

# Deep Learning Project

## Charity Funding Predictor Report

Birmingham Data Camp

Meng Tong

## 1 Introduction

In this project, on behalf of the non-profit foundation Alphabet Soup, we use deep learning and neural networks to create an algorithm to predict if the applicants for funding will be successful. Using a rich data in a CSV file containing more than 34,000 organizations who received funding from Alphabet Soup over the past years. After choosing a cutoff value and create a list of application types for binning variables, we applying the neural network with two hidden layers to compile the model. Our estimation accuracy shows a moderated performance of our neural networks model.

## 2 Data

### 2.1 Data collection and preparation

Our data is provided by Alphabet Soup business team with number of columns that capture various features of applicants. Table 1 below shows all variables we have from raw data with its name and definitions. Variables include identity, affiliation, government organization classification,

Table 1: Alphabet Soup Applicant Data

Alphabet Soup Applicant Data	
EIN	Identification columns
NAME	Identification columns
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 consideration for application
ASK_AMT	Funding amount requested
IS_SUCCESSFUL	Was the money used effectively

We create a python file using Jupyter notebook with the name "AlphabetSoupCharity.ipynb" (see Github submission link). After introducing the raw CSV data file into the ipynb file, we obtain the data with brief summary captured in Table 2 shows below. We observe that the raw data includes text, numerical and mixed data type for our analysis.

Table 2: Alphabet Soup Data Summary

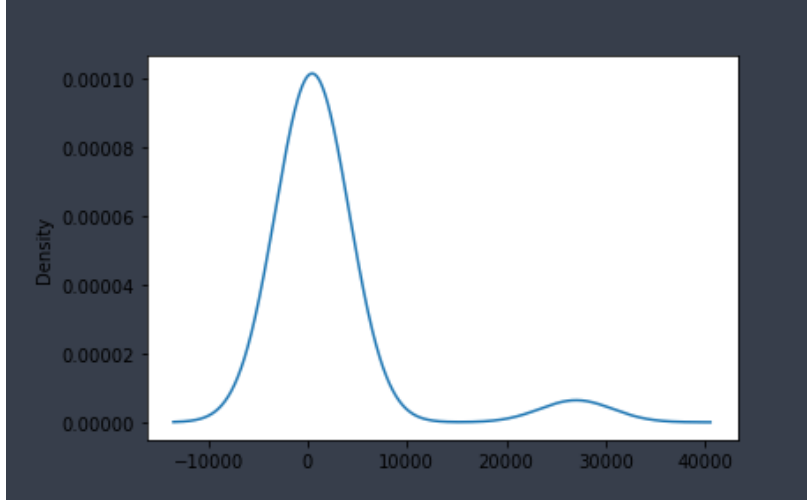
	EIN	NAME	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	STATUS	INC
0	10520599	BLUE KNIGHTS MOTORCYCLE CLUB	T10	Independent	C1000	ProductDev	Association	1	0
1	10531628	AMERICAN CHESAPEAKE CLUB CHARITABLE TR	T3	Independent	C2000	Preservation	Co-operative	1	1-99
2	10547893	ST CLOUD PROFESSIONAL FIREFIGHTERS	T5	CompanySponsored	C3000	ProductDev	Association	1	0
3	10553066	SOUTHSIDE ATHLETIC ASSOCIATION	T3	CompanySponsored	C2000	Preservation	Trust	1	100
4	10556103	GENETIC RESEARCH INSTITUTE OF THE DESERT	T3	Independent	C1000	Heathcare	Trust	1	100

## 2.2 Data processing

We focus on the numerical values that offering efficient information for our neural networks analysis and drop two identity-related variable "EIN" and "NAME" the dataset and convert the rest of non-numerical values into dummy values. We use all variables apart from the identity to construct our neural network model. Then to simplify our computation, we summarize the number of application type and drop those types with number of appearance less than 500 times <sup>1</sup>). In other words, we drop those outliers from our raw data (See [Figure A1](#) in the appendix). To simplify our model by reducing the number of variables, we further use classification value counts for binning to drop those outliers with classification counts equals to 1. Finally, we convert the categorical data to numeric value and split our cleaned data into training data section and test data section. We standardized the data using "StandardScaler" before evaluating the model.

