https://doi.org/10.17163/abyaups.146.3

Part III

Historic Algorithms

We study here three algorithms developed in the 40s, 50s and 60s of the twentieth century; and we do so because they deserve to be studied, even now, after many years of their appearance. The McCulloch-Pitts neuron, studied in Chapter 7, is the mathematical unit from which many algorithms have been built, including the modern *Neural Networks* and *Deep Learning* algorithms; the *Perceptron* algorithm, Chapter 8, was the first one with the ability to "learn" (modify its weights according to the data); and *Adaline*, Chapter 9, is a pioneer in the application of neural computation to regression problems.

Most likely, you will never use these algorithms in real applications nowadays. Knowing them, however, will help you appreciate the great effort of human ingenuity that has led us to our modern intelligent systems. Also, understanding the behaviour of McCulloch-Pitts neurons will be the basis for a deep comprehension of Neural Networks in coming chapters.

Chapter 7

McCulloch-Pitts Neuron

7.1 Theoretical Briefing

Biological neurons. A neuron is a cell. A cell specialised for signal processing and transmission. In Figure 7.1, we show the morphology and basic components of a neuron.

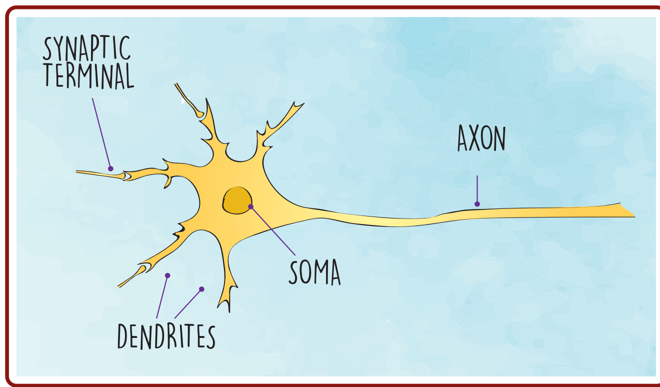


Figure 7.1: Scheme of the biological neuron

The neuron consists of a central **soma**, surrounded by branches or extensions called **dendrites**. The signals coming from other neurons through their **synaptic terminals**, arrive at these dendrites. Then, our neuron itself can emit a signal through its **axon**. At its end (not shown), the axon has synaptic terminals as well, which in turn deliver signals to other neurons' dendrites. Nevertheless, there is a detail: synaptic terminals and dendrites are not connected physically. Looking closer, we would see that there is a gap between them, a gap called **synaptic cleft**. The communication across the synaptic cleft is carried out by means of intermediate molecules, called **neurotransmitters**. Neurotransmitters are stored in tiny bags called **synaptic vesicles**, located in the synaptic terminals. When a signal arrives through the synaptic terminal, the

neurotransmitters act as "filters" of the signal: if there are scarce neurotransmitters, for example, you will have a weak signal entering the neuron, however strong the original signal was.

Firing of neurons. Imagine the following experiment: you insert an electrode to stimulate the interior of a neuron and another electrode to measure its response (in millivolts). What would you get? Here are some results: even without any stimulus, there is a voltage of around -70 mV in the interior of the neuron with respect to the exterior. (This voltage is called **resting potential**.) If you apply a stimulus, you can depolarise the neuron (with a positive stimulus) or hyperpolarise it (with a negative one). There is a special value, -50 mV, called the **threshold potential**. If you do not surpass this value, the neuron reacts only proportionally to the stimulus applied. But if you do surpass it, the response of the neuron is kind of exaggerated: its potential shoots up to +40 mV! This value is called the **action potential**. We say in this case that the neuron has **fired**.

McCulloch-Pitts neuron. The American scientists Warren McCulloch (who studied philosophy, psychology and medicine) and Walter Pitts (logician, autodidact) created in 1943 the first model of an artificial neural network, made up from abstract units now called **McCulloch-Pitts neurons**. In Figure 7.2, we show a sketch of a McCulloch-Pitts neuron.

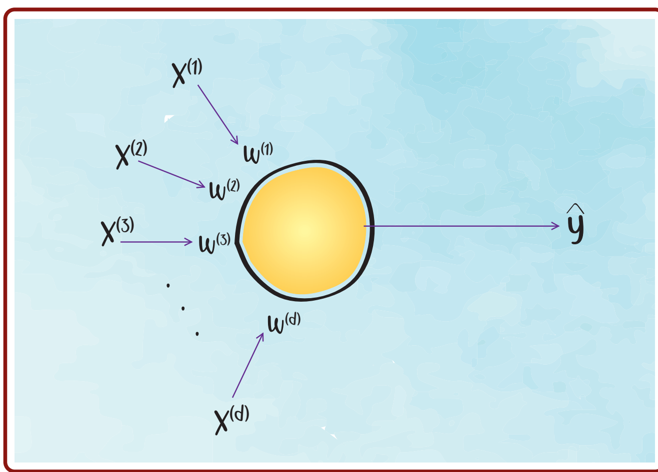


FIGURE 7.2: Scheme of the McCulloch-Pitts neuron

This abstract unit mimics the biological neuron in several ways: it receives an input signal (mathematically, a feature vector $\mathbf{x}$). Each one of the components of the feature vector goes through a "dendrite" and is multiplied by a **weight** (analogous to the neurotransmitters). These contributions are summed up in the "soma", represented here by the central circle. The outgoing arrow represents the axon, which delivers the output signal, $\hat{y}$. The sum obtained in the "soma" is given by:

$$s = \sum_{j=1}^{d} x^{(j)} w^{(j)} \tag{7.1}$$

Or, in a compact way:

$$s = \mathbf{x}^T \cdot \mathbf{w} \tag{7.2}$$

where

$$\mathbf{w} = \begin{bmatrix} w^{(1)} & w^{(2)} & w^{(3)} & \cdots & w^{(d)} \end{bmatrix}^T$$

is the weight vector of the neuron. Other elements "copied" from nature are the threshold and the firing: if you surpass a threshold value, the neuron fires. Otherwise, it does not. This is, mathematically:

$$y = \begin{cases} 1 & \text{if} \quad s > \theta \\ 0 & \text{if} \quad s \leq \theta \end{cases} \tag{7.3}$$

where "1" means *firing*, "0" means *no firing* and θ is the threshold.

Working with data matrices. As usual, we can enter several feature vectors at once, by arranging them in a data matrix $\mathbf{X}$. In this case, we could apply the McCulloch-Pitts criteria simultaneously by using matrix calculus:

$$\mathbf{s} = \mathbf{X} \cdot \mathbf{w} \tag{7.4}$$

where $\mathbf{s}$ will be a $m \times 1$ vector. Then, we should apply the "firing" criterion (Equation 7.3) to each of the components of $\mathbf{s}$, in order to obtain the vector $\hat{\mathbf{y}}$ of outputs (dimension: $m \times 1$) for the m instances.

McCulloch-Pitts neuron as a classifier. Since the output of a McCulloch-Pitts neuron can be either a "0" or a "1", you can use it to solve classification problems with two classes. For instance, suppose you are interested in classifying images from men and women by genre. To do this using a McCulloch-Pitts neuron, assign the label "1" to women and "0" to men (or the other way round, if you feel offended.)

Activation functions. A function that takes the sum s (see Equation 7.1) and computes the output y of a neuron is called an **activation function**, represented by $f(s)$. This is,

$$y = f(s) \tag{7.5}$$

The activation function corresponding to Equation 7.3 is:

$$f(s) = \begin{cases} 1 & \text{if} \quad s > \theta \\ 0 & \text{if} \quad s \leq \theta \end{cases} \tag{7.6}$$

Of course, this is not the only activation function that exists. In the coming chapters we will see other possibilities.

7.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let $\mathbf{X}$ be the matrix and $\mathbf{y}$ be the vector

$$\mathbf{X} = \begin{bmatrix} -10 & 6 \\ -5 & 2 \\ -1 & 5 \\ 2 & 9 \\ 2 & -3 \\ 5 & 8 \\ 6 & -1 \\ 8 & -4 \\ 9 & 9 \\ 11 & 5 \\ 12 & 10 \\ 13 & 13 \\ 14 & -3 \\ 19 & 2 \\ 21 & 3 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

of data for a classification problem, where $\mathbf{y}$ is the vector of actual labels.

1. Locate, in a Cartesian plane, the feature vectors. Use different colours and markers to represent their actual classes.
2. Compute $\hat{\mathbf{y}}$, the vector of predictions given for the data in $\mathbf{X}$ by a McCulloch-Pitts neuron with threshold θ=0 and weight vector

$$\mathbf{w} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

3. In the same Cartesian plane of numeral 1, locate the misclassified instances by overlapping their predicted markers over the actual ones.
4. Compute the confusion matrix of this prediction.
5. Compute the number of true positives, false positives, false negatives and true negatives for class "0".

7.3 Practical in MATLAB

General Objective:

To make the student capable of computing predictions for a classification problem using a McCulloch-Pitts neuron with given weights and threshold.

Specific Objectives:

- The student should be able to implement a McCulloch-Pitts neuron in MATLAB.
- The student should be able to use the implemented neuron to compute the predictions for a set of feature vectors.
- The student should be able to evaluate the performance of McCulloch-Pitts neurons.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr7_nameLastname.m`. You will write your code for the next numerals in this script.
2. Load into the *Workspace* the matrices contained in the file `matricesPr7.mat`. This file can be found in the folder *Data Sets*.
3. Write a function "`myMcC_P(X,w,Theta)`", which returns a vector `yhat` of the predictions given for a data matrix `X` by a McCulloch-Pitts neuron with weight vector `w` and a threshold `Theta`. (*Challenge*: Do not use `for` loops to write this function.)
4. In numeral 2 you loaded four 3-D vectors: `x1`, `x2`, `x3` and `x4`. Assemble a data matrix "`X1`" with these vectors, in the standard Machine-Learning way. Then, use your function `myMcC_P` to compute the predictions for the data in `X1` by a McCulloch-Pitts neuron with threshold $\theta = 3$ and weight

vector

$$\mathbf{w} = \begin{bmatrix} 2 \\ -1 \\ -3 \end{bmatrix}$$

5. In numeral 2 you loaded two 20-D vectors: `x5` and `x6`. Assemble a data matrix “`X2`” with these vectors, in the standard Machine-Learning way. Then, use your function `myMcC_P` to compute the predictions for the data in `X2` by a McCulloch-Pitts neuron with threshold $\theta = 1$ and weight vector `w2`, also loaded in numeral 2.

6. In numeral 2 you loaded a data matrix `X` with 2-D feature vectors, and a `y` vector of actual labels. Use your function `myMcC_P` to compute the predictions for the data in **X** by a McCulloch-Pitts neuron with threshold $\theta = 10$ and weight vector

$$\mathbf{w} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$$

7. Write a function “`plotClasses_McC_P(X,y,yhat)`”, which does not return anything but creates a plot with the feature vectors in `X` (suppose they are 2-D), with the following characteristics: vectors of class “0”, black; class “1”, red; actual classification (given by the labels in `y`), circles; predicted classification (given by the labels in `yhat`), stars. Use the MATLAB function `legend` to add a legend to the plot, indicating the actual and predicted classes. (You can look at the answer of numeral 8 to get a better idea of what is expected of your function `plotClasses_McC_P`.)

8. Call your function `plotClasses_McC_P` from the script, passing as arguments the matrix `X` and the vector `y` mentioned in numeral 6, and the vector `yhat` computed in the same numeral.

9. Compute the misclassification error for the predictions obtained in numeral 6.

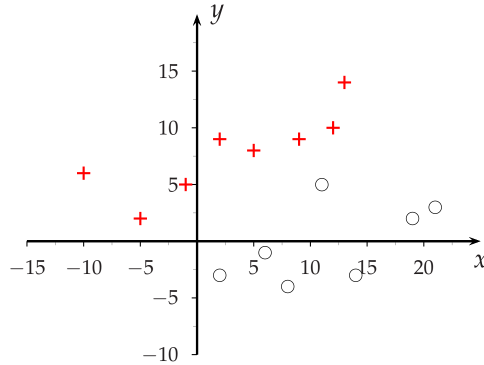
10. Compute the confusion matrix for the predictions obtained in numeral 6.

11. Compute the number of true positives, false positives, false negatives and true negatives for class “1”.

7.4 Answers to selected exercises

Exercises

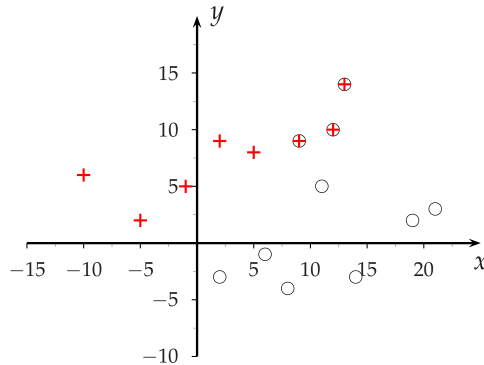
1. (a)



(b) We present the vector transposed, to save space:

$$\mathbf{y}^T = [1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

(c)



(d)

		Predicted	
		0	**1**
Actual	**0**	7	0
	1	3	5

(e) True positives: 7
False positives: 3

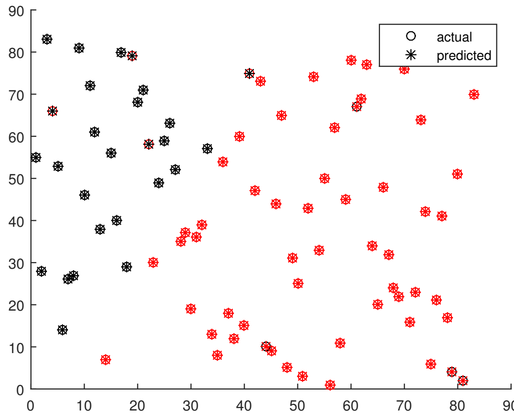
False negatives: 0
True negatives: 5

Practical

4. $\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T$

5. $\begin{bmatrix} 1 & 0 \end{bmatrix}^T$

8.



9. 0.0964

10.

		Predicted	
		0	**1**
Actual	**0**	23	4
	1	4	52

11. True positives: 52
    False positives: 4
    False negatives: 4
    True negatives: 23

CHAPTER 8

PERCEPTRON

8.1 Theoretical Briefing

Artificial neuron for the perceptron. With subtle differences (created for mathematical reasons) the neuron for the perceptron is basically a McCulloch-Pitts neuron. In Figure 8.1, we depict the perceptron unit.

