Mini-Project

Name: Vedant Sanjay Dhamale

Roll No:2337032

Batch: B

In [1]:
```
import numpy as np

class ANN:
    def __init__(self,input_size,hidden_layers,hidden_neurons,output_size,learning_
        self.weights=[]
        self.bias=[]
        self.hidden_layers=hidden_layers
        self.learning_rate=learning_rate;

        for i in range(hidden_layers+1):
            if i==0:
                self.weights.append(np.random.randn(hidden_neurons,input_size))
                self.bias.append(np.full((hidden_neurons,1),1))
            elif i==hidden_layers:
                self.weights.append(np.random.randn(output_size,hidden_neurons))
                self.bias.append(np.full((output_size,1),1))
            else:
                self.weights.append(np.random.randn(hidden_neurons,hidden_neurons))
                self.bias.append(np.full((hidden_neurons,1),1))

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def first_order_sigmoid(self, x):
        return self.sigmoid(x) * (1 - self.sigmoid(x))

    def forward(self,x):
        activations=[]
        activations.append(x)
        for i in range(self.hidden_layers+1):
            x=np.dot(self.weights[i],activations[i])+self.bias[i]
            activations.append(self.sigmoid(x))
        return activations

    def backward(self,activations,di,m):
        delta=(activations[-1]-di.T) * self.first_order_sigmoid(np.dot(self.weights
        for i in range(self.hidden_layers,-1,-1):
            if i==self.hidden_layers:
                prev=np.array(self.weights[i])
                self.weights[i]=self.weights[i]-(self.learning_rate/m) * np.dot(del
                self.bias[i]=self.bias[i]-(self.learning_rate/m) * np.sum(delta,axi
            else:
                delta=np.dot(prev.T, delta) * self.first_order_sigmoid(np.dot(self.
                prev=np.array(self.weights[i])
```

```
                self.weights[i]=self.weights[i]-(self.learning_rate/m) * np.dot(del
                self.bias[i]=self.bias[i]-(self.learning_rate/m) * np.sum(delta,axi

    def train(self,x,y,epochs):
        for i in range(epochs):
            activations=self.forward(x)
            m=x.shape[1]
            self.backward(activations,y,m)
            if(i%1000==0):
                print("Error at %d epoch : "%(i),np.sum(activations[-1]-y.T))

    def predict(self,x):
        predictions=[]
        for input in x:
            prediction = self.forward(np.array(input))
            predictions.append(prediction[-1])
        return predictions
```

In [2]:
```
input_size=20
hidden_layers=4
neurons_in_hidden_layer=40
output_size=2
learning_rate=0.1

model=ANN(input_size,hidden_layers,neurons_in_hidden_layer,output_size,learning_rat
```

In [3]:
```
import pandas as pd

df=pd.read_csv('churn-bigml-80.csv')
```

In [4]:
```
from sklearn.preprocessing import LabelEncoder
lc = LabelEncoder()

df['State'] = lc.fit_transform(df['State'])
df['International plan'] = lc.fit_transform(df['International plan'])
df['ground_floor_type'] = lc.fit_transform(df['Voice mail plan'])
df['Churn'] = lc.fit_transform(df['Churn'])
df['Voice mail plan'] = lc.fit_transform(df['Voice mail plan'])
```

In [5]:
```
x = df.drop('Churn', axis=1)
y=df['Churn']
```

In [6]:
```
df.shape
```

Out[6]: (2666, 21)

In [7]:
```
df.dtypes
```

Out[7]:
```
State                       int32
Account length              int64
Area code                   int64
International plan          int32
Voice mail plan             int32
Number vmail messages       int64
Total day minutes         float64
Total day calls             int64
Total day charge          float64
Total eve minutes         float64
Total eve calls             int64
Total eve charge          float64
Total night minutes       float64
Total night calls           int64
Total night charge        float64
Total intl minutes        float64
Total intl calls            int64
Total intl charge         float64
Customer service calls      int64
Churn                       int64
ground_floor_type           int32
dtype: object
```

In [8]:
```
import numpy as np
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(y)

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,encoded_labels,test_size=0.2,rando

from tensorflow.keras.utils import to_categorical

labels = np.array(y_train)
y_train = to_categorical(labels)
y_train=np.array(y_train)
```

```
WARNING:tensorflow:From d:\New folder\envs\ds_env\Lib\site-packages\keras\src\losse
s.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [9]:
```
x_train=x_train.values
x_train=np.array(x_train)
x_train=x_train.T
x_test=x_test.values
```

In [10]:
```
y_train
```

Out[10]:
```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [0., 1.],
       [1., 0.]], dtype=float32)
```

In [11]:
```
epochs=100000
model.train(x_train,y_train,epochs)
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_2600\1263177278.py:22: RuntimeWarning:
overflow encountered in exp
  return 1 / (1 + np.exp(-x))
```

