Evaluating Marketing Campaigns of Banking Using Artificial Neural Networks(ANN)

Final Project

Claire Hu

April 21, 2019

Outline

        - Optimal Model:12, 8, 4, 1
        - Additional testing: 8, 6, 4,1
    - Change Hidden Layers
        - 3 layers
        - 2 layers
    - Change Optimazer
        - relu
        - adam
    - Reference(APA Format)

Objective

The goal is to predict whether clients will deposit (“yes”) money or not (“no”) in Portuguese bank in order to improve future marketing campaigns for the bank. I will use Artificial Neural Networks classifier and data visualization to show the perdition result. The classification goal is to predict if a client will subscribe a term deposit (variable y).

The Data

The data set here is from UCI machine learning repository. It is derived from the direct marketing campaigns of a Portuguese banking institution. The data set has 17 total input, which contains 12 categorical variables and 7 numerical variables as its features, a binary category as its response variable, and 4522 number of rows.

The following categories of information are included in the data set:

- Consumer data: Age, sex, job, marital status, education, loan status, and so on.
- Campaign activities: When and how to contact, times to contact, outcome of previous campaigns.
- Predictor y: A 'yes' for the bank if there is a term deposit subscribed, otherwise, a 'no'.

Data source link: https://archive.ics.uci.edu/ml/datasets/bank+marketing

Final ANN Model

1. Pre-Processing Dataset

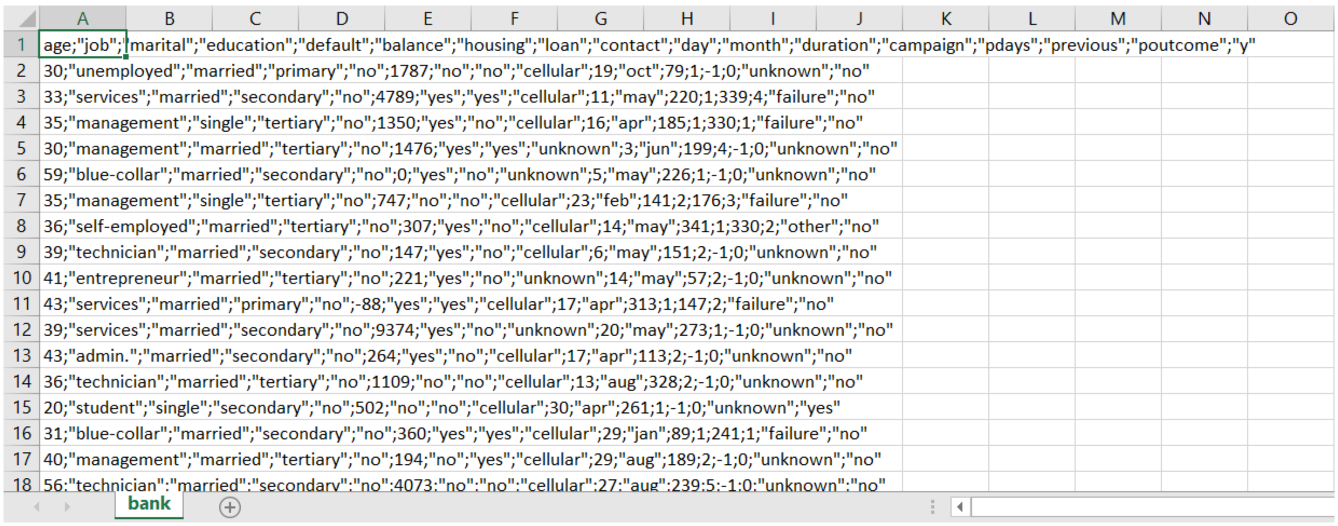
Library

I will also common on each line for all the lines.

```python
import numpy as np # linear algebra
import pandas as pd # data processing
import seaborn as sns # data visualization
import keras
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.callbacks import EarlyStopping
from keras.utils import to_categorical
from matplotlib import pyplot as plt
from keras.metrics import categorical_accuracy
from sklearn.preprocessing import MinMaxScaler # scale data
from sklearn.model_selection import GridSearchCV # Grid Search
```

Raw data

First, understand the raw data. The raw data file has all the information in one column.



```
age;"job";"marital";"education";"default";"balance";"housing";"loan";"contact";"day";"month";"duration";"campaign";"pdays";"previous";"poutcome";"y"
30;"unemployed";"married";"primary";"no";1787;"no";"no";"cellular";19;"oct";79;1;-1;0;"unknown";"no"
33;"services";"married";"secondary";"no";4789;"yes";"yes";"cellular";11;"may";220;1;339;4;"failure";"no"
35;"management";"single";"tertiary";"no";1350;"yes";"no";"cellular";16;"apr";185;1;330;1;"failure";"no"
30;"management";"married";"tertiary";"no";1476;"yes";"yes";"unknown";3;"jun";199;4;-1;0;"unknown";"no"
59;"blue-collar";"married";"secondary";"no";0;"yes";"no";"unknown";5;"may";226;1;-1;0;"unknown";"no"
35;"management";"single";"tertiary";"no";747;"no";"no";"cellular";23;"feb";141;2;176;3;"failure";"no"
36;"self-employed";"married";"tertiary";"no";307;"yes";"no";"cellular";14;"may";341;1;330;2;"other";"no"
39;"technician";"married";"secondary";"no";147;"yes";"no";"cellular";6;"may";151;2;-1;0;"unknown";"no"
41;"entrepreneur";"married";"tertiary";"no";221;"yes";"no";"unknown";14;"may";57;2;-1;0;"unknown";"no"
43;"services";"married";"primary";"no";-88;"yes";"yes";"cellular";17;"apr";313;1;147;2;"failure";"no"
39;"services";"married";"secondary";"no";9374;"yes";"no";"unknown";20;"may";273;1;-1;0;"unknown";"no"
43;"admin.";"married";"secondary";"no";264;"yes";"no";"cellular";17;"apr";113;2;-1;0;"unknown";"no"
36;"technician";"married";"tertiary";"no";1109;"no";"no";"cellular";13;"aug";328;2;-1;0;"unknown";"no"
20;"student";"single";"secondary";"no";502;"no";"no";"cellular";30;"apr";261;1;-1;0;"unknown";"yes"
31;"blue-collar";"married";"secondary";"no";360;"yes";"yes";"cellular";29;"jan";89;1;241;1;"failure";"no"
40;"management";"married";"tertiary";"no";194;"no";"yes";"cellular";29;"aug";189;2;-1;0;"unknown";"no"
56;"technician";"married";"secondary";"no";4073;"no";"no";"cellular";27;"aug";239;5;-1;0;"unknown";"no"
```

```python
import pandas as pd
data = pd.read_csv(r"bank.csv", sep=';')
```

I am using `panda.read_csv` function to read my csv file and using `sep=';'` to separate the data.

