Deep Learning Challenge

Overview

In this Challenge we have created a model that can help nonprofit foundation Alphabet Soup select the applicants for funding with the best chance of success in their ventures. With our knowledge of machine learning and neural networks, we have used the features in the provided dataset to create a binary classifier that can predict whether applicants will be successful if funded by Alphabet Soup.

CSV containing more than 34,000 organisations that have received funding over the years from Alphabet Soup is provided to us in CSV format. Within this dataset are several columns that capture metadata about each organisation as below:

- **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

1) Data Preprocessing

Initial Model

- ID columns **`EIN`** and **`NAME`** are dropped as in instructions for the initial model.
- **IS_SUCCESSFUL** is identified as the **Target** of our model and the **rest of the variables as the features**.
- Binning of **APPLICATION_TYPE** and **CLASSIFICATION** columns is performed as there are more than 10 unique values in each. For doing

so, cutoff points are identified and "rare" categorical variables are binned together in a new value, **Other**.

```
# Checking to make sure binning was successful
application_df['APPLICATION_TYPE'].value_counts()
```

```
Out[6]: T3        27037
        T4         1542
        T6         1216
        T5         1173
        T19        1065
        T8          737
        T7          725
        T10         528
        Other       276
        Name: APPLICATION_TYPE, dtype: int64
```

```
# Checking to make sure binning was successful
application_df['CLASSIFICATION'].value_counts()
```

```
Out[10]: C1000    17326
         C2000     6074
         C1200     4837
         Other     2261
         C3000     1918
         C2100     1883
         Name: CLASSIFICATION, dtype: int64
```

- Categorical data are converted to numeric with **pd.get_dummies**.
- Pre-processed data is split into target **X** and features **y** arrays.
- **train_test_split** from **sklearn.model_selection** was used to split the data into training and testing datasets **X_train, X_test, y_train, y_test**.
- Training and testing features datasets are scaled by creating a **StandardScaler** instance, fitting it to the training data, then using the **transform** function.

2) Compiling, Training, and Evaluating the Model

- A neural network model is created by assigning the number of input features (**44**) and nodes for each layer using TensorFlow and Keras.

- The first hidden layer has an activation function **relu** and **80** neurons. **Relu is used because introducing a nonlinear activation function will help train the model better.**
- A second hidden layer with an activation function **relu** is created with **30** neurons.
- Output layer is added with **1** neuron and activation function **sigmoid since we were creating a binary classification model**

Ref: [How to Choose an Activation Function for Deep Learning - MachineLearningMastery.com]()

```
In [20]: # Defining the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len(X_train[0])
hidden_nodes_layer1 =  80
hidden_nodes_layer2 = 30

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"))
```

- Structure of the model created is as below:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 80)                3520

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 5981 (23.36 KB)
Trainable params: 5981 (23.36 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- The model is compiled.
- A callback that saves the model's weights every five epochs is created.

```python
# Train the model saving model weights every 5 epochs using callback
mc = tf.keras.callbacks.ModelCheckpoint('Weights/weights{epoch:04d}.h5',
                                        save_weights_only=True, period=5)
fit_model = nn.fit(X_train_scaled,y_train,epochs=100, callbacks=[mc])
```

```
804/804 [==============================] - 3s 4ms/step - loss: 0.5347 - accuracy: 0.7412
Epoch 92/100
804/804 [==============================] - 3s 3ms/step - loss: 0.5346 - accuracy: 0.7413
Epoch 93/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5350 - accuracy: 0.7416
Epoch 94/100
804/804 [==============================] - 3s 4ms/step - loss: 0.5349 - accuracy: 0.7409
Epoch 95/100
804/804 [==============================] - 2s 3ms/step - loss: 0.5348 - accuracy: 0.7417
Epoch 96/100
804/804 [==============================] - 3s 4ms/step - loss: 0.5350 - accuracy: 0.7412
Epoch 97/100
804/804 [==============================] - 3s 4ms/step - loss: 0.5348 - accuracy: 0.7408
Epoch 98/100
804/804 [==============================] - 3s 3ms/step - loss: 0.5345 - accuracy: 0.7414
Epoch 99/100
804/804 [==============================] - 3s 3ms/step - loss: 0.5344 - accuracy: 0.7414
Epoch 100/100
804/804 [==============================] - 3s 3ms/step - loss: 0.5344 - accuracy: 0.7416
```

- The model is evaluated using the test data to determine the loss and accuracy.

```python
# Evaluating 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: 0.5611 - accuracy: 0.7250 - 1s/epoch - 4ms/step
Loss: 0.5611390471458435, Accuracy: 0.7250145673751831
```

