Exercise-9

Python Program for Artificial Neural Network (Manual).

In [12]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from numpy.random import seed
seed(1)
import tensorflow
tensorflow.random.set_seed(1)
import warnings
warnings.filterwarnings("ignore")
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
import seaborn as sns
from sklearn import metrics

# Functions for the ANN Model
def initialize_parameters(n_x,n_h,n_y):
    W1=np.random.randn(n_h,n_x)*0.01
    b1=np.zeros((n_h,1))
    W2=np.random.randn(n_y,n_h)*0.01
    b2=np.zeros((n_y,1))
    parameters={
            "W1":W1,
            "b1":b1,
            "W2":W2,
            "b2":b2}
    return parameters

def layer_sizes(X,Y):
    n_x=X.shape[0]
    n_h=10
    n_y=Y.shape[0]
    return (n_x,n_h,n_y)

def compute_cost(A2,Y,parameters):
    m=Y.shape[1]
    logprobs=np.multiply(np.log(A2),Y)+np.multiply(np.log(1-A2),(1-Y))
    cost=-(1/m)*np.sum(logprobs)
    return cost

def forward_propagation(X,parameters):
    W1=parameters["W1"]
    b1=parameters["b1"]
    W2=parameters["W2"]
    b2=parameters["b2"]

    Z1=np.dot(W1,X)+b1
    A1=np.tanh(Z1)
    Z2=np.dot(W2,A1)+b2
    A2=1/(1+np.exp(-Z2))

    cache={
            "Z1":Z1,
            "A1":A1,
            "Z2":Z2,
            "A2":A2}
    return A2,cache

def backward_propagation(parameters,cache,X,Y):
    m=X.shape[1]
    W1=parameters["W1"]
    b1=parameters["b1"]
    W2=parameters["W2"]
    b2=parameters["b2"]

    A1=cache["A1"]
    A2=cache["A2"]

    dZ2=A2-Y
    dW2=(1/m)*np.dot(dZ2,A1.T)
    db2=(1/m)*np.sum(dZ2,axis=1,keepdims=True)
    dZ1 = np.multiply((np.dot(W2.T,dZ2)),(1-np.power(A1,2)))
    dW1 = 1/(m)*np.dot(dZ1,X.T)
    db1 = 1/(m)*np.sum(dZ1,axis=1,keepdims=True)

    grads = {"dW1": dW1,
             "db1": db1,
             "dW2": dW2,
             "db2": db2}
    return grads

def update_parameters(parameters, grads, learning_rate = 0.1):
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    dW1 = grads["dW1"]
    db1 = grads["db1"]
    dW2 = grads["dW2"]
    db2 = grads["db2"]

    W1 = W1-((learning_rate)*dW1)
    b1 = b1-((learning_rate)*db1)
    W2 = W2-((learning_rate)*dW2)
    b2 = b2-((learning_rate)*db2)

    parameters = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2}
    return parameters

def predict(parameters, X):
    A2, cache = forward_propagation(X,parameters)
    predictions = (A2>0.5)*1
    return predictions

# The ANN Model
def nn_model(X, Y, n_h, num_iterations = 128, print_cost=False):
    np.random.seed(3)
    n_x = layer_sizes(X, Y)[0]
    n_y = layer_sizes(X, Y)[2]

    parameters = initialize_parameters(n_x, n_h, n_y)
    W1 = parameters["W1"]
    b1 = parameters["b1"]
    W2 = parameters["W2"]
    b2 = parameters["b2"]

    for i in range(0, num_iterations):
          A2, cache = forward_propagation(X,parameters)
          cost = compute_cost(A2,Y,parameters)
          grads = backward_propagation(parameters,cache,X,Y)
          parameters = update_parameters(parameters,grads,2)
          if print_cost:
             print ("Epoch %i/%i : \t loss:%f" %(i+1,num_iterations, cost))
    return parameters

