csoc-ig report

Shambhavi

May 2025

NEURAL NETWORK

1 Introduction

The code involves executing Neural Network at two levels:

1. Neural network from scratch (only using pure Python and libraries like numpy)
2. PyTorch Implementation

2 Preprocessing

There were no null values. The columns Gender and No-Show contained string values. For efficient functioning of the model, they have to be converted to integer-type values. I used the replace function. In the gender column, Male and Female replaced to 0 and 1 respectively. In No-Show No and Yes replaced to 0 and 1 respectively. The columns used for training or the input values are age, hypertension, diabetes, handcap, gender, scholarship, SMS-received. These are the columns correlated with No-Show.

3 Model

Part 1: Neural Network from Scratch A function name ANN is created where input value is 7 and the size of hidden layer is taken to be 50. All the weights and biases are initialized randomly. In Forward Process,

$$net_h = w_1 * x_1 + w_2 * x_2 + ... + w_n * x_n + b_1$$

,

$$b_1$$

is bias,

$$w_i$$

are the weights and n=7 for our model. here we get net input for hidden layer. For the net output of the hidden layer, the ReLU function is used which is simply to get the maximum value. Net input for output is,

$$net_o = w_1 * h_1 + h_2 * x_2 + ... + h_n * x_n + b_2$$

,

$$b_2$$

is bias,

$$w_i$$

are the weights(these weights are different from above weights). For output, **Sigmoid** function is used. It is a mostly used function in neural networks.

In Backward process, the error calculation is based on this formula:

$$E = \frac{1}{2} \sum (y - output)^2$$

using chain rule we find derivative/gradient of Error with respect to weight,

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial output} * \frac{\partial output}{\partial net_o} * \frac{\partial net_o}{\partial w_i}$$

After finding gradients, All the weights and biases will be updated

$$w_i = w_i - \frac{\partial E}{\partial w_i} * learning\ rate$$

where learning factor is the size of the step we should take to reach the minimum in our case I found the $learning\ rate = 0.01$ to be fine. The same formuala is used for calulating bias. We repeat this step for a given number of iterations. **number of iterations are 500**

Part 2: PyTorch Implementation class simpleNN simply feedforward neural network with hidden layer and one output layer . In forward function, Processes input (x) through the hidden layer, applying Leaky ReLU activation. At first tried with ReLU but it works better with Leaky ReLU. It uses 50 neurons in the hidden layer for better learning. For loss function ,Binary Cross-Entropy Loss is used since BCEWithLogitsLoss applies sigmoid internally. While training,Iterates through 1000 epochs, updating weights using gradient descent. Applies threshold of 0.5 to convert probabilities into binary classification.

4 Convergence Time

For only Python implementation, the converging time is 105 sec for 500 iterations. For PyTorch implementation, the converging time is 30.8 sec for 1000 iterations. This indicates that PyTorch is faster.

5 Accuracy

For only the Python implementation, the accuracy is about 79.9 percent, the f1 score is 0, and the precision-recall accuracy is 0.5. This means that the model only predicted the majority class or value. It failed to predict the minority values. For the PyTorch implementation, the accuracy is about 74.7 percent, with an f1 score of 0.19 and precision-recall accuracy of 0.54. This means that it accurately predicted some of the minority values.

6 Summary

Overall PyTorch implementation is much faster and accurate. It is better in predicting minority values.