

Deep Learning (Homework 1)

Prob1

我的設計為: 首先將輸入的 training data $\mathbf{x}^1 = [x_1^1 x_2^1 x_3^1 x_4^1 x_5^1 x_6^1]$ 經過第 2 層網路，網路中包括 weight 與 bias，接

著會到達第 2 層 $\mathbf{x}^2 = [x_1^2 x_2^2 x_3^2 x_4^2 x_5^2 x_6^2]$ ，我還是給它 6 個節點，希望藉由較寬的網路，可以有更多的參數更新

空間，在經過第 3 層 weight 與 bias，會到達第 3 層 $\mathbf{x}^3 = [x_1^3 x_2^3 x_3^3 x_4^3]$ ，最後到第 4 層 $\mathbf{y}^1 = [y_1^1 y_2^1]$ 也就是最後一

層。但現在題目是要求生存與死亡的機率，也就是說機率應該小於 1，並且兩者加總為 0，所以我在最後一

層再加上了 soft max 層， $\hat{\mathbf{y}}^1 = [\hat{y}_1^1 \hat{y}_2^1] = \left[\frac{e^{y_1^1}}{e^{y_1^1} + e^{y_2^1}} \frac{e^{y_2^1}}{e^{y_1^1} + e^{y_2^1}} \right]$ 。目前所計算的只是 1 個 mini-batch 中的其中 1 筆

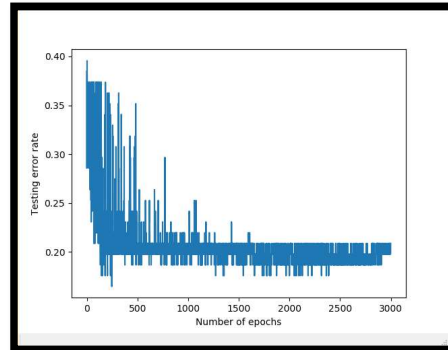
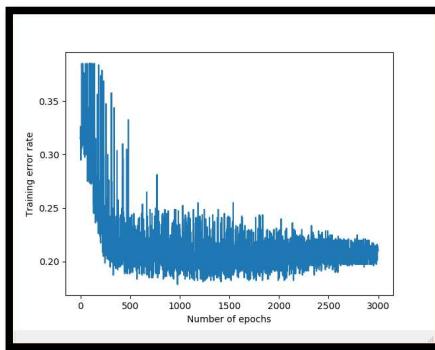
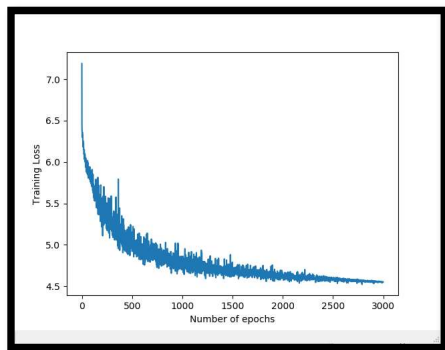
資料的輸出，接著我會計算所有 mini-batch 中的資料，在此處我把全部 training data 共 800 筆分成 80 個 mini-batch，每個 batch 有 10 個 data，即 batch size=10。總 cross entropy

$$E = -\left(t_1^1 \ln \hat{y}_1^1 + t_2^1 \ln \hat{y}_2^1\right) - \left(t_1^2 \ln \hat{y}_1^2 + t_2^2 \ln \hat{y}_2^2\right) \cdots - \left(t_1^{10} \ln \hat{y}_1^{10} + t_2^{10} \ln \hat{y}_2^{10}\right)$$

完整的 Forward 與 Backward 會在 prob2 做推導

總結這次測試的參數如下:

節點: [6,6,4,2+maxout], Learning rate=0.001, Training data 800, Testing data 91, Batch size 10, Iteration 3000 epoch
執行結果如下:



PS. 其中我的 Training Loss 為整個 mini-batch 的全部 Cross entropy，若要表現如範例的圖，可以將全部的值除以 10，也就是我每個 mini-batch 的資料數量，便是 Average Training Loss。

由結果可以看到 Average Training Loss 大概收斂到 0.45，Training error rate 與 Testing error rate 大概都收斂到 0.2 附近。

