Rafael Orihuela Brindis

Neural Network Model Report

The main objective of this report is to explain the results that I got using the "Neural Network Model" analyzing which of a group of applicants have the best chance of success in their ventures.

- **Data**: The information was stored in a csv file, where each column was a variable, these columns are:
  - **EIN** and **NAME**—Identification columns
  - **APPLICATION_TYPE**—Alphabet Soup application type
  - **AFFILIATION**—Affiliated sector of industry
  - **CLASSIFICATION**—Government organization classification
  - **USE_CASE**—Use case for funding
  - **ORGANIZATION**—Organization 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.

In the first attempt I used all the variables, with the exceptions of the EIN and NAME variables because that information wasn't useful, then I made some preprocessing adjustments making cutoff point to bin rare categorical variables together in a new value and finally converted all the categorical data to numeric using "pd.get_dummies".

- **First Result:** For my first model I used the next combination of layers:

```
nn = tf.keras.models.Sequential()
# First hidden layer
nn.add(tf.keras.layers.Dense(units=40, activation="relu", input_dim=43))

# Second hidden 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()
```

Unfortunately, with that model I couldn't achieve more than the 0.7368 of accuracy, that's the reason why I did some optimization to my model.

- **Optimization:** For my optimization I removed all the "rare" categorical variables that I considered non-significant, and also tried a different model with more layers and neurons:
  - # Create the Neural Network
  - model = tf.keras.models.Sequential()
  - # Add the first hidden layer (and the input layer)
  - model.add(tf.keras.layers.Dense(units=100, input_dim=40, activation='sigmoid'))
  - # Add more hidden layers... as much as you want...
  - model.add(tf.keras.layers.Dense(units=50, activation='tanh'))
  - model.add(tf.keras.layers.Dense(units=20, activation='tanh'))
  - model.add(tf.keras.layers.Dense(units=20, activation='relu'))
  - model.add(tf.keras.layers.Dense(units=20, activation='relu'))
  - # Add the output layer...
  - # In classification problems, you need one neuron per each possible label
  - # In this particular case (Binary label) we just need to have one neuron with activation function of sigmoid
  - model.add(tf.keras.layers.Dense(units=1, activation='sigmoid')) # Binary label prediction
  - #model.add(tf.keras.layers.Dense(units=20, activation='relu'))

Again, pitfully, the best accuracy that the new model obtained was 0.7339, quite similar to the old model.

- **Conclusion:** The optimization results wasn't the best, because with an accuracy of %74 it will be really difficult to have good predictions in the future to the new applicants. I'm sure that the problem is in the optimization, sadly I couldn't find the best adjustment.

```
igo  + Texto
Epoch 15/30
804/804 [==============================] - 2s 3ms/step - loss: 0.5447 - accuracy: 0.7349
Epoch 16/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5438 - accuracy: 0.7364
Epoch 17/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5446 - accuracy: 0.7353
Epoch 18/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5436 - accuracy: 0.7356
Epoch 19/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5432 - accuracy: 0.7334
Epoch 20/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5432 - accuracy: 0.7357
Epoch 21/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5435 - accuracy: 0.7350
Epoch 22/30
804/804 [==============================] - 2s 3ms/step - loss: 0.5428 - accuracy: 0.7368
Epoch 23/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5427 - accuracy: 0.7354
Epoch 24/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5427 - accuracy: 0.7358
Epoch 25/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5425 - accuracy: 0.7359
Epoch 26/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5419 - accuracy: 0.7367
Epoch 27/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5415 - accuracy: 0.7367
Epoch 28/30
804/804 [==============================] - 2s 2ms/step - loss: 0.5416 - accuracy: 0.7362
Epoch 29/30
804/804 [==============================] - 2s 3ms/step - loss: 0.5413 - accuracy: 0.7358
Epoch 30/30
804/804 [==============================] - 2s 3ms/step - loss: 0.5408 - accuracy: 0.7355
```

Image 1: Results obtained from the first model.

```
Epoch 5/20
804/804 [==============================] - 5s 6ms/step - loss: 0.5571 - accuracy: 0.7299
Epoch 6/20
804/804 [==============================] - 2s 2ms/step - loss: 0.5563 - accuracy: 0.7287
Epoch 7/20
804/804 [==============================] - 2s 3ms/step - loss: 0.5548 - accuracy: 0.7305
Epoch 8/20
804/804 [==============================] - 2s 2ms/step - loss: 0.5543 - accuracy: 0.7288
Epoch 9/20
804/804 [==============================] - 2s 3ms/step - loss: 0.5534 - accuracy: 0.7303
Epoch 10/20
804/804 [==============================] - 3s 4ms/step - loss: 0.5526 - accuracy: 0.7317
Epoch 11/20
804/804 [==============================] - 2s 3ms/step - loss: 0.5509 - accuracy: 0.7324
Epoch 12/20
804/804 [==============================] - 2s 3ms/step - loss: 0.5507 - accuracy: 0.7309
Epoch 13/20
804/804 [==============================] - 2s 3ms/step - loss: 0.5496 - accuracy: 0.7324
Epoch 14/20
804/804 [==============================] - 2s 3ms/step - loss: 0.5489 - accuracy: 0.7326
Epoch 15/20
804/804 [==============================] - 3s 3ms/step - loss: 0.5490 - accuracy: 0.7329
Epoch 16/20
804/804 [==============================] - 3s 3ms/step - loss: 0.5483 - accuracy: 0.7328
Epoch 17/20
804/804 [==============================] - 2s 3ms/step - loss: 0.5473 - accuracy: 0.7331
Epoch 18/20
804/804 [==============================] - 2s 3ms/step - loss: 0.5476 - accuracy: 0.7334
Epoch 19/20
804/804 [==============================] - 2s 2ms/step - loss: 0.5463 - accuracy: 0.7339
Epoch 20/20
804/804 [==============================] - 2s 2ms/step - loss: 0.5467 - accuracy: 0.7333
268/268 [==============================] - 1s 2ms/step - loss: 0.5532 - accuracy: 0.7293
```

Image 2: Results obtained from the model optimized