Workshop
Introdução a Programação C

Aplicação em Aprendizagem Computacional e Robótica

1 2 9 0 FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE D COIMBRA

deer.uc
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES

Título: Workshop Introdução a Programação C

Subtítulo: Aplicação em Aprendizagem Computacional e Robótica

O presente documento foi desenvolvido no contexto do "**Workshop: Introdução a programação C**" e visa guiar os participantes na componente prática deste workshop. Nomeadamente, providenciando um contexto suficiente (mas não exaustivo) para a introdução de conceitos da linguagem de programação C. Estes conceitos da linguagem de programação C são enquadrados em exercícios relacionados com a aprendizagem computacional aplicado à robótica móvel, ou seja, os participantes terão de implementar os conceitos lecionados no workshop num problema de aprendizagem computacional num contexto de aplicação de robótica móvel.

Autores:

Tiago Barros

Jérôme Mendes

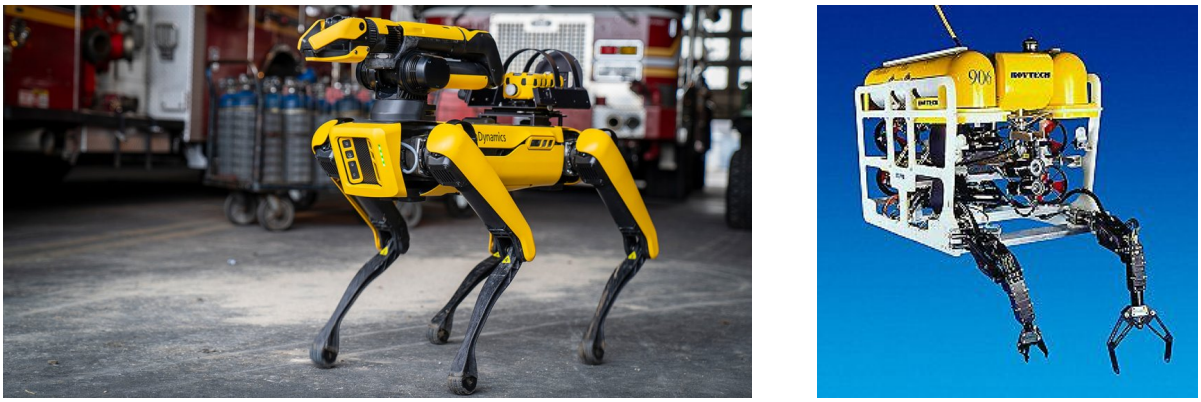
Luís Garrote

Data: janeiro, 2023

Conteúdo

1 Introdução

1.1 Formulação do Problema de Aplicação em Robótica Móvel



(a) (b)

Figura 1: Robôs desenvolvidos pela Boston Dynamics[1].

Robôs móveis possuem, entre outros, um sistema de locomoção que lhes permitem moverem no espaço físico [Kel13]. Robótica móvelcompreende uma categoria da família mais ampla da robótica sendo que, naquela categoria, incluem-se robôs terrestres (e.g., AGVs), robôs espaciais, robôs aéreos (e.g., drones), e robôs aquáticos (e.g., barcos autónomos, e submarinos). A Figura 1 ilustra alguns robôs móveis, onde o robô na Figura 1a é um "cão guarda"desenvolvido pela Boston Dynamics e na Figura 1b é um robô para fazer monitorização submarina.

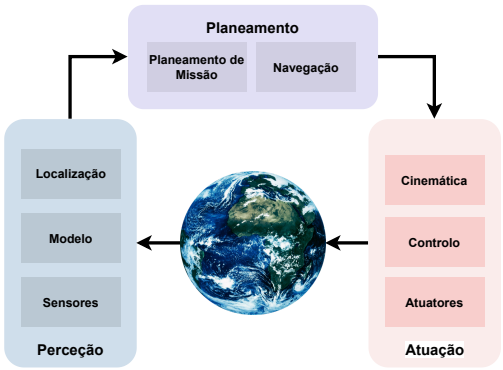


Figura 2: Representação simplificada do ciclo de execução de um robô.

A capacidade de se moverem permite, aos robôs móveis, interagirem com o seu meio-ambiente, nomeadamente com objetos, estruturas ou até com humanos que os rodeiam. Portanto, é necessário adotarem um comportamento (ação) por forma a poderem cumprir a tarefa para a qual foram programados e, simultaneamente, operar em segurança para não se coloquem a si ou/e terceiros em risco.

O comportamento de um robô é definido por um conjunto de módulos fundamentais que são transversais a todos os robôs móveis, independentemente do meio onde se movem. Estes módulos são responsáveis por: percecionar o mundo (e atualizar os modelos com novo conhecimento); estimar a posição e orientação (em inglês *pose*) do robô no mundo; e atuar, com base em objetivos, no mundo [DJ10]. Um sistema robótico móvel típico pode sersubdividido nos seguintes módulos fundamentais: perceção, planeamento e atuação. Estes módulos, ilustrados na Figura 2, são apenas uma abstração de sistemas mais complexos, cujo foco está fora do âmbito deste workshop.

Neste projeto focaremos apenas no **módulo de Perceção**, para o qual será desenvolvido uma metodologia simples para **reconhecimentos** de pontos/objetos de referência (em inglês *landmarks*). Estas *landmarks* são essenciais em algoritmos de localização e/ou mapeamento, tais como: *Simultaneous Localization and Mapping* (SLAM)[Gar+19], odometria visual ou odometria com base em "lasers" 3D [Bar+19]. Nomeadamente, num sistema de localização de odometria com base em "laser" 3D (ilustrados na Figura 3), são frequentemente usados objetos que sejam simultaneamente estáticos e facilmente distinguíveis de outros objetos. Em contexto

