Deep Learning Challenge Analysis Report

1. Overview (background information)

Funding organization selects applicants based on the chance of success of the proposed project. One of such funding organizations, a non-profit foundation Alphabet Soup, wanted to create a model or algorithm that can help screen applicants that will be successful based on certain criteria that can help predict success. So, the objective of this challenge was to use Alphabet Soup's data that has more than 34,000 organizations that receive funding from Alphabet Soup and perform deep learning analysis and see if a model can predict whether an applicant's project will be successful or not. This challenge is done to help us to apply our knowledge of machine learning and neural networks that we have learned in class.

2. Data processing:

To perform this, first I started by creating a repository in GitHub, cloning the repo, adding file to it and pushing to the GitHub. Then I uploaded the notebook in google Colab, read the "charity_data.csv" into Pandas DataFrame and identified the target variable and features. The data was processed by dropping EIN and NAME the remaining columns were to be considered features for the model. Also, more processing was done by getting unique value counts and binning. All categorical data was converted into numeric values using "pd.get_dummies" function.

```
# Import our dependencies
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import tensorflow as tf

#  Import and read the charity_data.csv.
import pandas as pd
application_df = pd.read_csv("https://static.bc-edx.com/data/dl-1-2/m21/lms/starter/charity_data.csv")
application_df.head()
```

	EIN	NAME	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	STATUS	INCOME_AMT	SPECIAL_CONSIDERATIONS	ASK_AMT	IS_SUCCESSFUL
0	10520599	BLUE KNIGHTS MOTORCYCLE CLUB	T10	Independent	C1000	ProductDev	Association	1	0	N	5000	1
1	10531628	AMERICAN CHESAPEAKE CLUB CHARITABLE TR	T3	Independent	C2000	Preservation	Co-operative	1	1-9999	N	108590	1
2	10547893	ST CLOUD PROFESSIONAL FIREFIGHTERS	T5	CompanySponsored	C3000	ProductDev	Association	1	0	N	5000	0
3	10553066	SOUTHSIDE ATHLETIC ASSOCIATION	T3	CompanySponsored	C2000	Preservation	Trust	1	10000-24999	N	6692	1
4	10556103	GENETIC RESEARCH INSTITUTE OF THE DESERT	T3	Independent	C1000	Heathcare	Trust	1	100000-499999	N	142590	1

```
[2] # Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
    application_df = application_df.drop(columns=['EIN', 'NAME'])
```

```
[ ] # Print the DataFrame
    application_df.head()
```

	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	STATUS	INCOME_AMT	SPECIAL_CONSIDERATIONS	ASK_AMT	IS_SUCCESSFUL
0	T10	Independent	C1000	ProductDev	Association	1	0	N	5000	1
1	T3	Independent	C2000	Preservation	Co-operative	1	1-9999	N	108590	1
2	T5	CompanySponsored	C3000	ProductDev	Association	1	0	N	5000	0
3	T3	CompanySponsored	C2000	Preservation	Trust	1	10000-24999	N	6692	1
4	T3	Independent	C1000	Heathcare	Trust	1	100000-499999	N	142590	1

```
[3] # Determine the number of unique values in each column.
    for column in application_df.columns:
        unique_values = application_df[column].nunique()
        print(f"{column}: {unique_values}")
```

```
APPLICATION_TYPE: 17
AFFILIATION: 6
CLASSIFICATION: 71
USE_CASE: 5
ORGANIZATION: 4
```

The data was splited ito features and target variables. The target variable for the model was labeled "IS_SUCCESSFUL" and has the value of 1 for yes and 0 for no. The features(x) were all other columns (inputs) after dropping the target variable (see screenshot below).

```
# Split our preprocessed data into our features and target arrays
y = application_df['IS_SUCCESSFUL'].values
y

# drop 'IS SUCCESSFUL'
X = application_df.drop('IS_SUCCESSFUL', axis=1).values
X
```

```
array([[       1,     5000,        0, ...,        0,        1,        0],
       [       1,   108590,        0, ...,        0,        1,        0],
       [       1,     5000,        0, ...,        0,        1,        0],
```

In this data processing step the following questions were answered:

- What variable(s) are the target(s) for your model? The target variable was if the project funded by the charity organization was successful or not, which the values 1 for yes and 0 for no.
- What variable(s) are the features for your model? All the variables after dropping the target variable, and the EIN and NAME columns (which were not important as target or input variables), the rest were used as features.
- What variable(s) should be removed from the input data because they are neither targets nor features? The EIN and NAME variables were removed.

Once the above process is completed, then, the data was split into training and testing data sets.

