Report on the Neural Network Model for Alphabet Soup

Overview of the Analysis

The purpose of this analysis is to evaluate the performance of a deep learning model designed to predict the success of applicants funded by the nonprofit organization Alphabet Soup. This report will detail the data preprocessing steps, the architecture of the neural network, the model's performance metrics, and recommendations for future model improvements.

Results

Data Preprocessing

- **Target Variable(s):**
    - The target variable for the model is the binary outcome indicating whether an applicant was successful if funded (e.g., 'IS_SUCCESSFUL')
- **Feature Variable(s):**
    - The features for the model may include various applicant characteristics such as application type, income_amt, Organisation).
- **Variables to Remove:**
    - Any variables that do not contribute to the prediction of success, such as unique identifiers (e.g., EIN, Name), were removed from the input data.

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

```
# Determine the number of unique values in each column.
application_df.nunique()
```

	0
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

However, Name was reintroduced during the second test to assist with binning. We reviewed the APLICATION data, and utilized the “CLASSIFICATION” column for binning purposes’ Several data points were designed as thresholds to group infrequent values under a new category, “Other”

for each unique value. Once binning was verified to be effective, categorical variables were transformed using the get-dummies () function to encode them.

```
# Convert categorical data to numeric with `pd.get_dummies`
application_df = pd.get_dummies(application_df,dtype=float)
application_df.head()
```

	STATUS	ASK_AMT	IS_SUCCESSFUL	APPLICATION_TYPE_Other	APPLICATION_TYPE_T10	APPLICATION_TYPE_T19	APPLICATION_TYPE_T
0	1	5000	1	0.0	1.0	0.0	0.
1	1	108590	1	0.0	0.0	0.0	1.
2	1	5000	0	0.0	0.0	0.0	0.
3	1	6692	1	0.0	0.0	0.0	1.
4	1	142590	1	0.0	0.0	0.0	1.

5 rows × 109 columns

Compiling, Training, and Evaluating the Model

- **Neurons, Layers, and Activation Functions:**
    - The model was structured with three layers, and the number of hidden nodes were dictated by the number of features. The activation functions used were “relu”, activation functions, for hidden layers it was also “relu” and sigmoid for the output layer to introduce non-linearity and enable the model to learn complex patterns.

```
# 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])
hidden_nodes_layer1=7
hidden_nodes_layer2=14
hidden_nodes_layer3=21

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'))

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

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

- **Achieving Target Model Performance:**
    - 890 parameters were created by the three-layer training model. The initial attempt was just over 73% accuracy, which was a little bit under the targeted outcome of 75%.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 7)	763
dense_1 (Dense)	(None, 14)	112
dense_2 (Dense)	(None, 1)	15

Total params: 890 (3.48 KB)

Trainable params: 890 (3.48 KB)

Non-trainable params: 0 (0.00 B)

```
# 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 - 0s - 2ms/step - accuracy: 0.7325 - loss: 0.5537
Loss: 0.553676962852478, Accuracy: 0.732478141784668
```

- **Steps to Increase Model Performance:**
  - To enhance model performance, the following steps were taken:
    - Reintroduced NAME to assist with binning
    - Adjusted the learning rate and batch size during training.
    - Changed cut-off value from counts<500 to counts<100

```
# Choose a cutoff value and create a list of classifications to be replaced
# use the variable name `classifications_to_replace`
classifications_to_replace = list(counts[counts<100].index)
classifications_to_replace

# Replace in dataframe
for cls in classifications_to_replace:
    application_df['CLASSIFICATION'] = application_df['CLASSIFICATION'].replace(cls,"Other")

# Check to make sure binning was successful
application_df['CLASSIFICATION'].value_counts()
```


    - Adjusted the learning rate and batch size during training.

```
# 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])
hidden_nodes_layer1=7
hidden_nodes_layer2=14
hidden_nodes_layer3=21
nn = tf.keras.models.Sequential()

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'))

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

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

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 7)	3,591
dense_1 (Dense)	(None, 14)	112
dense_2 (Dense)	(None, 1)	15

```
Total params: 3,718 (14.52 KB)
Trainable params: 3,718 (14.52 KB)
Non-trainable params: 0 (0.00 B)
```

The optimization attempt achieved an accuracy of almost 79% which over the targeted outcome of 75%.

```
# 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 - 0s - 1ms/step - accuracy: 0.7871 - loss: 0.4698
Loss: 0.4698360562324524, Accuracy: 0.7870553731918335
```

Summary

The deep learning model demonstrated promising results in predicting the success of applicants for Alphabet Soup funding. The model achieved an accuracy of 73% in the first instance and 79% after taking some steps to increase performance. This shows a strong ability to generalize on unseen data.

Recommendation for Alternative Model: While the deep learning model performed well, a simpler model such as a Random Forest classifier could also be effective for this classification problem. Random Forests are robust to overfitting and can handle both numerical and categorical

data without extensive preprocessing. This could provide a more interpretable model while maintaining competitive performance.