[1] https://www.bostondynamics.com

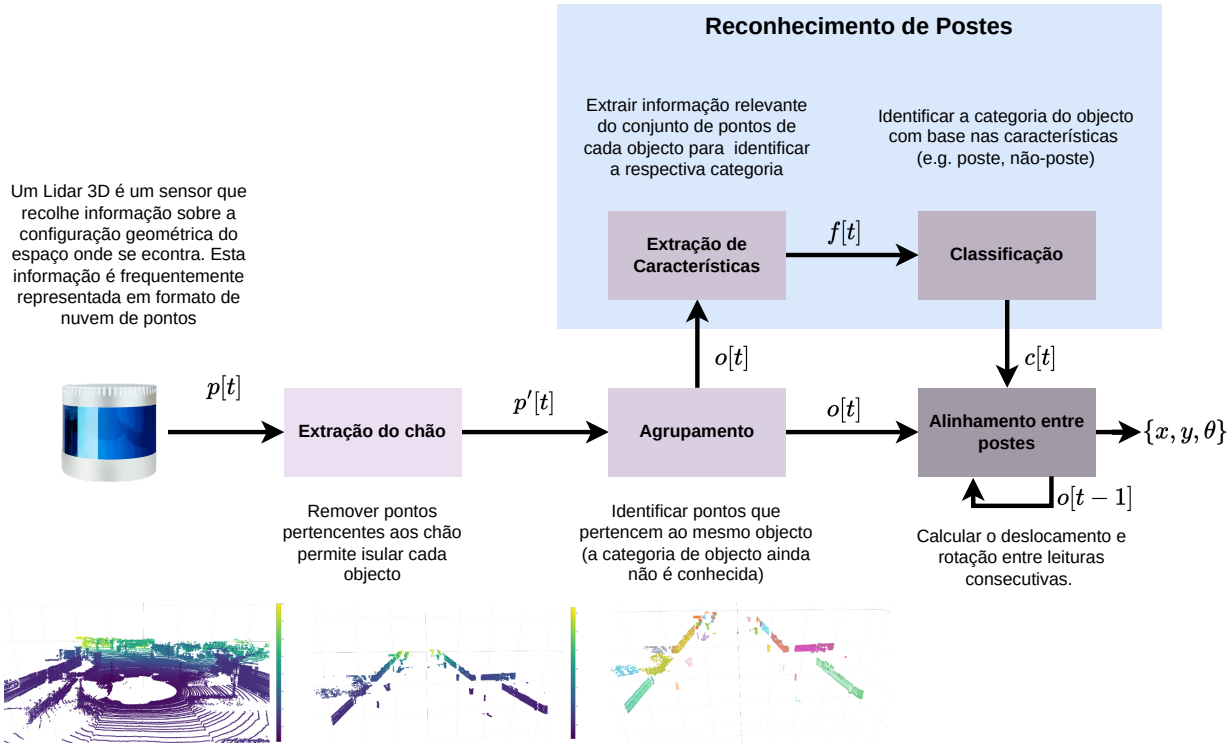


Figura 3: Sistema típico de localização com base em objetos de referência [Bar+19].

urbano, é muito usual recorrer a objetos tais como **postes, árvores ou outros objetos com formato cilíndrico** para esta tarefa.

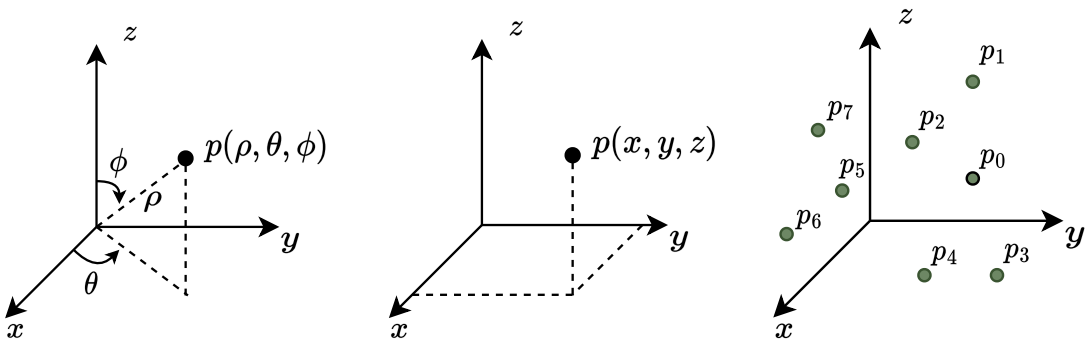
Neste sentido, o objetivo deste projeto é desenvolver um sistema de reconhecimento de postes usando uma rede neuronal. Para tal, os principais passos estão apresentados na Figura 3:

- ler/carregar a nuvem de pontos ($p[t]$).
- extrair os pontos pertencentes ao chão e devolver uma nuvem de pontos sem chão ($p'[t]$).
- Agrupar pontos em objetos (i.e., identificar pontos que potencialmente pertencem ao mesmo objeto), formando um conjunto de pontos por objeto ($o[t]$).
- extrair, de cada objeto, as características (*features*) relevantes ($f[t]$) que permitam identificar a sua categoria. Neste projeto as categorias são 'poste' ou 'não-poste'.
- as características de cada objeto são fornecidas ao classificador - que neste projeto é uma rede neuronal — para este estimar uma categoria ($c[t]$).
- calcular o deslocamento e rotação do robô. Os objetos identificados como postes são usados num método de alinhamento de pontos para obter a transformada rígida (i.e., deslocação e rotação) entre leituras de postes consecutivas (i.e., entre $o[t]$ e $o[t-1]$).

No âmbito deste projeto focaremos os exercícios nos seguintes módulos: extração de características e classificação. Os restantes módulos são fornecidos.

2 Nuvem de Pontos

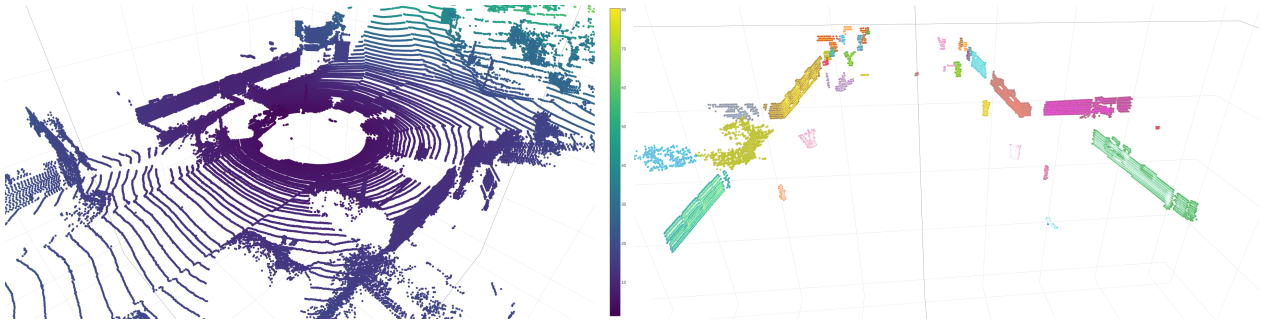
Neste trabalho usaremos como dados um conjunto de nuvens de pontos retornadas por LiDAR ("laser") 3D. Basicamente, sensores LiDAR capturam a configuração geométrica do espaço onde operam, retornando pontos num sistema de coordenadas esféricas onde cada ponto é dado por $p_s = \{\rho, \theta, \phi\}$ com a origem centrada no sensor[2]. Estes pontos são depois convertidos para um espaço em coordenadas Cartesianas, onde cada ponto é definido por $p = \{x, y, z\}$. Resumindo, uma nuvem de pontos é uma coleção de pontos $NP = \{p_1, p_2, ..., p_n\}$ com $p_i \in \mathbb{R}^3$. As Figuras 4a,b ilustram estes dois sistemas de coordenadas, enquanto a Fig. 4c ilustra uma nuvem de pontos com apenas 8 pontos.



(a) (b) (c)

Figura 4: Representação de: a) ponto num sistema de coordenadas esférico, b) o mesmo ponto num sistema de coordenadas cartesiano e c) uma nuvem de pontos com 8 pontos.

Uma nuvem de pontos de um sensor LiDAR 3D pode conter algumas centenas de milhares de pontos. A Figura 5a ilustra uma das nuvens de pontos usadas neste trabalho onde é possível observar a densidade e a ordem de grandeza da quantidade dos pontos existentes numa nuvem de pontos. Por outro lado, a Figura 5b ilustra a mesma nuvem de pontos mas, desta vez, sem pontos "do chão"e com os pontos agrupados em objetos: representando cada cor um objeto diferente. O agrupamento dos pontos (i.e., *clustering*) foi realizada através de um algoritmo denominado DBSCAN [Est+96].



