Alphabet Soup Nonprofit Deep Learning 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 your knowledge of machine learning and neural networks, you'll use 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, you have received a CSV containing more than 34,000 organizations that have received funding from Alphabet Soup over the years. Within this dataset are a number of columns that capture metadata about each organization, 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—Income classification
- SPECIAL_CONSIDERATIONS—Special considerations for application
- ASK_AMT—Funding amount requested
- IS_SUCCESSFUL—Was the money used effectively

Results:

Data Processing:

To start processing the data, we cleaned the data and removed any information that would not be relevant to creating a predictor. This resulted in dropping EIN and NAME. However, NAME was eventually added back during the second test for binning purposes. Next, the data was split into two separate sets, training and testing. The variable targeted was IS_SUCCESSFUL. This field captured yes (1) and no (0). The applicate dataframe was analyzed and was determined the CLASSIFICATION value was needed for binning.

Several other data points were used as a stopping point to bin together rare variables with a new value of Other for each unique value. Categorical variables were encoded to get_dummies() after checking to see if the binning was successful.

```python
# 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_T3	APPLICATION_TYPE_T4	APPLICATION_TYPE_T5	APPLICATION_TYPE_T6	...	INCOME_AMT_1-9999	INCOME_AMT_10000-24999	INCOME_A
0	1	5000	1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
1	1	108590	1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	1.0	0.0	
2	1	5000	0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	
3	1	6692	1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	1.0	
4	1	142590	1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	

5 rows × 50 columns

Compiling, Training, and Evaluating the Model:

There were approximately three layers for each model after applying Neural Networks. The number of hidden nodes were dictated by the number of features. In total, there were 477 parameters created and the first attempt was about 73%. This is unfortunately under the desired 75%.

```python
# 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()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 350

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

=================================================================
Total params: 477 (1.86 KB)
Trainable params: 477 (1.86 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```python
# 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: 0.5516 - accuracy: 0.7306 - 555ms/epoch - 2ms/step
Loss: 0.5516066551208496, Accuracy: 0.7306122183799744
```

Optimization:

Next, the module was adjusted to attempt to meet the accuracy goal. To try and achieve this, the NAME field was added back in. This time, accuracy exceeded with 79% and had a total of 3,298 parameters.

```python
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()

nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation='relu'))

nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation='relu'))

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

nn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 7)                 3171

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

=================================================================
Total params: 3298 (12.88 KB)
Trainable params: 3298 (12.88 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Summary:

In conclusion, deep learning models should have multiple layers because the machine based teaches the model (computer) to filter inputs through layers on how to predict and classify more information.