Kimberly Toro
Module 21 Challenge
UC Berkeley Berkeley Extension
Data Analysis and Visualization

Neural Network Model Report

Overview of the Analysis:

The nonprofit 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 your knowledge of machine learning and neural networks, you'll 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, you have received a CSV containing more than 34,000 organizations that have received funding from Alphabet Soup over the years. This analysis aims to develop a machine learning tool that can predict the success of applicants if funded by this company.

Results

Data Preprocessing

- What variable(s) are the target(s) for your model?
- What variable(s) are the features for your model?
- What variable(s) should be removed from the input data because they are neither targets nor features?

- For all four attempts, I decided that the model's target would be "IS_SUCCESSFUL" while the features were the rest of the columns. I removed identification columns such as EIN and NAME since they did not fit previous categories.

Compiling, Training, and Evaluating the Model

- How many neurons, layers, and activation functions did you select for your neural network model, and why?
- Were you able to achieve the target model performance?
- What steps did you take in your attempts to increase model performance?

For the first attempt

- I tried to mimic the given output. I have done some research and found that 'input_dim" means the number of features the dataset has.
- The other two were random and the units correspond to the output shape. So, I was wondering if my input_dim has to accumulate or total throughout.
- Then, I spread the input_dim as 10 and 30. I noticed no change once I trained and compiled the model. As a result, I knew something wasn't right. I checked the accuracy and it was low.

Kimberly Toro
Module 21 Challenge
UC Berkeley Berkeley Extension
Data Analysis and Visualization

```
nn = tf.keras.models.Sequential()
nn.add(tf.keras.layers.Dense(input_dim=43, units=80, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=10, units=30, activation="relu"))
nn.add(tf.keras.layers.Dense(units=1, activation="linear"))
nn.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_39 (Dense)	(None, 80)	3520
dense_40 (Dense)	(None, 30)	2430
dense_41 (Dense)	(None, 1)	31

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

Results: 268/268 - 0s - loss: 7.2033 - accuracy: 1.1662e-04 - 452ms/epoch - 2ms/step
Loss: 7.203272819519043, Accuracy: 0.00011661807366181165

Second attempt

I added another layer input_dim=20, units=10, activation="relu")). The previous layers remained the same and the activation also remained unchanged. Based on the epochs, the accuracy was around 72-73%. When I calculated the accuracy it was 0.7219 and a loss of 0.6307. Which was a decent increase from no accuracy from the first attempt. However, the ideal accuracy is 75%.

```
#Second Attempt
nn = tf.keras.models.Sequential()
nn.add(tf.keras.layers.Dense(input_dim=43, units=80, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=10, units=30, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=20, units=10, activation="relu"))
nn.add(tf.keras.layers.Dense(units=1, activation="linear"))
nn.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #

dense_50 (Dense)	(None, 80)	3520
dense_51 (Dense)	(None, 30)	2430
dense_52 (Dense)	(None, 10)	310
dense_53 (Dense)	(None, 1)	11

==========

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

Results: 268/268 - 1s - loss: 0.6307 - accuracy: 0.7219 - 842ms/epoch - 3ms/step
Loss: 0.6306994557380676, Accuracy: 0.7218658924102783

For the third attempt

I added different activations (sigmoid activation), which have hidden layers total of the input layer. I read that sigmoid helps with binary classification which is the goal of this machine learning model.

```
#Third Attempt
# Add different activations
# have hidden layers total the input layer?
#Added sigmoid activation, I read this helps with classifcation
nn = tf.keras.models.Sequential()
nn.add(tf.keras.layers.Dense(input_dim=43, units=80, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=10, units=30, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=10, units=20, activation="sigmoid"))
nn.add(tf.keras.layers.Dense(input_dim=10, units=20, activation="sigmoid"))
nn.add(tf.keras.layers.Dense(input_dim=20, units=10, activation="relu"))
nn.add(tf.keras.layers.Dense(units=1, activation="linear"))
nn.summary()
```

Model: "sequential_8"

Kimberly Toro
Module 21 Challenge
UC Berkeley Berkeley Extension
Data Analysis and Visualization

Layer (type)	Output Shape	Param #
dense_59 (Dense)	(None, 80)	3520
dense_60 (Dense)	(None, 30)	2430
dense_61 (Dense)	(None, 20)	620
dense_62 (Dense)	(None, 20)	420
dense_63 (Dense)	(None, 10)	210
dense_64 (Dense)	(None, 1)	11

Total params: 7211 (28.17 KB)
Trainable params: 7211 (28.17 KB)
Non-trainable params: 0 (0.00 Byte)

Results: Based on the epochs,I noticed more 0.73s and started to see an increase in ranges of 74%. In this attempt the loss is 0.5885 - accuracy: 0.7292 - 1s/epoch which is 72.9%. There is an increase of 0.08% but still not the ideal percentage I'm aiming at. I have also considered changing the optimizer (but will leave that for the next section) and the number of the shape of the trained model. The parameters have increased so perhaps making sure the model matches the shape of the trained/fit model.