(a) (b)

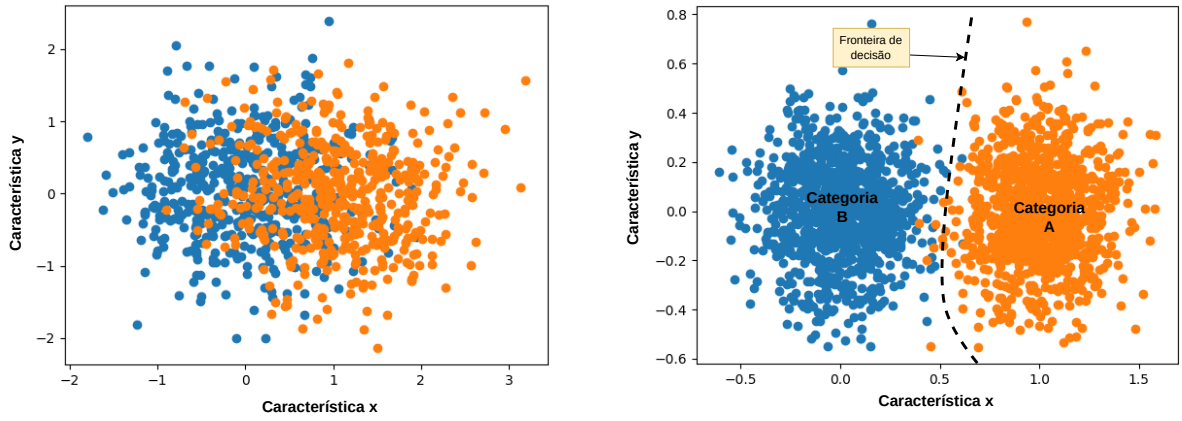
Figura 5: Nuvem de pontos: a) original do sensor, e b) sem chão e agrupada em objectos.

3 Extração de Características

Em problemas complexos e de larga escala, em aprendizagem computacional, a qualidade dos dados de treino (*training set*) de um modelo (e.g., rede neuronal) é tão ou mais importante do que a própria escolha

[2] VLP-16-User-Manual

do modelo em si[BB01]. Os dados para treinar o modelo (dados de treino) têm de ser representativos do problema para que o modelo consiga generalizar para novos exemplos (dados) que não foram usados durante o treino. Portanto, para treinar um modelo com sucesso é essencial usar dados que cumpram estes requisitos. Contudo, frequentemente os dados originais vindos de sensores podem não ter informação suficiente, ou podem conter informação redundante, ou aindapodem requerer demasiados recursos para treinar um modelo [Gér22].



(a) (b)

Figura 6: Sistema de coordenadas usados em nuvem de pontos: a) sistema esférico, b) sistema Cartesiano, e c) nuvem de pontos no sistema Cartesiano.

Assim, em vez de treinar um modelo nos dados originais, transformam-se estes dados para um espaço de características (*features*). A Figura 6 ilustra dois espaços de características distintos, contendo dados de duas categorias diferentes (pontos laranjas e azuis). Na Figura 6b consta, por exemplo, um espaço no qual as características são pouco adequadas pois as amostras de ambas as categorias sobrepõem-se, tornando a separação das duas categorias difícil. Já na Figura 6b é possível separar as duas categorias com uma fronteira de decisão, concluindo assim que as características escolhidas na Figura 6b são mais adequadas.

A escolha de características é, portanto, um processo essencial no desenvolvimento de um sistema baseado em aprendizagem computacional. É por este motivo que, frequentemente, as características são definidas por um/a especialista de uma dada tarefa ou problema a resolver pois é necessário ter conhecimento 'especializado' para extrair informação relevante de um conjunto de dados.

Neste trabalho, pretende-se "encontrar" conjuntos (grupos) de pontos que representem **postes** no mundo real. Para isso, os pontos de cada objeto são mapeados para um espaço de características onde o classificador terá mais facilidade de aprender uma fronteira de decisão entre o grupo de amostras pertencentes à categoria "poste" e "não-poste" (semelhante à Figura 6b). É necessário definir um conjunto de características que façam com que os objetos sejam separáveis (ou formem grupo) no espaço das características.

Em nuvem de pontos, os momentos estatísticos são características com capacidade descritiva dos dados. Neste sentido, serão usados neste trabalho a média e a variância aplicados a cada coordenada (i.e. x, y e z). Para simplificar, o cálculo da média e variância da coordenada x é dado por:

$$\text{Média}_x = m_x = \frac{1}{n}\sum_{i=1}^{n} x_i \leftarrow \bar{x} \tag{1}$$

$$\text{Variância}_x = v_x = \frac{1}{n}\sum_{i=1}^{n} (x_i - \overline{x})^2 \tag{2}$$

Assim, para cada conjunto de pontos (por objeto) obtêm-se as seguintes 6 características:

$$f_i = \{\overline{x}, \overline{y}, \overline{z}, v_x, v_y, v_z\}_i \tag{3}$$

onde f_i é o vetor de características do objeto i; m_x, m_y e m_y representam, respetivamente, a média dos pontos na coordenada x, y e z; e v_x, v_y e v_z representam a variância nas coordenadas x, y e z, respetivamente.

Para o conjunto de n objetos obtém-se uma matriz $M_{n\times 6}$ (ou um *array* 2D) com todas as características:

$$M = \begin{bmatrix} \overline{x}_1 & \overline{y}_1 & \overline{z}_1 & v_{x_1} & v_{y_1} & v_{z_1} \\ ... & ... & ... & ... & ... & ... \\ \overline{x}_n & \overline{y}_n & \overline{z}_n & v_{x_n} & v_{y_n} & v_{z_n} \end{bmatrix}_{[n\times 6]} \tag{4}$$

Este *array* 2D é particularmente importante quando for necessário carregar as características de um ficheiro.

4 Redes Neuronais Artificiais

4.1 Perceptrão

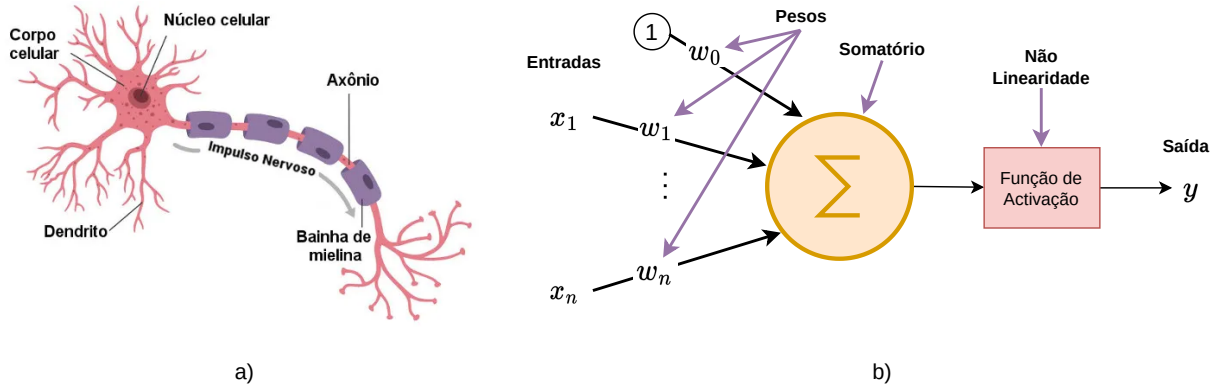


Figura 7: Ilustração de a) um neurónio e b) um perceptrão.

