Neural Network Model: Charity Funding Predictor

1. Overview:

The non-profit foundation Alphabet Soup wants to create an algorithm that can predict whether applicants will be successful with received funding. With knowledge of machine learning and neural networks, we must use dataset features to create a binary classifier that has the capability of predicting whether applicants will be successful if funded by Alphabet Soup. Over the years, more than 34,000 organizations have received funding from Alphabet Soup.

2. Results:

- Data Preprocessing

Because of the giant dataset, we start by processing the data and removing irrelevant information. After dropping EIN and NAME, the rest of the column names are considered useful features for the model and NAME was later added back into the second test for the purpose of binning.

```
# Drop the non-beneficial ID columns, 'EIN' and 'NAME'.
application_df = application_df.drop(columns = ['EIN', 'NAME'])
application_df
```

	APPLICATION_TYPE	AFFILIATION	CLASSIFICATION	USE_CASE	ORGANIZATION	STATUS	INCOME_AMT	SPECIAL
0	T10	Independent	C1000	ProductDev	Association	1	0	
1	T3	Independent	C2000	Preservation	Co-operative	1	1-9999	
2	T5	CompanySponsored	C3000	ProductDev	Association	1	0	
3	T3	CompanySponsored	C2000	Preservation	Trust	1	10000-24999	
4	T3	Independent	C1000	Heathcare	Trust	1	100000-499999	
...	...	...	...	...	...	...	...	
34294	T4	Independent	C1000	ProductDev	Association	1	0	
34295	T4	CompanySponsored	C3000	ProductDev	Association	1	0	
34296	T3	CompanySponsored	C2000	Preservation	Association	1	0	
34297	T5	Independent	C3000	ProductDev	Association	1	0	
34298	T3	Independent	C1000	Preservation	Co-operative	1	1M-5M	

34299 rows × 10 columns

We then look at the unique values for each of the columns by using several data points as a cutoff to bin “rare” variables together and value of “Other” for unique values. The data was then split into two categories, training, and testing sets. The target variable for the model is labeled “IS_SUCCESSFUL” which has either a vale of 1 for yes or value of 0 for no.

Categorical variables were encoded by get_dummies() after checking if the binning was successful.

```
# Choose a cutoff value and create a list of application types to be replaced
# use the variable name `application_types_to_replace`

application_types_to_replace = list(atv_counts.iloc[8:].index)
cutoff_value = 528

# Replace in dataframe
for app in application_types_to_replace:
    application_df['APPLICATION_TYPE'] = application_df['APPLICATION_TYPE'].replace(app,"Other")

# Check to make sure binning was successful
application_df['APPLICATION_TYPE'].value_counts()
```

```
# Convert categorical data to numeric with `pd.get_dummies`
numeric_dummies = pd.get_dummies(application_df)
```

```
# Split our preprocessed data into our features and target arrays
X = numeric_dummies.drop(['IS_SUCCESSFUL'], axis=1)
y = numeric_dummies['IS_SUCCESSFUL']

# Split the preprocessed data into a training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=58)
```

```
# Create a StandardScaler instances
scaler = StandardScaler()

# Fit the StandardScaler
X_scaler = scaler.fit(X_train)

# Scale the data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

- Compiling, Training, and Evaluating the Model

There was a total of three layers for each model after applying Neural Networks. The number of hidden nodes were determined by the number of input features which was two.

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

 dense_1 (Dense)             (None, 5)                 55

 dense_2 (Dense)             (None, 1)                 6

=================================================================
Total params: 501
Trainable params: 501
Non-trainable params: 0
_________________________________________________________________
```

With my optimization model, I was able to achieve target performance with four layers and four hidden layers.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_4 (Dense)              (None, 10)                750

 dense_5 (Dense)              (None, 8)                 88

 dense_6 (Dense)              (None, 6)                 54

 dense_7 (Dense)              (None, 1)                 7

=================================================================
Total params: 899
Trainable params: 899
Non-trainable params: 0
_________________________________________________________________
```

3. Summary:

In conclusion, my first deep learning model that I developed was unable to achieve accuracy higher than 73%. With the optimization model I was able to achieve accuracy higher then 75%. Furthermore, my recommendation to solve the classification problem would be to try a different type of algorithm testing. Much like the previous module I learned, we used random forest to provide valuable insights to importance of different features. Random forest can help by identifying the key predictor columns, allowing us to potentially improve the accuracy.

Another method I could recommend is adding more dataset columns useful and even going back to general basics of simple pandas for identifying more important key features. As a Data Analyst, it is with great importance to experiment with as many steps as possible to fine tune the data as much as necessary to increase a model's overall importance.