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 17 columns):
age          4521 non-null int64
job          4521 non-null object
marital      4521 non-null object
education    4521 non-null object
default      4521 non-null object
balance      4521 non-null int64
housing      4521 non-null object
loan         4521 non-null object
contact      4521 non-null object
day          4521 non-null int64
month        4521 non-null object
duration     4521 non-null int64
campaign     4521 non-null int64
pdays        4521 non-null int64
previous     4521 non-null int64
poutcome     4521 non-null object
y            4521 non-null object
dtypes: int64(7), object(10)
memory usage: 600.5+ KB
```

Defining Variables (x) and Predictor(y)

```
X = data.iloc[:,0:16]
y = data.iloc[:,16].values.tolist()
```

X: Using all rows and column from index 2 to the last one.

y: Using all rows and only the last column y, and `values.tolist()` change data frame to list.

Encoding: Dummy variable X

```
import pandas as pd
X = pd.get_dummies(X)
```

Using `pd.get_dummies(x)` to convert categorical variable into dummy/indicator variables. From the dataset, the variables “job”, “marital”, “education”, “housing”, “loan”, “contact” are categorical event, convert to dummy variables.

Predictor Y - Binary answer

```
y=[1 if x =='yes'
   else 0 for x in y]
```

Using a simple for loop, if clients deposit money, the answer is “‘yes”, then output is 1;

if clients does not deposit money, the answer is “no”, then output is 0.

Training Data and Testing Data

a. Splitting Dataset

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test  =train_test_split(X,y,test_size = 0.2)
```

Now let's split our data into training and testing datasets. The proportion of training data and testing data is 80:20.

b. Training data

- The train x has 3616 row, float 64. `print(X_train)` to see the snapshot of the X_train
- The train y has has 3616 row, a list of integer. `print(Y_train)` to see the snapshot of the Y_train

> In [7]: print(X_train)
>
> [[0.26470588 0.0444692 0.83333333 ... 0. 0. 1.] [0.54411765 0.08926055 0.53333333 ... 0. 0. 1.] [0.36764706 0.04432155 0.26666667 ... 0. 0. 1.] ... [0.16176471 0.04767721 0.2 ... 0. 0. 1.] [0.32352941 0.04103301 0.06666667 ... 0. 0. 1.] [0.27941176 0.10265634 0.63333333 ... 0. 0. 1.]]

> print(Y_train) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,....]

c. Testing Data

- The test x has 905 rows, float 64. `print(X_test)`
- The test has has 905 rows, a list of integer. `print(Y_test)`

> In [9]: print(X_test) [[0.64705882 0.05083153 0.8 ... 0. 0. 1.] [0.23529412 0.10507242 0.6 ... 0. 0. 1.] [0.36764706 0.04311352 0.3 ... 0. 0. 1.] ... [0.23529412 0.05511335 0.53333333 ... 0. 0. 1.] [0.11764706 0.0444692 0.23333333 ... 0. 0. 1.] [0.44117647 0.05942202 0.53333333 ... 0. 0. 1.]]

> In [10]: print(Y_test) [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...]