Redes neuronais artificiais (RNA) são modelos de aprendizagem computacional inspirados nos neurónios existentes num cérebro. Numa RNA, o equivalente ao neurónio é designado por **perceptrão**, cuja estrutura de processamento, inspirada no neurónio, está ilustrada na Fig.7 b).

A Figura 7 ilustra um perceptrão com um conjunto de entradas $\{x_1, x_2, ..., x_n\}$, pesos $\{b, w_1, ..., w_n\}$, uma "função de soma"e uma função de ativação. Os pesos são aplicados às entradas e o resultado passa por um "somatório"ie., soma ponderada. Matematicamente, esta operação pode ser interpretada por um produto escalar de vetores. Ao resultado é aplicado uma função de ativação que representa uma não-linearidade. Neste trabalho usaremos como função de ativação a função sigmoide que é dada pela seguinte equação:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

A saída do perceptrão é, assim, dada pela seguinte equação:

$$y = \sigma\big(b + \sum_{i=1}^{n} w_i x_i\big) \tag{6}$$

onde $\sigma(\cdot)$ é a função sigmoide (5) e b é o *bias*.

Esta equação também pode ser definida vetorialmente da seguinte forma:

$$y = \sigma\left(\begin{bmatrix} b & w_1 & w_2 & \dots & w_n \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}\right) \tag{7}$$

Numa RNA, um perceptrão é representado por um nó, podendo a RNA ter um único ou múltiplos nós. Uma RNA só com um nó é a rede mais elementar/simples possível, enquanto, que RNAs com vários nós podem originar arquiteturas complexas. Um elemento essencial na organização dos nós são as camadas (*layers*).

4.2 Redes Neuronais simples

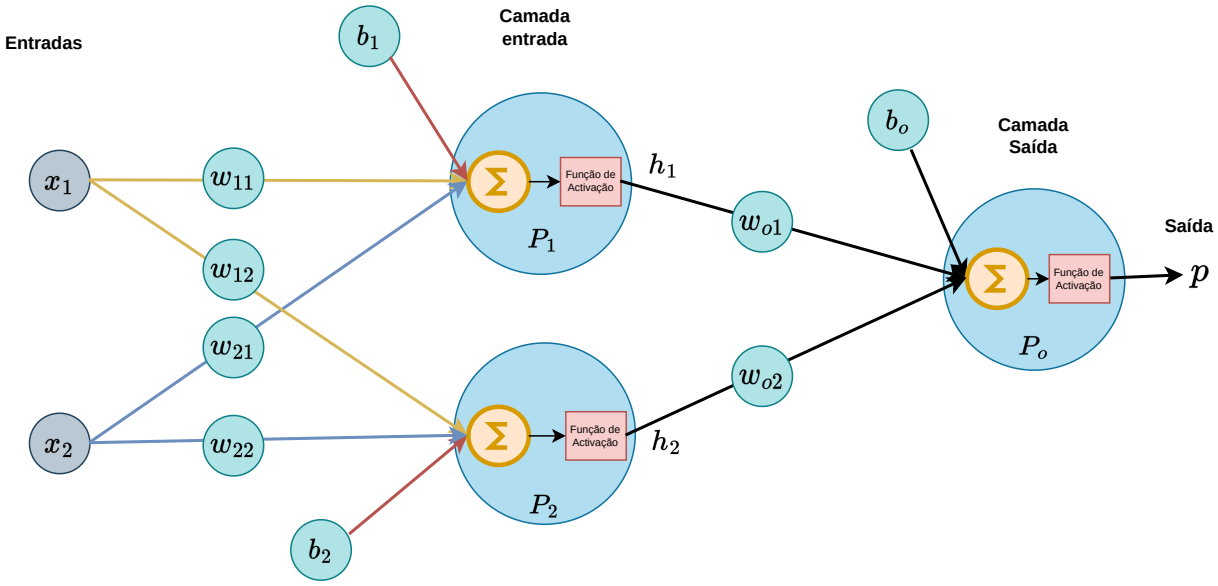


Figura 8: Rede Neuronal Artificial (RNA) com duas entradas, uma camada de entrada, e uma camada de saída. A rede tem duas entradas $\{x_1, x_2\}$, uma saída p e tem no total 3 nós $\{P_1, P_2, P_3\}$. Por razões didáticos, alguns termos nesta ilustração podem não corresponder à terminologia normalmente adotada na literatura. Por exemplo, os termos "entradas"e "camada de entrada"nesta figura são usualmente designadas por "camada de entrada"e "camada *hidden*", respectivamente.

As camadas permitem que a complexidade de uma RNA aumente em largura, se o número de nós por camada aumentar; e permitem que a complexidade aumente em profundidade se o número de camadas aumentar. A interligação entre duas camadas consecutivas forma-se ligando as saídas de uma camada aos nós (entradas) da camada seguinte, com o detalhe que todas as saídas são ligadas a todas as entradas (*full-connected*). A camada de saída, por norma, tem o mesmo número de nós que o número de categorias, excetuando no caso binário, onde um nó é suficiente. Assim, tendo uma previsão p (entre [0,1]) à saída da rede, a decisão c de pertencer à categoria "0"ou à categoria "1"é conseguida usando um limiar - normalmente usa-se 0.5 como limiar:

$$c(p) = \begin{cases} 1 & \text{se } p >= 0.5 \\ 0 & \text{se } p < 0.5 \end{cases} \tag{8}$$

O número de camadas numa RNA pode variar de um até n (várias) camadas encadeadas. A complexidade da arquitetura é definida pela complexidade da tarefa. É, também, objetivo durante a definição da arquitetura

procurar sempre a rede com menor complexidade (princípio *Occam'sRazor*). A Figura 8 ilustra uma RNA com duas camadas encadeadas: uma camada de entrada com dois nós e uma camada de saída com um nó. A rede tem duas entradas $\{x_1, x_2\}$, que estão ligadas aos dois nós da camada de entrada que, por sua vez, retornam duas saídas $\{h_1, h_2\}$ (uma para cada nó). Com base na equação (7), é possível generalizar para duas saídas, obtendo a seguinte equação:

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \sigma \left(\begin{bmatrix} b_1 & w_{11} & w_{21} \\ b_2 & w_{12} & w_{22} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \right) \tag{9}$$

A saída da rede p é obtida aplicando o mesmo procedimento que na camada anteriorusando os pesos da camada de saída $\{b_o, w_{o1}, w_{o2}\}$ e como entrada $\{h_1, h_2\}$ (saídas da camada anterior). Aplicando novamente a equação (7), obtém-se a seguinte equação para a saída da RNA:

$$p = \sigma \left(\begin{bmatrix} b_o & w_{o1} & w_{o2} \end{bmatrix} \begin{bmatrix} 1 \\ h_1 \\ h_2 \end{bmatrix} \right) \tag{10}$$

Os pesos de uma RNA são obtido durante a fase de treino, onde é usado uma função de custo que mede o erro entre previsão e a categoria real. O erro (da camada de saída), juntamente com um método de gradiente descendente, é usado para atualizar os pesos em cada nó de forma a minimizar a função de custo. O aprofundamento deste tema está fora do âmbito deste trabalho portanto, para mais detalhes, pede-se ao leitor para consular [Bis06; LBH15]. Neste trabalho, a rede já foi treinada e os pesos são dados.

Em resumo, uma RNA pode ter múltiplos nós. Estes, podem estar distribuídos por camadas. Quando existem mais do que uma camada, estas estão encadeadas, onde a saída de uma camada é a entrada da camada seguinte. As saídas das camadas são obtidas aplicando a equação (7) ou (6).

5 Avaliação de Desempenho

A avaliação de desempenho de um modelo é uma tarefa essencial no desenvolvimento de um sistema baseado em aprendizagem computacional. Pois é nesta fase (avaliação) que se apura se os objetivos estabelecidos são cumpridos. Para tal, são necessárias métricas que meçam quão bom (ex. numa escala de 0 a 1) as predições do modelo são em comparação com os valores verdadeiros/reais (*ground truth*, na designação inglesa).

5.1 Métricas e Matriz de Confusão

Na classificação existem enumeras métricas, cada uma medindo especificamente um aspeto do desempenho. As mais usadas para avaliar são: a exatidão (11), a precisão (12), a *recall* e a $\text{F}_{\text{1-score}}$ (respetivamente: *accuracy*, *precision*, *recall* e $\text{F}_{\text{1-score}}$, na designação inglesa). Uma forma complementar de avaliar o desempenho de um modelo passa por obter a matriz de confusão (*confusion matrix*).

A Figura 9 ilustra uma matriz de confusão típica, cujo objetivo é separar em campos as previsões do modelo que são: Verdadeiros Positivos (VP), Falsos Positivos (FP), Falsos Negativos (FN) e Verdadeiros Negativos (VN). Para construir a matriz de confusão é necessário ter as categorias reais.

$$\text{exatidão} = \frac{\#\{\text{Predições Correctas}\}}{\#\{\text{Total de Amostras}\}} = \frac{VP + VN}{VP + VN + FP + FN} \tag{11}$$

$$\text{precisão} = \frac{\#\{\text{Predições Positivas Correctas}\}}{\#\{\text{Predições Positivas}\}} = \frac{VP}{VP + FP} \tag{12}$$

$$\text{recall} = \frac{\#\{\text{Predições Positivas Correctas}\}}{\#\{\text{Positivos}\}} = \frac{VP}{VP + FN} \tag{13}$$

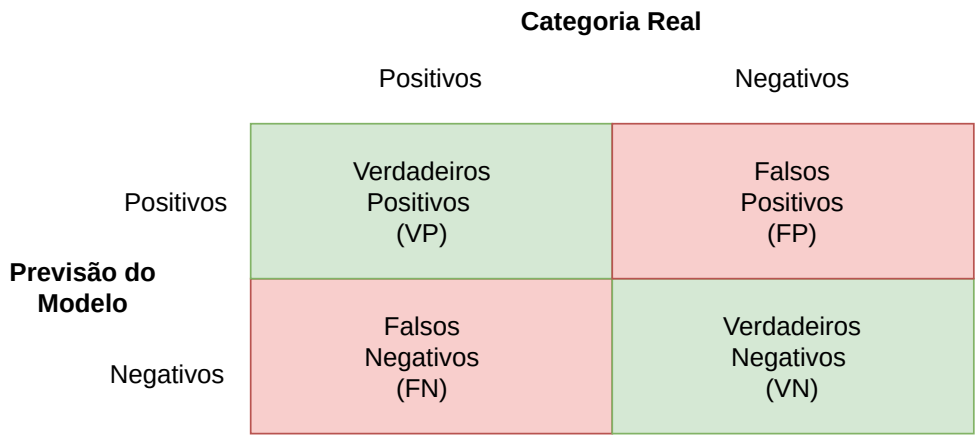


Figura 9: Estrutura de uma matriz de confusão. O rótulo real é representado nas colunas enquanto a previsão do modelo é representado nas linhas da matriz.

$$\mathrm{F}_{\text{1-score}} = 2 \times \frac{\text{precisão} \times \mathit{recall}}{\text{precisão} + \mathit{recall}} \tag{14}$$

No seguinte caso, considera-se que um dado classificador (e.g., uma RNA) devolveu as seguintes previsões para uma sequência de exemplos/dados $p = [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$, cujas categorias reais são os seguintes $y = [0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$. Usando a matriz de confusão obtém-se a seguinte matriz de confusão (Fig. 10):

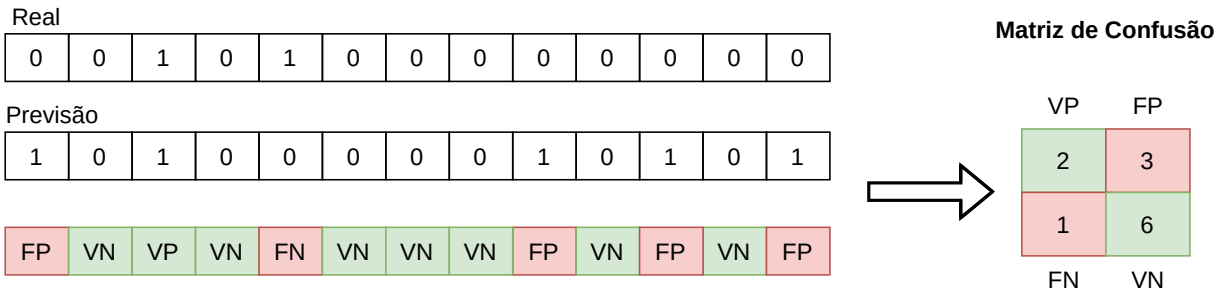


Figura 10: Exemplo da avaliação de desempenho utilizando uma matriz de confusão.

Tendo os valores de VP, VN, FP e FN consegue-se calcular as métricas. Usando as equações (11), (12), (13) e (14), obtêm-se os seguintes resultados:

$$\text{exatidão} = \frac{VP + VN}{VP + VN + FP + FN} = \frac{2 + 8}{2 + 8 + 2 + 1} = 0.77 \tag{15}$$

$$\text{precisão} = \frac{VP}{VP + FP} = \frac{2}{2 + 2} = 0.5 \tag{16}$$

$$\text{recall} = \frac{VP}{VP + FN} = \frac{2}{2 + 1} = 0.66 \tag{17}$$

$$\mathrm{F}_{\text{1-score}} = 2 \times \frac{\text{precisão} \times \mathit{recall}}{\text{precisão} + \mathit{recall}} = 2 \times \frac{0.5 \times 0.66}{0.5 + 0.66} = 0.56 \tag{18}$$

Dos resultados obtidos, é notório que usando só a exatidão não retrata o desempenho real do modelo. Se fosse apenas usado a exatidão como métrica de avaliação, poder-se-ia concluir que o modelo acerta 77% das vezes, representando um desempenho satisfatório em certas aplicações. Contudo, este modelo só tem uma precisão de 50%, ou seja, o modelo está 50% das vezes errado quando retorna a categoria 'positiva'.

6 Exercícios - M1

Nesta secção constam os exercícios por forma a aplicar, de uma maneiragradual, os conhecimentos adquiridos no workshop.

6.1 Construção de um Programa em C

Comece por construir o seu programa "básico"e compilar.

```c
// bibliotecas
#include <stdio.h>

//=================
// Funcoes


int main()
{
    //=================
    // Variaveis locais
    int ano = 2023;
    char nome[50] = "Robotica e ML";

    // ....
    nome = "Introducao ao Workshop de Programacao em C";

    printf("Workshop (%d): %s", ano, nome);

    return 0;
}
```

6.2 Características: Calcular a Média e a Variância

Cálculo das características. Veja a Secção 3.

Desenvolva código para calcular a média do seguinte vetor de valores reais, de 'coordenadas' $x = [0.0, 1.0, 2.0, 4.0]$. Use como base o código disponibilizado abaixo.

a) Comece por calcular a média de forma explícita para 4 pontos:

$$\overline{x} = \frac{1}{4} \times (x_1 + x_2 + x_3 + x_4) \tag{19}$$

b) Calcule a variância de forma explícita para 4 pontos:

$$v = \frac{1}{4} \times \left((x_1 - \overline{x})^2 + (x_2 - \overline{x})^2 + (x_3 - \overline{x})^2 + (x_4 - \overline{x})^2\right) \tag{20}$$

c) Imprima a média obtida em a) e a variância obtida em b)

```c
#include <stdio.h>

int main () {
    float x[4] = {0.0, 1.0, 2.0, 4.0};
    float m=0, v=0;
    int i=0;

    // ========================================
    // Ex6.a)

    //m = media
```

```
    // Ex6.b)
    // v =  variancia


    // imprima os valores de m e v

    // ==========================================
}
```

6.3 Avaliação de Desempenho

Desenvolva código que implemente e imprime no terminal (i.e., no ecrã) as métricas de avaliação (Equações (11), (13), (14) e (12)). Use como base o código que se segue , criando por exemplo um ficheiro "ex63.c". Para testar o código considere que uma dada rede teve como desempenho os seguintes resultados: VP=3, FP=2, VN=1 e FN=10.

```
#include <stdio.h>

int main () {
    int vp = 0, vn = 0, fn = 0, fp = 0;
    float exatidao=0.0;
    float recall=0.0;
    float precisao=0.0;
    float f1=0.0;

    //========================================
    // desenvolva o codigo aqui
    //
    // exatidao = ....
    // recall = ....
    // precisao = ....
    // f1 = ....
    //
    //========================================
    // Imprima aqui o resultados
    // .....
}
```

Resultado Esperado:

```
    Metricas:
        exatidao = 0.25
        recall   = 0.23
        precisao = 0.60
              f1 = 0.33
```

7 Exercícios - M2 e M3

7.1 Características

7.1.1 Calcular a Média e a Variância

Cálculo das características, similar à aos exercícios da Secção 6.2, recorrendo aos ciclos.

Usando ciclos, desenvolva código para calcular a média do seguinte vetor de valores reais, de coordenadas $x = [0.0, 1.0, 2.0, 4.0]$. Use como base o código disponibilizado abaixo, criando um ficheiro "ex711.c".

a) Calcule a média para n pontos (equação (1)) recorrendo a ciclos.

b) Calcule a variância para n pontos (equação (2)) recorrendo a ciclos.

c) Imprima a média obtida em a) e a variância obtida em b)

```
#include <stdio.h>

int main () {
    int n=4; //numero de pontos
    float x[4] = {0.0, 1.0, 2.0, 4.0};
```

```
    float m=0.0, v=0.0;
    // ========================================
    // Ex7.1.1a)
    //m = media

    // Ex7.1.1b)
    // v =  variancia

    // Ex7.1.1c)
    // imprima os valores de m e v

    // ========================================
}
```

7.1.2 Obter um vetor de características

a) Usando ciclos, calcule a média e a variância das coordenadas x, y e z por forma a obter o vetor de características representado na equação (2) de um dado objeto. Guarde os resultados num vetor de 6 elementos (vetor c) e imprima os seus valores como consta do *ResultadoEsperado*.

```
int main(){
    // pontos da coordenada x de um dado objeto
    float x[] = {48.831295, 48.567097, 48.642704, 48.422306, 48.573914, 48.445518, 48.510799,
        48.541405, 48.596207, 48.800819, 48.427418, 48.356022, 48.464630, 48.496235, 48.743748,
        48.601154, 48.588757, 48.559364, 48.612564, 48.541168, 48.553772, 48.566158, 48.272411,
        48.259216, 48.387821, 48.445427, 48.806393, 48.357189, 48.625401, 48.742447, 48.345242};

    // pontos da coordenada y de um dado objeto
    float y[] = {9.098334, 9.128318, 9.300251, 9.416203, 9.604131, 9.737077, 8.832438, 8.995374,
        9.084340, 9.281265, 9.368225, 9.512168, 9.692100, 9.857035, 9.295262, 9.505177, 9.662117,
        9.815056, 9.338249, 9.483191, 9.644128, 9.487194, 9.323258, 9.399228, 9.582159, 9.752093,
        9.434226, 9.426223, 9.796082, 9.394245, 9.397239};

    // pontos da coordenada z de um dado objeto
    float z[] = {1.888154, 1.879147, 1.883116, 1.876094, 1.882060, 1.879035, 1.562202, 1.564171,
        1.566155, 1.572120, 1.563102, 1.561076, 1.565043, 1.567014, 1.337117, 1.335078, 1.335049,
        1.335021, 0.959109, 0.958082, 0.959053, 0.705081, 0.360110, 0.360096, 0.360063, 0.361032,
        0.127092, 0.128092, 0.128025, -0.187901, -0.184903};

    int n = 31; // Numero de pontos de um dado objeto
    float c[] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0}; // c={mx,my,mz,vx,vy,vz} (vetor c)
    float mx=0.0, my=0.0, mz=0.0, vx=0.0, vy=0.0, vz=0.0;

    // ==============================
    // codigo aqui ...
    for(..;...;..){
            mx = // media do x
            my = // media do y
            mz = // media do z
    }
    // ...
    for(..;...;..){
            vx = // variancia do x
            vy = // variancia do y
            vz = // variancia do z
    }
    // ...
    // ==============================

    // Garde os valores no vector "c"
    // ...

    // Imprimir no terminal
    // ...
}
```

Resultado Esperado:

```
    mx = 48.538208, vx =  0.021723
    my = 9.440077, vy =  0.059699
    mz = 1.102703, vz =  0.456924
```

b) Usando o código desenvolvido na alínea anterior (7.1.2.a), desenvolva a função *features* usando o prototipo abaixo definido. A função *features* recebe os valores das coordenadas x, y, z de um dado objeto, obtém as características e guarda o seus valores no vetor c.

```
int features(float x[], float y[], float z[], unsigned int n, float c[]){
    // --------------------------
    // Corpo da funcao
    //--------------------------
    return(0);
}
int main(){
    // declaracao das variaveis da  definidas em a)

    features(x,y,z,n,c);

   // codigo para imprimir definido em a)
}
```

7.2 Avaliação de Desempenho

Esta secção trata de exercícios para obter as métricas de avaliação de desempenho (ver Secção 5 para mais detalhes).

Na resolução dos exercícios desta secção, use o código que se segue como base. Crie um ficheiro "ex72.c" com o código e resolva as várias alíneas nos sítios indicados.

```
#include <stdio.h>

float exatidao(...){
// ================================
// Faca o codigo aqui - codigo alinea 7.2.2a)
// ================================
}
float recall(...){
// ================================
    // Faca o codigo aqui - codigo alinea 7.2.2a)
// ================================
}
float precisao(...){
// ================================
    // Faca o codigo aqui - codigo alinea 7.2.2a)
// ================================
}
float f1(...){
// ================================
    // Faca o codigo aqui - codigo alinea 7.2.2a)
// ================================
}
void printEvalMetric(...){
    // ================================
    //  Faca o codigo aqui - codigo alinea 7.2.2 b)
    //================================
}
void printMatrizConfusao(...)
{
    // ================================
    //  Faca o codigo aqui - codigo alinea 7.2.1 b)
    //================================
}

int main () {
    int previsao[] = {0, 1, 0, 0, 1}; //previsao
    int y[] = {1, 0, 1, 0, 1}; // categorias reais
    int n = 5; //numero de previsoes
    int vp=0, vn=0, fn=0, fp=0;
    float e=0.0, r=0.0, p=0.0, f=0.0; // e-exatidao; r-recall ; p-precisao; f-f1;

    //=========================================
    //
    // Faca o codigo aqui - codigo alinea 7.2.1a)
    //
    // =========================================
    //
    // Faca o codigo aqui - codigo alinea 7.2.1b)
    //
```

```
    // printMatrizConfusao(...);
    //========================================
    //
    // Faca o codigo aqui - codigo alinea 7.2.2)
    // e = exatidao(...);
    // r = recall(...);
    // p = precisao(...);
    // f = f1(...);

    //========================================
    // Faca o codigo aqui - codigo alinea 7.2.2b)
    // printEvalMetric(...);

}
```

7.2.1 Obter a Matriz de Confusão

a) Desenvolva código para calcular os verdadeiros positivos (VP), os falso positivos (FP), os verdadeiros negativos (VN) e os falsos negativos (FN). Para testar o código considere os seguintes dados: previsão $p = [0, 1, 0, 0, 1]$ e categorias reais $y = [1, 0, 1, 0, 1]$.

Resultado esperado:

```
    vp = 1
    vn = 1
    fn = 2
    fp = 1
```

b) Desenvolva código para imprimir no terminal os valores obtidos na alínea anterior em formato de matriz de confusão como ilustrado na Figura 9. Use o código desenvolvido na alínea anterior e construa a função com base no seguinte protótipo:

```
void printMatrizConfusao(int vp, int vn, int fp, int fn);
```

Resultado esperado:

```
      VP  |  FP
    -------------
    | 1   |  1  |
    -------------
    | 2   |  1  |
    -------------
      FN  |  VN
```

7.2.2 Métricas de avaliação

a) Desenvolva código para calcular as métricas de avaliação dadas pelas equações (11), (12), (13) e (14). Desenvolva uma função para cada uma das métricas. Estas funções recebem como argumentos de entrada os VP, VN, FP e FN (use os resultados obtidos na alínea anterior), e devolvem o valor calculado com a respetiva métrica. Note que as métricas de avaliação (Equações (11), (13), (14) e (12)) foram já implementadas no Exercício 6.3 (Ficheiro "ex63.c").

```
float exatidao(int vp, int vn, int fp, int fn);
float recall(int vp, int vn, int fp, int fn);
float precisao(int vp, int vn, int fp, int fn);
float f1(int vp,int vn,int fp,int fn);
```

Resultado esperado:

```
    exatidao -> 0.40; recall -> 0.50; precisao -> 0.33; f1 -> 0.40
```

b) Desenvolva código para imprimir no terminal os valores das métricas calculados. Desenvolva uma função que recebe como parâmetros de entrada os valores das quatro métricas e imprima no terminal os valores como ilustrado no *Resultadoesperado*.

```
void printEvalMetric(float exatidao, float precisao, float recall, float f1);
```

Resultado esperado:

```
    exatidao = 0.40
    recall   = 0.50
    precisao = 0.33
          f1 = 0.40
```

7.3 Redes Neuronais Artificiais

Neste exercício, crie um ficheiro "ex73.c". Resolva todas as alínea dentro deste ficheiro.

7.3.1 Produto escalar de vetores

a) Desenvolva código que implemente o produto escalar entre dois vetores $w \cdot f$. Para testar o código, use os seguintes vetores:
$w = [2.981465, -0.469540, 0.282919, -3.160719, 0.431559, -0.181928, -0.583663]$
$f = [1, 48.538208, 9.440077, 1.102703, 0.021723, 0.059699, 0.456924]^T$.
Desenvolva ainda código para imprimir o resultado no terminal.

Resultado esperado:

```
        Produto Escalar: -20.891898
```

7.3.2 Sigmoid

Desenvolva código em forma de uma função que implemente uma *sigmoid* representada pela equação (5). Para testar o código use o valor obtido na alínea anterior e mostre o resultado no terminal. No código use o seguinte protótipo:

```
double sigmoid(double X);
```

Nota: use a função exponencial existente na biblioteca <math.h>. Para isso, tem que a incluir. Se eventualmente obter erro na compilação terá que compilar o código incluindo a "-lm":

```
gcc -o ex73 ex73.c -lm
```

Resultado esperado:

```
    Sigmoid: 0.000000 //sim, o resultado e' approx zero.
```

Dica: teste a vossa função *sigmoid* com X = -1.230. O resultado deverá ser = 0.226181.

7.3.3 Perceptrão

Desenvolva código em forma de função para implementar um perceptrão - veja a equação (6). Pode usar o código desenvolvido nos exercícios anteriores (p.ex, produto escalar e sigmoid) e combiná-los numa função cujo protótipo segue abaixo. Para testar o código use os seguintes valores dos pesos (w) e entradas (x):

```
double pesos[] = {1.342612, -0.158436, 0.073571, -0.221724, -6.347154, -10.661768, 19.218855};
double entrada[] = {48.538208, 9.440077, 1.102703, 0.021723, 0.059699, 0.456924};
```

```
double perceptron(double * entrada, double * pesos, int n);
```

Nota: A entrada não tem o mesmo número de elementos do que os pesos (ver equação (7)).