- Weights are saved every 5 epochs in **Weights** folder as file name format **weights{epoch:04d}.h5**

- Model evaluation:

  This model with a total of 3 layers including input, output and a hidden layer with **44** input features and activation methods **relu, relu and sigmoid** had an accuracy of **72.5%** and loss of **56%.**

- The model is saved and export to an HDF5 file: **AlphabetSoupCharity.h5** in **Models** folder

Steps taken in an attempt to increase model performance

- A new Jupyter Notebook file **AlphabetSoupCharity_Optimisation.ipynb** is created and Data is **pre-processed** same as for the **Initial Model**
- Based on results from the initial model, four attempts are made on designing a neural network model with target predictive accuracy higher than **75%** as per details in below table:

Image 1

No.	Method	Number of Layers	Neurons per layer	Activation functions per layer	Epochs	Loss	Accuracy
1	Increasing number of layers and neurons	Input/Hidden1/Hidden2/ Output	80/30/10/1; input_dim = 44	relu/relu/relu/sigmoid	100	55.7%	72.3%
2	Changing activation method and neurons	Input/Hidden1/Hidden2/ Output	80/30/15/1; input_dim = 44	relu/relu/tanh/sigmoid	100	55.7%	72.3%
3	Dropping STATUS and Special Considerations related columns	Input/Hidden1/ Output	80/30/1; input_dim = 41	relu/relu/ sigmoid	100	56.6%	72.4%
3b	Increasing number of epochs to 200 for the previous method	Input/Hidden1/ Output	80/30/1; input_dim = 41	relu/relu/ sigmoid	200	59.8%	72.3%
4	Checking Name/EID column values for duplicates (reappearance over time). Dropping only EIN column while retaining Name	Input/Hidden1/ Output	20/10/1; input_dim = 447	relu/relu/ sigmoid	20	44.3%	79.4%

Model optimisation method 1

Increasing number of layers and neurons

Without changing the inputs from the previous model, number of neurons and number of layers are increased to check if this increases accuracy of the model.

No.	Method	Layers	Neurons per layer	Activation functions per layer	Epochs	Loss	Accuracy
1	**Increasing number of layers and neurons**	Input/Hidden1/Hidden2/Output	80/30/10/1; input_dim = 44	relu/relu/relu/sigmoid	100	55.7%	72.3%

Model Structure:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 80)                3520

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 10)                310

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 6271 (24.50 KB)
Trainable params: 6271 (24.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model evaluation:

```
268/268 - 1s - loss: 0.5569 - accuracy: 0.7234 - 814ms/epoch - 3ms/step
Loss: 0.556873083114624, Accuracy: 0.7233819365501404
```

Result

No change from initial model is observed.

Note: Tried many permutations and combinations by increasing neurons, increasing number of layers, increasing number of layers and neurons and kept one of the iterations in the notebook. **No observation had better accuracy.**

Model optimisation method 2

Changing activation method and neurons

Without changing the inputs from the previous model, **tanh** is used **instead of relu** in one of the layers next to the output layer. Updated to tanh in the layer closer to the output layer as they are both similar in slope. Also added 5 extra neurons to check if there was any improvement in accuracy which these steps. Note: Many permutations and combinations of activation functions (tanh, relu and softmax on various layers in various combinations) and varying neurons/layers were tried and only one iteration kept in the notebook which performed better than the rest.

No.	Method	Layers	Neurons per layer	Activation functions per layer	Epochs	Loss	Accuracy
2	**Changing activation method and neurons**	Input/Hidden1/ Hidden2/Output	80/30/15/ 1; input_dim = 44	relu/relu/tanh /sigmoid	100	55.7%	72.3%

Model Structure:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 80)                3520

 dense_5 (Dense)             (None, 30)                2430

 dense_6 (Dense)             (None, 15)                465

 dense_7 (Dense)             (None, 1)                 16

=================================================================
Total params: 6431 (25.12 KB)
Trainable params: 6431 (25.12 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model evaluation:

```
268/268 - 1s - loss: 0.5569 - accuracy: 0.7234 - 562ms/epoch - 2ms/step
Loss: 0.556873083114624, Accuracy: 0.7233819365501404
```

Result

No change from previous model is observed at **72.3%** accuracy by changing the activation methods. In fact, all activation function permutations produced similar results.

Model optimisation method 3

Dropping status and special considerations related columns.

Since on analysing the unique values of each column it was found that **status** and **special considerations** had two unique values each with very few records on one of the two, it should not have too much contribution to the model. I planned to check this by dropping the two columns.

