COMP4220: Machine Learning, Spring 2022, Assignment 5

Please submit one pdf file for all questions.

1. List five hyperparameters you can tweak in a basic neural network?

Numbers of layers, number of neurons per layer, activation functions, learning rate, and batch size.

2. What is backpropagation and how does it work?

Going from the output layer to the input layer of neural network to optimize weights to minimize loss function using the chain rule.

Programming Assignment (Artificial Neural Network-ANN)

```
# Importing libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.compose import ColumnTransformer
import keras
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, precision_score,
accuracy_score, f1_score, recall_score
import matplotlib.pyplot as plt
```

```
# Importing the dataset. This dataset describes churning, which is
# the rate at which customers stop doing business with a company
dataset = pd.read_csv('Churn_Modelling.csv')
dataset
```

```
      RowNumber  CustomerId    Surname  CreditScore Geography  Gender
Age  \
0             1    15634602   Hargrave          619    France  Female
42
1             2    15647311       Hill          608     Spain  Female
41
2             3    15619304       Onio          502    France  Female
42
3             4    15701354       Boni          699    France  Female
```

```
39
4             5    15737888   Mitchell          850     Spain  Female
43
...         ...         ...        ...          ...       ...     ...
...
9995       9996    15606229   Obijiaku          771    France    Male
39
9996       9997    15569892  Johnstone          516    France    Male
35
9997       9998    15584532        Liu          709    France  Female
36
9998       9999    15682355  Sabbatini          772   Germany    Male
42
9999      10000    15628319     Walker          792    France  Female
28

      Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0          2       0.00              1          1               1
1          1   83807.86              1          0               1
2          8  159660.80              3          1               0
3          1       0.00              2          0               0
4          2  125510.82              1          1               1
...      ...        ...            ...        ...             ...
9995       5       0.00              2          1               0
9996      10   57369.61              1          1               1
9997       7       0.00              1          0               1
9998       3   75075.31              2          1               0
9999       4  130142.79              1          1               0

      EstimatedSalary  Exited
0           101348.88       1
1           112542.58       0
2           113931.57       1
3            93826.63       0
4            79084.10       0
...               ...     ...
9995         96270.64       0
9996        101699.77       0
9997         42085.58       1
9998         92888.52       1
9999         38190.78       0

[10000 rows x 14 columns]
```

1. Looking at the dataset we can see that the first 3 columns are not essential for our model.

Make a X variable that contains all other columns except the first three columns and Exited (label) Make a Y variable (the Exited column)

```
X = dataset.iloc[:, 3:13].values # Select input features X
y = dataset.iloc[:, 13].values   # The last column "Exited" is the
output variable Y

print(X.shape, y.shape)
```

```
(10000, 10) (10000,)
```

2. In X there are Geography and Gender columns that are in string format which we can't use for training. Thus we should transform them into numerical type to train our model.

Use LabelEncoder and OneHotEncoder from sklearn.preprocessing to transform the "Geography" and "Gender" columns into numberical data type

```
# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X_1 = LabelEncoder()
X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
labelencoder_X_2 = LabelEncoder()
X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
ct = ColumnTransformer([("Geography", OneHotEncoder(), [1])],
remainder = 'passthrough')
X = ct.fit_transform(X)
```

3. Split the dataset into the Training set and Test set (test_size = 0.2)

```
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, stratify=y, random_state=0)
```

4. Apply Feature Scaling to all features before training a neural network

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train))
X_test = pd.DataFrame(scaler.transform(X_test))
```

5. Let's build ANN model by using the Keras sequential package

Initalize the sequential model Add the input layer and the first hidden layer Hint: For the first layer use (units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11)

```
import keras
from keras.models import Sequential
from keras.layers import Dense

classifier = Sequential()
classifier.add(Dense(units = 6,  kernel_initializer  = 'uniform',
activation = 'relu'))
```

6. Add the second hidden layer

Hint:(units = 6, kernel_initializer = 'uniform', activation = 'relu')

```
classifier.add(Dense(units = 6,  kernel_initializer = 'uniform',
activation = 'relu'))
```

7. Add the output layer

Hint: (units = 1, kernel_initializer = 'uniform', activation = 'sigmoid')

```
classifier.add(Dense(units = 1,kernel_initializer = 'uniform',
activation = 'sigmoid'))
```

8. Compile the ANN

hint: (optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy']))

