Weekly Report 4 - Intro to Neural Network

Ganji Varshitha
AI20BTECH11009

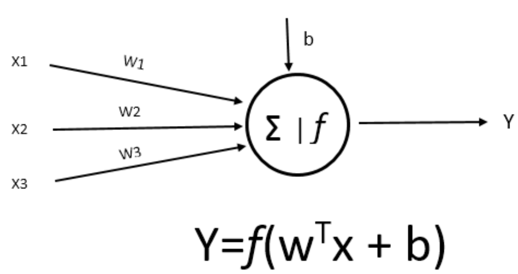
Introduction

Neural Networks also known as Artificial Neural Networks is one of the important tool in machine learning. It is the heart of deep learning models. The concept was inspired by human brain and the way neurons of the human brain function together to understand inputs from human senses.
It is used in supervised learning domain.

What are Neural Networks?

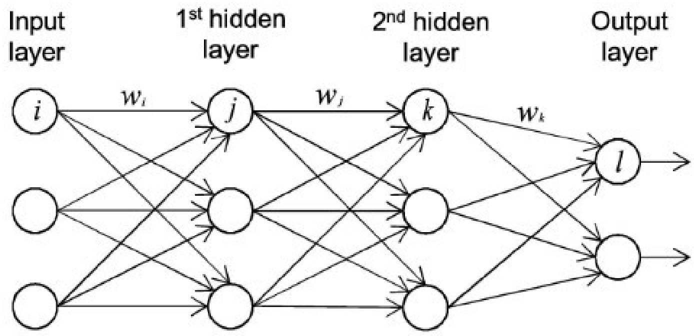
It is a system which uses a network of functions to understand and translate a data input of one form into a desired output.
It consists of node layers with 1 input layer, hidden layers and an output layer.
It is used in non linear classification as the decision boundary is not straight line.
Node can be seen as the following:

Figure 1: Neuron



If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, data is not passed to the next layer.
Neural Network creates a network of nodes so as to classify data. It is also known as multilayer perceptron.

Figure 2: Neural Network with 2 hidden layers



Neural Net Architecture

Let the output at node j,k,l be y_j, y_k, y_l respectively. Let the bias for hidden layer 1, hidden layer 2 and output layer be b_1, b_2, b_3.

$$y_j = f(\sum_i w_i x_i + b_1) \tag{1}$$

$$y_k = f(\sum_j w_j y_j + b_2) \tag{2}$$

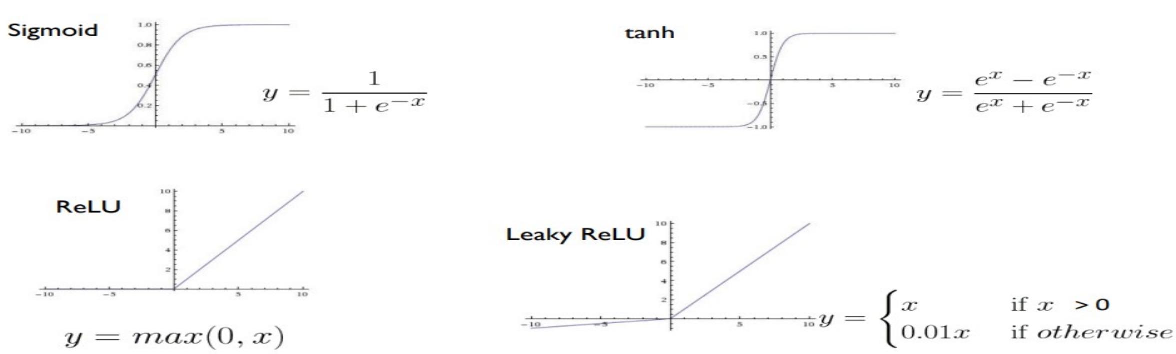
$$y_l = f(\sum_k w_k y_k + b_3) \tag{3}$$

Here, f(x) refers to the activation function. These functions are used to activate the node to pass through next layers. It adds the non linearity to the algorithm which is the sole purpose of classifying non linear data.

Activation functions

There are many activating function, most commonly used is ReLu(Rectified Linear Unit). Others include sigmoid, tanh, Leaky ReLu.

Figure 3: Activation functions with graph



Training Neural Network

For quantifying the performance of the Neural Network, we need cost function also known as loss.
There are many cost functions like Euclidean or squared loss and cross entropy loss.
Consider mean squared loss:

$$L = \sum_{i=1}^{N} \frac{1}{2} \|y_i - \hat{y}_i\|^2 \tag{4}$$

We use gradient descent to minimize the loss or error.
We need to calculate gradients(partial derivatives) w.r.t to the weight parameters to adjust them.
This is done by **Backpropagation** from outputs to hidden layers to input.
The parameter which controls how fast we train is **learning rate**. The equation goes:

$$w = w - \eta \frac{\partial L}{\partial w} \tag{5}$$

Rough Implementation

This is a python code where we train a feed forward model of 1 input layer, 1 hidden layer and 1 output layer.

