Report on the Neural Network Model

Victoria Giles

Module 21 Challenge

Overview

The nonprofit foundation Alphabet Soup wants a tool that can help it select the applicants for funding with the best chance of success in their ventures. With my knowledge of machine learning and neural networks, I used the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup.

From Alphabet Soup's business team, I received a CSV containing more than 34,000 organisations that have received funding from Alphabet Soup over the years. Within this dataset are a number of columns that capture metadata about each organisation, such as:

- **EIN** and **NAME** - Identification columns
- **APPLICATION_TYPE** - Alphabet Soup application type
- **AFFILIATION** - Affiliated sector of industry
- **CLASSIFICATION** - Government organization classification
- **USE_CASE** - Use case for funding
- **ORGANIZATION** - Organization type
- **STATUS** - Active status
- **INCOME_AMT** - ncome classification
- **SPECIAL_CONSIDERATIONS** - Special consideration for application
- **ASK_AMT** - Funding amount requested
- **IS_SUCCESSFUL** - Was the money used effectively

Data Preprocessing

I began the data preprocessing by removing the non-beneficial ID columns **EIN** and **NAME** from the input data because they are neither targets nor features. The target variable used for the model was **IS_SUCCESSFUL** which displayed a value of 1 if the funding was successful or 0 if it was not.

The feature variables used for the model are listed below;

- **APPLICATION_TYPE**
- **AFFILIATION**
- **CLASSIFICATION**
- **USE_CASE**
- **ORGANIZATION**
- **STATUS**
- **INCOME_AMT**
- **SPECIAL_CONSIDERATIONS**
- **ASK_AMT**

In [3]:
```
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df.drop(columns = ['EIN', 'NAME'], inplace=True)
application_df.head()
```

Out[3]:

	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	STATUS	INCOME_AMT	SPECIAL_CONSIDERATIONS	ASK_AMT	IS_
0	T10	Independent	C1000	ProductDev	Association	1	0	N	5000	
1	T3	Independent	C2000	Preservation	Co-operative	1	1-9999	N	108590	
2	T5	CompanySponsored	C3000	ProductDev	Association	1	0	N	5000	
3	T3	CompanySponsored	C2000	Preservation	Trust	1	10000-24999	N	6692	
4	T3	Independent	C1000	Heathcare	Trust	1	100000-499999	N	142590	

In [5]:
```
# Determine the number of unique values in each column.
application_df.nunique()
```

Out[5]:
```
APPLICATION_TYPE            17
AFFILIATION                  6
CLASSIFICATION              71
USE_CASE                     5
ORGANIZATION                 4
STATUS                       2
INCOME_AMT                   9
SPECIAL_CONSIDERATIONS       2
ASK_AMT                   8747
IS_SUCCESSFUL                2
dtype: int64
```

Compiling, Training, and Evaluating the Model

The first model I built with the parameters of;

- Two hidden layers
- With 80, 30 neurons split
- Hidden layer activation function of 'relu'

Compile, Train and Evaluate the Model

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len(X_train_scaled[0])

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=80, activation="relu", input_dim = number_input_features))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=30, activation="relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units= 1, activation="sigmoid"))

# Check the structure of the model
nn.summary()
```

The model performance was below 75% accuracy which is not satisfactory.

```
# Evaluate the model using the test data to determine the loss and accuracy
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 0s - loss: 0.5628 - accuracy: 0.7290 - 234ms/epoch - 873us/step
Loss: 0.5628108382225037, Accuracy: 0.7289795875549316
```

So to increase model performance, I experimented with changing other parameters, such as dropping an additional two ID columns (**SPECIAL_CONSIDERATIONS** and **USE_CASE**) and increasing nodes and neurons in the two hidden layers.

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len(X_train_scaled[0])

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=90, activation="relu", input_dim = number_input_features))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=50, activation="relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units= 1, activation="sigmoid"))

# Check the structure of the model
nn.summary()
```

Despite this, the second model also returned a below 75% accuracy score

```
# Evaluate the model using the test data to determine the loss and accuracy
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
268/268 - 0s - loss: 0.5628 - accuracy: 0.7289 - 406ms/epoch - 2ms/step
Loss: 0.5628221035003662, Accuracy: 0.728863000869751
```

Summary

The models I worked on, before and after optimisation, were only able to achieve around 73% accuracy. I tested several models, changing the number of hidden layers, nodes, epochs and activation functions. The optimisation changes to the model only made slight improvements in accuracy.
Unfortunately, I was not able to reach the target performance of 75%. Each model tested would not get an accuracy rate higher than about 73%. I would recommend to work on a larger dataset and finding the optimal number of nodes and hidden layers, as well as the best activation function for each hidden layer.