ARTIFICIAL NEURAL NETWORK

and INTRODUCTION to DEEP LEARNING

by Alexandre Genette A18

SUMMARY :

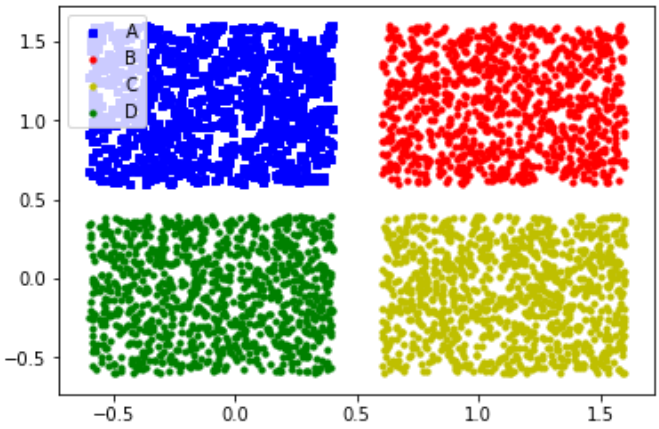
Exercise : Classification of 4 Clusters

Github : https://github.com/alexg06/Jupyter-Python/blob/master/DeepLearning/Classification%20MLP%204%20clusters.ipynb

Random dataset generated as follow:

```
#A
dfA['X'] = np.random.random([1000]) - q
dfA['Y'] = np.random.random([1000]) + q
dfA['Label'] = 0
#B
dfB['X'] = np.random.random([1000]) + q
dfB['Y'] = np.random.random([1000]) + q
dfB['Label'] = 1
#C
dfC['X'] = np.random.random([1000]) + q
dfC['Y'] = np.random.random([1000]) - q
dfC['Label'] = 0
#D
dfD['X'] = np.random.random([1000]) - q
dfD['Y'] = np.random.random([1000]) - q
dfD['Label'] = 1
```

A and C have the same label 0, B and D labeled as 1.
We plot the clusters in order to visualize the full dataset:



The four clusters are well separated. The four datasets are concatenated in one dataframe then splitted in 2, a training set and a test set. The splitting is done by sklearn with train_test_split function:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =  train_test_split(df, label, test_size=0.2,
random_state=777)
```

The model is a simple MultiLayerPerceptron of the sklearn library: MLPClassifier()
I will use this one because for classification prediction problems MLP is the most suitable Artificial Neural Network. Specially for features and target coming from a basic dataframe.

```
from sklearn.neural_network import MLPClassifier
model = MLPClassifier(activation='relu', alpha=1e-3, solver='lbfgs', validation
_fraction=0.1)
mlp = model.fit(X_train, y_train)
```

The activation function is a relu, alpha is set to 1e-3 for the best result, but no real differences between 1e-5 and 0.1. the validation fraction used is set to 10% of the training dataset.

A confusion matrix is used to represent the prediction of the test set:

```
from sklearn.metrics import confusion_matrix
score = mlp.score(X_test, y_test)
print('Score is {}'.format(score))
confusion = confusion_matrix(y_test, y_predict)
confusion = pd.DataFrame(confusion, columns=['0', '1'], index=['0', '1'])
print(confusion)
```

```
Confusion Matrix :

Score is 1.0
     0    1
0  397    0
1    0  403
```

➔ The prediction is 100% accurate.

Then every different dataset A, B, C and D get a different label (0,1,2 and 3), as follow:

```
#A
dfA_b['X'] = np.random.random([1000]) - q
dfA_b['Y'] = np.random.random([1000]) + q
dfA_b['Label'] = 0
#B
dfB_b['X'] = np.random.random([1000]) + q
dfB_b['Y'] = np.random.random([1000]) + q
dfB_b['Label'] = 1
#C
dfC_b['X'] = np.random.random([1000]) + q
dfC_b['Y'] = np.random.random([1000]) - q
dfC_b['Label'] = 2
#D
dfD_b['X'] = np.random.random([1000]) - q
dfD_b['Y'] = np.random.random([1000]) - q
dfD_b['Label'] = 3
```

The same MLP model is applied to this dataset:

```
from sklearn.model_selection import train_test_split
X_train_b, X_test_b, y_train_b, y_test_b =  train_test_split(df_b, label_b, tes
t_size=0.2, random_state=777)
model_b = MLPClassifier(activation='relu', alpha=1e-3, solver='lbfgs', validati
on_fraction=0.1)
mlp_b = model_b.fit(X_train_b, y_train_b)
y_predict_b = mlp_b.predict(X_test_b)

from sklearn.metrics import confusion_matrix
score_b = mlp_b.score(X_test_b, y_test_b)
print('Score is {}'.format(score_b))
confusion_b = confusion_matrix(y_test_b, y_predict_b)
confusion_b = pd.DataFrame(confusion_b, columns=['0', '1', '2', '3'], index=['0
', '1', '2', '3'])
print(confusion_b)
```

```
Score is 1.0
     0    1    2    3
0  211    0    0    0
1    0  208    0    0
2    0    0  186    0
3    0    0    0  195
```

⇨ The model is again 100% accurate on the test set for the 4 classes.

Exercise IRIS classification:

GitHub : https://github.com/alexg06/Jupyter-Python/blob/master/DeepLearning/IRIS%20MLP%20and%20unsupervised%20RBM%20.ipynb

The goal is to make the best model for a classification of Iris from the Iris dataset, with 3 classes, 4 features (sepal length, sepal width, petal length, petal width) and 150 observations.
Classes are:

0 = Setosa
1 = Versicolor
2 = Virginica

Model MLP with sklearn :

```
X_train, X_test, y_train, y_test = train_test_split(
    df, target, test_size=0.2, random_state=0)
model = MLPClassifier(activation='relu', alpha=1, solver='lbfgs', vali
dation_fraction=0.1)
mlp = model.fit(X_train, y_train)
y_predict = mlp.predict(X_test)

from sklearn.metrics import confusion_matrix
score = mlp.score(X_test, y_test)
print('Score is {}'.format(score))
confusion = confusion_matrix(y_test, y_predict)
confusion = pd.DataFrame(confusion, columns=['setosa', 'versicolor', '
virginica'], index=['setosa', 'versicolor', 'virginica'])
print(confusion)
```

Algorithm was tested with different values of alpha:

Alpha = 1e-3 ======> score = 1.0
Alpha = 1e-1 ======> score = 1.0
Alpha = 1 ======> score = 1.0
Alpha = 10 ======> score = 1.0
Alpha = 100 ======> score = 0.5666666666666667

Best result is (confusion matrix):
Score is 1.0

	setosa	versicolor	virginica
setosa	11	0	0
versicolor	0	13	0
virginica	0	0	6

Unsupervised modelization with RBM:

Testing with BernoulliRBM of sklearn was bad, something went wrong with the pipeline because I only got 1 class for the prediction:

	setosa	versicolor	virginica
setosa	0	0	11
versicolor	0	0	13
virginica	0	0	6

Exercise: MackeyGlass Time Series Prediction:

GitHub : https://github.com/alexg06/Jupyter-Python/blob/master/DeepLearning/LSTM%20Mackey_Glass.ipynb

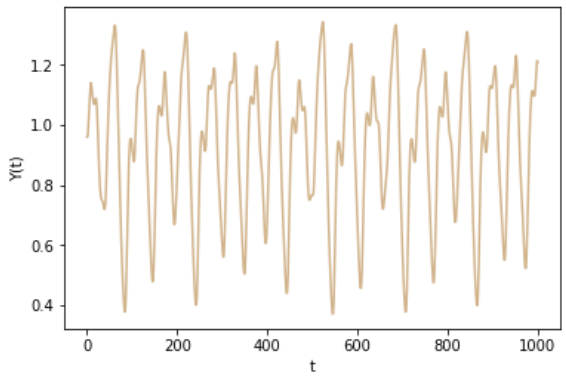
The goal of this exercise is to predict the next 500 iterations of a Mackey-Glass differential equation from 500 iterations.

The function is defined by:

- b = 0.1
- q = 0.6
- tau = 17
- N = 1000

```
def mackey_glass(N, b=0.1, c=0.2, tau=17):
        y = np.array([0.96, 0.96, 0.97, 1.00, 1.03, 1.07, 1.1, 1.13, 1.14, 1.14,
        1.13, 1.12, 1.1, 1.09, 1.08, 1.07, 1.07, 1.07])
        for t in range(17,N-1):
                y_t_plus1 = y[t] -b*y[t]+( (c*y[t-tau]) / (1+(y[t-tau]**10)))
                y = np.append(y, y_t_plus1)
        return y
```

For 1000 iterations the Graph is the following one:



The model will be a RNN(Recurrent Neural Network) with the 50 past observations. The keras library will be used to define the RNN.
But first the inputs should be adapted to fit the inputs (50 past observations), using the TimeSeriesGenerator function of keras.

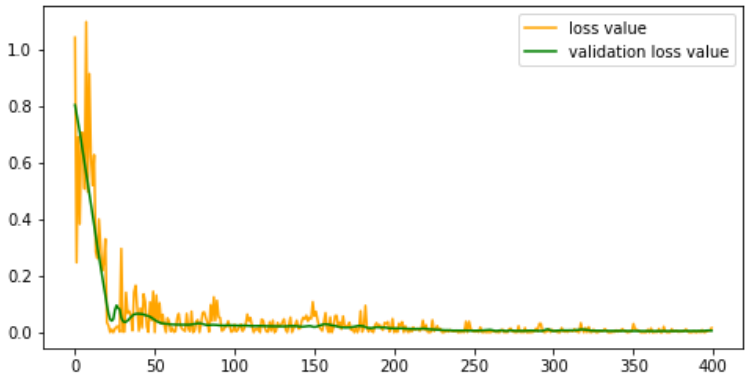
The RNN will be a LSTM (Long Short-Term Memory), because a simple recurrent network(MLP) is not being able to capture long-term dependencies of a sequence (here 500 predictions over 100O iterations).

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.preprocessing.sequence import TimeseriesGenerator

n_input = 50 #number of past observations used for the RNN

series_training = np.array(df_train['Y(t)'])
series_test = np.array(df_test['Y(t)'])
series_validation = np.array(df_validation['Y(t)'])

#RESHAPE SERIES FOR LSTM
series_training = series_training.reshape(len(series_training), 1)
series_test = series_test.reshape(len(series_test), 1)
series_validation = series_validation.reshape(len(series_validation), 1)

generator_training = TimeseriesGenerator(series_training, series_training, leng
th=n_input, batch_size=1)
generator_test = TimeseriesGenerator(series_test, series_test, length=n_input,
batch_size=1)
generator_validation = TimeseriesGenerator(series_validation, series_validation
, length=n_input, batch_size=1)
```

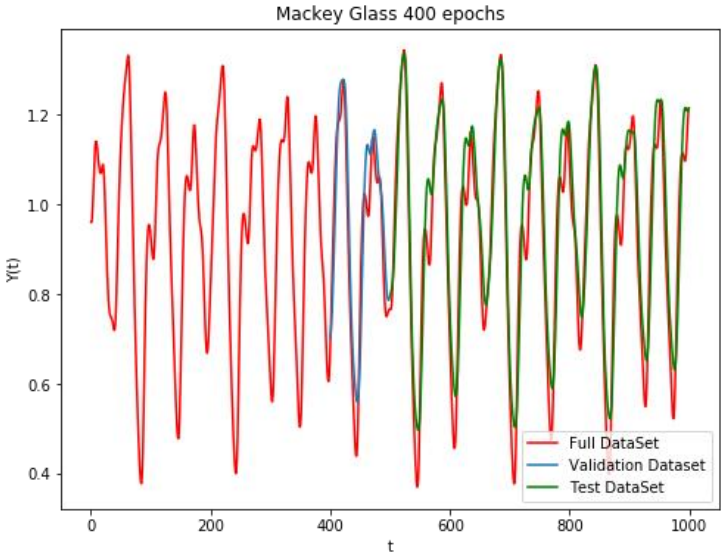
The model RNN is created as follow:

```
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_input, 1)))
#model.add(Dense(128, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit_generator(generator_training, steps_per_epoch=1, epochs=400,
verbose=0, validation_data=generator_validation)
```

With 400 epochs :



The final Graph is the following one:

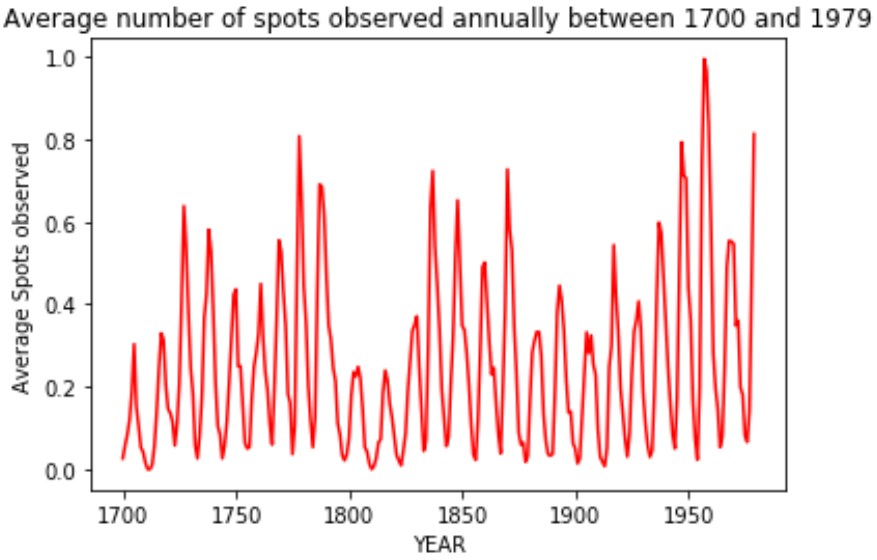


Exercise: Sunspots from 1700 to 1979 (source : 'Sun Spots NN.ipynb')

On Github : https://github.com/alexg06/Jupyter-Python/blob/master/DeepLearning/Sun%20Spots%20NN.ipynb

The goal is to predict the average annual spots of the sun from 1956 to 1979 (Test set), with validation from 1921 to 1955(validation set) and training set from 1712 to 1920 (training set). The prediction per year will be made using a neural network with the 12 last year observations.
The model will be an MLP but tanks to the Time Series Generator the input will be turned into a buffer of the past 12 years. We have only one feature in the dataset (#spots) and a few observations.

First, we plot the whole dataset:




```
plt.plot(df['Year'], df['Spots'], color='r')
plt.xlabel('YEAR')
plt.ylabel('Average Spots observed')
plt.title('Average number of spots observed annually between 1700 and 1979')
plt.show()
```

This is a univariate Time Series dataset and I decided to use Keras library under python.
First, we need to convert the input data using a Keras tools, TimeSeriesGenerator() that permits to create a vector of the 12 last observations(12 years).

```
from keras.models import Sequential
from keras.layers import Dense
from keras.preprocessing.sequence import TimeseriesGenerator
n_input = 12
generator_training = TimeseriesGenerator(series_training, series_training, leng
th=n_input, batch_size=1)
```

```
generator_training[0]
```

```
Out[45]:(array([[0.0262, 0.0575, 0.0837, 0.1203, 0.1883, 0.3033, 0.1517, 0.
1046, 0.0523, 0.0418, 0.0157, 0.    ]]), array([0.]))
```

Training, validation and Test sets need to be converted by the generator as well, then an artificial neural network is defined with 12 dimensions input (corresponding to the 12 last years of observation) thanks to the TimeSeriesGenerator. The output has 1 node because it will return only one 1 float corresponding to the average number of spots during the year(the target). There is no hidden layer.

```
#Creation of the Model
model = Sequential() #add layers
model.add(Dense(12, activation='relu', input_dim=n_input))#input layer
model.add(Dense(1)) #output layer 1 node
model.compile(optimizer='adam', loss='mse')
```

Then the model is trained using the training dataset (1712-1920) and the validation dataset (1921-1955) with 1000 iterations(epoch)

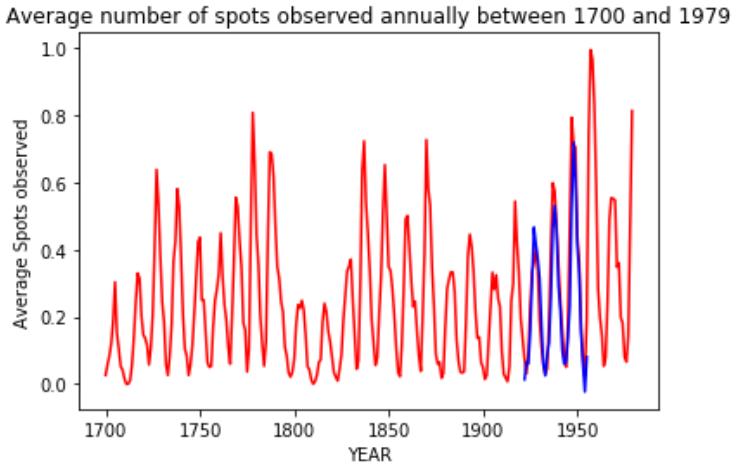
```
#TRAINING THE MODEL
model.fit_generator(generator_training, steps_per_epoch=1, epochs=1000, verbose
=0, validation_data=(generator_validation))
```

Prediction for the validation is superposed on the initial graph:

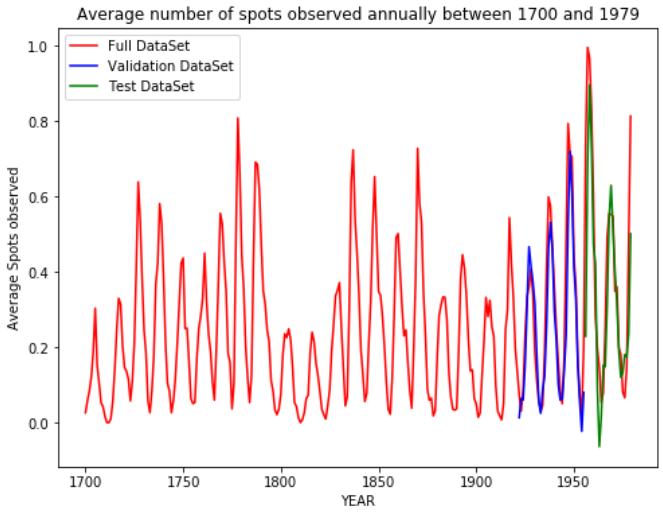
```
Y_validation = []
```

```
for i in range(0,len(generator_validation)):
    x_input = generator_validation[i][0].reshape((1, n_input))
    yhat = model.predict(x_input, verbose=0)
    Y validation = Y validation + [yhat]
Y_valid = np.reshape(Y_validation, (34,))
```

Same is done for the Test set.



Then the Test set is predicted and superposed on the graph (in green):



There is a discontinuity in the test curve (in green), maybe the first prediction is not really accurate and more memory could be used(increasing the number of past observations used for prediction).

The AVR is calculated but I don't know if I did something wrong with the formula because the prediction fits nicely but the AVR is not really good:

```
Sum_num = 0 #numerator
for i in range(0, len(generator_test)):
    Sum_tmp = ((df_test['Spots'][256+i] - Y_pred[i])**2)
    Sum_num = Sum_num + Sum_tmp

Sum_den = 0 #denominator
for i in range(0, len(df['Spots'])):
    Sum_tmp = ((df['Spots'][i] - np.average(df['Spots']))**2)
    Sum_den = Sum_num + Sum_tmp
ARV = (Sum_num * len(df['Spots'])) / (Sum_den * len(generator_test))
```

```
In [60]:    ARV

Out[60]:    8.012073373938952
```

THE COMPLETE SOURCE IS THE FILE : Sun Spots NN.ipynb

Exercise : Classification for healthcare (source: Heart Disease ANN.ipynb)

On Github : https://github.com/alexg06/Jupyter-Python/blob/master/DeepLearning/Heart%20Disease%20ANN.ipynb

The goal is to predict a diagnosis for a heart disease from a dataset with 3 features and 1 target (binary classification positive or negative).

	sample	cholesteral	thalac	oldpeak	disease
0	train	261	141	3	positive
1	train	263	105	2	negative
2	train	269	121	2	negative

First, I need to convert the label into 0(negative) and 1(positive)

```
df = pd.get_dummies(df, columns=['disease'], drop_first=True)
df.rename(columns={'disease_positive':'label'}, inplace=True)
```

If the the label is positive then the value is 1 else it is 0(negative)
I design the Artificial Neural Network with 3 inputs(corresponding to 3 features dimension=3), an hidden layer of 64 nodes (because I got the best result with this layer) and an input of 1 node because the result is the prediction.

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(3, activation='relu', input_dim=3))
model.add(Dense(64, activation='relu')) #best 64 so far
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy
'])
```

The activation function is relu with input_dim=3 (3 dimensions because of 3 features) and the output activation function is a sigmoid for binary classification.

```
model.fit(df_training,label_train, epochs=500, verbose=1, validation_split=0.5,
batch_size=1)
```

The learning is done with 500 iterations(epochs), validation split corresponds to the size of the training data set used for the validation during the process(here 50% of the observations of the training dataset)

```
y_predict = model.predict(df_test)
y_predict = np.round(y_predict)
```

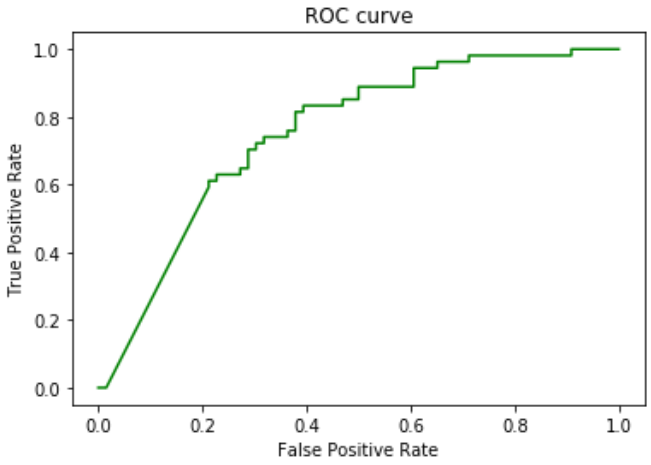
y_predict corresponds to the prediction made by the model with Test dataset (df_test here) and rounded to give 0 or 1 for the class (the output returns a probability between 0 and 1)
the confusion matrix is given by:

```
from sklearn.metrics import confusion_matrix
confusion = confusion_matrix(label_test, y_predict)
confusion = pd.DataFrame(confusion, columns=['Disease', 'Healthy'], index=['Dis
ease', 'Healthy'])
print(confusion)
```

```
           Disease  Healthy
Disease         42        24
```

Healthy 14 40

The ROC curve is:



Conclusion: The result is not very good, may be improved with more iterations and changing the size of validation set to avoid overfitting.

A Prediction was made with a sklearn MLP model, in order to compare the results, they are quite equivalent.

```
from sklearn.neural_network import MLPClassifier
model_mlp = MLPClassifier(activation='relu', alpha=0.001, solver='lbfgs', valid
ation_fraction=0.1,batch_size=1)
mlp = model_mlp.fit(df_training, label_train)
y_predict_mlp = mlp.predict(df_test)
from sklearn.metrics import confusion_matrix
score = mlp.score(df_test, label_test)
print('Score is {}'.format(score))
confusion_mlp = confusion_matrix(label_test, y_predict_mlp)
confusion_mlp = pd.DataFrame(confusion_mlp, columns=['Disease', 'Healthy'], ind
ex=['Disease', 'Healthy'])
print(confusion_mlp)
```

```
Score is 0.725
         Disease  Healthy
Disease       47       19
Healthy       14       40
```