Hiam Debsi
February 5th, 2022

DEEP LEARNING HOMEWORK

Charity Funding Predictor

OVERVIEW

The purpose of this task is to evaluate the efficiency of neural networks and deep learning to help predict whether applicants for funding will be successful by the non-profit foundation Alphabet Soup. This will be achieved using a provided CSV with over 34,000 organizations that have received funding from this non-profit.

RESULTS

Data Processing

- In all three attempts, our target variable was the IS_SUCCESSFUL column, which indicated whether the money was used effectively.
- All the remaining columns (including our target variable) were the features of our model, except those that were identification columns.
- The columns EIN and NAME didn't contain any relevant information to the model and therefore were dropped.
- To further optimize the data, values for CLASSIFICATION and APPLICATION_TYPE were binned in "Other" if less than 528 and 900 respectively.
- Before splitting the data into training and test subsets, the data was converted from categorical to numerical through the one-hot encoding method (pd.get_dummies), which resulted in 44 features in total.

Compiling, Training, and Evaluating the Model

The overall goal was to create a model that will achieve a predictive accuracy higher than 75%. Three different models were tested using a neural network model:

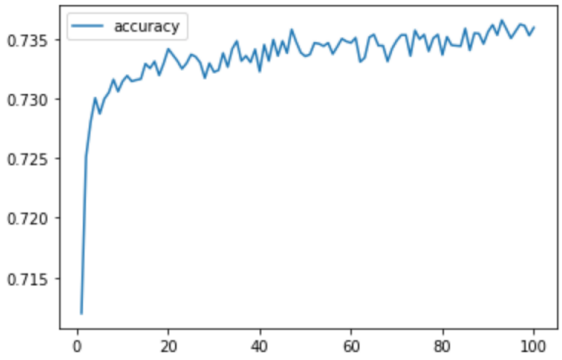
Model #1 (AlphabetSoupCharity.ipynb) Hyperparameters:

- Neurons = 10 (layer 1), 15(layer 2), 1(output layer)
- Layers = 3
- Epochs = 100
- Activation Function = ReLU (layers 1, 2), Sigmoid (output layer)

For this first model, the hyperparameters were chosen in a random matter but with purpose to have a baseline to work with. To have a relatively speedy model and a good connectivity, I ensured that the number of neurons is higher than one digit and that more than 1 layer was evaluated. More precisely, I had one layer with 10 neurons, and a second layer with 15

neurons. As for the activation function, given its computational simplicity and effectiveness, I used ReLU for the input layers. As for the output layer, I decided to use Sigmoid to get a binary outcome.

Overall Result:




```
# 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: 0.5554 - accuracy: 0.7277
Loss: 0.5554370880126953, Accuracy: 0.7276967763900757
```

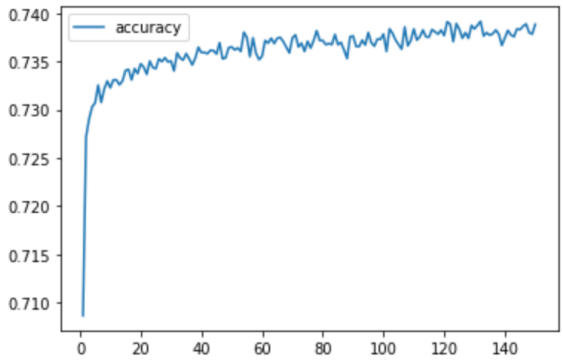
As seen above, the accuracy score of this model was of 72.77%. This means that about 72.8% of the model's predicted values were correct. Given that it isn't quite yet 75%, I investigated a second set of hyperparameters in Model #2.

Model #2 (AlphabetSoupCharity_Optimization_Attempt_1.ipynb) Hyperparameters:

- Neurons = 20 (layer 1), 15 (layer 2), 10 (layer 3), 1(output layer)
- Layers = 4
- Epochs = 150
- Activation Function = ReLU (layers 1, 2), Tanh (layer 3), Sigmoid (output layer)

Here, the hyperparameters were chosen to try and improve our baseline results. To keep a relatively speedy model and a good connectivity, I increased the number of neurons and the number of layers. More precisely, I had one layer with 20 neurons, and a second layer with 15 neurons, and a third with 10. As for the activation function, given its computational simplicity and effectiveness, I used ReLU for the input layers 1 and 2. For the third input layer, I applied the tanh activation layer. Tanh is identified by a characteristic S curve; and it transforms the output range between -1 and 1. As for the output layer, I kept it as Sigmoid. To try an increase the likelihood that the model achieves optimal weight coefficients, I increase the number of epochs by 50 (total of 150).

