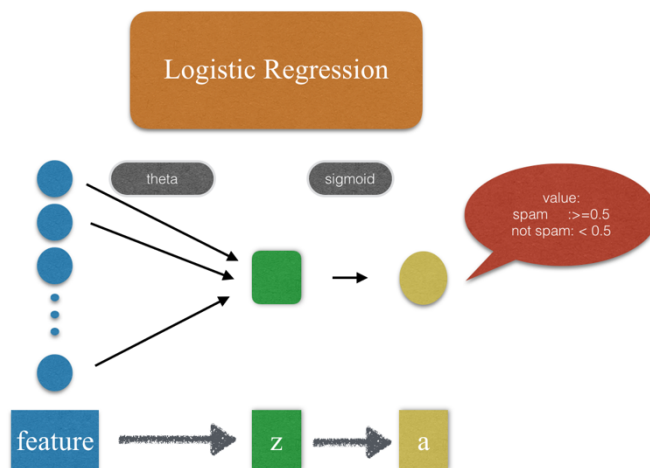


## [Machine Learning HW2] Logistic Regression R04921006 謝仲凱

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.



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:

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

- **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

```
118 def compute_grad(X, y, theta):
119     m = X.shape[0]
120     theta = np.reshape(theta, (len(theta), 1))
121     h = Sigmoid(X.dot(theta))
122
123     # m: number of data, k: number of feature
124     # grad: 1xk
125     # grad = -np.transpose(1.0*(y-h).dot(X))
126     grad = (-1.0/m) * np.transpose((y - h).dot(X))
127
128     return grad
129
130
131 #-----Gradient (adagrad)-----
132 #-----
133
134
135 def gradient_descent(X, y, X_validation, y_validation, theta, learning_rate, epochs):
136     #
137     # Performs gradient descent to learn theta
138     # by taking epochs gradient steps with learning_rate
139     #
140     X_tmp = X
141     n = X.shape[0]
142     bias_ones = np.ones((n, 1), dtype='float32')
143     X_tmp = np.append(X, bias_ones, axis=1)
144
145     X_validation_tmp = X_validation
146     n_validation = X_validation.shape[0]
147     bias_ones = np.ones((n_validation, 1), dtype='float32')
148     X_validation_tmp = np.append(X_validation, bias_ones, axis=1)
149
150     g_history = np.zeros((len(theta), epochs))
151     cost_training_history = np.zeros(shape=(epochs, 1))
152     cost_validation_history = np.zeros(shape=(epochs, 1))
153
154     # print("X_validation shape: " + str(X_validation.shape))
155     # print("X_validation tmp shape: " + str(X_validation_tmp.shape))
156
157     for i in range(epochs):
158         predictions = X_tmp.dot(theta)
159         theta_size = theta.shape[0]
160
161
162         g = compute_grad(X_tmp, y, theta)[1:]
163
164         g_history[:, i] = g
165         adagrad_coeff = compute_adagrad_coeff(g_history)
166         g = g.reshape(g.shape[0], 1)
167         adagrad_coeff = adagrad_coeff.reshape(adagrad_coeff.shape[0], 1)
168         theta = theta - learning_rate * g/adagrad_coeff
169
170         print("Iteration numbers: " + str(i))
171         # print("theta: " + str(theta))
172
173         print("cost: training data: " + str(compute_cost(X_tmp, y, theta)))
174         compute_accuracy(X_tmp, y, theta)
175         print("cost: validation data: " + str(compute_cost(X_validation_tmp, y_validation, theta)))
176         compute_accuracy(X_validation_tmp, y_validation, theta)
177         cost_training_history[i] = compute_cost(X_tmp, y, theta)
178         cost_validation_history[i] = compute_cost(X_validation_tmp, y_validation, theta)
179         validation_accuracy = compute_accuracy(X_validation_tmp, y_validation, theta)
180
181     return theta, cost_training_history, cost_validation_history, validation_accuracy
182
183
184 def gradient_descent_minibatch(X, y, X_validation, y_validation, theta, learning_rate, epochs, batch_size):
```

- **[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

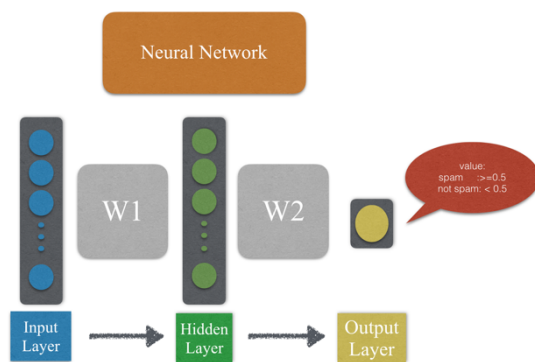
之所以有這樣的結果，我認為是因為如果有使用mini batch 每看完

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 )



```

def neural_network(x, y_validation, y_validation, w1, w2, learning_rate, epochs):
    # Training gradient descent to learn weights
    # by training epochs gradient steps with learning_rate

    X_train = x
    bias_ones = np.ones((n1, 1), dtype='float32')
    X_train = np.append(X_train, bias_ones, axis=1)

    X_validation = x_validation
    bias_ones = np.ones((n2, 1), dtype='float32')
    X_validation = np.append(X_validation, bias_ones, axis=1)

    # y_validation = y_validation

    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('y_validation shape' + str(y_validation.shape))

    w1_adapted_coeff = 0
    w2_adapted_coeff = 0

    for i in range(epochs):
        # predictions = X_train.dot(weights)
        # cost = cost_function(predictions, y_train)

        # Forward propagation
        z1 = np.dot(w1, X_train)
        a1 = sigmoid(z1)
        z2 = np.dot(w2, a1)
        y2 = sigmoid(z2)

        # Backward propagation
        dw1 = (- (y1 - y2) * y2 * (1 - y2)) * (w2.T.reshape(w2.T.shape[0], 1) * y_validation - y2).reshape(1, X_train[1].shape[0])
        dw2 = (- (y2 - y_validation) * y_validation * (1 - y_validation)) * X_train[1].reshape(1, X_train[1].shape[0])

        w1_adapted_coeff = np.subtract(w1, dw1 * learning_rate)
        w2_adapted_coeff = np.subtract(w2, dw2 * learning_rate)

        w1 = w1_adapted_coeff
        w2 = w2_adapted_coeff

        # print('Iteration numbers: ' + str(i))

        print('cost: training_data ' + str(cost_function(neural_network(X_train, w1, w2)))
        training_accuracy = compute_accuracy(neural_network(X_train, w1, w2))
        print('cost: validation_data ' + str(cost_function(neural_network(X_validation, w1, w2)))
        validation_accuracy = compute_accuracy(neural_network(X_validation, w1, w2))

    return w1, w2, training_accuracy, validation_accuracy
    
```

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，如果不够好就重設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。