Import dataset¶

In [25]:

```
import pandas as pd
path="C:\\Users\\TANUJA HARISH\\Desktop\\ML and DL Summer
Internship\\diabetes.csv"
dataset=pd.read_csv(path)
dataset.head()
```

Out[25]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [26]:

```
print(dataset.shape)
```

```
(768, 9)
```

Divide dataset into inputs and output¶

In [27]:

```python
import numpy as np
x=np.array(dataset.iloc[:,0:8])
y=np.array(dataset[["Outcome"]])
print(x.shape)
print(y.shape)
```

```
(768, 8)
(768, 1)
```

Standardize the dataset¶

In [28]:

```python
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_norm=sc.fit_transform(x)
print(x_norm.shape)
```

```
(768, 8)
```

Split dataset into test and train¶

In [29]:

```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_norm,y,test_size=0.2)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(614, 8)
(154, 8)
(614, 1)
(154, 1)
```

implement binary classification for single layer from scratch¶

In [30]:

```
def initialize_parameters(x_train):
    w1=np.random.randn(5,x_train.shape[1])
    w2=np.random.randn(5,5)
    w3=np.random.randn(1,5)
    b1=np.random.randn(5,1)
    b2=np.random.randn(5,1)
    b3=np.random.randn(1,1)
    return w1,w2,w3,b1,b2,b3
def sigmoid_act(x):
    s=1/(1+np.exp(-x))
    return s
def forward_path(x,w1,b1,w2,b2,w3,b3):
    #Hidden layer 1
    z1=np.reshape(np.matmul(w1,x.T),b1.shape)+b1
    A1=sigmoid_act(z1)
    # Hidden layer 2
    z2=np.matmul(w2,A1)+b2
    A2=sigmoid_act(z2)
    # Output layer
    z3=np.matmul(w3,A2)+b3
    A3=sigmoid_act(z3)
    return A3,A2,A1
def compute_cost(y_pred,y_true):

cost=(y_true*np.log(y_pred+(10**-7))+(1-y_true)*np.log(1-y_pred+(10**-7)))
    return -cost
def backward_path(A3,A2,A1,w3,y,x):
    #output layer
```

```
    dA3=(A3-y)/(A3*(1-A3))
    dz3=(A3-y)
    dw3=np.matmul(dz3,A2.T)
    db3=dz3.mean(axis=1)

    # Hidden layer 2
    dA2=np.matmul(w3.T,dz3)
    dz2=np.multiply(dA2,A2*(1-A2))
    dw2=dz2*A1.T
    db2=dz2.mean(axis=1)

    # Hidden layer 1
    dA1=np.matmul(w2.T,dz2)
    dz1=np.multiply(dA1,A1*(1-A1))
    dw1=dz1*x
    db1=dz1.mean(axis=1)
    return dw3,dw2,dw1,db3,db2,db1
def update_parameters(w,b,dw,db,learning_rate):
    w=w-learning_rate*dw
    db=np.reshape(db,b.shape)
    b=b-learning_rate*db
    return w,b
```

Training using Stocastic Gradient Descent (SGD)¶

In [31]:

```
num_iter=500
learning_rate=0.001
his1=[]
w1,w2,w3,b1,b2,b3=initialize_parameters(x_train)
for i in range(num_iter):
```

```
    cost=0
    for j in range(len(x_train)):
        A3,A2,A1=forward_path(x_train[j],w1,b1,w2,b2,w3,b3)
        y_pred=A3
        cost+=compute_cost(y_pred,y_train[j])

dw3,dw2,dw1,db3,db2,db1=backward_path(A3,A2,A1,w3,y_train[j],x_train[j])
        w3,b3=update_parameters(w3,b3,dw3,db3,learning_rate)
        w2,b2=update_parameters(w2,b2,dw2,db2,learning_rate)
        w1,b1=update_parameters(w1,b1,dw1,db1,learning_rate)
    his1.append(cost/len(x_train))
```

In [32]:

```
import matplotlib.pyplot as plt
his=np.reshape(np.array(his1),(len(his1)))
plt.plot(his)
plt.show()
```

In [33]:

```
print(w1)
print(b1)
```

```
[[ 0.71287075  1.11606433 -0.48579931  0.74696682 -1.33434895  0.61525406
   0.41866515  1.53598114]
 [ 0.75046472 -2.03609108  0.23604482  1.68139441 -1.52143249 -1.2374753
  -0.78279605 -0.15746179]
```

```
 [ 0.04601611  1.8335858   0.14407443  0.56371905  0.53599772  0.83389947
   1.15395765  0.26482344]
 [-0.90133508 -0.48008995 -1.30604823  0.38592033  0.74367347  2.11835339
  -0.82211243  0.78698559]
 [ 0.01844536 -1.09405939  0.46125204 -0.19236912  0.15413852 -0.02301238
   1.23959077  1.26615095]]
[[ 1.25055613]
 [ 0.69794104]
 [-0.85304562]
 [-0.78551639]
 [-0.75965967]]
```