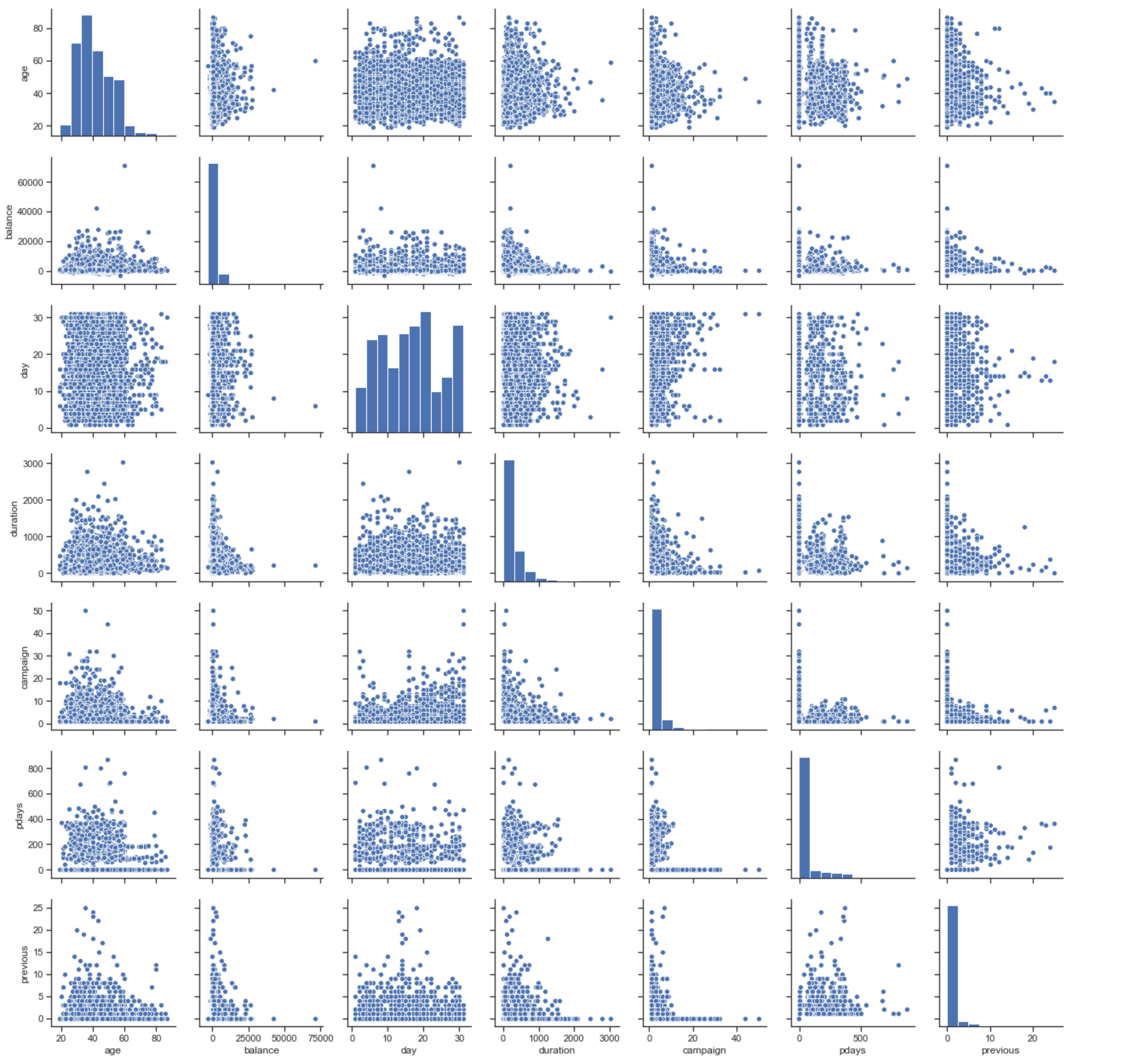
d. Scaling

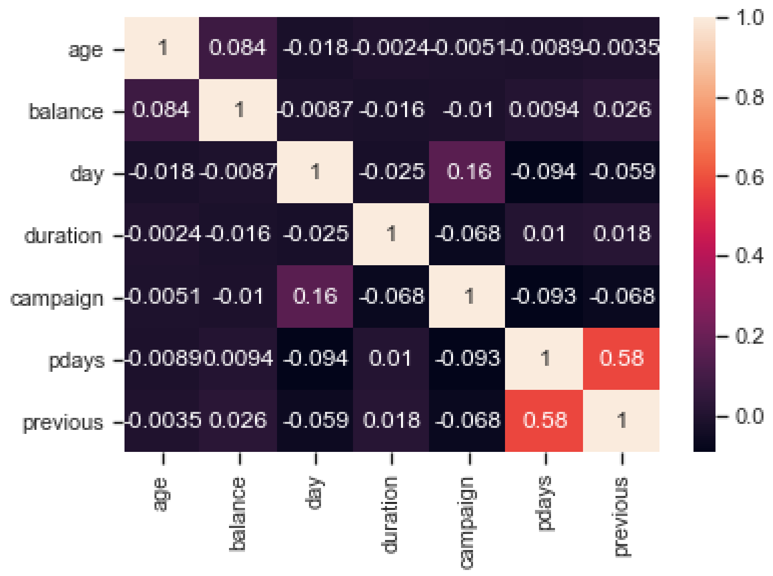
Now let's apply features scaling. Scaling ensures that just because some features are big, the model won't lead to using them as a main predictor.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

X_train.shape, len(Y_train), X_test.shape, len(Y_test)
```

e. Variables Correlation & Data Visualization