Prob2

這次測試的參數如下：

節點: [6,3,3,2+maxout],

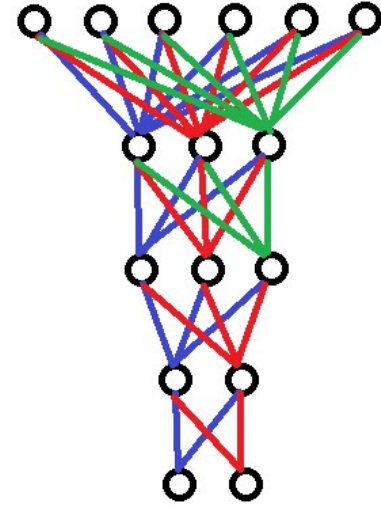
Learning rate=0.001,

Training data 800,

Testing data 91,

Batch size 10,

Iteration 10000 epoch



Forward 部分:

$$\mathbf{x}^2 = \begin{bmatrix} x_1^2 & x_2^2 & x_3^2 \end{bmatrix} = \mathbf{x}^1 \mathbf{w}^2 + \mathbf{b}^2 = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & x_4^1 & x_5^1 & x_6^1 \end{bmatrix} \begin{bmatrix} w_{11}^2 & w_{21}^2 & w_{31}^2 \\ w_{12}^2 & w_{22}^2 & w_{32}^2 \\ w_{13}^2 & w_{23}^2 & w_{33}^2 \\ w_{14}^2 & w_{24}^2 & w_{34}^2 \\ w_{15}^2 & w_{25}^2 & w_{35}^2 \\ w_{16}^2 & w_{26}^2 & w_{36}^2 \end{bmatrix} + \begin{bmatrix} b_1^2 \\ b_2^2 \\ b_3^2 \\ b_4^2 \\ b_5^2 \\ b_6^2 \end{bmatrix}$$

$$\mathbf{x}^3 = \begin{bmatrix} x_1^3 & x_2^3 & x_3^3 \end{bmatrix} = \mathbf{x}^2 \mathbf{w}^3 + \mathbf{b}^3 = \begin{bmatrix} x_1^2 & x_2^2 & x_3^2 \end{bmatrix} \begin{bmatrix} w_{11}^3 & w_{21}^3 & w_{31}^3 \\ w_{12}^3 & w_{22}^3 & w_{32}^3 \\ w_{13}^3 & w_{23}^3 & w_{33}^3 \end{bmatrix} + \begin{bmatrix} b_1^3 \\ b_2^3 \\ b_3^3 \end{bmatrix}$$

$$\mathbf{y}^1 = \begin{bmatrix} y_1^1 & y_2^1 \end{bmatrix} = \mathbf{x}^3 \mathbf{w}^4 + \mathbf{b}^4 = \begin{bmatrix} x_1^3 & x_2^3 & x_3^3 \end{bmatrix} \begin{bmatrix} w_{11}^4 & w_{21}^4 \\ w_{12}^4 & w_{22}^4 \\ w_{13}^4 & w_{23}^4 \end{bmatrix} + \begin{bmatrix} b_1^4 \\ b_2^4 \\ b_3^4 \end{bmatrix}$$

$$\hat{\mathbf{y}}^1 = \begin{bmatrix} \hat{y}_1^1 & \hat{y}_2^1 \end{bmatrix} = \begin{bmatrix} \frac{e^{y_1^1}}{e^{y_1^1} + e^{y_2^1}} & \frac{e^{y_2^1}}{e^{y_1^1} + e^{y_2^1}} \end{bmatrix}$$

$$E = -\left(t_1^1 \ln \hat{y}_1^1 + t_2^1 \ln \hat{y}_2^1\right) - \left(t_1^2 \ln \hat{y}_1^2 + t_2^2 \ln \hat{y}_2^2\right) \cdots - \left(t_1^{10} \ln \hat{y}_1^{10} + t_2^{10} \ln \hat{y}_2^{10}\right)$$

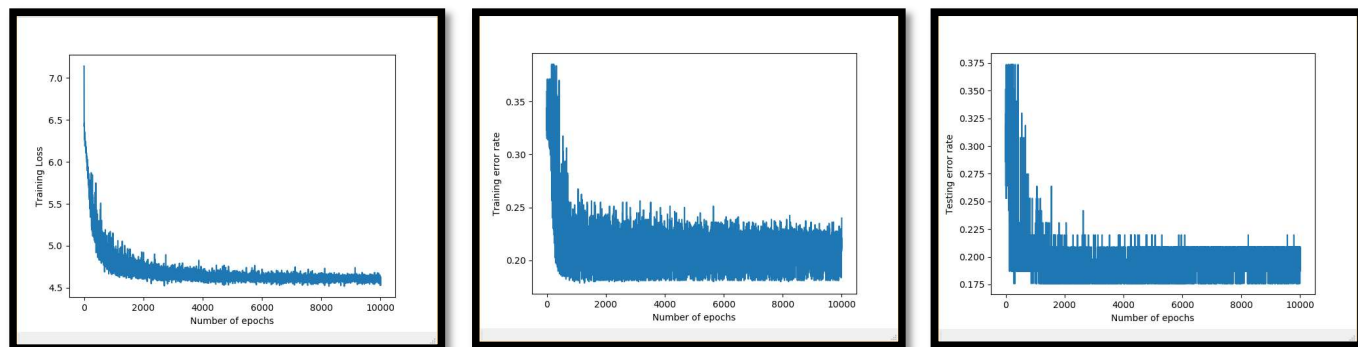
Backward 部分:

