# Homework 2 three layer Neural Network

Tadesse Zemicheal

In this assignment I trained a vectorized implementation of three layer neural network on CIFAR training data. Later, the performance of the model is evaluated on separated testing data. The result shows both testing accuracy and objective loss of the model.

## 1. Prediction function and Forward layer function

The function predict evaluates both accuracy and average entropy loss prediction.

```
#compute accuracy and entropy loss prediction
def predict(self, x_test, y_test):
    z, l1, l2 = self.forward(x_test) #forward pass computes prediction and linear
transform
    accuray = 1 - np.mean(np.abs(np.round(z) - y_test.T))
    loss = self.evaluate_loss(y_test, z)
    return accuray, loss/len(y_test)
```

A vectorized implementation of backpropagation. The input are X =NxD matrix , Y = NX1 test label, Z=Nx1 score prediction from forward pass , l1 and l2 are output of layer one and layer two from forward pass. The method returns gradient of W1, W2 , b1 and b2.

```
"""
Backprop error to layers
"""
def backprop(self, X, Y,Z, l1, l2):
    N = len(Y)
    # error from output layer
    l2_error = Z-Y
    # gradient of output layer
    nonl_l1 =self.nonlinear(l1)
    dldw2 = nonl_l1.dot(l2_error).T /N  # gradient of w2

    # error from hidden unit layer
    l1_error = l2_error.dot(self.w2)
    dldw1_delta = l1_error.T * self.nonlinear(l1,deriv=True)
    dldw1 = dldw1_delta.dot(X)/N #gradient of input layer

    #gradient of bias
    grad_b1 = np.mean(dldw1_delta, axis=1, keepdims=True)
    grad_b2 = np.mean(l2_error,keepdims=True)

    return dldw1, dldw2, grad_b1, grad_b2
```

## Stochastic Mini Batch Gradient descent algorithm (SGD).

In each batch I compute gradient of weight and bias. Then momentum is updated by taking average from the batch gradients.

```
# takes training set and training label
#perform gradient update on a minibatch size of batch
#Gradient update is done using momentum.
def SGD(self, train_size, x_train, y_train):
    n_sample = range(train_size)
    np.random.shuffle(n_sample)
    # random shuffle in every epoch
    it = 0
    while it < train_size - self.minbatch:
        # Forward pass for all batch size  and compute error
        # update w1 and w2 based on the error size
        # for i in range(it,it+self.minbatch-1):

        x = x_train[it:self.minbatch + it, ]  # slice on batch of training examples
        y = y_train[it:self.minbatch + it]

        z, l1, l2 = self.forward(x)  # forward pass
        dw1, dw2, grad_b1, grad_b2 = self.backprop(x, y,z.T,l1, l2)  # backprop batch
examples

        # update momentum
        self.momentum_w1 = self.momentum * self.momentum_w1 - self.learning_rate * dw1
        self.momentum_w2 = self.momentum * self.momentum_w2 - self.learning_rate * dw2

        self.momentum_b1 = self.momentum * self.momentum_b1 + self.learning_rate *
grad_b1
        self.momentum_b2 = self.momentum * self.momentum_b2 + self.learning_rate *
grad_b2

        # update weight
        self.w2 = self.w2 + self.momentum_w2
        self.w1 = self.w1 + self.momentum_w1
        self.b1 = self.b1 + self.momentum_b1
        self.b2 = self.b2 + self.momentum_b2

        it += self.minbatch
```

## Experimental result

In this setting, I tried different configuration for the neural network. I fixed momentum 0.8 and changed the following settings.

Learning rate $\alpha$= 0.1, 0.01, 0.001 and 0.0001

Number of hidden unit = 50, 80, 100, 200,300

Batch size of = 30, 50, 120 and 250

The best performance of my result for 40 epoch

| Training accuracy | Testing Accuracy | Training loss | Testing loss |
|---|---|---|---|
| 96.21 | 83.85 | 0.129 | 0.433 |

# hidden unit: 300   Learning rate: 0.001   batch size: 30   momentum =0.9

Code description

The code is modularized into two classes and different methods. The 'Activation' class is for generating different activation function. The 'NueralNetwork' is the main network class. Inside, the NueralNetwork class, I have tried to separate the linear transform and gradient function into its separate modules.