Figure 1: The density distribution of application types

<sup>1</sup>using code `application_types_to_replace = list(application_counts[application_counts < 500].index)`



### 3 Quantitative analysis

#### 3.1 Model construction

We construct three neural network models by setting different number of nodes in each hidden layer. After data cleaning, the total number of columns representing the number of variables we use for our analysis is 76 including the target variable. Therefore, the number of dimensions we have is 75. In the first model, we construct a hierarchy structure of number of neurons in each hidden layer equals to 10,20,40 respectively. In the second model, we prefers to have identical number of neurons in each hidden layer that is smaller than the number of dimensions. Finally, we setup the number of neurons in each hidden layer to be identical with the number of dimensions in the last model. The random state (Pseudo-random number) in our model is set equals to 152<sup>2</sup>. Using *keras* as the open-source library of neural networks, the activation method in the first and second hidden layer are 'relu'(Rectified Linear Unit) and we applies a non-linear function of the weighted sum of inputs 'sigmoid' which defines the final output value is between 0 and 1 as the activation method in the final hidden layer. Figure 2 below shows our first model sequential information defined.

Figure 2: The density distribution of application types

---

<sup>2</sup>We also tried to set the random state less than the number of dimensions. However, the accuracy of prediction is less effective than setting larger random state

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	760
dense_1 (Dense)	(None, 20)	220
dense_2 (Dense)	(None, 1)	21

```

=====
Total params: 1,001
Trainable params: 1,001
Non-trainable params: 0
=====

```

### 3.2 Neural Networks accuracy results

We trained the model by setting the validation splitting data into 85/15 in between training and testing data while utilizing Adam optimization algorithm to setup the gradient descent. We made three attempts aiming to achieve higher than 75% accuracy target. However, neither of the model achieve the accuracy over 75%. The hierarchy number of neurons model obtains the highest accuracy 0.7353935837745667 or approximately 73.5%<sup>3</sup>.

---

<sup>3</sup>the accuracy of other two models are 0.7323614954948425 (73.2%) and 0.7351603507995605(73.5%) respectively

Figure 3: The first hierarchy number of neurons model Hidden Layer setting

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	760
dense_1 (Dense)	(None, 20)	220
dense_2 (Dense)	(None, 1)	21

```

=====
Total params: 1,001
Trainable params: 1,001
Non-trainable params: 0
=====

```

Figure 4: The first hierarchy number of neurons model training

```
Epoch 1/100
684/684 [=====] - 1s 838us/step - loss: 0.6087 - accuracy: 0.6905 - val_loss: 0.5581 - val_accuracy: 0.7357
Epoch 2/100
684/684 [=====] - 0s 647us/step - loss: 0.5665 - accuracy: 0.7208 - val_loss: 0.5515 - val_accuracy: 0.7346
Epoch 3/100
684/684 [=====] - 0s 647us/step - loss: 0.5596 - accuracy: 0.7238 - val_loss: 0.5456 - val_accuracy: 0.7362
Epoch 4/100
684/684 [=====] - 0s 659us/step - loss: 0.5567 - accuracy: 0.7260 - val_loss: 0.5428 - val_accuracy: 0.7354
Epoch 5/100
684/684 [=====] - 0s 654us/step - loss: 0.5541 - accuracy: 0.7258 - val_loss: 0.5414 - val_accuracy: 0.7367
Epoch 6/100
684/684 [=====] - 0s 682us/step - loss: 0.5531 - accuracy: 0.7266 - val_loss: 0.5402 - val_accuracy: 0.7328
Epoch 7/100
684/684 [=====] - 0s 648us/step - loss: 0.5525 - accuracy: 0.7270 - val_loss: 0.5391 - val_accuracy: 0.7310
Epoch 8/100
684/684 [=====] - 0s 659us/step - loss: 0.5516 - accuracy: 0.7273 - val_loss: 0.5423 - val_accuracy: 0.7326
Epoch 9/100
684/684 [=====] - 0s 646us/step - loss: 0.5505 - accuracy: 0.7293 - val_loss: 0.5391 - val_accuracy: 0.7380
Epoch 10/100
684/684 [=====] - 0s 647us/step - loss: 0.5499 - accuracy: 0.7291 - val_loss: 0.5379 - val_accuracy: 0.7403
Epoch 11/100
684/684 [=====] - 0s 652us/step - loss: 0.5493 - accuracy: 0.7287 - val_loss: 0.5398 - val_accuracy: 0.7359
Epoch 12/100
```

