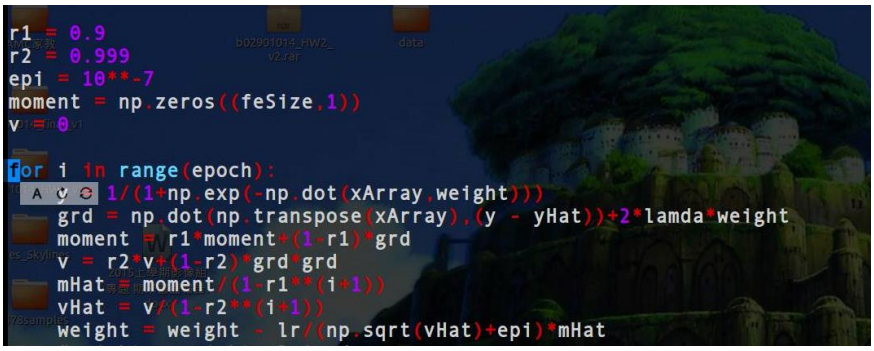
Machine Learning HW2 report b02901014 王崇勳

1. Logistic regression



```
r1 = 0.9
r2 = 0.999
epi = 10**-7
moment = np.zeros((feSize,1))
v = 0

for i in range(epoch):
    1/(1+np.exp(-np.dot(xArray,weight)))
    grd = np.dot(np.transpose(xArray),(y - yHat))+2*lamda*weight
    moment = r1*moment+(1-r1)*grd
    v = r2*v+(1-r2)*grd*grd
    mHat = moment/(1-r1**(i+1))
    vHat = v/(1-r2**(i+1))
    weight = weight - lr/(np.sqrt(vHat)+epi)*mHat
```

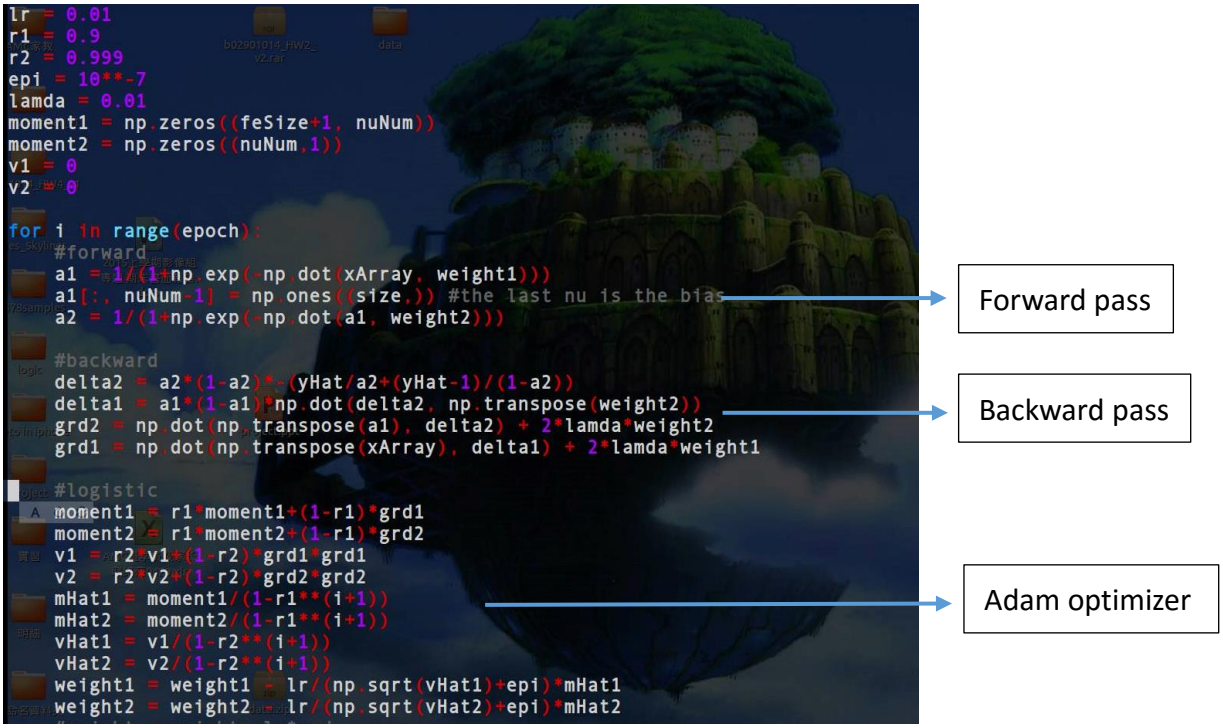
Figure.1

(1.) 除了使用基本的 logistic regression 之外，為了能讓 gradient descent 更有效率，我使用 Adam (Adaptive Moment Estimation) optimizer 來加速。其中 r1, r2 為 Adam 會使用到的兩個 forgetting factor。

