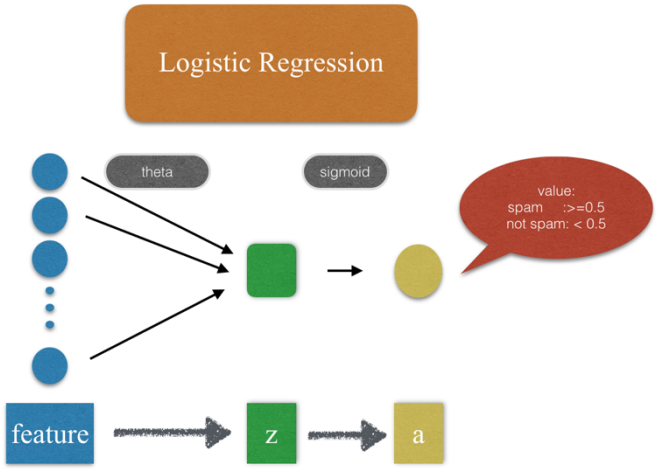
[Machine Learning HW2]　Logistic Regression　R04921006 謝仲凱

這次的作業是來分辨信件是否為垃圾郵件 ， Data是由助教提供的。

[Train]./train.sh spam_train.csv model [Test]./test.sh model spam_test.csv prediction.csv

1. **(1%) Logistic regression function.**



```
def compute_grad(X, y, theta ):
        m = X.shape[0]
        theta = np.reshape(theta, (len(theta)),1 )
        h = sigmoid(X.dot(theta))

        # m: number of data, k number of feature
        #grad: 1xk
        # grad = np.transpose( (1.0/m)*(h - y).dot(X))
        grad = ( - 1.0/m) * np.transpose( (y -  h).dot(X)

        return grad

#**********************
#-----Gradient (adagrad)-----
#**********************

def gradient_descent(X, y, X_validation, y_validation, theta, learning_rate, epochs):
        # ------------------------------------
        # Performs gradient descent to learn theta
        # by taking epochs gradient steps with learning_rate
        # ------------------------------------
        X_tmp = X
        n = X.shape[0]
        bias_ones = np.ones( (n,1) , dtype='float32')
        X_tmp = np.append(X, bias_ones, axis=1)

        X_validation_tmp = X_validation
        n_validation = X_validation.shape[0]
        bias_ones = np.ones( (n_validation,1) , dtype='float32')
        X_validation_tmp = np.append(X_validation, bias_ones, axis=1)

        g_history = np.zeros( [len(theta), epochs] )
        cost_training_history = np.zeros(shape=(epochs, 1))
        cost_validation_history = np.zeros(shape=(epochs, 1))

        # print("X_validation.shape"  + str(X_validation.shape))
        # print("X_validation_tmp.shape"  + str(X_validation_tmp.shape))

        for i in range(epochs):
                predictions = X_tmp.dot(theta)
                theta_size = theta.shape[0]



                g = compute_grad(X_tmp, y, theta)[1,:]

                g_history[ : , i ] = g
                adagrad_coeff = compute_adagrad_coeff(g_history)
                g = g.reshape( g.shape[0] , 1)
                adagrad_coeff = adagrad_coeff.reshape(adagrad_coeff.shape[0] , 1)

                theta = theta - learning_rate * g/adagrad_coeff

                print("iteration numbers : " + str(i))
                # print("theta : " + str(theta))
                print(" [cost] - training data   : " + str( compute_cost(X_tmp, y, theta)))
                compute_accuracy(X_tmp, y, theta)
                print(" [cost] - validation data : " + str( compute_cost(X_validation_tmp, y_validation, theta)))
                compute_accuracy(X_validation_tmp, y_validation, theta)
                cost_training_history[i, 0] = compute_cost(X_tmp, y, theta)
                cost_validation_history[i,0] = compute_cost(X_validation_tmp,y_validation,theta)
                validation_accuracy =  compute_accuracy(X_validation_tmp, y_validation, theta)

        return theta, cost_training_history, cost_validation_history , validation_accuracy

def gradient_descent_minibatch(X, y, X_validation, y_validation, theta, learning_rate, epochs,batch_size):
```

Cost function: Cross Entropy

硬生生的比Square Error 還厲害

比較不會卡在saddle points

Parameter update formula:

$$w_i \leftarrow w_i - \eta \sum_n -\left(\hat{y}^n - f_{w,b}(x^n)\right) x_i^n$$

- **Data**

  Training Data 分成：

  Training Set = (9/10)筆 Training Data ; Validation Set = (1/10)筆 Training Data Validation 非常的重要，因為它才是能正比於 Testing Data 正確率的標準，因此判斷 function 的好壞我會以 validation 為準則。

- **Initial parameters**

  我認為參數值越接近 0 越好，因為這樣受到 Noise 的影響會越少因此我的初始參數的 mean 為 0。 Theta = 隨機的高斯函數值 (mu = 0, sigma=0.001)

- **Learning Rate**

  大概試了一下，大約在 (4.5 * 10^-3) performance 最好

- **[Gradient Descent] mini batch vs full**

  看完全部更新一次： Validation performance 都只有在 90% 左右

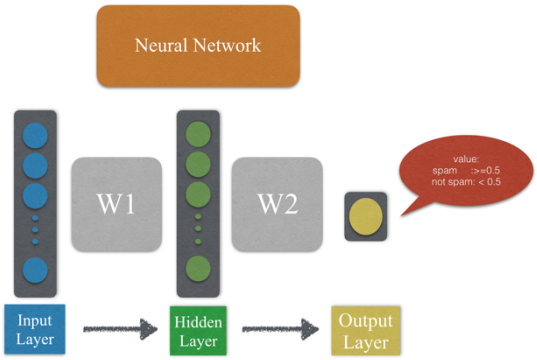
  mini batch : Validation performance 可以到達92% 左右

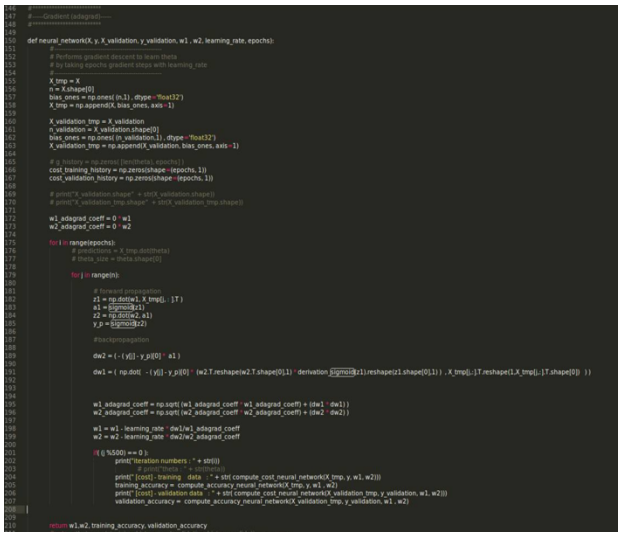
  之所以有這樣的結果，我認為是因為如果有使用mini batch 每看完 batch_size = (3,5,8,10 一起試) 筆資料就更新一次，可以使parameters 更符合資料的特性，使performance 得以提升。

(1%) Describe your another method, and which one is best.

Neural Network

(Hidden Layer 3~15個neuron 一起試)





Cost function: Cross Entropy

Parameter update formula: Backward Propagation (參考老師去年的講義)

- **Data, Initial parameters, Learning Rate 設定方式與上述差不多**
- **Training Method (判斷function 夠不夠好):**

  與自己上次做法不同的地方是這次有使用accuracy threshold

  if(train_acc >= train_acc_th and validation_acc>=validation_acc_th)

  才會判定為是不錯的function，如果不夠好就重設initial parameter :因為我認為有好的開始是成功的一半，有好的initial parameter 分佈會讓accuracy 提高。

- **[Public Score] Neural Network(0.94667) VS Logistic Regression(0.9233)**

  在這次的功課中，我的Logistic Regression 只能勉強比baseline多對一題，我認為是因為 Logistic Regression 只能做線性的分類，然而這次的spam並非單純的線性model就可以分得很好，會有些data剛好就是無法被線性正確分類因此Logistic Regression的 accuracy 比較差。