*Activation class*: Factory class for activation units

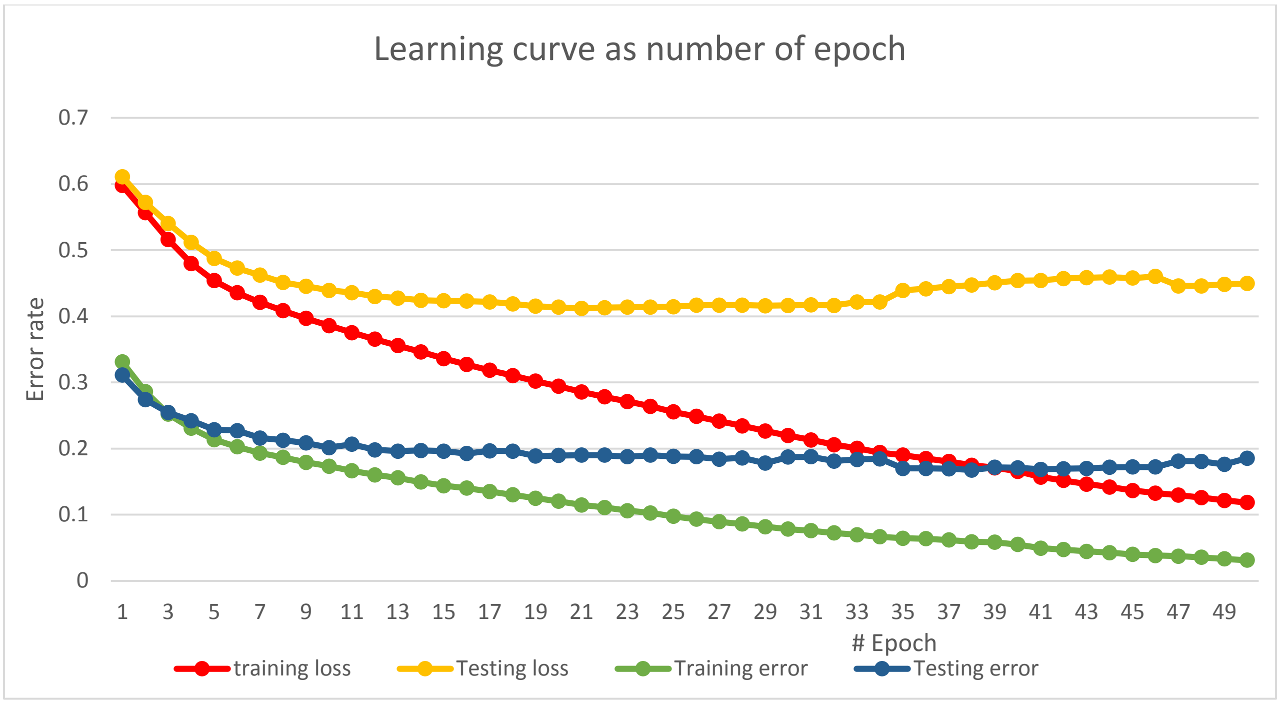
- Generates non-linear function and its derivative with different configuration.

*NeuralNework class*: main class for the 3-hidden layer unit. It consists of the following methods