```
import seaborn as sns
sns.set(style="ticks")
sns.pairplot(data, palette="Set1")
plt.show()
sns.heatmap(data.corr(), annot=True)
```

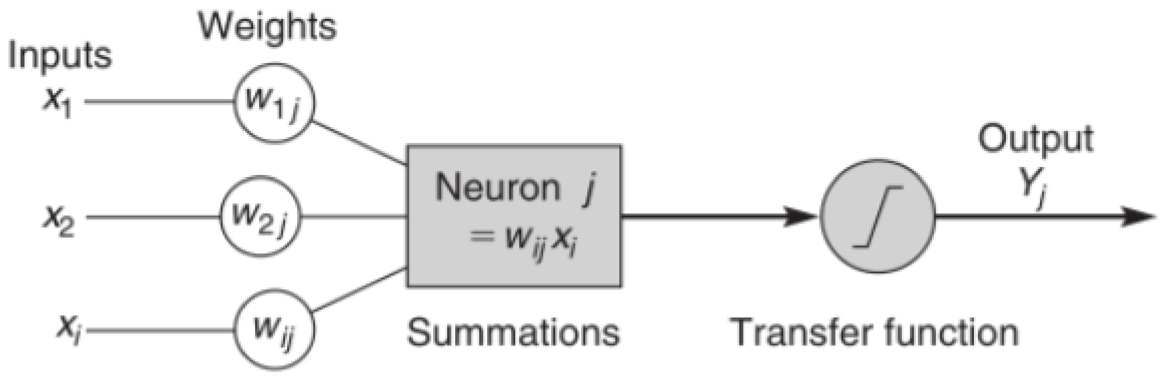

age
balance
day
duration
campaign
pdays
previous


age
balance
day
duration
campaign
pdays
previous
1 0.084 -0.018 -0.0024 -0.0051 -0.0089 -0.0035
0.084 1 -0.0087 -0.016 -0.01 0.0094 0.026
-0.018 -0.0087 1 -0.025 0.16 -0.094 -0.059
-0.0024 -0.016 -0.025 1 -0.068 0.01 0.018
-0.0051 -0.01 0.16 -0.068 1 -0.093 -0.068
-0.0089 0.0094 -0.094 0.01 -0.093 1 0.58
-0.0035 0.026 -0.059 0.018 -0.068 0.58 1
1.0
0.8
0.6
0.4
0.2
0.0

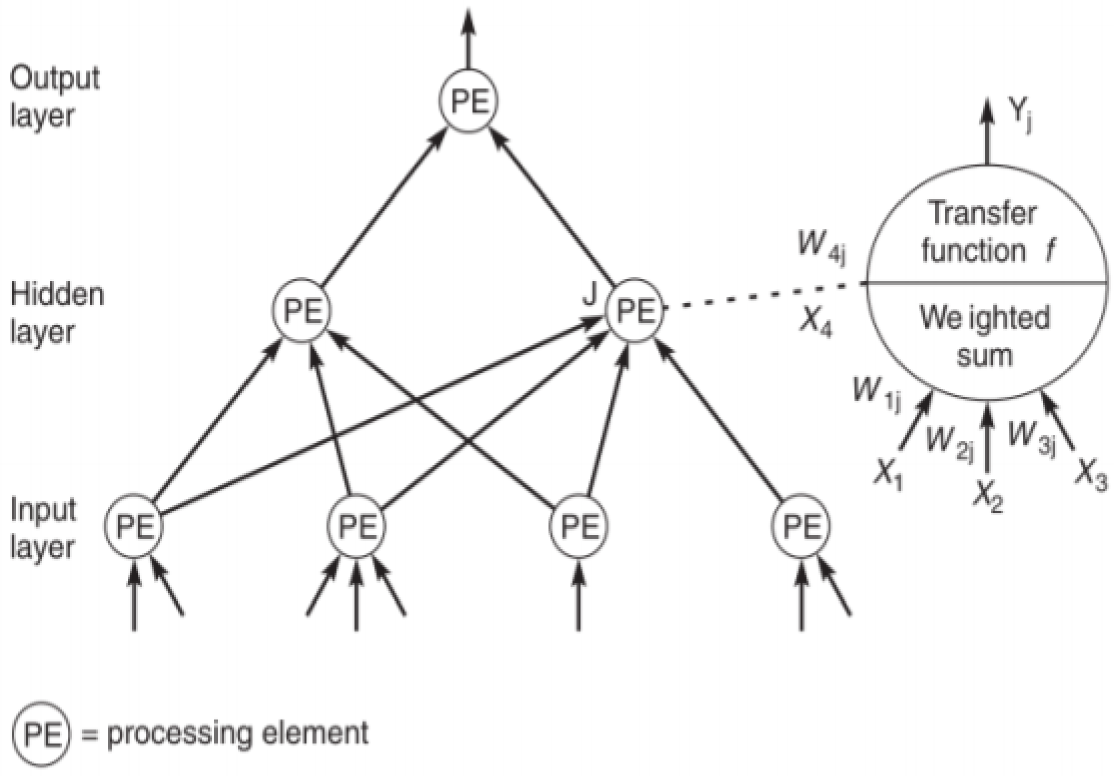
2. Preparing ANN

- What is ANN?

An artificial neural networks ANN model emulates a biological neural network. Neural computing actually uses a very limited set of concepts from biological neural systems. It is more of an analogy to the human brain than an accurate model of it. Neural concepts are usually implemented as software simulations of the massively parallel processes that involve processing elements (also called artificial neurons) interconnected in network architecture. The artificial neuron receives inputs analogous to the electrochemical impulses the dendrites of biological neurons receive from other neurons. The output of the artificial neuron corresponds to signals sent out from a biological neuron over its axon. These artificial signals can be changed by weights in a manner similar to the physical changes that occur in the synapses as shown below:



An artificial neural network (ANN) is a computational model that attempts to account for the parallel nature of the human brain. An (ANN) is a network of highly interconnecting processing elements (neurons) operating in parallel. These elements are inspired by biological nervous systems. As in nature, the connections between elements largely determine the network function. A subgroup of processing element is called a layer in the network. The first layer is the input layer and the last layer is the output layer. Between the input and output layer, there may be additional layer(s) of units, called hidden layer(s). Neural network can be train to perform a particular function by adjusting the values of the connections (weights) between elements.



- What is ANN Classification?

ANN Classification is an example of Supervised Learning. Known class labels help indicate whether the system is performing correctly or not. This information can be used to indicate a desired response, validate the accuracy of the system, or be used to help the system learn to behave correctly. The known class labels can be thought of as supervising the learning process; the term is not meant to imply that you have some sort of interventionist role.

3. Building the Layers

- Let's build the Layers. We can play around and change number of units but if we are not sure what number to initialize with then simply initialize the units of all layers except the last one.
- the optimal number of hidden nodes in the first hidden layer is: **sqrt[(m+2)N] + 2sqrt[N/(m+2)]** and in the second hidden layer, the optimal number of hidden nodes is: **m*sqrt[N/(m+2)]**, where **N = number of inputs**, and **m = number of outputs**. In this case, the **n= 17, m =2, the estimated hidden nodes should be approximately 8 for the the first layer, the second layers is 4.**
- Result

  test_loss: 0.3060592791647007 test_acc: 0.8961325966850828

  The accuracy is 0.8961 with time required running time 0:00:45.845316, which is the optimal value among all the experimentations(I will list all my experimentations later in the paper.

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(12, activation = 'relu', input_shape=(51,)))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(4, activation = 'relu'))
model.add(Dense(1, activation='sigmoid'))
```

My results were improved by setting units = 8 for the first layer and decreasing the units in the hidden layers. Also we have to provide input dimension for the first layer only. `activation ='relu'` refers to rectified linear unit and `sigmoid` refers to sigmoid activation function. With the help of sigmoid activation function, we can get the probabilities of the classification which might be beneficial in some cases to conduct further research.

Fitting Model

- Layers
  - **Input Layer**: Input variables, sometimes called the visible layer.
  - **Hidden Layers**: Layers of nodes between the input and output layers. There may be one or more of these layers.
  - **Output Layer**: A layer of nodes that produce the output variables. Here, we use ANN classifier, then it has a single node. unit = 1.

Compiling ANN

```
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Compiling classifier. Using adam optimizer. Using binary_crossentropy for accuracy function since classification is binary, i.e. only two classes 'Yes' or 'No'.

4. Fitting the data

Use Grid Search to Find Parameters

I use a grid search to evaluate different hyper parameters for neural network model and report on the combination that provides more accurate predictions.

Grid search is the process of performing hyper parameter tuning in order to determine the optimal values for a given model. This is significant as the performance of the entire model is based on the hyper parameter values specified.

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from keras.metrics import categorical_accuracy

def build_model(optimizer):
    model = Sequential()
    model.add(Dense(12, activation = 'relu', input_shape=(51,)))
    model.add(Dense(8, activation = 'relu'))
    model.add(Dense(4, activation = 'relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    return model

model = KerasClassifier(build_fn = build_model)
# The function that we specify to the build_fn argument when creating the KerasClassifier
wrapper can take arguments.

parameters = {'batch_size': [8, 16],
              'epochs': [40, 60],
              'optimizer': ['adam', 'rmsprop']}

grid_search = GridSearchCV(estimator = model,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 10)
grid_search = grid_search.fit(X_train, Y_train)
```

First, we need to import `GridSearchCV` from the `sklearn` library. The `estimator` parameter of `GridSearchCV`requires the model we are using for the hyper parameter tuning process. The `parameters` parameter requires a list of parameters and the range of values for each parameter of the specified *estimator*. In my model, I'll tune `batch_size`, `epochs` and `optimizer`.