# Input: Dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values

# Encoding Categorical Values
labelencoder_X = LabelEncoder()
X[:, 2] = labelencoder_X.fit_transform(X[:, 2])
ct = ColumnTransformer([("Geography", OneHotEncoder(), [1])],
                        remainder = 'passthrough')
X = ct.fit_transform(X)

# Splitting the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Preprocessing Data
X_train=X_train.T
X_test=X_test.T
y_train=y_train.reshape(y_train.shape[0],1)
y_test=y_test.reshape(y_test.shape[0],1)
y_train=y_train.T
y_test=y_test.T
shape_X=X_train.shape
shape_Y=y_train.shape
m=X_train.shape[1]

# Train the model using the training sets
(n_x,n_h,n_y)=layer_sizes(X_train,y_train)
parameters=initialize_parameters(n_x,n_h,n_y)
A2,cache=forward_propagation(X_train,parameters)
grads = backward_propagation(parameters, cache, X_train, y_train)
parameters = update_parameters(parameters, grads)
parameters = nn_model(X_train, y_train, 10, num_iterations=128,print_cost=True)

# Make predictions using the testing set
y_pred = predict(parameters, X_test)

# Calculating the Confusion Matrix and Accuracy of the Model
cm = metrics.confusion_matrix(y_test.T, y_pred.T)
accuracy = 100*(cm[0][0]+cm[1][1])/X_test.shape[1]
```

```
Epoch 1/128 :    loss:0.693171
Epoch 2/128 :    loss:0.559884
Epoch 3/128 :    loss:0.523445
Epoch 4/128 :    loss:0.511414
Epoch 5/128 :    loss:0.506760
Epoch 6/128 :    loss:0.504578
Epoch 7/128 :    loss:0.503120
Epoch 8/128 :    loss:0.501564
Epoch 9/128 :    loss:0.499329
Epoch 10/128 :   loss:0.495829
Epoch 11/128 :   loss:0.490441
Epoch 12/128 :   loss:0.482719
Epoch 13/128 :   loss:0.472824
Epoch 14/128 :   loss:0.461878
Epoch 15/128 :   loss:0.451617
Epoch 16/128 :   loss:0.443372
Epoch 17/128 :   loss:0.437473
Epoch 18/128 :   loss:0.433560
Epoch 19/128 :   loss:0.431093
Epoch 20/128 :   loss:0.429586
Epoch 21/128 :   loss:0.428669
Epoch 22/128 :   loss:0.428092
Epoch 23/128 :   loss:0.427699
Epoch 24/128 :   loss:0.427397
Epoch 25/128 :   loss:0.427135
Epoch 26/128 :   loss:0.426880
Epoch 27/128 :   loss:0.426615
Epoch 28/128 :   loss:0.426330
Epoch 29/128 :   loss:0.426018
Epoch 30/128 :   loss:0.425676
Epoch 31/128 :   loss:0.425302
Epoch 32/128 :   loss:0.424895
Epoch 33/128 :   loss:0.424456
Epoch 34/128 :   loss:0.423987
Epoch 35/128 :   loss:0.423488
Epoch 36/128 :   loss:0.422964
Epoch 37/128 :   loss:0.422416
Epoch 38/128 :   loss:0.421847
Epoch 39/128 :   loss:0.421262
Epoch 40/128 :   loss:0.420662
Epoch 41/128 :   loss:0.420053
Epoch 42/128 :   loss:0.419436
Epoch 43/128 :   loss:0.418815
Epoch 44/128 :   loss:0.418193
Epoch 45/128 :   loss:0.417573
Epoch 46/128 :   loss:0.416957
Epoch 47/128 :   loss:0.416347
Epoch 48/128 :   loss:0.415747
Epoch 49/128 :   loss:0.415157
Epoch 50/128 :   loss:0.414579
Epoch 51/128 :   loss:0.414015
Epoch 52/128 :   loss:0.413465
Epoch 53/128 :   loss:0.412931
Epoch 54/128 :   loss:0.412413
Epoch 55/128 :   loss:0.411913
Epoch 56/128 :   loss:0.411429
Epoch 57/128 :   loss:0.410963
Epoch 58/128 :   loss:0.410514
Epoch 59/128 :   loss:0.410083
Epoch 60/128 :   loss:0.409668
Epoch 61/128 :   loss:0.409271
Epoch 62/128 :   loss:0.408890
Epoch 63/128 :   loss:0.408526
Epoch 64/128 :   loss:0.408177
Epoch 65/128 :   loss:0.407843
Epoch 66/128 :   loss:0.407525
Epoch 67/128 :   loss:0.407220
Epoch 68/128 :   loss:0.406930
Epoch 69/128 :   loss:0.406652
Epoch 70/128 :   loss:0.406388
Epoch 71/128 :   loss:0.406135
Epoch 72/128 :   loss:0.405894
Epoch 73/128 :   loss:0.405664
Epoch 74/128 :   loss:0.405445
Epoch 75/128 :   loss:0.405236
Epoch 76/128 :   loss:0.405036
Epoch 77/128 :   loss:0.404846
Epoch 78/128 :   loss:0.404664
Epoch 79/128 :   loss:0.404491
Epoch 80/128 :   loss:0.404325
Epoch 81/128 :   loss:0.404167
Epoch 82/128 :   loss:0.404015
Epoch 83/128 :   loss:0.403871
Epoch 84/128 :   loss:0.403732
Epoch 85/128 :   loss:0.403600
Epoch 86/128 :   loss:0.403473
Epoch 87/128 :   loss:0.403352
Epoch 88/128 :   loss:0.403236
Epoch 89/128 :   loss:0.403125
Epoch 90/128 :   loss:0.403018
Epoch 91/128 :   loss:0.402915
Epoch 92/128 :   loss:0.402816
Epoch 93/128 :   loss:0.402721
Epoch 94/128 :   loss:0.402630
Epoch 95/128 :   loss:0.402542
Epoch 96/128 :   loss:0.402458
Epoch 97/128 :   loss:0.402376
Epoch 98/128 :   loss:0.402297
Epoch 99/128 :   loss:0.402221
Epoch 100/128 :          loss:0.402147
Epoch 101/128 :          loss:0.402076
Epoch 102/128 :          loss:0.402006
Epoch 103/128 :          loss:0.401939
Epoch 104/128 :          loss:0.401874
Epoch 105/128 :          loss:0.401811
Epoch 106/128 :          loss:0.401749
Epoch 107/128 :          loss:0.401689
Epoch 108/128 :          loss:0.401631
Epoch 109/128 :          loss:0.401573
Epoch 110/128 :          loss:0.401518
Epoch 111/128 :          loss:0.401463
Epoch 112/128 :          loss:0.401409
Epoch 113/128 :          loss:0.401357
Epoch 114/128 :          loss:0.401306
Epoch 115/128 :          loss:0.401255
Epoch 116/128 :          loss:0.401205
Epoch 117/128 :          loss:0.401156
Epoch 118/128 :          loss:0.401108
Epoch 119/128 :          loss:0.401060
Epoch 120/128 :          loss:0.401014
Epoch 121/128 :          loss:0.400967
Epoch 122/128 :          loss:0.400921
Epoch 123/128 :          loss:0.400876
Epoch 124/128 :          loss:0.400831
Epoch 125/128 :          loss:0.400786
Epoch 126/128 :          loss:0.400742
Epoch 127/128 :          loss:0.400698
Epoch 128/128 :          loss:0.400655
```

Exercise-9

Python Program for Artificial Neural Network (Keras).

In [9]:

```python
# Input: Dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values

# Encoding Categorical Values
labelencoder_X = LabelEncoder()
X[:, 2] = labelencoder_X.fit_transform(X[:, 2])
ct = ColumnTransformer([("Geography", OneHotEncoder(), [1])],
                        remainder = 'passthrough')
X = ct.fit_transform(X)