$$\frac{\partial E}{\partial y^1} = \left[\frac{\partial E}{\partial y_1^1} \frac{\partial E}{\partial y_2^1} \right] = \left[t_1^1 (\hat{y}_1^1 - 1) + t_2^1 \hat{y}_1^1, t_1^1 \hat{y}_2^1 + t_2^1 (\hat{y}_2^1 - 1) \right]$$

$$\frac{\partial E}{\partial w^4} = (\mathbf{x}^3)^T \frac{\partial E}{\partial y^1} \quad \frac{\partial E}{\partial x^3} = \frac{\partial E}{\partial y^1} (\mathbf{w}^4)^T \quad \frac{\partial E}{\partial w^3} = (\mathbf{x}^2)^T \frac{\partial E}{\partial x^3} \quad \frac{\partial E}{\partial x^2} = \frac{\partial E}{\partial x^3} (\mathbf{w}^3)^T \quad \frac{\partial E}{\partial w^2} = (\mathbf{x}^1)^T \frac{\partial E}{\partial x^2}$$

$$\frac{\partial E}{\partial b^4} = \frac{\partial E}{\partial y^1} \frac{\partial y^1}{\partial b^4} = \frac{\partial E}{\partial y^1} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \frac{\partial E}{\partial b^3} = \frac{\partial E}{\partial x^3} \frac{\partial x^3}{\partial b^3} = \frac{\partial E}{\partial x^3} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \frac{\partial E}{\partial b^2} = \frac{\partial E}{\partial x^2} \frac{\partial x^2}{\partial b^2} = \frac{\partial E}{\partial x^2} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

於是我將每筆 **batch** 當中的資料依序丟入 **Forward** 當中去計算，可以得到 $\hat{\mathbf{y}}^1 = [\hat{y}_1^1 \hat{y}_2^1]$ ，若其中的 \hat{y}_1^1 大於 0.5，就表示判斷結果為此人生存，反之則此人被判斷為死亡。每筆資料都可以用 **Backward** 算出要更新的梯度 **gradient**，我把這個 **batch** 的 10 個梯度做平均，每做完整個 **batch** 我會把 **weight** 與 **bias** 參數做 1 次更新，所以 1 個 **epoch** 共 80 個 **mini-batch** 就會參數更新 80 次。下圖為我做了 10000 個 **epoch** 的結果：



PS. 其中我的 **Training Loss** 為整個 **mini-batch** 的全部 **Cross entropy**，若要表現如範例的圖，可以將全部的值除以 10，也就是我每個 **mini-batch** 的資料數量，便是 **Average Training Loss**。

由結果可以看到 **Average Training Loss** 大概收斂到 0.45，**Training error rate** 與 **Testing error rate** 大概都收斂到 0.2 附近。

