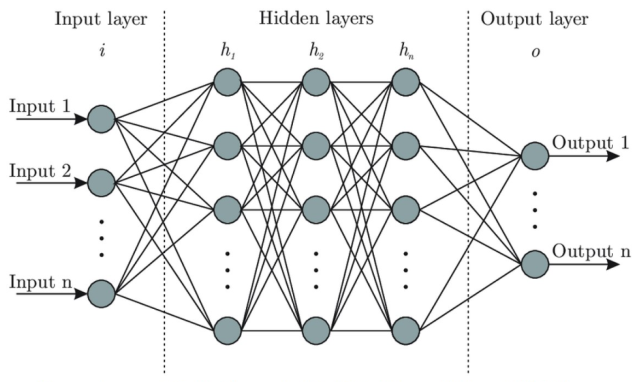

Figure: Image of Fully Connected Artificial Neural Network (ANN)

Fig. 1. Caption

II. Derivation of Gradient Descent

Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be a Convex and Differentiable function, Gradient Descent aims to find the value of x where $f(x)$ is minimum.
Gradient Descent does this by iteratively choosing a new value for x that minimizes the function by moving along the negative Gradient.

$$x_{t+1} = x_t - \alpha \nabla f(x_t) \tag{1}$$

We can get this by performing quadratic expansion of f at x_{t+1} where x_{t+1} is a point near x_t

$$f(x_{t+1}) \approx f(x_t) + \nabla \mathbf{f}(\mathbf{x_t})^\top (x_{t+1} - x_t) + \frac{1}{2}(\mathbf{x_{t+1}} - \mathbf{x_t})^\top \nabla^2 f(x_t)(x_{t+1} - x_t) \quad (2)$$

Since the **Hessian** is hard to compute in practice, we can replace it by $\frac{1}{2\alpha}I$, where α is the **Learning Rate** of the algorithm. Pick a Learning Rate at each iteration that best approximates the Hessian of the underlying function.

$$f(x_{t+1}) \approx f(x_t) + \nabla \mathbf{f}(\mathbf{x_t})^\top (x_{t+1} - x_t) + \frac{1}{2\alpha}||x_{t+1} - x_t||^2 \quad (3)$$

Since we want to find the x_{t+1} that minimizes f, we get the following

$$x^*_{t+1} = \arg\min_{x_{t+1}} f(x_t) + \nabla \mathbf{f}(\mathbf{x_t})^\top (x_{t+1} - x_t) + \frac{1}{2\alpha}||x_{t+1} - x_t||^2) \quad (4)$$

$$= \arg\min_{x_{t+1}} ||x_{t+1} - (x_t - \alpha \nabla f(x_t))||^2 \quad (5)$$

where x^* is the optimal value of x that minimizes f.
From *eq* (5) we can say that the optimal value of x_{t+1} is $x_t - \alpha \nabla f(x_t)$.

III. Convergence analysis of Gradient Descent

To get convergence bounds,
Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be a Convex and Differentiable function, further assume that f is **Lipschitz Continuous**

$$||\nabla f(x) - \nabla f(y)|| \leq L||x - y|| \quad (6)$$

for any x, y and $L > 0$
Since the Gradient of f is Lipschitz Continuous, the largest Eigenvalue of Hessian of f is bounded by L. In the following, we denote this by the expression $\nabla^2 f \preccurlyeq LI$. Since f is convex, its Hessian is Positive Definite. Combining these properties, we get that $LI \nabla^2 f$ is Positive Semi Definite,

$$(\mathbf{x} - \mathbf{y})^\top (LI - \nabla^2 f(z))(x - y) \geq 0 \quad (7)$$

$$L||x - y||^2 \geq (\mathbf{x} - \mathbf{y})^\top \nabla^2 f(z)(x - y) \quad (8)$$

Using Taylor expansion on $f(y)$ around some point x and the Lagrange form of the remainder, we have $\forall x, y, \exists z \in [x, y]$ such that

$$f(y) = f(x) + \nabla \mathbf{f}(\mathbf{x})^\top (y - x) + \frac{1}{2}(\mathbf{x} - \mathbf{y})^\top \nabla^2 f(z)(x - y) \quad (9)$$

$$\leq f(x) + \nabla \mathbf{f}(\mathbf{x})^\top (y - x) + \frac{L}{2}||y - x||^2 \quad (10)$$