- Batches for varying the number of samples before a weight update. `'batch_size': [8, 16]`, means that test batch size equal 8 or 16, which provide better performance.
- Epochs for training the model for a different number of exposures to the training dataset. `'epochs': [40, 60]` , means that test epochs equal 40 or 60, which provide better performance.
- Optimizers for searching different weight values. `'optimizer': ['adam', 'rmsprop']`, means that choosing adam or rmsprop will provide better performance

The result of Grid Search

```
best_parameters = grid_search.best_params_
best_accuracy = grid_search.best_score_
print("best_parameters: ")
print(best_parameters)
print("\nbest_accuracy: ")
print(best_accuracy)
```

> best_parameters: {'batch_size': 16, 'epochs': 60, 'optimizer': 'adam'}
>
> best_accuracy: 0.8879977876106194

We can see that the grid search discovered that using a adam optimizer, 60 epochs and a batch size of 16 achieved the best accuracy of approximately 89% on this problem.

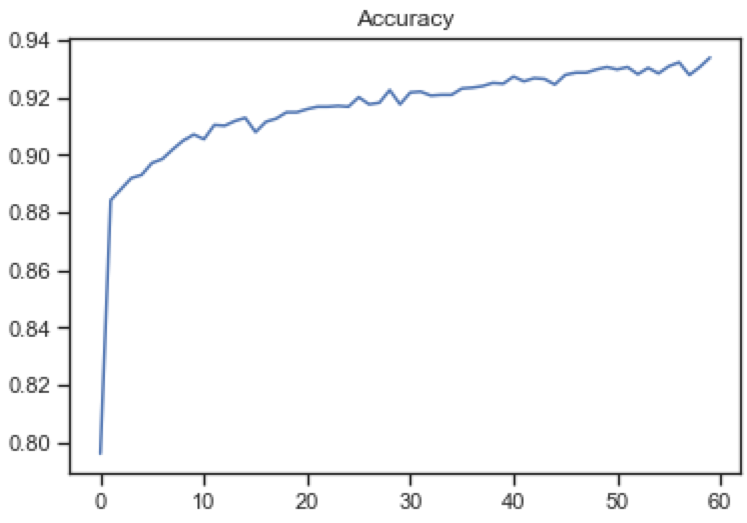
```
history= model.fit(X_train,Y_train,batch_size =16, epochs = 60)
score = model.evaluate(X_test, Y_test, verbose=0)
score[1]
```

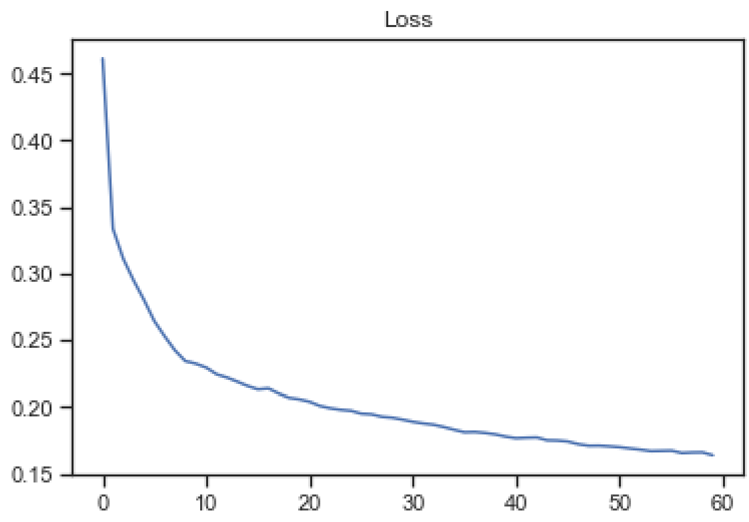
Output

Artificial neural networks have powerful pattern classification and prediction capabilities. In this study feedforward back propagation neural network with tan-sigmoid transfer functions in both the hidden and the output layer is applied for predict if the customer subscribes the deposit thus evaluate the bank marketing. In all these cases neural networks performed better prediction and less variation across different subgroups.

After testing all the parameter, 3 layers ANN with 60 epoch, 16 batch size, the accuracy is **0.8961** and the error is 0.306, with time **required running time 0:00:45.845316**, which is the optimal value among all the experimentations(I will list all my experimentations later in the paper.

```
import datetime
from matplotlib import pyplot as plt


test_loss, test_acc = model.evaluate(X_test, Y_test)
print('test_loss:', test_loss)
print('test_acc:', test_acc)

#plot
plt.plot(np.arange(0,len(history.history['loss'])), history.history['loss'])
plt.title("Loss")
plt.show()
plt.plot(np.arange(0,len(history.history['acc'])), history.history['acc'])
plt.title("Accuracy")
plt.show()


stop_time = datetime.datetime.now()
print ("Time required for training:",stop_time - start_time)
```





```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_17 (Dense)             (None, 12)                624       
_________________________________________________________________
dense_18 (Dense)             (None, 8)                 104       
_________________________________________________________________
dense_19 (Dense)             (None, 4)                 36        
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 5         
=================================================================
Total params: 769
Trainable params: 769
Non-trainable params: 0
```

```
Epoch 45/60
3616/3616 [==============================] - 1s 203us/step - loss: 0.1749 - acc: 0.9245
Epoch 46/60
3616/3616 [==============================] - 1s 203us/step - loss: 0.1742 - acc: 0.9278
Epoch 47/60
3616/3616 [==============================] - 1s 217us/step - loss: 0.1721 - acc: 0.9287
Epoch 48/60
3616/3616 [==============================] - 1s 208us/step - loss: 0.1710 - acc: 0.9287
Epoch 49/60
3616/3616 [==============================] - 1s 215us/step - loss: 0.1711 - acc: 0.9298
Epoch 50/60
3616/3616 [==============================] - 1s 216us/step - loss: 0.1705 - acc: 0.9306
Epoch 51/60
3616/3616 [==============================] - 1s 204us/step - loss: 0.1699 - acc: 0.9298
Epoch 52/60
3616/3616 [==============================] - 1s 203us/step - loss: 0.1689 - acc: 0.9306
Epoch 53/60
3616/3616 [==============================] - 1s 203us/step - loss: 0.1681 - acc: 0.9281
Epoch 54/60
3616/3616 [==============================] - 1s 209us/step - loss: 0.1670 - acc: 0.9303
Epoch 55/60
3616/3616 [==============================] - 1s 206us/step - loss: 0.1671 - acc: 0.9284
Epoch 56/60
3616/3616 [==============================] - 1s 212us/step - loss: 0.1673 - acc: 0.9309
Epoch 57/60
3616/3616 [==============================] - 1s 214us/step - loss: 0.1656 - acc: 0.9322
Epoch 58/60
3616/3616 [==============================] - 1s 206us/step - loss: 0.1659 - acc: 0.9278
Epoch 59/60
3616/3616 [==============================] - 1s 202us/step - loss: 0.1660 - acc: 0.9306
Epoch 60/60
3616/3616 [==============================] - 1s 207us/step - loss: 0.1638 - acc: 0.9339
905/905 [==============================] - 0s 29us/step
test_loss: 0.3060592791647007
test_acc: 0.8961325966850828
```