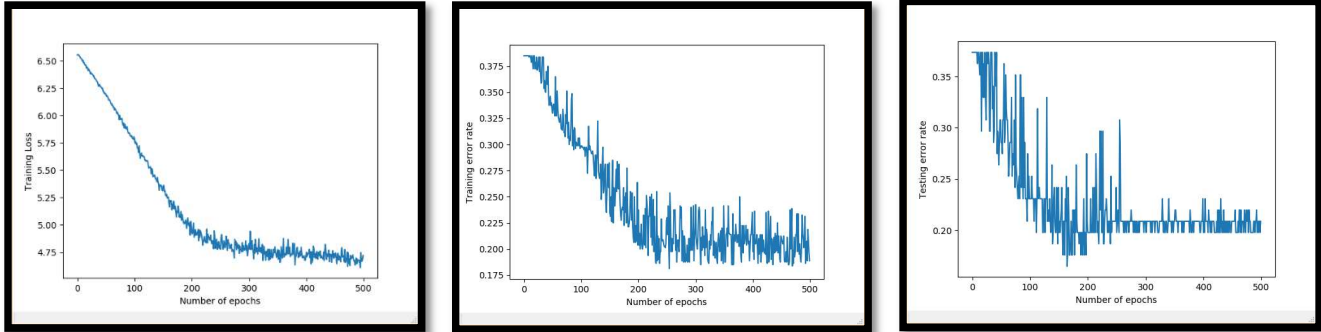
Prob3

這次測試的參數如下：

節點: [6,3,3,2+maxout], Learning rate=0.001

Training data 800, Testing data 91, Batch size 10, Iteration 500 epoch,

在一般的狀況下(沒有做任何 normalize)，得到的結果如下：



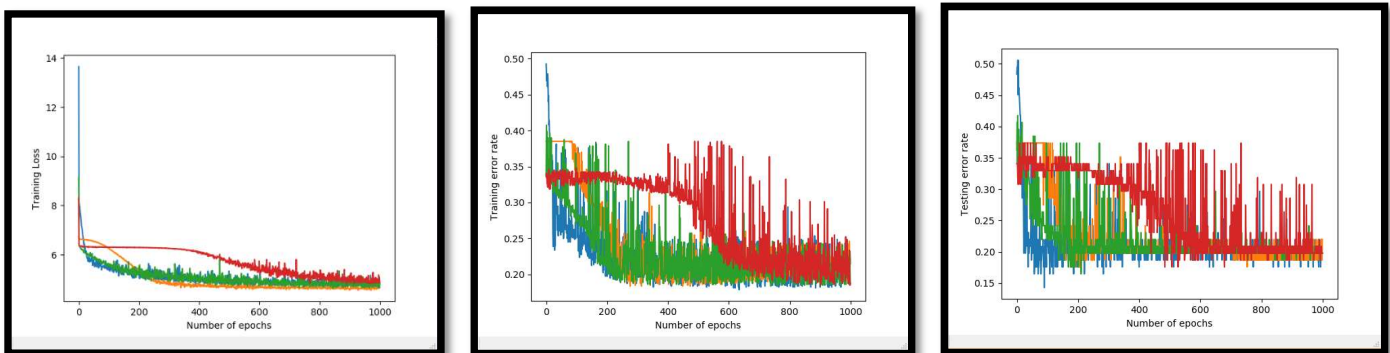
PS. 其中我的 Training Loss 為整個 mini-batch 的全部 Cross entropy，若要表現如範例的圖，可以將全部的值除以 10，也就是我每個 mini-batch 的資料數量，便是 Average Training Loss。

如同先前所做的，圖形的準確率會有小幅度震盪

接著為依序去 normalize fare, age 與同時 normalize 的結果，共 iteration 1000 epoch

沒做 Normalize	Normalize fare	Normalize age	Normalize both
藍色	橘色	綠色	紅色

實驗結果如下：



可以看出來，當我把 Fare 這個 feature 做 normalize 後，loss 與 error 都更快速的收斂了，收斂到達的 error 不僅降低了，震盪幅度也小於原本的结果，如果我改 normalize age 這個 feature，效果卻沒有 normalize fare 那麼好。同時 normalize 兩者甚至造成反效果。

Prob4

為了比較不同 feature 所具有的重要性，我會依序丟棄不同 feature 去訓練我的網路
如果訓練結果 error rate 與原本的差不多，表示這個 feature 並不重要
反之若訓練結果 error rate 有所上升，表示這個 feature 對判斷結果影響很關鍵