There were no changes to the inputs from the initial model for number of layers, neurons and activation methods.

No.	Method	Layers	Neurons per layer	Activation functions per layer	Epochs	Loss	Accuracy
3	**Dropping STATUS and Special Considerations related columns**	Input/Hidden1/ Output	80/30/1; input_dim = 41	relu/relu/ sigmoid	100	56.6%	72.4%

Model structure:

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 80)                3280

 dense_9 (Dense)             (None, 30)                2430

 dense_10 (Dense)            (None, 1)                 31

=================================================================
Total params: 5741 (22.43 KB)
Trainable params: 5741 (22.43 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model evaluation:

```
268/268 - 1s - loss: 0.5661 - accuracy: 0.7245 - 857ms/epoch - 3ms/step
Loss: 0.566061794757843, Accuracy: 0.7245481014251709
```

Result

As expected, no change from previous model is observed **at 72.4%** accuracy.

Method 3b - Increasing number of epochs to 200

Increased number of epochs to 200 in this trial to check if allowing more time to train had any impact.

No.	Method	Layers	Neurons per layer	Activation functions per layer	Epochs	Loss	Accuracy
3b	**Increasing number of epochs to 200 for the previous method**	Input/Hidden1/ Output	80/30/1; input_dim = 41	relu/relu/ sigmoid	200	59.8%	72.3%

Model training:

```
In [40]: # Increasing number of epochs
         fit_model = nn.fit(X_train_scaled,y_train,epochs=200)
804/804 [==============================] - 2s 3ms/step - loss: 0.5301 - accuracy: 0.7430
Epoch 192/200
804/804 [==============================] - 2s 3ms/step - loss: 0.5301 - accuracy: 0.7429
Epoch 193/200
804/804 [==============================] - 2s 3ms/step - loss: 0.5303 - accuracy: 0.7434
Epoch 194/200
804/804 [==============================] - 2s 3ms/step - loss: 0.5302 - accuracy: 0.7434
Epoch 195/200
804/804 [==============================] - 2s 3ms/step - loss: 0.5298 - accuracy: 0.7435
Epoch 196/200
804/804 [==============================] - 2s 3ms/step - loss: 0.5314 - accuracy: 0.7434
Epoch 197/200
804/804 [==============================] - 2s 3ms/step - loss: 0.5297 - accuracy: 0.7432
Epoch 198/200
804/804 [==============================] - 2s 3ms/step - loss: 0.5299 - accuracy: 0.7432
Epoch 199/200
804/804 [==============================] - 2s 3ms/step - loss: 0.5297 - accuracy: 0.7429
Epoch 200/200
804/804 [==============================] - 2s 3ms/step - loss: 0.5305 - accuracy: 0.7426
```

Model evaluation:

```
268/268 - 1s - loss: 0.5981 - accuracy: 0.7234 - 556ms/epoch - 2ms/step
Loss: 0.598114550113678, Accuracy: 0.7233819365501404
```

Result

We achieved **72.3%** accuracy which is again like the last models. **Hence increasing epochs also did not have any impact on the accuracy of the model.** Also, it was noted that loss was around **53%** in all above trials.

Model optimisation method 4

Checking Name/EID column values for duplicates (reappearance) and if one of them needs to be included for creating our model.

We could see that names of organisation are not unique like EIN which is the sole unique identifier for this data. This could indicate that **applications** from certain organizations might have been preferred due to factors like infrastructure, technical know-how and support etc by the organization. Therefore, the name of the organization might actually be equivalent to a whole lot of features which might be contributing to success.

With this thought process, **we dropped only EIN column** and retained **NAME** in this attempt to generate a model to see if it has impact on accuracy of model when tested.

```
PARENT BOOSTER USA INC                                          1260
TOPS CLUB INC                                                    765
UNITED STATES BOWLING CONGRESS INC                               700
WASHINGTON STATE UNIVERSITY                                      492
AMATEUR ATHLETIC UNION OF THE UNITED STATES INC                  408
                                                                ...
VETERANS OF FOREIGH WARS OF THE UNITED STATES DEPT OF COLORADO     2
ETA PHI BETA SORORITY INC                                          2
POINT MAN INTERNATIONAL MINISTRIES                                 2
MULTI COMMUNITY DIVERSIFIED SERVICES INC                           2
VETERANS OF FOREIGN WARS AUXILIARY                                 2
Name: NAME, Length: 792, dtype: int64
```

- Binning was performed on **NAME** to reduce number of features. For doing so, cutoff points were identified, and "rare" categorical variables i.e. Count of NAME < 5 were binned together in a new value, **Other**
- No changes to the inputs from the initial model for number of layers, neurons and activation methods.