Experimentation

Change Layer hidden units (3 experimentations)

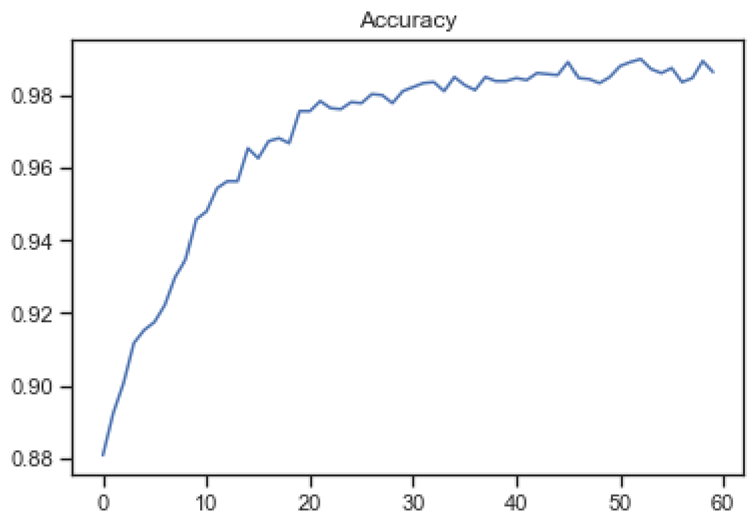
1. Original Model

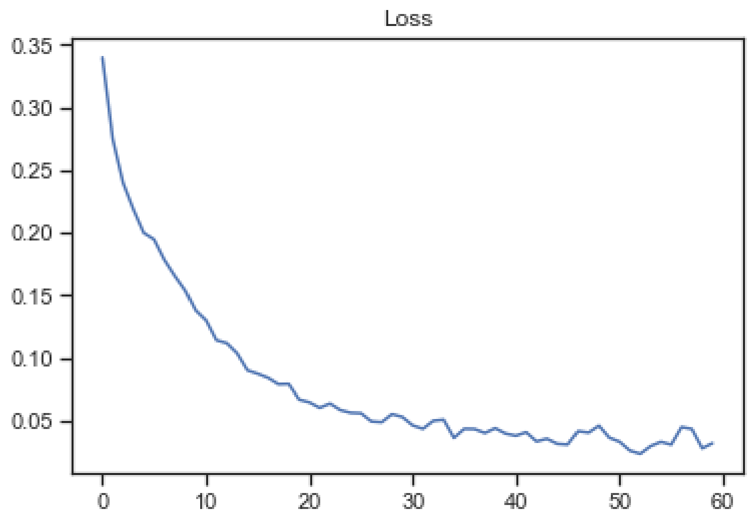
This is not an optimal value, it is use to compare with the other models.

test_loss: 0.7886, test_acc: 0.8917, with time required running time 0:00:43.845316.

```
model.add(Dense(150, activation = 'relu', input_shape=(51,)))
model.add(Dense(100, activation = 'relu'))
model.add(Dense(50, activation = 'relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 150)               7800
_________________________________________________________________
dense_2 (Dense)              (None, 100)               15100
_________________________________________________________________
dense_3 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 51
=================================================================
Total params: 28,001
Trainable params: 28,001
Non-trainable params: 0
```





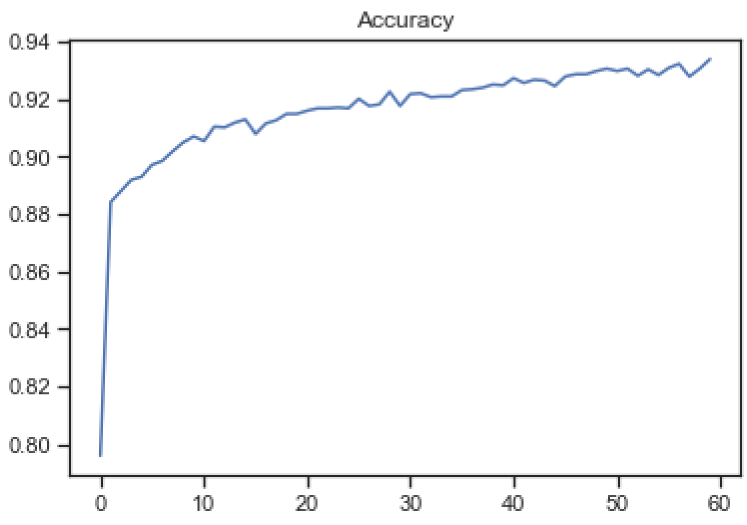
2. Change Hidden Layer Units to 12, 8, 4, 1 -- Optimal Value

