Deep Learning Challenge - Neural Network Model Report

Overview:

The purpose of this analysis is to review the neural network model built to support the nonprofit foundation Alphabet Soup. The neural network is designed to help select applicants for funding with the best chance of venture success of their ventures.

Results:

The highest performing results of this model were 67% accuracy, falling below the defined success threshold of 75%.

Data Preprocessing:

What variable(s) are the target(s) for your model?

The variables set as features were the IS_SUCCESSFUL field.

What variable(s) are the features for your model?

The variables set as features were application type, affiliation, classification, use case, organization, status, income amount, special considerations, ask amount and success.

What variable(s) should be removed from the input data because they are neither targets nor features?

Variables that should be removed from the input data included EIN and name, as they were neither target nor features. The .drop() function was used to remove these columns from the dataset.

```
[6] # Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
    updated_applications = application_df.drop(columns=['NAME' , 'EIN'], axis=1)
    updated_applications.head()
```

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

```
[14] # Split our preprocessed data into our features and target arrays
     x = numeric_data.drop(columns = ['IS_SUCCESSFUL']).values
     y = numeric_data['IS_SUCCESSFUL'].values

     # Split the preprocessed data into a training and testing dataset
     X_train, X_test, Y_train, Y_test = train_test_split(x,y, random_state=42)
```

Compiling, Training, and Evaluating the Model

I selected the ReLu activation function for my hidden layers and assigned sigmoid to the output layer. I selected ReLU as it is widely used on hidden layers in large datasets for its ability to speed up training as well as help prevent vanishing gradients. In my original model, I created two hidden layers and assigned eight neurons to the first hidden layer and five to the second hidden layer as a starting point to test initial performance.

```
#Keras Sequential Model
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu"))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation="relu"))

# Fourth hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer4, activation="relu"))

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

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

```
Input Features = 19611
Hidden Nodes - Layer 1 = 100
Hidden Nodes - Layer 2 = 50
Hidden Nodes - Layer 3 = 20
Hidden Nodes - Layer 4 = 20
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               1961200

 dense_1 (Dense)             (None, 50)                5050

 dense_2 (Dense)             (None, 20)                1020

 dense_3 (Dense)             (None, 20)                420

 dense_4 (Dense)             (None, 1)                 21

=================================================================
Total params: 1,967,711
Trainable params: 1,967,711
Non-trainable params: 0
```

Were you able to achieve the target model performance?

I was not able to achieve the target model performance of 75% in my neural network optimizations. My final optimization test reached 67% accuracy, increasing from 45% on the original trial, followed by 65% accuracy before reaching the final output.

```
[21] # Evaluate the model using the test data
     model_loss, model_accuracy = nn.evaluate(X_test_scaled,Y_test,verbose=2)
     print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

     268/268 - 1s - loss: 0.8809 - accuracy: 0.6767 - 998ms/epoch - 4ms/step
     Loss: 0.8808674812316895, Accuracy: 0.6767346858978271
```

What steps did you take in your attempts to increase model performance?

Steps I took to increase model performance in my optimizations included adding additional hidden layers, increasing the neurons for each hidden layer, and adjusting the number of total epochs when training the model. I found that increasing the number of epochs each time did not consistently result in higher accuracy as potential overfitting occurred. I began with only two hidden layers and concluded the optimization tests with four hidden layers and as high as 100 neurons on one of the hidden layers.

Summary:

The results of the model created indicate that there is still room for further optimization of the model and that the model will not perfectly indicate success of the nonprofit funding applicants.

A larger dataset could have benefitted this model, as well as a different model. Other model types that could be well-suited for this data analysis would be SVM or Random Forest Classifier as they have a great track record for classification problems.