Module 21: Deep Learning Challenge

Clare Sakauye

UC Davis Data Analytics Bootcamp

August 20th, 2023

Overview

The purpose of this analysis is to create a deep learning model that will help Alphabet Soup determine which applicants will be successful in their ventures and therefore should be awarded funding.

Results

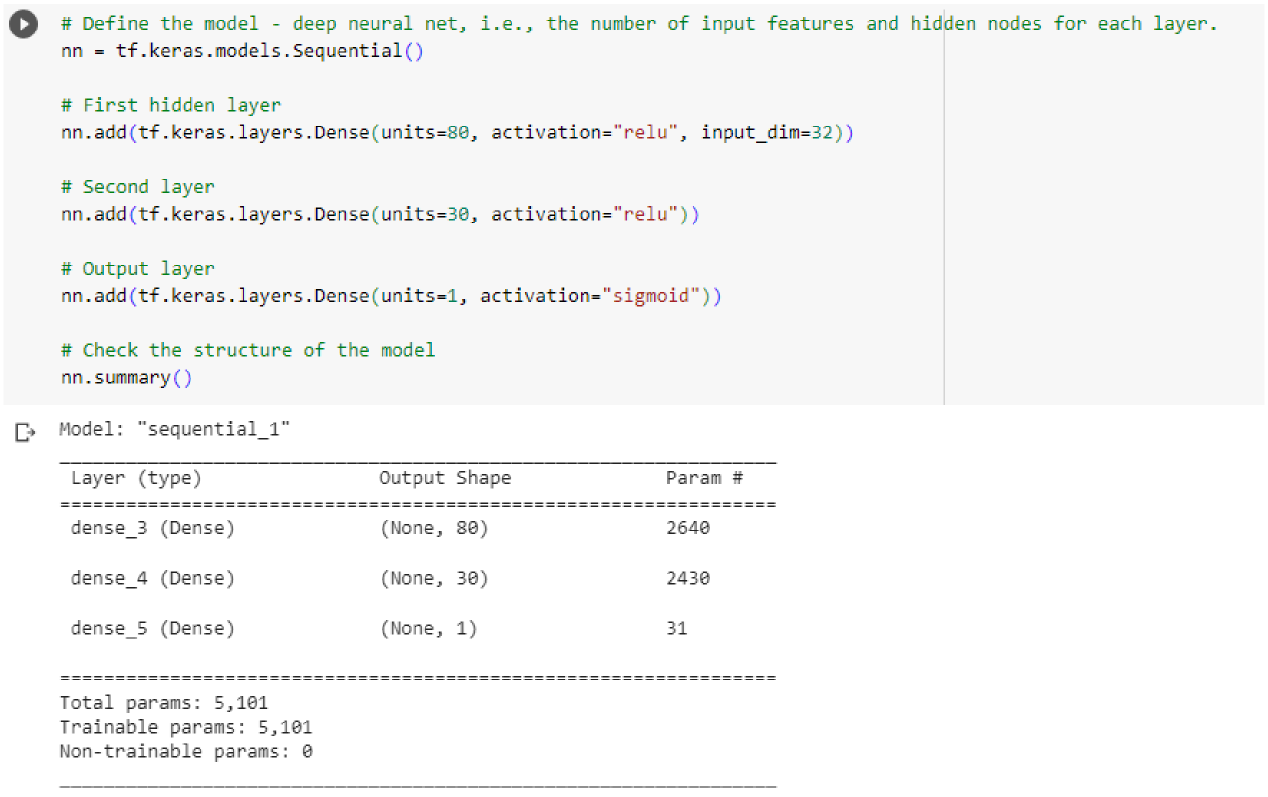
Data Preprocessing

- What variable(s) are the target(s) for your model?
    - IS_SUCCESSFUL is the target for this model because it is a dependent variable and what we want the model to help us predict.
- What variable(s) are the features for your model?
    - APPLICATION_TYPE, AFFILITAION, CLASSIFICATION, USE_CASE, ORGANIZATION and SPECIAL_CONSIDERATIONS are the features for this model. These are features because they are all independent variables/factors that help the model determine if a venture will be successful.
- What variable(s) should be removed from the input data because they are neither targets nor features?
    - NAME and EIN should be removed from the data because they do not influence whether the venture will be successful but are instead used for identification.

Compiling, Training, and Evaluating the Model

- How many neurons, layers, and activation functions did you select for your neural network model, and why?
    - For the original model, I used three layers. The first layer had 80 neurons and RELU activation. It was followed by a second RELU activated layer with 30 neurons and a last output layer with one neuron and a sigmoid activation. I followed the starter code model for this one since I knew we would be creating different models aimed at higher accuracy.
    - RELU is a good activation function to start with since it can learn relatively fast compared to the other activation functions. Additionally, sigmoid is a good activation function to end with because it is helpful for predicting probabilities. In addition to the activation functions, I thought that the number of layers and neurons was a good

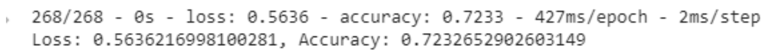
starting point. Three layers was the maximum number of layers we experimented with in class and the descending number of neurons from 80 to 30 and the output of one, allowed for enough power for deep learning and our dataset, but not too much that it was too expensive to do.



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

# First hidden layer
nn.add(tf.keras.layers.Dense(units=80, activation="relu", input_dim=32))

# Second layer
nn.add(tf.keras.layers.Dense(units=30, activation="relu"))

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

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

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 80)                2640

 dense_4 (Dense)             (None, 30)                2430

 dense_5 (Dense)             (None, 1)                 31

=================================================================
Total params: 5,101
Trainable params: 5,101
Non-trainable params: 0
_________________________________________________________________
```

- Were you able to achieve the target model performance?
  - For this challenge, we were tasked with exceeding 75% accuracy. The original model as described above did not reach 75% or greater accuracy. It was very close with an accuracy of 72.3% but not good enough. Additionally, there was a loss of 0.5635, which should ideally be closer to 0.

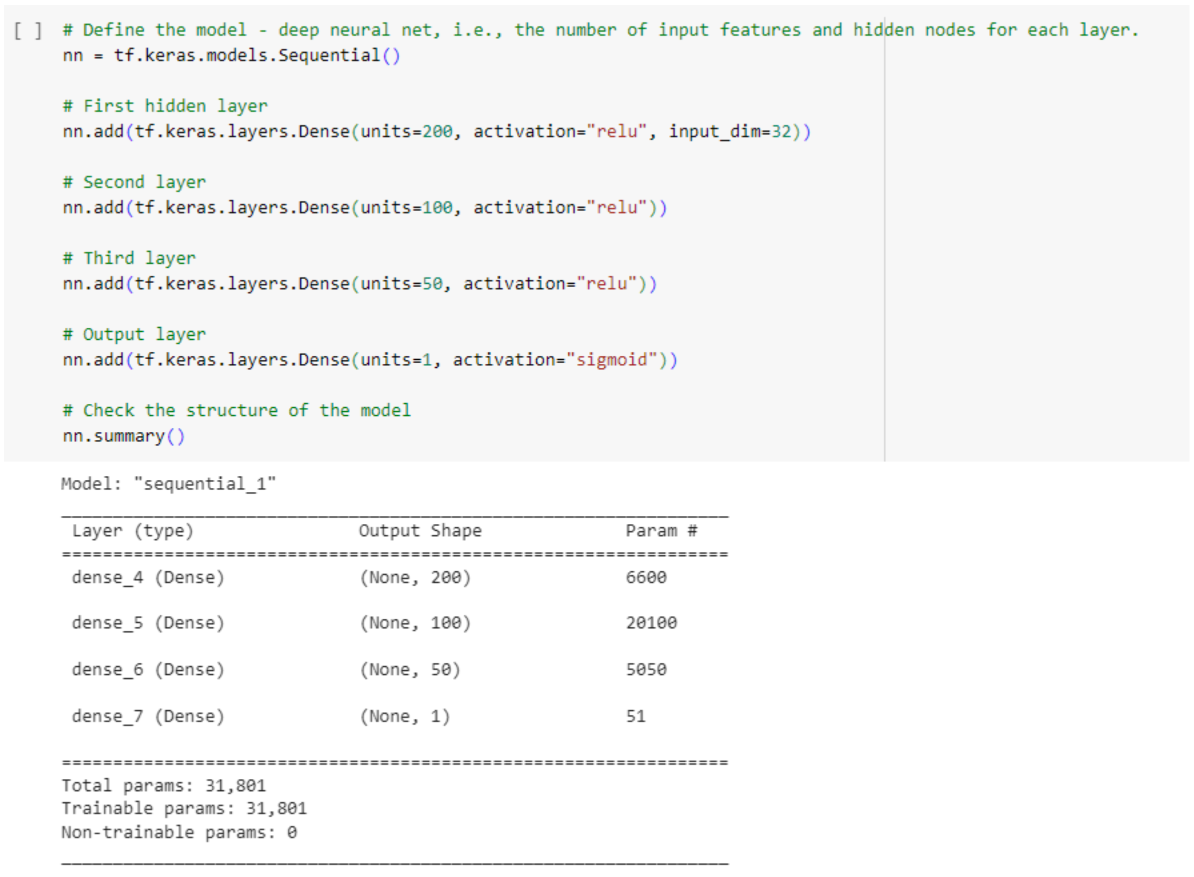


```
268/268 - 0s - loss: 0.5636 - accuracy: 0.7233 - 427ms/epoch - 2ms/step
Loss: 0.5636216998100281, Accuracy: 0.7232652902603149
```

- What steps did you take in your attempts to increase model performance?
  - In an attempt to increase model performance, the model was modified three different times.
    - **Optimization Attempt 1:** In my first attempt to increase model performance, I added more neurons to each layer, one additional layer and increased the epochs. Increasing these hyperparameters made the model more accurate and thus increased performance, but not significantly. The original accuracy and

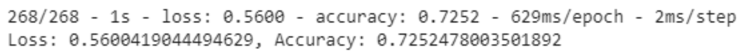
loss were 72.3% and 0.564. With the first attempt at optimization at three layers of 200, 100 and 50 neurons, all activated by RELU, the same output layer and 20 epochs, the accuracy and loss only improved to 72.5% and 0.560.

- Given the added time solve to account for the deeper learning model, I'm not sure if the slight gain in accuracy and loss is worth it. Despite an additional layer, nearly double the number of neurons and fifteen more epochs, the model did not improve significantly.



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

    # First hidden layer
    nn.add(tf.keras.layers.Dense(units=200, activation="relu", input_dim=32))

    # Second layer
    nn.add(tf.keras.layers.Dense(units=100, activation="relu"))

    # Third layer
    nn.add(tf.keras.layers.Dense(units=50, activation="relu"))

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

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

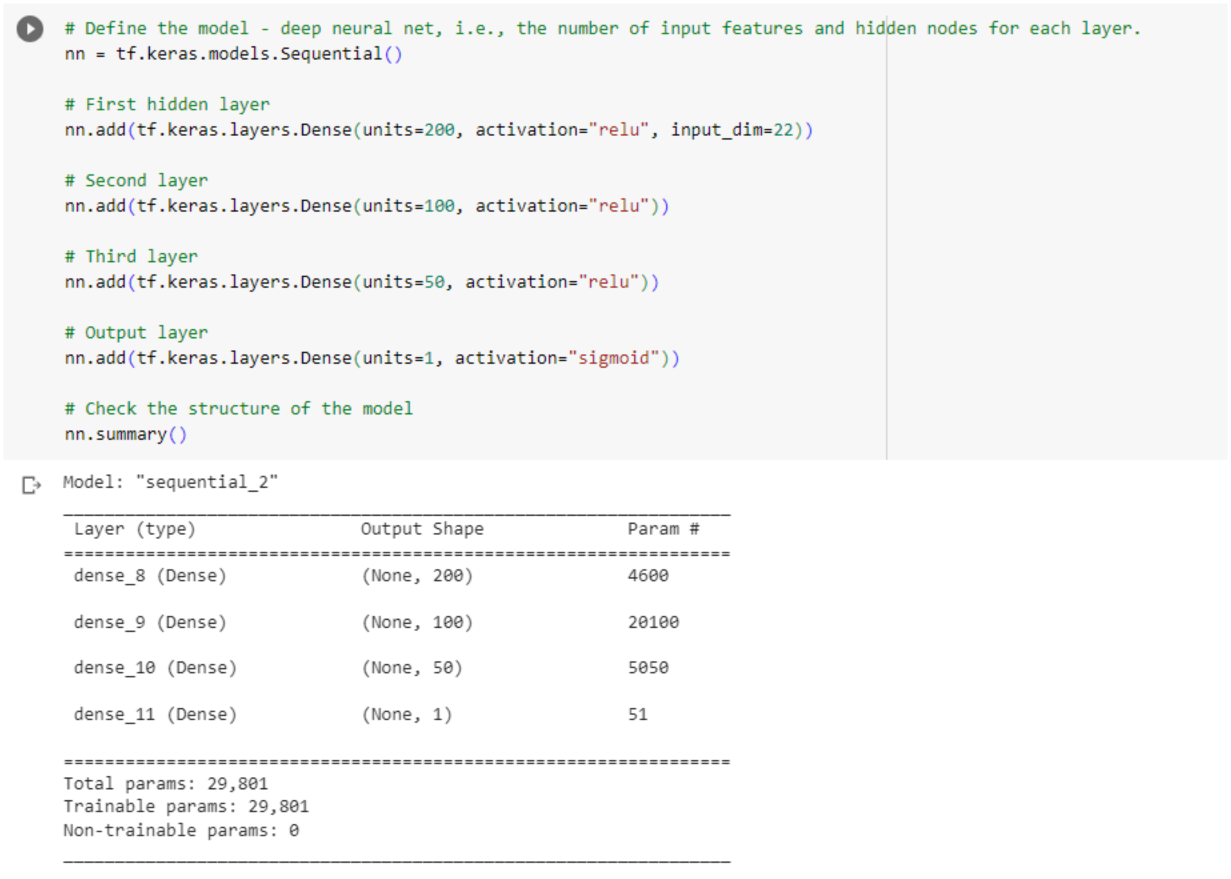
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 200)               6600

 dense_5 (Dense)             (None, 100)               20100

 dense_6 (Dense)             (None, 50)                5050

 dense_7 (Dense)             (None, 1)                 51

=================================================================
Total params: 31,801
Trainable params: 31,801
Non-trainable params: 0
_________________________________________________________________
```



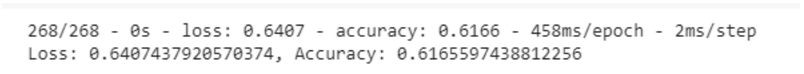
```
268/268 - 1s - loss: 0.5600 - accuracy: 0.7252 - 629ms/epoch - 2ms/step
Loss: 0.5600419044494629, Accuracy: 0.7252478003501892
```

- **Optimization Attempt 2:** For my second optimization attempt, I kept the number of layers, neurons and epochs the same as my first optimization attempt, as well as the types of activation functions. The deeper model did account for a slight increase in improvement and I wanted to see if I could build further on that. Instead of looking to make the model any deeper for this attempt, I tried to reassess the importance of some of the features.

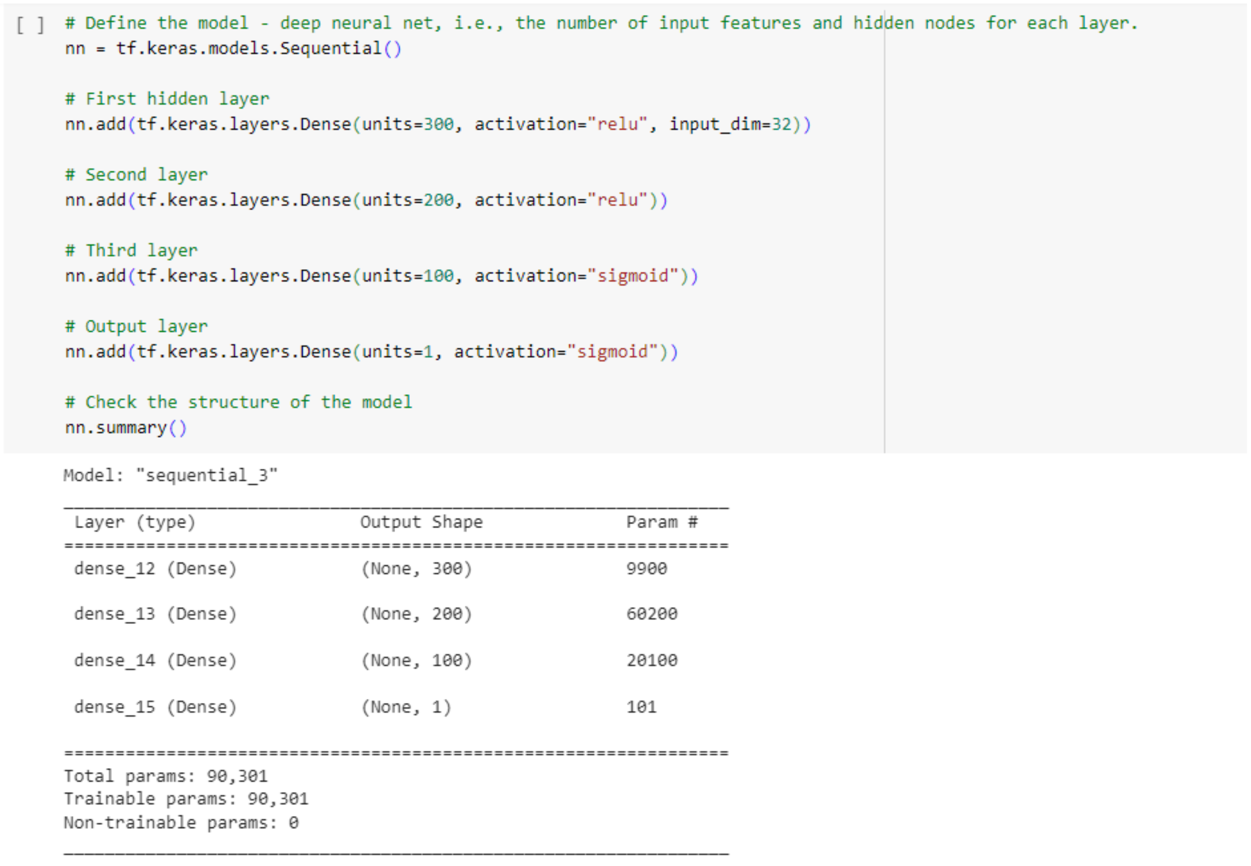
- Given the features of APPLICATION_TYPE, AFFILIATION, CLASSIFICATION, USE_CASE, ORGANIZATION, STATUS, INCOME_AMT, SPECIAL_CONSIDERATIONS and ASK_AMT, I thought that affiliation and organization might be the least impactful. To determine if this was correct, I dropped affiliation and organization from the features list. This actually made my model significantly worse than the original one. The accuracy dropped to 62% and the loss increased to 0.64. Affiliation and organization are important to the model because when included they raised the accuracy nearly ten percent to 72 and dropped the loss nearly ten percent as well. This was not a good optimization attempt.


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

# First hidden layer
nn.add(tf.keras.layers.Dense(units=200, activation="relu", input_dim=22))

# Second layer
nn.add(tf.keras.layers.Dense(units=100, activation="relu"))

# Third layer
nn.add(tf.keras.layers.Dense(units=50, activation="relu"))

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

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

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 200)               4600

 dense_9 (Dense)             (None, 100)               20100

 dense_10 (Dense)            (None, 50)                5050

 dense_11 (Dense)            (None, 1)                 51

=================================================================
Total params: 29,801
Trainable params: 29,801
Non-trainable params: 0
_________________________________________________________________
```

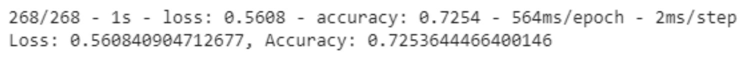

```
268/268 - 0s - loss: 0.6407 - accuracy: 0.6166 - 458ms/epoch - 2ms/step
Loss: 0.6407437920570374, Accuracy: 0.6165597438812256
```

- **Optimization Attempt 3:** For my third and final optimization types, I looked at increasing the number of neurons and changing the activation functions. In my first attempt at improving model performance, there was a slight improvement with making the model deeper. Recognizing that the model could easily become too costly, I decided to keep the number of layers the same as well as the epochs. For each layer except the output, I doubled the number of neurons. I also changed the third layer activation from RELU to sigmoid. This led to the best accuracy out of the original model and other two optimization attempts at 72.54% versus the second highest of 72.52%. Interestingly, it did not lead to the best loss. The loss for this model was 0.5608, but Optimization Attempt 1, with 72.52% accuracy, had the best loss at 0.5666.



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

     # First hidden layer
     nn.add(tf.keras.layers.Dense(units=300, activation="relu", input_dim=32))

     # Second layer
     nn.add(tf.keras.layers.Dense(units=200, activation="relu"))

     # Third layer
     nn.add(tf.keras.layers.Dense(units=100, activation="sigmoid"))

     # 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_12 (Dense)            (None, 300)               9900

 dense_13 (Dense)            (None, 200)               60200

 dense_14 (Dense)            (None, 100)               20100

 dense_15 (Dense)            (None, 1)                 101

=================================================================
Total params: 90,301
Trainable params: 90,301
Non-trainable params: 0
_________________________________________________________________
```



```
268/268 - 1s - loss: 0.5608 - accuracy: 0.7254 - 564ms/epoch - 2ms/step
Loss: 0.560840904712677, Accuracy: 0.7253644466400146
```

Summary

Across four different deep learning models to assess whether an applicant's venture will succeed, the highest achieved accuracy was 72.54%. The best loss was 0.5600 but from a different model than the one with the highest accuracy. Despite Optimization Attempt 2, all the models were roughly 72% accurate with 0.56 loss. I don't think that the additional layers, neurons, epochs or different activation functions in Optimization Attempts 1 and 3 improve the model. While they did improve the accuracy and loss minutely, I don't think the impact was big enough to outweigh the longer run times for the code. The original model is the best because it gives the best accuracy and loss with the least expensive model. To get a model over 75% accurate, I think other forms of machine learning may need to be utilized. Neural networks are expensive in time and computing power. I would recommend exploring more basic forms of supervised learning such as decision trees or logistic regression. It would also be important to assess how much over 75% accuracy is worth. I think a 72% accuracy is decent, but it would be up to Alphabet Soup to determine how much an additional 3-4% is worth compared to the time, effort and money that would go into achieving it.