Minimization of the Loss function in neural-networks using Conjugate gradient method

Srikrishnan Sengottai Kasi

December 3, 2021

Abstract

This report focuses on implementing the Fletcher-Reeves conjugate gradient method to minimize the loss function implemented in the neural network. It also focuses on comparing the results of minimizing the loss function with the help of fixed step size gradient descent method. Since, the loss function chosen is non-linear, the report will discuss on the results of periodically restarting the conjugate gradient method algorithm and see if it is able to reach the global minimum.

1 Introduction

The artificial neural networks are supervised machine learning models. The network has neurons which are interconnected with each other. Each neurons has weights and bias associated with it. In general the neural networks are trained with some sample data sets. Then it is tested with new data to classify it based on the information gathered during the training phase. Each neuron in the network applies an activation function. There are 3 layers in the network namely input layer, hidden layer and the output layer. The neurons are trained in such a way, the loss function is minimized when the parameters of the network does the expected classification. The minimization of loss function can be done using the conjugate gradient method. In this report the Fletcher-Reeves conjugate gradient method is used to minimize a non-linear loss function which is used to train the neural network. Initially, the report discuss in detail about how the loss function is minimized with the help of the gradient descent method with fixed step size and then it compares its results with the conjugate gradient method.

2 Background

Before going into the implementation, the technical background to understand the experiment is needed. The activation function used by the neurons in neural network is sigmoid function,

$$\Theta(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The 'x' is the input in neuron which has a weight and bias associated with it. Then, lets have a look at the loss function used. The nature of loss function has to be continuous and we are considering a non-linear one for our experiment.

$$f(\theta) = -\sum_{k=1}^{m} \ln\left(\frac{1 + sign(\upsilon_k)y(\xi_k)}{2}\right)$$

The θ is nothing but the parameters present in the neural network. The υ_k is the expected output of the network we want. The 'm' is the size of the training set. The ξ is the training set for the network. The 'y' is the output of the neural network. In our experiment for the conjugate gradient method we use the Fletcher-Reeves method to find the direction of minimum in the next steps with the help of the below formulation. The mutual conjugacy holds when we use the below.

$$p_0 = -\nabla(f(x));$$

$$p_k = -\nabla f(x_k) + \frac{(\nabla f(x_k)^T \nabla f(x_k)}{\nabla f(x_{k-1}^T \nabla f(x_{k-1}))} * p_{k-1}$$

p_0 is the search direction vector computed at 0^{th} step. After that we will use p_k to find the direction of the minimum. Since, the loss function we are considering is non-linear we might require resets periodically. In order to find the minimizer in the above computed search direction we apply the below formulation,

$$x_{k+1} = x_k + \delta_k p_k$$

and find the δ for which the $f(x_k + 1)$ is minimum. Then I will update the $x_k + 1$ value with the computed δ which gives the minimum in the new found search direction.

3 Experiment

In this experiment I tried to employ a feed-forward network with n_0 input units, a single hidden layer and one output unit. The activation function used is the sigmoidal function. The construction of the neural network will be taken care by the *makeNet* step of the Driver Algorithm. To train our neural network we will consider a Boolean classification problem. It will be a simple XOR function. We define a training set of size 2 and we will have 2 neurons in the hidden layer. The algorithm to start our experiment is defined as follows.

Algorithm 1 Driver Algorithm

$\xi \leftarrow [-1 - 111; -11 - 11]$
$\upsilon \leftarrow [-111 - 1]$
$n1 \leftarrow 2$
$makeNet(size(\xi, 1), n1)$ ▷ Neural network is constructed
$runs \leftarrow 0$
while $runs \leq 100$ **do**
  $trainNet(\xi, \upsilon, n1)$ ▷ The training of the constructed net takes place
end while

Once the network is constructed, the network is trained 100 times to see how it minimizes the loss function so that the network behaves the way it is expected to. The training algorithm is described below.

