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?

Number of Neurons, optimizer, learning rate, epochs, and batch size.

2. What is backpropagation and how does it work?

Backpropogation is an algorithm thats used to train forward fed neural networks. These algorithms process the gradient of a loss function with respect to its own weights. This is also done with a single input and output only, instead of a table of outputs like naive algorithms

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	Tenur
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	1
9997	9998	15584532	Liu	709	France	Female	36	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	
9999	10000	15628319	Walker	792	France	Female	28	

10000 rows × 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
```

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([("Geogrophy", 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=0.20)
```

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

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = 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(X_train, y_train, batch_size = 5, epochs = 20)
```

```
Epoch 1/20
1600/1600 [==============================] - 3s 1ms/step - loss: 0.4640 - accuracy: 0.7950
Epoch 2/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4303 - accuracy: 0.7950
Epoch 3/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4233 - accuracy: 0.8095
Epoch 4/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4179 - accuracy: 0.8279
Epoch 5/20
1600/1600 [==============================] - 2s 2ms/step - loss: 0.4155 - accuracy: 0.8306
Epoch 6/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4127 - accuracy: 0.8310
Epoch 7/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4118 - accuracy: 0.8338
Epoch 8/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4101 - accuracy: 0.8341
Epoch 9/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4096 - accuracy: 0.8339
Epoch 10/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4075 - accuracy: 0.8341
Epoch 11/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4070 - accuracy: 0.8336
Epoch 12/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4069 - accuracy: 0.8356
Epoch 13/20
1600/1600 [==============================] - 2s 2ms/step - loss: 0.4064 - accuracy: 0.8353
Epoch 14/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4059 - accuracy: 0.8340
Epoch 15/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4058 - accuracy: 0.8345
Epoch 16/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4050 - accuracy: 0.8355
Epoch 17/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4049 - accuracy: 0.8347
Epoch 18/20
```

```
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4045 - accuracy: 0.8360
Epoch 19/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4040 - accuracy: 0.8360
Epoch 20/20
1600/1600 [==============================] - 2s 1ms/step - loss: 0.4039 - accuracy: 0.8331
```

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

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: 0.8405
Precision: 0.7468354430379747
Recall: 0.2972292191435768
F1 Score: 0.4252252252252252
```

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?
2. 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?
3. 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?
4. 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?
5. 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?

1. By increasing the number of layers in the network, the amount of training and test loss is eventually reduced to 0. The weight values in the output also increase towards -1 and 1 as their are only two groups in the first example.
2. The relu function is given as a piecewise linear function that will always output a positive directly, and otherwise output a 0. This takes out dealing with any negative weights values.
3. The run time when using one layer with three neurons varies significantly. Sometimes the training time is reasonable, other times the training is stuck on the local minima itself.
4. by removing a neuron we remove a parameter. When there's only two neurons with one layer, a solution cannot be found because of the lack of parameters. This causes the data to visually be underfitted.
5. Setting the amount of neurons to eight increases the speed significantly and the training becomes much more consistent. This means that with more features (neurons) there's much less likely for training to stop at a local minima.

✓ 0s completed at 8:12 PM