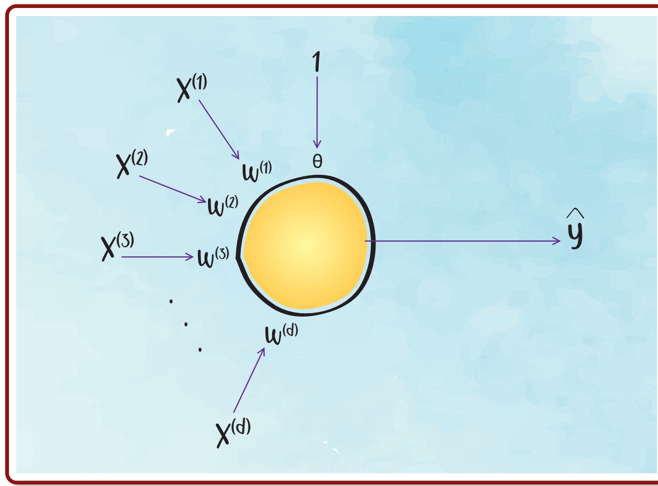


FIGURE 8.1: The artificial neuron for the perceptron

Sum in a perceptron. As you might have noted, the neuron in Figure 8.1 incorporates an additional "dendrite" or entry. The value of the signal for this entry is always 1. Always. Therefore, quite reasonably, this artificial dendrite is called a **bias** entry. The corresponding weight is called θ and should remind you of the threshold of the McCulloch-Pitts neuron. The sum inside the "soma" of the neuron is, then:

$$s = \theta + \sum_{j=1}^{d} x^{(j)} w^{(j)} \tag{8.1}$$

We can write this equation in a more compact way:

$$s = \mathbf{x}_{aug}^T \times \mathbf{w} \tag{8.2}$$

where

$$\mathbf{x}_{aug} = \begin{bmatrix} 1 & x^{(1)} & x^{(2)} & x^{(3)} & \cdots & x^{(d)} \end{bmatrix}^T$$

and

$$\mathbf{w} = \begin{bmatrix} \theta & w^{(1)} & w^{(2)} & w^{(3)} & \cdots & w^{(d)} \end{bmatrix}^T$$

both of which are $(d+1) \times 1$ vectors. The "aug" subscript stands for "augmented", since we have appended a "1" at the top of the feature vector.

Activation function for the perceptron. The activation function, $f(s)$, for the perceptron is:

$$f(s) = \begin{cases} 1 & \text{if} \quad s > 0 \\ -1 & \text{if} \quad s \leq 0 \end{cases} \tag{8.3}$$

Sum for a data matrix. Again, it is possible to collect several feature vectors in a data matrix $\mathbf{X}$ and use this whole matrix as the entry for the perceptron. The vector of sums for such a matrix would be, then:

$$\mathbf{s} = \mathbf{X}_{aug} \times \mathbf{w} \tag{8.4}$$

where

$$\mathbf{X}_{aug} = \begin{bmatrix} 1 & \longleftarrow & \mathbf{x}_1^T & \longrightarrow \\ 1 & \longleftarrow & \mathbf{x}_2^T & \longrightarrow \\ \vdots & & \vdots & \\ 1 & \longleftarrow & \mathbf{x}_m^T & \longrightarrow \end{bmatrix} \tag{8.5}$$

(In other words, $\mathbf{X}_{aug}$ is the just the same usual matrix $\mathbf{X}$, but with a column of ones appended at its left.)

The two-dimensional case. For the two-dimensional case, we have the chance to visualise what a perceptron does. To see how to do this, consider Equations 8.1 and 8.3. From the latter, you can infer that something special must happen at 0. Therefore, taking into account Equation 8.1, you can infer that something special must happen when

$$\theta + \sum_{j=1}^{d} x^{(j)} w^{(j)} = 0 \tag{8.6}$$

Since $d = 2$ in the two-dimensional case, the last equation turns out to be

$$\theta + x^{(1)}w^{(1)} + x^{(2)}w^{(2)} = 0 \tag{8.7}$$

But this is the equation of a straight line! To see how this is true, let us solve for $x^{(2)}$. By doing this, we have:

$$x^{(2)} = \frac{-w^{(1)}}{w^{(2)}}x^{(1)} + \frac{-\theta}{w_2} \tag{8.8}$$

which is totally analogous to the equation $y = mx + b$, the equation of a straight line. So, a *weight vector corresponds to a* ***straight line*** *in the two-dimensional case*. The aim of the perceptron algorithm will be then to find a straight line which separates the class "1" vectors from the class "-1" vectors. Generalising, we will state that a weight vector corresponds to a **plane** in the three-dimensional case and that, in higher-dimensional spaces, it corresponds to a **hyperplane**; both on a mission to separate class "1" vectors from class "-1" vectors.