Auxilary functions

```
#Defining Sigmoid Function
def sigmoid(Q):
  # It is an activation function which takes the input and return value
    between 0 and 1
  return 1/(1+np.exp(-Q))

#Defining mean squared error to measure performance
def error(y,y_hat):
  squared_error=np.sum((y-y_hat)**2)/y.shape[0] #formula is (output
   value - predicted value)^2/Number of samples
  return squared_error


#Computing sigmoid derivative
def sigmoid_derivative(B):
  return sigmoid(B)*(1-sigmoid(B))

```

Feed forward model

```
#Function to implement Feedforward model
#Passing input matrix, hidden weights matrix, hidden bias, output weights matrix, output bias through the model
def forward_model(x_train,w_h,w_out):
  #Computing the input at hidden layer
  Z=np.dot(x_train,w_h)
  #Passing the net input at hidden layer through activation function(sigmoid)
  Z_out=sigmoid(Z)
  Z_ones=np.ones((Z.shape[0],1))
  Z_in=np.concatenate((Z_ones,Z_out),axis=1)
  #Computing the input at output layer
  Y=np.dot(Z_in,w_out)
  #Activating the output layer
  Y_out=sigmoid(Y)
  return Z,Z_out,Y,Y_out
```

Mini-batch Gradient descent with backpropagation

```
#Training the mlp with back prop algorithm
def BACK_PROPAGATION_MLP(x_train,x_test,w_h,w_out):

  #Count of iterations of passing through the network
  epoch=100
  #We are iterating in minibatches hence making a list of indices of all the rows of input matrix X
  id=np.arange(x_train.shape[0])
  #Initialising 1D array to store testing error and training error after each epoch
  ERROR=np.empty((1,100))
  ERROR_TEST=np.empty((1,100))


  for i in range(epoch):
    #We are iterating in minibatches hence making a list of indices of all the rows of input matrix X
    id=np.arange(x_train.shape[0])

    #Taking the value of size of minibatch m=0.1*N
    m=int(N//10)

    for indices in range(0,id.shape[0],m):

      #Considering only m samples at a time, we declare indices of only size m at every iteration
      index_for_iter=id[indices:indices+m]
      #Passing through the network once and computing outputs at hidden layer and output layer
      z_h=forward_model(x_train[index_for_iter],w_h,w_out)[0]
```

```
    z_out=forward_model(x_train[index_for_iter],w_h,w_out)[1]
    y_h=forward_model(x_train[index_for_iter],w_h,w_out)[2]
    y_out=forward_model(x_train[index_for_iter],w_h,w_out)[3]



    E=y_out-output_train[index_for_iter] #computing difference between predicted label and ground truth label

    slope_out=sigmoid_derivative(y_h) #computing sigmoid derivative for output matrix

    slope_hidden=sigmoid_derivative(z_h) #computing sigmoid derivative for hidden layer

    grad_out=np.empty((3,1)) # Creating an empty array for gradients with respect to weights at output layer
    grad_hidden=np.empty((3,2)) #Creating an empty array for gradients with respect to weights at hidden layer

    #initialising values for gradients
    beta0=0
    beta1=0
    beta2=0
    alpha01=0
    alpha02=0
    alpha11=0
    alpha12=0
    alpha21=0
    alpha22=0
    for p in range(index_for_iter.shape[0]):
      beta0 += 2*E[p]*slope_out[p]
      beta1 += 2*E[p]*slope_out[p]*z_out[p,0]
      beta2 += 2*E[p]*slope_out[p]*z_out[p,1]
      alpha01 += 2*E[p]*slope_out[p]*w_out[1]*slope_hidden[p,0]*x_train[p,0]
      alpha02 += 2*E[p]*slope_out[p]*w_out[2]*slope_hidden[p,1]*x_train[p,0]

      alpha11 += 2*E[p]*slope_out[p]*w_out[1]*slope_hidden[p,0]*x_train[p,1]
      alpha21 += 2*E[p]*slope_out[p]*w_out[1]*slope_hidden[p,0]*x_train[p,2]

      alpha12 += 2*E[p]*slope_out[p]*w_out[2]*slope_hidden[p,1]*x_train[p,1]
      alpha22 += 2*E[p]*slope_out[p]*w_out[2]*slope_hidden[p,1]*x_train[p,2]


    grad_out[0,0]=beta0
    grad_out[1,0]=beta1
    grad_out[2,0]=beta2
```

```
            grad_hidden[0,0]=alpha01
            grad_hidden[0,1]=alpha02
            grad_hidden[1,0]=alpha11
            grad_hidden[2,0]=alpha21
            grad_hidden[1,1]=alpha12
            grad_hidden[2,1]=alpha22


            #Taking learning rate as gamma=0.05
            gamma=0.05
            #updating weights
            w_out-=gamma*grad_out
            w_h-=gamma*grad_hidden


        #Passing the training data through the network after 1 epoch
        Y1=forward_model(x_train,w_h,w_out)[3]
        ERROR[0,i]=error(output_train,Y1) #stores mean square error of training data at each iteration in array ERROR
        #Passing the testing data through the network after 1 epoch
        Y2=forward_model(x_test,w_h,w_out)[3]
        ERROR_TEST[0,i]=error(output_test,Y2) #stores mean square error of testing data at each iteration in array ERROR_TEST


    return ERROR,ERROR_TEST,w_h,w_out
```

Questions

1. Which of the following is not a choice in the algorithm?

   A. gradient descent method

   B. activation function

   C. back propagation

   D. learning rate

   Solution:
   C. back propagation

2. State few applications of neural networks.
   Solution:
   It is extensively applied in image recognition, speech recognition, and natural language processing.

3. What are types of neural networks?
   Solution:
   Artificial Neural Networks(ANN), Convolution Neural Network(CNN), Reinforcement Neural Network(RNN).

4. Pick the wrong option.
   Activation function need to be

   A. continuous

   B. decreasing

   C. differentiable

   Solution:
   B. decreasing. Since we need to compute gradient of activating functions w.r.t weights, it is mandatory that it is continuous, differentiable, non-decreasing.

5. Does back propagation algorithm learns a global optimal network with hidden layers?
   Solution:
   No, it does not reach global optima.