Hessian Free Optimization on Deep Neural Networks

Nitish Kumar Gundapu
MTech, Department of AI
ai22mtech14004@iith.ac.in

Vishnu Vijay Tiwari
MTech, Department of AI
ai22mtech14003@iith.ac.in

Abstract

Training a neural network while minimizing the loss function wrt the parameters has been quite a difficult task for the past few years. Recent approaches such as adaptive learning rates and momentum addition to the gradient have shown promise in improving performance. However, instead of tuning hyper-parameters, we will take a deeper approach by implementing the Hessian Free Optimization technique, one of the most powerful optimization methods that can significantly improve neural network performance. This Project aims to explore and implement this technique to achieve the best results in neural network training.

Key Terms

hyper-parameter, gradient, learning rate, Hessian

I. Introduction

Traditional optimization methods, such as gradient descent, have been widely used for neural network training. However, these methods have limitations, including slow convergence rates, sensitivity to hyper-parameters choice, and difficulty escaping local minima. Recent advancements in optimization techniques have focused on adaptive learning rates, momentum-based Optimization, and regularization methods to overcome these limitations. Despite these advances, optimizing neural networks remains a challenging problem. This Project aims to tackle this challenge by implementing Hessian Free Optimization, which has shown to be one of the most efficient optimization techniques that can significantly improve neural network performance. We will see through the different optimization methods till we reach the hessian-free technique:

1) **Stochastic Gradient Descent:** Stochastic Gradient descent is a popular optimization algorithm that minimizes the loss function in neural network training. It is a first-order optimization method that iteratively updates the network parameters in the opposite direction of the gradient of the loss function with respect to the parameters. This direction is known as the steepest descent, and a learning rate hyper-parameter determines the step size. The updated equation for SGD [1] is as follows:

$$W_{i+1} = W_i - \alpha \nabla f(x_i, y_i)$$

Here W_n is the parameter , α is learning rate , $\nabla f(x_i, y_i)$ is gradient of W_n with respect to loss function

2) **Newtons Method**:
Newton's method is a second-order optimization technique that addresses some of the limitations of SGD. It works by approximating the Hessian matrix, which estimates the curvature of the loss surface. By using the Hessian matrix, Newton's method can take more significant steps in the direction of the minimum and converge faster than SGD. The updated equation for Newton's Method [1] is:

$$W_{i+1} = W_i - \alpha \left(H\left(x_i, y_i\right)\right)^{-1} g\left(x_i, y_i\right)$$

Here W_n is the parameter , α is learning rate, $H(x_i, y_i)$ is hessian matrix and $g(x_i, y_i)$ is gradient of W_n with respect to loss function

3) **Hessian Free Optimization**: Hessian Free Optimization is a modified version of Newton's method that overcomes its limitations. It uses the conjugate gradient method to approximate the inverse of the Hessian matrix. This approximation is used to update the parameters in the direction of the minimum, similar to Newton's method [1].

II. Architecture

In this Project, we have taken a simple neural network architecture with two layers, one input layer, and one output layer. The activation function at the output layer is Sigmoid, and the loss function is defined as follows:

$$f(\underline{w}) = \left\|g\left(W^T \underline{x_i}\right) - \underline{y_i}\right\|^2 \quad where \;\; g(\underline{x}) = \frac{e^{\underline{x}}}{1+e^{\underline{x}}}$$

The input layer has four input neurons, and the output layer has two neurons; for simplicity purposes, we consider having no bias and no hidden layers in the network.

