

MOOC REPORT

Deep Learning Specialization

Achieved by:

Jean Eudes Ayayi AYILO

Academic year : 2019-2020

Magistère 2

September 2020

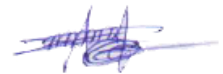
NON-PLAGIARISM UNDERTAKING

I, AYILO Jean Eudes Ayayi, declare being fully aware that plagiarism by copying documents or a portion of a document published in all forms and media, including online publications, constitutes copyright infringement and related rights, as well as outright fraud.

Consequently, I hereby undertake to quote all sources and authors that I have used to write my mooc report and its appendices.

Date: 09/09/2020

Signature:



AYILO Jean Eudes Ayayi

Acknowledgement

I would like to thank all those who contributed directly or indirectly to the realization of this report. I am grateful to:

- ✧ The AMSE director, Tanguy van Ypersele
- ✧ The pedagogical head of the master 1 and magisterium 2, Céline Poilly
- ✧ Pedagogical secretary of the master 1 and magisterium 2, Sarah Wuillemot

Contents

List of figures and tables.....	3
Introduction.....	4
Chapter 1: Description of the mooc.....	5
1.1 General presentation.....	5
1.2 Neural network and Deep learning.....	6
1.2.1 Definition and examples.....	6
1.2.2 General structure of a multi-layer single neural network (perceptron)	7
1.2.3 Forward propagation, activation function, loss function.....	8
1.2.4 Backward propagation and optimization algorithms: Gradient descent and its refinement	10
1.2.5 Vanishing or Exploding gradient	12
1.2.6 Parameters initialization.....	13
1.2.7 Batch-normalization	13
1.2.8 Training set, validation or development set, test set, bias and variance.....	14
1.2.9 Solve bias and variance problems	15
1.2.10 Variance correction: L2 Regularization, Dropout regularization and Early Stopping	16
1.2.11 Synthesis.....	17
1.2.12 About the practice in the first three courses.....	18
1.3 Recurrent Neural Network (RNN)	18
1.3.1 Scope of application of RNN	18
1.3.2 Definition and basic operation	19
1.3.3 Calculation of activations and outputs in a basic RNN.....	20
1.3.4 Backward propagation through time	20
1.3.5 The different types of recurrent neural networks	21
1.3.6 Vanishing or Exploding gradient problem in basic RNN	22
1.3.7 Long Short Term Memory (LSTM)	22
1.3.8 Bidirectional RNN (BRNN) and Deep RNN (DRNN)	23
1.3.9 Word embedding (Word2vec, Glove) and cosine similarity	24
1.3.10 The Many-to-many mode (Case $Tx \neq Ty$) or sequence to sequence model.....	25
1.3.11 Beam search	26
1.3.12 Attention model.....	26
1.3.13 About the practice in the RNN course.....	28
1.4 Convolutional Neural Network (CNN)	28
1.4.1 Scope of application of CNN	28
1.4.2 Structure of a convolutional network: the different types of layers	28

1.4.3	Advantages of CNN	33
1.4.4	Residual Network (ResNet).....	33
1.4.5	Inception Network.....	33
1.4.6	Face verification and face recognition	34
1.4.7	Siamese Network.....	34
1.4.8	About the practice in the CNN course	36
Chapter 2:	Applications.....	37
2.1	Application 1: Sentence textual similarity	37
2.1.1	Data description.....	37
2.1.2	Methodology	38
2.1.3	Results and Limits	40
2.2	Application 2: Daily News for Stock Market Prediction	41
2.2.1	Data description.....	42
2.2.2	Methodology	43
2.2.3	Results and Limits	44
Conclusion		47
Bibliography		48
APPENDIX		a

List of figures and tables

Figure 1: Illustration of a shallow perceptron	7
Figure 2: Forward propagation in one unit of a perceptron.....	9
Figure 3: Dropout regularization.....	16
Figure 4: Basic model of an RNN.....	19
Figure 5: Types of recurrent artificial neural networks.....	21
Figure 6: An LSTM unit.....	23
Figure 7: Many-to-many (Case $Tx \neq Ty$).....	25
Figure 8: Attention model	27
Figure 9: 2D convolution in CNN (cross-correlation)	30
Figure 10: Max pooling and Average pooling	31
Figure 11: An example of simple feed-forward CNN like LeNet 5 (LeCun et al., 1998).....	32
Figure 12: Siamese network with binary classification.....	35
Figure 13: Bar plot of entailment_jugment variable	38
Figure 14: Confusion matrices	40
Figure 15: Bar plot of Label variable	43
Figure 16: Words clouds	45
Figure 17: ROC Curves.....	46
Table 1: Illustration of the fields of activity of the recurrent networks.....	18
Table 2: Brief overview of the SICK dataset	37
Table 3: Statistical report for textual similarity classification using Siamese network with bidirectional LSTM and CNN.....	40
Table 4: Illustration of some entailment and neutral pairs.	41
Table 5: Head of the CombinedNewsDJIA dataset after shifting Label column	42
Table 6: Statistical report for stock market prediction using different models based on daily news.....	44

Introduction

Over the last two decades, the term artificial neural networks has often caused a media frenzy, partly because of the accuracy of their prediction and the image of a kind of artificial brain capable of surpassing that of humans, to which it refers in the popular imagination. Their fields of application are many and varied. Indeed, whether it be facial or voice recognition, computer vision (necessary in autonomous vehicles), natural language processing, text processing (e.g. sentiment analysis), classification or even recommendation systems, predictions in finance and many other tasks in supervised and unsupervised learning, very convincing results have been obtained thanks to these neural networks. They are indeed the building blocks of deep learning, which is a set of learning methods that attempt to model data with complex architectures combining different non-linear transformations.

Thanks to the ability of Deep learning to produce good results, more and more companies are using it in their systems, which in part is increasingly expanding the job opportunities for Data scientists, for example. These latter must have a good knowledge in the field to benefit the company. In order to take our first steps in this field we have turned to a mooc, which can give us the basis for Deep Learning. Therefore, in the absence of an internship, we followed the mooc entitled: Deep Learning Specialization. The rest of this report is structured in 2 chapters: the first aims to present the content of the mooc, the competences developed and the areas in which these competences could be useful, and the second consists of an application of the notions acquired during this mooc.

Chapter 1: Description of the mooc

1.1 General presentation

The mooc Deep learning Specialization aims to provide the foundation in deep learning. It teaches how to build different types of neural networks for the purpose of supervised learning, starting from standard networks or perceptrons that have a relatively less dense architecture to denser architectures such as CNN (Convolutional Neural Network) and RNN (Recurrent Neural Network). Presented by professor Andrew Ng from Stanford University in video, this specialization is available on the coursera platform at this link <https://www.coursera.org/specializations/deep-learning>. It consists of 5 courses, both theoretical and practical. Each course lasts from 2 to 4 weeks, but one can progress at his own pace. At the end of each week, there is a quiz in order to test if the notions covered during the week are understood, and one or two python programming exercises. It should be noted that it is not a question of coding from beginning to end during these programming exercises, but rather of completing certain parts (notably the gaps known as "None") of the code already written. Here is an illustration:

```
def neutralize(word, g, word_to_vec_map):
    """
    Removes the bias of "word" by projecting it on the space orthogonal to the bias axis.
    This function ensures that gender neutral words are zero in the gender subspace.

    Arguments:
        word -- string indicating the word to debias
        g -- numpy-array of shape (50,), corresponding to the bias axis (such as gender)
        word_to_vec_map -- dictionary mapping words to their corresponding vectors.

    Returns:
        e_debiased -- neutralized word vector representation of the input "word"
    """

    ### START CODE HERE ###
    # Select word vector representation of "word". Use word_to_vec_map. (~ 1 line)
    e = None

    # Compute e_biascomponent using the formula given above. (~ 1 line)
    e_biascomponent = None

    # Neutralize e by subtracting e_biascomponent from it
    # e_debiased should be equal to its orthogonal projection. (~ 1 line)
    e_debiased = None
    ### END CODE HERE ###

    return e_debiased
```

Source: Deep Learning Specialization (cours 5), Notebook: Operations_on_word_vectors_v2a

The main packages used in this specialization are NumPy, TensorFlow and Keras of Python. The detailed program for this specialization can be found in Appendix 4 and certifications are in Appendix 5. Here are the different courses:

- ✧ **Course 1: Neural network and deep learning**
- ✧ **Course 2: Improving deep neural networks: Hyperparameter tuning, Regularization and Optimization**
- ✧ **Course 3: Structuring machine learning project**
- ✧ **Course 4: Convolutional Neural Network (CNN)**
- ✧ **Course 5: Recurrent Neural Network (RNN)**

First in this chapter, we will look at perceptrons, then we focus on the recurrent networks and we end up with the convolutional networks.

1.2 Neural network and Deep learning

In this section we attempt to summarize the key concepts developed in the first three courses.

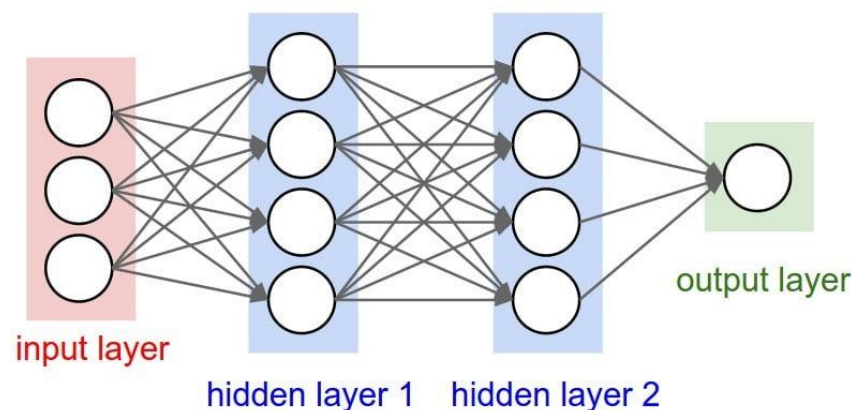
1.2.1 Definition and examples

An artificial neural network is an interconnected assembly of simple elements called processing units or nodes whose functionality vaguely alludes to that of a biological neuron. Each of these nodes receives an input from nodes preceding it and calculates a generally non-linear function, the result of which, the output, is transmitted to other nodes and weighted by weights relative to these nodes. This process is repeated many times in order to update these weights and thus allows the network to learn how to match as many of the input variables as possible to an output variable. For example, for the purpose of classification (or more generally supervised learning), such a network can be provided with variables about clients who have taken out a loan from a bank and the target variable that indicates whether or not the loan has been repaid (this sample of clients is the training sample), so that the network can train itself to recognize good or bad clients and be able to predict whether or not the clients (in a test sample) are good clients or not. A neural network can also be used to recognize manually written numbers (a classic example of LeNet 5), or for regression purposes, for example to determine the price of a real estate asset based on its characteristics.

1.2.2 General structure of a multi-layer single neural network (perceptron)

Neural networks are generally like computational graphs, i.e. instead of representing a model in an equation, or system of equations, models are represented as directed graphs. The perceptron is the simplest of the neural networks that process structured or tabulated data. At the very beginning of a perceptron and of artificial neural networks in general, there is a first layer, called the input layer, which represents all the variables available on the learning sample (in the example of the previous regression, the input variables can be: area, number of rooms, distance from the city centre, age of the house) and at the end of the network there is an output layer which gives an estimate of the target variable (the price of the house, for example). Between these two layers are generally found the hidden layers, which are each made up of nodes called hidden units calculating activation functions. Each node of a hidden layer is connected to all the nodes of the past layer. The number of hidden layers determines the depth of the network, so it is called a deep neural network if there are several hidden layers and a shallow neural network if there are fewer hidden layers. The deeper the network is, the more complex the operation it can perform. In the diagram below, the perceptron has $L=3$ layers (two hidden layers, and an output layer). The input layer has only 3 inputs variables.

Figure 1: Illustration of a shallow perceptron



Source: Neural Networks and Introduction to Deep Learning, wikistat

1.2.3 Forward propagation, activation function, loss function

Forward propagation

To estimate the target variable, each node performs two operations: it calculates a linear combination of the responses from the nodes (or neurons) of the previous layer by assigning a weight to their response, then calculates a function (activation function) of this combination, the result of this function is transmitted to the following nodes and this process continues until the last layer where the output is estimated and evaluated using a loss function to be minimized in order to bring the predicted value closer to the actual value of the target variable. This process of calculating the activations of each node and calculating the value of the cost function is called: **forward propagation**.

Activation function

Basically, there are four types of activation function: the sigmoid or logistic function, the hyperbolic tangent function, the ReLU (Rectified Linear Unit) function and the Leaky ReLU function:

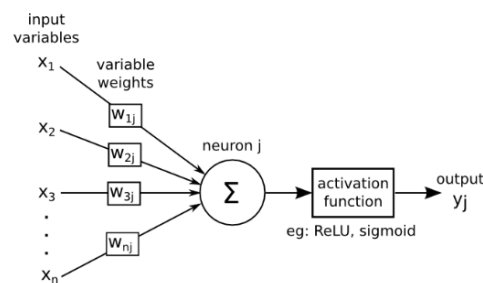
- The sigmoid or logistic function: $\sigma(z) = \frac{1}{1+e^{-z}}$; often used for the output layer in a binary classification. For several classes, the softmax function is used: $\sigma(z_j) = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$ with z a K dimensional vector.
- Hyperbolic tangent function: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, often used in the intermediate layers, its value is between -1 and 1, which makes it possible to center the data in some extents. But like the logistic function, it has the disadvantage of having a derivative which becomes very small for very high or very low values of z .
- ReLU function: $\text{ReLU}(z) = \max(z; 0)$, widely used in hidden layers, or for the output layer in the case of a prediction of a continuous variable for example. Having a derivative always equal to 1 except in 0, it makes it possible to mitigate the disadvantage of the two preceding functions.

- Leaky ReLU function: $\text{LReLU}(z) = \max(0, 0.01; z)$. 0.01 is a hyperparameter to be chosen.

Like the ReLU, it is intended to correct for the fact that the 0 derivative of ReLU is 0.

For a node j in the first hidden layer (any hidden layer is noted l , for the first layer $l=1$), we can observe that in the following figure, the node weights each entry by its weight and then adds a constant, which is shown by the sum symbol : $Z_j^{[l]} = \sum_{k=1}^n w_{kj}^{[l]} * x_k + b_j^{[l]}$. Activation is then calculated by applying an activation function and the output or activation of this node is obtained.: $a_j^{[l]} = f(Z_j^{[l]})$, where f is the logistics function or ReLU in the following figure:

Figure 2: Forward propagation in one unit of a perceptron



Source: <http://andrewjamesturner.co.uk/ArtificialNeuralNetworks.php>

In general, the weights or coefficients, the different linear combinations and activations at a given l are denoted $W^{[l]}, A^{[l]}, Z^{[l]}$ and calculated as follows in the forward propagation phase:

$$\begin{cases} Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]} \\ A^{[l]} = f^{[l]}(Z^{[l]}) \end{cases}, \text{ where } A^{[l-1]} \text{ is the activation of the previous layer, especially for the}$$

first layer $l = 1$, and $A^{[l-1]} = A^{[0]}$ is in this case the whole training sample X . $f^{[l]}$ is the activation function for the layer l . For a network of L layers, the activation of the last layer, the L -th, is noted $A^{[L]}$ and is nothing more than the prediction of the target variable. Initially the parameters $W^{[l]}$ and $b^{[l]}$ of each layer are initialized randomly.

Cost function

As mentioned above, the target variable is estimated and evaluated using a cost function to be minimized in order to bring the predicted value closer to the actual value of the target variable, i.e. to increase accuracy. The type of cost function depends on the nature of the target variable

but also on the objective that one aims for the neural network. In the case of the classification for a qualitative variable, the principle of maximum likelihood is used, but the opposite of the log likelihood is minimized. We then have a cost function on the training sample of size n with y as the variable to be predicted:

$J(W, b) = \frac{1}{m} \sum_{i=1}^m L(\widehat{y}^{(i)}; y^{(i)})$ with $L(\widehat{y}^{(i)}; y^{(i)}) = -[y^{(i)} \log(\widehat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \widehat{y}^{(i)})]$ for a binary classification and $L(\widehat{y}^{(i)}; y^{(i)}) = -\sum_{j=1}^K y_j^{(i)} \log(\widehat{y}_j^{(i)})$ when there are more than two categories. $L(\widehat{y}^{(i)}; y^{(i)})$ is the loss function, it is a measure of entropy, it indicates the extent to which the prediction $\widehat{y}^{(i)}$ is close to the actual value $y^{(i)}$ for a given observation i . It is therefore necessary to be able to find the parameters W and b such that the cost function $J(W, b)$ is minimal.

1.2.4 Backward propagation and optimization algorithms: Gradient descent and its refinement

Backward propagation

Once the forward propagation (calculation of activations) has been performed and the cost function calculated, the derivative of this function is calculated with respect to the parameters W and b . Thus one calculates the derivative of J with respect to the parameters $W^{[L]}$ and $b^{[L]}$ of the last layer, then one calculates the derivative of J with respect to the parameters $W^{[L-1]}$ et $b^{[L-1]}$ of the penultimate layer. We proceed in this way until we get back to the derivatives of J with respect to the parameters $W^{[1]}$ and $b^{[1]}$ of the first hidden layer, hence the name back propagation. With these derivatives, we update the values of the parameters W and b of each layer. This updating can be done thanks to optimization algorithms such as the gradient descent, which is the basic optimization algorithm.

Gradient descent or Batch Gradient descendant

The idea of this algorithm is based on the fact that starting from random values for W and b at each step or iteration, we look for the direction that minimizes the most J and so on until we converge to a global minimum. The updating of the parameters $W^{[l]}$ and $b^{[l]}$ of a layer l where **the derivatives of J with respect to $W^{[l]}$ and $b^{[l]}$ are respectively $dW^{[l]}$ and $db^{[l]}$** is done as

follows, for an iteration: $\begin{cases} W_{new}^{[l]} = W_{old}^{[l]} - \alpha dW^{[l]} \\ b_{new}^{[l]} = b_{old}^{[l]} - \alpha db^{[l]} \end{cases}$, avec α the learning rate which is a to be calibrated.

It is often comprised in $]0, 1]$. The number of iterations is also a hyperparameter to be calibrated. The smaller α is, the more iterations are needed to converge. The larger α is, the fewer iterations will be needed, but there is a risk of not reaching the minimum or even exceeding it. For a single iteration, we calculate the forward propagations on all the given observations of the training sample X , calculate the cost function for the whole X , calculate the derivatives and then update the parameters. This process is repeated many times, in order to converge to a minimum of the cost function.

Stochastic Gradient Descent (SGD) and Mini-batch gradient descent

These are two variants of the gradient descent algorithm. While in the batch gradient descent an iteration consists in using the whole training sample to do the forward propagation, calculate the cost function and do the backward propagation to update the parameters, in the SGD, an iteration, called **epoch**, consists in reviewing the whole training set by randomly drawing each time (and without resetting) an observation and performing the forward and backward propagation and updating the parameters. In a single iteration (**epoch**) of the SGD, the parameters are therefore updated as many times as the size of the training sample, or to make it simpler, the gradient is calculated for each observation and then updated relative to the average of these gradients. The SGD is therefore very expensive in terms of calculation, so we prefer the mini-batch gradient descent which works on the same principle but instead of using a single observation, we use a portion (**mini-batch**) of the training sample. The latter is therefore initially divided into equal portions of similar size (**mini-batch size**) with powers of 2: 32, 64, 128, 256 ... The mini-batch size is also a hyperparameter to be calibrated. Using the whole training sample at once can cause the gradient descent algorithm not to converge, or even skip the minimum, hence the idea of evolving step by step by taking parts of the training set.

Gradient descent with Momentum, Root Mean Square propagation (RMSprop), Adaptive moment estimation (Adam)

These algorithms are the refinement of the previous ones, and the most sophisticated is the Adam algorithm which combines both downward gradient with momentum and RMSprop. The basic idea is to include an exponential smoothing to consider past values of the gradient to give an update of the parameters. By smoothing the oscillations of the descending gradient algorithm, these algorithms allow to converge faster in a reduced number of iterations. To lighten the mathematical presentation of this report, we will detail only the descending gradient with momentum as follows:

- Initialization: We pose $VdW^{[l]} = 0$ and $Vdb^{[l]} = 0$; these quantities represent velocities

At the t-th iteration: $t \in \{1, 2 \dots \text{total number of iteration}\}$:

- We calculate the gradients $dW^{[l]}$ and $db^{[l]}$ for the portion (mini-batch) that is being processed
- We then calculate the quantities

$$\begin{cases} VdW_{new}^{[l]} = \beta * VdW_{old}^{[l]} + (1 - \beta) * dW^{[l]} \\ Vdb_{new}^{[l]} = \beta * Vdb_{old}^{[l]} + (1 - \beta) * db^{[l]} \end{cases}, \text{ with } \beta \text{ a hyperparameter to be calibrated,}$$

default value is set to 0.9 generally.

- Update the parameters: $\begin{cases} W_{new}^{[l]} = W_{old}^{[l]} - \alpha VdW^{[l]} \\ b_{new}^{[l]} = b_{old}^{[l]} - \alpha Vdb^{[l]} \end{cases}$

As we can see, building a neural network requires choosing several hyperparameters. They influence the parameters and the results of the model we are trying to build. Furthermore, it can happen that the different optimization algorithms mentioned do not converge or diverge, this is the problem of vanishing or exploding gradient.

1.2.5 Vanishing or Exploding gradient

When the networks are getting deeper and deeper, one of the problems we face is the vanishing or exploding gradient problem. Vanishing gradient (respectively exploding gradient) occurs when

the computed activations and \hat{y} become exponentially very small (respectively very large) as the network gets deeper and deeper. As a result, the gradients of the cost function become exponentially very small (respectively very large). This problem can be partially solved by judicious choice of the initialization of the weights W .

1.2.6 Parameters initialization

There are different ways to initialize the parameters of a network. Set W to 0, initially cause that the $W^{[l]}$ in the different layer will be the same. Set b to 0 does not matter. And, to partially solve the problem of vanishing or exploding gradient, Xavier or He initialization are used. They consist in multiply randomly chosen values by a specific coefficient that consider the number of units of the previous layer to initialize the $W^{[l]}$ of the current layer. We will not report their specific expressions.

1.2.7 Batch-normalization

Just like standardizing input variables before processing them through the network, batch normalization consists in giving the same scale of magnitude to the different outputs of the hidden layers. To do this, before calculating the output $A^{[l](i)}$ of a given observation i in a mini-batch, $Z^{[l](i)}$ must be centered and reduced with respect to the mean and variance of this mini-batch: $Z_{norm}^{[l](i)} = \frac{Z^{[l](i)} - \mu}{\sqrt{\delta^2 + \epsilon}}$, where μ and δ^2 refer respectively to the mean and variance of the $Z^{[l](i)}$ on the mini-batch, ϵ is a small number to avoid a division by 0. However we do not necessarily want the variance to be 1 and the mean to be zero, so we calculate $\widetilde{Z^{[l](i)}} = \tau * Z_{norm}^{[l](i)} + \vartheta$, where τ and ϑ are respectively the new variance and the new mean that all the outputs of the layer l will have before applying the activation function. As a result, τ and ϑ are also parameters to which the optimization algorithms are applied. Batch normalization makes the parameters of the deeper layers more robust to variations in the parameters of the shallower layers of the network. Batch normalization also provides the ability to use a high learning rate, which increases the speed of learning.

1.2.8 Training set, validation or development set, test set, bias and variance

Training set is the one on which the optimization algorithm is run in order to estimate the model parameters. One can try to estimate the parameters of various models with this sample.

validation or development set is the one on which we test the different models designed, in order to designate the one that maximizes the performance evaluation criteria of the models we have chosen. This measure can be, for example, the area under the ROC curve, the F1 score, accuracy and many others.

Test set is one on which the model selected above is tested in order to evaluate it on real-life data in order to obtain an unbiased estimate of the effectiveness of the selected model.

It is important that the validation sample and the test sample have the same distribution, otherwise the assessment will be biased. However, the training set may have a different distribution. Regarding the appropriate subdivision of the data between these three samples, in this new era of big data, training neural networks requires a large amount of data, otherwise the performance will be poor. For example, for a data set of 1,000,000 observations, a subdivision into 98% of the data for the learning sample, and the rest shared between the validation sample and the test sample (1% and 1%) is more appropriate than a division into 60%, 20% and 20%. The latter is only appropriate when relatively little data is available.

Bias and variance

Before addressing the notions of bias and variance, we need to define the notion of **Bayes' optimal error**. The Bayes error is a theoretical limit that represents the smallest error rate with respect to which a human or a system cannot do better in predicting a target variable Y , given the explanatory variables X . For example, if a general practitioner has a 3% chance of being wrong when making a medical diagnosis based on the image of a scanner, while a group of expert physicians has a 0.5% chance of being wrong, then 0.5% is considered the bias error, because it is the smallest level of error with respect to which one can no longer do better. This measure allows us to realize the shortcomings that the neural network, or in general a model has in the prediction.

Indeed :

- A very high classification error rate on the learning sample (e.g. 15%) compared to the Bayes error (e.g. 3%), and a classification error rate on the validation sample not too far (e.g. 17%) from the error on the training set means that the model more suffers from a **problem of bias**. This means that the model is not able to predict well on the sample that was used as a basis for learning; this is called **underfitting**.
- If the difference between the classification error rate on the validation sample (e.g. 17%) and the classification error rate on the learning sample (e.g. 5%) is greater than the difference between the Bayes error (e.g. 3%) and the classification error rate on the training set, then it means that the constructed model more suffers from a **variance problem**. This means that the model is not able to generalize what it learns from the learning sample to the validation sample; this is called overfitting.

It is also possible that we have a problem of bias and variance at the same time: the difference between the classification error rate on the validation sample (for example 30%) and the classification error rate on the learning sample (for example 15%) is high and also the difference between the Bayes error (for example 3%) and the classification error rate on the learning sample is high.

1.2.9 Solve bias and variance problems

To correct the bias problem, the following solutions are available:

- Increase the network: e.g. increase the number of hidden layers and increase and/or the number of nodes per layer
- Increase the number of iterations of the optimization algorithm
- Use advanced algorithms
- Or even change the architecture (try convolutional or recurrent networks).

To correct the variance problem, the following solutions are available:

- Obtain more data if possible; data augmentation techniques are available

- Proceed to regularization or dropout or early stopping
- Or change the architecture entirely

1.2.10 Variance correction: L^2 Regularization, Dropout regularization and Early Stopping

L^2 Regularization

It consists in adding a term to the cost function which aims at decreasing the value of the $W^{[l]}$ of the different hidden layers. The new cost function is then written as follows:

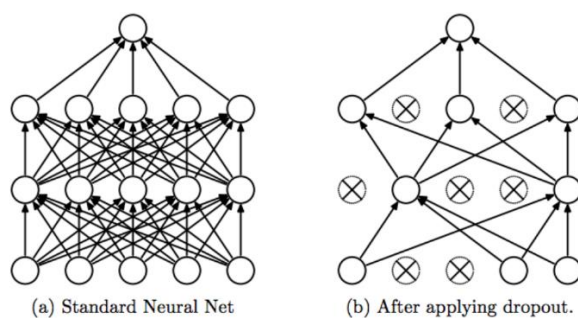
$J(W, b) = \frac{1}{m} \sum_{i=1}^m L(\widehat{y}^{(i)}; y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|W^{[l]}\|_F^2$, with $\|W^{[l]}\|_F^2$ the Frobenius norm of the weight matrix $W^{[l]}$, which is nothing more than the sum of the squares of the coefficients of the matrix.

The λ term acts as a penalty on the value of the coefficients of the parameter matrices $W^{[l]}$. The higher it is, the more it will have the effect of reducing the value of the coefficients of the $W^{[l]}$ matrices, thus avoiding over-fitting. λ is therefore also a hyperparameter to be calibrated.

Dropout regularization

The purpose of dropout regularization is to put a probability on each node of a layer (and independently of the other nodes of the same layer) so that the output of this layer is zero. Each node can thus be randomly deleted according to a probability. This has the effect of shrinking the network so that the network is not overfitting.

Figure 3: Dropout regularization



Source: Neural Networks and Introduction to Deep Learning, wikistat

In this figure, the knots with a cross are those for which the activation has been set to 0. The probability with which the activation of a node will be zero is also a hyperparameter to be calibrated.

Early Stopping

It is a technique that aims to track the evolution of the error rate on the validation sample over iterations in order to take the number of iterations that minimizes the error rate on the validation sample. However, this method has the disadvantage of not allowing the cost function to be minimal before stopping the iteration, which means that a bias problem can be created by trying to solve the variance problem. Therefore, the dropout is most often preferred or the L^2 regularization.

1.2.11 Synthesis

All along this section we can notice that, training a neural network requires to decide on the choice of several hyperparameters. It is not very appropriate to carry out a grid search, even more so when one does not have high technical means; but it is rather necessary to choose randomly (because it is impossible to know in advance the values of the hyperparameters that will work better), this allows to better explore all the possible values. These hyperparameters also have their order of importance: first we can find the learning rate, then the other hyperparameters linked to the optimization algorithms (such as the β of Momentum) as well as the size of the mini-batches and the number of iterations. Finally, we can consider the hyperparameters related to the network structure: the number of hidden layers, the number of nodes per layer, and many others.

We also insisted on the fact that training a neural network requires a lot of data. In the case where we do not have enough data, we can resort to **transfer learning**, which consists of transferring the knowledge acquired on task A for which we had a lot of data to task B for which we have less data. The idea is to take the network developed for task A and use these parameters, and remove certain layers and then complete other layers that are more in line with the objectives of task B. Data augmentation techniques can also be used to have a larger learning sample.

All the optimization algorithms that we have mentioned as well as batch normalization and regularization techniques remain valid in other neural network architectures. Finally, we can say that standard neural networks are adapted to structured data, and can therefore be used to predict, for example, in the banking sector whether to grant a loan or not, in logistics (predicting transit time), in targeted online advertising, etc. They cannot be used directly for unstructured data, for this other type of networks are needed, but these standard networks can form compartments in these other types of neural networks.

1.2.12 About the practice in the first three courses

We build a multilayer perceptron (with the gradient descent and its refinement, initialization and regularization) step by step using NumPy package and apply it to cat detection in images for example. We have seen how to perform error analysis, how to check and solve mismatched data distribution in some cases of study. We also get an introduction to TensorFlow. Thanks to course 3 we also practice decision-making as a machine learning project leader. This provides industry experience that we might otherwise get only after years of machine learning work experience.

1.3 Recurrent Neural Network (RNN)

1.3.1 Scope of application of RNN

This type of neural network is particularly adapted to sequential data, i.e. data that show changes over time. It can handle certain types of unstructured data such as text, sound or video as well as structured data such as time series. In the following table we have grouped for each type of task the nature of the input variable and the variable to be predicted (output).

Table 1: Illustration of the fields of activity of the recurrent networks

Tasks	Input (X)	Output (y)	Objective
Speech recognition	Audio	Text	Transcribe audio to text
Music generation	Nothing, or just an integer	A sound (sequence of musical notes)	Randomly compose music
Time series prediction	Times series	Times series	Predict future values of a series over a time horizon
Sentiments classification	Text	categorical variable	For example, to give the feeling of the clients based on their written opinions
Machine translation	Text in language A	Text in language B	Translate from French to English for example

Name entity recognition	Text	Qualitative variable or text	Identify in a text, expressions that are names of people, companies, or things.
DNA sequence analysis	DNA sequence	categorical variable	Determine the type of protein in a DNA sequence
Recognition of action	Video	categorical variable	Recognize the action that is done in a video sequence

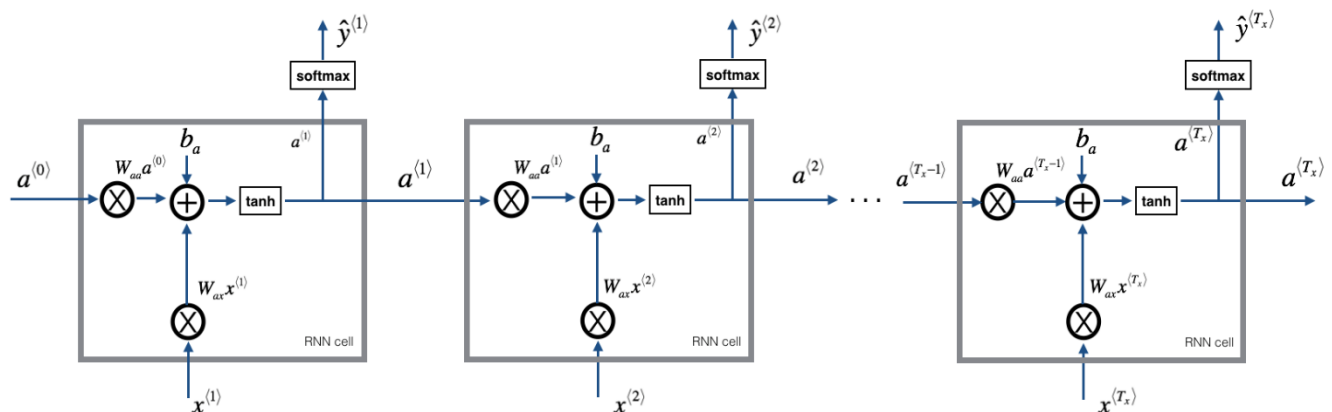
Source: Author of this report

Recurrent networks are often used in Natural Language Processing (NLP) tasks where input sequences are unstructured data and need to be converted into structured data via features extraction

1.3.2 Definition and basic operation

As its name suggests, a recurrent artificial neural network is a network of artificial neurons with recurrent connections. In fact, everything happens as if we had a single hidden layer that we repeat in a loop, so that this layer receives as inputs the output it had calculated and a new input to give a new output.

Figure 4: Basic model of an RNN



Source: <http://www.easy-tensorflow.com/tf-tutorials/recurrent-neural-networks/vanilla-rnn-for-classification>

Suppose our goal is to identify in a sentence X as: « Jeanne visite l'Afrique en Septembre », expressions that are names of persons (Name Entity Recognition). The length of this sequence is $T_x = 6 = \text{number of words}$ (the length of the input is noted T_x and that of the output is T_y). According to this figure, the input layer is nothing more than this sentence where every word $x^{<t>}$ is represented as a vector in order to be able to apply transformations. The hidden layer is

represented by the set of gray rectangles where the activation functions are calculated and the output layer is represented by the softmax layers where the probabilities that each word has to be a person's name or not will be calculated (if for example the probability is $> 0,5$ for a word $x^{<t>}$ we will say that it is a name, and in this case the output o_t will be 1). Since it is the same unit that is repeated in the hidden layer, the parameters matrices W_{ax} , W_{aa} , and W_{ay} are respectively identical for all units (i.e. W_{ax} is the same for the first unit, as for the second unit, up to T_x , the same for W_{aa} , and W_{ya} respectively). Compared to standard networks, a recurrent network has the advantage of processing inputs of any length without having a high number of parameters because the same parameters are used all along the network. Moreover, such a network considers past information. But it requires more computing time.

1.3.3 Calculation of activations and outputs in a basic RNN

In the previous example, to calculate the activations the recurrent network proceeds as follows: each specific unit t of the hidden layer, receives as input the vector representing the word $x^{<t>}$ as well as the activation coming from the previous unit. We add the multiplication of the vector $x^{<t>}$ by a matrix of parameters W_{ax} and the multiplication of the activation from the previous unit (also a vector) by a matrix W_{aa} and we add a constant term b_a . An activation function is applied to this sum, which gives the activation of the current specific unit t . This activation is transmitted to the next unit and is also multiplied by a matrix W_{ya} , multiplication to which an activation function is applied to calculate the output $y^{<t>}$. Mathematically, we have:

$$\begin{cases} a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \\ y^{<t>} = f(W_{ya}a^{<t>} + b_y) \end{cases} \text{ that we rewrite in: } \begin{cases} a^{<t>} = g(W_a[a^{<t-1>}; x^{<t>}] + b_a) \\ y^{<t>} = f(W_y a^{<t>} + b_y) \end{cases},$$

with

$$W_a = (W_{aa} | W_{ax}).$$

1.3.4 Backward propagation through time

As in the standard network, we proceed to the back propagation, starting from the end of the network and going back to the beginning of the network to compute the derivatives of the cost (or loss) function with respect to the different parameters W and b , and thus use an optimization

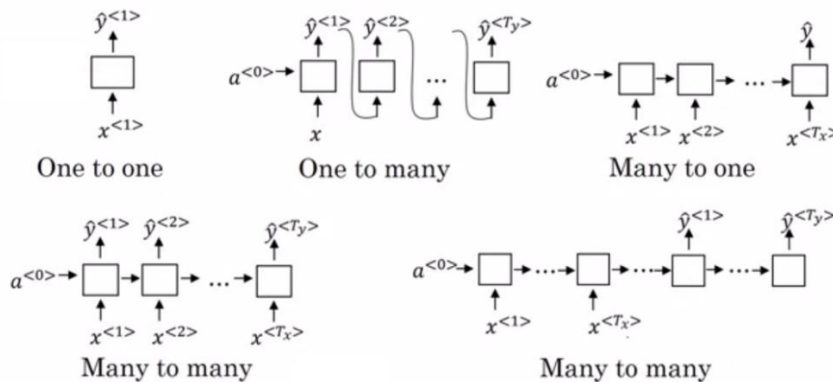
algorithm like those developed in the previous section to minimize the loss function. Note that the derivative with respect to the parameters W and b implies the computation of the derivative with respect to the activations $a^{<t>}$, which is done by backward propagation through time, hence the name backward propagation through time. As an illustration, the loss function in the case of the previous example of recognizing named entities for a single sentence is calculated as follows:

$L(\hat{y}, y) = \sum_{t=1}^T L_t(\hat{y}_t, y_t)$; avec $L_t(\hat{y}_t, y_t) = -[y_t \log(\hat{y}_t) + (1 - y_t) \log(1 - \hat{y}_t)]$ with L_t the loss function for a word in the sequence, \hat{o}_t the probability that word x_t is estimated to be a name.

1.3.5 The different types of recurrent neural networks

In the example concerning named entities recognition that we have used so far, the length of input sequence and output sequence have the same length and each input word has output ($T_x = T_y = T$). However, it is quite possible that the input and output sequences are not the same length, as in the case of machine translation, and that each input does not have an output. These configurations determine the type of network suitable for each task. Thus, we have the following types:

Figure 5: Types of recurrent artificial neural networks



Source : <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>

- ✧ One-to-one: $T_x = T_y = 1$. There is a single vector as input and a single output: this is the case of the standard networks described in section 1.2.

- ✧ One-to-many: $T_x = 1$, and $T_y > 1$. There is only one vector for the input and one vector for the output. This is the case for music generation. The first output is used as input to the next unit and so on.
- ✧ Many-to-one: $T_x > 1$, and $T_y = 1$, there are several vectors for the input and only one vector for the output, which is the case for the sentiment classification.
- ✧ Many-to-Many: Case $T_x = T_y > 1$. Here, each input has an output, so the input and output have the same length. This is the case of name entity recognition.
- ✧ Many-to-Many: Case $T_x \neq T_y$, the length of the input sequence, is not necessarily the same as the length of the output sequence. In addition, the entire input sequence is reviewed before producing the output. This is the example of machine translation.

1.3.6 Vanishing or Exploding gradient problem in basic RNN

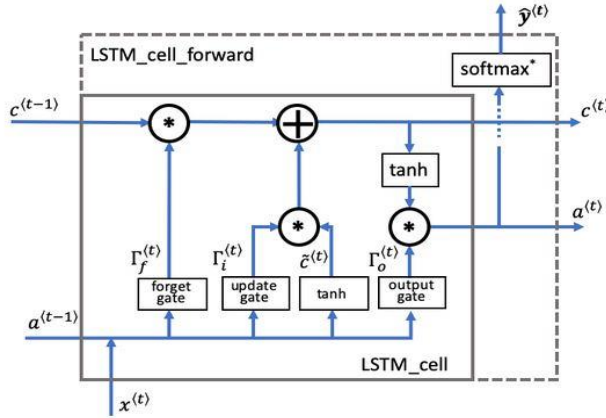
This problem is due to the fact that basic RNN does not have the possibility to take into account long term dependencies (for example a basic RNN will not be able to recognize how to tune a verb in a translation, when the subject is very far from the verb). Moreover, this problem is related to the fact that the deeper the network is (i.e. the more units in the hidden layer), the more the derivative of the loss function with respect to the parameters of the first units of the hidden layer tends to decrease or even cancel out or increase excessively. In the case of an exploding gradient problem, an upper bound can be set on the gradient value so that if the gradient exceeds this limit, the gradient is assigned the value of the upper bound. This method is called clipping. On the contrary, in the case of gradient cancellation, a more powerful architecture is required than that of basic RNN. Hence the GRU (Gated Recurrent Unit) and LSTM models. The LSTM model is more general and more powerful than the GRU model but requires more computing time. Being more general, we will only present the LSTM model, the GRU model can be deduced from it.

1.3.7 Long Short Term Memory (LSTM)

Unlike a basic RNN unit which has only one layer to compute the activation $a^{<t>}$, an LSTM unit has four layers (including three "gates"): a layer or "gate" to compute the proportion of the past activation that must be forgotten, a layer to compute the current information that is a

candidate for the replacement of the past activation, a layer or " gate " to compute the proportion of the past activation that must be replaced by the candidate and finally a layer or " gate " to compute the proportion of the past activation that must be taken into account in the calculation of the present output. Schematically, an LSTM unit is presented as follows:

Figure 6: An LSTM unit



Source: https://datascience-enthusiast.com/DL/Building_a_Recurrent_Neural_Network-Step_by_Step_v1.html

The equations governing an LSTM unit are as follows:

$$\Gamma_f^{<t>} = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f), \sigma \text{ is the logistic function, } \Gamma_f^{<t>} : \text{forget gate}$$

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c); \tilde{c}^{<t>} \text{ candidate for the replacement of the past activation}$$

$$\Gamma_i^{<t>} = \sigma(W_i[a^{<t-1>}, x^{<t>}] + b_i), \Gamma_i^{<t>} : \text{update gate}$$

$$c^{<t>} = \Gamma_f^{<t>} \odot c^{<t-1>} + \Gamma_i^{<t>} \odot \tilde{c}^{<t>}, \text{ where } \odot \text{ is the Hadamard's product or the element-wise product}$$

$$\Gamma_o^{<t>} = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o), \Gamma_o^{<t>} : \text{output gate}$$

$$a^{<t>} = \Gamma_o^{<t>} \odot \tanh(c^{<t>})$$

$$\hat{y}^{<t>} = \text{softmax}(W_y a^{<t>} + b_y)$$

1.3.8 Bidirectional RNN (BRNN) and Deep RNN (DRNN)

While a simple RNN network calculates activations from left to right, a bidirectional network calculates activations in both directions. This allows past and present information as well as future information to be considered in the prediction for the present. Therefore, the entire sequence needs to be available and processed before giving the outputs.

A simple RNN network has only one hidden layer. A deep RNN will have two or three hidden layers of stacked LSTM units. This allows more complex tasks to be performed.

1.3.9 Word embedding (Word2vec, Glove) and cosine similarity

As mentioned above in the example of named entity recognition, words in sentences used as input to the recurrent network need to be represented in vector form, in order to be processed. There are generally two ways to represent words: one-hot-encoding, which consists in considering a vocabulary of words and representing each word by a vector of equal size to the dictionary one, it contains only 0s and then 1s at the index equal to the word position in the dictionary. Under this representation the words have no link between them. The second method is the encoding of words, which takes into account the similarity between words. To obtain this encoding, a neural network can be trained on a large corpus of text using methods such as Word2vec or Glove (Global vector for word representing). At the end of the training, this network produces a matrix where each column represents a word.

Word2vec

It is a method of predicting the words that may be in the neighborhood (context) of a word or predicting from a context (neighborhood) the word that one may have. This can be done using the CBOW (Continuous Bag Of Word) or the skip-gram.

- ✧ The skip-gram learns the encoding of words by trying to predict the words that may be in the neighborhood of a word. For this, given a word in the middle of a sentence, the neural network will calculate the probability for each word in the vocabulary to be the neighboring word (for example to be among the two words before the chosen word and to be among the two words after the chosen word). For example, in French it will have to find: the, cat, the, mouse; when we provide: __ catches __.
- ✧ The CBOW, on the other hand, seeks to determine a target word given a context. To do this, the neural network is provided with the words surrounding the target word and the neural network will determine the probability for each word in the vocabulary to be the

target word sought. For example, in: the cat _ the mouse, it will have to find that the word with the highest probability is: catches. (Wikipedia)

Glove

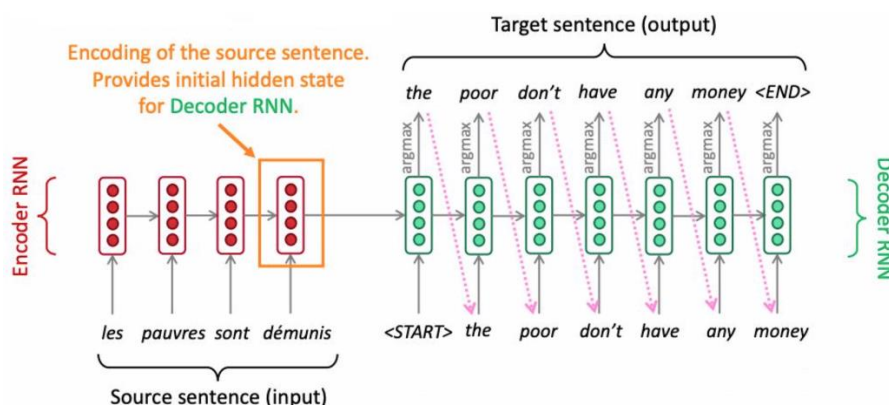
Breifly we can say that Glove is an unsupervised learning algorithm which rather than to be limited to local statistic as Word2vec relies on words co-occurrence (so, global statistic) in a large corpus to obtain the words embedding. The model is trained with a kind of least square lost where the goal is to minimize the square difference between the co-occurrence of words and the scalar product of their embeddings vector that are sought.

Cosine similarity

It is a measure of similarity between words that is given by the cosine of the angle between the representative vectors of the two words. For two words represented by their vectors resulting from the encoding w_1 and w_2 ; $\text{similarity} = \frac{\langle w_1, w_2 \rangle}{\|w_1\| \cdot \|w_2\|}$. This measure can be used, for example, to measure how similar two documents are; or when looking for analogies: “Man is to woman as king is to ?” : queen.

1.3.10 The Many-to-many mode (Case $T_x \neq T_y$) or sequence to sequence model.

Figure 7: Many-to-many (Case $T_x \neq T_y$)



Source: <https://towardsdatascience.com/introduction-to-rnns-sequence-to-sequence-language-translation-and-attention-fc43ef2cc3fd>

It is a type of RNN where input and output are sequences, and is suitable for tasks such as machine translation, trigger word detection in audio, or image captioning. Such a network has two parts: the first is the encoder part, which is responsible for extracting a set of characteristics from the input in the form of an x-vector, then this vector is transmitted to the second part of the network, which is the decoder part, responsible for generating the output, conditional on this x-vector. Thus for the translation of the sentence: “Les pauvres sont démunis”, the sentence will be reduced to a vector x by the encoder part and will be used by the first unit of the decoder part to produce the first English word by looking for the English word that has the highest probability in view of x, this word will be transmitted to the second unit that will determine the second English word by looking for the English word that has the highest probability in view of x and the first English word generated. We proceed as follows and obtain the translation "The poor don't have any money".

1.3.11 Beam search

This is an algorithm used in speech recognition and machine translation to find the most likely sentence y given an x input. In the previous paragraph we said that the decoder will look for the most probable English word knowing x. But in the case of the beam search, for the first English word, we look for the B (B is a hyperparameter equal to the number of words to be considered) words in the English vocabulary that are the most probable knowing x. For each of these B words, we determine the B most probable words that could be the second word after this first one knowing x and the first word. We proceed in this way until we obtain the phase with the highest probability. This algorithm requires a lot of computation, because at each step the decoder part must be multiplied B times. Mathematically, this objective function is maximized

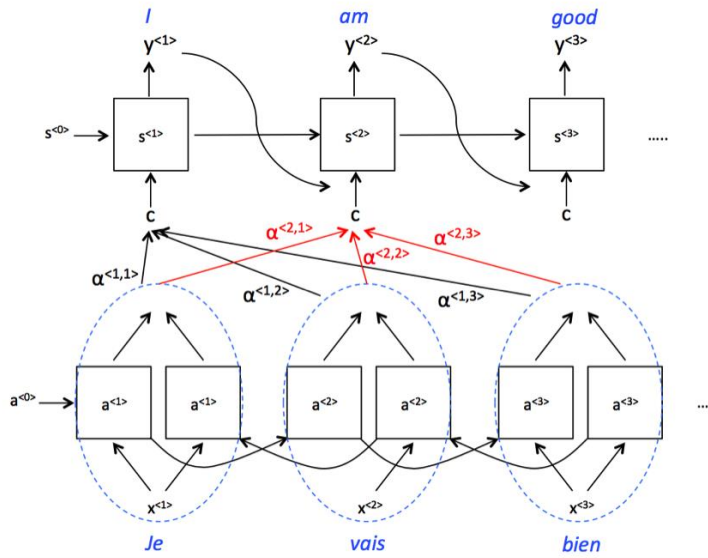
objective = $\frac{1}{T_y^\alpha} \sum_{t=1}^{T_y} \log [p(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})]$; T_y^α is used as a normalization, to force the algorithm to produce not only short sentences.

1.3.12 Attention model

The many-to-many model described above requires that the entire input sequence be processed before output estimation. However, it is more judicious when a long text must be translated from

French to English, to proceed step by step, taking the text piece by piece, than to memorize all the text before translating it. This is what the attention model does by allowing an RNN to pay attention to specific parts of the input that are considered important, thus improving performance. An attention model has two blocks: a bidirectional RNN block that calculates activations for each word input, and an RNN block that uses these activations to estimate the output. Here is a schematic representation:

Figure 8: Attention model



Source : <https://mc.ai/deeplearning-series-attention-model-and-speech-recognition/>

More precisely, for each activation: $a^{<t'>} = (\vec{a}^{<t'>}, \tilde{a}^{<t'>})$ of the BRNN block is computed a coefficient $\alpha^{<t,t'>}$ which indicates the importance score of the activation $a^{<t'>}$ from the word $x^{<t'>}$ in the estimation of the output $y^{<t>}$. Coefficients $\alpha^{<t,t'>}$ are used to compute a context $c^{<t>}$ which serves as input for the RNN block in the calculation of the output $y^{<t>}$. Note that this RNN block also takes as input the previously generated word $y^{<t-1>}$ as well as the activation $s^{<t-1>}$ resulting from the context $c^{<t-1>}$. We have : $c^{<t>} = \sum_{t'=1}^{T_x} \alpha^{<t,t'>} a^{<t'>}$ with

$$\sum_{t'} \alpha^{<t,t'>} = 1 \text{ and } \alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t''=1}^{T_x} \exp(e^{<t,t''>})}$$

1.3.13 About the practice in the RNN course

We have built step by step a basic RNN and LSTM layer (programming the forward, mainly) with Numpy. In Keras we build a character level language for dinosaurs' names that can generate dinosaurs' names and a model for Jazz improvisation with LSTM, we have also worked on word debiasing and sentiment classification. Furthermore, we apply an attention model which regardless of the form in which a date is found, the model converts this date into the format year-month-day and finally, we build an RNN model for trigger word detection in audio clips.

1.4 Convolutional Neural Network (CNN)

1.4.1 Scope of application of CNN

Convolutional networks are best suited for image processing. They have revolutionized the field of computer vision, especially in tasks such as image classification, facial recognition and object detection (identifying objects in an image and surrounding them to delineate them). Images can sometimes reach very large dimensions, which makes the use of the perceptron ineffective because it requires a large number of parameters. Indeed, an image is represented by an array of 3 matrices: each of them indicates the red, green and blue color intensities of each pixel of the image. Thus an image of size 1000×1000 i.e. 1 megapixel, implies that one must have $3 \times 1000 \times 1000$ input variables, therefore 3,000,000 parameters (without the constant term) just for a single node of the first hidden layer of a simple network that would process this image. Moreover, using a perceptron requires transforming the image into a vector, which therefore leads to distort the initial representation of the image. However, with a convolutional network, fewer parameters will be used, and the tensor representation of the image is preserved. Note also that convolutional networks can also be adapted to sequential data such as texts and time series, and videos.

1.4.2 Structure of a convolutional network: the different types of layers

There are three types of layers in a convolutional network: convolutional layers, pooling layers and fully connected layers.

✓ **Convolutional layers:**

In the perceptron, each node first calculates the scalar product between an input (or the output of a previous layer) and a vector of parameters (Figure 2). Similarly, convolutional layers are layers that compute a convolution product and more precisely a cross-correlation (noted $*$) between the output of the previous layer and a matrix (or more generally a tensor) of parameters called kernel or filter. For example, in dimension 2, if we have an image matrix I of dimension $(H \times W)$ and a filter F of dimension $f \times f$, the cross-correlation consists in dragging the filter matrix F on the matrix I . At each position, we get the cross-correlation between the filter and the part of the image that is currently treated. Then, the filter F moves by a number s of pixels, s is called the stride. By proceeding in this way, we end up with a matrix of reduced size. So as not to end up with a matrix that is too small, we can pad the initial matrix by adding p columns on both sides of the matrix and p rows on both sides as well, these new columns and lines contain only 0. And doing so, the result of the cross-correlation between I and F is the matrix $(F * I)$ with a height

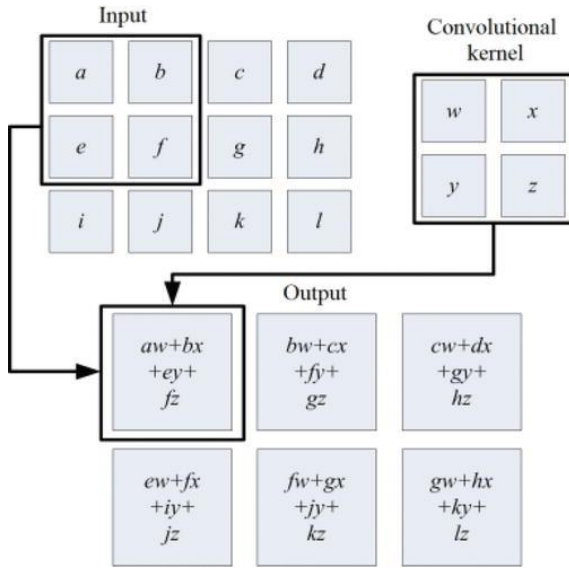
$H' = \left\lfloor \frac{H+2p-f}{s} \right\rfloor + 1$ and a width $W' = \left\lfloor \frac{W+2p-f}{s} \right\rfloor + 1$. If we wish to have the same height and width for the output of the convolution product (this is named same convolution), the input tensor must be padded with a $p = \frac{f-1}{2}$. The calculation formula for the convolution product of image matrix I by a kernel F is:

$$(F * I)(i, j) = \sum_{m, n} F(m, n) I(i + n, j + m)$$

Note that we use the terms convolution product and cross-correlation interchangeably, but it is really the cross-correlation computation which is done in a convolutional layer (the matrix I must be flipped to compute the real convolution product) since the real convolution product involves more complexity.

In the figure below we see the convolution product of a 3×4 matrix by a 2×2 , with no padding : $p=0$ and a stride $s= 1$.

Figure 9: 2D convolution in CNN (cross-correlation)



Source: <https://i2.wp.com/khshim.files.wordpress.com/2016/10/2d-convolution-example.png>

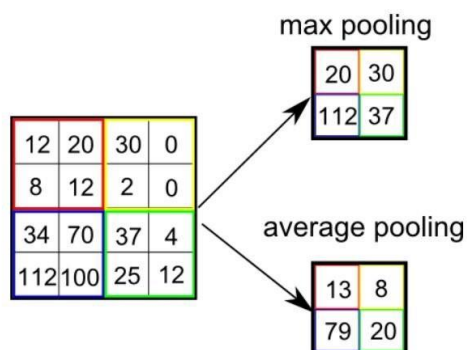
The same principle is applied if I is of one dimension or 3 dimensions. In the latter case, when I has the dimension $(H \times W \times C)$ with C the number of channels or the depth, the filter F must also have C as depth, so F is a $f \times f \times C$ tensor ($f < H$ and W). Then, we compute the element-wise product between F and the first slice of I which has same dimension as F , and we sum all the elements of the tensor which resulting from this element-wise product. Then the filter F moves by s slides, and we restart the operation. Again, we end up with a matrix which dimension is $(H' \times W')$ as mentioned above. If we have many filters in the convolutional layer, for example $n_c^{[l]}$ filters for the layer l , then we stack the different $(H' \times W')$ matrices together to form a $(H' \times W' \times n_c^{[l]})$ tensor. After the convolution product, a bias b is added and the activation is compute: $g((F * I) + b)$, where g is an activation function. The filters F and the bias b of a given convolutional layer are the parameter that will be updated with an optimization algorithm. If in a layer l we have $n_c^{[l]}$ filters of dimension $f \times f \times n_c^{[l-1]}$ (where $n_c^{[l-1]}$ is the number of channels of the output of the previous layer) then the number of parameters for this layer is: $(f \times f \times n_c^{[l-1]} + 1) \times n_c^{[l]}$, the 1 represent the bias parameter. Thus for our introductive example of 1 megapixel image, if we use 32 filters of $5 \times 5 \times 3$ for the first convolutional layer, we will only have $(5 \times 5 \times 3 + 1) \times 32 = 2432$ parameters (including the bias). Note that the convolution

operation is important in edges and features detection in an image, it is the filters which play this role.

✓ Pooling layer

There are two types of pooling layer: max pooling and average pooling layers, but the latter is less used. In the pooling layer there is not a filter or kernel, but each matrix channel in the input tensor is browsed by evolving by sub-matrix of size $f \times f$ with a stride s . For each sub-matrix traversed, the maximum (respectively the average) is calculated for max pooling (respectively average pooling). In doing so, the number of channels of an input tensor is the same as the output tensor of a pooling layer. Thus, if the input tensor is a $(H \times W \times C)$ tensor, then the dimension of the output is $(H' \times W' \times C)$ where H' and W' are computed as mentioned in the convolutional layer paragraph. An advantage of the pooling is that it makes the network less sensitive to small translations of the input images. The figure below, shows a max pooling and an average pooling with $f = 2$ on a 4×4 matrix. There is no padding and the stride $s = 1$.

Figure 10: Max pooling and Average pooling



Source: <https://qph.fs.quoracdn.net/main-qimg-cf2833a40f946faf04163bc28517959c>

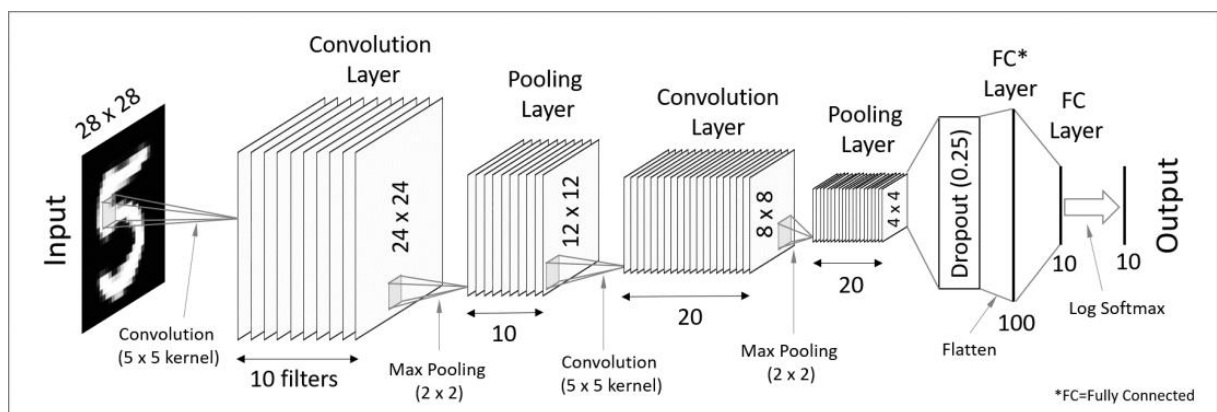
It should be noted that a layer of a CNN can be considered as being a convolutional layer followed by a pooling layer. In this case, the activation function is applied to the pooling result but not on the convolutional product result. A pooling layer will generally be between two convolutional layers.

✓ Fully connected layers

Fully connected layers are same as the perceptron layers. They come after the convolutional layers and the pooling layers. To feed a tensor into a fully connected layer we need to flatten the tensor into vector. The softmax or sigmoid layer that can end the CNN is also a fully connected layer.

Here is an example of CNN to classify hand written digits, which takes a 28×28 pixel, greyscale, input image (so $28 \times 28 \times 3$, considering the color channels red, blue, green: RGB) with no padding. It is fed into a convolutional layer of 10 filters of $5 \times 5 \times 3$, with a stride of 1. The output is a tensor of size $24 \times 24 \times 3$, and the convolutional layer is followed by a max pooling layer with $f = 2$ and $s = 2$. That divides by 2 the height and the width and gives an output of $12 \times 12 \times 3$ size. The latter is fed into another convolutional layer of 20 filters of $5 \times 5 \times 3$ with a stride of 1. The $8 \times 8 \times 20$ output is again fed into a max pooling layer with $f = 2$ and $s = 2$. The output is a $4 \times 4 \times 20$ tensor to which it's applied a dropout and the result is flattened and fed into two consecutive fully connected layers of 100 and 10 units respectively. Then, the softmax layer (with 10 units) computes the probability that the input image is 0, 1, 2, ..., 9. Other classical examples of CNN are AlexNet and VGG.

Figure 11: An example of simple feed-forward CNN like LeNet 5 (LeCun et al., 1998)



Source: <https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/>

1.4.3 Advantages of CNN

Use a CNN allows to reduce the number of parameters. This is since a same filter browse the whole of the input tensor, so there is parameter sharing. Besides the filter is smaller than the input, so that involve a sparsity of connection. Since the pooling makes the network less sensitive to small translations of the input images, then the CNN is good at capturing translation invariance.

To build very deep convolutional neural network, we face problem of exploding or vanishing gradient, and the number of parameters rises largely. Therefore, it is no longer enough to just stack blocks of convolutional and pooling layer. We need more sophisticated architectures like residual and inception network.

1.4.4 Residual Network (ResNet)

This network is based on the idea that we skip some layers with skip connections, or shortcuts. This allowed to avoid the problem of exploding or vanishing gradient. Using the notation of section 1.2, if there is a connection between layers l and $l + 2$, which skips the layer $l + 1$, then the activation of layer $l + 2$, is computed as followed in a residual network:

$$A^{[l+2]} = f^{[l+2]}(Z^{[l+2]} + A^{[l]}) = f^{[l+2]}[(W^{[l+2]}A^{[l+1]} + b^{[l+2]}) + A^{[l]}]$$
 (if $Z^{[l+2]}$ and $A^{[l]}$ have same dimension).

But when there was not skip connection i.e. in a plain network, we had: $A^{[l+2]} = f^{[l+2]}(Z^{[l+2]})$. Adding the previous activation $A^{[l]}$ in the computation of $A^{[l+2]}$ makes easier to the deep layers to learn identity function and that does not hurt the performance. Note that if $Z^{[l+2]}$ and $A^{[l]}$ did not have the same dimension, then $A^{[l]}$ is pre-multiplied by a weight matrix for the skipped connection that should be learned during the training.

1.4.5 Inception Network

The inception network is a kind of network in the network. It allows to apply filters of different size on the input. So rather than use, just a 5×5 filter in a convolutional layer, in the inception network, we can use at the same time a 1×1 , 3×3 , 5×5 and a max pooling in a same layer. The results of the convolutional product between the input tensor and the different filters are then

concatenated. However, doing so, is computationally expensive. To reduce the computation cost, the 3×3 , 5×5 kernels of convolution and the max pooling are respectively applied to the result of the convolution product of the input by the 1×1 kernel (this type of kernel allows to reduce the number of channels of the input). The different results are then concatenated to give the output of the inception network layer. Stacking many of this type of layer together forms the inception network.

As mentioned in the beginning of this section, CNN presents many applications such as face recognition, object detection and neural style transfer. It will not be possible to develop all these fields in this report, however we will give an overview of face recognition by introducing the Siamese Network.

1.4.6 Face verification and face recognition

A face verification system consists in verifying whether an input image with a name (or an ID) corresponds to the alleged person. This is a one to one problem. Face recognition is a bit more complicated and we face a problem of one-shot learning. This mean that a face recognition system should recognize a person given a single image of this person. To do this task, a similarity function is learned to evaluate the degree of similarity between two images, and the Siamese Network is the most adapted to do this task.

1.4.7 Siamese Network

It consists in turning the same CNN on two images x^i and x^j in order to compare them. The CNN therefore acts as an encoder that transforms the two input images into vectors $f(x^i)$ and $f(x^j)$, and the rest of the network will determine whether the distance $\|f(x^i) - f(x^j)\|$ is small enough to decide whether the images x^i and x^j represent the same person. There are two ways to train the parameters of a Siamese Network. The first method is to use a triplet loss function, and the second is to use a binary classification

- The triplet loss function:

As its name suggests, three images are used for a same i person: an anchor image $A^{(i)}$ as the reference, a positive image $P^{(i)}$ which represents the same person as $A^{(i)}$ and a negative image $N^{(i)}$ which does not represent the same person as $A^{(i)}$ and $P^{(i)}$. The role of the triplet loss function, is to find the parameters of the CNN such that the encodings of the three images verify:

$\|f(A^i) - f(P^i)\|^2 + \alpha \leq \|f(A^i) - f(N^i)\|^2$, where the role of α is to make the distance between $f(A^i)$ and $f(P^i)$ be smaller enough than $f(A^i)$ and $f(N^i)$. The triplet loss function is then given by:

$$L(A^i, P^i, N^i) = \max(\|f(A^i) - f(P^i)\|^2 + \alpha - \|f(A^i) - f(N^i)\|^2; 0).$$

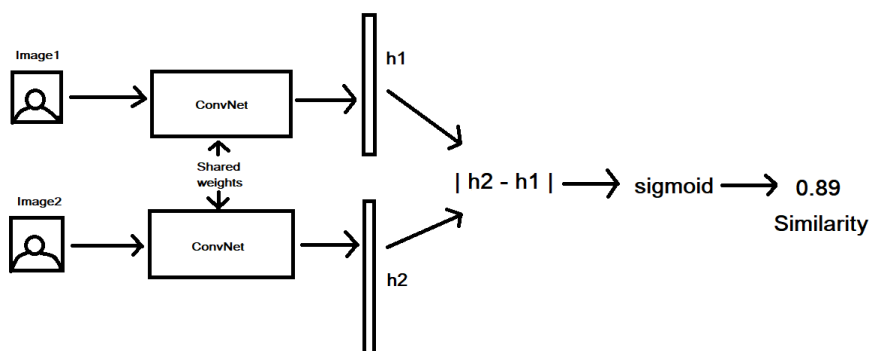
The cost function is just the sum over m different persons: $J = \sum_{i=1}^m L(A^i, P^i, N^i)$

This function is minimized thanks to optimization algorithms, and the parameters of the CNN are then computed.

- Face verification and binary classification

Another way to solve the face verification problem is to use a binary classification that will evaluate the probability that two input images are similar. Here is a figure of this kind of Siamese network:

Figure 12: Siamese network with binary classification.



Source: <https://heartbeat.fritz.ai/one-shot-learning-part-2-2-facial-recognition-using-a-siamese-network-5aee53196255>

In the figure above, the input is a pair of images, each of them is respectively fed into a convolutional neural network and the convolutional neural network has the same parameters or weights. The CNN allows an encoding of each image represented by flatten vectors h_1 and h_2 that are merged via the computation of the distance. The latter is fed into a sigmoid layer ($\sigma(W \cdot |h_2 - h_1| + b)$) to compute the probability that the two images represent the same person.

1.4.8 About the practice in the CNN course

As previously, we also build a convolutional model step by step with Numpy and after, we build a convolutional model to classify 6 images of signs representing numbers from 0 to 5. Using Keras we build a block of residual networks. Then we used YOLO (You Only Look Once) algorithm to build a convolutional network for car detection. We have also experimented art generation with neural style transfer and build a small face recognition system.

Chapter 2: Applications

In this chapter it will be question of implementing what we learned during the mooc. We have mainly worked in the field of NLP (Natural Language Processing) specifically text data. Two applications are provided: one concerning text similarity and the other is about text classification.

2.1 Application 1: Sentence textual similarity

This application is an extension of the Siamese network applied to texts. As a reminder, the Siamese network was presented in the last section of the previous chapter.

2.1.1 Data description

The goal is to determine, given two sentences, if they are contradictory (recoded into $y = 0$), if they do not evoke at all the same thing (recoded into $y=1$) or if they have the same meaning (recoded into $y=2$). The dataset we used is named SICK dataset and is available on kaggle at the following address : <https://www.kaggle.com/ozgeozkaya/sick-dataset>. It contains 4500 pairs of sentences and 5 columns: the first column is the pair ID, the two columns following contain the pairs of sentences, the fourth gives a relatedness score, and the last column is the target variable or the label, it indicates at each line if the pair of sentences is in contradiction, neutral or entailment.

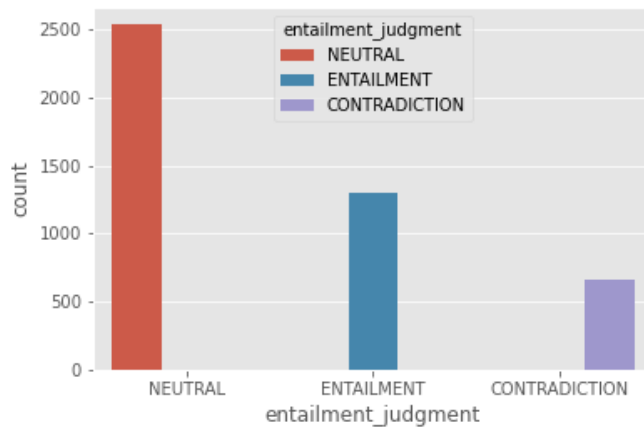
Table 2: Brief overview of the SICK dataset

pair_ID	sentence_A	sentence_B	relatedness_score	entailment_judgment
14	A brown dog is attacking another animal in front of the man in pants	Two dogs are fighting	3.5	NEUTRAL
266	The brown horse is near a red barrel at the rodeo	The brown horse is far from a red barrel at the rodeo	3.6	CONTRADICTION
5102	Some food is being prepared by a chef	A chef is preparing some food	4.815	ENTAILMENT

Source: Author of this report

In the dataset, there are 665 pairs in contradiction, 1299 pairs in entailment and 2536 pairs in neutral. The figure bellow present a bar plot of entailment_jugment variable which is the target variable.

Figure 13: Bar plot of entailment_jugment variable



Source: Author of this report

It can be noted that on the link of the dataset there is not a clear description of this database. But this dataset is pretty similar to the Quora Question Pairs dataset (available on the link: <https://www.kaggle.com/c/quora-question-pairs>), where the aim is to classify whether question pairs are duplicates or not. Indeed, since many people ask similarly worded questions on Quora, that makes writers feel they need to answer multiple versions of the same question, and at the opposite seekers to spend more time finding the best answer to their question (from : Quora Question Pairs dataset description). We choose the SICK dataset which is less used, but with the same task. Besides, the sentences included in the SICK dataset are quite cleaned and short, so we judge that they do not need much text preprocessing such as stopwords removing, stemming and lemmatization. Only punctuation removing and some clearing (like replacing the contracted form of verbs) were done on the data.

2.1.2 Methodology

The dataset was divided into train set and validation set with 30% of data allocated to the validation set. To classify the pairs of sentences according to their similarity, two networks were built: a Siamese network that uses bidirectional LSTM as encoder and another Siamese network that uses CNN as encoder. Then the cosine similarity between the encoding vector of each sentence in a pair is computed, and this cosine similarity is concatenated with the vector representing the difference between the encoding vectors, and the square of this vector. The

concatenated vector is then fed into a softmax layer. Note that a pretrained Glove dictionary for word embedding is used during the sentences encoding. After hyperparameters tuning, we end up the following structures:

- Structure of the Siamese network with bidirectional LSTM
 - ✓ An embedding layer based on the pretrained Glove: it takes each word in a sentence and associated to its embedding vector which is in the Glove dictionary.
 - ✓ A bidirectional LSTM layer made up of 64 units with a tanh activation function
 - ✓ A Lambda layer that compute the scalar product of the pair of vectors representing the sentences
 - ✓ A concatenation of the scalar product, the vector representing the difference between the encoding vectors of the sentences, and the square of this vector. These replace the $|h_2 - h_1|$ vector that was mentioned in figure 12.
 - ✓ A dense layer (a dense layer just computes: $f^{[l]}(W^{[l]}A^{[l-1]} + b^{[l]})$) of 100 units with a tanh function followed by a dropout regularization with a probability of 0.1
 - ✓ A softmax layer 3 units to compute the probability of the sentences in pair to be in contradiction, in entailment or in neutral.
- Structure of the Siamese network with CNN (slightly inspired by the work of Yoon Kim (2014))
 - ✓ The embedding layer based on Glove
 - ✓ Three 1D convolutional layers respectively with filter of length: 3,4 and 5; followed respectively by a 1D max pooling layer. The outputs of these layers are then concatenated to form the sentence encoding. Note that a dropout with probability of 0.1 was applied to the concatenated vector, and then flattened.
 - ✓ A dense layer of 64 units with 64 units
 - ✓ A Lambda layer that compute the scalar product of the pair of vectors representing the sentences
 - ✓ A concatenation of the scalar product, the vector representing the difference between the encoding vectors of the sentences, and the square of this vector

- ✓ A dense layer of 100 units with a tanh function followed by a dropout regularization with a probability of 0.1
- ✓ A softmax layer of 3 units to compute the probability of the sentences in pair to be in contradiction, in entailment or in neutral.

For the two structures, training is done through mini batch gradient descent with a mini batch size of 32 and 150 epochs. The Adam optimizer was used to minimize the cost function (categorical cross entropy) with a learning rate of 0.00001. The accuracy is used as the principal metric. We will compare the two approaches to see which one performs better. Note that, in application 1 as in application 2, the hyperparameters were arbitrarily chosen as hyperparameter tuning implies more computational cost requirement. But different values were tested for the hyperparameters and the ones which are in this structure are the ones that we have retained in definitive.

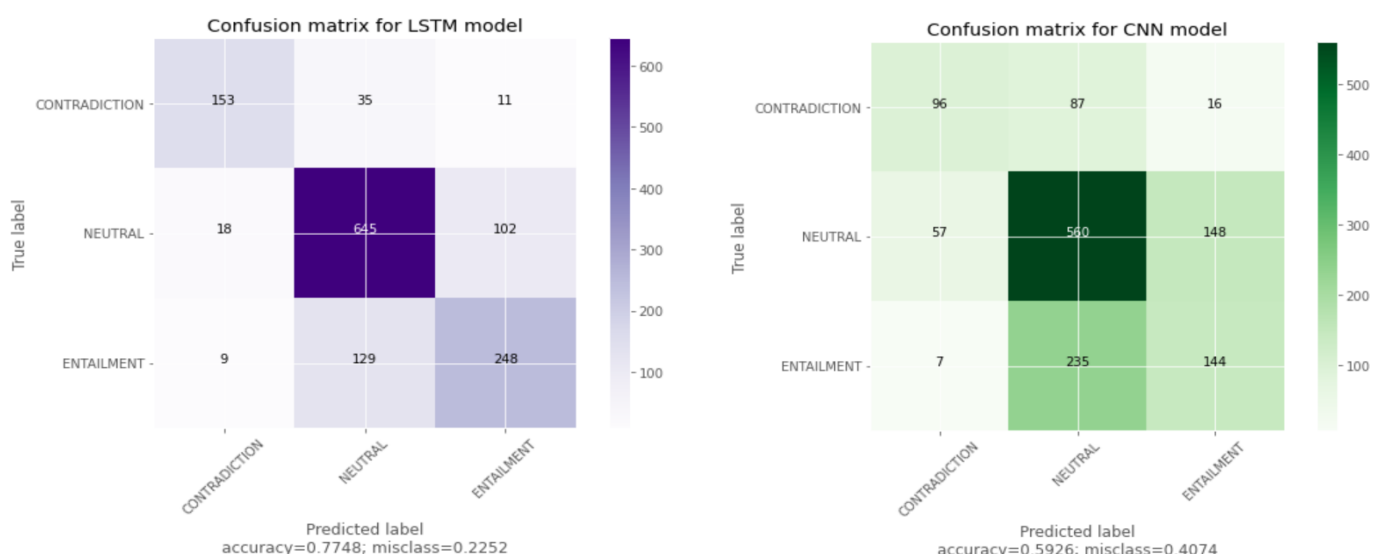
2.1.3 Results and Limits

Table 3: Statistical report for textual similarity classification using Siamese network with bidirectional LSTM and CNN

	Accuracy		AUC		F1-score (validation)
	Training set	Validation set	Training set	Validation set	
Bi-LSTM	0.9629	0.7748	0.9960	0.9125	0.764
CNN	0.9219	0.5926	0.9892	0.7837	0.543

Source: Author of this report

Figure 14: Confusion matrices (Source: Author of this report)



The table 3 shows that Siamese network with bidirectional LSTM (Bi-LSTM) seems to perform better than the one with CNN. The accuracy set of the model with Bi-LSTM is about 77,5% on

validation which is not very great but not too much bad also, whereas the model with CNN is worse with just 59%, and so the CNN is overfitting since the accuracy on training set is about 92%. May be change the structure of the CNN and the hyperparameters could result in a better performance. However, the AUC is quite good for the Bi-LSTM. Note that in the validation set, there are 199 pairs in contradiction, 386 pairs in entailment and 765 pairs in neutral. So, the validation test is somewhat unbalanced. The two models tend to misclassify several pairs that are neutral into entailment class and vice versa (see the confusion matrix in Figure 14). This can be because the frontier between the entailment and neutral is not clear, and in some extents, we can say that many pairs that are in neutral can be in entailment. In table 4 we select two examples of pairs that are in neutral, but which can be considered as in entailment by considering their meaning, and we note also that the relatedness score of a neutral pair can be even higher than the relatedness score of a entailment pair. (The python code use for this application is available at Appendix 2 and the cost evolutions during training are available at Appendix 1, figure a and b.)

Table 4: Illustration of some entailment and neutral pairs.

pair_ID	sentence_A	sentence_B	relatedness_score	entailment_judgment
26	A person is riding the bicycle on one wheel	A man in a black jacket is doing tricks on a motorbike	3.7	NEUTRAL
1	A group of kids is playing in a yard and an old man is standing in the background	A group of boys in a yard is playing and a man is standing in the background	4.5	NEUTRAL
240	A woman is wearing an Egyptian hat on her head	A woman is wearing an Egyptian headdress	4.3	ENTAILMENT

Source: Author of this report

2.2 Application 2: Daily News for Stock Market Prediction

This application is about text classification. The main aim is to use words from daily news headlines to predict the movement of stock market indices specifically the Dow Jones Industrial Average index (DJIA)

2.2.1 Data description

The data that we use are available on the link: <https://www.kaggle.com/aaron7sun/stocknews>.

Three files were provided but we mainly use the CombinedNewsDJIA dataset made up of 27 columns: the first column is Date in format year-month-day, the second is Label, and the following ones are news headlines ranging from Top1 to Top25. Top1 to Top25 columns are the top 25 headlines from Reddit WorldNews Channel ranked by reddit users' votes to predict. The Label column provides two labels: "1" when DJIA Adj Close value rose or stayed as the same (the close value is the same or bigger than the open value), "0" when DJIA Adj Close value decreased (the close value is smaller than the open value). The dates go from 2008-06-08 to 2016-07-01 (1989 trading days). So, the CombinedNewsDJIA dataset is a merging of News and stock data into a single dataset by aligning the news headlines with the trading days of the stock data : for each of the 1989 trading days all the DJIA index values for that day together with the most voted 25 news headlines for the same day. But since experiments showed that predicting stock value from yesterday's news gives higher predictive accuracy (Kavšek, 2017), we shift the column Label so that the label of the date t is aligned with top 25 headlines of the previous day.

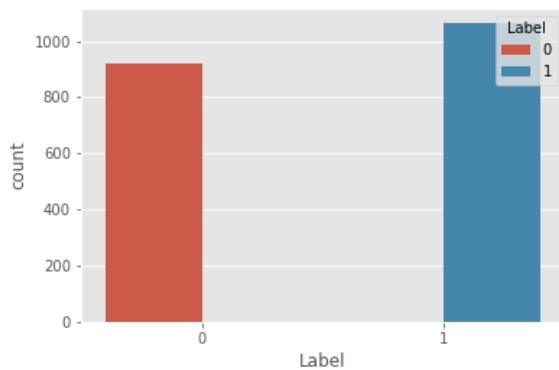
Here a view of the dataset:

Table 5: Head of the CombinedNewsDJIA dataset after shifting Label column

	Date	Label	Top1	Top2	Top3	Top4	Top5	Top6	Top7	Top8	...	Top17	Top18	Top19
0	2008-08-11	1	georgia downs two russian warplanes countries ...	breaking musharraf impeached	russia today columns troops roll south ossetia...	russian tanks moving towards capital south oss...	afghan children raped impunity american offici...	russian tanks entered south ossetia whilst geo...	breaking georgia invades south ossetia russia ...	enemy combatant trials nothing sham salim hama...	...	al qaeda faces islamist backlash	condoleezza rice america would act prevent isr...	busy day european americanion approved new san...
1	2008-08-12	0	wont america nato help us wont help us help iraq	bush puts foot georgian conflict	jewish georgian minister thanks israeli traini...	georgian army flees disarray russians advance ...	olympic opening ceremony fireworks faked	mossad fraudulent new zealand passports iraq	russia angered israeli military sale georgia	american citizen living sossetia blames americ...	...	believe tv neither russian georgian much victims	riots still going montreal police murdered boy...	china overtake america largest manufacturer
2	2008-08-13	0	remember adorable year old sang opening ceremo...	russia ends georgia operation	sexual harassment would children	al qaeda losing support iraq brutal crackdown ...	ceasefire georgia putin outmaneuvers west	microsoft intel tried kill xo laptop	stratfor russo georgian war balance power	trying get sense whole georgia russia war vote...	...	russias response georgia right	gorbachev accuses americas making serious blun...	russia georgia nato cold war two

Source: The author of this report

Figure 15: Bar plot of Label variable



Source: The author of this report

From 2008-08-11 to 2016-07-01 there are 923 days for which the close value of the DJIA is smaller than the open value and 1065 days for which the close value is the same or bigger than the open value.

2.2.2 Methodology

Our goal is to build a prediction model that will use the textual information from today's Reddit top 25 news headlines to predict tomorrow's rise or fall of the DJIA index. As recommended, for task evaluation we use data from 2008-08-11 to 2014-12-31 as training set, and test set is then the following two years data (from 2015-01-02 to 2016-07-01), this is roughly a 80% / 20% split. As argued in the literature, since the goal is to simulate the process of predicting new events from old, the sequence splitting is most appropriate. The AUC (Area under the ROC Curve) is used as the main evaluation metric. Note that the 25 news headlines were pool together for each date. The dataset has been preprocessed by removing undesirable characters, lowercasing letters, eliminating the stopwords, numbers, punctuation. To reach our goal, we choose deep learning and machine learning approaches, and we compare their result. Note that for the deep learning we do not add stemming to the preprocessing phase, because we use a pretrained Glove dictionary for word embedding and we felt that by stemming, the numbers of word in the dataset and not present in the Glove dictionary could rise and then hurt the performance of the deep learning approach. At the opposite, stemming was added to the preprocessing phase in the case of machine learning approaching.

In the deep learning approach, we build a network with two LSTM layers preceded by an embedding layer. The structure used is the following:

- ✓ An embedding layer based on the pretrained Glove: it takes each word in a sentence and associated to its embedding vector which is in the Glove dictionary.
- ✓ A LSTM layer made up of 120 units with a tanh activation function, followed by a batch normalization
- ✓ A unidirectional LSTM layer made up of 64 units with a ReLU activation function, followed by a batch normalization
- ✓ A sigmoid layer to compute the probability that the DJIA index rises/is stable or decreases

Training is done through mini batch gradient descent with a mini batch size of 32 and 75 epochs. The Adam optimizer was used to minimize the cost function (categorical cross entropy) with a learning rate of 0.01. Note that different values were tested for the hyperparameters, the ones which are in this structure are the ones that we have retained in definitive.

As regards machine learning approach, we use TF-IDF (Term Frequency-Inverted Document Frequency) with 3-grams for features extraction and we apply a logistic regression, a Passive Aggressive Classifier, a Random Forest Classifier and a linear Support Vector Machine. The python code use for this application is available at Appendix 3.

2.2.3 Results and Limits

About the models results, we get very poor level of performance. The following table summarizes the statistics of the models:

Table 6: Statistical report for stock market prediction using different models based on daily news. (Source: Author of this report)

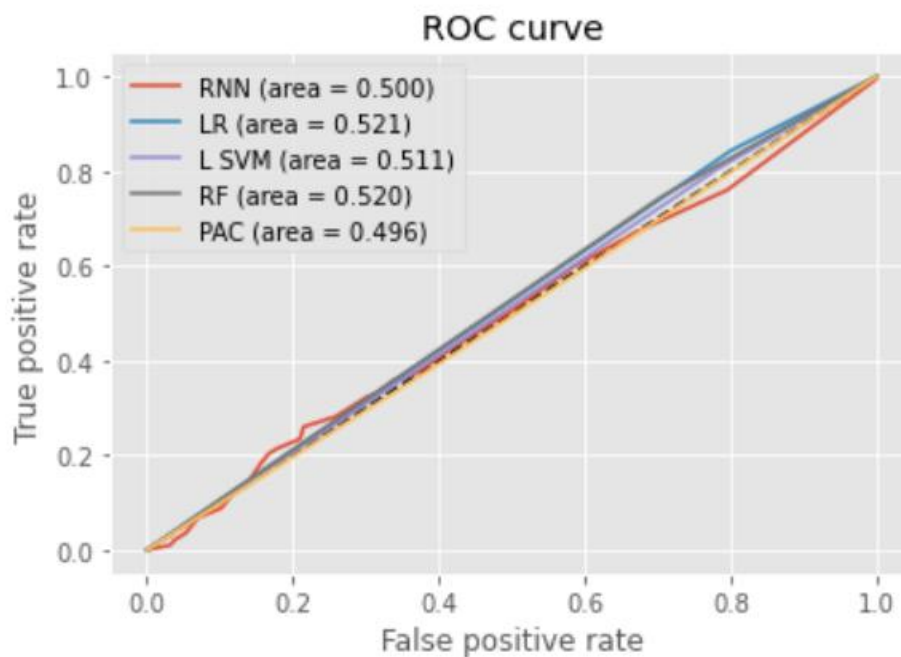
	Accuracy		AUC (validation set)
	Training set	Validation set	
RNN	0.542	0.507	0.500
LR	0.567	0.526	0.521
Linear SV	0.565	0.516	0.511
RF	0.670	0.526	0.523
PAC	0.492	0.500	0.504

LR: Logistic regression; Linear SV: Linear support vector machine; RF: Random Forests; PAC: Passive Aggressive Classifier.

fitting. However, Kavšek (2017) got 79% of accuracy by using Random Forests on the same dataset, yet we have used relatively the same procedures as he did for machine learning approaches. In his case, he got a variance problem and argued the fact that classifiers show amount of variance is perfectly normal and typical in data mining. We need to spend more time to try to improve the performance. We also think that, since obviously there is not only news which could affect evolution of the DJIA index, we may need to consider other variables in order to build a model more effective.

In the figure bellow we show the ROC curve of the different models since it was the main evaluation metric. (see the RNN cost function evolution during training at Appendix 1, figure c)

Figure 17: ROC Curves



Source: Author of this report

Conclusion

The deep learning is arousing more and more interest and is so much used in several fields. The craze to understand the underbelly of this part of the learning machine led us to follow the mooc entitled "Deep learning Specialization" presented by Professor Andrew Ng. At the end of this report on this mooc, we can say that it has really allowed us to understand the basic functioning of the different types of artificial neural network architectures used in deep learning.

From perceptrons to convolutional networks to recurrent networks, we have understood how to use them to solve supervised learning problems. We have also acquired knowledge for example about forward and backward propagation, optimization algorithms, hyperparameter tuning and regularization, convolution network, face recognition, YOLO algorithm, Neural style transfer, Long short term memory, sentiment analysis, trigger word detection, use of Keras and TensorFlow.

We have applied the acquired knowledge to two databases in the field of NLP. The first was to identify whether two sentences given were unrelated, contradictory or have common meaning. The results obtained in this context with a recurrent network are more or less encouraging. The second consisted in predicting the evolution of the DJIA index using news headlines. But the results obtained whether they are from the recurrent network and traditional machine learning algorithms are not so good. It may be necessary to try with other models than those we have presented. In the two applications, we may need more data since we use deep learning models that usually need huge dataset.

Later, we hope to be able to further develop what we have learned in this mooc.

Bibliography

Wikistat. *Neural Networks and Introduction to Deep Learning*

Kavšek Branko. 2017. Using Words from Daily News Headlines to Predict the Movement of Stock Market Indices. <https://doi.org/10.26493/1854-6935.15.109-121>

Lakshay Sharma, Laura Graesser, Nikita Nangia, Utku Evci. 2019. Natural Language Understanding with the Quora Question Pairs Dataset.

Abien Fred M. Agarap, Paul M. Grafilon. 2018. Statistical Analysis on E-Commerce Reviews, with Sentiment Classification using Bidirectional Recurrent Neural Network.

Charles-Emmanuel Dias, Clara Gainon de Forsan de Gabriac, Vincent Guigue, Patrick Gallinari. RNN & modèle d'attention pour l'apprentissage de profils textuels personnalisés.

Huimin Chen, Maosong Sun, Cunchao Tu, Yankai Lin, Zhiyuan Liu. 2016. Neural Sentiment Classification with User and Product Attention. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1650–1659, Austin, Texas, November 1-5, 2016*

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification

Michael Nielson. 2019. Neural Networks and Deep Learning. <http://neuralnetworksanddeeplearning.com>

APPENDIX

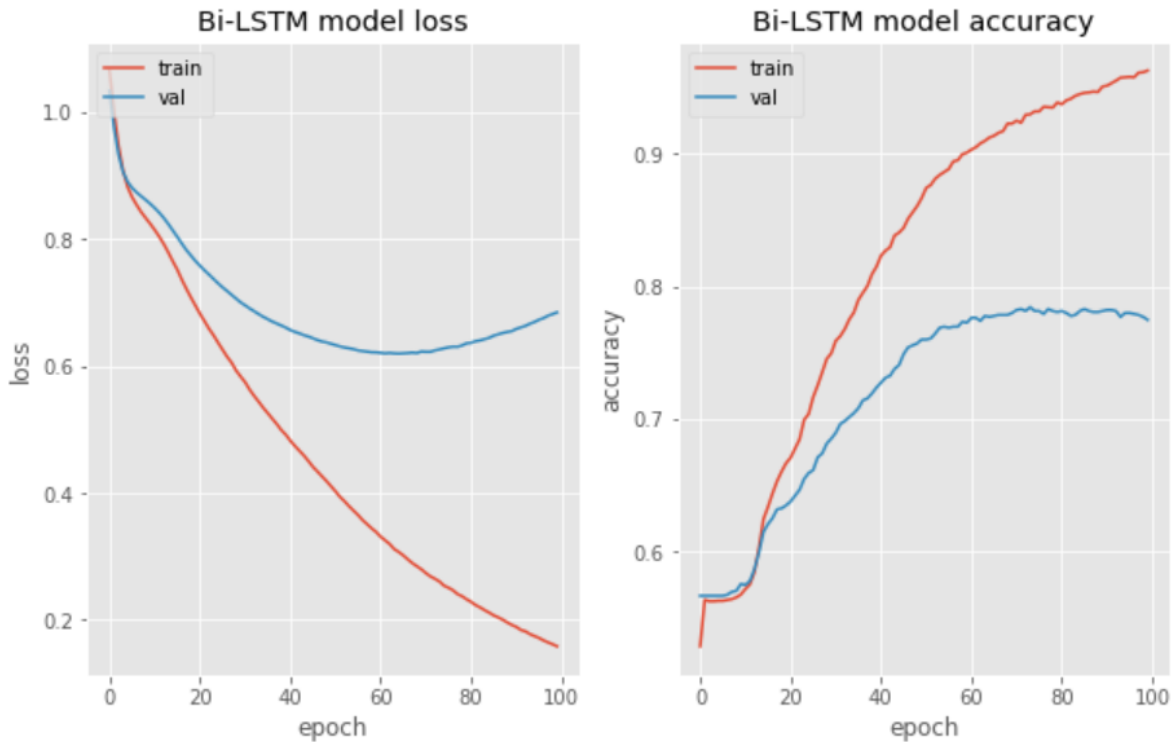
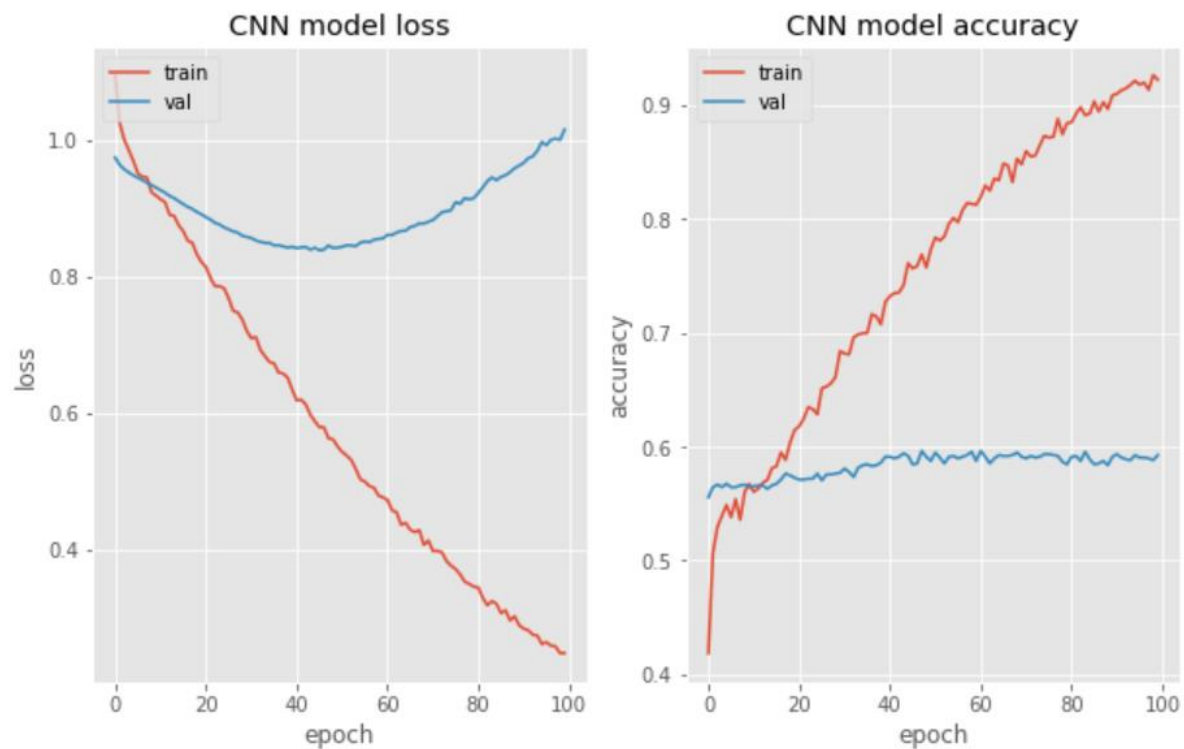
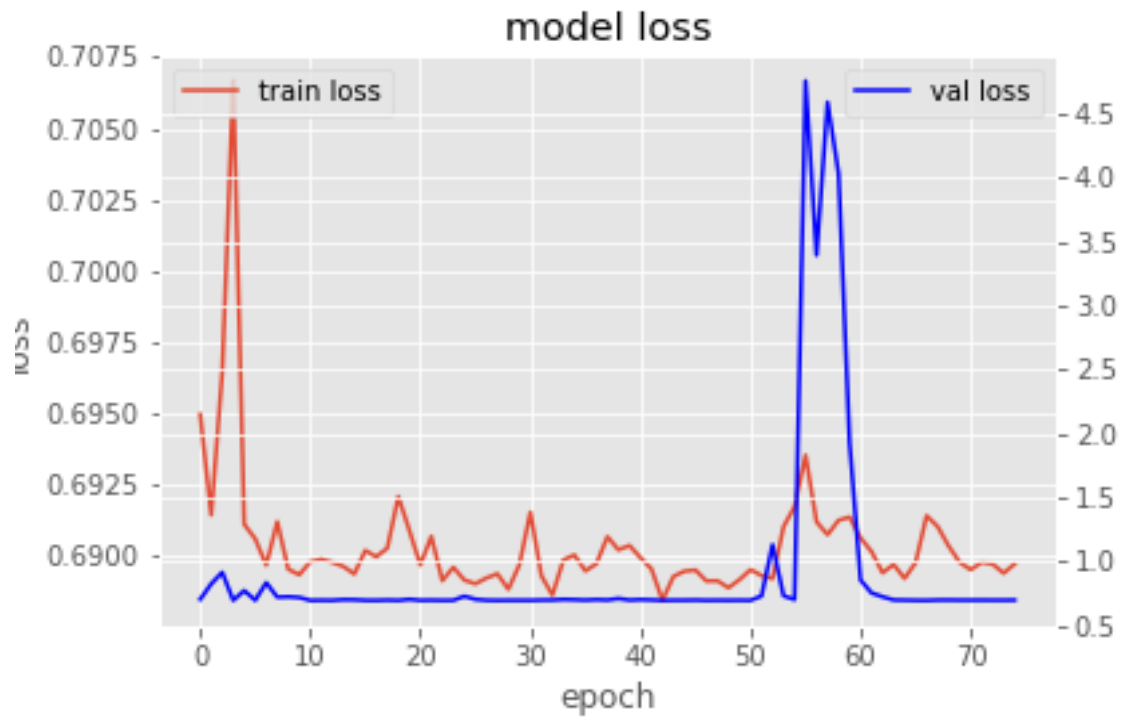
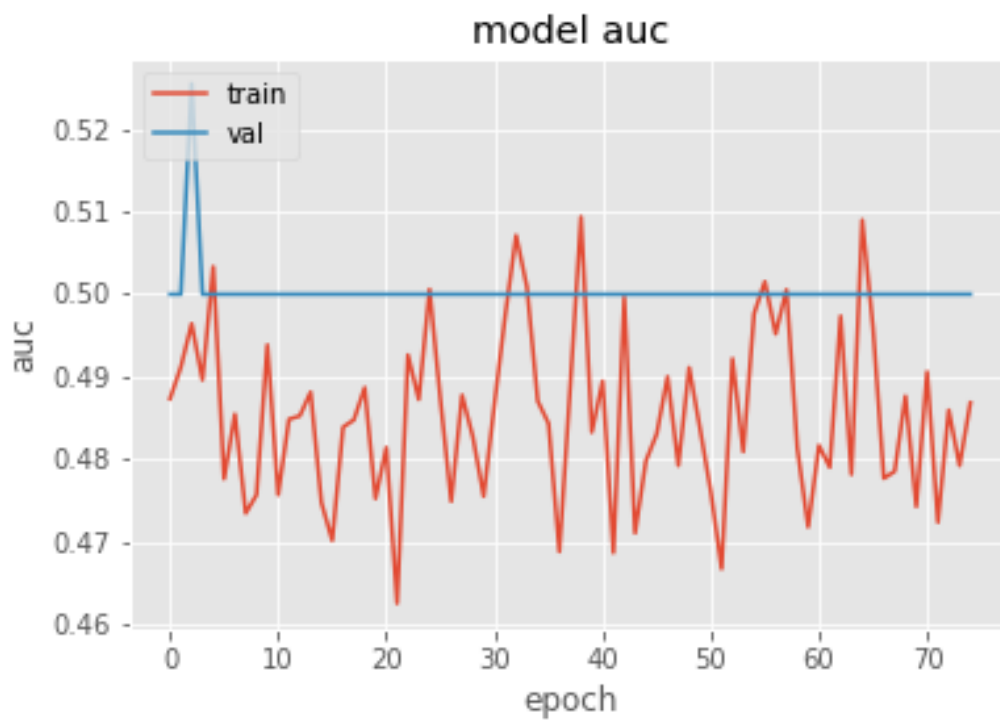
Appendix 1**Figure a:** Evolution of the loss and the accuracy of the Bi-LSTM model in application 1**Figure b:** Evolution of the loss and the accuracy of the CNN models of application 1

Figure c: Evolution of the loss and the accuracy of the LSTM model in application 2



The cost function does not decrease monotonously.



Appendix 2: Script of application 1 (I certify by honor that I have wrote these codes by myself)

The jupyter notebook is available on:

<https://drive.google.com/file/d/1ukBHMVdnPBTelRrmwEtZDbDnK7UGVBG0/view?usp=sharing>

```
##import all required packages
import pandas as pd
import numpy as np
import re
import string
import nltk
nltk.download("punkt")
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
stemmer = SnowballStemmer('english')
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

nltk.download('stopwords')

##need to install VC_redist.x64 before installing tensorflow
pip install tensorflow
pip install keras

import tensorflow as tf
import keras
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model
from keras.layers import Bidirectional, Dense, Input, LSTM, Embedding,
Dropout, Multiply, Concatenate, Activation, Lambda, Subtract, Flatten
from keras.layers.normalization import BatchNormalization
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.initializers import glorot_uniform
from keras.optimizers import Adam
from keras.utils import to_categorical
import keras.metrics
from keras.layers.embeddings import Embedding
from keras import backend as K
from keras.layers.core import Lambda
from sklearn.metrics import confusion_matrix, f1_score
from keras.layers import Conv1D
from keras.layers import MaxPooling1D
from keras.layers import Flatten
from keras.layers.merge import concatenate

##import the data and visualize
data = pd.read_csv("D:/datasets_633450_1126605_SICK_train.txt", header
= 0 ,sep="\t")
display(data)

##barplot of the 'entailment_judgment' variable
plt.style.use('ggplot')
sns.countplot(x='entailment_judgment', hue='entailment_judgment',
data=data)
```



```

plt.savefig("stat_desc")

##categories size in the dataset
print(data["entailment_judgment"].tolist().count('CONTRADICTION'))
print(data["entailment_judgment"].tolist().count('NEUTRAL'))
print(data["entailment_judgment"].tolist().count('ENTAILMENT'))

##recode the the 'entailment_judgment' variable

conditions = [
    (data['entailment_judgment'] == 'CONTRADICTION'),
    (data['entailment_judgment'] == 'NEUTRAL'),
    (data['entailment_judgment'] == 'ENTAILMENT')]
choices = [0,1,2]
data['Y'] = np.select(conditions, choices)

## define a data preprocessing function

def clean_sentence(x,stem_words=True):
    tokens = word_tokenize(x) ## tokenize
    tokens = [w.lower() for w in tokens] ## lowercase
    table = str.maketrans(' ', '', string.punctuation) ## remove
punctuation
    words = [w.translate(table) for w in tokens]
    if stem_words: ##stem if you
want
        words = [stemmer.stem(word) for word in words]
    text = " ".join(words)
    return text

###define a sub-function that removes undesirable characters, and
replace contracted forms
def general_clean_text(column_text,stem_words=True):
    column_text = column_text.apply(lambda x: re.sub(r"^[A-Za-z0-9^,!.\/'+-=]", " ", x))
    column_text = column_text.apply(lambda x: re.sub(r"what's", "what
is ", x))
    column_text = column_text.apply(lambda x: re.sub(r"\'s", " ", x))
    column_text = column_text.apply(lambda x: re.sub(r'\ve', " have
", x))
    column_text = column_text.apply(lambda x: re.sub(r"can't", "cannot
", x))
    column_text = column_text.apply(lambda x: re.sub(r"n't", " not ",
x))
    column_text = column_text.apply(lambda x: re.sub(r"i'm", "i am ",
x))
    column_text = column_text.apply(lambda x: re.sub(r"\'re", " are ",
x))
    column_text = column_text.apply(lambda x: re.sub(r"\d", " would
", x))
    column_text = column_text.apply(lambda x: re.sub(r"\ll", " will
", x))
    column_text = column_text.apply(lambda x:
clean_sentence(x,stem_words))
    return column_text

##apply the function

data['sentence_A']=general_clean_text(data['sentence_A'],stem_words=False)
data['sentence_B']=general_clean_text(data['sentence_B'],stem_words=False)
display(data)

```




```

##split into train set and test set (validation set more precisely)
X_train, X_test, Y_train, Y_test =
train_test_split(data.iloc[:,1:3],data["Y"],test_size=0.3,random_state
=random.seed(2))

print(len(X_train))
print(len(X_test))

##categories size in the test set

print(Y_test.tolist().count(0))
print(Y_test.tolist().count(1))
print(Y_test.tolist().count(2))

## transform the target variable into one-hot-encoding variable
Y_train_onehot=LabelBinarizer().fit_transform(Y_train)
Y_test_onehot=LabelBinarizer().fit_transform(Y_test)

##combine the pair of sentences in a column named 'text'
data['text'] = data[['sentence_A','sentence_B']].apply(lambda
x:str(x[0])+" "+str(x[1]), axis=1)

##tokenize the entire text in the coulumn text
tokenizer_obj = Tokenizer()
tokenizer_obj.fit_on_texts(data['text'].values)

##assign an inter to each word found in the text column
tokenizer_obj.word_index
len(tokenizer_obj.word_index) ##number of unique words

##transform the the columns of data frame into lists
X_train_sentA=X_train['sentence_A'].tolist()
X_train_sentB=X_train['sentence_B'].tolist()
X_test_sentA=X_test['sentence_A'].tolist()
X_test_sentB=X_test['sentence_B'].tolist()

##vectorize text corpus, by turning each text into a sequence of
integers
X_train_seqA = tokenizer_obj.texts_to_sequences(X_train_sentA)
X_train_seqB = tokenizer_obj.texts_to_sequences(X_train_sentB)
X_test_seqA = tokenizer_obj.texts_to_sequences(X_test_sentA)
X_test_seqB = tokenizer_obj.texts_to_sequences(X_test_sentB)

X_train_sentA[0:2]

##each sentence is now a sequence of integer, this interger will be
matched to the embeddings vector of the word
##in a dictionary
X_train_seqA[0:2]

#research among all the sentences the one which has the largest number
of words , this will serve to determine
## the dimension of the embedding layer

max_len = max(max([len(sent_vec) for sent_vec in X_train_seqA]),
               max([len(sent_vec) for sent_vec in X_train_seqB]),
               max([len(sent_vec) for sent_vec in X_test_seqA]),
               max([len(sent_vec) for sent_vec in X_test_seqB]))

max_len

```



```

##since all sentences don't have the same length, we pad them until
with 0 so tht all the sentences will be
##as long as the sentence which has the largest number of words
X_train_seqA = pad_sequences(X_train_seqA, maxlen=max_len,
padding='post')
X_train_seqB = pad_sequences(X_train_seqB, maxlen=max_len,
padding='post')
X_test_seqA = pad_sequences(X_test_seqA, maxlen=max_len,
padding='post')
X_test_seqB = pad_sequences(X_test_seqB, maxlen=max_len,
padding='post')

X_train_seqA[0:2]

X_train_seqA.shape

#import a pretrained Glove for word embedding via
https://www.kaggle.com/thanakomsn/glove6b300dtxt
#and create the dictionary that will contain each word and its vector
embeddings_index = {}
f =
open('D:/5504_8240_compressed_glove_6B_300d_txt/glove_6B_300d.txt',enc
oding = "utf-8")
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))

embeddings_index['patato']

len(embeddings_index['patato'])

not_present_list = [] ## create a empty list, it will keep the words
which are in our vocabulary of the data and not in the glove
dictionnayr

vocab_size = len(tokenizer_obj.word_index) + 1 # adding 1 to fit
Keras embedding (requirement)

print('Loaded %s word vectors.' % len(embeddings_index))

embedding_matrix = np.zeros((vocab_size, len(embeddings_index['no'])))
## initialize the embeddind matrix of shape (2169,300) to 0

for word, i in tokenizer_obj.word_index.items(): ## we browser
whole of couples (word , index) that exist in the vocabulary of the
data
    if word in embeddings_index.keys(): ## if the word which is in our
vocabulary is in glove dictionnary, take its embedding vector
        embedding_vector = embeddings_index.get(word)
    else:
        not_present_list.append(word) ##else, add the word to the
not_present_list, its embedding vector is None for the moment
        if embedding_vector is not None: ## set the embedding vector of
word in the embedding matrix
            embedding_matrix[i] = embedding_vector
        else:
            embedding_matrix[i] = np.zeros(300) ## the embedding vector

```



```

for unknown words, is 0

##create a function that computes the cosine cosine similarity
###https://stackoverflow.com/questions/51003027/computing-cosine-
similarity-between-two-tensors-in-keras/51003359#51003359

def cosine_distance(vests):
    x, y = vests
    x = K.l2_normalize(x, axis=-1)    ##compute normalize x to be a
vector of norm 1
    y = K.l2_normalize(y, axis=-1)    ##compute normalize y to be a
vector of norm 1
    return -K.mean(x * y, axis=-1, keepdims=True)

def cos_dist_output_shape(shapes):
    shape1, shape2 = shapes
    return (shape1[0],1)

##Build the model

#### we introduce the the pair of sentences in the model, each
sentence is an array of
#### length X_train_seqA.shape[1] =32= max_len

input_1 = Input(shape=(X_train_seqA.shape[1],))
input_2 = Input(shape=(X_train_seqB.shape[1],))

##create the embedding layer
common_embed = Embedding(name="embed_layer", input_dim = vocab_size,

output_dim=len(embeddings_index['patato']),weights=[embedding_matrix],

input_length=X_train_seqA.shape[1],trainable=False)

##get the embedded form of each word in the sentences
sent_1 = common_embed(input_1)
sent_2 = common_embed(input_2)

##create the bi-LSTM that will be run on the embedded sequences
common_bilstm = Bidirectional(LSTM(64,return_sequences=True,
activation="tanh"))

vector_1 = common_bilstm(sent_1)
vector_1 = Flatten()(vector_1) ## flatten the vector

vector_2 = common_bilstm(sent_2)
vector_2 = Flatten()(vector_2)

## compute the square of each element of the vector vector_1-vector_2
x3 = Subtract()([vector_1, vector_2])
x3 = Multiply()([x3, x3])

## compute the square of each element of the vector vector_1 and the
vector_2 and substract
x1_ = Multiply()([vector_1, vector_1])
x2_ = Multiply()([vector_2, vector_2])
x4 = Subtract()([x1_, x2_])

##compute the cosine
x5 = Lambda(cosine_distance,
output_shape=cos_dist_output_shape)([vector_1, vector_2])

```



```

# concatenate
conc = Concatenate(axis=-1) ([x5,x4,x3])

## apply tanh function, a dropout and then a softmax function
x = Dense(100, activation="tanh", name='conc_layer')(conc)
x = Dropout(0.1)(x)
out = Dense(3, activation="softmax", name = 'out')(x)

model_lstm = Model([input_1, input_2], out)

model_lstm.summary()

##compile the model by choosing the loss, the metrics and the
optimizer
model_lstm.compile(loss="categorical_crossentropy",
metrics=['accuracy',keras.metrics.AUC()], optimizer=Adam(0.00001))

##fit the model
hist_lstm=model_lstm.fit([X_train_seqA,X_train_seqB],
Y_train_onehot,validation_data=([X_test_seqA,X_test_seqB],Y_test_oneho
t),epochs=100,batch_size = 32)

##plot the loss function and the accuracy
plt.figure(figsize=(10,6))

plt.subplot(1, 2, 1)
plt.plot(hist_lstm.history['loss'])
plt.plot(hist_lstm.history['val_loss'])
plt.title('Bi-LSTM model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(hist_lstm.history['accuracy'])
plt.plot(hist_lstm.history['val_accuracy'])
plt.title('Bi-LSTM model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.savefig("loss_acc_lstm.PNG")
plt.show()

##prediction on the test set
Y_pred_lstm = np.argmax(model_lstm.predict([X_test_seqA,X_test_seqB]),
axis=1)
Y_test_array = np.asarray(Y_test)

##compute the confusion matrix and the F1-score
cm_lstm = confusion_matrix(Y_test_array,Y_pred_lstm,labels=[0,1,2])
print(cm_lstm)
print("LSTM F1-Score: %0.3f " % (f1_score(Y_test_array,Y_pred_lstm,
average="macro"))))

## Here we define a fonction to plot the confusion matrix

##https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-
matrix-with-labels

def plot_confusion_matrix(cm,
                           target_names,
                           title='Confusion matrix',

```

```

        cmap=None,
        normalize=True,color='Greens'):
    """
    given a sklearn confusion matrix (cm), make a nice plot

    Arguments
    -----
    cm:                confusion matrix from
sklearn.metrics.confusion_matrix

        target_names: given classification classes such as [0, 1, 2]
                        the class names, for example:
['CONTRADICTION','NEUTRAL','ENTAILMENT']

        title:         the text to display at the top of the matrix

        cmap:          the gradient of the values displayed from
matplotlib.pyplot.cm
                        see
http://matplotlib.org/examples/color/colormaps\_reference.html
                        plt.get_cmap('jet') or plt.cm.Blues

        normalize:     If False, plot the raw numbers
                        If True, plot the proportions

    Usage
    ----
    plot_confusion_matrix(cm                = cm,                #
confusion matrix created by                #

sklearn.metrics.confusion_matrix          #
                        normalize         = True,                # show
proportions                               #
                        target_names = y_labels_vals,            # list
of names of the classes                   #
                        title          = best_estimator_name) # title
of graph

    Citiation
    -----
    http://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_confusion\_matrix.h
tml

    """
    import itertools

    accuracy = np.trace(cm) / np.sum(cm).astype('float')
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap(color)

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:

```



```

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

thresh = cm.max() / 1.5 if normalize else cm.max() / 2
for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
    if normalize:
        plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                  horizontalalignment="center",
                  color="white" if cm[i, j] > thresh else "black")
    else:
        plt.text(j, i, "{:,}".format(cm[i, j]),
                  horizontalalignment="center",
                  color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label\naccuracy={:0.4f};
misclass={:0.4f}'.format(accuracy, misclass))
plt.show()

####plot the confusion matrix with the defined function
plot_confusion_matrix(cm_lstm,['CONTRADICTION','NEUTRAL','ENTAILMENT']
,
                        title= 'Confusion matrix for LSTM model',
                        cmap=None,
                        normalize=False,color='Purples')
plt.savefig("matrix_conf_lstm.PNG")

##Build the model CNN

####we introduce the the pair of sentences in the model, each sentence
is an array of length X_train_seqA.shape[1] =32

input_1 = Input(shape=(X_train_seqA.shape[1],))
input_2 = Input(shape=(X_train_seqB.shape[1],))

##create the embedding layer
common_embed = Embedding(name="embed_layer", input_dim = vocab_size,

output_dim=len(embeddings_index['patato']),weights=[embedding_matrix],

input_length=X_train_seqA.shape[1],trainable=False)

##get the embedded form of each word in the sentences
sent_1 = common_embed(input_1)
sent_2 = common_embed(input_2)

##build 1D convolutional layer with different filter sizes, each of
them is followed by max pooling layer
filter_sizes = [3,4,5]
convsl = []

for filter_size in filter_sizes:
    l_conv1 = Conv1D(filters=64, kernel_size=filter_size,
activation='relu')(sent_1)
    l_pool1 = MaxPooling1D(pool_size=3)(l_conv1)
    convsl.append(l_pool1)

##merge the different output
l_merge1 = concatenate([convsl[0],convsl[1],convsl[2]],axis=1)

```



```

##apply dropout, flatten and apply a tanh function
vector_1 = Dropout(0.1)(l_merge1)
vector_1 = Flatten()(vector_1)
vector_1 = Dense(64, activation='tanh')(vector_1)

convs2 = []
for filter_size in filter_sizes:
    l_conv2 = Conv1D(filters=64, kernel_size=filter_size,
activation='relu')(sent_2)
    l_pool2 = MaxPooling1D(pool_size=3)(l_conv2)
    convs2.append(l_pool2)

##same as before

l_merge2 = concatenate([convs2[0],convs2[1],convs2[2]],axis=1)

vector_2 = Dropout(0.1)(l_merge2)
vector_2 = Flatten()(vector_2)
vector_2 = Dense(64, activation='tanh')(vector_2)

x3 = Subtract()([vector_1, vector_2])
x3 = Multiply()([x3, x3])

x1_ = Multiply()([vector_1, vector_1])
x2_ = Multiply()([vector_2, vector_2])
x4 = Subtract()([x1_, x2_])

x5 = Lambda(cosine_distance,
output_shape=cos_dist_output_shape)([vector_1, vector_2])

conc = Concatenate(axis=-1)([x5,x4,x3])

## apply tanh function, a dropout and then a softmax function

x = Dense(100, activation="tanh", name='conc_layer')(conc)
x = Dropout(0.5)(x)
out = Dense(3, activation="softmax", name = 'out')(x)

model_cnn = Model([input_1, input_2], out)

model_cnn.summary

model_cnn.compile(loss="categorical_crossentropy",
metrics=['accuracy',keras.metrics.AUC()], optimizer=Adam(0.00001))

hist_cnn = model_cnn.fit([X_train_seqA,X_train_seqB],
Y_train_onehot,validation_data=([X_test_seqA,X_test_seqB],Y_test_oneho
t),epochs=100,batch_size = 32)

plt.figure(figsize=(10,6))
plt.subplot(1, 2, 1)
plt.plot(hist_cnn.history['loss'])
plt.plot(hist_cnn.history['val_loss'])
plt.title('CNN model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(hist_cnn.history['accuracy'])
plt.plot(hist_cnn.history['val_accuracy'])
plt.title('CNN model accuracy')

```

```
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.savefig("loss_acc_cnn.PNG")
plt.show()

##prediction with the CNN model
Y_pred_cnn = np.argmax(model_cnn.predict([X_test_seqA,X_test_seqB]),
axis=1)

##confusion matrix and F1-score
cm_cnn = confusion_matrix(Y_test_array,Y_pred_cnn,labels=[0,1,2])
print(cm_cnn)
print("CNN F1-Score: %0.3f " % (f1_score(Y_test_array,Y_pred_cnn,
average="macro"))))

##plot the confusion matrix with the defined function
plot_confusion_matrix(cm_cnn,['CONTRADICTION','NEUTRAL','ENTAILMENT']
,
title='Confusion matrix for CNN model',
cmap=None,
normalize=False,color='Greens')
plt.savefig("matrix_conf_cnn.PNG")
```



Appendix 3: Script of application 2 (I certify by honor that I have wrote these codes by myself)

The jupyter notebook is available on:

https://drive.google.com/file/d/1m2OsPTwYFIswfZaQmUsJaOSBM1Vt_OL7/view?usp=sharing

```
#First section
'''In this section we will have to present the data at our disposal,
we will also have to clean the database
(pre-processing), for this purpose we will import the necessary
packages.'''
import pandas as pd
import numpy as np
import re
import string
import nltk
nltk.download("punkt")
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
nltk.download('stopwords')
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

pip install wordcloud

from wordcloud import WordCloud

## importer la base
news_djia =
pd.read_csv("D:/129_792900_compressed_Combined_News_DJIA/Combined_News_
_DJIA.csv", header = 0 ,sep=",")
news_djia.head()
len(news_djia)

##shift the columns Date and Label to make that the today label is
aligned with the previous day news headlines
news_djia['Label'] = news_djia.Label.shift(-1)
news_djia['Date'] = news_djia['Date'].shift(-1)
news_djia.drop(news_djia.index[len(news_djia)-1], inplace=True)
news_djia.head()
len(news_djia)

## There is the character b\' which returns several times, we are
going to get rid of it in the whole database.
news_djia = news_djia.replace('b\'|b\'|\\\\\\\\\\\\\\\\', '', regex=True)

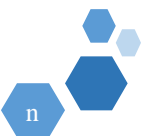
##convert the label column into integer, it have been changed to float
during the shift
news_djia["Label"] = news_djia["Label"].astype(int)

##define a function to remove undesirable characters

def clean_text(column_text):

    # voir des types de caractères comme [PICS] dans
news_djia.iloc[895,27], [VIDEO] dans news_djia.iloc[2,27]
    column_text = column_text.apply(lambda x: re.sub('[^\w\s]*',
'',x))

    # enlever les parenthèses :voir news_djia.iloc[2,27] pour
```



```

(video),des url comme CNN.com ou *.co.uk
    column_text = column_text.apply(lambda x:
re.sub('\w*.co.uk*|BBC*|NEWS*|BBC News*|Reuters BBC News*|Reuters*|ABC
News*|\w*\..com*|' + '^\\]*\\', '', x))

    #enlever les tirets en double ou triple voir news_djia.iloc[14,27]
pour remarquer
    column_text = column_text.apply(lambda x: re.sub('\-', ' ',x))

    #enlever les pipes | pour remarquer
    column_text = column_text.apply(lambda x: re.sub('\|', '',x))

    ##remplacer US ou U.S. ou U.S.A. par america, voir dans le
news_djia.iloc[2,27] pour remarquer
    column_text = column_text.apply(lambda x:
re.sub('US*|U\.S\.A\.*|USA', 'america',x))

    ##remplacer UK ou U.K. par britain, voir news_djia.iloc[89,27]
pour remarquer
    column_text = column_text.apply(lambda x:
re.sub('UK*|U\.K\.', 'britain',x))

    ##enlever les caractères \n par britain, voir
news_djia.iloc[966,27] pour remarquer
    column_text = column_text.apply(lambda x: re.sub(r"\n", '',x))

    ##enlever les URL
    column_text = column_text.apply(lambda x:
re.sub(r' (https|http)?:\//\//(\w|\.|\/|\?|\=|\&|\%)*\b', '',x))

    ##remplacer UN ou U.N. par united nations, voir
news_djia.iloc[89,27] pour remarquer
    column_text = column_text.apply(lambda x:
re.sub('UN*|U\.N\.', 'united nations',x))

    return column_text

##convert to string , before appying the defined function
for i in (news_djia.columns[2:27]):
    news_djia[i] = news_djia[i].astype(str)

for i in (news_djia.columns[2:27]):
    news_djia[i] = clean_text(news_djia[i])

##lowercase, form a word list of each title (tokenize), remove
punctuation marks and non-alphabetic characters,
##and finally remove stopwords

def final_data(x):

    tokens = word_tokenize(x) ## tokenizer
    tokens = [w.lower() for w in tokens] ## rendre miniscule
    table = str.maketrans('', '', string.punctuation) ## enlever
la ponctuation
    stripped = [w.translate(table) for w in tokens]
    words = [word for word in stripped if word.isalpha()]
    ##enlever les chiffres par exemples
    words = [word for word in words if word not in
stopwords.words('english')] ## enlever les stopwords anglais
    text = " ".join(words)
    return text

def reclean_column(column_text):

```



```

column_text = column_text.apply(lambda x: final_data(x))
return column_text

##apply the function
for i in (news_djia.columns[2:27]):
    news_djia[i] = reclean_column(news_djia[i])

#gather the 25 news headlines in a single column
news_djia['combined_news'] = None
for i in range(len(news_djia)):
    news_djia.iloc[i,27] = ' '.join(str(x) for x in
news_djia.iloc[i,2:27]) ## la colonne combined_news est la 28ème

#view
news_djia.iloc[2,27]

##select all news headlines associated respectively with an increase
in the DJIA index and a decrease
X_djia_up =
news_djia[news_djia["Label"]==1]['combined_news'].tolist()
X_djia_down =
news_djia[news_djia["Label"]==0]['combined_news'].tolist()

##form the train and test sets by selecting news headlines in the
period 2008-08-08 to 2014-12-31, and 2015-01-01 to 2016-07-01
###initialize
X_train = pd.DataFrame()
X_test = pd.DataFrame()

X_train['combined_news'] = news_djia[news_djia['Date'] < '2015-01-
01']['combined_news'] ## selectionner l'ensemble des titres de
08/08/2008 au 31/12/2014
X_test['combined_news'] = news_djia[news_djia['Date'] > '2014-12-
31']['combined_news'] ## selectionner l'ensemble des titres de
01/01/2015 au 01/07/2016

Y_train = np.reshape(np.array(news_djia[news_djia['Date'] < '2015-01-
01']['Label'],dtype="float64"), (len(X_train),1))
Y_test = np.reshape(np.array(news_djia[news_djia['Date'] > '2014-12-
31']['Label'],dtype="float64"), (len(X_test),1))

##do the following when instead of pooling, we use each column
'''
for i in (news_djia.columns[2:27]):
    X_train[i] = news_djia[news_djia['Date'] < '2015-01-01'][i]
    X_test[i] = news_djia[news_djia['Date'] > '2014-12-31'][i]
'''
print(len(X_train))
print(len(X_test))

## gather in into a single text all the texts contained in X_djia_up
djia_up_list = []
for sentence in X_djia_up:
    e = sentence.split()
    djia_up_list.append(e)
djia_up_list = [j for sub in djia_up_list for j in sub]

## gather in into a single text all the texts contained in X_djia_down
djia_down_list = []
for sentence in X_djia_down:
    f = sentence.split()
    djia_down_list.append(f)
djia_down_list = [j for sub in djia_down_list for j in sub]

```



```
len(djia_up_list)

##bar plot of Label variable
plt.style.use('ggplot')
sns.countplot(x='Label', hue='Label', data=news_djia)
plt.savefig("fall0_rise1.PNG")

##classes size
print(news_djia["Label"].tolist().count(0))
print(news_djia["Label"].tolist().count(1))

from collections import Counter

##count number of each word in djia_up_list and in djia_down_list
djia_up_counter = Counter(djia_up_list)
djia_down_counter = Counter(djia_down_list)

djia_up_top_30_words = pd.DataFrame(djia_up_counter.most_common(30),
columns=['word', 'count'])
djia_down_top_30_words =
pd.DataFrame(djia_down_counter.most_common(30), columns=['word',
'count'])

##word cloud when DJIA goes up/stable
wordcloud1 = WordCloud(background_color='black', width=3000,
height=2500).generate(' '.join(djia_up_top_30_words.word.tolist()))
plt.figure(1,figsize=(8,8))
plt.imshow(wordcloud1)
plt.axis('off')
plt.title("Top 30 words when DJIA goes up/stable ")
plt.show()
wordcloud1.to_file("rise_stable_djia.png")

word cloud when DJIA goes down
wordcloud2 = WordCloud(background_color='white', width=3000,
height=2500).generate(' '.join(djia_down_top_30_words.word.tolist()))
plt.figure(1,figsize=(8,8))
plt.imshow(wordcloud2)
plt.axis('off')
plt.title("Top 30 words when DJIA goes down")
plt.show()
wordcloud2.to_file("down_djia.png")

#Second section : developp the RNN model, start importing the required
packages
##we need to install VC_redist.x64 on the computer for tensorflow
installation
pip install tensorflow
pip install keras

import tensorflow as tf
import keras
from keras.preprocessing.text import Tokenizer
from keras.layers.embeddings import Embedding
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model
from keras.layers import Activation, Bidirectional, Dense, Input,
LSTM,GRU, Embedding, Dropout,Concatenate,Flatten
from keras.layers.normalization import BatchNormalization
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.initializers import glorot_uniform
from keras.optimizers import Adam
```

```
from keras.layers.core import Flatten
from keras.utils import to_categorical
import keras.metrics
from keras.layers.merge import Concatenate
from sklearn.metrics import roc_curve

##tokenize the entire text in the coulumn combined_news
tokenizer_obj = Tokenizer()
tokenizer_obj.fit_on_texts(news_djia['combined_news'].values)

##transform into list
X_train_general_list = X_train['combined_news'].tolist()
X_test_general_list = X_test['combined_news'].tolist()

#research among all the combined new headlines of each date the one
which has the largest number of words,
##this will serve to determine the dimension of the embedding layer
max_length = max(max([len(s.split()) for s in
X_train_general_list]),max([len(s.split()) for s in
X_test_general_list]))
max_length

#define vocabulary size
vocab_size = len(tokenizer_obj.word_index)+1

## tokenize, and assign a number to each word.
X_train_general_seq =
tokenizer_obj.texts_to_sequences(X_train_general_list)
X_test_general_seq=
tokenizer_obj.texts_to_sequences(X_test_general_list)

##since all sentences don't have the same length, we pad them until
with 0 so tht all the sentences will be
##as long as the sentence which has the largest number of words
X_train_general_pad = pad_sequences(X_train_general_seq,
maxlen=max_length, padding = 'post')
X_test_general_pad = pad_sequences(X_test_general_seq,
maxlen=max_length, padding = 'post')

''' if we want to use each column, rather than pool them together
##transform the the columns of data frame into lists, in this code we
have consider the case where the 25 columns
## are used
X_train_general_list = []
X_test_general_list = []

for i in range(len(X_train.columns)):
    a = X_train.iloc[:,i].tolist()
    X_train_general_list.append(a)

    b = X_test.iloc[:,i].tolist()
    X_test_general_list.append(b)

##vectorize text corpus, by turning each text into a sequence of
integers
X_train_general_seq = []
X_test_general_seq = []

for i in range(len(X_train.columns)):
    c = tokenizer_obj.texts_to_sequences(X_train_general_list[i])
    X_train_general_seq.append(c)

    d = tokenizer_obj.texts_to_sequences(X_test_general_list[i])
```



```

        X_test_general_seq.append(d)
    maxtrain = []
    for i in range(len(X_train.columns)):
        maxi_a = max([len(sent_vec) for sent_vec in
            X_train_general_seq[i]]),
        maxtrain.append(maxi_a)

    maxtest = []
    for i in range(len(X_test.columns)):
        maxi_b = max([len(sent_vec) for sent_vec in
            X_test_general_seq[i]]),
        maxtest.append(maxi_b)

    maxlen = max(max(maxtrain), max(maxtest))
    X_train_general_pad = []
    X_test_general_pad = []

    for i in range(len(X_train.columns)):
        e = pad_sequences(X_train_general_seq[i], maxlen=39,
            padding='post')
        X_train_general_pad.append(e)

        f = pad_sequences(X_test_general_seq[i], maxlen=39,
            padding='post')
        X_test_general_pad.append(f)

    print(X_test_general_pad[0].shape[1])
'''

##import a pretrained Glove for word embedding via
https://www.kaggle.com/thanakomsn/glove6b300dtxt
##and create the dictionary that will contain each word and its vector

embeddings_index = {}
f =
open('D:/5504_8240_compressed_glove_6B_300d_txt/glove_6B_300d.txt', encoding = "utf-8")
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))

not_present_list = [] ## create a empty list, it will keep the words
which are in our vocabulary of the data and not in the glove
dictionnayr

vocab_size = len(tokenizer_obj.word_index) + 1 # adding 1 to fit
Keras embedding (requirement)

print('Loaded %s word vectors.' % len(embeddings_index))

embedding_matrix = np.zeros((vocab_size, len(embeddings_index['no'])))
## initialize the embeddind matrix of shape (2169,300) to 0

for word, i in tokenizer_obj.word_index.items(): ## we browser
whole of couples (word , index) that exist in the vocabulary of the
data
    if word in embeddings_index.keys(): ## if the word which is in our
vocabulary is in glove dictionnayr, take its embedding vector

```



```

        embedding_vector = embeddings_index.get(word)
    else:
        not_present_list.append(word)    ##else, add the word to the
not_present_list, its embedding vector is None for the moment
        if embedding_vector is not None:    ## set the embedding vector of
word in the embedding matrix
            embedding_matrix[i] = embedding_vector
        else:
            embedding_matrix[i] = np.zeros(300) ## the embedding vector
for unknown words, is 0

##Build the model
model_rnn = Sequential()
##embedding layer
model_rnn.add(Embedding(name="embed_layer", input_dim = vocab_size,

output_dim=len(embeddings_index['patato']),weights=[embedding_matrix],
                    input_length=max_length,trainable=False))

##first layer of lstm
model_rnn.add(LSTM(units = 120,return_sequences= True))
##apply a batch normalization
model_rnn.add(BatchNormalization())
##second layer of lstm
model_rnn.add(LSTM(units = 64, return_sequences= False))
##apply a batch normalization
model_rnn.add(BatchNormalization())
##sigmoid layer
model_rnn.add(Dense(units = 1,activation = 'sigmoid'))

model_rnn.summary()

##compile the model
model_rnn.compile(loss="binary_crossentropy",
metrics=['acc',keras.metrics.AUC()], optimizer=Adam(0.01))

##fit the model
hist_rnn=model_rnn.fit(X_train_general_pad
,Y_train,validation_data=(X_test_general_pad,Y_test), epochs = 75,
batch_size=32,shuffle=True)

##keep in mind the keys values of this dictionary in order to use the
correct name, they are used in the plotting.
hist_rnn.history

##plot the cost evolution
plt.figure(figsize=(8,8))
fig,axloss1 = plt.subplots()
axloss2 = axloss1.twinx()
axloss1.plot(hist_rnn.history['loss'])
axloss2.plot(hist_rnn.history['val_loss'],color='b')
axloss1.set_title('model loss')
axloss1.set_ylabel('loss')
axloss1.set_xlabel('epoch')
axloss1.legend(['train loss'], loc='upper left')
axloss2.legend(['val loss'], loc='upper right')
plt.savefig("loss_rnn.PNG")
plt.show()

##plot the auc evolution
plt.plot(hist_rnn.history['auc_5'])
plt.plot(hist_rnn.history['val_auc_5'])
plt.title('model auc')
plt.ylabel('auc')

```



```

plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.savefig("auc_rnn.PNG")
plt.show()

##evaluate and polt the accuracy and the auc
loss_rnn, acc_rnn, auc_rnn =
model_rnn.evaluate(X_test_general_pad,Y_test)
print()
print("RNN accuracy: %0.3f " % (acc_rnn))
print("RNN AUC: %0.3f " % (auc_rnn))

##for the RNN model roc curve
Y_pred_rnn = model_rnn.predict(X_test_general_pad).ravel()
fpr_rnn, tpr_rnn, thresholds_rnn = roc_curve(Y_test, Y_pred_rnn)

#Third section : machine learning approaches, import the required
packages
from sklearn.feature_extraction.text import TfidfTransformer,
TfidfVectorizer

###Stemming
from nltk.stem import SnowballStemmer
stemmer = SnowballStemmer('english')
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.metrics import auc,accuracy_score

##apply stemming for the combined_news
news_djia['stem_combined_news'] =
news_djia['combined_news'].apply(lambda x: [stemmer.stem(y) for y in
x.split()])
news_djia['stem_combined_news'] =
news_djia['stem_combined_news'].apply(lambda x: ' '.join(x))
news_djia.head()

##use the stemmed colum to define the train and test set for the
machine learning approaches
X_other_train = news_djia[news_djia['Date'] < '2015-01-
01']['stem_combined_news']
X_other_test = news_djia[news_djia['Date'] > '2014-12-
31']['stem_combined_news']

# using LogisticRegression
logReg_pipeline = Pipeline([
    ("tfidf vectorizer", tfvec),
    ('LogR_model', LogisticRegression())
])

logReg_pipeline.fit(X_other_train, Y_train)
Y_pred_logReg = logReg_pipeline.predict(X_other_test).ravel()
fpr_logReg, tpr_logReg, thresholds_logReg = roc_curve(Y_test,
Y_pred_logReg)
auc_logReg = auc(fpr_logReg, tpr_logReg)
accuracy_logReg = accuracy_score(Y_test,Y_pred_logReg) ##accuracy
print("Logistic Regression accuracy: %0.3f " % (accuracy_logReg))
print("Logistic Regression AUC: %0.3f " % (auc_logReg))
print(pd.crosstab(Y_test.ravel(), Y_pred_logReg, rownames=["Actual"],
colnames=["Predicted"]))

```




```

##accuracy on train set using logReg
Y_pred_logReg_train = logReg_pipeline.predict(X_other_train).ravel()
accuracy_logReg = accuracy_score(Y_train,Y_pred_logReg_train)
print(accuracy_logReg)

# using SVM
svm_pipeline = Pipeline([
    ('tfidf vectorizer', tfvec),
    ('svm_model', svm.LinearSVC())
])

svm_pipeline.fit(X_other_train, Y_train)
Y_pred_svm = svm_pipeline.predict(X_other_test).ravel()
fpr_svm, tpr_svm, thresholds_svm = roc_curve(Y_test, Y_pred_svm)
auc_svm = auc(fpr_svm, tpr_svm)
accuracy_svm = accuracy_score(Y_test,Y_pred_svm) ##accuracy
print("Linear SV accuracy: %0.3f " % (accuracy_svm))
print("Linear SV AUC: %0.3f " % (auc_svm))
print(pd.crosstab(Y_test.ravel(), Y_pred_svm, rownames=["Actual"],
colnames=["Predicted"]))

##accuracy on train set using linear sv
Y_pred_svm_train = svm_pipeline.predict(X_other_train).ravel()
accuracy_svm = accuracy_score(Y_train,Y_pred_svm_train)
print(accuracy_svm)

# using random forest
rf_pipeline = Pipeline([
    ('tfidf vectorizer', tfvec),
    ('rf_model', RandomForestClassifier(n_estimators=400, n_jobs=4))
])

rf_pipeline.fit(X_other_train, Y_train.ravel())
Y_pred_rf = rf_pipeline.predict(X_other_test).ravel()
fpr_rf, tpr_rf, thresholds_rf = roc_curve(Y_test, Y_pred_rf)
auc_rf = auc(fpr_rf, tpr_rf)
accuracy_rf = accuracy_score(Y_test,Y_pred_rf) ##accuracy
print("Random forest accuracy: %0.3f " % (accuracy_rf))
print("Random forest AUC: %0.3f " % (auc_rf))
print(pd.crosstab(Y_test.ravel(), Y_pred_rf, rownames=["Actual"],
colnames=["Predicted"]))

##accuracy on train set using RF
Y_pred_rf_train = rf_pipeline.predict(X_other_train).ravel()
accuracy_rf = accuracy_score(Y_train,Y_pred_rf_train)
print(accuracy_rf)

# using Passive aggressive classifier
pac_pipeline = Pipeline([
    ('tfidf vectorizer', tfvec),
    ('pac', PassiveAggressiveClassifier())
])

pac_pipeline.fit(X_other_train, Y_train)
Y_pred_pac = pac_pipeline.predict(X_other_test).ravel()
fpr_pac, tpr_pac, thresholds_pac = roc_curve(Y_test, Y_pred_pac)
auc_pac = auc(fpr_pac, tpr_pac)
accuracy_pac = accuracy_score(Y_test,Y_pred_pac) ##accuracy
print("Passive aggressive classifier accuracy: %0.3f " %
(accuracy_pac))
print("Passive aggressive classifier AUC: %0.3f " % (auc_pac))
print(pd.crosstab(Y_test.ravel(), Y_pred_pac, rownames=["Actual"],

```



```
colnames=["Predicted"]))

##accuracy on train using PAC
Y_pred_pac_train = pac_pipeline.predict(X_other_train).ravel()
accuracy_pac = accuracy_score(Y_train,Y_pred_pac_train)
print(accuracy_pac)
```



Appendix 4: Program of the specialization

Cours 1 (4 semaines, 20 hours) : Neural network and deep learning	Semaine 1 (2 hours)	What is a neural network? Supervised Learning with Neural Networks Why is Deep Learning taking off? Quiz
	Semaine 2 (8 hours) : Neural Networks Basics	Binary classification : Logistic regression Cost Function, Gradient Descent Vectorization and Broadcasting with Numpy in Python Quiz : Neural Network Basics
	Semaine 3 (5 hours pour terminer):Shallow neural network	Random Initialization Activation function Gradient descent for Neural Networks Quiz : Shallow Neural Networks
	Semaine 4 (5 hours) : Deep Neural Networks	Deep L-layer neural network Forward and Backward Propagation Parameters vs Hyperparameters Quiz : Key concepts on Deep Neural Networks Programming exercise 1 : Building a deep neural network step by step, Programming exercise 2 : Deep Neural Network Application
Cours 2 (3 semaines, 18 hours) : Improving Deep Neural Networks: Hyperparameter tuning, Regularisation and Optimization	Semaine 1 (8 hours) : Practical aspects of Deep Learning	Training, Development and Test sets Bias and Variance problems Regularization : L2 regularization, Dropout regularization, other regularization Vanishing and Exploding gradients Weight Initialization for Deep Networks Numerical approximation of gradients and Gradient checking Quiz : Practical aspects of deep learning Programming exercise 1 : Initialization Programming exercise 2 : Regularization Programming exercise 3 : Gradient checking
	Semaine 2 (5 hours) : Optimization algorithms	Mini-batch gradient descent Exponentially weighted averages Bias correction in exponentially weighted averages



		<p>Other optimization algorithm : Gradient descent with momentum, RMSprop, Adam optimization algorithm</p> <p>Learning rate decay</p> <p>The problem of local optima</p> <p>Quiz : Optimization algorithms</p> <p>Programming exercise: Optimization</p>
	<p>Semaine 3 (5 hours): Hyperparameter tuning, Batch Normalization and Programming Frameworks</p>	<p>Using an appropriate scale to pick hyperparameters</p> <p>Normalizing activations in a network</p> <p>Batch Norm</p> <p>Softmax Regression</p> <p>Introduction to TensorFlow</p> <p>Quiz</p> <p>Programming exercise: TensorFlow</p>
<p>Cours 3 (5 hours): Structuring Machine Learning Projects</p>	<p>Semaine 1 : ML Strategie 1</p>	<p>Orthogonalization</p> <p>Single number evaluation metric</p> <p>Satisficing and Optimizing metric</p> <p>Size of the dev and test sets</p> <p>When to change dev/test sets and metrics</p> <p>Understanding human-level performance</p> <p>Avoidable bias</p> <p>Surpassing human-level performance</p> <p>Improving model performance</p> <p>Quiz: Bird recognition in the city of Peacetopia (case study)</p>
	<p>Semaine 2 (3 hours): ML Strategie 1</p>	<p>Error analysis</p> <p>Bias and Variance data distribution</p> <p>Transfer learning</p> <p>Multi-task learning</p> <p>End-to-end deep learning</p> <p>Quiz: Autonomous driving (case study)</p>



Cours 4 (4 semaines, 20 hours) : Convolutional Neural Networks	Semaine 1 (6 hours) : Foundations of Convolutional Neural Networks	Edge detection Padding,Stride Convolution over volume One layer of CNN Convolution network example Pooling layer Why convolutions Quiz : The basics of ConvNets Programming exercise 1 : Convolutional model step by step with Numpy Programming exercise 2 : Convolutional model : application with TensorFlow
	Semaine 2 (5 hours) : Deep convolutional models: case studies	Classic ConvNets Residual Nets, and Why Residuals Nets Inception Network Practical advices for using ConvNets: Open source Implementation Using, Transfer learning, Data augmentation, State of Computer Vision Quiz : Deep convolutional models Keras Tutorial Programming exercise : Residual networks
	Semaine 3 (4 hours) : Object detection	Object Localization Landmark Detection Convolutional implementation of Sliding window Bounding boxes predictions Intersection over Union and Non-Max Suppression Anchor boxes YOLO Algorithm Region Proposals (R-CNN) Quiz : Detection algorithms Programming exercise : Car detection with YOLO



	Semaine 4 (5 hours) : Special applications: Face recognition & Neural style transfer	<p>Face Verification vs Face recognition</p> <p>One shot learning problem</p> <p>Triplet loss</p> <p>Face verification and Binary Classification</p> <p>Neural Style Transfer : Definition and Cost function (Content cost function and Style Cost Function)</p> <p>1D and 3D generalizations for ConvNets</p> <p>Quiz : Special application : Face recognition and Neural style transfer</p> <p>Programming exercise 1 : Art generation with Neural Style Transfer</p> <p>Programming exercise 2 : Face Recognition</p>
Cours 5 (3 semaines, 15 hours) : Recurrent Neural Network	Semaine 1 (6 hours) :Recurrent Neural Networks	<p>Why sequence models</p> <p>Notation and Reccurent neural network presentation</p> <p>Backpropagation through time</p> <p>Language model and sequence generation</p> <p>Sampling novel sequences</p> <p>Vanishing gradients with RNNs</p> <p>Gated Recurrent Unit (GRU)</p> <p>Long Short Term Memory (LSTM)</p> <p>Bidirectional RNN</p> <p>Quiz : Recurrent Neural Networks</p> <p>Programming exercise 1 : Building a RNN step by step</p> <p>Programming exercise 2 : Dinosaur Island - Character-Level Language Modeling</p> <p>Programming exercise 3 : Jazz improvisation with LSTM</p>

	<p>Semaine 2 (4 hours) : Natural Language Processing & Word Embeddings</p>	<p>Word embedding : properties, embedding matrix Learning Word Embeddings: Word2vec & GloVe Negative sampling Applications using Word Embeddings : Sentiment Classification Debiasing word embeddings Quiz : Natural Language Processing & Word Embeddings Programming exercise 1 : Operations on word vectors - Debiasing Programming exercise 2 : Emojify</p>
	<p>Sequence 3 (5 hours) : Sequence models & Attention mechanism</p>	<p>Basic Models Picking the most likely sentences Beam Search Blue score Attention model Speech recognition Trigger Word Detection Conclusion Quiz: Sequence models & Attention mechanism Programming exercise 1: Neural Machine Translation with Attention Programming exercise 2: Trigger word detection</p>

Appendix 5: Certifications