3. Compiling, Training, and Evaluating the Model:

This step was completed as instructed in the module by doing all the steps that are needed to compile and train the model and evaluate the model using the test data. See optimization steps below.

4. Optimize the Model:

After completing all the codes in the base file, 3 different optimizations were done by varying different parameters to see which one will give us a better accuracy of the model. For details, please refer to the output screenshots below and the optimization file attached in GiHub.

i) **Optimization attempt #1**: for the 1st attempt, the accuracy score was 0.7267(72.67%).

Optimization attempt No.1

- Dropping more or fewer columns.(dropping an additional column: use_case)
- Creating more bins for rare occurrences in columns.(changing thresholds for binning application_typ and classification columns)

Compile, Train and Evaluate the Model

In [13]:

```
# 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=80
hidden_nodes_layer2=30

nn1 = tf.keras.models.Sequential()

# First hidden layer
nn1.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation='relu'))

# Second hidden layer
nn1.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation='relu'))
# Third hidden layer
#nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation='sigmoid'))

# Output layer
nn1.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

# Check the structure of the model
nn1.summary()

nn1.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics=['accuracy'])

# Train the model
fit_model = nn1.fit(X_train_scaled,y_train,validation_split=0.15, epochs=100)

# Evaluate the model using the test data
model_loss, model_accuracy = nn1.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

ii) **Optimization attempt #2:** for parameters adjusted here from step one, see screenshot below. The accuracy (72.75%) was a little bit better than the first attempt.

Optimization no.2

- Add more neurons to a hidden layer.(adding additional nuerons to the 1st and 2nd layers)
- Add more hidden layers.(adding a 3rd hiiden layer)

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=100
hidden_nodes_layer2=60
hidden_nodes_layer3=30

nn2 = tf.keras.models.Sequential()

# First hidden layer
nn2.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation='relu'))

# Second hidden layer
nn2.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation='relu'))
# Third hidden layer
nn2.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation='sigmoid'))

# Output layer
nn2.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

# Check the structure of the model
nn2.summary()

nn2.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics=['accuracy'])

# Train the model
fit_model = nn2.fit(X_train_scaled,y_train,validation_split=0.15, epochs=100)

# Evaluate the model using the test data
model_loss, model_accuracy = nn2.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

iii) **Optimization attempt #3:** for parameters adjusted here from step one and two for the 3rd attempt, see screenshot below. The accuracy (72.52%) was somehow lower than both the 1st and 2nd attempts.

Optimization no.3

- Use different activation functions for the hidden layers.(using sigmoid as the activation function for all layers)
- Add or reduce the number of epochs to the training regimen.(increasing epox to 200)

[15]:

```
# 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=100
hidden_nodes_layer2=60
hidden_nodes_layer3=30

nn3 = tf.keras.models.Sequential()

# First hidden layer
nn3.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation='sigmoid'))

# Second hidden layer
nn3.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation='sigmoid'))
# Third hidden layer
nn3.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation='sigmoid'))

# Output layer
nn3.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

# Check the structure of the model
nn3.summary()

nn3.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics=['accuracy'])

# Train the model
fit_model = nn3.fit(X_train_scaled,y_train,validation_split=0.15, epochs=200)

# Evaluate the model using the test data
model_loss, model_accuracy = nn3.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

At this step, the following critical questions were answered based on attempt 2 which gave better accuracy:

- How many neurons, layers, and activation functions did you select for your neural network model, and why? I have used a total of 190 neurons, 3 layers, and 1 activation function (sigmoid) that was used for all the layers.
- Were you able to achieve the target model's performance? I am very close, 73% compared to the 75% mark.
- What steps did you take in your attempts to increase model performance? I have done 3 attempts as indicated above varying different parameters including manipulating the number of neurons, layers, as well as activation codes. But the performance was better at the parameters I have on step 2.

5. Summary of the Analysis:

Based on the optimization attempts above, optimization step 2 gives better accuracy of the model. The loss value of 0.5607" indicates the value of the loss function at the end of the training. The loss function is a measure of how well the model can predict the correct output for the given input. We would prefer to have lower values of loss which indicate better performance a model in predicting output. On the other hand, the accuracy value of 0.7275" indicates the accuracy of the model on the training dataset. It is a measure of how well the model can correctly classify the input. Higher values of accuracy indicate better performance. The accuracy is in this analysis 73%, which is close to the 75% mark.

Recommendation: To be applicable for practical use, the accuracy of the model should be higher. This lower accuracy could arise from different factors including representativeness of the data and underfitting of the model to the data. So, better to consider resampling of the data, consider using different number of layers, and maybe adjust the splitting of the data set into training and testing.