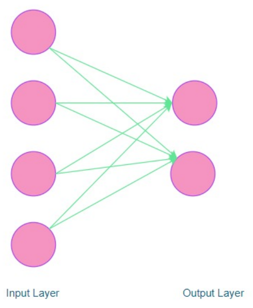

Fig. 1. Neural network architecture

III. Dataset

The Data used in this Project is a simple classification dataset where xi is taken randomly from a d-dimensional plane using $np.random.rand(d, N)$, having d dimensional N points where y_i from the below condition:

$$y_i = \begin{cases} 1 & \text{if } x_i(1) - x_i(2) + x_i(4) \geqslant 0 \\ 0 & \text{otherwise.} \end{cases}$$

In this project, we have considered N = 1000 and d = 4 , training data as 75% and testing as 25%.
The Plot of the dataset in 2D using PCA is :

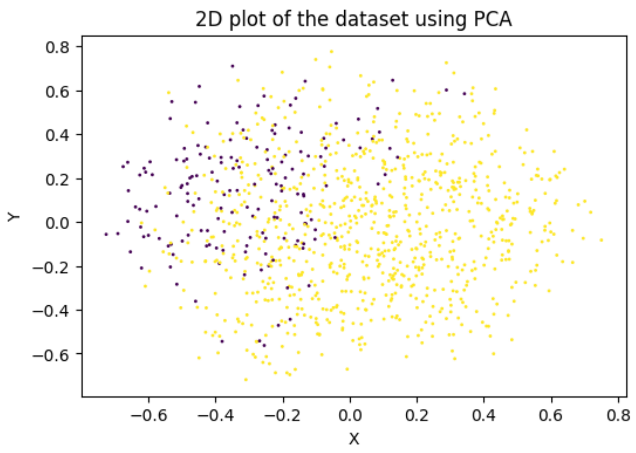

Fig. 2. Plot of dataset

IV. Problem Statement

As we have defined the architecture, we can now formulate the optimization problem of the neural network as follows.

$$\min_{W\in\mathbb{R}^{4*2}, x_i\in\mathbb{R}^4, y_i\in\{0,1\}} \left\|g\left(W^T \underline{x_i}\right) - h(\underline{y_i})\right\|^2, \quad where \;\; g(\underline{x}) = \frac{e^{\underline{x}}}{1+e^{\underline{x}}}$$

Here , $g(x)$ is the sigmoid function and $h(x)$ is used to **one-hot encode** y_i depending on no of classes present in dataset, which is two in our case. Now our main task is to minimize the problem using the above mentioned algorithms and try to get the Optimal value of W and compare the results.

V. Algorithms

Below are the algorithms that we have ran through our neural network and have noted the results obtained by them.

Algorithm 1: Stochastic Gradient Descent

```
Data: X,Y training data
Result: Neural network with trained parameters
Loss function: $f(x,y)$
pick up $x_0$ as your initial guess;
while epochs do
    while $x_i$,$y_i$ in training data do
        $g_i = \nabla f\left(x_i, y_i\right)$;
        $W_{i+1} = W_i - \alpha g_i$;
    end
    epochs -= 1;
end
where $W_i$ is the parameter of the neural network , $g_i$ is the gradient of $W_i$ wrt Loss function $f(x,y)$
```

The SGD is an iterative optimization method that updates the parameters in the opposite direction of the gradient, where the gradient is first ordered. This has been proven to be a significant limitation of this method because it means we are assuming that the error surface of the neural network locally looks and behaves like a plane. We are ignoring the curvature of the surface, which leads toward the slower convergence towards minima of the loss function. Hence it will take more updates to converge, which is less efficient.
Now we will look for the second order minimization method which is Newton's Method.

Algorithm 2: Newtons Method

```
Data: X,Y training data
Result: Neural network with trained parameters
Loss function: $f(x,y)$
pick up $x_0$ as your initial guess;
while epochs do
    while $x_i$,$y_i$ in training data do
        $g_i = \nabla f\left(x_i, y_i\right)$;
        $h_i = H(f)\left(x_i, y_i\right)$;
        $W_{i+1} = W_i - \left(h_i\right)^{-1} g_i$;
    end
    epochs -= 1;
end
where $W_i$ is the parameter of the neural network , $g_i$ is the gradient of $W_i$ wrt Loss function $f(x,y)$
  and $h_i$ is the hessian matrix of the neural network.
```

It performs better than the Stochastic Gradient Descent because we are calculating the Hessian Matrix, which is a second-order gradient that estimates the loss function's curvature. This method helps parameters take more significant steps toward the global minima and converge faster than SGD.

