1



In [11]:
```
import numpy as np

X = np.array([0,1])
y = 1

alpha1 = np.array(([0.1,0.3]), dtype=float)
alpha2 = np.array(([0.3,0.4]), dtype=float)
beta = np.array(([0.4,0.6]), dtype=float)
learning_rate = 0.01
def sigmoid(x):
    return 1/(1+np.exp(-x))
def sigmoidPrime(z):
    return np.exp(-z)/((1+np.exp(-z))**2)
def costFunction(y, yHat):
    J = (y-yHat)**2
    return J
def costFunction_prime(y, yHat):
    #derivative corresponds to y_hat
    return -2*(y-yHat)

def compute_y_hat(alpha1, alpha2, beta, X, y):
    input_for_z1 = np.dot(alpha1, X)
    z1= sigmoid(input_for_z1)
    input_for_z2 = np.dot(alpha2, X)
    z2= sigmoid(input_for_z2)
    z = np.array([z1,z2])
    output_for_z = np.dot(z,beta)
    y_hat = sigmoid(output_for_z)
    return y_hat

def compute_y_hat_prime_beta(alpha1, alpha2, beta, X, y):
    y_hat = compute_y_hat(alpha1, alpha2, beta, X, y)
    y_hat_prime = costFunction_prime(y, yHat)

    input_for_z1 = np.dot(alpha1, X)
    z1= sigmoid(input_for_z1)
    input_for_z2 = np.dot(alpha2, X)
    z2= sigmoid(input_for_z2)
    z = np.array([z1,z2])
    betaT_z_prime = sigmoidPrime(z)

    beta_prime1 = y_hat_prime*betaT_z_prime*z[0]
    beta_prime2 = y_hat_prime*betaT_z_prime*z[1]
    return beta_prime1, beta_prime2

def compute_y_hat_prime_beta(alpha1, alpha2, beta, X, y):
    y_hat = compute_y_hat(alpha1, alpha2, beta, X, y)
    y_hat_prime = costFunction_prime(y, y_hat)
    input_for_z1 = np.dot(alpha1, X)
    z1= sigmoid(input_for_z1)
    input_for_z2 = np.dot(alpha2, X)
    z2= sigmoid(input_for_z2)
    z = np.array([z1,z2])

    betaT_z_prime = sigmoidPrime(np.dot(z,beta))
    beta_prime1 = y_hat_prime*betaT_z_prime*z[0]
```

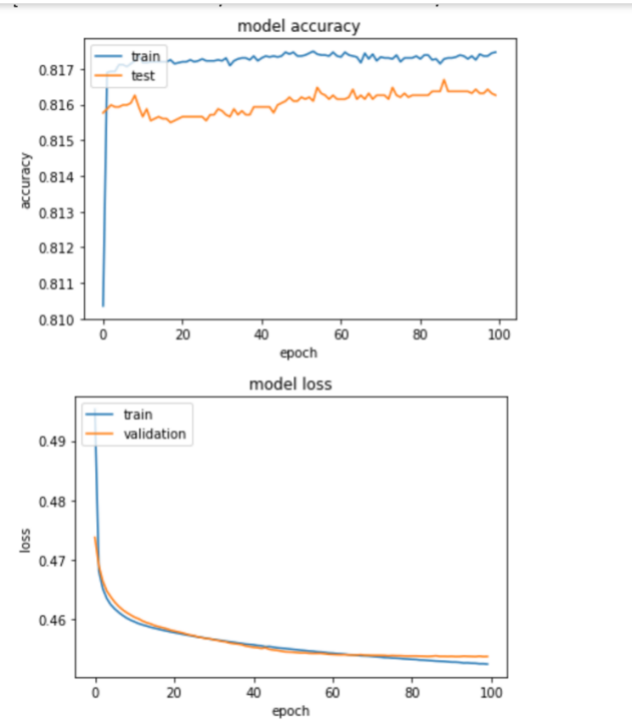
```
    beta_prime2 = y_hat_prime*betaT_z_prime*z[1]
    return beta_prime1, beta_prime2

def compute_y_hat_prime_alpha(alpha1, alpha2, beta, X, y):
    y_hat = compute_y_hat(alpha1, alpha2, beta, X, y)
    y_hat_prime = costFunction_prime(y, y_hat)

    input_for_z1 = np.dot(alpha1, X)
    z1= sigmoid(input_for_z1)
    input_for_z2 = np.dot(alpha2, X)
    z2= sigmoid(input_for_z2)
    z = np.array([z1,z2])

    betaT_z_prime = sigmoidPrime(np.dot(z,beta))

    alpha_prime11 = y_hat_prime*betaT_z_prime*beta[0]*sigmoidPrime(input_fo
    alpha_prime12 = y_hat_prime*betaT_z_prime*beta[0]*sigmoidPrime(input_fo
    alpha_prime21 = y_hat_prime*betaT_z_prime*beta[1]*sigmoidPrime(input_fo
    alpha_prime22 = y_hat_prime*betaT_z_prime*beta[1]*sigmoidPrime(input_fo

    return alpha_prime11,alpha_prime12,alpha_prime21,alpha_prime22
def update_beta(alpha1, alpha2, beta, X, y, learning_rate):
    beta_prime1, beta_prime2 = compute_y_hat_prime_beta(alpha1, alpha2, bet
    beta_prime = np.array([beta_prime1, beta_prime2])
    beta_update = beta-learning_rate*beta_prime
    return beta_update
def update_alpha1(alpha1, alpha2, beta, X, y, learning_rate):
    alpha_prime11,alpha_prime12,alpha_prime21,alpha_prime22 = compute_y_hat
    alpha1_prime = np.array([alpha_prime11,alpha_prime12])
    alpha_update = alpha1-learning_rate*alpha1_prime
    return alpha_update

def update_alpha2(alpha1, alpha2, beta, X, y, learning_rate):
    alpha_prime11,alpha_prime12,alpha_prime21,alpha_prime22 = compute_y_hat
    alpha2_prime = np.array([alpha_prime21,alpha_prime22])
    alpha_update = alpha2-learning_rate*alpha2_prime
    return alpha_update

def main_function(alpha1, alpha2, beta, X, y, learning_rate):
    while y-compute_y_hat(alpha1, alpha2, beta, X, y)>0.01:
        alpha1 = update_alpha1(alpha1, alpha2, beta, X, y, learning_rate)
        alpha2 = update_alpha2(alpha1, alpha2, beta, X, y, learning_rate)
        beta = update_beta(alpha1, alpha2, beta, X, y, learning_rate)
    return compute_y_hat(alpha1, alpha2, beta, X, y)
```

In [12]:
```
main_function(alpha1, alpha2, beta, X, y, learning_rate)
```

Out[12]: 0.9900000229217649

In [ ]:
```

```

2.
Model1

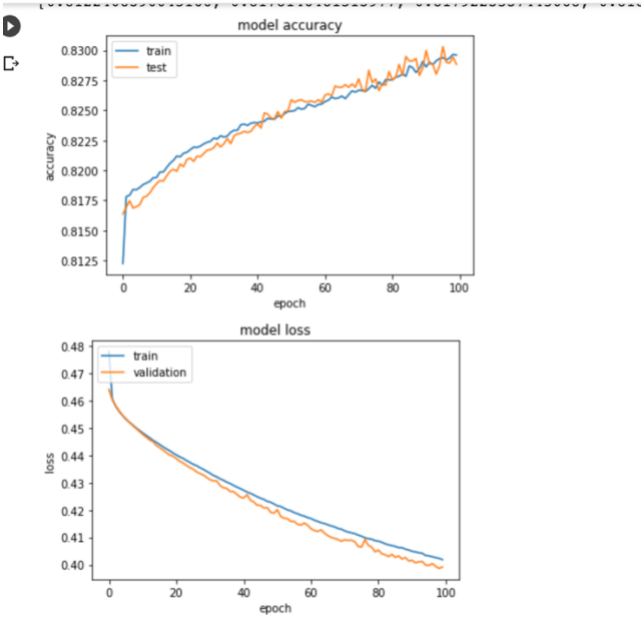



Yes, logistic regression is like this model 1.

```
[25] from sklearn.metrics import confusion_matrix
     predictions = neural_network.predict(X_test)
     y_pred = (predictions > 0.5)
     confusion_matrix(y_test, y_pred)

     193/193 [==============================] - 0s 1ms/step
     array([[5040,    0],
            [1111,    8]])
```

```
[26] neural_network.summary()

     Model: "sequential_1"
     _________________________________________________________________
      Layer (type)                 Output Shape              Param #
     =================================================================
      dense_1 (Dense)              (None, 1)                 22

     =================================================================
     Total params: 22
     Trainable params: 22
     Non-trainable params: 0
     _________________________________________________________________
```

Model 2




1 output layer as before.

