SI221 Assignment #4:
Perceptron - 03/03/2020

Instructions

Read all the instructions below carefully before you start working on the assignment.

- Please submit the source code and a pdf report of your work. Each file must have as title: *TP_Perceptron_Student1_Student2*.
- Late assignments will not be corrected.
- You must do this assignment in groups of 2. Please submit no more than one submission per group.
- Send your work in a zipped file to: `giovanna.varni@telecom-paris.fr` and `emile.chapuis@telecom-paris.fr`

Practical assignment objective

This assignement is aimed at coding a perceptron from scratch in order to learn how this simple but powerful linear binary classifier works.

In the following sections you can find a brief summary on what a perceptron is and how it works; a description of the data set you will use; and finally the text of the assignement.

Reminders about perceptron[1]

Historical introduction

The perceptron occupies a special place in the historical development of neural networks: It was the first algorithmically described neural network. Its invention by Rosenblatt, a psychologist, inspired engineers, physicists, and mathematicians alike to devote their research effort to different aspects of neural networks in the 1960s and the 1970s. The perceptron is the simplest form of a neural network used for the classification of data said to be linearly separable (i.e., data that lie on opposite sides of a hyperplane). Basically, it consists of a single neuron with adjustable synaptic weights and bias. The algorithm used to adjust the free parameters of this neural network first appeared in a learning procedure developed by Rosenblatt for his perceptron brain model. Indeed, Rosenblatt proved that if the data used to train the perceptron are drawn from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes. A perceptron is limited to performing

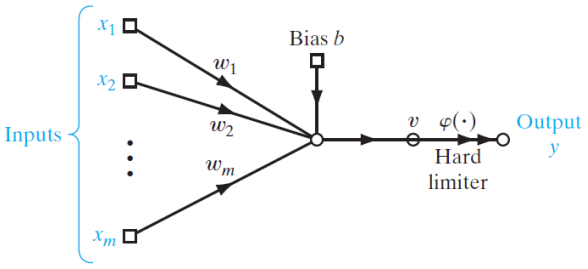
[1] the text and the pictures of this section are taken from *Haykin, O., Neural Networks and Learning Machines, Pearson, 2009*

pattern classification with only two classes. By expanding the output (computation) layer of the perceptron to include more than one neuron, we may correspondingly perform classification with more than two classes.

Working with a perceptron

A peceptron consists of a linear combiner followed by a hard limiter (e.g. the sign function, the Heaviside function or a simple function of these) (see Figure 1).

Figure 1: Perceptron's structure.



The summing node of the neural model computes a linear combination of the inputs applied to its synapses, as well as incorporates an externally applied bias. The resulting sum is applied to a hard limiter. Accordingly, for example if the hard limiter is the Heaviside function, the neuron produces an output equal to 1 if the hard limiter input is positive, and 0 if it is negative. The synaptic weights of the perceptron are denoted by w_1, w_2, ...,w_m. Correspondingly, the inputs applied to the perceptron are denoted by x_1, x_2, ..., x_m. The externally applied bias is denoted by b. From the model, we find that the hard limiter input of the neuron is :

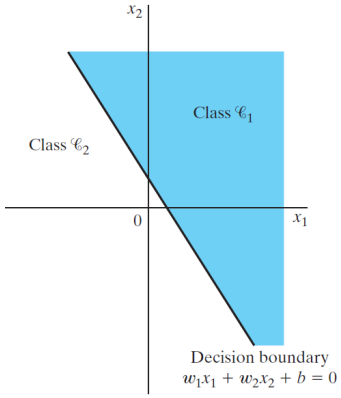
$$v = \sum_{i=1}^{m} w_i x_i + b$$

The goal of the perceptron is to correctly classify the set of externally applied stimuli x_1, x_2, ..., x_m into one of two classes, $\mathfrak{C}_1$ or $\mathfrak{C}_2$.The decision rule for the classification is to assign the sample represented by the inputs x_1, x_2, ..., x_m to class $\mathfrak{C}_1$ if the perceptron output y is 1 and to class $\mathfrak{C}_2$ if it is 0. To develop insight into the behavior of a classifier, it is customary to plot a map of the decision regions in the m-dimensional signal space spanned by the m input variables x_1, x_2, ..., x_m. In the simplest form of the perceptron, there are two decision regions separated by a hyperplane, which is defined by:

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

This is illustrated in Figure 2 for the case of two input variables x_1 and x_2, for which the decision boundary takes the form of a straight line.

Figure 2: Example of hyperplane for a bidimensional, 2-classes classification problem.



A sample (x_1, x_2) that lies above the boundary line is assigned to class $\mathfrak{C}_1$, and a sample (x_1, x_2) that lies below the boundary line is assigned to class $\mathfrak{C}_2$. Note also that the effect of the bias b is merely to shift the decision boundary away from the origin. The synaptic weights w_1, w_2, ..., w_m of the perceptron can be adapted on an iteration by iteration basis called *error-correction rule*. To derive this rule, it is more convenient to work with the modified model in Figure 3. In this model, equivalent to that of Figure 1, the bias $b(n)$ is treated as a synaptic weight driven by a fixed input equal to +1. We may thus define the $(m + 1)$-by-1 input vector:

$$\boldsymbol{x}(n) = [x_1(n), x_2(n), x_m(n), ...1]^T$$

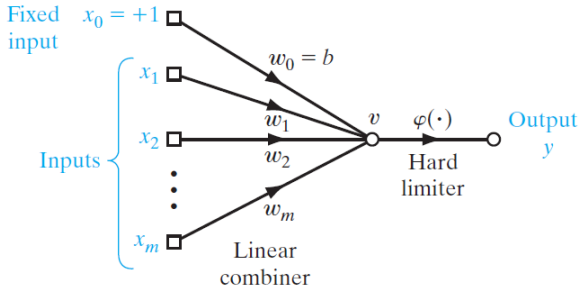
where n denotes the time-step in applying the algorithm. Correspondingly, we define the $(m + 1)$-by-1 weight vector as:

$$\boldsymbol{w}(n) = [w_1(n), w_2(n), w_m(n), ...b]^T$$

Accordingly, the linear combiner output is written in the compact form:

$$v = \boldsymbol{w}^T(n)\boldsymbol{x}(n)$$

Figure 3: Modified model of perceptron



Suppose then that the input variables of the perceptron originate from two linearly separable classes. Let $\mathfrak{H}_1$ be the subspace of training vectors that belong to class $\mathfrak{C}_1$, and let $\mathfrak{H}_2$ be the subspace of training vectors that belong to class $\mathfrak{C}_2$.The union of $\mathfrak{H}_1$ and $\mathfrak{H}_2$ is the complete space denoted by $\mathfrak{H}$. Given the sets of vectors $\mathfrak{H}_1$ and $\mathfrak{H}_2$ to train the classifier, the training process involves the adjustment of the weight vector $\boldsymbol{w}$ in such a way that the two classes $\mathfrak{C}_1$ and $\mathfrak{C}_2$ are linearly separable. That is, there exists a weight vector $\boldsymbol{w}$ such that we may state:

$$\begin{aligned}\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} > 0 \text{ for every input vector } \boldsymbol{x} \text{ belonging to class } \mathfrak{C}_1\\ \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} \leq 0 \text{ for every input vector } \boldsymbol{x} \text{ belonging to class } \mathfrak{C}_2\end{aligned}$$

We have arbitrarily chosen to say that the input vector $\boldsymbol{x}$ belongs to class $\mathfrak{C}_2$ if $\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} = 0$. Given the subsets of training vectors $\mathfrak{H}_1$ and $\mathfrak{H}_2$, the training problem for the perceptron is then to find a weight vector $\boldsymbol{w}$ such that the two inequalities above are satisfied. The algorithm for adapting the weight vector of the elementary perceptron may now be formulated as follows:

- If the nth member of the training set, $\boldsymbol{x}(n)$, is correctly classified by the weight vector $\boldsymbol{w}(n)$ computed at the nth iteration of the algorithm, no correction is made to the weight vector of the perceptron according to the rule:

$$\begin{aligned}\boldsymbol{w}(n+1) = \boldsymbol{w}(n) \text{ if } \boldsymbol{w}^{\boldsymbol{T}}(n)\boldsymbol{x}(n) > 0 \text{ and } \boldsymbol{x}(n) \text{ belongs to class } \mathfrak{C}_1\\ \boldsymbol{w}(n+1) = \boldsymbol{w}(n) \text{ if } \boldsymbol{w}^{\boldsymbol{T}}(n)\boldsymbol{x}(n) \leq 0 \text{ and } \boldsymbol{x}(n) \text{ belongs to class } \mathfrak{C}_2\end{aligned}$$

- Otherwise, the weight vector of the perceptron is updated according to the rule:

$$\begin{aligned}\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \eta(n)\boldsymbol{x}(n) \text{ if } \boldsymbol{w}^{\boldsymbol{T}}(n)\boldsymbol{x}(n) > 0 \text{ and } \boldsymbol{x}(n) \text{ belongs to class } \mathfrak{C}_2\\ \boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \eta(n)\boldsymbol{x}(n) \text{ if } \boldsymbol{w}^{\boldsymbol{T}}(n)\boldsymbol{x}(n) \leq 0 \text{ and } \boldsymbol{x}(n) \text{ belongs to class } \mathfrak{C}_1\end{aligned}$$

  where the learning-rate parameter $\eta(n)$ controls the adjustment applied to the weight vector at iteration n. If $\eta(n) = \eta > 0$, where η is a constant independent of the iteration number n, then we have a fixed-increment adaptation rule for the perceptron. The adaptation of the weight vector $\boldsymbol{w}(n)$ is summed up nicely also in the form:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \eta[d(n) - y(n)]\boldsymbol{x}(n) \tag{1}$$

  where $d(n)$ is the desired output for $\boldsymbol{x}(n)$.

