Alphabet Soup Neural Network Model Report

OVERVIEW

With the goal to support the Non-Profit Organization Alphabet Soup in optimizing the distribution of funds, the aim is to build a predictive tool that identifies the applicants who are likely to be successful if funded.

To achieve this goal, I create a binary classification model using deep learning and neural networks to classify metadata for over 34,000 organizations that have received funding from Alphabet Soup over the years. The classification considers the following features:

APPLICATION_TYPE—Alphabet Soup application type
AFFILIATION—Affiliated sector of industry
CLASSIFICATION—Government organization classification
USE_CASE—Use case for funding
ORGANIZATION—Organization type
STATUS—Active status
INCOME_AMT—Income classification
SPECIAL_CONSIDERATIONS—Special considerations for application
ASK_AMT—Funding amount requested
IS_SUCCESSFUL—Was the money used effectively

The analysis involved the following steps:

Step 1: Preprocessing the data to prepare it for modeling
Step 2: Building and evaluating a neural network for binary classification
Step 3: Optimizing the model to "hopefully" achieve an accuracy above 75%

RESULTS

A- Data Processing:

Target variable:

A binary variable indicating if an organization's funding was used effectively (1-Yes, 0= No)

Feature Variable:

All columns except for and EIN, NAME, IS_SUCCESFUL

Variables Removed:

EIN and NAME because they are identification columns
IS_SUCCESFUL is separated as the target

Other Processing Techniques:

Combined rare categories in APPLICATION_TYPE and CLASSIFICATION into"Other" to reduce dimensionality.
Used pd.get_dummies() to one-hot encode categorical features.
Scaled all numeric features using StandardScaler.

B- Compiling, Training, and Evaluating the Model

Initial Model:

Neural Network Architecture:

1- Input: Scaled features (based on dummies + numeric features)
2- Hidden Layer 1: 80 neurons relu activation

3- Hidden Layer2: 30 neurons relu activation
4- Output Layer: 1 neuron, sigmoid activation
Results:
1- Epochs:50
2- Test Accuracy: Approx 72.8 %
3- Model did not meet the 75% accuracy goal

Optimized Model:

Modifications:
1- Increased category binning (cutoff from 500 to 100) to better group rare categories
2 - Expanded model depth and complexity:
a - Hidden layer 1: 10
b - Hidden layer 2: 64
c - Hidden layer 3: 32
3- Increased training epochs from 50 to 75
Results:
1- Test accuracy approx 73.1%
2- The model's accuracy improved by 0.3% but has not met the target performance of 75%

 # SUMMARY

After processing and testing several neural network architectures, I successfully improved the model, yet more improvements are required to reach the desired 75% accuracy threshold.

A- The improvements were achieved through:

Better encoding of categorical variables
Expanding and deepening the neural network.
Training the model for more epochs.

B - Recommendation:

While the neural network achieved the target accuracy, alternative models may perform better since 72 -75% accuracy is not accurate enough to make fair descisions. It might help sift through the data to start, however, greater human involvement, or the use of other methods such as the following, is recomended:

1. ****Random Forest or XGBoost:**** Often more effective for structured tabular data
2. ****Feature Engineering:**** Deeper analysis of feature importance
3. ****AutoML:**** Platforms like H2O.ai for automated model optimization

1- Random Forrest
2- Gradient Boosting (e.g. XGBoost or LightGBM)
3- AutoML platforms (e.g. Google AutoML or H2O.ai
These models are often more interpretable and can in most cases outperform neural networks on structured datasets. For further analysis and feature importance insights, I recommend testing with XGBoost, which also handles missing and categorical data effectively.

Images of First Model

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 80)	2,960
dense_10 (Dense)	(None, 30)	2,430
dense_11 (Dense)	(None, 1)	31

```
Total params: 5,421 (21.18 KB)

Trainable params: 5,421 (21.18 KB)

Non-trainable params: 0 (0.00 B)
```

```
804/804 ━━━━━━━━━━━━━━━━━━━━ 1s 991us/step - accuracy: 0.7400 - loss: 0.5359
Epoch 50/50
804/804 ━━━━━━━━━━━━━━━━━━━━ 1s 866us/step - accuracy: 0.7373 - loss: 0.5403
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```
(variable) X_test_scaled: Any
# 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}")
```

[63] ✓ 0.4s Python

```
268/268 - 0s - 1ms/step - accuracy: 0.7285 - loss: 0.5555
Loss: 0.5555174350738525, Accuracy: 0.7285131216049194
```

```
# Export our model to HDF5 file
```

Images of Optimized Model

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

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWar
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	380
dense_1 (Dense)	(None, 64)	704
dense_2 (Dense)	(None, 32)	2,080
dense_3 (Dense)	(None, 1)	33

Total params: 3,197 (12.49 KB)
Trainable params: 3,197 (12.49 KB)
Non-trainable params: 0 (0.00 B)

```
804/804 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.7362 - loss: 0.5408
Epoch 75/75
804/804 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7460 - loss: 0.5326
```

```python
# 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: 0.7305 - loss: 0.5555
Loss: 0.5555225610733032, Accuracy: 0.7304956316947937
```

```python
# Export our model to HDF5 file
nn.save("AlphabetSoupCharityOptimization.h5")
```