Neural Network Model Analysis
Cory Parker
6/01/24

Purpose

In this project, a tool was created for the nonprofit foundation Alphabet Soup to help select applicants for funding with the best chance of success in their ventures. I used the features in a 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, I 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.

Preprocessing the Data

Using my knowledge of Pandas and scikit-learn's StandardScaler(), I preprocessed the dataset in a Google Colab ipynb file. I created a dataframe while identifying the target variable (IS_SUCCESSFUL) and feature variables (APPLICATION_TYPE, AFFILIATION, CLASSIFICATION, USE_cASE, ORGANIZATION, STATUS, INCOME_AMT, SPECIAL_CONSIDERATIONS, and ASK_AMT) and also dropping irrelevant columns (EIN and NAME) as they are neither targets nor features. I grouped outliers or uncommon ventures. Finally I split the preprocessed data into training and testing datasets. StandardScaler was used to fit the training data.

Compile, Train, and Evaluate the Model

To train the model I created a neural network using our training dataset and 2 hidden layers, the first layer with 20 nodes and the 2nd layer with 3 nodes. I ran 10 epochs on the output layer. This resulted in roughly 71% accuracy.

Optimize the Model

Optimizing the model involved some guess-and-check work tinkering with the layers, nodes per layer, and epochs. Ultimately, by adding a 3rd layer and increasing the nodes of each of the 3 layers to 50, I was able to achieve an accuracy of 73%. I found that increasing the number of epochs started to have a diminishing return on accuracy after about 20 or so.

```
[ ]  # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
     input_features = len(x_train_scaled[0])
     nn = tf.keras.models.Sequential()
     layer1 = 50
     layer2 = 50
     layer3 = 50

     # First hidden layer
     nn.add(tf.keras.layers.Dense(units=layer1, input_dim = input_features, activation = "relu"))

     # Second hidden layer
     nn.add(tf.keras.layers.Dense(units=layer2, activation = "relu"))

     nn.add(tf.keras.layers.Dense(units=layer3, activation = "relu"))

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

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

Summary

After preprocessing, training & evaluation, and optimization, I did not achieve my target accuracy of 75% but came close at 73%. While starting from 71% and making various changes to achieve a more robust model, a lot of insight was gained about the neural network functioning that can be applied to future projects.

```
[ ]  # 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: 0.5547 - accuracy: 0.7294 - 510ms/epoch - 2ms/step
Loss: 0.5546830892562866, Accuracy: 0.7294460535049438
```