However, this method has two significant drawbacks :

1) This is a computationally expensive method because we have to calculate Hessian for each data point in the training loop. It takes time complexity of $O(n^2)$ where n is the total no of vectors of all parameters. This method is quite expensive and becomes more complicated with increased parameters.
2) The Calculation of the Hessian itself is tricky due to the complexity of the neural network. Generally, we use the back-propagation method in neural networks to obtain the gradients at any layer in the network. However, this method only works for the Gradient calculation, and we have to use an entirely different approach for Hessian calculation.

This is the trade-off that we can observe from the Stochastic Gradient Descent method for a better convergence rate. Hence we now look at the hessian free Optimization, which will take care of the drawbacks of Newton's method. Before that, we will look at a different iterative method, conjugate gradient descent, which minimizes the Quadratic functions. Although this method can also be used to optimize non-linear functions, we are now focusing on and using this Conjugate gradient descent method for optimizing a linear function.

Algorithm 3: Conjugate Gradient Descent

```
Data: X,Y training data
Result: Neural network with updated parameters
Objective function: $f(x) = \frac{1}{2}x^T Ax + b^T x + c$
pickup $x_0$ as your initial guess;
pickup $d_0$ as $-\nabla f(x_0)$;
while $x_i$,$y_i$ in training data do
    while $n$ do
        $\alpha = -\frac{d_i^T(Ax_i+b)}{d_i^T Ad_i}$;
        $x_{i+1} = x_i + \alpha d_i$;
        $\beta_i = \frac{\nabla f(x_{i+1})^T Ad_i}{d_i^T Ad_i}$;
        $d_{i+1} = -\nabla f(x_{i+1}) + \beta_i d_i$;
        $n- = 1$
    end
end
where $d_i$ is gradient wrt Objective function and $\alpha, \beta$ are the hyper-parameters
```

This Algorithm is similar to SGD, where we calculate the gradients and update the parameters. However, unlike SGD, here we are not fixing the α value. Instead, we are performing a line search to find the best α. This α is the value of α which brings us to the minimum of the loss function. As we move in the new direction, .i.e, after updating the parameters, the following gradient will be in another direction compared to the previous one; hence here, we also introduce β to save the cumulative direction of the gradients to reach the minimum of the loss function.
Now we use the above-mentioned concept to implement Hessian Free Optimization [2]

Algorithm 4: Hessian Free Optimization

```
Data: X,Y training data
Result: Neural network with Updated parameters
Loss function: $f(x, y)$
Objective function: $g(x, y) = f(x + \Delta x, y) = f(x, y) + \nabla f(x, y)^T \Delta x + \Delta x^T H(f)(x, y)\Delta x$
pick up $x_0$ as your initial guess;
while $W_n$ is converged do
    while $x_i$,$y_i$ in training data do
        update $W_{n+1}$ using Conjugate Gradient for $g(x_i, y_i)$
    end
end
```

This Algorithm is very similar to conjugate gradient descent, where we use a quadratic function to optimize. Here, in this case, we take Taylor's series expansion up to the second derivatives, which act as the quadratic function for the conjugate gradient descent. Let us see how this solves the problem of newton's method.
the CGD calculates the gradient of the linear function at the start, hence the gradient of $g(x, y)$ is

$$g'(x, y) = \nabla f(x, y) + H(f)(x, y)\Delta x$$

This step is similar to the newtons method, where we need to calculate Hessian for parameter updation. However, here, instead, we can calculate the Hx Hessian-vector product without calculating Hessian at all. Hence this method is known as **Hessian free Optimization**. To get this, let us see what the hessian-vector product is represented like.