# Splitting the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# The ANN Model
def deep_model():
    classifier = Sequential()
    classifier.add(Dense(units=12, kernel_initializer='uniform',
                         activation='relu', input_dim=12))
    classifier.add(Dense(units=12, kernel_initializer='uniform',
                         activation='relu'))
    classifier.add(Dense(units=1, kernel_initializer='uniform',
                         activation='sigmoid'))
    classifier.compile(optimizer='rmsprop', loss='binary_crossentropy',
                       metrics=['accuracy'])
    return classifier

# Create ANN object
classifier = deep_model()

# Train the model using the training sets
classifier.fit(X_train, y_train, batch_size=4, epochs=128)

# Make predictions using the testing set
y_pred1 = classifier.predict(X_test)
y_pred1 = (y_pred1 > 0.5)

# Calculating the Confusion Matrix and Accuracy of the Model
cm1 = metrics.confusion_matrix(y_test, y_pred1)
accuracy1 = (cm1[0][0]+cm1[1][1])/(cm1[0][0]+cm1[0][1]+cm1[1][0]+cm1[1][1])
```

```
Epoch 1/128
2000/2000 [==============================] - 2s 559us/step - loss: 0.5045 - accuracy:
0.7907
Epoch 2/128
2000/2000 [==============================] - 1s 586us/step - loss: 0.4325 - accuracy:
0.8119
Epoch 3/128
2000/2000 [==============================] - 1s 551us/step - loss: 0.4033 - accuracy:
0.8341
Epoch 4/128
2000/2000 [==============================] - 1s 544us/step - loss: 0.4161 - accuracy:
0.8236
Epoch 5/128
2000/2000 [==============================] - 1s 532us/step - loss: 0.4012 - accuracy:
0.8389
Epoch 6/128
2000/2000 [==============================] - 1s 525us/step - loss: 0.4037 - accuracy:
0.8406
Epoch 7/128
2000/2000 [==============================] - 1s 525us/step - loss: 0.4066 - accuracy:
0.8385
Epoch 8/128
2000/2000 [==============================] - 1s 520us/step - loss: 0.4120 - accuracy:
0.8358
Epoch 9/128
2000/2000 [==============================] - 1s 525us/step - loss: 0.3940 - accuracy:
0.8360
Epoch 10/128
2000/2000 [==============================] - 1s 536us/step - loss: 0.4047 - accuracy:
0.8375
Epoch 11/128
2000/2000 [==============================] - 1s 566us/step - loss: 0.4072 - accuracy:
0.8326
Epoch 12/128
2000/2000 [==============================] - 1s 525us/step - loss: 0.3922 - accuracy:
0.8437
Epoch 13/128
2000/2000 [==============================] - 1s 573us/step - loss: 0.4062 - accuracy:
0.8329
Epoch 14/128
2000/2000 [==============================] - 1s 583us/step - loss: 0.4053 - accuracy:
0.8319
Epoch 15/128
2000/2000 [==============================] - 1s 586us/step - loss: 0.3950 - accuracy:
0.8408
Epoch 16/128
2000/2000 [==============================] - 1s 583us/step - loss: 0.3856 - accuracy:
0.8464
Epoch 17/128
2000/2000 [==============================] - 1s 596us/step - loss: 0.4056 - accuracy:
0.8344
Epoch 18/128
2000/2000 [==============================] - 1s 568us/step - loss: 0.4041 - accuracy:
0.8405
Epoch 19/128
2000/2000 [==============================] - 1s 517us/step - loss: 0.3947 - accuracy:
0.8414
Epoch 20/128
2000/2000 [==============================] - 1s 552us/step - loss: 0.3853 - accuracy:
0.8476
Epoch 21/128
2000/2000 [==============================] - 1s 559us/step - loss: 0.3988 - accuracy:
0.8377
Epoch 22/128
2000/2000 [==============================] - 1s 551us/step - loss: 0.4083 - accuracy:
0.8301
Epoch 23/128
2000/2000 [==============================] - 1s 551us/step - loss: 0.4056 - accuracy:
0.8331
Epoch 24/128
2000/2000 [==============================] - 1s 578us/step - loss: 0.3884 - accuracy:
0.8451
Epoch 25/128
2000/2000 [==============================] - 1s 544us/step - loss: 0.3857 - accuracy:
