**Overview:**

Deep learning and neural networks were used to determine if Alphabet Soup, who previously funded over 34,000 organizations, would successfully fund applicants.

**Results:**

***Data Processing***

Any irrelevant information was removed from the dataset. After dropping EIN and NAME, the remaining columns were considered the features for the model. Although NAME was added back in the second test. CLASSIFICATION and APPLICATION\_TYPE was replaced with ‘Other’ due to high fluctuation. The data was split into training and testing sets of data. The target variable for the model is “IS\_SUCCESSFUL” which is verified by the value, 1 was considered yes and 0 was no. APPLICATION data was analyzed and CLASSIFICATION’s value was used for binning. Each unique value used several data point as a cutoff point to bin “rare” categorical variable together in a new value, ‘Other’. Afterwards, binning was checked for success. In addition, the categorical variables were encoded by pd.get\_dummies ().

***Compiling, Training, and Evaluating the Model***

Neural Network was applied on each model multiple layers, three layers total. The number of features dictated the number of hidden nodes.

*# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.*

*# YOUR CODE GOES HERE*

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*

*# YOUR CODE GOES HERE*

nn.add(tf.keras.layers.Dense(units**=**hidden\_nodes\_layer1, input\_dim**=**number\_input\_features, activation**=**'relu'))

​

*# Second hidden layer*

*# YOUR CODE GOES HERE*

nn.add(tf.keras.layers.Dense(units**=**hidden\_nodes\_layer2, activation**=**'relu'))

​

*# Output layer*

*# YOUR CODE GOES HERE*

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

​

*# Check the structure of the model*

nn.summary()

A three-layer training model generated 477 parameters. The first attempt accuracy was 73%, which was less than the desired 75%.

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 (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: 0.5533 - accuracy: 0.7315 - 1s/epoch - 4ms/step

Loss: 0.5533456802368164, Accuracy: 0.7315452098846436

***Optimization***

The second attempt added ‘NAME’ back into the dataset and achieved 79%, which is 4% over the target accuracy level of 75%. A total of 3,298 params.

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)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*# 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: 0.4677 - accuracy: 0.7903 - 714ms/epoch - 3ms/step

Loss: 0.467724084854126, Accuracy: 0.7903206944465637

In summary, multiple layers should be used for deep learning models since it learns how to predict and classify information based on computer filtering inputs through layers.

Although I was able to attain the desired optimization level of 75%, it would be worthwhile to explore alternative approaches like incorporating the Random Forest Classifier and experimenting with different preprocessing modifications. Also, making changes to the dropout layers, trying out various activation functions, and adjusting the number of layers and neurons could also contribute to optimizing the model and attaining the desired goal of 75% accuracy.