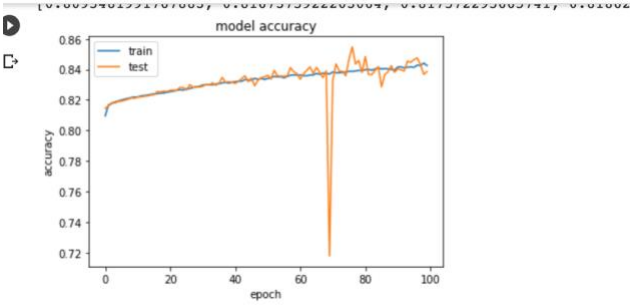
```
[28] from sklearn.metrics import confusion_matrix
     predictions = model.predict(X_test)
     y_pred = (predictions > 0.5)
     confusion_matrix(y_test, y_pred)
     model.summary()
```

```
193/193 [==============================] - 0s 1ms/step
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 15)                330

 dense_3 (Dense)             (None, 1)                 16

=================================================================
Total params: 346
Trainable params: 346
Non-trainable params: 0
_________________________________________________________________
```

```
[29] confusion_matrix(y_test, y_pred)
```

```
array([[5009,   31],
       [1023,   96]])
```

## Model 3




```
mod.summary()
```

```
193/193 [==============================] - 0s 1ms/step
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 25)                550

 dense_5 (Dense)             (None, 15)                390

 dense_6 (Dense)             (None, 1)                 16

=================================================================
Total params: 956
Trainable params: 956
Non-trainable params: 0
_________________________________________________________________
```

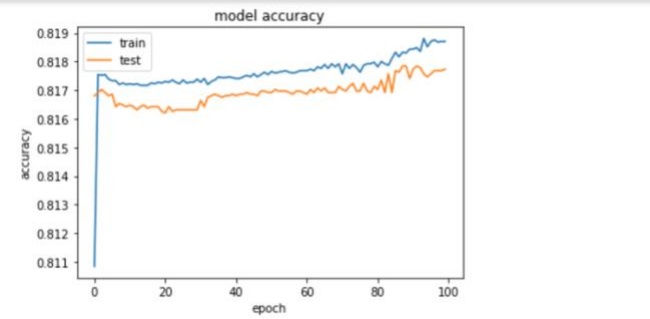
+ Code + Text

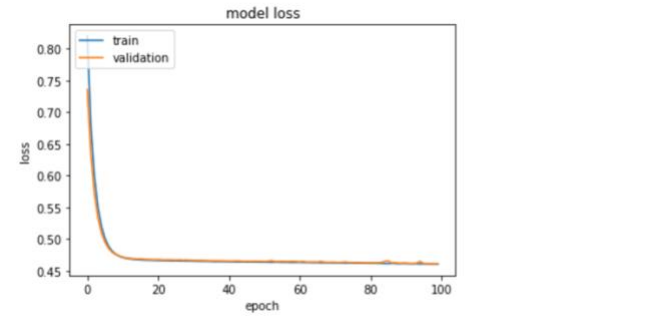
Double-click (or enter) to edit

```
[32] confusion_matrix(y_test, y_pred)
```

```
array([[4988,   52],
       [ 965,  154]])
```

## Model 4




```
[34] from sklearn.metrics import confusion_matrix
     predictions = mod4.predict(X_test)
     y_pred = (predictions > 0.5)
     mod4.summary()
```

```
193/193 [==============================] - 0s 1ms/step
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 25)                550

 dense_8 (Dense)             (None, 15)                390

 dense_9 (Dense)             (None, 1)                 16

=================================================================
Total params: 956
Trainable params: 956
Non-trainable params: 0
_________________________________________________________________
```

```
confusion_matrix(y_test, y_pred)
```

```
array([[5027,   13],
       [1094,   25]])
```

Yes, model 3 is overfitting so I need to add regulations for model 4.

3.

```
print("y_pred",y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
y_pred [2 2 1 2 1 1 2 1 2 2 2 1 2 2 2 1]
Accuracy: 0.9375
              precision    recall  f1-score   support

           1       1.00      0.86      0.92         7
           2       0.90      1.00      0.95         9

    accuracy                           0.94        16
   macro avg       0.95      0.93      0.94        16
weighted avg       0.94      0.94      0.94        16

Accuracy: 0.9375
```

In [57]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.001,0.1,1, 10],'gamma' : [0.1, 0.01, 1, 10],'kernel': ['rbf', 'poly', 'sigmoid','linear']}
grid2 = GridSearchCV(SVC(), param_grid)

grid2.fit(X_train, y_train)
# print best parameter after tuning
print(grid2.best_params_)
grid_predictions2 = grid2.predict(X_test)
print(classification_report(y_test, grid_predictions2))
```

```
{'C': 10, 'gamma': 0.01, 'kernel': 'sigmoid'}
              precision    recall  f1-score   support

           1       1.00      0.86      0.92         7
           2       0.90      1.00      0.95         9

    accuracy                           0.94        16
   macro avg       0.95      0.93      0.94        16
weighted avg       0.94      0.94      0.94        16
```

In [58]:
```
print("Accuracy:",metrics.accuracy_score(y_test, grid_predictions2))
```

```
Accuracy: 0.9375
```