The Hessian vector product can be represented as the dot product of the vector and second-order partial derivatives of the ith row and jth column in the Hessian matrix, which can be represented using the following equation.

$$(Hv)_i = \sum_{j=1}^{N} \frac{\partial^2 f}{\partial x_i x_j}(x) \cdot v_j = \nabla \frac{\partial f}{\partial x_i}(x) \cdot v$$

where $\partial f/\partial x_i$ is the directional derivative in the direction of v. We can define and approximate the directional derivative using finite differences [1] as follows

$$\nabla_v f \approx \frac{f(x+\varepsilon v) - f(x)}{\varepsilon}$$

using the above equation we can approximate $(Hv)_i$ using finite differences with $\frac{\partial f}{\partial x_i}$. When we can bundle this all up into vector form, we can define Hv as :

$$Hv \approx \frac{\nabla f(x+\varepsilon v) - \nabla f(x)}{\varepsilon}$$

Hence, the Hessian vector product is approximately equal to the difference between the gradients, which is much more computationally efficient than Newton's method, and we do not need to compute Hessian itself. This way, the Algorithm tackles the problems of Newton's method. This method also performs better than SGD because the convergence rate is faster due to α, β parameters in the conjugate gradient descent. Hence the Hessian free Optimization generally outperforms SGD and Newton's method.

VI. Results

The below plot is the training losses vs. epochs implemented for the different algorithms mentioned.

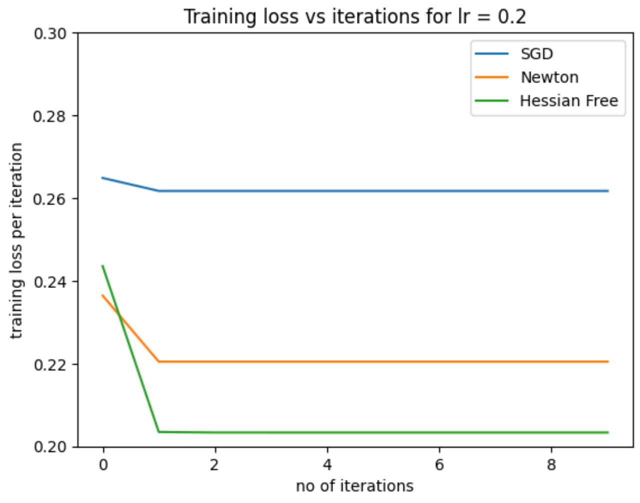

Fig. 3. Plot of dataset

The plot shows that the Hessian free Optimization algorithm outperforms the other two algorithms and reaches minima faster than the other two. We can also see that newtons method will reach minima at better than SGD after a few epochs due to its consideration of second-order derivatives in its Algorithm. The technical results for the algorithms are as follows.

Algorithm	learning rate	no of epochs	time taken	accuracy	training loss at last epoch
SGD	0.2	10	3.59 sec	62.8 %	0.26
Newton	0.2	10	44.28 sec	93.6 %	0.22
Hessian Free	0.2	10	15.48 sec	96.4 %	0.20

From the results table, we can observe that the Hessian Optimization method takes less time and performs better than Newtons method and is more efficient than SGD.

VII. Conclusion

This Project explores advanced optimization techniques for neural networks, focusing on Hessian Free Optimization. While traditional optimization methods have limitations, recent advancements have shown promise in improving performance. Hessian Free Optimization approximates the Hessian matrix using a conjugate gradient method, providing a computationally efficient way of updating parameters. Results show superior performance compared to traditional methods, with significant implications for applications in natural language processing. Hessian-free Optimization has been primarily used for convex optimization problems, but there is growing interest in using it for non-convex problems. Developing methods that can handle non-convex objective functions is a significant open problem. In this Project, we have used a simple neural network, but depending upon the architecture and complexity of the neural network, hessian-free optimization performs much better. The performance of the hessian-free Optimization can be further improved by damping techniques, adding termination conditions for conjugate gradients, and using regularization techniques.

VIII. Code

1) Github https://github.com/NitishKumarGundapu/Convex_Optimization_Project

2) Colab https://colab.research.google.com/drive/1bsAuDYGunECum1T9Esiu5t6uVrbtOmXh?usp=sharing

References

[1] A. Gibiansky, “Hessian free optimization,” in *Blog on Hessian free Optimization*. Blog article, 2014.

[2] U. o. T. James Martens, “Deep learning via hessian-free optimization,” pp. 1–4.