Module 21 – Deep Learning

Overview

The non-profit Alphabet Soups has asked for a tool that can help to select the applicants with the best chance of success. We will use machine learning and neural networks to create a model that will help to predict if an applicant will be successful if funded. We were provided with a CSV containing more than 34,000 organizations that had received funding from Alphabet Soup. Below are the columns:

- **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

Results

Data processing

- What variables were the target: IS_SUCCESSFUL
- What variables were the features: APPLICATION_TYPE, AFFILIATION, CLASSIFICATION, USE_CASE, ORGANIZATION, STATUS, INCOME_AMT, SPECIAL_CONSIDERATIONS, ASK_AMT
- What variables were removed because they are neither a target nor a feature: EIN, NAME

Compiling, Training, and Evaluating the Model

- The model did not achieve the accuracy threshold of 75%. The best accuracy was from Optimization 3:
    - Original Model: 64.2%
    - Optimization1: 70.1%
    - Optimization 2: 46.5%
    - Optimization 3: 73.1%

Model

- Model contained 2 hidden Layers, activation function was ‘relu’. Layer 1 had 10 neuron, layer 2 had 6 neurons. The output layer used the activation function ‘sigmoid’. The model ran for 25 epochs
- The model did not achieve the accuracy threshold of 75%: model accuracy was 64.2%

```python
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
# Number of input features
input_features = X_train.shape[1]

# Define the model - deep neural net
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=10, input_dim=input_features, activation='relu'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=6, activation='relu'))

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

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                440

 dense_1 (Dense)             (None, 6)                 66

 dense_2 (Dense)             (None, 1)                 7

=================================================================
Total params: 513 (2.00 KB)
Trainable params: 513 (2.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

Optimization 1

- This optimization contained 3 hidden Layers, activation function was ‘elu’. Layer 1 had 20 neuron, layer 2 had 20 neurons, layer 3 had 20 neurons. The output layer used the activation function ‘sigmoid’. The model ran for 50 epochs
- The model did not achieve the accuracy threshold of 75%: model accuracy was 70.1%

```python
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
# Number of input features
input_features = X_train.shape[1]

# Define the model - deep neural net
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=20, input_dim=input_features, activation='elu'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=20, activation='elu'))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=20, activation='elu'))

# 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_5 (Dense)             (None, 20)                820

 dense_6 (Dense)             (None, 20)                420

 dense_7 (Dense)             (None, 20)                420

 dense_8 (Dense)             (None, 1)                 21

=================================================================
Total params: 1681 (6.57 KB)
Trainable params: 1681 (6.57 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```python
# Compile the model
nn.compile(optimizer='adamax', loss='binary_crossentropy', metrics=['accuracy'])
```

Optimization 2

- This optimization contained 4 hidden Layers, activation function was 'elu'. Layer 1 had 20 neuron, layer 2 had 20 neurons, layer 3 had 20 neurons, layer 4 had 20 neurons. The output layer used the activation function 'sigmoid'. The model ran for 100 epochs
- The model did not achieve the accuracy threshold of 75%: model accuracy dropped to 46.5%

```python
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
# Number of input features
input_features = X_train.shape[1]

# Define the model - deep neural net
nn2 = tf.keras.models.Sequential()

# First hidden layer
nn2.add(tf.keras.layers.Dense(units=20, input_dim=input_features, activation='elu'))

# Second hidden layer
nn2.add(tf.keras.layers.Dense(units=20, activation='elu'))

# Third hidden layer
nn2.add(tf.keras.layers.Dense(units=20, activation='elu'))

# Forth hidden layer
nn2.add(tf.keras.layers.Dense(units=20, activation='elu'))

# Output layer
nn2.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

# Check the structure of the model
nn2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_13 (Dense)            (None, 20)                820

 dense_14 (Dense)            (None, 20)                420

 dense_15 (Dense)            (None, 20)                420

 dense_16 (Dense)            (None, 20)                420

 dense_17 (Dense)            (None, 1)                 21

=================================================================
Total params: 2101 (8.21 KB)
Trainable params: 2101 (8.21 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```python
# Compile the model
nn2.compile(optimizer='adamax', loss='binary_crossentropy', metrics=['accuracy'])
```

Optimization 3

- For this optimization, I created a new Sequential model with hyperparameter options. The best model's hyperparameters were:

```
{'activation': 'tanh',
 'first_units': 11,
 'num_layers': 3,
 'units_0': 11,
 'units_1': 5,
 'units_2': 7,
 'units_3': 11,
 'units_4': 17,
 'tuner/epochs': 20,
 'tuner/initial_epoch': 0,
 'tuner/bracket': 0,
 'tuner/round': 0}
```

- The model did not achieve the accuracy threshold of 75%: model accuracy dropped to 73.1%

```python
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
# Create a method that creates a new Sequential model with hyperparameter options
def create_model(hp):
    nn_model = tf.keras.models.Sequential()

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

    # Allow kerastuner to decide number of neurons in first layer
    nn_model.add(tf.keras.layers.Dense(units=hp.Int('first_units',
        min_value=1,
        max_value=30,
        step=2), activation=activation, input_shape=(40,)))

    # Allow kerastuner to decide number of hidden layers and neurons in hidden layers
    for i in range(hp.Int('num_layers', 1, 6)):
        nn_model.add(tf.keras.layers.Dense(units=hp.Int('units_' + str(i),
            min_value=1,
            max_value=20,
            step=2),
            activation=activation))

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

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

    return nn_model
```

```python
# Import the kerastuner library
import keras_tuner as kt

tuner = kt.Hyperband(
    create_model,
    objective="val_accuracy",
    max_epochs=20,
    hyperband_iterations=2)
```

```python
# Run the kerastuner search for best hyperparameters
tuner.search(X_train_scaled, y_train, epochs=25, validation_data=(X_test_scaled, y_test))
```

```python
best_model = tuner.get_best_models(1)[0]
model_loss, model_accuracy = best_model.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

```
215/215 - 0s - loss: 0.5598 - accuracy: 0.7315 - 494ms/epoch - 2ms/step
Loss: 0.5598312020301819, Accuracy: 0.7314868569374084
```

Summary

The models did not achieve the required accuracy for prediction purposes. Given that Optimization 2's accuracy decreased when neurons and layers, as well as epochs were increased, we would need to explore over-fitting issues. The Random Forest model could be an option given the complexity of their data.