In [34]:

```
print(w2)
print(b2)
```

```
[[ 1.97842185e+00 -1.80118556e-01 -6.13713724e-01 -6.98749728e-01
  -2.02541622e+00]
 [-1.31810972e+00  9.70194831e-01 -1.00122787e+00 -6.52831583e-01
   3.10428482e-01]
 [-1.87062475e+00  1.06839934e+00 -2.16870925e-01  2.86192350e-01
  -5.69659226e-01]
 [ 2.25711823e+00 -1.82019546e-01  6.76280715e-01  3.73833996e-01
   1.15477595e+00]
 [-4.28758539e-01  8.98886159e-01 -1.65633026e+00 -2.04391551e-01
   1.72390151e-03]]
[[-0.28719297]
 [ 0.14525655]
 [-0.34758604]
 [-0.41504736]
 [ 0.84909829]]
```

In [35]:

```
print(w3)
print(b3)
```

```
[[ 1.77330699 -3.29947851 -2.36798579  1.1943152  -1.62244391]]
[[0.22470193]]
```

Prediction on test dataset¶

In [36]:

```
y_pred=np.zeros(y_test.shape)
for i in range(len(x_test)):
    A3,A2,A1=forward_path(x_test[i],w1,b1,w2,b2,w3,b3)
    A3=np.where(A3>0.5,1,0)  #thresholding-a2 is 1 id greater then 0.5 else 0
    y_pred[i]=A3
print(y_pred)
print(y_test)
```

```
[[1.]
 [1.]
 [0.]
 [1.]
 [1.]
 [1.]
 [0.]
 [1.]
 [1.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [0.]
 [0.]
 [1.]
 [1.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [0.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]
```

```
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
[0.]
```

```
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[1.]
[0.]
[0.]
[0.]
[0.]
[1.]
[1.]
[0.]
[0.]
[0.]
[1.]
[0.]
[0.]
[1.]
[0.]
[1.]
[1.]
[0.]
[0.]
[1.]
[0.]
[0.]
[0.]
[1.]
[0.]
[1.]
[0.]
[0.]
[0.]
[0.]
[0.]
```

```
 [0.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [1.]
 [0.]
 [1.]
 [1.]
 [0.]
 [1.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]]
[[1]
 [1]
 [0]
 [1]
 [0]
 [1]
 [0]
 [1]
 [1]
 [1]
 [1]
 [0]
 [0]
 [1]
 [0]
 [1]
 [0]
```

[0]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[1]
[1]
[1]
[0]
[1]
[0]

[0]
[1]
[0]
[1]
[1]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[0]
[1]
[0]
[0]
[1]
[1]
[0]
[0]
[0]
[0]
[1]

[0]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[0]
[1]
[0]
[0]
[1]
[1]
[1]
[0]
[0]
[0]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[1]
[0]
[0]
[0]
[0]
[1]
[1]
[0]
[1]
[0]
[0]
[1]
[0]
[1]

```
 [1]
 [0]
 [1]
 [0]
 [0]
 [0]
 [0]
 [1]]
```

In [37]:

```
print("No of wrong prediction:",sum(y_pred!=y_test))
```

```
No of wrong prediction: [32]
```

Evaluating Quality metrics of model on the test set¶

In [38]:

```
TP=sum(np.logical_and(y_test==1,y_pred==1))
TN=sum(np.logical_and(y_test==0,y_pred==0))
FP=sum(np.logical_and(y_test==0,y_pred==1))
FN=sum(np.logical_and(y_test==1,y_pred==0))
Accuracy=sum(y_pred==y_test)/len(y_test)
```

```
print("Accuracy",Accuracy)
precision=TP/(TP+FP)
print("Precision",precision)
recall=TP/(TP+FN)
print("Recall",recall)
f1=(2*precision*recall)/(precision+recall)
print("F1-score",f1)
```

```
Accuracy [0.79220779]
Precision [0.76086957]
Recall [0.625]
F1-score [0.68627451]
```

In [39]:

```
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score
cm=np.array(confusion_matrix(y_test,y_pred,labels=[0,1]))
print(cm)
confusion=pd.DataFrame(cm,index=["Diabetics","No
Diabetics"],columns=["Diabetics","No Diabetics"])
print(confusion)
print(classification_report(y_test,y_pred,labels=[0,1]))
acc=accuracy_score(y_test,y_pred)
print(acc)
```

```
[[87 11]
 [21 35]]
              Diabetics  No Diabetics
```

```
Diabetics            87          11
No Diabetics         21          35
              precision    recall  f1-score   support

           0       0.81      0.89      0.84        98
           1       0.76      0.62      0.69        56

    accuracy                           0.79       154
   macro avg       0.78      0.76      0.77       154
weighted avg       0.79      0.79      0.79       154

0.7922077922077922
```

In [ ]: