Building an ANN using Python.

Here the ANN is built and trained to predict weather the customer stays or leaves a bank based on the data given with many dependent variables and single independent variable. The network would return True if the customer stays in a bank otherwise it returns False.

Procedure:

1. Initializing randomly the weights to a value close to zero (but not zero).
2. The first observation of the datasheet is given as an input to the input layer where each feature acts as one input node.
3. Then the forward propagation takes place from left to right. Here the neurons are activated which impacts the neurons weights affecting their activation. It propagates the activation till the predicted result y is obtained.
4. The predicted output is compared with the actual result and their error is found using the cost function.
5. Then the backpropagation takes place where the error is back propagated, and the weights are updated.
6. After repeating the steps from 1 to 5 if the weight is updated after each observation (Reinforcement Learning) or if the weights are updated only after a batch of observation (Batch Learning).
7. Once the complete training set is passed through the ANN, it makes an Epoch. we can redo more number of Epochs.
8. An ANN network here is built using Python programming which involves the following steps.
    a. At first, we take the data for which an ANN network is to be built. We are building a ANN for a data from a bank given with 11 different independent features or input data represented as X (as shown in the figure 4) where an ANN needs to predict weather or not the customer stays in a bank or leaves the bank based on the given data for features. So here the output is a true or false statement hence making it binary output given by Y as shown in figure 4.
    b. Then the data is preprocessed and differentiate the Independent and the Dependent variables in the given data. Here the complete data set is split into training set and testing data set as shown in the figure 5 and figure 6 for both X and Y. Where the network is trained on the training set obtain some experience and knowledge about the output. This knowledge is used to predict the output of the test set as shown in the figure 8.
    c. This step involves the Building of an ANN network. Here our network has three layers namely
        - One Input layer with 11 neurons as our data has 11 independent features.
        - One hidden layer with 6 neurons.
        - One output layer with 1 neuron as our output is binary (0 or 1) that is true or false.
    d. This step involves the prediction and evaluation of the network output. Here the output for the test set is obtained and the performance is evaluated using the confusion matrix that is generated in this step as shown in the figure 7. This matrix has the entries with number of correct prediction (sum of top left diagonal elements) and number of

incorrect prediction (sum of top right diagonal elements). By which the accuracy of the network is determined to be 83%.

e. The output for a single case is predicted as false as shown in the figure 9.

f. This accuracy can be improved by Evaluating, Improving and tuning the ANN. This gives an accuracy of 84.9% as shown in the figure 10.

Figure 5 Training set(left) and Test set (Right) for Input data(X)

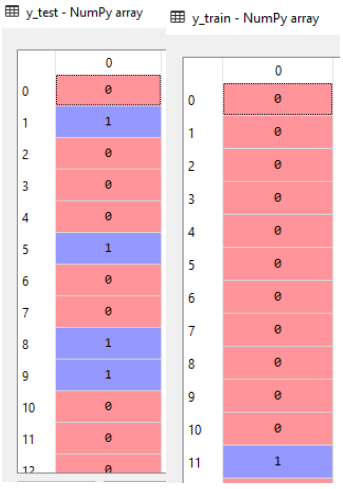


Figure 6. Training set(left) and Test set (Right) for Output data(Y).

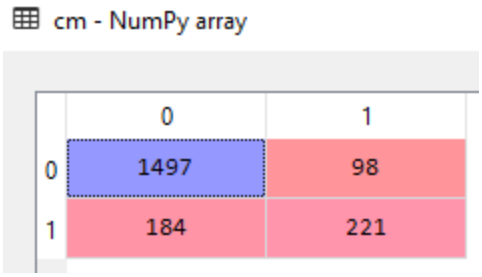


Figure 7. Confussion matrix.

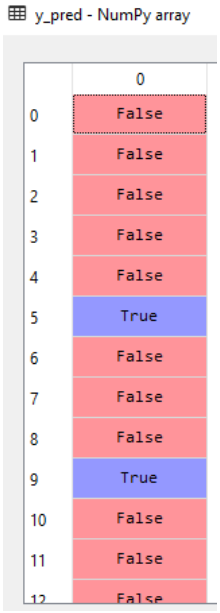


Figure 8. Predicted output for the Test Set.

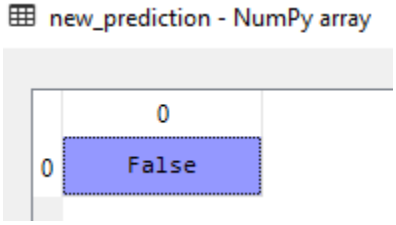
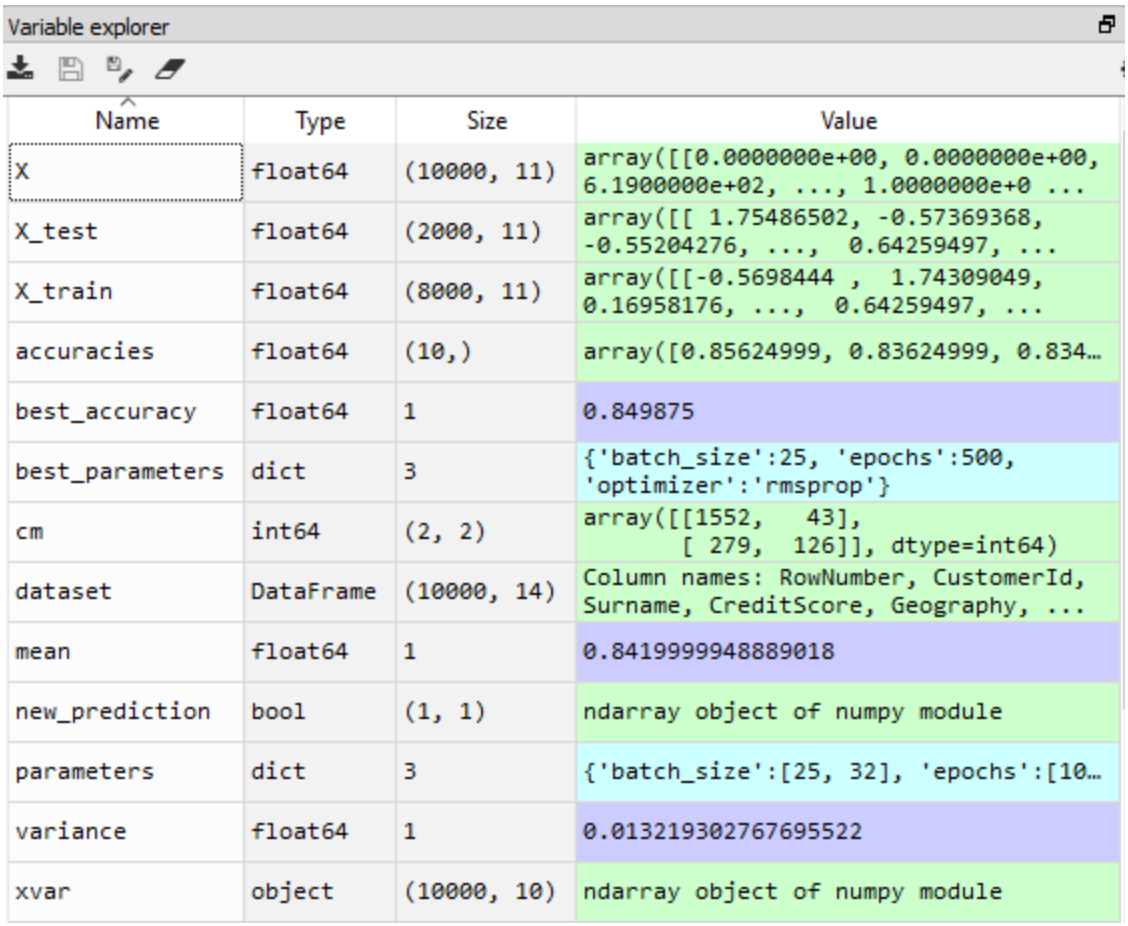


Figure 9 Output return for a single pridection.



Name	Type	Size	Value
X	float64	(10000, 11)	array([[0.0000000e+00, 0.0000000e+00, 6.1900000e+02, ..., 1.0000000e+0 ...
X_test	float64	(2000, 11)	array([[1.75486502, -0.57369368, -0.55204276, ..., 0.64259497, ...
X_train	float64	(8000, 11)	array([[-0.5698444 , 1.74309049, 0.16958176, ..., 0.64259497, ...
accuracies	float64	(10,)	array([0.85624999, 0.83624999, 0.834…
best_accuracy	float64	1	0.849875
best_parameters	dict	3	{'batch_size':25, 'epochs':500, 'optimizer':'rmsprop'}
cm	int64	(2, 2)	array([[1552, 43], [279, 126]], dtype=int64)
dataset	DataFrame	(10000, 14)	Column names: RowNumber, CustomerId, Surname, CreditScore, Geography, ...
mean	float64	1	0.8419999948889018
new_prediction	bool	(1, 1)	ndarray object of numpy module
parameters	dict	3	{'batch_size':[25, 32], 'epochs':[10…
variance	float64	1	0.013219302767695522
xvar	object	(10000, 10)	ndarray object of numpy module

Figure 10. The variables of ANN with accuracy of 84.9%.

Building a CNN using python.

Procedure :

A CNN is built and trained to identify between the images of cats and Dogs using python programming.

a) The Given data set (Images of dogs and cats) are split into two sets i.e., Training set and testing set. That are stored in separate folders.
b) This data is convoluted by specifying the number of rows, columns and the number of features for the feature detector to obtain the feature map and also Relu function is applied at the same time.
c) Next, we apply the pooling operation by specifying the size of the pooling window.
d) Then the pooled feature map is flattened.
e) This flattened vector is used to build the full connection by specifying the number of neurons in input layer, Hidden layers and the output layers.
f) Then this CNN is compiled.

g) The final stage is to preprocess the image. The main aim is to enrich our training set without adding more images to get good performance without or little overfitting.
h) The network was built and obtained an accuracy of 79.4% as shown in the figure 14.
i) When the single input image is given the output was predicted to be dog. Which was rightly pridected.as the input image given was a dog and the CNN prediction was also a Dog.
j) The network can be further evaluated, improved and tuned to improve the accuracy.

```
Epoch 20/25
8000/8000 [==============================] - 6061s 758ms/step - loss: 0.0093 -
acc: 0.9971 - val_loss: 1.6180 - val_acc: 0.8055
Epoch 21/25
8000/8000 [==============================] - 3773s 472ms/step - loss: 0.0083 -
acc: 0.9975 - val_loss: 1.7529 - val_acc: 0.8068
Epoch 22/25
8000/8000 [==============================] - 2218s 277ms/step - loss: 0.0080 -
acc: 0.9976 - val_loss: 1.6340 - val_acc: 0.8137
Epoch 23/25
8000/8000 [==============================] - 2234s 279ms/step - loss: 0.0082 -
acc: 0.9976 - val_loss: 1.7307 - val_acc: 0.8043
Epoch 24/25
8000/8000 [==============================] - 2250s 281ms/step - loss: 0.0084 -
acc: 0.9975 - val_loss: 1.6449 - val_acc: 0.8037
Epoch 25/25
8000/8000 [==============================] - 2269s 284ms/step - loss: 0.0074 -
acc: 0.9978 - val_loss: 1.8248 - val_acc: 0.7948
Out[14]: <keras.callbacks.History at 0x1d92c5ae4e0>
```

Figure 14. The CNN completing 25 epoch with an accuracy of 79.4%.

Building a RNN using Python

Procedure:

1. A RNN is built and trained on the data i.e. google stock price of four years and then tested to predict the stock price of google of 2017 .
2. In this process of building a RNN, the python programming is used.
3. Here we first preprocess the data (google stock prices),which involves the importing of the dataset then feature scaling it where it is followed by reshaping the dataset.
4. Then we build the RNN by assigning different sizes for the input layer ,hidden layer and the output layer.
5. Then this built network is compiled and fitted to the training set. The figure 24 shows different variables of the RNN .

6. Finally the stock price for the test data are predicted and the output is visualized as shown in the figure 25.

Name	Type	Size	Value
X_test	float64	(20, 60, 1)	array([[[0.9299055], [0.93113327],
X_train	float64	(1198, 60, 1)	array([[[0.08581368], [0.09701243],
dataset_test	DataFrame	(20, 6)	Column names: Date, Open, High, Low, Close, Volume
dataset_total	Series	(1278,)	Series object of pandas.core.series module
dataset_train	DataFrame	(1258, 6)	Column names: Date, Open, High, Low, Close, Volume
i	int	1	79
inputs	float64	(80, 1)	array([[0.9299055], [0.93113327],
predicted_stock_price	float32	(20, 1)	array([[782.33405], [779.62805],
real_stock_price	float64	(20, 1)	array([[778.81], [788.36],
training_set	float64	(1258, 1)	array([[325.25], [331.27],
training_set_scaled	float64	(1258, 1)	array([[0.08581368], [0.09701243],
y_train	float64	(1198,)	array([0.08627874, 0.08471612, …

Figure 24. The variables of the RNN

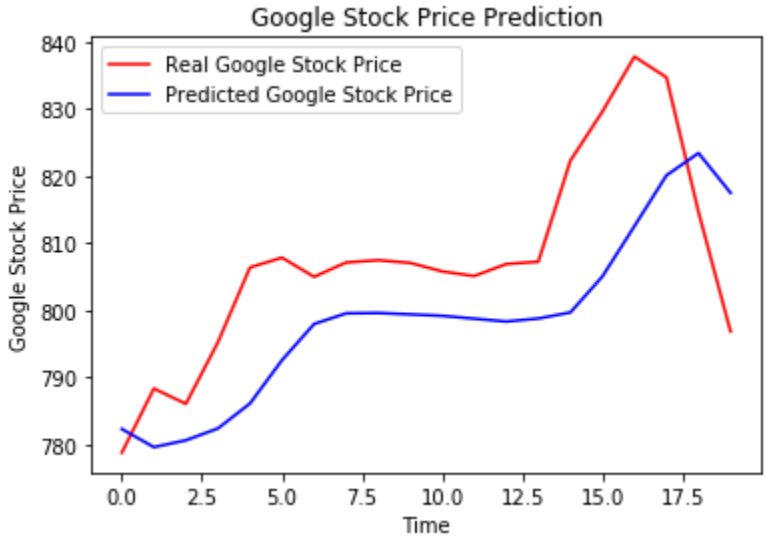


Figure 25. The real and the predicted output of Google stock price for the year 2017.