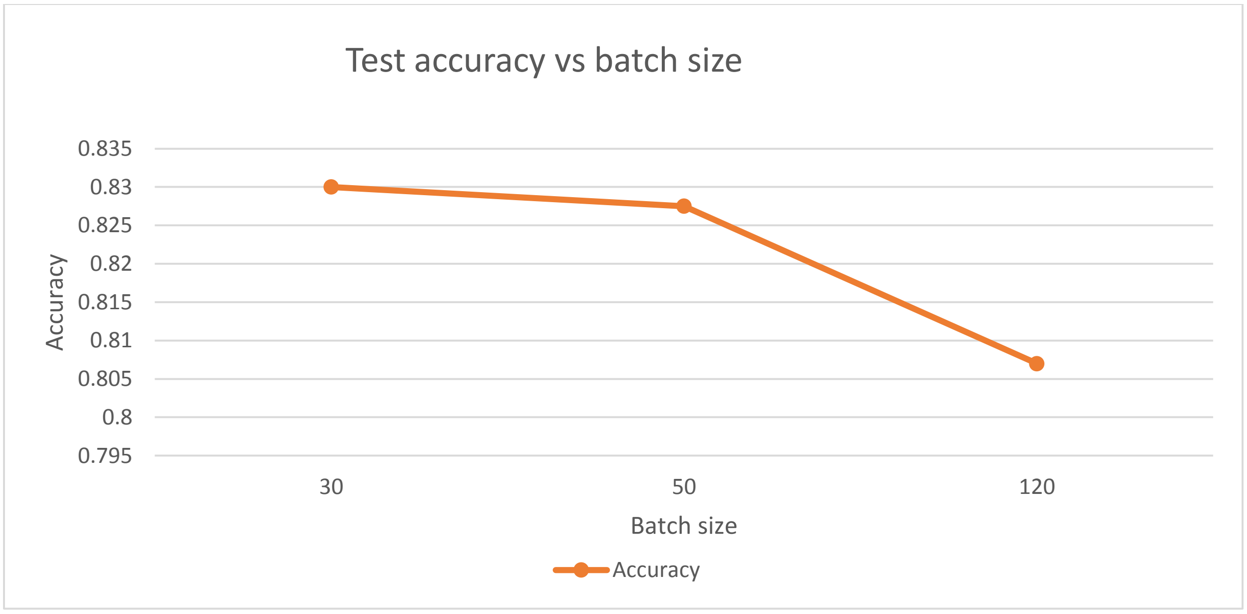
- *train* (self, x_train, y_train,x_test,y_test) : accepts training and testing data. Then learns model from the training data. Output training object, testing objective, training error and testing error to file.
- *SGD* : stochastic minibatch gradient descent function
- Forward : forward pass for class prediction
- *Backprop* (self,X,Y,Z, l1, l2): backpropagation function takes batch size examples, label and prediction score, layer1 and layer2 value. Returns gradient of weights and biases.
- Predict(self,x_test,y_test): takes training examample and its labels. It returns accuracy and objective loss (entropoy loss) of the function
- *evaluate_loss*(self,score,pred): returns entropy loss of prediction from true score.

Training Monitoring:

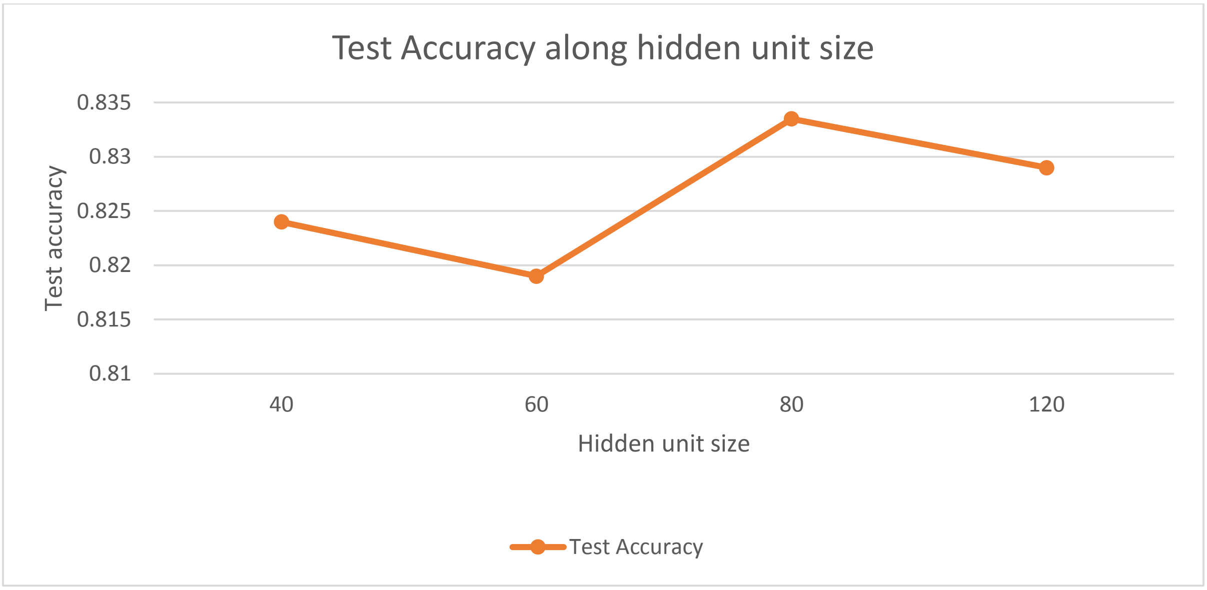
The graph below shows training and testing error along testing and training objective loss. Clearly, training error and testing error is decreasing with increasing number of epoch. However, at same level of epoch, testing error doesn't decrease much. In the other hand, the objective loss of training data is gradually decreasing but for testing data the entropy loss is slowly decreasing.