0.8408
Epoch 26/128
2000/2000 [==============================] - 1s 559us/step - loss: 0.3974 - accuracy:
0.8400
Epoch 27/128
2000/2000 [==============================] - 1s 551us/step - loss: 0.3954 - accuracy:
0.8375
Epoch 28/128
2000/2000 [==============================] - 1s 537us/step - loss: 0.4007 - accuracy:
0.8369
Epoch 29/128
2000/2000 [==============================] - 1s 563us/step - loss: 0.3881 - accuracy:
0.8435
Epoch 30/128
2000/2000 [==============================] - 1s 512us/step - loss: 0.3903 - accuracy:
0.8451
Epoch 31/128
2000/2000 [==============================] - 1s 543us/step - loss: 0.3859 - accuracy:
0.8428
Epoch 32/128
2000/2000 [==============================] - 1s 557us/step - loss: 0.3951 - accuracy:
0.8413
Epoch 33/128
2000/2000 [==============================] - 1s 552us/step - loss: 0.3895 - accuracy:
0.8453
Epoch 34/128
2000/2000 [==============================] - 1s 520us/step - loss: 0.3923 - accuracy:
0.8458
Epoch 35/128
2000/2000 [==============================] - 1s 552us/step - loss: 0.3824 - accuracy:
0.8458
Epoch 36/128
2000/2000 [==============================] - 1s 517us/step - loss: 0.3948 - accuracy:
0.8413
Epoch 37/128
2000/2000 [==============================] - 1s 509us/step - loss: 0.3952 - accuracy:
0.8421
Epoch 38/128
2000/2000 [==============================] - 1s 520us/step - loss: 0.3884 - accuracy:
0.8479
Epoch 39/128
2000/2000 [==============================] - 1s 517us/step - loss: 0.3905 - accuracy:
0.8453
Epoch 40/128
2000/2000 [==============================] - 1s 528us/step - loss: 0.3923 - accuracy:
0.8432
Epoch 41/128
2000/2000 [==============================] - 1s 552us/step - loss: 0.3893 - accuracy:
0.8452
Epoch 42/128
2000/2000 [==============================] - 1s 552us/step - loss: 0.3910 - accuracy:
0.8441
Epoch 43/128
2000/2000 [==============================] - 1s 559us/step - loss: 0.3911 - accuracy:
0.8472
Epoch 44/128
2000/2000 [==============================] - 1s 559us/step - loss: 0.3929 - accuracy:
0.8463
Epoch 45/128
2000/2000 [==============================] - 1s 544us/step - loss: 0.3926 - accuracy:
0.8432
Epoch 46/128
2000/2000 [==============================] - 1s 543us/step - loss: 0.3977 - accuracy:
0.8458
Epoch 47/128
2000/2000 [==============================] - 1s 559us/step - loss: 0.3867 - accuracy:
0.8465
Epoch 48/128
2000/2000 [==============================] - 1s 525us/step - loss: 0.3890 - accuracy:
0.8488
Epoch 49/128
2000/2000 [==============================] - 1s 512us/step - loss: 0.3960 - accuracy:
0.8463
Epoch 50/128
2000/2000 [==============================] - 1s 540us/step - loss: 0.3901 - accuracy:
0.8400
Epoch 51/128
2000/2000 [==============================] - 1s 567us/step - loss: 0.3969 - accuracy:
0.8446
Epoch 52/128
2000/2000 [==============================] - 1s 536us/step - loss: 0.3842 - accuracy:
0.8471
Epoch 53/128
2000/2000 [==============================] - 1s 548us/step - loss: 0.3904 - accuracy:
0.8485
Epoch 54/128
2000/2000 [==============================] - 1s 561us/step - loss: 0.3851 - accuracy:
0.8541
Epoch 55/128
2000/2000 [==============================] - 1s 567us/step - loss: 0.3782 - accuracy:
0.8555
Epoch 56/128
2000/2000 [==============================] - 1s 520us/step - loss: 0.3884 - accuracy:
0.8489
Epoch 57/128
2000/2000 [==============================] - 1s 525us/step - loss: 0.3867 - accuracy:
0.8504
Epoch 58/128
2000/2000 [==============================] - 1s 536us/step - loss: 0.3711 - accuracy:
0.8547
Epoch 59/128
2000/2000 [==============================] - 1s 544us/step - loss: 0.3528 - accuracy:
0.8650
Epoch 60/128
2000/2000 [==============================] - 1s 552us/step - loss: 0.3561 - accuracy:
0.8617
Epoch 61/128
2000/2000 [==============================] - 1s 559us/step - loss: 0.3567 - accuracy:
0.8611
Epoch 62/128
2000/2000 [==============================] - 1s 567us/step - loss: 0.3532 - accuracy:
0.8636
Epoch 63/128
2000/2000 [==============================] - 1s 536us/step - loss: 0.3488 - accuracy:
0.8674
Epoch 64/128
2000/2000 [==============================] - 1s 543us/step - loss: 0.3545 - accuracy:
0.8613
Epoch 65/128
2000/2000 [==============================] - 1s 564us/step - loss: 0.3466 - accuracy:
0.8640
Epoch 66/128
2000/2000 [==============================] - 1s 626us/step - loss: 0.3446 - accuracy:
0.8706
Epoch 67/128
2000/2000 [==============================] - 1s 570us/step - loss: 0.3400 - accuracy:
0.8689
Epoch 68/128
2000/2000 [==============================] - 1s 551us/step - loss: 0.3469 - accuracy:
0.8666
Epoch 69/128
2000/2000 [==============================] - 1s 533us/step - loss: 0.3564 - accuracy:
0.8676
Epoch 70/128
2000/2000 [==============================] - 1s 519us/step - loss: 0.3426 - accuracy:
0.8673
Epoch 71/128
2000/2000 [==============================] - 1s 507us/step - loss: 0.3435 - accuracy:
0.8649
Epoch 72/128
2000/2000 [==============================] - 1s 493us/step - loss: 0.3507 - accuracy:
0.8655
Epoch 73/128
2000/2000 [==============================] - 1s 536us/step - loss: 0.3520 - accuracy:
0.8672
Epoch 74/128
2000/2000 [==============================] - 1s 551us/step - loss: 0.3508 - accuracy:
0.8645
Epoch 75/128
2000/2000 [==============================] - 1s 517us/step - loss: 0.3559 - accuracy:
0.8655
Epoch 76/128
2000/2000 [==============================] - 1s 493us/step - loss: 0.3501 - accuracy:
0.8644
Epoch 77/128
2000/2000 [==============================] - 1s 528us/step - loss: 0.3533 - accuracy:
0.8656
Epoch 78/128
2000/2000 [==============================] - 1s 477us/step - loss: 0.3497 - accuracy:
0.8654
Epoch 79/128
2000/2000 [==============================] - 1s 557us/step - loss: 0.3558 - accuracy:
0.8645
Epoch 80/128
2000/2000 [==============================] - 1s 536us/step - loss: 0.3546 - accuracy:
0.8636
Epoch 81/128
2000/2000 [==============================] - 1s 552us/step - loss: 0.3609 - accuracy:
0.8606
Epoch 82/128
2000/2000 [==============================] - 1s 525us/step - loss: 0.3532 - accuracy:
0.8687
Epoch 83/128
2000/2000 [==============================] - 1s 492us/step - loss: 0.3544 - accuracy:
0.8653
Epoch 84/128
2000/2000 [==============================] - 1s 525us/step - loss: 0.3450 - accuracy:
0.8693
Epoch 85/128
2000/2000 [==============================] - 1s 564us/step - loss: 0.3456 - accuracy:
0.8689
Epoch 86/128
2000/2000 [==============================] - 1s 525us/step - loss: 0.3470 - accuracy:
0.8674
Epoch 87/128
2000/2000 [==============================] - 1s 492us/step - loss: 0.3426 - accuracy:
0.8732
Epoch 88/128
2000/2000 [==============================] - 1s 515us/step - loss: 0.3526 - accuracy:
0.8615
Epoch 89/128
2000/2000 [==============================] - 1s 556us/step - loss: 0.3567 - accuracy:
0.8644
Epoch 90/128
2000/2000 [==============================] - 1s 504us/step - loss: 0.3650 - accuracy:
0.8625
Epoch 91/128
2000/2000 [==============================] - 1s 524us/step - loss: 0.3654 - accuracy:
0.8603
Epoch 92/128
2000/2000 [==============================] - 1s 513us/step - loss: 0.3503 - accuracy:
0.8703
Epoch 93/128
2000/2000 [==============================] - 1s 527us/step - loss: 0.3419 - accuracy:
0.8729
Epoch 94/128
2000/2000 [==============================] - 1s 525us/step - loss: 0.3561 - accuracy:
0.8617
Epoch 95/128