Using *eq* (1)

$$f(x_{t+1}) \leq f(x_t) + \nabla \mathbf{f}(\mathbf{x_t})^\top (x_{t+1} - x_t) + \frac{L}{2}||x_{t+1} - x_t||^2 \quad (11)$$

$$= f(x_t) - \nabla \mathbf{f}(\mathbf{x_t})^\top \alpha \nabla f(x_t) + \frac{L}{2}||\alpha \nabla f(x_t)||^2 \quad (12)$$

$$= f(x_t) - (1 - \frac{\alpha L}{2})\alpha ||\nabla f(x_t)||^2 \quad (13)$$

For $\alpha < \frac{1}{L}$, we can write,

$$f(x_{t+1}) \leq f(x_t) - \frac{\alpha}{2}||\nabla f(x_t)||^2 \quad (14)$$

Since f is convex, $f(x_t) \leq f(x^*) + \nabla \mathbf{f}(\mathbf{x_t})^\top (x_t - x^*)$. combining this inequality with above inequality,

$$f(x_{t+1}) \leq f(x^*) + \nabla \mathbf{f}(\mathbf{x_t})^\top (x_t - x^*) - \frac{\alpha}{2}||\nabla f(x_t)||^2 \quad (15)$$

$$= f(x^*) + \nabla \mathbf{f}(\mathbf{x_t})^\top (x_t - x^*) - \frac{\alpha}{2}||\nabla f(x_t)||^2 + \frac{1}{2\alpha}||x_t - x^*||^2 - \frac{1}{2\alpha}||x_t - x^*||^2 \quad (16)$$

$$= f(x^*) + \frac{1}{2\alpha}(||x_t - x^*||^2 - ||x_t - x^* - \alpha \nabla f(x_t)||) \quad (17)$$

$$= f(x^*) + \frac{1}{2\alpha}(||x_t - x^*||^2 - ||x_{t+1} - x^*||) \quad (18)$$

$$f(x_{t+1}) - f(x^*) \leq \frac{1}{2\alpha}(||x_t - x^*||^2 - ||x_{t+1} - x^*||) \quad (19)$$

Summing over all iterations

$$\sum_{i=1}^{k} f(x_i) - f(x^*) \leq \sum_{i=1}^{k} \frac{1}{2\alpha}(||x_{i-1} - x^*||^2 - ||x_i - x^*||) \quad (20)$$

$$\sum_{i=1}^{k} f(x_i) - f(x^*) \leq \frac{1}{2\alpha}(||x_0 - x^*||^2) \quad (21)$$

From *eq* (14) we can say that the function is non-increasing, we can say,

$$f(x_k) - f(x^*) \leq \frac{1}{k}\sum_{i=1}^{k} f(x_i) - f(x^*) \quad (22)$$

$$f(x_k) - f(x^*) \leq \frac{||x_0 - x^*||^2}{2k\alpha} \quad (23)$$

Hence, from the above equation we can see that the Gradient Descent Algorithm converges.

IV. Training the ANN

Here comes the part of updating the weights. We randomly initialize the weights in the ANN. We start training the ANN with our Training Inputs. We choose a Loss Function so that the weights are updated based on the value of the Loss function. The greater the value of the Loss function, the weights are updated greatly. So we minimize the Loss Function Value and find the Optimal Weights. This is an Unconstrained Optimization Problem. Gradient Descent is one of the most popular methods for Optimization. So we use Gradient Descent to optimize the weights and minimize Loss Function iteratively.

$$\theta_{(r)} = \theta_{(r-1)} - \eta \nabla_\theta L(f, x, y) \tag{24}$$

Here, we update the weights iteratively based on the Gradient of the Loss Function with respect to the weights, since the Gradient takes us to the local minimum of the function.

V. Implementing the Artificial Neural Network

We implemented a Artificial Neural Network without using the inbuilt models or libraries in Python using only Numpy Library. The code and implementation can be found here.

The Artificial Neural Network takes arguments such as Number of Hidden Layers and their sizes, Batch Size, Number of Epochs, Activation Function, Learning Rate using Gradient Descent and Stochastic Gradient Descent. Based on the given arguments, the Neural Network is trained with the Input Data.

For every epoch, the training data is shuffled and passed through the model in batches and it's weights are changed based on their Gradients following Gradient Descent(for no batch size) or Stochastic Gradient Descent or Mini Batch Gradient Descent(for a given batch size).

We decided to train our model on MNIST Handwritten Digits Dataset and generate

A. Challenges Faced

1. Since we coded the model from scratch, it took us many attempts to optimise the Neural Network and to be a bit more efficient than our previous attempt.

2. We encountered overflow errors while computing the probabilities at the SoftMax Layer. We got a solution by scaling down the exponents with respect to the maximum element of the Output Layer.

VI. Results

The configuration of our Artificial Neural Network is an Input Layer containing 784 nodes, a Hidden Layer containing 64 nodes, an Output Layer containing 10 nodes compatible with the MNIST Handwritten Data.

We trained the Artificial Neural Network with 50 Epochs, Batch Size of 128, Learning Rate of 0.05 using Gradient Descent, Stochastic Gradient Descent and Mini Batch Gradient Descent.

A. Plot of Training Loss and Testing Loss vs Number of Epochs

Training Loss and Testing Loss vs Number of Epochs for the 3 Gradient methods are as follows

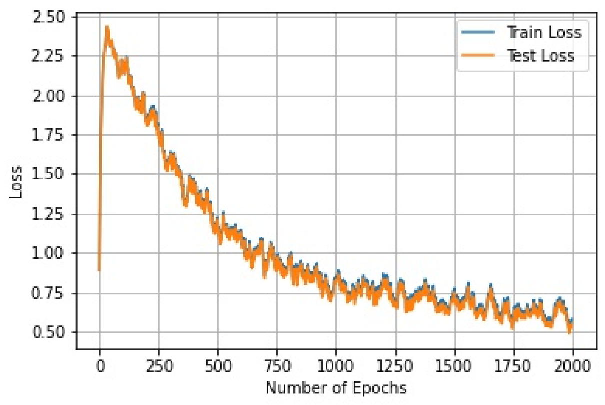

Fig. 2. Loss for Stochastic Gradient Descent

We can observe that for Stochastic Gradient Descent, in the initial epochs, the loss is increasing. This is due to the fact that random sample of training data is passed to the network and after some epochs, the weights are updated in a way which results in the decrease of loss.

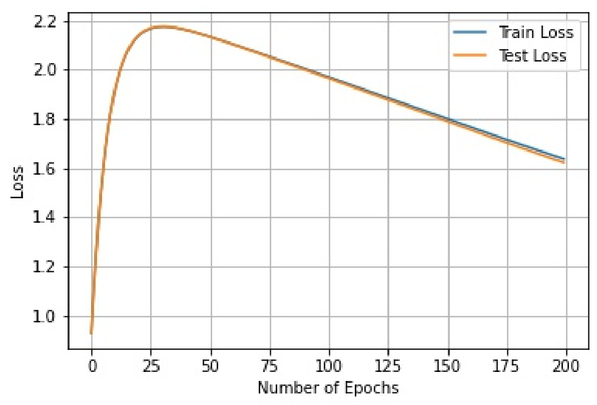

Fig. 3. Loss for Gradient Descent

In a similar way, for Gradient Descent, the randomly generated Weights are updated until a point and then the Total Loss decreases with number of Epochs.

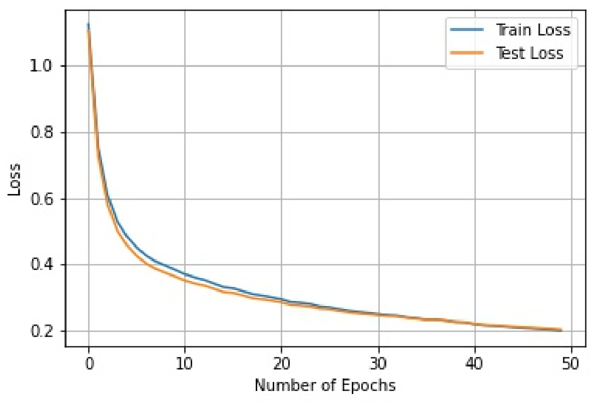

Fig. 4. Loss for Mini Batch Gradient Descent

In Mini Batch Gradient Descent, the randomly generated Weights adjust themselves from the start since we are updating them for every random mini batch which in turn decreases the Loss.

B. *Plot of Training Accuracy and Testing Accuracy vs Number of Epochs*

Training Accuracy and Testing Accuracy vs Number of Epochs for the 3 Gradient methods are as follows

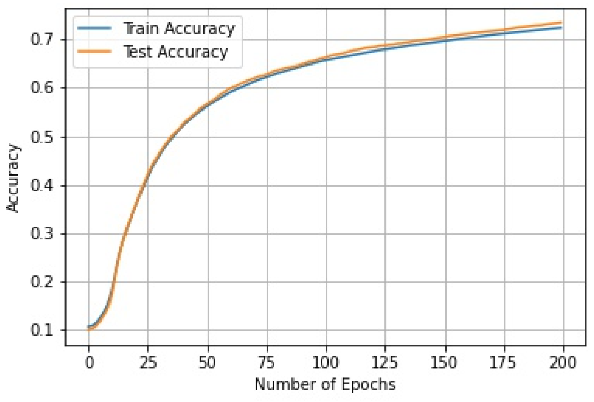

Fig. 5. Accuracy for Gradient Descent

In Gradient Descent, the weights are updated after the whole data is passed, due to this the model's weights are updated very slowly with respect to epochs but in a Consistent manner.So for higher accuracy, we need more number of epochs.

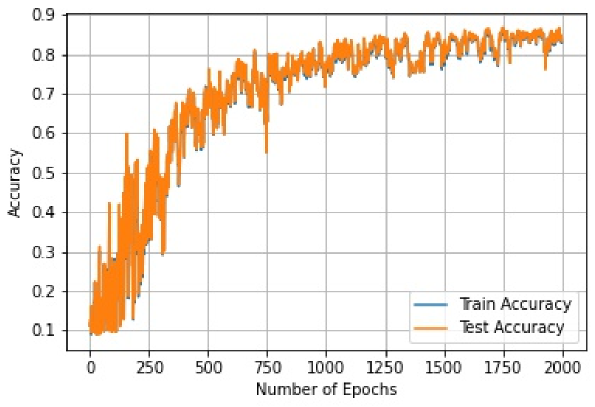

Fig. 6. Caption

Gradient Descent may stop at Local Minimum after some number of epochs. Stochastic Gradient Descent avoids this Local Minimum by fluctuating values of the output for small change in input. Since one data example is passed for one epoch, the model can't predict output of other examples well. Dues to this, we need more epochs for higher accuracy. Hence it reaches a Global Maxima for sufficient number of epochs.

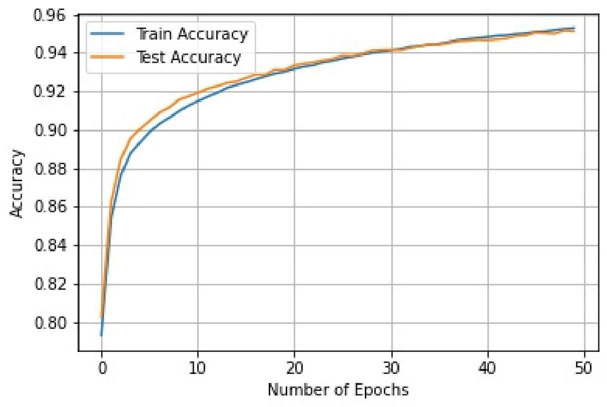

Fig. 7. Accuracy for Mini Batch Gradient Descent

In Mini Batch Gradient Descent, since for a given Mini Batch, the weights are updated, the model can predict the output of other examples well based on the previous batches passed. Hence, Mini Batch Gradient Descent does not require large number of epochs for higher accuracy.

C. *Plot of Gradients of a Parameter(Weight) vs Number of Epochs*

For optimal convergence of any Gradient Descent method, Gradient of every Weight should tend towards zero so that no Weights are updated and the current Weights are Optimal.

The Gradients of a Random Parameter(Weight) vs Number of Epochs for the 3 Gradient methods are as follows

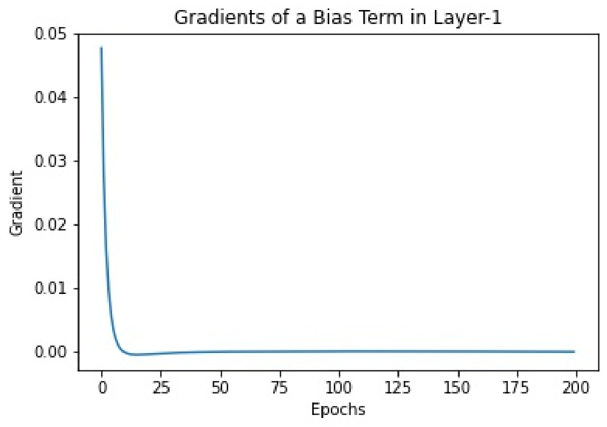

Fig. 8. Gradients in Gradient Descent

In the case of Gradient Descent, it tries to converge to zero or maybe increase up to some Epochs and then decrease.

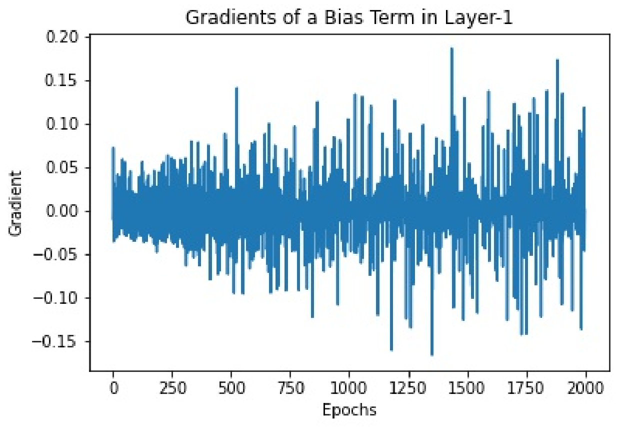

Fig. 9. Gradients in Stochastic Gradient Descent

In the Stochastic Gradient Descent case, the Gradients are always fluctuating and they may increase or decrease.

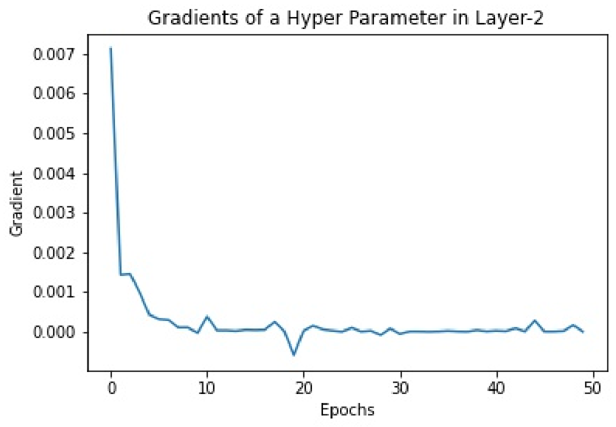

Fig. 10. Gradients in Mini Batch Gradient Descent

For Mini Batch Gradient Descent also, there are small bumps and the Gradients tend to converge to zero.

VII. Conclusions

Mini Batch Gradient Descent, Stochastic Gradient Descent and Gradient Descent are one of the best methods to compute the Optimal values for problems which does not have an analytical solution. One such example is Artificial Neural Networks.

From the above data, Mini Batch Gradient Descent is the best among the three in terms of Time Taken, Usage of Computational Resources followed by Stochastic Gradient Descent and Gradient Descent.

References

[1] https://medium.com/oberman-lab/proof-for-stochastic-Gradient-Descent-335bdc8693d
[2] Hastie, T., Tibshirani, R., Friedman, J.H. and Friedman, J.H., 2009. The elements of statistical learning: data mining, inference, and prediction (Vol. 2, pp. 1-758). New York: springer.
[3] Ruder, S., 2016. An overview of Gradient Descent optimization algorithms. arXiv preprint arXiv:1609.04747.
[4] https://raghumeka.github.io/CS289ML/gdnotes.pdf
[5] https://www.andrew.cmu.edu/user/anubhava/bettersgd.pdf