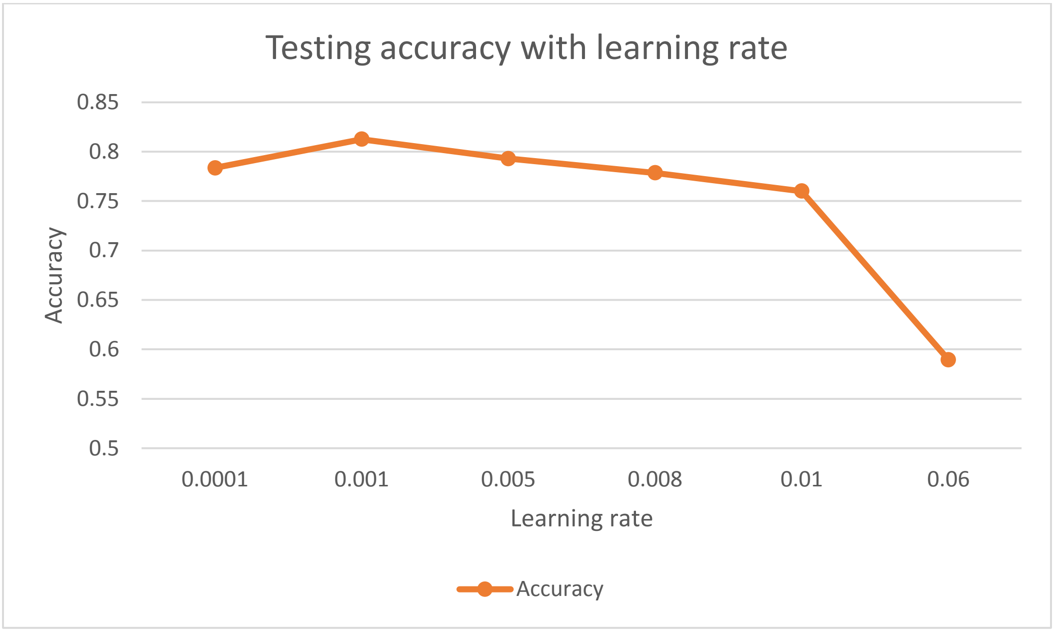
<u>Tuning parameters:</u>



Increasing batch size looks to decrease accuracy of testing data.   For large batch size of 120 the model accuracy tends to decrease gradually.



Learning rate influence on the testing accuracy



The performance of neural network increases with increasing number of epoch. Large learning rate makes the gradient to overshoot and results in numeric overflow and bad performance.   Learning rate between 0.001 and 0.005 looks good compared to smaller and larger learning rate.

Influence on the number of hidden unit, increasing size increase the performance. However, in my experiment larger number of hidden unit 120 have lower performance to medium size 80 .