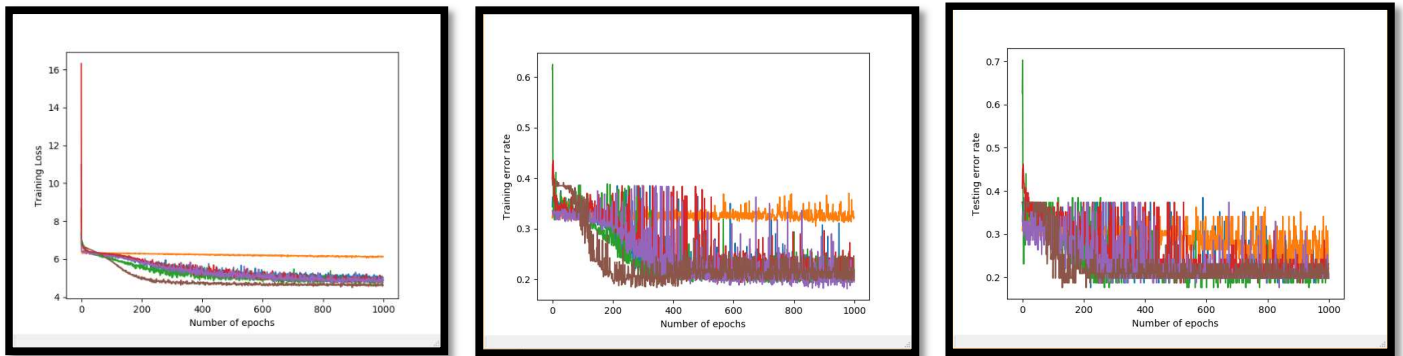
這次測試的參數如下：

節點: [5,3,3,2+maxout], Learning rate=0.001

Training data 800, Testing data 91, Batch size 10, Iteration 1000 epoch

Drop	Pclass	Sex	Age	SibSp	Parch	Fare
Color	藍色	橘色	綠色	紅色	紫色	咖啡色

以下分別為我丟棄不同 feature 後的訓練結果



PS. 其中我的 Training Loss 為整個 mini-batch 的全部 Cross entropy，若要表現如範例的圖，可以將全部的值除以 10，也就是我每個 mini-batch 的資料數量，便是 Average Training Loss。

由結果可以看出來，**"Sex"**這個 feature 對結果影響甚鉅，如果少了這個關鍵的特徵，判斷出來的結果錯誤率會大大增加。相反的，少了**"Fare"**這個參數，結果幾乎沒差，表示它的影響並不太重要。

Prob5

因為 PClass 是以等級來分成 1~3 的，但這些等級之間的關係又不如同線性的關聯

所以我們常常會將分成 3 類以上的 feature 轉成 one-hot vectors

若原本 PClass=1，可以改寫成[0 0 1]；PClass=2，可以改寫成[0 1 0]；PClass=3，可以改寫成[1 0 0]

也就是把原本 1 個 feature PClass 變成 3 個 feature 來輸入給神經網路

這次測試的參數如下：

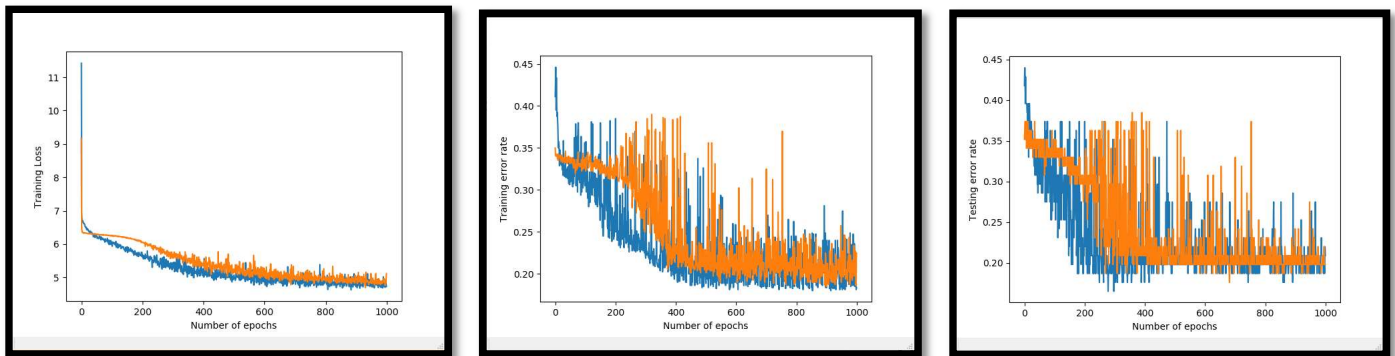
節點: [8,3,3,2+maxout], Learning rate=0.001

Training data 800, Testing data 91, Batch size 10, Iteration 1000 epoch

修改網路的輸入層與第 2 層 weight 和 bias

[6,3,3,2+maxout]	[8(PClass ->one hot),3,3,2+maxout]
橘色	藍色

實驗結果如下：



PS. 其中我的 Training Loss 為整個 mini-batch 的全部 Cross entropy，若要表現如範例的圖，可以將全部的值除以 10，也就是我每個 mini-batch 的資料數量，便是 Average Training Loss。

