Deep Learning Model for Funding Prediction

Overview:

This analysis aims to develop a deep learning model to help the nonprofit foundation, Alphabet Soup, identify applicants most likely to succeed with funding. The dataset includes over 34,000 organizations that have received funding, with features capturing various metadata about each organization.

Results:

Data Preprocessing

- Target variable (y): IS_SUCCESSFUL
- Features variable (X): APPLICATION_TYPE, AFFILIATION, CLASSIFICATION, USE_CASE, ORGANIZATION, STATUS, INCOME_AMT, SPECIAL_CONSIDERATIONS, ASK_AMT.
- Variables removed: EIN, NAME (not relevant for prediction)
- Unique Value Analysis:
  - APPLICATION_TYPE, CLASSIFICATION, and ASK_AMT had more than 10 unique values.
  - APPLICATION_TYPE categories with fewer than 500 occurrences were grouped as "Other."
  - CLASSIFICATION categories with fewer than 1,883 occurrences were grouped as "Other."
  - ASK_AMT remained unchanged.

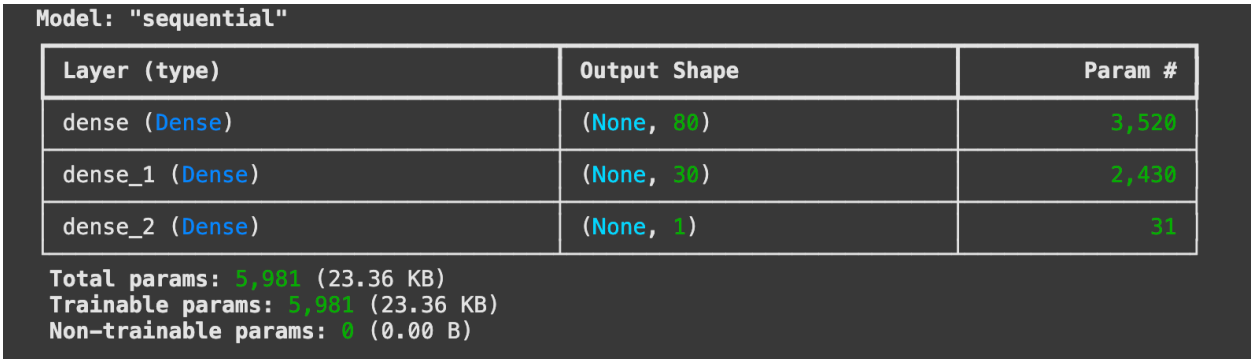
Compiling, Training, and Evaluating the Model:

Model 1: Initial Neural Network

- First Layer: 80 Neurons, Activation: ReLU
- Second Layer: 30 Neurons, Activation: ReLU
- Output Layer: 1 Neuron, Activation: Sigmoid

The chosen architecture was selected for the following reasons:

- ReLU for hidden layers: It offers computational efficiency and helps mitigate the vanishing gradient problem, making training deep neural networks faster and more effective.
- Sigmoid for the output layer: Suitable for binary classification as it outputs probabilities between 0 and 1.
- 80 -> 30 neuron structure: The first layer captures complex patterns, while the second layer refines them, reducing dimensionality before making the final classification.



Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 80)	3,520
dense_1 (Dense)	(None, 30)	2,430
dense_2 (Dense)	(None, 1)	31

Total params: 5,981 (23.36 KB)
Trainable params: 5,981 (23.36 KB)
Non-trainable params: 0 (0.00 B)

Training & Evaluation

- Loss Function: Binary Crossentropy
- Optimizer: Adam
- Metrics: Accuracy
- Epochs: 100

Results:

- Accuracy: 72.61%
- Loss: 55.56%

```
    # 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 - 2ms/step - accuracy: 0.7261 - loss: 0.5556
Loss: 0.555637001991272, Accuracy: 0.726064145565033
```

Optimization:

Attempt 1: Adjusted Cutoff & Architecture

Cutoff Change: Increased APPLICATION_TYPE threshold to 1,000.

New Architecture:

First Layer: 30 Neurons, ReLU
Second Layer: 20 Neurons, ReLU
Third Layer: 10 Neurons, ReLU
Fourth Layer: 5 Neurons, ReLU
Output Layer: 1 Neuron, Sigmoid

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 30)	1,230
dense_1 (Dense)	(None, 20)	620
dense_2 (Dense)	(None, 10)	210
dense_3 (Dense)	(None, 5)	55
dense_4 (Dense)	(None, 1)	6

```
Total params: 2,121 (8.29 KB)
Trainable params: 2,121 (8.29 KB)
Non-trainable params: 0 (0.00 B)
```

Results:

Accuracy: 72.68%
Loss: 56.02%

```
    # 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 - 3ms/step - accuracy: 0.7268 - loss: 0.5602
Loss: 0.5602139234542847, Accuracy: 0.7267638444900513
```

Attempt 2: Different Activation & More Neurons

Cutoff Change: Reduced APPLICATION_TYPE threshold to 700.

New Architecture:

First Layer: 100 Neurons, ReLU
Second Layer: 50 Neurons, LeakyReLU
Third Layer: 25 Neurons, LeakyReLU
Output Layer: 1 Neuron, Sigmoid
Training: Increased to 180 epochs.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 100)	4,300
dense_5 (Dense)	(None, 50)	5,050
leaky_re_lu_2 (LeakyReLU)	(None, 50)	0
dense_6 (Dense)	(None, 25)	1,275
leaky_re_lu_3 (LeakyReLU)	(None, 25)	0
dense_7 (Dense)	(None, 1)	26

Total params: 10,651 (41.61 KB)
Trainable params: 10,651 (41.61 KB)
Non-trainable params: 0 (0.00 B)

Results:

Accuracy: 72.59%
Loss: 57.41%

```
    # 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 - 3ms/step - accuracy: 0.7259 - loss: 0.5741
Loss: 0.574077308177948, Accuracy: 0.7259474992752075
```

Attempt 3: Further Architecture Modification

New Architecture:

First Layer: 126 Neurons, ReLU
Second Layer: 63 Neurons, ReLU
Third Layer: 20 Neurons, ReLU
Output Layer: 1 Neuron, Sigmoid
Training: 150 epochs.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 126)	5,418
dense_5 (Dense)	(None, 63)	8,001
dense_6 (Dense)	(None, 20)	1,280
dense_7 (Dense)	(None, 1)	21

Total params: 14,720 (57.50 KB)
Trainable params: 14,720 (57.50 KB)
Non-trainable params: 0 (0.00 B)

Results:

Accuracy: 72.44%
Loss: 57.82%

```
    # 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 - 2ms/step - accuracy: 0.7244 - loss: 0.5782
Loss: 0.5781964063644409, Accuracy: 0.7244315147399902
```

Summary :

Despite multiple optimization attempts, the deep learning model did not achieve the target accuracy of 75%. The best accuracy recorded was 72.68% (Attempt 1). Increasing neurons, adding layers, and adjusting cutoffs did not significantly enhance performance, suggesting that a deep learning model may not be the best fit for this classification problem.
A traditional machine learning model, such as **Random Forest,** could be more effective for this problem. Random Forest aggregates multiple decision trees, handling non-linear data and outliers effectively, potentially improving predictive performance.

Hence, I would recommend a Random Forest Classifier for the task.