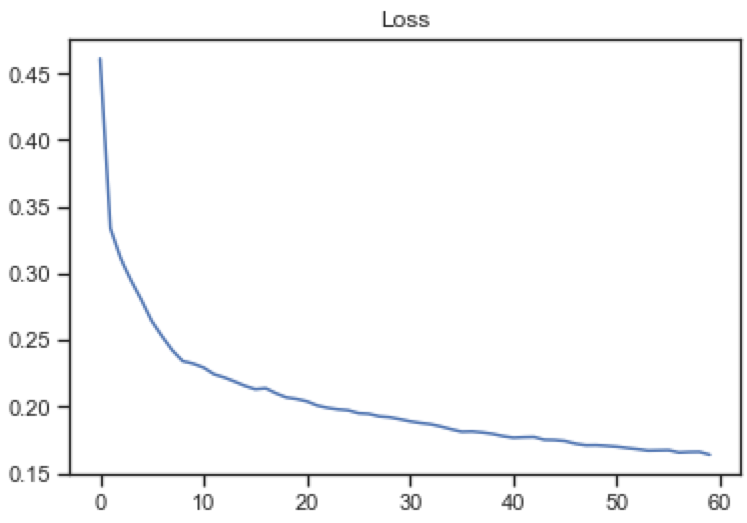
From a very inspirational research paper from by Huang et al. (2003) that suggests the optimal number of hidden neurons in each of the two hidden layers of the neural. Essentially, it is saying that the optimal number of hidden nodes in the first hidden layer is: sqrt[(m+2)N] + 2sqrt[N/(m+2)] and in the second hidden layer, the optimal number of hidden nodes is: m*sqrt[N/(m+2)], where N = number of inputs, and m = number of outputs. The estimated hidden nodes should be approximately 8 for the the first layer, the second layers is 4.

test_loss: 0.3061, test_acc: 0.8961

The accuracy is slightly increase from 0.8917 to 0.8961by 0.044 with time required running time 0:00:45.845316.

```
model = Sequential()
model.add(Dense(12, activation = 'relu', input_shape=(51,)))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(4, activation = 'relu'))
model.add(Dense(1, activation='sigmoid'))
```



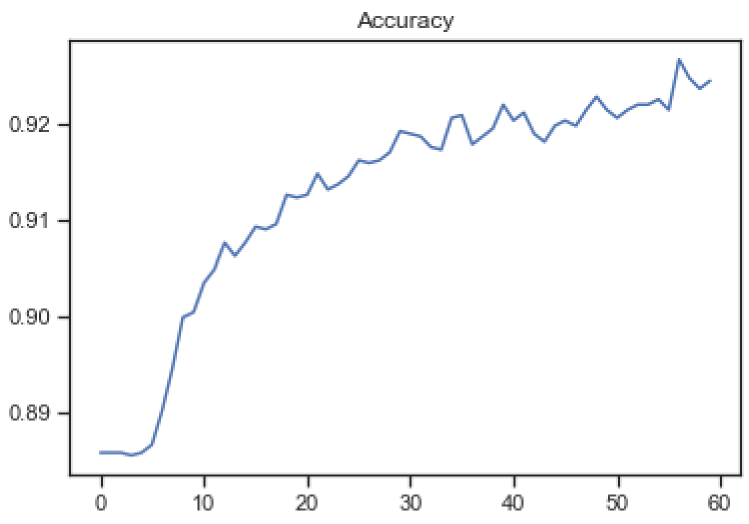


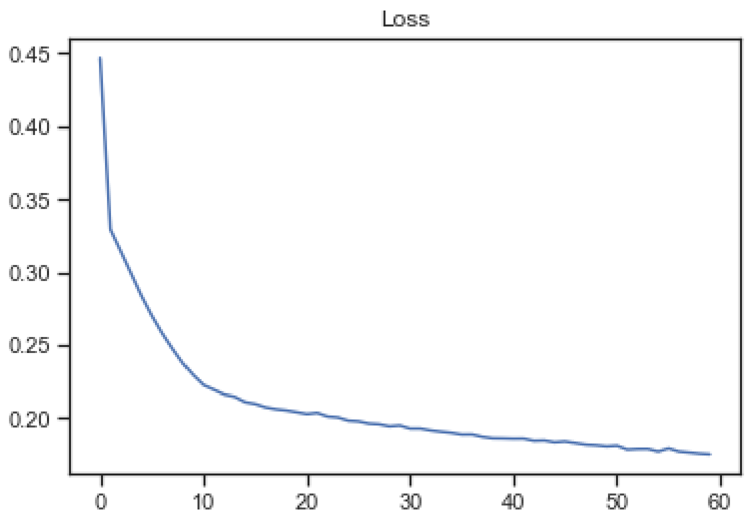
3. Change Hidden Layer Units to 8, 6, 4, 1

To test the possible optimal value, I change the hidden unit to 8, 6, 4, and 1. However, the accuracy is slightly increase from 0.8917 to 0.8950 by 0.03 with time required running time 0:00:40.165091

test_loss: 0.2617, test_acc: 0.8950

```
model.add(Dense(8, activation = 'relu', input_shape=(51,)))
model.add(Dense(6, activation = 'relu'))
model.add(Dense(4, activation = 'relu'))
model.add(Dense(1, activation='sigmoid'))
```



