Deep Learning Challenge (Module 21) charity funding predictor analysis report

Overview:

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 the knowledge of machine learning and neural networks, I have 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.

From Alphabet Soup's business team, I have received a CSV containing more than 34,000 organisations that have received funding from Alphabet Soup over the years. Within this dataset are several columns that capture metadata about each organisation, such as:

- **EIN** and **NAME**—Identification columns
- **APPLICATION_TYPE**—Alphabet Soup application type
- **AFFILIATION**—Affiliated sector of industry
- **CLASSIFICATION**—Government organisation classification
- **USE_CASE**—Use case for funding
- **ORGANIZATION**—Organisation 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

Results:

The first step involved dropping irrelevant columns, specifically the EIN and NAME columns. The remaining columns were considered as features for the model. In the second test, the NAME column was added back for binning purposes.

The data was then split into training and testing sets, with the target variable labeled as "IS_SUCCESSFUL" and having a binary value of 1 for "yes" and 0 for "no".

The "CLASSIFICATION" value was used for binning, and several data points were used as a cutoff to bin "rare" variables together with the new value of "Other" for each unique value.

Categorical variables were encoded using get_dummies() after checking to see if the binning was successful.
It's important to note that binning and encoding are common techniques used in data preprocessing to prepare data for machine learning models. Binning helps to group similar data points together, which can help improve model accuracy. Encoding converts categorical variables into numerical values that can be used in the model. Checking for the success of these preprocessing steps is also important to ensure the data is properly prepared for the model.

Compiling, Training, and Evaluating the Model:
While describing the results of training a neural network model, the model had three layers in total, and the number of hidden nodes in each layer was determined by the number of features in the input data. This is a common approach to designing neural network architectures.

After training the model, it was found that the model had a total of 477 parameters. These parameters are learned during the training process and are used to make predictions on new data.

The first attempt at training the model resulted in an accuracy of just under 73%. This accuracy fell short of the desired 75%. It's important to note that model accuracy can be influenced by many factors, including the size and quality of the training data, the complexity of the model architecture, and the choice of hyperparameters. It may be necessary to experiment with different model architectures and hyperparameters to achieve higher accuracy.

In [14]:

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

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation='relu'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation='relu'))


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

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

```
Model: "sequential"
_________________________________________________________________
 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
Trainable params: 477
Non-trainable params: 0
_________________________________________________________________
```

In [17]:

```
# 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: 0.5527 - accuracy: 0.7287 - 554ms/epoch - 2ms/step
Loss: 0.552720308303833, Accuracy: 0.7287463545799255
```

Optimization:

It's great to found that the second attempt at training the model with the "NAME" column included in the dataset was successful, achieving an accuracy of almost 79%. This is a significant improvement over the first attempt and meets the target accuracy of 75%. The model had 3,298 parameters, which is larger than the previous model with 477 parameters. This increase in the number of parameters may have allowed the model to better capture the patterns in the data and improve its accuracy.

It's worth noting that deep learning models typically involve multiple layers, as mentioned. These layers allow the model to learn complex representations of the input data, which can improve its performance. However, it's important to strike a balance between model complexity and performance. Too many layers or too many parameters can lead to overfitting, where the model becomes too specialized to the training data and performs poorly on new data. Regularization techniques can be used to help prevent overfitting and improve the generalization performance of the model.

In [47]:
```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
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
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation='relu'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation='relu'))

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

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

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 7)                 3171

 dense_4 (Dense)             (None, 14)                112

 dense_5 (Dense)             (None, 1)                 15

=================================================================
Total params: 3,298
Trainable params: 3,298
Non-trainable params: 0
_________________________________________________________________
```

In [51]:
```
# 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 - 0s - loss: 0.4791 - accuracy: 0.7888 - 432ms/epoch - 2ms/step
Loss: 0.4791410267353058, Accuracy: 0.7888046503067017
```