Fourth attempt:

- I continued with the same activations, units, and input dims. The difference here is I changed the activation of the final output layer from 'linear' to 'sigmoid':

```
#Fourth attempt
#Continue with relu but change last layer to sigmoid
nn = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=80, activation="relu", input_dim=43),
    tf.keras.layers.Dense(units=30, activation="relu"),
    tf.keras.layers.Dense(units=20, activation="relu"),
    tf.keras.layers.Dense(units=20, activation="relu"),
    tf.keras.layers.Dense(units=10, activation="relu"),
    tf.keras.layers.Dense(units=1, activation="sigmoid")
])
nn.summary()
```

Model: "sequential_10"

Layer (type)	Output Shape	Param #

dense_71 (Dense)	(None, 80)	3520
dense_72 (Dense)	(None, 30)	2430
dense_73 (Dense)	(None, 20)	620
dense_74 (Dense)	(None, 20)	420
dense_75 (Dense)	(None, 10)	210
dense_76 (Dense)	(None, 1)	11

Total params: 7211 (28.17 KB)
Trainable params: 7211 (28.17 KB)
Non-trainable params: 0 (0.00 Byte)

Sometimes third time wasn't the charm and I continued with training. Based on the epochs, there was a similar output as the previous one. with scores loss: 0.5588 - accuracy: 0.7293. Therefore, adding different activations and perhaps units.

Using checkpoints and callbacks

I noticed I had to add a call back every 5 epochs which led me to modify the above. Below are the results of each attempt:

Attempt 1:

Kimberly Toro
Module 21 Challenge
UC Berkeley Berkeley Extension
Data Analysis and Visualization

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

ıble-click (or enter) to edit

```
# First hidden layer
nn.add(tf.keras.layers.Dense(input_dim=43, units=80, activation="relu"))
# Second hidden layer
nn.add(tf.keras.layers.Dense(input_dim=10, units=30, activation="relu"))
# Output layer
nn.add(tf.keras.layers.Dense(units=1, activation="linear"))
nn.summary()
```

```
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)
_________________________________________________________________
```

✓ 0s completed at 8:20 PM

268/268 - 0s - loss: 0.5965 - accuracy: 0.7293 - 443ms/epoch - 2ms/step
Loss: 0.5965037941932678, Accuracy: 0.7293294668197632

Attempt 2:

```
#Second Attempt
nn = tf.keras.models.Sequential()
nn.add(tf.keras.layers.Dense(input_dim=43, units=80, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=10, units=30, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=20, units=10, activation="relu"))
nn.add(tf.keras.layers.Dense(units=1, activation="linear"))
nn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 80)                3520

 dense_4 (Dense)             (None, 30)                2430

 dense_5 (Dense)             (None, 10)                310

 dense_6 (Dense)             (None, 1)                 11

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

268/268 - 1s - loss: 0.5931 - accuracy: 0.7284 - 978ms/epoch - 4ms/step
Loss: 0.5930800437927246, Accuracy: 0.728396475315094

Attempt 3:

Kimberly Toro
Module 21 Challenge
UC Berkeley Berkeley Extension
Data Analysis and Visualization

```
nn = tf.keras.models.Sequential()
nn.add(tf.keras.layers.Dense(input_dim=43, units=80, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=10, units=30, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=10, units=20, activation="sigmoid"))
nn.add(tf.keras.layers.Dense(input_dim=10, units=20, activation="sigmoid"))
nn.add(tf.keras.layers.Dense(input_dim=20, units=10, activation="relu"))
nn.add(tf.keras.layers.Dense(units=1, activation="linear"))
nn.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 80)                3520

 dense_8 (Dense)             (None, 30)                2430

 dense_9 (Dense)             (None, 20)                620

 dense_10 (Dense)            (None, 20)                420

 dense_11 (Dense)            (None, 10)                210

 dense_12 (Dense)            (None, 1)                 11

=================================================================
Total params: 7211 (28.17 KB)
Trainable params: 7211 (28.17 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

268/268 - 0s - loss: 7.1791 - accuracy: 0.5292 - 482ms/epoch - 2ms/step
Loss: 7.17914342880249, Accuracy: 0.5292128324508667

