Deep Learning Project: Charity Funding Predictor

Ronnie Phillips

Overview:

Deep learning and neural networks were used to determine if applicants would be successfully funded by Alphabet Soup, whom previously funded over 34,000 organizations. The purpose was to determine which model could most accurately have the best chance of success.

Data Processing:

Using knowledge of TensorFlow I designed a neural network to create a binary classification model that can predict if an Alphabet Soup-funded organization will be successful based on the features in the dataset. The dataset removed any irrelevant information; therefore, EIN and NAME were dropped from the model. The remaining columns were considered features for the model. In identifying relevant features NAME was added to the second test. CLASSIFICATION and APPLICATION_TYPE was replaced with 'Other' due to high fluctuation. Our identified target was 'IS_SUCCESFUL'

The data was split into training and testing sets of data. The target variable for the model is "IS_SUCCESSFUL" and is verified by the value, 1 was considered yes and 0 was no. During data reduction APPLICATION data was analyzed, and CLASSIFICATION's value was used for binning. Each unique value used several data points as a cutoff point to bin rare categorical variables together in a new value, 'Other'. Afterwards checked to see if binning was successful. Categorical variables were encoded by 'pd.get_dummies ().

Compiling, Training, and Evaluation the Model:

Keras tuner was used to determine the number of neurons, layers and activation functions for the Neural Network. Neural Network was applied on each model multiple layers, three in total. The number of features dictated the number of hidden nodes. A three-layer model generated 477 params. My first attempt was around 73% which is less than the desired 75%

Compile, Train and Evaluate the Model

```
# 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)                 350

 dense_1 (Dense)             (None, 14)                112

 dense_2 (Dense)             (None, 1)                 15

=================================================================
Total params: 477
Trainable params: 477
Non-trainable params: 0
```

```
# 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: 0.5526 - accuracy: 0.7298 - 251ms/epoch - 937us/step
Loss: 0.5526303648948669, Accuracy: 0.7297959327697754
```

Optimization and Summary:

The second attempt added 'NAME' back into the dataset, this time I achieved 79% which was 4% over target for a total of 3,298 params. There is no significant difference in the accuracy of the two models. The one I selected had 477 total params and the suggested model had 3,298 params. It seems you can achieve marginally better results with way less parameters. I would recommend Logistical Regression as its ease of use and wide spread usage.

```
# Check the structure of the model
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: 3,298
Trainable params: 3,298
Non-trainable params: 0
_________________________________________________________________
```