Overview of the analysis:

The non-profit foundation Alphabet Soup wants a tool that can help it select the applicants for funding with the best chance of success in their ventures. With our knowledge of machine learning and neural networks, we will use the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup.

Results:

Data Preprocessing

What variable(s) are the target(s) for your model?

In this case we will use "IS_SUCCESSFUL" variable as our target it is the main indicator to see if the money will be use effectively after applying our model

What variable(s) are the features for your model? The rest of the data, excluding "EIN" and "NAME" will be used as our features:

- APPLICATION_TYPE
- AFFILIATION
- CLASSIFICATION
- USE_CASE
- ORGANIZATION
- STATUS
- INCOME_AMT
- SPECIAL_CONSIDERATIONS
- ASK_AMT

What variable features would be removed from the input data because they are neither targets nor features?

"EIN" and "NAME" will be removed as it is an identification column.

Compiling, Training, and Evaluating the Model

For my second attempt at evaluating the model, I used two hidden layers and one output layer. The first was a Relu activation with 10 units. The second was another Relu activation with 16 units. And the output layer was a 'sigmoid' activation with 1 unit. I was able to achieve the target model performance, achieving a 79% accuracy. This is 7% higher than my first attempt, which achieve a 72% accuracy. To increase the model's performance, I included the 'NAME' column, which was previously dropped from the database, and binned it along with 'APPLICATION_TYPE', and 'CLASSIFICATION'. I played around with the number of hidden layers and units to use. Ultimately, I settled on using three more units on my first hidden layer, and two more units on my second hidden layer than I did previously on my first attempt.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)              Output Shape            Param #
=================================================================
 dense_3 (Dense)           (None, 7)               308

 dense_4 (Dense)           (None, 14)              112

 dense_5 (Dense)           (None, 1)               15

=================================================================
Total params: 435
Trainable params: 435
Non-trainable params: 0
```

```
268/268 - 0s - loss: 0.5557 - accuracy: 0.7256 - 374ms/epoch - 1ms/step

Loss: 0.5557180643081665, Accuracy: 0.7255976796150208
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)              Output Shape            Param #
=================================================================
 dense_32 (Dense)          (None, 10)              4470

 dense_33 (Dense)          (None, 16)              176

 dense_34 (Dense)          (None, 1)               17

=================================================================
Total params: 4,663
Trainable params: 4,663
Non-trainable params: 0
```

```
268/268 - 0s - loss: 0.4474 - accuracy: 0.7930 - 474ms/epoch - 2ms/step

Loss: 0.44742581248283386, Accuracy: 0.7930029034614563
```

Summary:

In conclusion, the second model performed better all-around at a 79% accuracy and 0.45 loss. Whereas the first model had a 72% accuracy with a 0.55 loss. My recommendation would be to use the second model. In the future I would use an automated neural network to determine the best model hyperparameters to use.