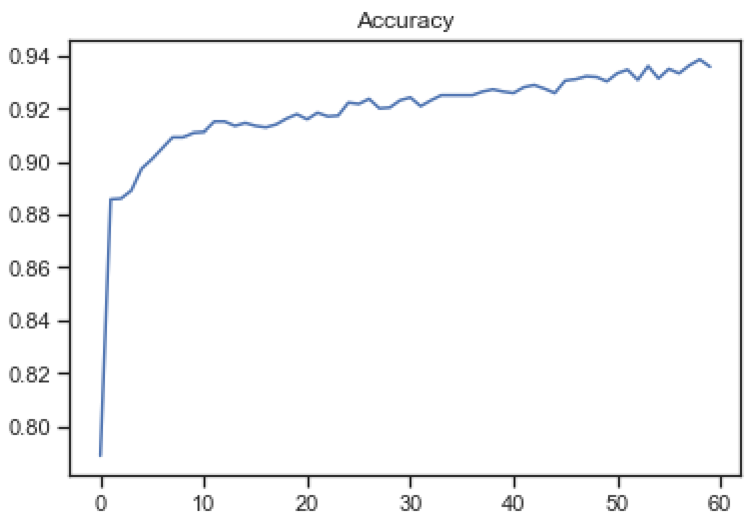


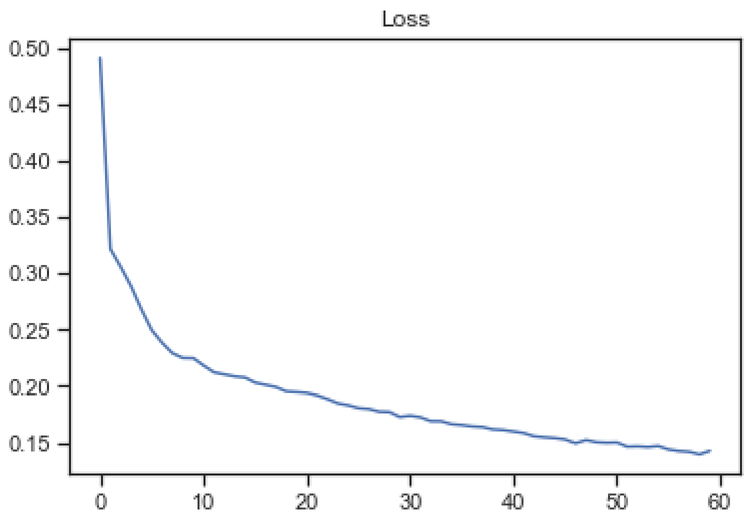
- Change Hidden Layer Units to 16, 8, 6, 1

For further experimental, I change the hidden unit to 16, 8, 6, and 1. However, the accuracy is slightly decrease from 0.8917 to 0.8873 by 0.1 with time required running time 0:00:40.165091

test_loss: 0.3267, test_acc: 0.8873

```
model.add(Dense(16, activation = 'relu', input_shape=(51,)))
model.add(Dense(8, activation = 'relu'))
model.add(Dense(6, activation = 'relu'))
model.add(Dense(1, activation='sigmoid'))
```





Change Number of Layer

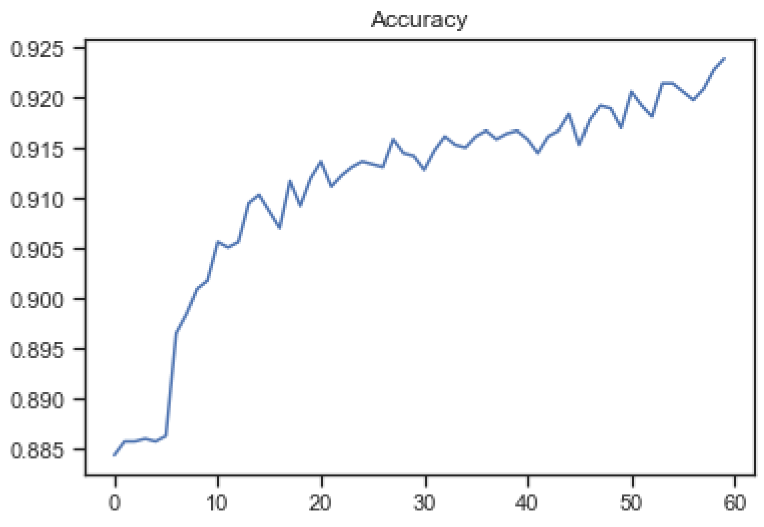
1. Halving number of layers

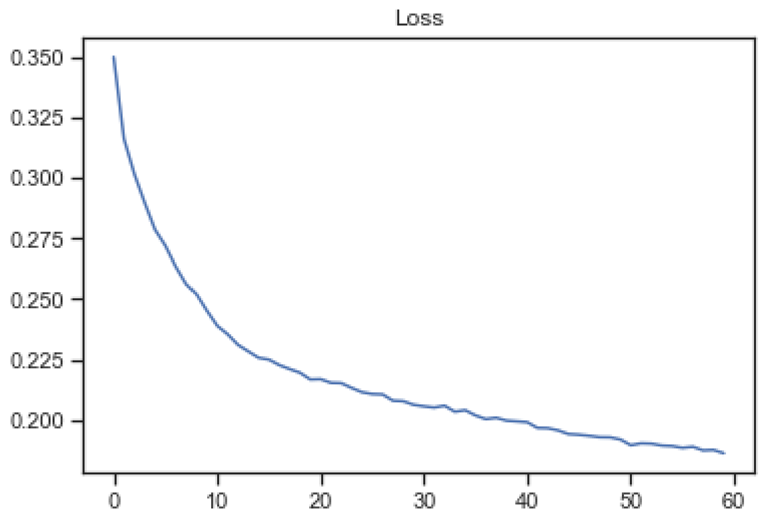
test_loss: 0.26555703782080287 test_acc: 0.8928176795580111

Time required for training: 0:01:07.567739

Comparing to the original model, after halving number of hidden layers, the test accuracy is decrease to 0.8928, the the time required is increase.

```
model = Sequential()
model.add(Dense(8, activation = 'relu', input_shape=(51,)))
model.add(Dense(4, activation = 'relu'))
model.add(Dense(1, activation='sigmoid'))
```





2. Optimizer

I test the optimizer on Grid Search, 'adam' is the optimal optimizer.

Reference

A Beginner's Guide to Neural Networks and Deep Learning. (n.d.). Retrieved April 21, 2019, from https://skymind.ai/wiki/neural-network

Hewa, K., & Hewa, K. (2019, January 05). An introduction to Grid search. Retrieved April 21, 2019, from https://medium.com/datadriveninvestor/an-introduction-to-grid-search-ff57adcc0998

Srivastava, S., & Srivastava, S. (2019, February 11). Feature Scaling in Scikit-learn. Retrieved April 21, 2019, from https://medium.com/@shobhitsrivastava18th/feature-scaling-in-scikit-learn-b11209d949e7

Use Keras Deep Learning Models with Scikit-Learn in Python. (2018, March 10). Retrieved April 21, 2019, from https://machinelearningmastery.com/use-keras-deep-learning-models-scikit-learn-python/