2000/2000 [==============================] - 1s 546us/step - loss: 0.3577 - accuracy:
0.8673
Epoch 96/128
2000/2000 [==============================] - 1s 505us/step - loss: 0.3606 - accuracy:
0.8618
Epoch 97/128
2000/2000 [==============================] - 1s 548us/step - loss: 0.3610 - accuracy:
0.8645
Epoch 98/128
2000/2000 [==============================] - 1s 573us/step - loss: 0.3442 - accuracy:
0.8707
Epoch 99/128
2000/2000 [==============================] - 1s 567us/step - loss: 0.3444 - accuracy:
0.8704
Epoch 100/128
2000/2000 [==============================] - 1s 575us/step - loss: 0.3653 - accuracy:
0.8627
Epoch 101/128
2000/2000 [==============================] - 1s 594us/step - loss: 0.3496 - accuracy:
0.8690
Epoch 102/128
2000/2000 [==============================] - 1s 586us/step - loss: 0.3440 - accuracy:
0.8721
Epoch 103/128
2000/2000 [==============================] - 1s 571us/step - loss: 0.3683 - accuracy:
0.8664
Epoch 104/128
2000/2000 [==============================] - 1s 559us/step - loss: 0.3586 - accuracy:
0.8656
Epoch 105/128
2000/2000 [==============================] - 1s 567us/step - loss: 0.3728 - accuracy:
0.8568
Epoch 106/128
2000/2000 [==============================] - 1s 567us/step - loss: 0.3406 - accuracy:
0.8766
Epoch 107/128
2000/2000 [==============================] - 1s 528us/step - loss: 0.3529 - accuracy:
0.8699
Epoch 108/128
2000/2000 [==============================] - 1s 532us/step - loss: 0.3568 - accuracy:
0.8644
Epoch 109/128
2000/2000 [==============================] - 1s 605us/step - loss: 0.3594 - accuracy:
0.8626
Epoch 110/128
2000/2000 [==============================] - 1s 602us/step - loss: 0.3564 - accuracy:
0.8657
Epoch 111/128
2000/2000 [==============================] - 1s 575us/step - loss: 0.3496 - accuracy:
0.8673
Epoch 112/128
2000/2000 [==============================] - 1s 594us/step - loss: 0.3615 - accuracy:
0.8660
Epoch 113/128
2000/2000 [==============================] - 1s 586us/step - loss: 0.3418 - accuracy:
0.8725
Epoch 114/128
2000/2000 [==============================] - 1s 628us/step - loss: 0.3589 - accuracy:
0.8638
Epoch 115/128
2000/2000 [==============================] - 1s 598us/step - loss: 0.3579 - accuracy:
0.8658
Epoch 116/128
2000/2000 [==============================] - 1s 620us/step - loss: 0.3513 - accuracy:
0.8622
Epoch 117/128
2000/2000 [==============================] - 1s 623us/step - loss: 0.3558 - accuracy:
0.8695
Epoch 118/128
2000/2000 [==============================] - 1s 603us/step - loss: 0.3514 - accuracy:
0.8691
Epoch 119/128
2000/2000 [==============================] - 1s 609us/step - loss: 0.3613 - accuracy:
0.8657
Epoch 120/128
2000/2000 [==============================] - 1s 586us/step - loss: 0.3489 - accuracy:
0.8701
Epoch 121/128
2000/2000 [==============================] - 1s 536us/step - loss: 0.3495 - accuracy:
0.8659
Epoch 122/128
2000/2000 [==============================] - 1s 578us/step - loss: 0.3565 - accuracy:
0.8697
Epoch 123/128
2000/2000 [==============================] - 1s 610us/step - loss: 0.3495 - accuracy:
0.8722
Epoch 124/128
2000/2000 [==============================] - 1s 602us/step - loss: 0.3470 - accuracy:
0.8668
Epoch 125/128
2000/2000 [==============================] - 1s 624us/step - loss: 0.3608 - accuracy:
0.8631
Epoch 126/128
2000/2000 [==============================] - 1s 586us/step - loss: 0.3505 - accuracy:
0.8726
Epoch 127/128
2000/2000 [==============================] - 1s 583us/step - loss: 0.3603 - accuracy:
0.8645
Epoch 128/128
2000/2000 [==============================] - 1s 578us/step - loss: 0.3451 - accuracy:
0.8677
```

Exercise-9

Output and Comparison of Both Methods.

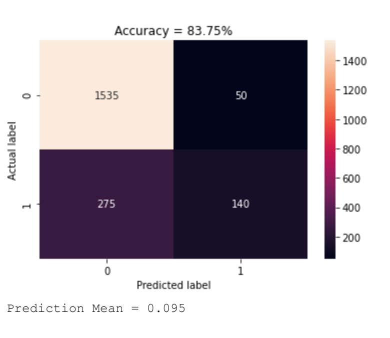
In [13]:

```python
# For Manual Method
# Output: The Confusion Matrix, Accuracy and the Prediction Mean
print("\n\nACCURACY METRIC OF ARTIFICIAL NEURAL NETWORK (MANUAL)")
print("\nThe Confusion Matrix for the ANN Model\n")
sns.heatmap(cm, annot=True, fmt='g')
plt.title('Accuracy = {0:.2f}%'.format(accuracy))
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
print("\nPrediction Mean = " + str(np.mean(y_pred)))
print("\n____________________________________________________________\n")

# For Keras Method
# Output: The Confusion Matrix, Accuracy and the Prediction Mean
print("\n\nACCURACY METRIC OF ARTIFICIAL NEURAL NETWORK (KERAS)")
print("\nThe Confusion Matrix for the ANN Model\n")
sns.heatmap(cm1, annot=True, fmt='g')
plt.title('Accuracy = {0:.2f}%'.format(accuracy1*100))
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
print("\nPrediction Mean = " + str(np.mean(y_pred1)))
```

```
ACCURACY METRIC OF ARTIFICIAL NEURAL NETWORK (MANUAL)

The Confusion Matrix for the ANN Model
```

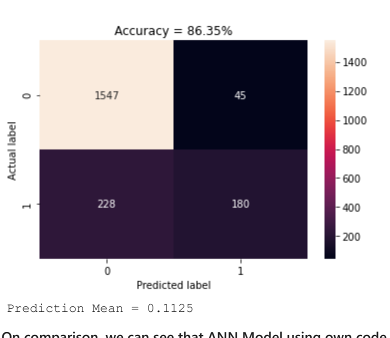



```
Prediction Mean = 0.095

____________________________________________________________


ACCURACY METRIC OF ARTIFICIAL NEURAL NETWORK (KERAS)

The Confusion Matrix for the ANN Model
```




```
Prediction Mean = 0.1125
```

On comparison, we can see that ANN Model using own code is less accurate as compared to ANN Model with Keras. But, both models can be considered as a good fit for prediction of unknown values.