```
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy',
metrics = ['accuracy'])
```

9. Fit the ANN to the training set

(batch_size = 5, epochs = 20)

```
batch_size = 5
epochs = 20
#history = classifier.fit()
classifier.fit(X_train, y_train, batch_size = 5, epochs = 20)
```

```
Epoch 1/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.4699 - accuracy: 0.7956
Epoch 2/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.4163 - accuracy: 0.7962
Epoch 3/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.4035 - accuracy: 0.8135
Epoch 4/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3920 - accuracy: 0.8303
Epoch 5/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3828 - accuracy: 0.8371
Epoch 6/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3762 - accuracy: 0.8421
Epoch 7/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3707 - accuracy: 0.8466
Epoch 8/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3669 - accuracy: 0.8519
```

```
Epoch 9/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3629 - accuracy: 0.8508
Epoch 10/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3604 - accuracy: 0.8529
Epoch 11/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3584 - accuracy: 0.8534
Epoch 12/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3576 - accuracy: 0.8561
Epoch 13/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3557 - accuracy: 0.8571
Epoch 14/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3555 - accuracy: 0.8561
Epoch 15/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3532 - accuracy: 0.8585
Epoch 16/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3536 - accuracy: 0.8585
Epoch 17/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3521 - accuracy: 0.8583
Epoch 18/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3525 - accuracy: 0.8571
Epoch 19/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3513 - accuracy: 0.8575
Epoch 20/20
1600/1600 [==============================] - 3s 2ms/step - loss:
0.3505 - accuracy: 0.8584

<keras.callbacks.History at 0x7fc566f64090>
```

10. Make predictions and evaluate the model

hint: just consider y_pred the values where y_pred is greater than 0.5 (y_pred = (y_pred > 0.5)) Make the confusion matrix and show the result Evalue the precision, accuracy, recall, and f1 score and show the result

```
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

from sklearn.metrics import confusion_matrix, accuracy_score,
f1_score, precision_score, recall_score
acc = accuracy_score(y_test, y_pred)
```

```
matrix = pd.DataFrame([['ANN', acc]],
                      columns = ['Model', 'Accuracy'])

matrix
```

```
  Model  Accuracy
0   ANN    0.8545
```

11. Compute the accuracy, precision, recall, and f1 score

```
print('Accuracy: {}'.format(accuracy_score(y_test, y_pred)))
print('Precision: {}'.format(precision_score(y_test, y_pred)))
print('Recall: {}'.format(recall_score(y_test, y_pred)))
print('F1 Score: {}'.format(f1_score(y_test, y_pred)))
```

```
Accuracy: 0.8545
Precision: 0.8186813186813187
Recall: 0.36609336609336607
F1 Score: 0.5059422750424448
```

12. Using Tensorflow Playground

Visit the TensorFlow Playground at https://playground.tensorflow.org/

Spend some time playing with this UI to grow your intuition about neural networks. Complete the following problems in a single sitting please.

1. Layers and patterns: try training the default neural network by clicking the run button (top left). Notice how it quickly finds a good solution for the classification task. Notice that the neurons in the first hidden layer have learned simple patterns, while the neurons in the second hidden layer have learned to combine the simple patterns of the first hidden layer into more complex patterns. What happens when you add more layers?

Adding more layers increases the number of weights in the neural network and increases the complexity

1. Activation function: try replacing the Tanh activation function with the ReLU activation function, and train the network again. Notice that it finds a solution even faster, but this time the boundaries are linear. What about the ReLU function causes this?

The boundaries are linear because it is linear for X>0, but non-differentiable at Z=0.

1. Local minima: modify the network architecture to have just one hidden layer with three neurons. Train it multiple times (to reset the network weights, click the reset button next to the play button). What do you notice about the training time?

The training time increases.

1. Too small: now remove one neuron to keep just 2. Notice that the neural network is now incapable of finding a good solution, even if you try multiple times. What do you observe about the number of parameters and the training set?

The number of parameters decreases since the number of neurons per layer is a hyperparameter and causes underfitting.

1. Large enough: next, set the number of neurons to 8 and train the network several times. Notice that it is now consistently fast and never gets stuck. What do you observe about local minima?

Less often of a local minima which means less chances of it getting stuck