Overall Result:




```
# 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: 0.5555 - accuracy: 0.7301
Loss: 0.5555146932601929, Accuracy: 0.7301457524299622
```

As seen above, the accuracy score of this model was of 73.01%. This means that about 73% of the model's predicted values were correct. Even though there is a slight improvement, just like the first model, it is less than 75%.

Model #3 (AlphabetSoupCharity_Optimization_Attempt_2.ipynb) Hyperparameters:
Given that in both previous models an accuracy score of 75% wasn't achievable, I decided to use the Keras Tuner library for my third and final attempt. The Keras Tuner library helps you pick the optimal set of hyperparameters for your model through what is called hypertuning. In addition, I included a 'stop-early' function to stop running the trial once I have 5 consecutive epochs with no score improvement, to make it more efficient.

```
# Create a method that creates a new Sequential model with hyperparameter options
def create_model(hp):
    nn = tf.keras.models.Sequential()

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

    # Allow kerastuner to decide number of neurons in first layer
    nn.add(tf.keras.layers.Dense(units=hp.Int('first_units',
        min_value=1,
        max_value=30,
        step=5), activation=activation, input_dim=number_input_features))

    # Allow kerastuner to decide number of hidden layers and neurons in hidden layers
    for i in range(hp.Int('num_layers', 1, 5)):
        nn.add(tf.keras.layers.Dense(units=hp.Int('units_' + str(i),
            min_value=1,
            max_value=30,
            step=5),
            activation=activation))

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

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

    return nn
```

Overall Results:

```
# Get top 3 model hyperparameters and print the values
top_hyper = tuner.get_best_hyperparameters(3)
for param in top_hyper:
    print(param.values)
```

```
{'activation': 'tanh', 'first_units': 26, 'num_layers': 5, 'units_0': 11, 'units_1': 11, 'u
nits_2': 6, 'units_3': 26, 'units_4': 6, 'tuner/epochs': 20, 'tuner/initial_epoch': 7, 'tun
er/bracket': 2, 'tuner/round': 2, 'tuner/trial_id': 'f16c4053d47f86870b1150cf1a35d128'}
{'activation': 'tanh', 'first_units': 21, 'num_layers': 4, 'units_0': 16, 'units_1': 21, 'u
nits_2': 16, 'units_3': 16, 'units_4': 26, 'tuner/epochs': 20, 'tuner/initial_epoch': 7, 't
uner/bracket': 2, 'tuner/round': 2, 'tuner/trial_id': '13c6fff028dc519120416f677d0de886'}
{'activation': 'tanh', 'first_units': 26, 'num_layers': 4, 'units_0': 11, 'units_1': 6, 'un
its_2': 26, 'units_3': 21, 'units_4': 1, 'tuner/epochs': 7, 'tuner/initial_epoch': 3, 'tune
r/bracket': 2, 'tuner/round': 1, 'tuner/trial_id': '738c17ad60dcd538df6ad541070c17be'}
```

Based on the results obtained, the best hyperparameters were the following:

- Neurons = 26 (layer 1), 6(output layer)
- Layers = 5
- Epoch = 20
- Activation Function = Tanh (all layers)

```
# Evaluate the top 3 models against the test dataset
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}")
```

```
268/268 - 1s - loss: 0.5566 - accuracy: 0.7334
Loss: 0.5565676093101501, Accuracy: 0.7334110736846924
268/268 - 1s - loss: 0.5541 - accuracy: 0.7327
Loss: 0.5541496276855469, Accuracy: 0.7327113747596741
268/268 - 1s - loss: 0.5551 - accuracy: 0.7326
Loss: 0.555087149143219, Accuracy: 0.7325947284698486
```

With these hyperparameters and using the Keras Tuner model, the best achievable accuracy score is of 73.34%. This indicates that about 73% of this model's predicted values aligned with those of the true dataset values. Although slightly higher than the other two models, it is still not at a 75% accuracy.

SUMMARY

Whether encoded manually or using Keras Tuner, all three attempted models were unable to achieve a target predictive accuracy higher than 73.34%. Hypertuning may have improved small marginal adjustments, however given the categorical nature of the problem, other methods such as Random Forest, could have been a better option. Random Forest Classifier is a method which groups a set of decision trees randomly from a subset of training data and then returns a prediction. Overall, Random Forest Classifiers are more commonly used given their high accuracy and medium interpretability.

If more time were to be allocated, many other adjustments could have been made to try and optimize the accuracy score:

- Reducing the number of overall features, as it is known to help improve accuracy scores, since noise is being removed.
- Looking into other common metrics to further classify the issue of the model, such as precision and f1 scores.
- Comparing the neural network model with other classification models such as the Random Forest Classifier to see if the accuracy score does improve.