Attempt 4:

```
nn = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=80, activation="relu", input_dim=43),
    tf.keras.layers.Dense(units=30, activation="relu"),
    tf.keras.layers.Dense(units=20, activation="relu"),
    tf.keras.layers.Dense(units=20, activation="relu"),
    tf.keras.layers.Dense(units=10, activation="relu"),
    tf.keras.layers.Dense(units=1, activation="sigmoid")
])

nn.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_13 (Dense)            (None, 80)                3520

 dense_14 (Dense)            (None, 30)                2430

 dense_15 (Dense)            (None, 20)                620

 dense_16 (Dense)            (None, 20)                420

 dense_17 (Dense)            (None, 10)                210

 dense_18 (Dense)            (None, 1)                 11

=================================================================
Total params: 7211 (28.17 KB)
Trainable params: 7211 (28.17 KB)
Non-trainable params: 0 (0.00 Byte)
```

Kimberly Toro
Module 21 Challenge
UC Berkeley Berkeley Extension
Data Analysis and Visualization

268/268 - 0s - loss: 0.5821 - accuracy: 0.7290 - 485ms/epoch - 2ms/step
Loss: 0.5821083188056946, Accuracy: 0.7289795875549316

Optimization of the model

To optimize a model there are several strategies, for example:

- Dropping more or fewer columns.
- Creating more bins for rare occurrences in columns.
- Increasing or decreasing the number of values for each bin.
- Add more neurons to a hidden layer.
- Add more hidden layers.
- Use different activation functions for the hidden layers.
- Add or reduce the number of epochs to the training regimen.

Therefore for the next attempts, I will consider these options:

- First attempt: drop columns
- Second attempt Adding more neurons, fewer layers
- Third attempt: Control the number of epochs
- Fourth (if time permits): Activation functions + maybe tuning

First attempt: Drop columns

- I created a copy of application_df to Optmodel_1. At first, I had thought about creating separate DataFrames for each model but I kept this copy as the one to use for optimization. I dropped an additional column 'SPECIAL CONSIDERATIONS' most applications had an 'N' in it. After this, I trained the model with the same as the first one.
    - Results:
        - 268/268 - 0s - loss: 0.6375 - Accuracy: 0.7282 - 488ms/epoch - 2ms/step
        - Loss: 0.6374562978744507, Accuracy: 0.7281632423400879

Second attempt: Adding more neurons and less layers

Kimberly Toro
Module 21 Challenge
UC Berkeley Berkeley Extension
Data Analysis and Visualization

```
] #Second Attempt
  nn = tf.keras.models.Sequential()
  nn.add(tf.keras.layers.Dense(input_dim=41, units=90, activation="relu"))
  nn.add(tf.keras.layers.Dense(input_dim=30, units=30, activation="relu"))
  nn.add(tf.keras.layers.Dense(input_dim=10, units=20, activation="relu"))
  nn.add(tf.keras.layers.Dense(units=1, activation="linear"))
  nn.summary()

  # what I think it is: param / units = suceeding units + 1? except first one.
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_17 (Dense)            (None, 90)                3780

 dense_18 (Dense)            (None, 30)                2730

 dense_19 (Dense)            (None, 20)                620

 dense_20 (Dense)            (None, 1)                 21

=================================================================
Total params: 7151 (27.93 KB)
Trainable params: 7151 (27.93 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- I added more units and increased the input_dim on layer #2.
  Results:
  268/268 - 0s - loss: 0.6079 - Accuracy: 0.7205 - 468ms/epoch - 2ms/step
  Loss: 0.6078738570213318, Accuracy: 0.7204664945602417

Third attempt: Controlling the number of epochs

- I revised the original third-attempt neurons
- I also used a callback (which I didn't do with the other two attempts).

```
#Third Attempt
#Revised a little bit  of the original third model as well.
# Control the number of epochs
#Adding more neurons and units lowered a bit, I might find some sort of middle

nn = tf.keras.models.Sequential()
nn.add(tf.keras.layers.Dense(input_dim=41, units=80, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=20, units=40, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=20, units=10, activation="relu"))
nn.add(tf.keras.layers.Dense(input_dim=11, units=5, activation="sigmoid"))
nn.add(tf.keras.layers.Dense(units=1, activation="linear"))
nn.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_35 (Dense)            (None, 80)                3360

 dense_36 (Dense)            (None, 40)                3240

 dense_37 (Dense)            (None, 10)                410

 dense_38 (Dense)            (None, 5)                 55

 dense_39 (Dense)            (None, 1)                 6

=================================================================
Total params: 7071 (27.62 KB)
Trainable params: 7071 (27.62 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- Results: 268/268 - 0s - loss: 8.1631 - accuracy: 0.4708 - 464ms/epoch - 2ms/step

Kimberly Toro
Module 21 Challenge
UC Berkeley Berkeley Extension
Data Analysis and Visualization

Loss: 8.163082122802734, Accuracy: 0.4707871675491333

Fourth attempt

- I had trouble importing keras_tuner and I do have it installed. Here I used fewer epochs, a few more units, and input_dim the same.

```
[ ] #Fourth attempt
    # I had trouble importing keras_tuner and I do have it installed so not sure what's the reason.

    #Here I will use less epochs, few more units, input_dim the same
    nn = tf.keras.models.Sequential([
        tf.keras.layers.Dense(units=90, activation="relu", input_dim=41),
        tf.keras.layers.Dense(units=40, activation="relu"),
        tf.keras.layers.Dense(units=1, activation="sigmoid")
    ])

    nn.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_43 (Dense)            (None, 90)                3780

 dense_44 (Dense)            (None, 40)                3640

 dense_45 (Dense)            (None, 1)                 41

=================================================================
Total params: 7461 (29.14 KB)
Trainable params: 7461 (29.14 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- Results: 268/268 - 0s - loss: 0.5565 - accuracy: 0.7300 - 450ms/epoch - 2ms/step
  Loss: 0.5565274357795715, Accuracy: 0.7300291657447815
  - This is the highest accuracy so far.

Fifth attempt

- Fining tuning based on class activity.
  a. Here I realized I had to install it directly on Colab which worked. I was able to then successfully import keras_tuner.
  b. I had to stop, it was running more than 100 trials and I stopped at 165.

Kimberly Toro
Module 21 Challenge
UC Berkeley Berkeley Extension
Data Analysis and Visualization

```
[ ]  #Fifth attempt
     #Using fine tuner
     # In class activity 22.2.04 modified code
     def create_model(hp):
         nn_model = tf.keras.models.Sequential()

         # Allow kerastuner to decide which activation function to use in hidden layers
         activation = hp.Choice('activation',['relu','tanh','sigmoid'])

         # Allow kerastuner to decide number of neurons in first layer
         nn_model.add(tf.keras.layers.Dense(units=hp.Int('first_units',
             min_value=1,
             max_value=90,
             step=3), activation=activation, input_dim=41))

         # Allow kerastuner to decide number of hidden layers and neurons in hidden layers
         for i in range(hp.Int('num_layers', 1, 6)):
             nn_model.add(tf.keras.layers.Dense(units=hp.Int('units_' + str(i),
                 min_value=1,
                 max_value=3,
                 step=3),
                 activation=activation))

         nn_model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

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

         return nn_model
```

```
[ ]  #Fifth attempt
     #Using fine tuner
     # Based on in class activity
     tuner = kt.Hyperband(
         create_model,
         objective="val_accuracy",
         max_epochs=80,
         hyperband_iterations=2)
```

Results: best. hyper.values =
{'activation': 'sigmoid',
'first_units': 10,
'num_layers': 4,
'units_0': 1,
'units_1': 1,
'units_2': 1,
'units_3': 1,
'units_4': 1,
'units_5': 1,
'tuner/epochs': 27,
'tuner/initial_epoch': 9,
'tuner/bracket': 2,

Kimberly Toro
Module 21 Challenge
UC Berkeley Berkeley Extension
Data Analysis and Visualization

'tuner/round': 1,
'tuner/trial_id': '0064'}

268/268 - 1s - loss: 0.5766 - accuracy: 0.7340 - 594ms/epoch - 2ms/step
Loss: 0.5766450762748718, Accuracy: 0.73399418592453

Summary

In the starter code, the target for the model is "IS_SUCCESSFUL." while the features were all columns except for identification columns like EIN and NAME. The first attempt had low accuracy and high loss, indicating poor performance. Adjustments made by spreading the input dimension resulted in no noticeable improvement. In the second attempt, An additional hidden layer was added with increased neurons in the second layer. Achieved an accuracy of around 72-73%, an improvement from the first attempt but still below the target of 75%. The third and fourth included different activation functions but accuracy remained around 72 to nearly 73%.

To optimize the model, dropping columns maintained similar accuracy, adding more neurons led to a slight drop, controlling epochs and adding a call back dropped significantly. Yet, I noticed the third attempt on the starter code had some drops. Fourth had a 73.00%, the highest so far making adjustments to the previous attempts. Finally, the last one implemented fine-tuning.

As a result, the report outlines efforts to develop a machine-learning tool for Alphabet Soup to predict the success of funding applicants. Through multiple attempts, adjustments were made to the model architecture, including changes to neurons, layers, and activation functions. Data preprocessing involved dropping irrelevant columns and controlling epochs. Despite these efforts, the accuracy remained below the target of 75%. Further fine-tuning and exploration of alternative techniques are recommended to improve model performance. I would recommend still using ‘relu’ and ‘sigmoid’ since both work well for binary classification. Also looking more into different functions and their purpose such as ‘drop’ and ‘pooling’. I am not certain if I would recommend something different from deep learning since as mentioned before, many functions can be explored within Keras and Tensorflow.