由結果可以發現，若把 feature 轉為 one-hot vectors 可以加快收斂速度，同時降低 error rate。

Prob6

為了要找到不同 **feature** 的數值對於最後判斷該人是生存或死亡的結果
我先將每個 **feature** 做統計

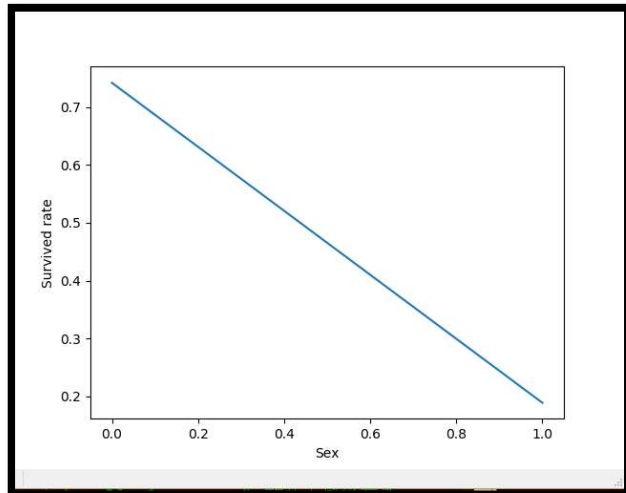
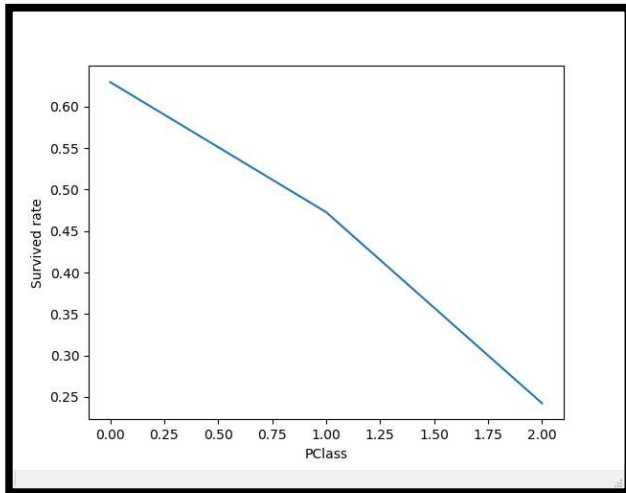
這次測試的參數如下:

節點: [6,3,3,2+maxout], Learning rate=0.001

Training data 800, Testing data 91, Batch size 10, Iteration 1000 epoch

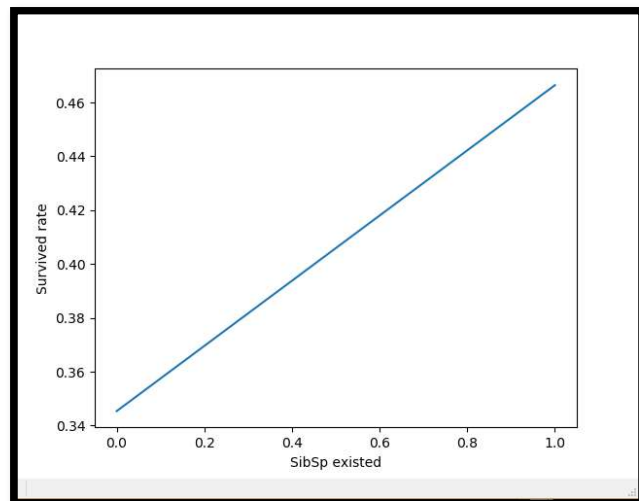
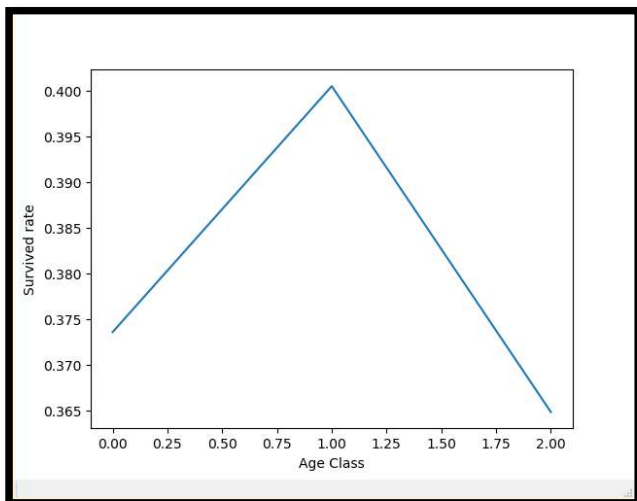
左圖為受到乘坐的 **PClass** 等級影響，可以看到在 **PClass1** 頭等艙的存活率有 60% 以上，**PClass2** 存活率約 48%，若在 **PClass3** 存活率只剩下 25%，可見頭等艙有較好的逃難設施。

右圖為受到性別 **Sex** 的影響，女性存活率高達 70%以上，然而男性只有 20%，可能受到“婦女與孩童優先上救生艇”影響，女性存活率遠高於男性，也是這點讓這個 **feature** 最為重要。



