DEEP LEARNING CHALLENGE

Neural Network Model Report

Note: This information is captured in the various sections of README file within the GitHub repository.

1. **Overview of the analysis**:

Alphabet Soup (A/S), a non-profit organization, is interested in leveraging its historical funding data over 34,000 organizations they've funded over the years. The request is to use this information to build a tool that will help AS select the applicants for funding with the best chance of success in future ventures. This tool will be created based on knowledge of machine learning and neural networks.

2. **Results**:
   - Data Preprocessing-
     - What variable(s) are the target(s) for your model?

       For modeling, the target was information contained in a column called "IS_SUCCESSFUL", which contains binary (0 or 1) information on whether the money used effectively

     - What variable(s) are the features for your model?

       STATUS; ASK_AMOUNT; APPLICATION_TYPE; AFFILIATION; INCOME_AMOUNT; and SPECIAL_CONSIDERATIONS

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

       - EIN; NAME; CLASSIFICATION; USE_CASE; and ORGANIZATION

   - Compiling, Training, and Evaluating the Model-
     - How many neurons, layers, and activation functions did you select for your neural network model, and why?

       My initial model had 3 layers (35 neurons, 35 neurons, and 1 neuron) using RELU as my activation function for two layers and SIGMOID for the last, with 100 Epochs

- Were you able to achieve the target model performance?

  The initial model did **not** meet model performance, as the Accuracy was calculated at 0.7332 and the Loss calculated at 0.5586.

- What steps did you take in your attempts to increase model performance?

  The first modification was to increase the number of Epochs to 200, but this lowered the Accuracy score to 0.7297 and increased the Loss to 0.5839.

  Subsequently, for the second modification, the number of Epochs was returned to 100 and a fourth layer of 35 neurons (with RELU) was added. The Accuracy score finished 0.7300 and Loss equaling 0.5648.

  Finally, the last modification, returned the model to 3 layers (35 neurons, 35 neurons, and 1 neuron) and changed the activation functions to SIGMOID for all layers. The Accuracy score finished as 0.7306, with Loss equaling 0.5525.

  For more specific information, see Table 1 below.

3. **Summary**:

This tool was proposed to help understand if any insight could be gleaned from historical information. Each iteration (original and three modifications) did not reach the requested threshold of 75%.

While having this data available is advantageous, there might need to be discussed about whether other information A/S has can augment this model further. Benefit may also result from my detailed review of each column's information to ensure it should be included in the tool's set up.

In data preprocessing, two columns (APPLICATION_TYPE and CLASSIFICATION) were given arbitrary cut offs. This could impact the model in terms of the configuration of the data fed into the model. Additional thought on these cut off values might prove to be beneficial in future iterations of the tool.

Table 1:

Iteration	Layers and Neurons	Accuracy and Loss

Original Model	`nn_model = tf.keras.models.Sequential()`
	
	`# First hidden layer`
	`nn_model.add(tf.keras.layers.Dense(units=35, activation="relu", input_dim=41))`
	
	`# Second hidden layer`
	`nn_model.add(tf.keras.layers.Dense(units=35, activation="relu"))`
	
	`# Output layer`
	`nn_model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))`
		`268/268 - 0s - 2ms/step - accuracy: 0.7332 - loss: 0.5586`
		`Loss: 0.5585745573043823, Accuracy: 0.7331778407096863`

Modification 1	`nn_model = tf.keras.models.Sequential()`
	
	`# First hidden layer`
	`nn_model.add(tf.keras.layers.Dense(units=35, activation="relu", input_dim=41))`
	
	`# Second hidden layer`
	`nn_model.add(tf.keras.layers.Dense(units=35, activation="relu"))`
	
	`# Third hidden layer`
	`nn_model.add(tf.keras.layers.Dense(units=35, activation="relu"))`
	
	`# Output layer`
	`nn_model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))`
	
	`# Check the structure of the model`
	`nn_model.summary()`
		`268/268 - 1s - 3ms/step - accuracy: 0.7297 - loss: 0.5839`
		`Loss: 0.5838788151741028, Accuracy: 0.72967928647995`

Modification 2	`nn_model = tf.keras.models.Sequential()`
	
	`# First hidden layer`
	`nn_model.add(tf.keras.layers.Dense(units=35, activation="relu", input_dim=41))`
	
	`# Second hidden layer`
	`nn_model.add(tf.keras.layers.Dense(units=35, activation="relu"))`
	
	`# Third hidden layer`
	`nn_model.add(tf.keras.layers.Dense(units=35, activation="relu"))`
	
	`# Output layer`
	`nn_model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))`
	
	`# Check the structure of the model`
	`nn_model.summary()`
		`268/268 - 0s - 2ms/step - accuracy: 0.7300 - loss: 0.5648`
		`Loss: 0.5648401975631714, Accuracy: 0.7300291657447815`

Modification 3	`nn_model = tf.keras.models.Sequential()`
	
	`# First hidden layer`
	`nn_model.add(tf.keras.layers.Dense(units=35, activation="sigmoid", input_dim=41))`
	
	`# Second hidden layer`
	`nn_model.add(tf.keras.layers.Dense(units=35, activation="sigmoid"))`
	
	`# Output layer`
	`nn_model.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))`
	
	`# Check the structure of the model`
	`nn_model.summary()`
		`268/268 - 0s - 1ms/step - accuracy: 0.7319 - loss: 0.5540`
		`Loss: 0.5540482997894287, Accuracy: 0.7318950295448303`