

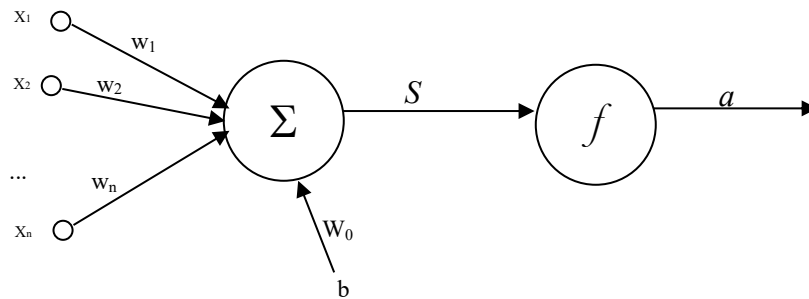
## Ficha de Trabalho nº 6

### Matlab: Redes Neurais (perceptron)

#### 1. Introdução

As Redes Neurais podem ser usadas para determinar relações e padrões entre entradas e saídas. A unidade mais simples de uma rede neuronal é o **perceptrão**. Nesta ficha será feita a implementação de um perceptrão para resolução de diferentes problemas.

A estrutura genérica de um perceptrão é a seguinte:



- Existe um conjunto de dados de entrada  $x_i$  (*inputs*), cada um deles com um peso  $w_i$  inicializado aleatoriamente;
- Existe um polo  $b$  (*bias*) com um peso  $w_0$  inicializado aleatoriamente;
- 

Através da aprendizagem iterativa dos pesos, o perceptrão é capaz de encontrar uma solução para dados linearmente separáveis (dados que possam ser separados por uma reta ou um hiperplano).

Em cada iteração, a saída  $S$  é calculada usando a seguinte equação:

$$S = \sum_{i=1}^n x_i w_i + w_0 b$$

A classificação  $a$  é decidida pela função de ativação escolhida: as funções mais usadas são a função *step*, a função linear, a função sigmoide, a função *tanh* ou a função *sign*. A função *step* é a função usada pelo perceptrão:

$$f(S) = \begin{cases} 1 & \text{se } S \geq 0 \\ 0 & \text{se } S < 0 \end{cases}$$

Na aprendizagem supervisionada, a classificação dada pela rede é comparada com o valor desejado, sendo o erro obtido (*delta*) usado para ajustar os pesos  $w_i$ . O pseudo-código seguinte resume este processo:

Dado um conjunto com k exemplos de treino (entradas e respectivas saídas)

Inicializar os pesos  $w_i$  de forma aleatória

Inicializar coeficiente de aprendizagem *coeff* e a entrada *b*

Repetir N iterações

Para cada exemplo de treino com input  $x_1, \dots, x_n$  e saída *t*

$$S = \sum_{i=1}^n x_i w_i + w_0 b$$

Calcular  $f(S)$

//usar *S* e a função de ativação

$\text{delta} = t - f(S)$

//cálculo do erro para o exemplo de treino *j*

Para *i* = 1 até numero de entradas

$w_i = w_i + \text{coeff} * x_i * \text{delta}$  //ajustar valor dos pesos

$w_0 = w_0 + \text{coeff} * b * \text{delta}$  //ajustar valor do peso do polo

## 2. Implementação de um perceptrão em Matlab

### 3.1 Descrição genérica do problema e do perceptrão a utilizar

Nesta secção será implementado e treinado um perceptrão para as operações lógicas OR, AND e NAND, cujas tabelas de verdade são as seguintes:

OR

$x_1$	$x_2$	<i>t</i>
0	0	0
0	1	1
1	0	1
1	1	1

AND

$x_1$	$x_2$	<i>t</i>
0	0	0
0	1	0
1	0	0
1	1	1

NAND

$x_1$	$x_2$	<i>t</i>
0	0	1
0	1	1
1	0	1
1	1	0

O código Matlab fornecido no Moodle está dividido em quatro secções partes principais:

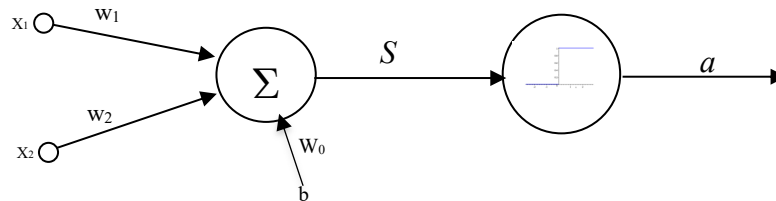
- 1) Inicialização da rede
- 2) Simulação do perceptrão aleatório
- 3) Treino da rede
- 4) Teste (simulação) do perceptrão treinado

O perceptrão a implementar tem as seguintes características:

- Arquitetura e Inicialização:
  - N° de entradas: duas ( $x_1$  e  $x_2$ ) com pesos ( $w_1$  e  $w_2$ )
  - Um polo ( $b$ ) com peso  $w_0$
  - Os pesos devem ser inicializados aleatoriamente usando a função *rand* do Matlab.
  - Coeficiente de aprendizagem *coeff*=1.0 e polo  $b$ =1 (estes valores não devem ser alterados)

- Simulação:

- O resultado (classificação) para um conjunto de inputs é obtido efectuando a soma pesada das entradas, a que se segue a aplicação da função de ativação:



- A função de ativação *step* é dada pela equação:  $f(S) = \begin{cases} 1 & \text{se } S \geq 0 \\ 0 & \text{se } S < 0 \end{cases}$

- Treino:

- Os pesos são alterados usando a regra delta:  $w_i = w_i + \text{coeff} * x_i * \text{delta}$ , onde *delta* é o erro dado pela diferença entre a saída desejada e a saída obtida.
- O peso do polo é alterado usando a expressão  $w_0 = w_0 + \text{coeff} * b * \text{delta}$

Em cada iteração da fase de treino deverá ser mostrada a evolução (pontos a classificar, fronteira de decisão) de forma similar à figura 2. A linha de decisão é dada pela função:

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{w_0}{w_2}$$

Para tal deve usar a função *plot* do Matlab

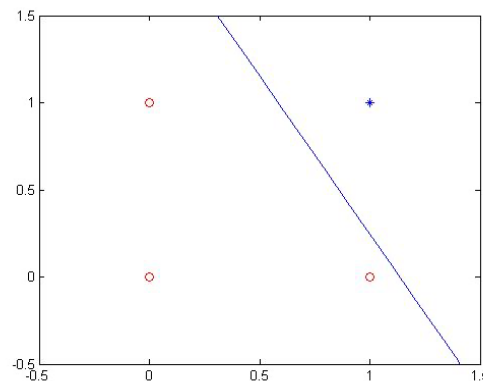


Figura 2 – Exemplo de possível visualização da evolução do perceptron

### 3.2 Tarefas a executar

- Grave para a sua área de trabalho o ficheiro **perceptrao.m** fornecido no Moodle. Use este ficheiro como base para a realização das atividades desta ficha de trabalho.
- Complete o código fornecido de acordo com a descrição dada na secção 3.1:
  - Inicialize os 3 pesos de forma aleatória (função **rand()**)
  - Complete o vector com as saídas para os casos ‘OR’ e ‘NAND’
  - Complete o código na fase de simulação do perceptron aleatório

- iv. Implemente o treino do perceptrão, usando o pseudo-código e os dados fornecidos (cálculo da saída, função de ativação, cálculo do erro e ajuste de pesos)
  - v. Complete o código na fase de teste
- c) Na linha de comando chame a função *perceptrão* para a função lógica AND
- ```
>> [w0, w1, w2, out_init, out_sim] = perceptrao('AND')
```
- i. O argumento da função indica qual a função lógica a simular ('AND', 'OR', 'NAND' ou 'XOR')
  - ii. A função devolve os pesos obtidos no final do treino, as saídas produzidas pelo perceptrão aleatório e as saídas produzidas pelo perceptrão treinado.

As classificações efetuadas pelo perceptrão devem ser comparadas com o vector *target* da função lógica que está a ser simulada. Se o perceptrão conseguir aprender a função, no final do treino deve reproduzir as classificações corretas para cada input.

- d) Teste a rede para esta função. Analise e comente os resultados.
- e) Repita as tarefas **c)** e **d)** para as funções OR e NAND.
- f) Altere o código para a incluir também a função XOR. A tabela de verdade do XOR é:

| x <sub>1</sub> | x <sub>2</sub> | t |
|----------------|----------------|---|
| 0              | 0              | 0 |
| 0              | 1              | 1 |
| 1              | 0              | 1 |
| 1              | 1              | 0 |

Treine a rede para este problema. A que conclusões chega? Como justifica os resultados obtidos?

- g) Altere o código para que seja funcional para qualquer número de entradas da variável **in**:
  - i. Os pesos *w* devem ser guardados num vetor  $w=[w_0 \ w_1 \ w_2 \ w_3 \ w_4 \ ... \ w_n]$ . O tamanho do vetor deve corresponder ao número de exemplos de entradas (linhas da matriz **in**) + 1 (para guardar o *w*<sub>0</sub>)
  - ii. As variáveis ***out\_init***, ***out*** e ***delta*** devem ser inicializadas com o número de zeros correspondente ao número de exemplos de treino (número de colunas da matriz de entradas).
  - iii. O ciclo de exemplos de treino deve ser generalizado de 4 para o número de exemplos de treino (número de colunas da matriz de entradas).
  - iv. O cálculo de *S* deve ser feito num ciclo **for** que percorra o vetor de pesos e multiplique cada um deles pela entrada correspondente. O peso *w*<sub>0</sub> deve ser tratado fora do ciclo.

### 3. Proposta de Trabalho

Crie e treine um perceptron em Matlab que implemente a “**regra da maioria**” para 5 entradas binárias. Neste problema, o perceptron deve produzir o resultado 1 sempre que a maioria das entradas tiver o valor 1.

Adapte o código feito em g) para este problema.

- Na matriz de entradas crie todas as combinações possíveis para 5 entradas
- No vetor de saída coloque 1 ou 0 consoante se “regra da maioria” se aplica ou não na respetiva entrada
- Comente o código do *plot*