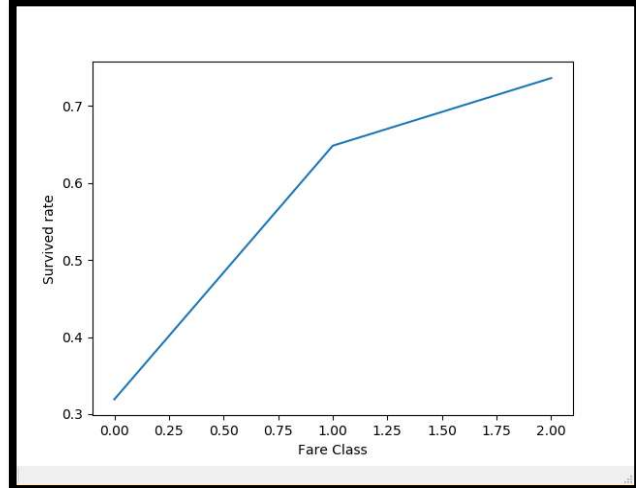
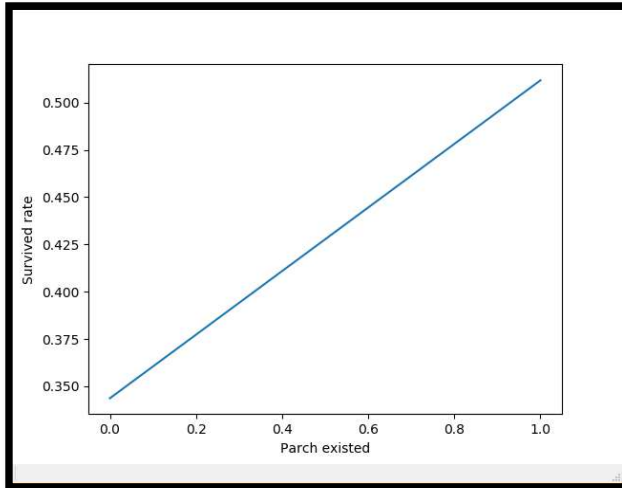
左圖為受到年齡 **Age** 的影響，我大致將年齡分成 3 類: 0~25 歲為幼年、25~50 為壯年、50 以上的為老年，而結果明顯壯年人口存活率較高，老年人口存活率最低，滿符合現實狀況的。

右圖為受到同行旁系血親 **SibSp** 影響，我分成沒有同行旁系血親與有同行旁系血親。結果為有同行旁系血親的存活率略為增加。



左圖為受到同行直系血親 SibSp 影響，我分成沒有同行直系血親與有同行直系血親。結果為有同行直系血親的存活率略為增加。

右圖為受到 Fare 多寡的影響，我大致將 Fare 分成 3 類: 0~50、50~100、100 以上，而結果 100 以上存活率較高



實驗結果:

我先設計一個我認為有很大機率會存活的個體，其特徵如下:

她是坐在頭等艙的女性，年齡約 30 歲，同行的有 2 個兄弟和她的父母親，她是有錢人家 Fare200

即其 feature 為 Pclass=1, Sex=0, Age=30, SibSp=2, Parch=2, Fare=200

將這筆資料丟入已經訓練過正確率為 80%左右的神經網路當中

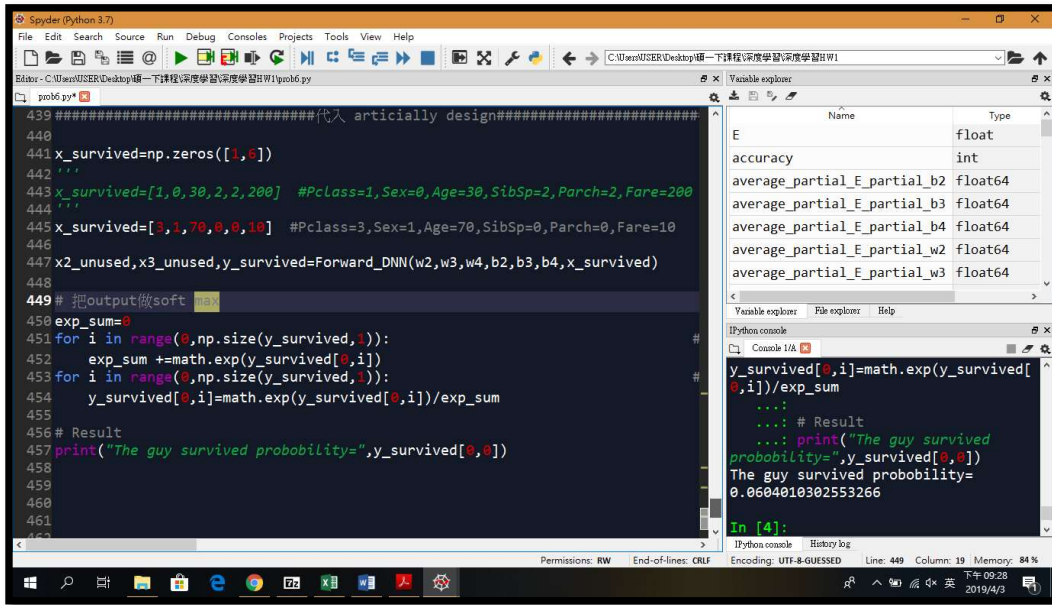
```
436 np.random.shuffle(data)
437
438
439 #####代人工藝設計#####
440
441 x_survived=np.zeros([1,6])
442 x_survived=[1,0,30,2,2,200] #Pclass=1,Sex=0,Age=30,SibSp=2,Parch=2,Fare=200
443
444 x2_unused,x3_unused,y_survived=Forward_DNN(w2,w3,w4,b2,b3,b4,x_survived)
445
446 # 把output做soft max
447 exp_sum=0
448 for i in range(0,np.size(y_survived,1)):
449     exp_sum +=math.exp(y_survived[0,i])
450 for i in range(0,np.size(y_survived,1)):
451     y_survived[0,i]=math.exp(y_survived[0,i])/exp_sum
452
453 # Result
454 print("The guy survived probability=",y_survived[0,0])
455
456
457
458
```