(2.) 除了 Adam 外還加上了 regularization，λ = 0.01。

(3.) 實作中使用的 learning rate 為 0.01，20000 個 epoch 以後 loss 就會降到 0.19 附近。

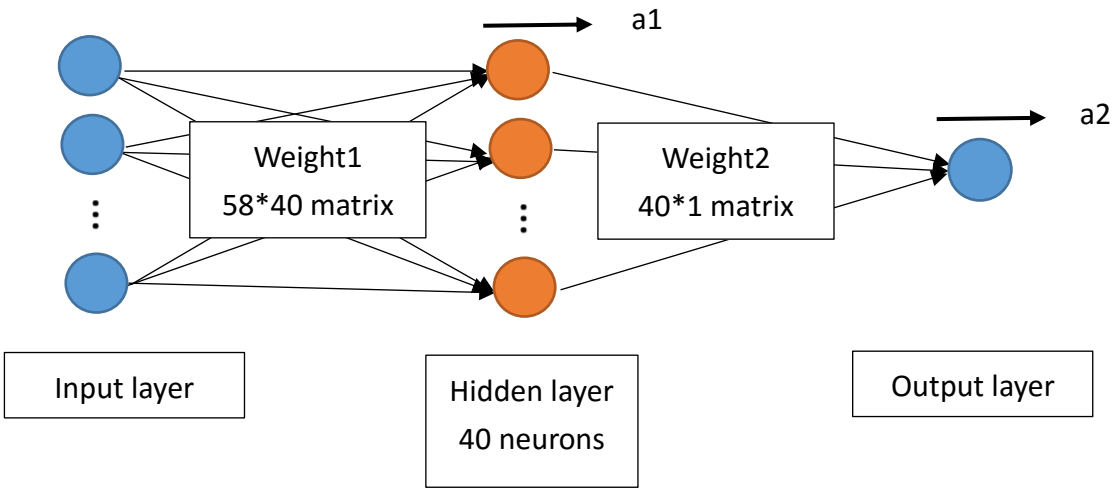
2. Neural network




```
lr = 0.01
r1 = 0.9
r2 = 0.999
epi = 10**-7
lamda = 0.01
moment1 = np.zeros((feSize+1, nuNum))
moment2 = np.zeros((nuNum,1))
v1 = 0
v2 = 0

for i in range(epoch):
    #forward
    a1 = 1/(1+np.exp(-np.dot(xArray, weight1)))
    a1[:, nuNum-1] = np.ones((size,)) #the last nu is the bias
    a2 = 1/(1+np.exp(-np.dot(a1, weight2)))

    #backward
    delta2 = a2*(1-a2)*-(yHat/a2+(yHat-1)/(1-a2))
    delta1 = a1*(1-a1)*np.dot(delta2, np.transpose(weight2))
    grd2 = np.dot(np.transpose(a1), delta2) + 2*lamda*weight2
    grd1 = np.dot(np.transpose(xArray), delta1) + 2*lamda*weight1

    #logistic
    moment1 = r1*moment1+(1-r1)*grd1
    moment2 = r1*moment2+(1-r1)*grd2
    v1 = r2*v1+(1-r2)*grd1*grd1
    v2 = r2*v2+(1-r2)*grd2*grd2
    mHat1 = moment1/(1-r1**(i+1))
    mHat2 = moment2/(1-r1**(i+1))
    vHat1 = v1/(1-r2**(i+1))
    vHat2 = v2/(1-r2**(i+1))
    weight1 = weight1 - lr/(np.sqrt(vHat1)+epi)*mHat1
    weight2 = weight2 - lr/(np.sqrt(vHat2)+epi)*mHat2
```

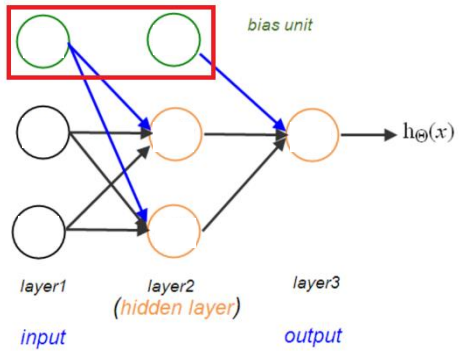
Figure.2

(1.) 第二個方法我使用一層的 neural network。示意圖如下：



此方法的 input 為 4001 封信的 57 個 features，此外 bias 也併入 input 一起進入 neural network，因此最終得到的 input 是 4001*58 的 matrix。