Figure 5: The first hierarchy number of neurons model accuracy results

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn_model.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

268/268 - 0s - loss: 0.5526 - accuracy: 0.7354 - 120ms/epoch - 448us/step
Loss: 0.5525544285774231, Accuracy: 0.7353935837745667
```

Figure 6: The second NNs model number of neurons model Hidden Layer setting

```
Model: "sequential_1"

Layer (type)                 Output Shape              Param #
=====
dense_3 (Dense)              (None, 32)                2432
dense_4 (Dense)              (None, 32)                1056
dense_5 (Dense)              (None, 32)                1056
dense_6 (Dense)              (None, 1)                 33
=====
Total params: 4,577
Trainable params: 4,577
Non-trainable params: 0
```

Figure 7: The second NNs model accuracy results

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn_model2.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

268/268 - 0s - loss: 0.5589 - accuracy: 0.7324 - 195ms/epoch - 728us/step
Loss: 0.5588911175727844, Accuracy: 0.7323614954948425
```

Figure 8: The third NNs model number of neurons model Hidden Layer setting

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
dense_7 (Dense)	(None, 75)	5700
dense_8 (Dense)	(None, 75)	5700
dense_9 (Dense)	(None, 75)	5700
dense_10 (Dense)	(None, 1)	76

=====

Total params: 17,176  
Trainable params: 17,176  
Non-trainable params: 0

=====

Figure 9: The third NNs model accuracy results

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn_model3.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

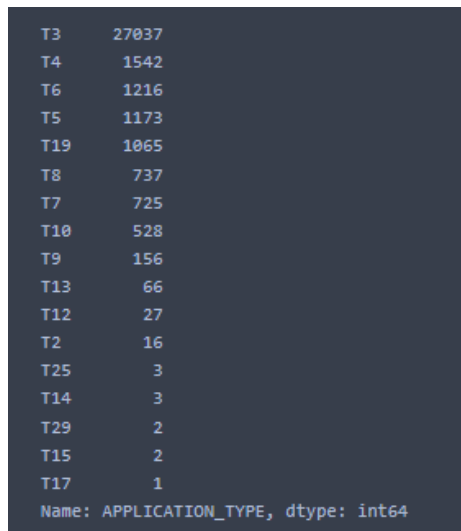
268/268 - 0s - loss: 0.5849 - accuracy: 0.7352 - 131ms/epoch - 489us/step
Loss: 0.5849457383155823, Accuracy: 0.7351603507995605
```

## 4 Conclusions

In conclusion, the hierarchy number of neurons obtain the highest accuracy among all three models we attempted. Our neuron network models obtain a moderate results with accuracy approximately equals to 73% although it is need to be further improved to achieve target accuracy value 75% in the future.

# Appendices

## A Figures



T3	27037
T4	1542
T6	1216
T5	1173
T19	1065
T8	737
T7	725
T10	528
T9	156
T13	66
T12	27
T2	16
T25	3
T14	3
T29	2
T15	2
T17	1

Name: APPLICATION\_TYPE, dtype: int64

Figure A1: The Uniqueness of application types