Detailed Description of fully connected feed forward neural network

Samin Yeasar Arnob [1]
samin.arnob@mail.mcgill.ca

I. Introduction

This pdf contains detailed description of the procedure followed to make fully connected feed forward neural network. Only numpy library has been used to matrix manipulation and other calculation.

II. Description

A. Fully Connected Feed forward Neural Network

The neural network is considered universal function approximation, which means that with enough nodes, a two-layer neural network can approximate any function within error bound. We build a fully connected neural network where the image data was flattened into an array and feed into the input node with randomly initialized weight and bias as zero at each layer. Using forward propagation we computed output approximation and error comparing original output and then backward propagation to calculate a gradient that is needed update our weights and biases.

I implemented the code from scratch using only numpy module.

1) Parameter Initialization: I did a random initialization for the weight W matrices and used "np.random.randn(shape) * 0.01". And zeros initialization for the biases b. Use "np.zeros(shape)". Random initialization is important when we look at the following forward propagation equations, where output of activation depends on weight W matrices. If W initializes as 0, neuron will not get activated, can be inferred from the forward propagation equation described below.

2) Forward Propagation: Now that I have initialized my parameters, forward propagation can be computed using following equations:

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$$

where $A^{[0]} = X$ our input data.

3) Activation Function: Pre-activation neuron is activated using an activation function. There are many activation functions available; ex- sigmoid, tanh, Relu, leaky Relu. The Mathematical relation is:

$$A^{[l]} = g(Z^{[l]}) = g(W^{[l]}A^{[l-1]} + b^{[l]})$$

where g is the the activation function. We used two activation functions:

Sigmoid: Sigmoid function squashes neuron's pre-activation with o and 1. I used sigmoid at the output layer to get a prediction in the range [0,1].

$$A = g(Z) = g(WA + b) = \frac{1}{1 + e^{-(WA+b)}}$$

if

$$A = g(Z) \quad then \quad g^{'}(Z) = 1 - a^2$$

As sigmoid function may cause vanishing gradient problem during backpropagation when Z is gets high or low value due to the nature it's derivative, I used ReLu in the hidden layers for activation.

ReLu: The mathematical formula for ReLu is

$$A = ReLu(Z) = max(0, Z)$$

Activation is simply thresholds at zero, doesn't have the vanishing gradient problem and computationally faster that tanh and sigmoid.

4) Cost Computation: I have implement forward and backward propagation. To check whether model is learning we computed the cost. I made comparison with the predicted output and actual output and calculate cost. Here we computed the cross-entropy cost J, using the following formula:

$$-\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log\left(A^{[L](i)}\right) + (1 - y^{(i)})\log\left(1 - A^{[L](i)}\right))$$

where, m is the number of example in per batch the cost computed, y is the actual output.

5) Backward Propagation: If $g(.)$ is the activation function, during back propagation derivative of activation will be needed an can be computed using following eqn:

$$dZ^{[l]} = dA^{[l]} * g'(Z^{[l]})$$

The three outputs $(dW^{[l]}, db^{[l]}, dA^{[l]})$ are computed using the input $dZ^{[l]}$ using following equations:

$$dW^{[l]} = \frac{\partial \mathcal{L}}{\partial W^{[l]}} = \frac{1}{m}dZ^{[l]}A^{[l-1]T}$$

$$db^{[l]} = \frac{\partial \mathcal{L}}{\partial b^{[l]}} = \frac{1}{m}\sum_{i=1}^{m}dZ^{[l](i)}$$

$$dA^{[l-1]} = \frac{\partial \mathcal{L}}{\partial A^{[l-1]}} = W^{[l]T}dZ^{[l]}$$

6) Update Parameter: In this section I updated the parameters of the model, using gradient descent:

$$W^{[l]} = W^{[l]} - \alpha dW^{[l]}$$

$$b^{[l]} = b^{[l]} - \alpha db^{[l]}$$

where α is the learning rate. We split the whole dataset into many subsets called batches, and update the weights and biases on every batch iteration. And this accelerate the convergence of training.

III. RESULT

1) Result for Feed Forward Fully Connected Neural Network: I shuffled and split the data randomly into 80:20 ratio to be used as train and validation set. As we need to tune the number of hidden layer and number of nodes per hidden layer to be used, we didn't do k-fold cross-validation. Initialized learning rate at 0.001 as min_{α} and with per batch iteration t we updated the learning rate using following equation, where maximum learning rate 0.1 with decay factor k

$$\alpha = min_{\alpha} + (max_{\alpha} - min_{\alpha}) * \exp(-t/k)$$

TABLE I
PERFORMANCE FOR 3 HIDDEN LAYER NN ON MNIST DATA

Network configuration	Train Performance	Validation Performance
Hidden Layer 1 = 400 Hidden layer 2 = 400 Hidden layer 3 = 200	96.72	96.1

.

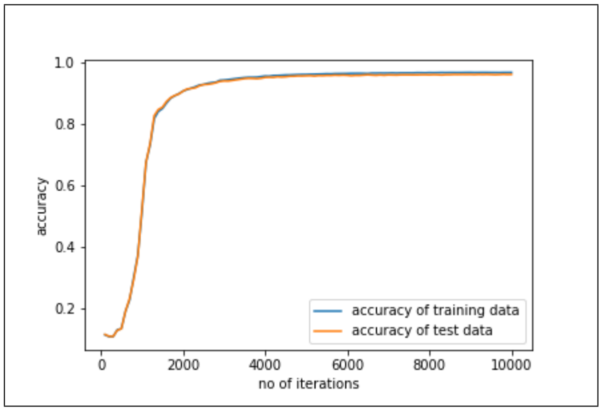

Fig. 1. Training and Validation Accuracy evaluation for 3 hidden layer fully connected feed forward NN

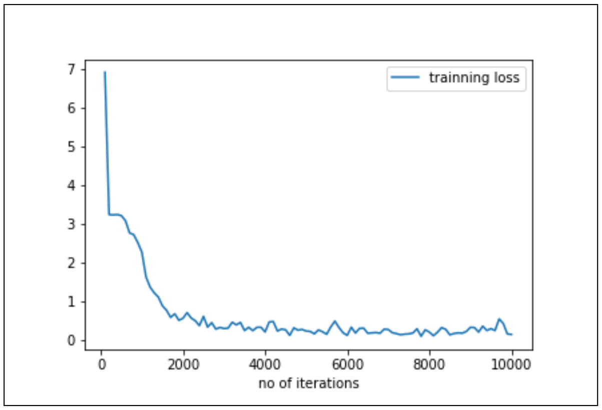

Fig. 2. Approximation Error of training data at each iteration

I sampled batch of 100 examples and updated parameter after each batch, and predication of train and validation set for each batch has shown in the the fig 1. and fig 1. shows the error computed in predicting real output at each iteration