Name	Type
E	float
accuracy	int
average_partial_F_partial_b2	float64
average_partial_F_partial_b3	float64
average_partial_F_partial_b4	float64
average_partial_F_partial_w2	float64
average_partial_F_partial_w3	float64

```
Error rate= 0.18681318681318682
Error rate= 0.19780219780219777
Error rate= 0.18681318681318682
Error rate= 0.19780219780219777
Error rate= 0.19780219780219777
Error rate= 0.21978021978021978
Error rate= 0.18681318681318682
The guy survived probability=
0.9887045651379047
```

算出來存活率為 0.9887045651379047，基本死不了

接著，我再設計一個我認為有很大機率會死亡的個體，其特徵如下：
他是坐在最低艙等的男性，年齡約 30 歲，沒有同行的家人，他沒甚麼錢 Fare=10
即其 feature 為 Pclass=3,Sex=1,Age=70,SibSp=0,Parch=0,Fare=10
將這筆資料丟入已經訓練過正確率為 80%左右的神經網路當中



The screenshot shows the Spyder Python IDE interface. The main editor window displays a Python script for a neural network prediction. The script includes a comment indicating it is an artificially designed input. The input features are defined as `x_survived = [1, 30, 2, 2, 200]` for a previous example and `x_survived = [3, 1, 70, 0, 10]` for the current example. The script uses a forward pass function `Forward_DNN` to calculate the output. The output is then used to calculate the survival probability using the formula `y_survived[0,i] = math.exp(y_survived[0,i]) / exp_sum`. The final result is printed as "The guy survived probability=" followed by the calculated value.

```
439 ##### 代入 artificially design #####
440
441 x_survived=np.zeros([1,6])
442 '''
443 x_survived=[1,30,2,2,200] #Pclass=1,Sex=0,Age=30,SibSp=2,Parch=2,Fare=200
444 '''
445 x_survived=[3,1,70,0,10] #Pclass=3,Sex=1,Age=70,SibSp=0,Parch=0,Fare=10
446
447 x2_unused,x3_unused,y_survived=Forward_DNN(w2,w3,w4,b2,b3,b4,x_survived)
448
449 # 把output做softmax
450 exp_sum=0
451 for i in range(0,np.size(y_survived,1)):
452     exp_sum +=math.exp(y_survived[0,i])
453 for i in range(0,np.size(y_survived,1)):
454     y_survived[0,i]=math.exp(y_survived[0,i])/exp_sum
455
456 # Result
457 print("The guy survived probability=",y_survived[0,0])
458
459
460
461
462
```

The Variable explorer on the right shows the following variables:

Name	Type
E	float
accuracy	int
average_partial_E_partial_b2	float64
average_partial_E_partial_b3	float64
average_partial_E_partial_b4	float64
average_partial_E_partial_w2	float64
average_partial_E_partial_w3	float64

The IPython console shows the following output:

```
y_survived[0,i]=math.exp(y_survived[0,i])/exp_sum
...:
...: # Result
...: print("The guy survived probability=",y_survived[0,0])
The guy survived probability=
0.0604010302553266
In [4]:
```

算出來存活率為 0.0604010302553266，基本活不了，除非剛好抓到海上漂浮的木塊