DEEP LEARNING

NEURAL NETWORK MODEL – REPORT

OVERVIEW

The main purpose of this analysis is to create a tool for a nonprofit organization Alphabet Soup. The binary classifier aims to predict whether the applicant will be successful in their ventures if funded by Alphabet Soup.

PROCESS

Alphabet Soup's business team provided a CSV that contains 34,000 organizations that received funding from Alphabet Soup over the years. The columns in the CSV 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.

Data Preprocessing

- Used Pandas and scikit-learn's "StandardScaler()" to preprocess the data.
- Identified the target and feature variables.
- **IS_SUCCESSFUL** is the target variable **y.** The remaining columns are features variables **X**.
- Dropped the columns **EIN** and **NAME**.
- Determined unique values for each column. Assigned a cutoff point to combine "rare" categorical variables together in a new value, "Other".
- Encoded the categorical variables. Split the preprocessed data into a features array, X, and a target array, y. Split the data into training and testing datasets.
- Scaled the training and testing features datasets by creating a StandardScaler instance, fitting it to the training data, then using the transform function.

COMPILE, TRAIN, AND EVALUATION OF THE MODEL

The main goal of the model is to achieve 75% accuracy.

- Used 3 to 4 hidden layers in different attempts.
- Attempted both 'relu' and 'tanh' activation function to check on the accuracy of the model.
- Tried different combinations of neuron units to achieve 75% accuracy of the model.
- Added/ reduced number of epochs.

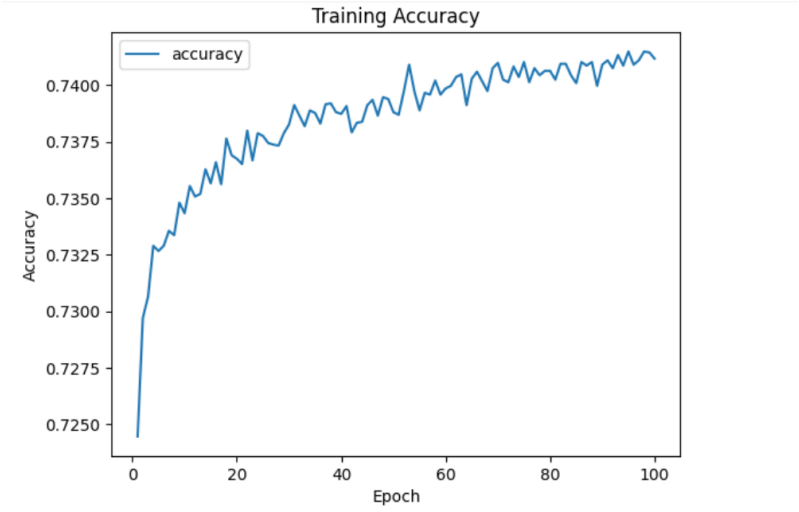
The final analysis folder has Alphabetsoup starter file, AlphabetSoupCharity_Optimization1.ipynb and AlphabetSoupCharity_Optimization_2.ipynb.

RESULTS

Alphabetsoup starter file

The Alphabetsoup starter file has 2 hidden layers with 80 and 30 units respectively. The activation function used is “relu” and epochs is 100. This model achieved an accuracy of 72.62% and loss of 55.57%.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 80)                3520

 dense_1 (Dense)             (None, 30)                2430

 dense_2 (Dense)             (None, 1)                 31

=================================================================
Total params: 5981 (23.36 KB)
Trainable params: 5981 (23.36 KB)
Non-trainable params: 0 (0.00 Byte)
```



AlphabetSoupCharity_Optimization1.ipynb file

In the AlphabetSoupCharity_Optimization.ipynb1 file tried different combinations of units and hidden layers.

Attempt 1: Hidden layers – 4, Activation function – ‘relu’.

Accuracy achieved – 72.50%, Loss - 56.31%

```
# Fourth hidden layer
nn.add(Dense(units=10, activation='relu'))