The perceptron algorithm. Created in 1957 by American psychologist Frank Rosenblatt, this was the first algorithm to perform "learning". That is, to "tune" the weight vector until the classification is correct (graphically, until the line, plane or hyperplane, separates the instances correctly.) As input, the algorithm takes a data set made up by $\mathbf{X}$ (dimension $m \times d$) and $\mathbf{y}$ (dimension $m \times 1$), and returns the "tuned" weight vector $\mathbf{w}$. In Algorithm 1 we show the perceptron learning algorithm in pseudocode.

In Algorithm 1, $\mathbf{x}_{aug}^{(i)}$ represents the ith feature vector, augmented. This is,

$$\mathbf{x}_{aug}^{(i)} = \begin{bmatrix} 1 & x_i^{(1)} & x_i^{(2)} & x_i^{(3)} & \cdots & x_i^{(d)} \end{bmatrix}^T$$

The stopping criterion for the perceptron algorithm, shown in line 3 of Algorithm 1, can be changed in order to soften the expectations of the algorithm. That is, instead of looking for perfection, the algorithm is only required to reach a "goal error", to be set by the user. For doing this, just change line 3 of Algorithm 1 for:

while misclassification error > goal error **do**

A mathematical expression such as that of line 6 is called a *learning rule*. Note that in the second term of this expression, a **scalar multiplication** appears. Check your books on physics or linear algebra to recall this concept, if needed.

Algorithm 1 The perceptron learning algorithm

1: $\mathbf{w} \leftarrow \mathbf{w}_0$ %random initialisation
2: Compute current misclassification error
3: **while** misclassification error $\neq 0$ **do**
4: **for** $i = 1 : m$ **do**
5: **if** $\widehat{y}_i \neq y_i$ **then**
6: $\mathbf{w} \leftarrow \mathbf{w} + y_i \cdot \mathbf{x}_{aug}^{(i)}$
7: **end if**
8: **end for**
9: Compute current misclassification error
10: **end while**

8.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let

$$\mathbf{X} = \begin{bmatrix} 0 & -2 \\ 1 & 4 \\ 2 & -1 \\ 4 & 4 \\ 4 & -3 \\ 5 & 5 \\ 6 & 2 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}^T$$

be the data matrix and the vector of actual labels for a classification problem, respectively.

1. Represent the feature vectors in a Cartesian plane. Use different marker shapes to differentiate among classes: little circles for vectors with "-1" label and plus signs for vectors with "1" label.

2. Take

$$\mathbf{w} = \begin{bmatrix} -7 \\ 2 \\ 1 \end{bmatrix}$$

as an initial weight vector. Draw the straight line that corresponds to this vector, in the same plot you made in numeral 1.

3. Execute, by hand, the Perceptron algorithm for these data and complete the Table 8.1 (in the first column write down how many times the algorithm has entered the **while** loop at that moment). NOTE: This table is incomplete, since the algorithm will have to enter the **while** loop four times until having all the instances correctly classified. Nevertheless, this is enough as a didactic exercise.

TABLE 8.1

Times at while	i	$\mathbf{x}_i^T$	$\hat{\mathbf{y}}_i$	Is $\hat{\mathbf{y}}_i \neq \mathbf{y}_i$? (y/n)	$\mathbf{w}^T$
0	–	–	–	–	[-7 2 1]
Misclassification error:					
1	1				
	2				
	3				
	4				
	5				
	6				
	7				
Misclassification error:					
2	1				
	2				[-5 6 -4]

8.3 Practical in MATLAB

General Objective:

To make the student capable of training and using a perceptron.

Specific Objectives:

- The student should be able to plot the perceptron solution together with the data for a classification problem, and interpret this plot.
- The student should be able to implement a perceptron to get predictions for a data matrix.
- The student should be able to implement the perceptron learning algorithm.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr8_nameLastname.m`. You will write your code for the next numerals in this script.

2. Load into the *Workspace* the matrices contained in the file `matricesPr8.mat`. This file can be found in the folder *Data Sets*.

3. In the folder *"MATLAB Functions"*, you will find a function `plotPerceptron.m`. Save it in your *Current Folder* and use it to make a plot of the data loaded in numeral 2, passing as weight vector:

$$\mathbf{w} = \begin{bmatrix} -30 \\ 2 \\ 3 \end{bmatrix}$$

4. Create a function "`perceptronOutput(X,w)`", which takes a data matrix `X` and a weight vector `w`, and computes a vector "`yhat`" containing the predictions given by the corresponding perceptron.

5. Use your function `perceptronOutput` to compute the predictions of the perceptron with the weight vector given in numeral 3, over the matrix `X` loaded in numeral 2.

6. Create a function "`perceptronLearning(X,y,w_ini)`", which takes a data matrix `X`, a vector of actual labels `y` and an initial weight vector `w_ini` and executes the Perceptron algorithm. This function must return the weight vectors generated during the learning process, collected in a matrix which we will call "`W_values`". This matrix will contain all the values of `w`, transposed, one below the other, starting with the initial value `w_ini`. Also, this function must display in the *Command Window* the following things: a message telling how many times it is entering the `while` loop; which instance it is analysing at that moment; the initial misclassification error; and the misclassification error each time it gets out the `while` loop. You can see the answer to numeral 7 to get a better idea of what we are asking for.

7. Use your function `perceptronLearning` with the following data:

$$\mathbf{X} = \begin{bmatrix} -1 & -1 \\ 1 & -1 \\ -1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} -1 & -1 & -1 & 1 \end{bmatrix}^T$$

$$\mathbf{w}_{ini} = \begin{bmatrix} 0.5 & 1 & 1 \end{bmatrix}^T$$

(a) What is the resulting `W_values` matrix?

(b) What are the messages printed in console?

8. Try your function `perceptronLearning` on the data loaded in numeral 2 to obtain a matrix "`W_values2`".

   (a) How many times does the algorithm enter the `while` loop?

   (b) What is the initial misclassification error?

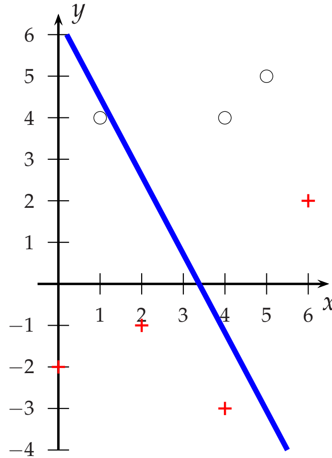
   (c) What is the misclassification error before entering the `while` loop for the last time?

9. Use the function `plotPerceptron` to plot the data with the last six weight vectors in `W_values2` and arrange them in a 2×3 array using **subplot**.

8.4 Answers to selected exercises

Exercises

2.

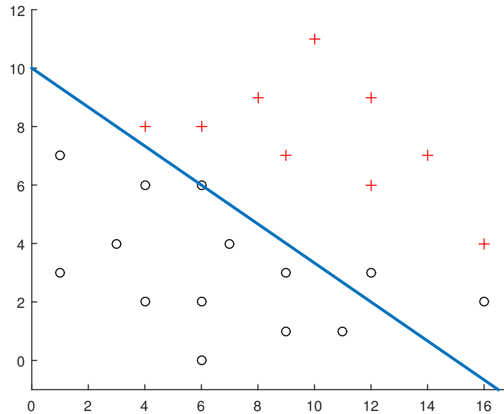


3.

Times at while	i	$\mathbf{x}_i^T$	$\widehat{\mathbf{y}}_i$	**Is** $\widehat{\mathbf{y}}_i \neq \mathbf{y}_i$ **? (y/n)**	$\mathbf{w}^T$
0	–	–	–	–	[-7 2 1]
Misclassification error: 0.7143					
1	1	[0 -2]	-1	yes	[-6 2 -1]
	2	[1 4]	-1	no	[-6 2 -1]
	3	[2 -1]	-1	yes	[-5 4 -2]
	4	[4 4]	1	yes	[-6 0 -6]
	5	[4 -3]	1	no	[-6 0 -6]
	6	[5 5]	-1	no	[-6 0 -6]
	7	[6 2]	-1	yes	[-5 6 -4]
Misclassification error: 0.2857					
2	1	[0 -2]	1	no	[-5 6 -4]
	2	[1 4]	-1	no	[-5 6 -4]

Practical

3.



5. Your `yhat` should be a 23×1 vector with twelve numbers "-1" followed by eleven numbers "1".

7. (a) `W_values` $= \begin{bmatrix} 0.5 & 1 & 1 \\ -0.5 & 0 & 2 \\ -1.5 & 1 & 1 \end{bmatrix}$

   (b)

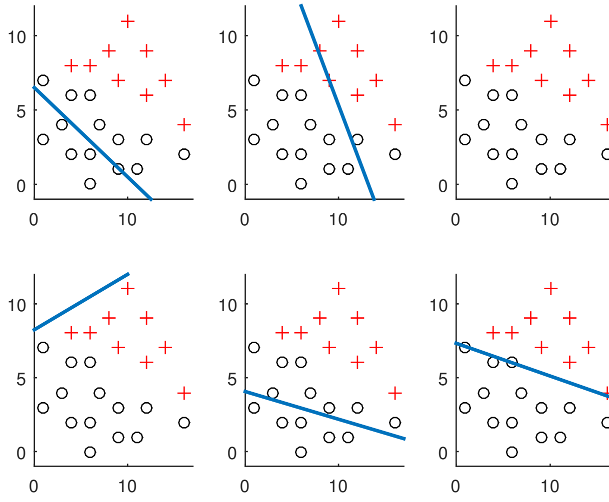
```
The initial misclassification error is: 0.500000
Entering the while for the 1th. time, instance 1...
Entering the while for the 1th. time, instance 2...
Entering the while for the 1th. time, instance 3...
Entering the while for the 1th. time, instance 4...
The misclassification error now is: 0.000000
```

8. (a) 69

   (b) 0.086957

   (c) 0.347826

9.



Chapter 9

Adaline

9.1 Theoretical Briefing

Artificial neuron for the ADALINE. The artificial neuron for this algorithm will be exactly the same as that for the perceptron (see Figure 8.1.)

Sum in an ADALINE. The sum in the ADALINE is the same as that for the perceptron (see Equations 8.1, 8.2 and 8.4.)

Activation function. Now, this is the first difference between the perceptron and the ADALINE. For the latter, the activation function is simply given by:

$$f(s) = s \tag{9.1}$$

Note that whereas the result from the activation function in the perceptron is discrete (i.e., it can take only discrete values, namely -1 and 1), the output from the activation function shown in Equation 9.1 is *continuous*. This implies that $\widehat{y} \in \Re$. In other words, *ADALINE is a regression algorithm*.

Error for ADALINE. Since ADALINE is a regression algorithm, the error to be computed here should be the mean squared error (see Equation 4.1.) However, in order to have some terms simplified when doing the math, a new error has been defined, with a small modification from the original one:

$$\varepsilon = \frac{1}{2}\sum_{i=1}^{m}(y_i - \widehat{y}_i)^2 \tag{9.2}$$

We will use Equation 9.2 when computing the error in all this section.

The ADALINE algorithm. Standing for Adaptive Linear Neuron, ADALINE was developed by American engineers Bernard Widrow and Ted Hoff in 1960. The corresponding learning algorithm is shown in Algorithm 2.

A new parameter has been introduced in line 5 of this algorithm, the so-called **learning rate**, represented here by the Greek letter γ. By tuning this parameter, a user can regulate how long the step will be from the previous to the next $\mathbf{w}$ vector. Using this mathematical device, a trade-off should be attained: for too large values of γ, the algorithm can become unstable and miss its target; for too small ones, it could take too long for the algorithm to hit it.

Algorithm 2 The ADALINE learning algorithm

1: $\mathbf{w} \leftarrow \mathbf{w}_0$ %random initialisation
2: Compute current error
3: **while** error $>$ goal error **do**
4: **for** $i = 1 : m$ **do**
5: $\mathbf{w} \leftarrow \mathbf{w} + \gamma(\mathbf{y}_i - \widehat{\mathbf{y}}_i)\mathbf{x}_{aug}^{(i)}$
6: **end for**
7: Compute current error
8: **end while**

9.2 Exercises

Solve the following exercises. You can use a calculator. Do not use MATLAB here

Let

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix}^T$$

be the data for a regression problem.

1. Execute, by hand, the ADALINE algorithm, and complete the Table 9.1 with the first iterations of the algorithm. Take $\gamma = 0.3$ as the learning rate, a goal error of 0.001 and an initial weight vector of

$$\mathbf{w}_o = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$$

(NOTE: This table is incomplete, since the algorithm will have to enter twenty-one times the while loop until overcoming the goal error. Nevertheless, this is enough as a didactic exercise.)

TABLE 9.1

Times at while	i	x_i^T	y_i	$\hat{y}_i$	w^T
0	–	–	–	–	[1 1 1 1]
Mean squared error (ADALINE): 13.5					
1	1	[0 0 1]	1	2	[0.7 1 1 0.7]
	2	[0 1 0]	2	1.7	
	3	[0 1 1]	3		
	4	[1 0 0]			
	5				
	6				
	7				
Mean squared error (ADALINE):					
2	1				
	2				

9.3 Practical in MATLAB

General Objective:

To make the student capable of training and using an ADALINE.

Specific Objectives:

- The student should be able to implement an ADALINE to get predictions for a data matrix
- The student should be able to implement the ADALINE learning algorithm.
- The student should be able to graphically represent the ADALINE solution for a regression problem with one-dimensional data.

Materials:

Computer, MATLAB.

Procedure:

Solve the following exercises in MATLAB

1. Create a script and save it following the format `Pr9_nameLastname.m`. You will write your code for the next numerals in this script.

2. Load into the *Workspace* the matrices contained in the file `matricesPr9.mat`. This file can be found in the folder *Data Sets*.

3. Create a function "`adalineOutput(X,w)`", which takes a data matrix `X` and a weight vector `w`, and computes the vector `yhat` of predictions given by the corresponding ADALINE.

4. Use your function `adalineOutput` to compute the predictions of an ADALINE with weight vector

$$\mathbf{w} = \begin{bmatrix} -2 \\ 5 \end{bmatrix}$$

   over the four first instances in the matrix `X` loaded in numeral 2.

5. Create a function "`adalineLearning(X,y,w_ini,gamma,goal_error)`", which takes a data matrix `X`, a vector of actual labels `y`, an initial weight vector `w_ini`, a learning rate `gamma` and a `goal_error` to overcome, and executes the ADALINE algorithm. This function must return the weight vectors generated during the learning process, collected in a matrix which we will call "`W_values`". This matrix will contain all the values of `w`, transposed, one below the other, starting with the initial value `w_ini`. Also, this function must display in the *Command Window* the following things: a message telling how many times it is entering the `while` loop; which instance it is analysing at that moment; the initial error; and the error each time it gets out the `while` loop. You can see the answer to numeral 6 to get a better idea of what we are asking for.

6. Use your function `adalineLearning` with learning rate `gamma`=0.2 and a goal error of 2 with the following data:

$$\mathbf{X} = \begin{bmatrix} -1 & -1 \\ -1 & 2 \\ 0 & 0 \\ 1 & 0 \\ 1 & -1 \\ 3 & 0 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 1 & 3 & 2 & 2 & 2 & 3 \end{bmatrix}^T$$

$$\mathbf{w}_{ini} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

(a) What is the resulting `W_values` matrix?

(b) What are the messages printed in console?

7. Now you will work with some data from a repository, the UCI Machine Learning Repository. Go to: `http://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength` and download the Excel file "`Concrete_Data.xls`", which is located in the *Data Folder*. Save this file in your *Current Folder*.

8. Once you have an Excel file in your Current Folder, you can read it using the MATLAB command `xlsread`. Use this command to store the data in `Concrete_Data.xls` in a matrix you will call "`dataC`".

9. Form matrices "`X_concrete`" and "`y_concrete`" extracting the adequate columns from `dataC`. To know which attributes are the features of the vectors and which is the dependent variable, read the *Attribute Information* section in the web page you opened above.

10. Use your function `adalineLearning` on this data, with learning rate `gamma` = 0.000001 and a goal error of 82000. Initialise the algorithm with a vector `w_ini` of suitable size (i.e., 9×1) and which entries are 0.005 (all of them.). This is,

$$\texttt{w_ini} = \begin{bmatrix} 0.005 \\ 0.005 \\ \vdots \\ 0.005 \end{bmatrix}_{9\times 1}$$

(a) How many times does the algorithm enter the `while` loop?

(b) What is the size of the resulting `W_values` matrix?

(c) What is the final error?

11. Use your function `adalineLearning` on the matrix `X` and the vector `y` that you loaded in numeral 2, with learning rate `gamma`=0.01 and a goal error of 21. Initialise the algorithm with a vector

$$\texttt{w_ini} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$$

(a) How many times does the algorithm enter the `while` loop?

(b) What is the size of the resulting `W_values` matrix?

(c) What is the final error?

12. Plot the data together with the models given by the weight vectors number 1, 75, 149 and 373 (contained in `W_values`) and arrange them in a 2×3 array using `subplot`. See the answer to this numeral to get a better idea of what you are meant to achieve. (*Hint*: In order to plot the models, you can create a vector "`x_plot`" using the MATLAB command **linspace**, and then compute the value of $\widehat{\mathbf{y}}$ for each of the values of this vector. In this way, you will have a "continuous" set of points, which you can plot using `plot`).

9.4 Answers to selected exercises

Exercises

1.

Times at while	i	x_i^T	y_i	$\hat{y}_i$	w^T
0	–	–	–	–	[1 1 1 1]
Mean squared error (ADALINE): 13.5					
1	1	[0 0 1]	1	2	[0.7 1 1 0.7]
	2	[0 1 0]	2	1.7	[0.79 1 1.09 0.7]
	3	[0 1 1]	3	2.58	[0.92 1 1.22 0.83]
	4	[1 0 0]	4	1.92	[1.54 1.63 1.22 0.83]
	5	[1 0 1]	5	3.99	[1.84 1.93 1.22 1.13]
	6	[1 1 0]	6	4.99	[[2.15 2.23 1.52 1.13]
	7	[1 1 1]	7	7.03	[2.14 2.22 1.51 1.12]
Mean squared error (ADALINE): 5.67					
2	1	[0 0 1]	1	3.26	[1.46 2.22 1.51 0.44]
	2	[0 1 0]	2	2.97	[1.17 2.22 1.22 0.44]

Practical

4. $\begin{bmatrix} -10.7879 \\ -9.7778 \\ -8.1616 \\ -7.9596 \end{bmatrix}$

6. (a) $\texttt{W_values} = \begin{bmatrix} 0 & 0 & 0 \\ 0.2 & -0.2 & -0.2 \\ 0.8 & -0.8 & 1 \\ 1.04 & -0.8 & 1 \\ 1.39 & -0.45 & 1 \\ 1.80 & -0.04 & 0.59 \\ 2.06 & 0.75 & 0.59 \end{bmatrix}$

(b)

```
error = 15.500000
Entering the while for the 1th. time, instance 1...
Entering the while for the 1th. time, instance 2...
Entering the while for the 1th. time, instance 3...
Entering the while for the 1th. time, instance 4...
```

```
Entering the while for the 1th. time, instance 5...
Entering the while for the 1th. time, instance 6...
error = 1.375781
```

10. (a) 23 times
    (b) 23691×9
    (c) 81940.199

11. (a) 12 times
    (b) 373×2
    (c) 19.92

12.