Deep Learning Challenge: Alphabet Soup Charity Optimization

Sakurako Kikuchi

Overview

The nonprofit foundation Alphabet Soup needs a tool that helps it determine the applicants for funding with the best chance of success in their ventures. The purpose of this report is to summarize the overall results of the deep learning model and answer the questions in terms of Data Preprocessing and Compiling, Training, and Evaluating the Model.

Step 1: Data Preprocessing

Since Alphabet Soup funded more than 34,000 organizations over the years, csv file provided by Alphabet Soup business team includes following columns that capture metadata about each organization.

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

From the above columns, the target variable for the model is "**IS_SUCCESSFUL"** .First of all, to delete the irrelevant dataset, the columns "**'EIN'** and **'NAME'** were dropped from application_df. The remaining columns are features for the model. As **APPLICATION_TYPE** and **CLASSIFICATION** have unique values of 17 and 71 respectively, a cutoff point of each columns were selected to combine "rare" categorical variables together in a new value "Other".

```
APPLICATION_TYPE
T3       27037
T4        1542
T6        1216
T5        1173
T19       1065
T8         737
T7         725
T10        528
Other      276
Name: count, dtype: int64
```

```
CLASSIFICATION
C1000    17326
C2000     6074
C1200     4837
Other     2261
C3000     1918
C2100     1883
Name: count, dtype: int64
```

Step2: Compiling, Training, and Evaluating the Model

Using TensorFlow, I created a binary classification model that can predict if an Alphabet Soup-funded organization will be successful based on the features in the dataset.

Input Layer(dense_18): The input layer has 80 neurons.The activation function used in this layer is ReLU (Rectified Linear Unit).

First Hidden Layer(dense_19): The first hidden layer consists of 80 neurons. It uses the ReLU activation function.

Second Hidden Layer(dense_20): The second hidden layer has 30 neurons. Like the first hidden layer, it also uses the ReLU activation function.

Output Layer: The output layer has a single neuron, which is appropriate for binary classification. The activation function used here is the sigmoid function.

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
input_features = X_train.shape[1]

nn = tf.keras.models.Sequential()

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

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 80)                3520

 dense_19 (Dense)            (None, 30)                2430

 dense_20 (Dense)            (None, 1)                 31

=================================================================
Total params: 5981 (23.36 KB)
Trainable params: 5981 (23.36 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

The model evaluation results indicate the following:
Loss: Approximately 0.558
Accuracy: Approximately 72.6%

```
# 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}")
```

```
8575/8575 - 0s - loss: 0.5578 - acc: 0.7263
Loss: 0.557812534073699, Accuracy: 0.7262973785400391
```

While the accuracy is decent, it falls short of the target performance. Further optimization techniques, such as adjusting hyperparameters, exploring different architectures, or increasing the amount of training data should be considered.