Algorithm 2 $trainNet(\xi, v, n1)$

$[n0, m] \leftarrow size(\xi)$
$n \leftarrow n0 * n1 + 2 * n1 + 1$
$\theta \leftarrow [randn(n - n1 - 1, 1); zeros(n1 + 1, 1)]$ ▷ Generating random inputs to train
$restart \leftarrow 5$
$iteration \leftarrow 0$
$[f(\theta), p_0) = runNet(\xi; \theta; v)$ ▷ This part runs the network for the generated θ
$p_0 \leftarrow -p_o$
while $iteration \leq 1000$ **do**
  $\gamma = Minimum_value(f(\theta + \gamma * p_0))$
  $\theta_{k+1} = \theta + \gamma * p_0$
  if remainder(iteration/restart) == 0 **then**
    ▷ reset the search direction with simple negative of the gradient
  else
    ▷ update the search direction with the fletcher reeves formula
  end if
  if $f(\theta) <= threshold$ **then**
    break
  end if
  $iteration \leftarrow iteration + 1$
end while

From the above algorithm we could see how the periodic restart is deployed in the fletcher-reeves method implementation. Moreover we are defining the stopping conditions of the algorithm. The algorithm terminates if the number of iterations to run the Conjugate gradient method reaches 1000. It's stopping is also made dependent on the $f(\theta)$ value computed in each iteration. In my experiment I tried to set that to three different values namely 0.1, 0.01 and 0.001. Also, in matlab implementation the algorithm terminates when the θ_{k+1} goes to infinity because of the γ value has chances of going to infinity. This will cause numerical issues and hinders us from performing valuable observation. Now lets observe the performance of the conjugate gradient method to minimize the loss function in comparison with the gradient descent method with a fixed γ of 2.0 in the observation section.

4 Observation

First, lets analyze how many iterations does the gradient descent method took for the loss function to reach the three thresholds we defined in the experiment. The number of iterations is the mean value of the 100 runs computed by the driver algorithm i.e out of the 100 running times, we find the mean of the iterations in which the particular threshold value for the function is reached and tabulated it as shown in the Figure 1. As I can clearly see that each threshold it takes more than 200 iterations for this approach to reach. One

Threshold	Mean No. of iterations to reach threshold in 100 runs of fixed gamma = 2.0
0.1	220.53
0.01	285.50
0.001	532.02

Figure 1: Iterations for gradient descent to reach the threshold

reason is because of the fixed step size, sometimes we tend to overstep in the direction of the minimum. So, it takes more time if we want to minimize the loss function threshold further to 0.001. In figure 2, we try to analyze the conjugate

No. of periodic restarts of conjugate gradient method	Mean of No. of iteration to reach a threshold of .1 in 100 runs	Mean of No. of iteration to reach a threshold of .01 in 100 runs	Mean of No. of iteration to reach a threshold of .001 in 100 runs
10	30.21	30.74	26.69
5	19.28	18.40	26.17
2	19.01	27.07	20.76
0	72.08	89.23	82.73

Figure 2: Iterations for conjugate gradient method to reach the threshold

gradient method by periodically setting the restarts between 0 and 10. When there is no restarts I can see from the figure 2 that the number of iteration to reach the threshold lies close to 100. While, if we keep the restart more frequent as 2, then the number of iteration to reach the threshold is minimum. But when we keep the restart at 5, the number of iteration is lower than at 2 and 10. But

No. of periodic restarts of conjugate gradient method	Mean of the Sum of total number of computations to find the step size using GSS to reach a threshold of 0.1 in 100 runs	Mean of the Sum of total number of computations to find the step size using GSS to reach a threshold of 0.01 in 100 runs	Mean of the Sum of total number of computations to find the step size using GSS to reach a threshold of 0.001 in 100 runs
10	104.90	110.51	123.55
5	78.19	119.24	103.79
2	136.90	190.14	162.95
0	302.48	430.83	379.86

Figure 3: Iterations for conjugate gradient method to reach the threshold

another main factor that is to be considered is the number of iteration it takes to find the minimum in the new search direction computed by the conjugate gradient search using the fletcher-reeves formula. When we consider the runs with zero restarts performed poorer than the gradient descent method with the fixed step size. While in other approaches of the conjugate gradient method, the number of iterations is considerably less than the gradient descent method. Alternatively frequent restarts makes our conjugate gradient setup look like a gradient descent but the only change would be rather than keeping a fixed step size we are finding the step in the new search direction using a line search method like golden section search.

5 Conclusion

From the report I have improved the rate of reaching the threshold by periodically restarting the conjugate gradient method. Thus, the loss function reaches the threshold faster than using the simple gradient descent with fixed step size. Further, we could try to compare the step size computed by each approaches. Also, when we tend to train immense data sets in the real time it is costly to compute the gradient in each step. So devising an algorithm that derives the gradient in randomly selected points and determining the gradient values in other areas probabilistic-ally.