Deep Learning Project: Charity Funding Predictor

Overview:

The purpose of this analysis is to develop a deep learning model for Alphabet Soup, a nonprofit foundation, to aid in the selection of applicants for funding. Alphabet Soup aims to identify organizations with the highest likelihood of success in their ventures. The dataset provided contains information on more than 34,000 organizations that have received funding from Alphabet Soup. The analysis involves the creation of a binary classifier using machine learning and neural networks. The ultimate goal is to predict whether applicants will be successful if funded by Alphabet Soup.

Results:

To begin processing the data, we removed any unnecessary details. EIN and NAME were dropped, leaving the remaining columns as features for our model. Although NAME was added back into the 2nd test for binning purposes. The dataset was then divided into sets for training and testing to evaluate how well the model performs on new, unseen data. The variable we aim to predict is labeled "IS_SUCCESSFUL," taking on values of 1 for yes and 0 for no. We analyzed the APPLICATION data, focusing on the "CLASSIFICATION" value for categorization. Using various data points as cutoffs, we grouped together infrequently occurring variables, assigning the new label "Other" for each unique value. To ensure our model can effectively interpret categorical information, we encoded these variables using the “**get_dummies()**” function after confirming the success of the binning process.

Compiling, Training, and Evaluating the Model:

In total, there were 3 layers for each model after applying Neural Networks. The number of hidden layers were dictated by the number of features.

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

    hidden_layers1 = 7
    hidden_layers2 = 14
    hidden_layers3 = 21


    nn = tf.keras.models.Sequential()

    # First hidden layer
    nn.add(tf.keras.layers.Dense(units=hidden_layers1, input_dim=input_features, activation='relu'))

    # Second hidden layer
    nn.add(tf.keras.layers.Dense(units=hidden_layers2, activation='relu'))

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

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

The training model with three layers produced 477 parameters. In the initial trial, the accuracy was approximately 72-73%, falling below the targeted 75%.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 7)                 350

 dense_7 (Dense)             (None, 14)                112

 dense_8 (Dense)             (None, 1)                 15

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

```
[ ] # 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: 0.5530 - accuracy: 0.7313 - 733ms/epoch - 3ms/step
Loss: 0.5529964566230774, Accuracy: 0.7313119769096375
```

Optimization:

In the second attempt, we added "NAME" back into the dataset. This time, the training model had 3,298 parameters and reached 79%, which is 4% higher than our target goal.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 7)                 3171

 dense_4 (Dense)             (None, 14)                112

 dense_5 (Dense)             (None, 1)                 15

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

Since deep learning models learn how to predict and classify information, the best option is to have multiple layers, so it teaches a computer to filter inputs because it's machine based.

```
# 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 - loss: 0.4529 - accuracy: 0.7910 - 481ms/epoch - 2ms/step
Loss: 0.45291465520858765, Accuracy: 0.791020393371582
```