Getting started!

1 Synthetic data

Consider a data set $\{\mathbf{x}_i, y_i\}_{i=1}^{200}$ generated as follows. Each input vector $\mathbf{x}_i \in \mathbb{R}^2$ has an independent bivariate Gaussian distribution $\mathbf{x}_i \sim \mathcal{N}(\mu, \sigma^2\mathbf{I})$ where the mean vector

$\mu \in \mathbb{R}^2$ is given by:

$$\begin{cases} \mu = [-1, 0]\,, \text{ if } y_i = 0 \\ \mu = [1, 0]\,, \text{ if } y_i = 1. \end{cases},$$

Half of the $\mathbf{x}_i$'s have label equal to 1, i.e. $y_i = 1$, and the other half has label equal to 0.

1. Consider three different values $\{0.05, 0.25, 0.50, 0.75\}$ of the noise variance σ^2. For each of these values, run the perceptron over 50 randomly generated datasets, compute the average error $e(\sigma^2)$ and its standard deviation $s(\sigma^2) = \sqrt{\frac{1}{50}\sum_{i=1}^{50}(e_i - e)^2}$, where e_i denotes the fraction of misclassified points. Represent graphically $e(\sigma^2)$ and $s(\sigma^2)$ for the three values of σ^2 (use error bars). Comment.

2. Generate one dataset with $\sigma^2 = 0.15$. A new random data set is now obtained by flipping each label y_i with probability p to obtain $\tilde{y}_i$. Repeat the previous experiments for $p \in \{0\%, 5\%, 10\%, 20\%\}$ and evaluate $e(p)$ and $\sigma^2(p)$.

2 Image data: LANDSAT on Tarascon

You can download this data set at `https://www.dropbox.com/sh/r1qu378v4fylh0b/AAA9AVXwKymcqyDcuIlfyAGDa?dl=0` in the folder TP_Perceptron. The data set consists of 8 channels of satellite images of a geographic region. The size of each pixel is 30m.

0.45-0.52 μ	`landsattarasconC1.ima`
0.52-0.60 μ	`landsattarasconC2.ima`
0.63-0.69 μ	`landsattarasconC3.ima`
0.75-0.90 μ (Near Infra Red)	`landsattarasconC4.ima`
2.08 2-35 μ (Medium Infra Red)	`landsattarasconC5.ima`
10.40-12.50 μ (Thermal Infra Red)	`landsattarasconC6.ima`
1.55-1.75 μ (Medium Infra Red)	`landsattarasconC7.ima`
0.52-0.90 μ (Panchromatic)	`landsattarasconC8.ima`

The following Python code will allow you to read and display images of the data set:

```
import tiilab
img=tiilab.imz2mat(image_name)
tiilab.visusar(img[0])
```

Labeling data

To understand how a perceptron can learn to classify each pixel of an image in two classes (land or water), consider the following image of the data set: `landsattarasconC4.ima.`

Write the function `create_labels` to associate a label to each pixel of this image: label 1 will be used to indicate a water's pixel (dark), label 2 will be used to indicate a land's pixel (light). The function has to evaluate the value of each pixel of the input image, and if that value is lower than 30 it will return label 1, 2 otherwise. The function will return a matrix containing all the labels.

Implementing the perceptron's error-correction rule

Focusing always on the image `landsattarasconC4.ima`, that will be used here as training set, write a the function `error_correction` implemeting the perceptron's rule described above. This function will take as inputs: the image, the matrix of the labels, η, and the weigths. Fix the value of η to 0.01. Then:

- Compute how many epochs the algorithm needs to converge. We call epoch using *once* the totality of the pixels of the image.
- Compute which percentage of pixels are classified in a right and wrong way, respectively.

Change now the value of η (but keep η constant with respect to n). What happens? Explain why the perceptron provides always good results.

Effect of an error on a value

Modify now the matrix containing the labels of the pixels of `landsattarasconC4.ima` in the following way: associate label 1 to each pixel having a value of 110. This basically means blinding the sensor to capture images for this value. Answer the following questions:

- How many pixels are affected by this change?
- Fix a maximum number of epochs for the perceptron. Compute the error rate.

Effect of saturation

Modify now the matrix containing the labels of the pixels of `landsattarasconC4.ima` in the following way: associate label 1 to each pixel having a value greater than 140. Answer the previous questions for this new setup.