Alphabet Soup Deep Learning Model Report

Overview of the Analysis

The purpose of this analysis is to create and optimize a deep learning model that predicts the success of charity funding applications. The goal is to help the organization, Alphabet Soup, make better funding decisions based on historical data. This analysis aims to build a model that can classify if a funding application will be successful, using various characteristics of each application as features.

Results

Data Preprocessing

- **Target Variable**: The target variable for the model is **IS_SUCCESSFUL**, which indicates whether the funding application was successful or not.
- **Features**: The features used for the model include **APPLICATION_TYPE**, **AFFILIATION**, **CLASSIFICATION**, **USE_CASE**, **ORGANIZATION**, **INCOME_AMT**, **ASK_AMT**, among others.
- **Variables Removed**: The columns **EIN**, **NAME**, and **STATUS** were removed from the input data as they are identifiers or irrelevant to predicting the success of the funding application.

Compiling, Training, and Evaluating the Model

- **Neurons, Layers, and Activation Functions**:
    - The original model (starting point) contains **two hidden layers**:
        - **First Hidden Layer**: 80 neurons with **relu** activation function.
        - **Second Hidden Layer**: 30 neurons with **relu** activation function.
    - The **output layer** contains **1 neuron** with a **sigmoid** activation function to predict the binary outcome.
    - The optimizer used was **adam**, with a **binary crossentropy** loss function.
- **Optimized Models**:
    - **Attempt 1**: Added an additional hidden layer and increased the number of neurons:
        - **First Hidden Layer**: 120 neurons with **relu** activation function.
        - **Second Hidden Layer**: 60 neurons with **relu** activation function.
        - **Third Hidden Layer** (newly added): 40 neurons with **relu** activation function.
        - **Output Layer**: 1 neuron with **sigmoid** activation function.

        - **Epochs** increased to **150**.
    - **Attempt 2**: Modified activation functions and adjusted the training parameters:
        - Increased the number of epochs and added a **dropout layer** to prevent overfitting.
        - **Dropout Layer** with a rate of **0.3** was added after the second hidden layer.
    - **Attempt 3**: Further increased the number of neurons in each layer and changed the **batch size** to **32** for better gradient updates.
- **Target Model Performance**:
    - The goal was to achieve an accuracy of **75%** or higher. The final accuracy achieved was **72.86%**, which is slightly below the target.
- **Steps to Increase Model Performance**:
    - Multiple attempts were made to increase model performance by:
        - **Adding more hidden layers** to increase the complexity of the model.
        - **Increasing the number of neurons** in the hidden layers to capture more complex relationships.
        - **Adjusting the number of epochs** to allow the model more time to learn.
        - **Adding dropout layers** to prevent overfitting.

Summary

The deep learning model built for Alphabet Soup initially achieved an accuracy of **72.86%**, which was close to but not quite reaching the target of **75%**. Despite several optimization attempts, the model did not reach the target accuracy of **75%**. The optimizations included adding hidden layers, increasing neurons, changing activation functions, and adjusting the training parameters such as epochs and batch size.

For future improvements, it may be beneficial to explore different model architectures. One recommendation is to try using a **Random Forest Classifier** or **Gradient Boosting Classifier**, which are often well-suited for tabular data and may provide better performance for this classification problem. Additionally, implementing **hyperparameter tuning** through tools like **GridSearchCV** could help find the optimal settings for the deep learning model or other machine learning models.

Images

Below are images showcasing the evaluation results of the model during different attempts:

- **Original Attempt**: Accuracy = 72.76%

```
[ ]  # Evaluate the model using the test data
     model_loss, model_accuracy = nn.evaluate(X_test_scaled, y_test, verb
     print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
215/215 - 0s - 2ms/step - accuracy: 0.7276 - loss: 0.5931
Loss: 0.59310382604599, Accuracy: 0.7275510430335999
```

- **Attempt 1**: Accuracy = 72.74%

```
# 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}")
```

```
215/215 - 0s - 2ms/step - accuracy: 0.7274 - loss: 0.5907
Loss: 0.5906961560249329, Accuracy: 0.7274052500724792
```

- **Attempt 2**: Accuracy = 72.86%

```
# 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}")
```

```
215/215 - 0s - 1ms/step - accuracy: 0.7286 - loss: 0.6339
Loss: 0.6339481472969055, Accuracy: 0.7285714149475098
```