Deep Learning Challenge Analysis Report

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.

1. 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 9inputs) 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],
```

Then the data was split into training and testing data sets.

2. Compiling, Training, and Evaluating the Model:

For this purpose, three hidden layers were used with "relu" and and "sigmoid" models as activations. The number of hidden nodes were dictated by the number of features.

```
[ ] # 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=8
    hidden_nodes_layer2=16
    hidden_nodes_layer3=24

    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_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 8)                 904

 dense_13 (Dense)            (None, 16)                144

 dense_14 (Dense)            (None, 1)                 17

=================================================================
Total params: 1,065
Trainable params: 1,065
Non-trainable params: 0
```

Compiling and training models screenshot below.

```
[ ] # Compile the model
    nn.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics=['accuracy'])
```

```
[ ] # Train the model
    fit_model = nn.fit(X_train_scaled,y_train,validation_split=0.15, epochs=100)
```

```
Epoch 1/100
684/684 [==============================] - 2s 2ms/step - loss: 0.6000 - accuracy: 0.6941 - val_loss: 0.5800 - val_accuracy: 0.7134
Epoch 2/100
684/684 [==============================] - 2s 2ms/step - loss: 0.5587 - accuracy: 0.7305 - val_loss: 0.5679 - val_accuracy: 0.7170
Epoch 3/100
684/684 [==============================] - 1s 2ms/step - loss: 0.5509 - accuracy: 0.7326 - val_loss: 0.5641 - val_accuracy: 0.7204
Epoch 4/100
684/684 [==============================] - 2s 2ms/step - loss: 0.5480 - accuracy: 0.7339 - val_loss: 0.5636 - val_accuracy: 0.7204
Epoch 5/100
684/684 [==============================] - 2s 3ms/step - loss: 0.5460 - accuracy: 0.7349 - val_loss: 0.5634 - val_accuracy: 0.7217
```

3. Summary of the Analysis:

Finally, the mode's accuracy to predict the success of a grant project was evaluated using the testing data. As we can see from the screenshot of the output of the testing model, the model has model has completed training on 268 batches of data. The loss value of 0.5516" 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 0.7262" 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.

```
[60] # 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: 0.5516 - accuracy: 0.7262 - 366ms/epoch - 1ms/step
     Loss: 0.5516282916069031, Accuracy: 0.7261807322502136
```