(2.) 從網路上找到把 bias 併入 input 的方法，需要處理 hidden layer 以及 input layer 之間的連結，如左圖所示。屬於 bias 的 neuron 只會跟 output layer 有連結，因此實作的時候會將 hidden layer 的 output a1 的最後一個 column 都設成 1。



(3.) Neural network 利用 backpropagation 來找出 layer 間的 gradient。Gradient descent 的部分一樣使用 Adam optimizer 來加速找到 global/local minimum。

(4.) 除了 Adam 外還加上了 regularization，λ = 0.01。

(5.) 實作中使用的 learning rate 為 0.01，1000 個 epoch。

Logistic regression vs. Neural network

從 Kaggle 上的結果來看，Logistic regression 得到的最好結果為 0.92667，Neural network 得到的最好結果為 0.95667，很明顯 Neural network 的 performance 較佳。另外 Neural network 只需要經過 1000 個 epoch 就可以得到很好的準確率，比起 Logistic 所需時間較短。

3.

Adam, Adagrad 比較

(1.)從 Table.1 中可以發現雖然 Adagrad 的 learning rate 較大，但是兩種方法在 Kaggle 上面測到的正確率一樣，原因可能是因為 Adagrad 雖然在一開始的時候能讓 gradient descent 較有效率，但是因為一直在累積 gradient 的關係，所以到後面 gradient decent 的速度非常慢。若是使用 Adam，只要遞迴 20000 次就可以得到很小的 loss。

	Original	Adagrad	Adam
Epoch	500000	500000	20000
Learning rate	10^-8	0.0001	0.01
Loss	0.29875034	0.27493049	0.1904659
Kaggle score	0.92333	0.92333	0.92667

Table.1

(2.) 從 Figure.3 可以看出來加上 Adam 以後 loss 下降的速率明顯比另外兩者快很多。

(3.) Figure.3 中加上 Adagrad 的方法一開始 loss 有先些微上升在下降，可能是因為 learning rate 太大的原因。從 Figure.4 中可以看出來 Adagrad 可以使 loss 一開始下降較快，但是到了後面兩者下降速率差不多。

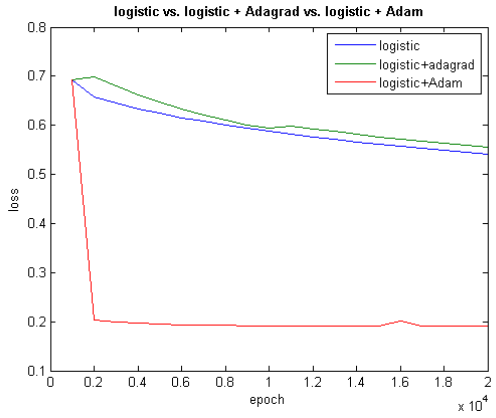


Figure.3

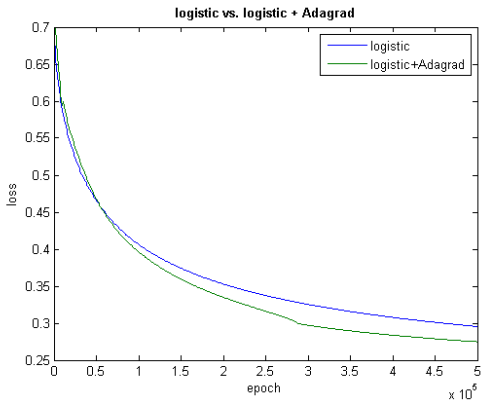


Figure.4

Batch 比較

在用 logistic regression 時嘗試使用不同的 batch size，發現當 batch size 越小，在 Kaggle 上得到的正確率越低。原因可能是因為 training data 本來就不大(4001 筆)，把資料分得太細反而沒辦法得到一個整體性的 model，因此其實沒有必要 batch。Table.2 為測試的數據，皆是使用 Adam，r1 = 0.9, r2 = 0.999, epsilon = 10^-7, learning rate = 0.01。

Batch size	10	100	1000	
Epoch	1000	1000	1000	20000
Kaggle score	0.83667	0.91667	0.92333	0.92667

Table.2

不同 neuron 數

嘗試在 hidden layer 使用不同的 neuron 數，發現 40 neuron 得到的結果最好，也是 Kaggle 上面最好的成績。50 個 neuron 可能因為 overfit 的原因，所以正確率較低。

Neural number	30	40	50
Kaggle score	0.943333	0.956667	0.94667

Table.3