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

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               4400

 dense_1 (Dense)             (None, 50)                5050

 dense_2 (Dense)             (None, 25)                1275

 dense_3 (Dense)             (None, 10)                260

 dense_4 (Dense)             (None, 1)                 11

=================================================================
Total params: 10996 (42.95 KB)
Trainable params: 10996 (42.95 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
[20] # 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 - 1s - loss: 0.5631 - accuracy: 0.7250 - 561ms/epoch - 2ms/step
Loss: 0.5631119012832642, Accuracy: 0.7250145673751831
```

Attempt 2: Increased the number of units. Accuracy decreased and loss increased.

Hidden layers – 4, Activation function – ‘relu’.

Accuracy achieved – 72.60%, Loss - 57.53%

```
# Fourth hidden layer
nn.add(Dense(units=10, activation='relu'))

# Output layer
nn.add(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, 150)               6600

 dense_6 (Dense)             (None, 80)                12080

 dense_7 (Dense)             (None, 40)                3240

 dense_8 (Dense)             (None, 10)                410

 dense_9 (Dense)             (None, 1)                 11

=================================================================
Total params: 22341 (87.27 KB)
Trainable params: 22341 (87.27 KB)
Non-trainable params: 0 (0.00 Byte)
```

0s

```
# 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 - 1s - loss: 0.5753 - accuracy: 0.7261 - 540ms/epoch - 2ms/step
Loss: 0.5753489136695862, Accuracy: 0.726064145565033
```

Attempt 3: Decreased the number of units and used a different activation function. Accuracy increased slightly and loss decreased.

Hidden layers – 3, Activation function – ‘tanh’.

Accuracy achieved – 72.61%, Loss - 55.61%

```
nn.add(Dense(units=10, activation='tanh'))

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

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

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 80)                3520

 dense_11 (Dense)            (None, 40)                3240

 dense_12 (Dense)            (None, 10)                410

 dense_13 (Dense)            (None, 1)                 11

=================================================================
Total params: 7181 (28.05 KB)
Trainable params: 7181 (28.05 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

1s

```
# 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 - 1s - loss: 0.5561 - accuracy: 0.7262 - 714ms/epoch - 3ms/step
Loss: 0.5561491250991821, Accuracy: 0.7261807322502136
```

Attempt 4: Decreased the number of units to 40 to check if there is an increase in accuracy. Used ‘relu’ activation function. Accuracy increased slightly and loss decreased.

Hidden layers – 3, Activation function – ‘relu’.

Accuracy achieved – 72.55%, Loss - 55.39%

```
[30]
# Third hidden layer
nn.add(Dense(units=5, activation='relu'))

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

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

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_14 (Dense)            (None, 40)                1760

 dense_15 (Dense)            (None, 20)                820

 dense_16 (Dense)            (None, 5)                 105

 dense_17 (Dense)            (None, 1)                 6

=================================================================
Total params: 2691 (10.51 KB)
Trainable params: 2691 (10.51 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# 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: 0.5539 - accuracy: 0.7256 - 492ms/epoch - 2ms/step
Loss: 0.5539494156837463, Accuracy: 0.7255976796150208
```

Attempt 5: Since the 'tanh' activation function achieved 72.6% accuracy used the same unit distribution with 'relu' activation function. Increased the number of units back to 80,40,10 to check if there is an increase in accuracy. Accuracy increased slightly and loss decreased.

Hidden layers – 3, Activation function – 'relu'.

Accuracy achieved – 72.46%, Loss - 55.65%

```
# Third hidden layer
nn.add(Dense(units=10, activation='relu'))

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

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

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 80)                3520

 dense_19 (Dense)            (None, 40)                3240

 dense_20 (Dense)            (None, 10)                410

 dense_21 (Dense)            (None, 1)                 11

=================================================================
Total params: 7181 (28.05 KB)
Trainable params: 7181 (28.05 KB)
Non-trainable params: 0 (0.00 Byte)
```

1s

```
# 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 - 1s - loss: 0.5566 - accuracy: 0.7247 - 503ms/epoch - 2ms/step
Loss: 0.5565716624259949, Accuracy: 0.7246647477149963
```

Analysis:

The above optimization techniques achieved 72% accuracy. This indicates that the model predicted 72% of instances in the dataset correctly (72% of instances are classified correctly). 55% of loss value indicate larger discrepancies between the predictions and the true values.