```
Error at 0 epoch :  -199.06152210171513
Error at 1000 epoch :  -5.054703585434193
Error at 2000 epoch :  -2.604023548334231
Error at 3000 epoch :  4.507120261589649
Error at 4000 epoch :  8.514630930092832
Error at 5000 epoch :  10.665267414401306
Error at 6000 epoch :  11.396080374098599
Error at 7000 epoch :  11.360570171596784
Error at 8000 epoch :  17.2612785029581
Error at 9000 epoch :  14.391118381710926
Error at 10000 epoch :  10.855447117664283
Error at 11000 epoch :  6.926461222736296
Error at 12000 epoch :  5.412061858694117
Error at 13000 epoch :  4.186964672288908
Error at 14000 epoch :  3.526682600506529
Error at 15000 epoch :  3.59054775029478
Error at 16000 epoch :  2.5268321078661877
Error at 17000 epoch :  2.324457526647266
Error at 18000 epoch :  3.3490071643860935
Error at 19000 epoch :  3.7625084110438043
Error at 20000 epoch :  3.127132778844558
Error at 21000 epoch :  3.5002399966766906
Error at 22000 epoch :  3.4166925978397327
Error at 23000 epoch :  2.8814306332411563
Error at 24000 epoch :  2.991196406049886
Error at 25000 epoch :  2.7356271096130644
Error at 26000 epoch :  3.269576162519914
Error at 27000 epoch :  3.1274986085133234
Error at 28000 epoch :  2.7978923444596338
Error at 29000 epoch :  2.837134897870877
Error at 30000 epoch :  2.3936973835704
Error at 31000 epoch :  2.857143179389536
Error at 32000 epoch :  1.769269139716918
Error at 33000 epoch :  2.155866044879013
Error at 34000 epoch :  2.5302369446847335
Error at 35000 epoch :  2.3881894763021734
Error at 36000 epoch :  1.8595865005310497
Error at 37000 epoch :  2.1747204115565624
Error at 38000 epoch :  2.6364944567066875
Error at 39000 epoch :  1.6629981939770233
Error at 40000 epoch :  1.9428376587492822
Error at 41000 epoch :  2.2252439938914463
Error at 42000 epoch :  1.823572730736907
Error at 43000 epoch :  0.9148995484151747
Error at 44000 epoch :  1.6253661075455024
Error at 45000 epoch :  1.8321233207833663
Error at 46000 epoch :  0.6624626846477053
Error at 47000 epoch :  1.236759816070169
Error at 48000 epoch :  1.0015920484273
Error at 49000 epoch :  1.3647590625351511
Error at 50000 epoch :  1.3172494460854587
Error at 51000 epoch :  1.4610818583093104
Error at 52000 epoch :  1.7765338360125509
Error at 53000 epoch :  1.2819934103990924
Error at 54000 epoch :  0.9056502801932904
Error at 55000 epoch :  0.6481704189280015
```

```
Error at 56000 epoch :  0.060363363206549
Error at 57000 epoch :  -0.04502144103641825
Error at 58000 epoch :  1.0973646315164665
Error at 59000 epoch :  0.1773272014031697
Error at 60000 epoch :  1.3925729598640704
Error at 61000 epoch :  0.4791099392863174
Error at 62000 epoch :  1.389177865060776
Error at 63000 epoch :  -0.21567819960488155
Error at 64000 epoch :  -0.07119834324197427
Error at 65000 epoch :  1.5980001593203603
Error at 66000 epoch :  0.6371660004110389
Error at 67000 epoch :  1.4948866467945852
Error at 68000 epoch :  0.7888420850383469
Error at 69000 epoch :  1.0689806821908547
Error at 70000 epoch :  -0.19332828470752483
Error at 71000 epoch :  0.34544179596564817
Error at 72000 epoch :  0.47403505309696436
Error at 73000 epoch :  -0.05599940077024934
Error at 74000 epoch :  0.36262320899004585
Error at 75000 epoch :  0.29782075763704086
Error at 76000 epoch :  -0.3422179693316165
Error at 77000 epoch :  0.7888574692259125
Error at 78000 epoch :  0.33964840028290855
Error at 79000 epoch :  0.7839379056777838
Error at 80000 epoch :  -0.4283117047393752
Error at 81000 epoch :  0.6269560809299266
Error at 82000 epoch :  0.2497278581428386
Error at 83000 epoch :  0.1944454392639794
Error at 84000 epoch :  -0.050892615477705405
Error at 85000 epoch :  0.36625340904856607
Error at 86000 epoch :  0.24079490036024076
Error at 87000 epoch :  0.6031635564666682
Error at 88000 epoch :  0.1578777806786813
Error at 89000 epoch :  0.6863935929634404
Error at 90000 epoch :  0.25479648633158547
Error at 91000 epoch :  0.38546976763008667
Error at 92000 epoch :  0.45238218755271387
Error at 93000 epoch :  0.18976997565365217
Error at 94000 epoch :  0.2923035801640128
Error at 95000 epoch :  0.7741072203410617
Error at 96000 epoch :  0.6982226751634171
Error at 97000 epoch :  0.3068898254671808
Error at 98000 epoch :  0.37900926161034487
Error at 99000 epoch :  0.800507394400964
```

In [14]:
```
test_sample=[]
for i in x_test:
    test_sample.append([[x] for x in i.tolist()])
```

In [15]:
```
y_pred=model.predict(test_sample)
y_pred = np.hstack([np.argmax(arr, axis=0) for arr in y_pred]).flatten()
```

```
C:\Users\lenovo\AppData\Local\Temp\ipykernel_2600\1263177278.py:22: RuntimeWarning:
overflow encountered in exp
  return 1 / (1 + np.exp(-x))
```

In [16]:
```
print(y_test)
```

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

In [17]:
```
print(y_pred)
```

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

In [18]:
```
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score,co
accuracy=accuracy_score(y_test,y_pred)
precision=precision_score(y_test,y_pred,average='weighted')
recall=recall_score(y_test,y_pred,average='weighted')
f1score=f1_score(y_test,y_pred,average='weighted')
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1score)
```

```
Accuracy: 0.8932584269662921
Precision: 0.8836942246810346
Recall: 0.8932584269662921
F1-score: 0.8719630093681084
```

In [ ]: