

Deep Learning Analysis

Overview

The purpose of this analysis was to look at data from a nonprofit, alphabet soup, and see if we can use deep learning to help predict whether candidates will be successful based on certain features, including organization type and classification of the candidate. I used a notebook within Google Colab to solve this question, utilizing tensorflow, keras tuner, pandas, and sklearn to perform the required analysis. I found a model with an accuracy of 0.7326 using a keras tuner, which I believe is a pretty good accuracy for this task.

Results

Data Preprocessing

- What variable is the target for my model?

The target variable in this case is the "IS_SUCCESSFUL" variable because we are trying to predict whether the candidates will be successful.

- What variables are the features for my model?

The features are all variables other than EIN, NAME, and IS_SUCCESSFUL. In other words, they are: APPLICATION_TYPE, AFFILIATION, CLASSIFICATION, USE_CASE, ORGANIZATION STATUS, INCOME_AMT, SPECIAL_CONSIDERATIONS, and ASK_AMT.

- What variables should be removed from the input data because they are neither targets nor features?

The variables that should be removed are NAME and EIN because they are basically just identifiers for each of the candidates.

Compiling, Training, and Evaluating the Model

- How many neurons, layers, and activation functions did I select for my neural network model, and why?

The original amount of layers I chose for my neural network was 2 hidden layers and one output layer. Then, the number of units for the two hidden layers was 8 and the number of units for the output layer was 1. The number of units in the hidden layers was largely random for me. I decided to go with what I considered to be a slightly high number, 8, because I thought that it might improve accuracy of the model. The number of layers was also largely random. I went with a lower amount of hidden layers to improve the performance of the fitting. The reason the output layer only had one unit was that we are just outputting a 1 or 0 for successful or not successful, respectively. I also originally chose a rectified linear activation function (relu) for the hidden layers and sigmoid for the output layer. I decided on relu for the hidden layers because I thought that we didn't want to send out a negative value. I chose sigmoid for the output layer because we want a result between 0 and 1.

I used a keras tuning object to find better hyperparameters. The result of that search, within the bounds of the search, was: 7 hidden layers, all with hyperbolic tangent (tanh) as the activation function, and 26, 1, 11, 16, 1, 26, and 6 units for each of the layers respectively. The output layer still had sigmoid as the activation function and still had 1 unit. I could not explain why exactly those are the best hyperparameters. It was what the keras tuner object found. For reference, the model found from the tuner had an accuracy of .7326, and the model I first found had an accuracy of .7285. I chose the former model because it had a slightly higher accuracy.

- Was I able to achieve the target model performance?

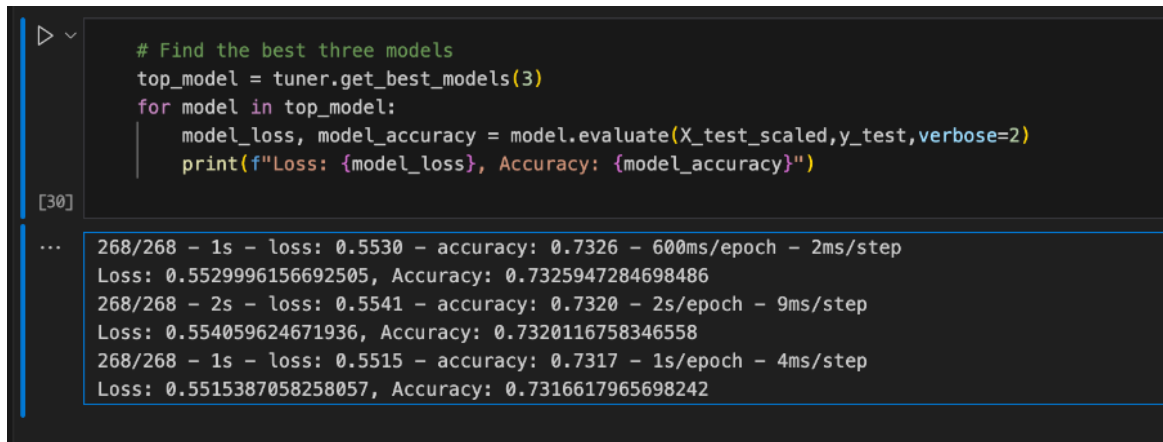
I was not able to achieve the target model performance. In my second attempt, in the optimization notebook, I did more bins for the APPLICATION_TYPE and CLASSIFICATION variables because I thought it would help improve the accuracy. Then, I tried three different models. All of them had an input dimension of 50 and one output unit, with sigmoid as the output activation function.

The first model had three hidden layers, each with 10 units and relu as the activation function. It ended up having an accuracy of 0.728.

The second model was the same except that it used tanh as the activation function. That had an accuracy of 0.725.

The third model had 5 hidden layers, all using relu, and they all had 10 units in them. That had an accuracy of 0.723.

In summary, all of these models had almost identical accuracy. I then tried to use a keras tuner to find better hyperparameters. Using that, as outlined in the previous bullet point, I achieved an accuracy of 0.7326. That was the best I could do, and none of it reached the .75 accuracy required. I've imbedded a screenshot of the top three models found using keras tuner:

A screenshot of a Jupyter Notebook interface. The top part shows a code cell with Python code for finding the best three models using Keras tuner. The code includes comments and function calls like `tuner.get_best_models(3)` and `model.evaluate`. The bottom part shows the output of the code, which is a series of lines reporting loss and accuracy for different models and epochs. The output is formatted with a progress bar (e.g., 268/268) and timing information (e.g., 1s, 2s, 600ms/epoch).

```
# Find the best three models
top_model = tuner.get_best_models(3)
for model in top_model:
    model_loss, model_accuracy = model.evaluate(X_test_scaled,y_test,verbose=2)
    print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

[30]

```
... 268/268 - 1s - loss: 0.5530 - accuracy: 0.7326 - 600ms/epoch - 2ms/step
      Loss: 0.5529996156692505, Accuracy: 0.7325947284698486
      268/268 - 2s - loss: 0.5541 - accuracy: 0.7320 - 2s/epoch - 9ms/step
      Loss: 0.554059624671936, Accuracy: 0.7320116758346558
      268/268 - 1s - loss: 0.5515 - accuracy: 0.7317 - 1s/epoch - 4ms/step
      Loss: 0.5515387058258057, Accuracy: 0.7316617965698242
```

- What steps did I take in my attempts to increase model performance?

As mentioned previously, I tried changing activation functions, tried changing the number of layers, and tried changing the binning procedure. I also used a keras tuner object to improve the performance (which did end up working, although only marginally).

Summary

Overall, I would say that the model I found using the keras tuner worked pretty well, with a loss of 0.5530 and an accuracy of 0.7326. I think what might be helpful to try in the future would be to try different activation functions for each of the hidden layers, to try a larger maximum node amount per layer, and more importantly to do more preprocessing of the data. I think that is where I didn't experiment as much as I could have, I mostly randomly binned the data differently for the two variables I mentioned and stopped experimenting there. I also noticed that the ASK_AMT variable overwhelmingly had 5000 as its value, so perhaps in the future I could have changed that to basically be a binary value of "5000" or "not 5000". Unfortunately, I would say that most of my attempts to optimize the model were largely in vain, as my original accuracy was 0.7285 and my best accuracy was 0.7326. I think preprocessing the data is perhaps where my future efforts would best be focused in this particular case.

To summarize, I would say a model with a different amount of layers and nodes with different activation functions per layer might do better here. I would try to find it with the keras tuner. However, I think in my case, the more important first step would be to preprocess the data.