INTELLIGENT SYSTEMS LAB-5 (20/09/2021)

***NAME**: Harshvardhan Agarwal | **SEC**: A | **REG NO**.: 201800524*

PROBLEM STATEMENT

Implement a backpropagation algorithm to experiment with the use of Neural Networks for a multiclass classification problem, and try and interpret the high-level or hidden representations learned by it.

1. Submit the pdf file with code and graphs (py, doc file will not be evaluated)
2. Analyze the algorithm by varying the number of input and hidden layers.
3. Analyze the output by varying the number of epochs.

PROBLEM SOLUTION

SOURCE CODE

```
# Artificial Neural Network
```

```
###·Importing·the·libraries
import·numpy·as·np
import·pandas·as·pd
import·tensorflow·as·tf
tf.__version__
```

```
'2.6.0'
```

```
## Part 1 - Data Preprocessing
### Importing the dataset

dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
print(X)
print(y)
```

```
[[619 'France' 'Female' ... 1 1 101348.88]
 [608 'Spain' 'Female' ... 0 1 112542.58]
 [502 'France' 'Female' ... 1 0 113931.57]
 ...
 [709 'France' 'Female' ... 0 1 42085.58]
 [772 'Germany' 'Male' ... 1 0 92888.52]
 [792 'France' 'Female' ... 1 0 38190.78]]
[1 0 1 ... 1 1 0]
```

```
### Encoding categorical data
##Label Encoding the "Gender" column
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```

```
print(X)
```

```
[[619 'France' 0 ... 1 1 101348.88]
 [608 'Spain' 0 ... 0 1 112542.58]
 [502 'France' 0 ... 1 0 113931.57]
 ...
 [709 'France' 0 ... 0 1 42085.58]
 [772 'Germany' 1 ... 1 0 92888.52]
 [792 'France' 0 ... 1 0 38190.78]]
```

```
##One Hot Encoding the "Geography" column
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passth
X = np.array(ct.fit_transform(X))
print(X)
```

```
[[1.0 0.0 0.0 ... 1 1 101348.88]
 [0.0 0.0 1.0 ... 0 1 112542.58]
 [1.0 0.0 0.0 ... 1 0 113931.57]
 ...
 [1.0 0.0 0.0 ... 0 1 42085.58]
 [0.0 1.0 0.0 ... 1 0 92888.52]
 [1.0 0.0 0.0 ... 1 0 38190.78]]
```

```
###Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
### Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
##Part 2 - Building the ANN
###Initializing the ANN
ann = tf.keras.models.Sequential()
```

```
##Adding the input layer and the first hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

```
###Adding the second hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
### Adding the output layer
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

```
## Part 3 - Training the ANN
### Compiling the ANN
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
### Training the ANN on the Training set
```

```
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
      250/250 [==============================] - 0s 1ms/step - loss: 0.3352 - accuracy:
      Epoch 73/100
       250/250 [==============================] - 0s 1ms/step - loss: 0.3352 - accuracy:
      250/250 [ ] 0s 1ms/step loss: 0.3352 accuracy:  Epoch 74/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3349 - accuracy:
      Epoch 75/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3349 - accuracy:
      Epoch 76/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3351 - accuracy:
      Epoch 77/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3346 - accuracy:
      Epoch 78/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3345 - accuracy:
      Epoch 79/100
      250/250 [==============================] - 0s 2ms/step - loss: 0.3346 - accuracy:
      Epoch 80/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3348 - accuracy:
      Epoch 81/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3349 - accuracy:
      Epoch 82/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3344 - accuracy:
      Epoch 83/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3338 - accuracy:
      Epoch 84/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3342 - accuracy:
      Epoch 85/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3347 - accuracy:
      Epoch 86/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3338 - accuracy:
      Epoch 87/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3340 - accuracy:
      Epoch 88/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3343 - accuracy:
      Epoch 89/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3343 - accuracy:
      Epoch 90/100
      250/250 [==============================] - 0s 2ms/step - loss: 0.3343 - accuracy:
      Epoch 91/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3342 - accuracy:
      Epoch 92/100
       250/250 [==============================] - 0s 1ms/step - loss: 0.3343 - accuracy:
      Epoch 93/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3341 - accuracy:
      Epoch 94/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3340 - accuracy:
      Epoch 95/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3340 - accuracy:
      Epoch 96/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3338 - accuracy:
      Epoch 97/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3340 - accuracy:
      Epoch 98/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3341 - accuracy:
      Epoch 99/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3338 - accuracy:
      Epoch 100/100
      250/250 [==============================] - 0s 1ms/step - loss: 0.3341 - accuracy:
      <keras.callbacks.History at 0x7f3dfd615550>
```

```
##Part 4 - Making the predictions and evaluating the model
print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])) > 0.5)
       di ti th t t lt

### Predicting the Test set results
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
 print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

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

```
###Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
     [[1530 65]
      [ 211 194]]
     0.862
```