AlphabetSoupCharity_Optimization_2.ipynb file

Included NAME column back in the pandas dataframe to check if the accuracy is increasing.

The number of units remained the same. Used 'relu' activation function. Accuracy increased drastically and loss decreased.

Hidden layers – 3, Activation function – 'relu'. Epochs- 50

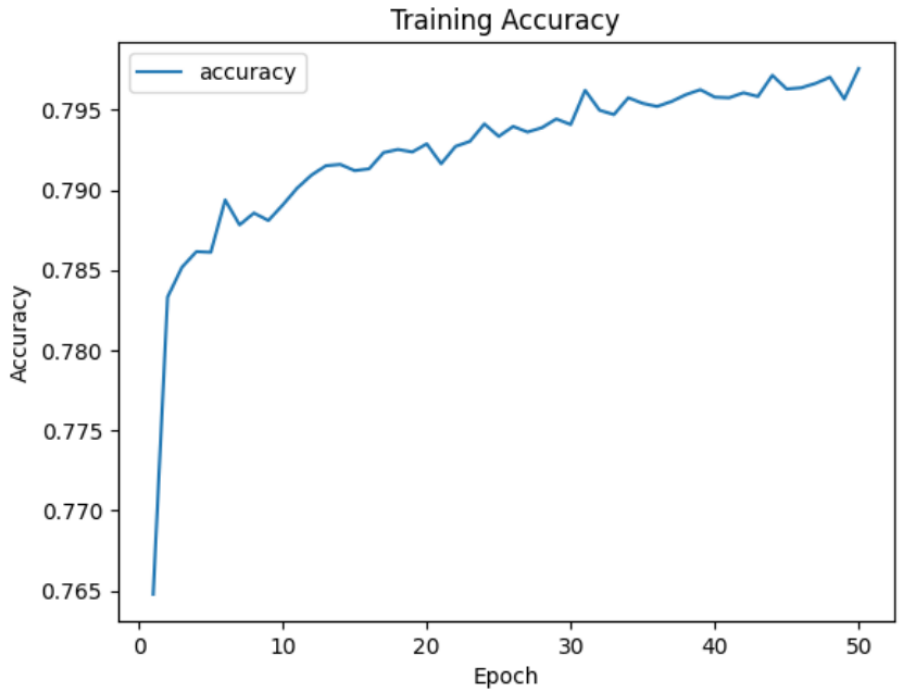
Accuracy achieved – 78.01%, Loss – 45.75%

```
nn.add(Dense(units=10, activation='relu'))


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

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

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 80)                19920

 dense_8 (Dense)             (None, 40)                3240

 dense_9 (Dense)             (None, 10)                410

 dense_10 (Dense)            (None, 1)                 11
```

```
# 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 - 1s - loss: 0.4574 - accuracy: 0.7802 - 712ms/epoch - 3ms/step
Loss: 0.457367867231369, Accuracy: 0.7801749110221863
```



SUMMARY:

- On many attempts of running the model with ‘relu’ and ‘tanh’ activation function with minimum of 3 hidden layers and trying different combinations of neurons the model did not achieve 75% of accuracy.

- In **AlphabetSoupCharity_Optimization1.ipynb file** the model achieved 72% of accuracy and 55% of loss.
- An accuracy of 72% suggests that the model correctly predicts the outcome (success or failure) for approximately 72% of the applicants.
- A loss of 55% is extraordinarily high and indicates significant errors in the model's predictions. One reason could be Overfitting.

- However, in **AlphabetSoupCharity_Optimization_2.ipynb file** the Name column is included back in the pandas dataframe and the model achieved 78% of accuracy and 45% of loss.

- Overall, this model did not meet the required accuracy. There may be opportunities for improvement in the model's optimization techniques. I would suggest trying a logistic regression, decision tree or random forest model to increase the accuracy of the model. Logistic regression is computationally efficient and can handle large datasets with many features. Here the goal is to predict the probability of a binary outcome (success or failure) based on one or more predictor variables and Logistic regression is a well-established method for binary classification tasks. Random forest is also a powerful method in predicting applicant success for Alphabet Soup and is less prone to overfitting. Its robust performance makes it a suitable alternative to neural networks.