No.	Method	Number of Layers	Neurons per layer	Activation functions per layer	Epochs	Loss	Accuracy
4	**Checking Name/EID column values for duplicates (reappearance). Dropping only EIN column while retaining Name**	Input/Hidden1/ Output	20/10/1; input_dim = 447	relu/relu/ sigmoid	20	44.3%	79.4%

Model structure:

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_11 (Dense)            (None, 20)                8940

 dense_12 (Dense)            (None, 10)                210

 dense_13 (Dense)            (None, 1)                 11

=================================================================
Total params: 9161 (35.79 KB)
Trainable params: 9161 (35.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model training:

```
fit_model = nn.fit(X_train_scaled,y_train,epochs=20)
```

```
Epoch 1/20
804/804 [==============================] - 4s 3ms/step - loss: 0.5019 - accuracy: 0.7633
Epoch 2/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4317 - accuracy: 0.7958
Epoch 3/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4255 - accuracy: 0.7966
Epoch 4/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4235 - accuracy: 0.7982
Epoch 5/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4215 - accuracy: 0.7989
Epoch 6/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4196 - accuracy: 0.8008
Epoch 7/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4191 - accuracy: 0.8001
Epoch 8/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4183 - accuracy: 0.8003
Epoch 9/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4174 - accuracy: 0.8016
Epoch 10/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4172 - accuracy: 0.8019
Epoch 11/20
804/804 [==============================] - 3s 3ms/step - loss: 0.4167 - accuracy: 0.8005
Epoch 12/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4162 - accuracy: 0.8022
Epoch 13/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4161 - accuracy: 0.8029
Epoch 14/20
804/804 [==============================] - 3s 3ms/step - loss: 0.4155 - accuracy: 0.8025
Epoch 15/20
804/804 [==============================] - 4s 5ms/step - loss: 0.4153 - accuracy: 0.8018
Epoch 16/20
804/804 [==============================] - 3s 4ms/step - loss: 0.4148 - accuracy: 0.8017
Epoch 17/20
804/804 [==============================] - 3s 4ms/step - loss: 0.4145 - accuracy: 0.8026
Epoch 18/20
804/804 [==============================] - 3s 4ms/step - loss: 0.4145 - accuracy: 0.8029
Epoch 19/20
804/804 [==============================] - 2s 3ms/step - loss: 0.4138 - accuracy: 0.8043
Epoch 20/20
804/804 [==============================] - 3s 3ms/step - loss: 0.4137 - accuracy: 0.8034
```

Model evaluation:

```
268/268 - 1s - loss: 0.4428 - accuracy: 0.7941 - 699ms/epoch - 3ms/step
Loss: 0.44281700253486633, Accuracy: 0.7940524816513062
```

Result

A model accuracy of **79.4%** is observed with model trained on as less as 20 epochs and 30 neurons between 3 layers with activation functions relu, relu and sigmoid. Loss has also reduced to **44.2%**

Summary

- **IS_SUCCESSFUL** was identified as the **Target** of our model and the **rest of the variables as the features**.
- It was observed that for given data when **EIN** and **NAME** were dropped from features, increasing number of neurons, number of layers, changing activation function combinations or increasing number of epochs **didn't have any impact on improving the accuracy of the model or reducing loss**.
- It was observed that, on retaining the **NAME** column and only **dropping only EIN from features,** there was an immediate improvement to the model. This could indicate that applications from certain organizations might have been preferred due to factors like infrastructure, technical know-how and support etc by the organization. Therefore, the name of the organization might be equivalent to a whole lot of un-accounted for features which might be contributing to success of an applicant.
- The below hyperparameters helped achieve an accuracy **> 75% at 79.4%**

No.	Method	Number of Layers	Neurons per layer	Activation functions per layer	Epochs	Loss	Accuracy
4	**Checking Name/EID column values for duplicates (reappearance). Dropping only EIN column while retaining Name**	Input/Hidden1/ Output	20/10/1; input_dim = 447	relu/relu/ sigmoid	20	**44.3%**	**79.4%**

Three layers with two hidden (including input layer) and an output layer were optimum. relu, relu and sigmoid activation functions in consecutive layers was found to be optimum.

- It was also observed that by keeping the name features, loss reduced **from >50% to 44.3%.**

- **Future scope:** As the next steps we could probably try some alternate models to improve the results: Binning names further to include lesser unique values, increasing number of neurons on the layers since the number of input features is large, and then increasing number of epochs to train the model and this may increase the accuracy of the model and reduce losses further.