Ficha de Trabalho nº 2
Matlab: Redes Neuronais (perceptrão)

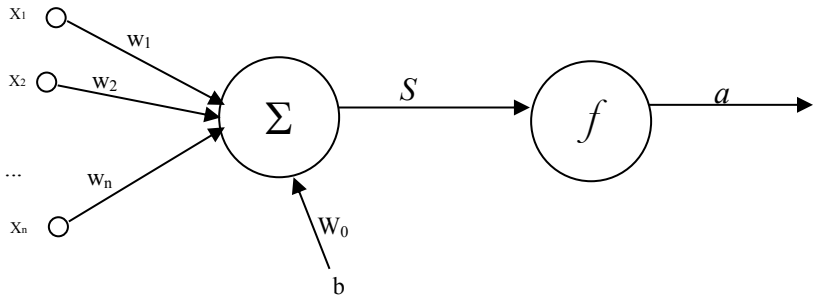
1. Bibliografia

Alunos que não estejam familiarizados com o Matlab devem ler o seguinte tutorial
https://www.mathworks.com/help/matlab/getting-started-with-matlab.html

Material de apoio disponível no Moodle.
Mathworks site: http://www.mathworks.com/help/nnet/ug/perceptron-neural-networks.html

2. Introdução

As Redes Neuronais podem ser usadas para determinar relações e padrões entre entradas e saídas. Uma rede neuronal monocamada *feedforward* que tenha a capacidade de aprender e diferenciar conjunto de dados é conhecida por **perceptrão**. Nesta ficha serão utilizadas as estruturas mais simples, constituídas apenas por um perceptrão.
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 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* pode ser usada para classificação binária, i.e., para escolher entre duas possíveis classes:

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

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 wi de forma aleatória
Inicializar coeficiente de aprendizagem coeff e polo b
Repetir N iterações
       Para cada exemplo de treino com input x1,...,xn e saída t
              S = Σ(i=1..n) xi*wi + w0*b
              Calcular f(S)                 //usar S e a função de ativação
              delta = t - f(S)              //cálculo do erro para o exemplo de treino j
              Para i = 1 até numero de entradas
                     wi = wi + coeff * xi * delta  //ajustar valor dos pesos
              w0 = w0 + coeff * b * delta       //ajustar valor do peso do polo
```

3. 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	t
0	0	0
0	1	1
1	0	1
1	1	1

AND

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

NAND

x_1	x_2	t
0	0	1
0	1	1
1	0	1
1	1	0

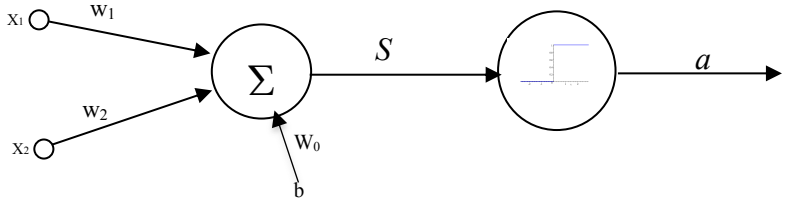
O código Matlab fornecido no Moodle está dividido em quatro secções partes principais:

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:
  - Nº de entradas: duas (***x₁*** e ***x₂***) com pesos (***w₁*** e ***w₂***)
  - Um polo (***b***) com peso ***w₀***
  - 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 efectuado a soma pesada das entradas, a que se segue a aplicação da função de ativação:



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

- Treino:
  - Os pesos são alterados usando a regra delta: $w_i = w_i + coeff * x_i * delta$, 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 $w_0 = w_0 + coeff * b * delta$

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

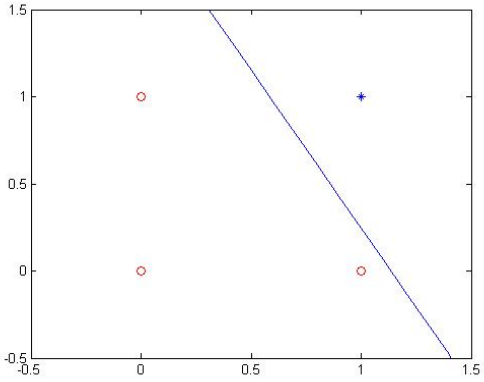


Figura 2 – Exemplo de possível visualização da evolução do perceptrão

3.2 Tarefas a executar

a) 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.

b) Complete o código fornecido de acordo com a descrição dada na secção 3.1:
   i. Inicialize os 3 pesos de forma aleatória (função `rand()`)
   ii. Complete o vector com as saídas para os casos 'OR' e 'NAND'
   iii. Complete o código na fase de simulação do perceptrão 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_1	x_2	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 <u>qualquer número de entradas</u> 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 w0)
   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 w0 deve ser tratado fora do ciclo.

4. Proposta de Trabalho

Crie e treine um perceptrão em Matlab que implemente a **"regra da maioria"** para 5 entradas binárias. Neste problema, o perceptrão 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*