Resultado esperado:

```
    Perceptron: 0.891834
```

7.3.4 Camada de nós

Desenvolva código que implemente uma camada com 2 nós (perceptrões). Reveja a equação (9). Implemente uma função com o seguinte protótipo:

```
void camada2p(double * entrada, double * pesos_p1, double * pesos_p2, int n_entradas, double * h);
```

Para testar o código use os seguintes valores:

```
double peso_p1[] = {2.981465, -0.469540, 0.282919, -3.160719, 0.431559, -0.181928,  -0.583663};
double peso_p2[] = {1.342612, -0.158436, 0.073571, -0.221724, -6.347154, -10.661768, 19.218855};
double entrada[] = {48.538208, 9.440077, 1.102703, 0.021723, 0.059699, 0.456924};
```

Resultado esperado:

```
    camada h1: 0.000000 h2: 0.891834
```

7.3.5 Rede Neuronal Artificial

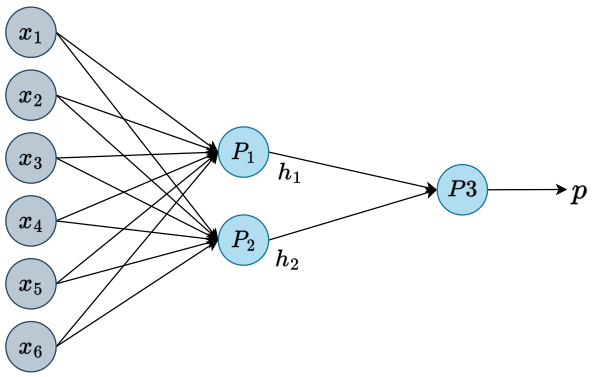


Figura 11: Diagrama de uma RNA com 6 entradas $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ e uma saída p. A rede tem duas camadas. A camada de entrada tem dois nós $\{p_1, p_2\}$ e a camada de saída tem um nó $\{p_3\}$.

Desenvolva código que implemente a rede neuronal da Figura 11. Esta rede tem 6 entradas $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ e uma saída p. A rede tem duas camadas. A camada de entrada tem dois nós $\{p_1, p_2\}$ e a camada de saída tem um nó $\{p_3\}$. Use o código anteriormente desenvolvido numa função cujo protótipo se encontra abaixo para construir esta rede.

```
double rede(double * entrada, double * pesos_p1, double * pesos_p2, double * poses_po, int
     n_entradas){

    double p;
    // neuronios da primeira camada
    // h1 = ....
    // h2 = .....

    // neuronios de daida
    // p = ...

    return(p);
}
```

Considere os seguintes pesos para a sua rede:

```
// no p1
double peso_p1[] = {2.981465, -0.469540, 0.282919, -3.160719, 0.431559, -0.181928,  -0.583663};
// no p2
double peso_p2[] = {1.342612, -0.158436, 0.073571, -0.221724, -6.347154, -10.661768, 19.218855 };
// no p3
double wo[] = {-7.215086, -9.565978, 20.784457};
```

Teste a rede nos vetores de características. v1 e v2. O vetor v1 foi extraído de um objeto tipo poste (categoria 1), enquanto v2 foi extraído de um objeto do tipo "não-poste"(categoria 0):

```
double v1[] = {48.538208, 9.440077, 1.102703, 0.021723, 0.059699, 0.456924};
double v2[] = {57.936264, 12.603884, 1.997085, 0.051343, 0.149340, 0.046218};
```

Resultado esperado:

```
    Entrada 1: predicao =  0.999988
    Entrada 2: predicao =  0.000738
```

8 Pipeline

Nesta secção, o objetivo é agregar todo o código anteriormente desenvolvido num único programa. O código começa por carregar características pertencentes a objetos que estão guardadas num ficheiro. As características são depois usadas como entrada no classificador, para que este devolva uma previsão de categoria entre $[0, 1]$ para cada objeto. Para obter a previsão (*prediction*), tem que usar a equação (8). Por fim, é calculado o desempenho da rede.

a) Desenvolva código para ler o ficheiro com os valores das características. Os objetos estão organizados por linha i.e., cada linha, com 7 elementos, representa um objeto. Os elementos de cada linha estão separados por espaços, sendo que o primeiro elemento representa a categoria real (c) e os restantes as características (mx, my, mz, vx, vy, vz).

```
    .....
    c mx my mz vx vy vz
    .....
```

Desenvolva uma função, cujo cabeçalho segue abaixo, que recebe como argumentos de entrada o nome do ficheiro a ler, o número de características a ler (por linha) e o número MÁXIMO de objetos a ler (i.e. o número MÁXIMO de linhas a ler). As características são devolvidas por referência, tendo cada característica um vetor com o tamanho igual ao número máximo de objetos a ler. A função tem que devolver via *return* o número de objetos que leu na realidade. Teste o código lendo as características presentes no seguinte ficheiro "features000011r.txt".

```
int  loadFeature(char * nome,int n_features,in * mx,float * my,float * mz,float * vx,float * vy,
    float * vz, int * targets, int num_objectos);
```

b) O código apresentado abaixo implementa o processo completo: desde a leitura das características até à avaliação do modelo. Use-o como base, e complete com o código em falta, nomeadamente com as funções que desenvolveu anteriormente. São incluídas 2 novas funções: eval e vizClusters. A função eval chama todos os métodos implementados ao longo do workshop para a avaliação de desempenho e tem como entradas a previsão dada pela rede, os *labels* e o número de *samples* (objetos) lido. A função vizClusters cria um ficheiro html onde representa os centroides dos *clusters* assim como o resultado da classificação num visualizador 3D. A função tem como entradas o nome do ficheiro a gravar, o número de *samples*, os centroides dos clusters (*arrays* mx, my e mz), o *array* com os *labels* e o *array* com as predições.

```
//========= bibliotecas ================
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>

//========= funcoes ================
double sigmoid(double value); // exer 7.3.2
double perceptron(double * entrada, double * pesos,int n_entradas); // exer 7.3.3
double rede(double * entrada, double * pesos_p1, double * pesos_p2, double * poses_po, int
    n_entradas); // exer 7.3.5
int  loadFeature(char * nome,int n_features,float * mx,float * my,float * mz,float * vx,float * vy,
    float * vz, int * targets, int num_objectos);
float exatidao(int vp, int vn, int fp, int fn);// exer 7.2.2
float recall(int vp, int vn, int fp, int fn); // exer 7.2.2
float precisao(int vp, int vn, int fp, int fn); // exer 7.2.2
float f1(int vp, int vn, int fp, int fn); // exer 7.2.2
void printEvalMetric(float exatidao, float precisao, float recall, float f1); // exer 7.2.2
void printMatrizConfusao(int vp, int vn, int fp, int fn); // exer 7.2.1
void eval(double * previsao, int * target, int n); // usando exer 7.2.1.a, prepreenchida apos main
void vizClusters(char *ficheiro, int n_samples, double *x, double *y, double *z, int *labels, double
     *y_pred);

int main(){
    //========= variaveis ================
    int N_FEATURES = 6;
    int N_CLASSES = 1;
    int max_samples = 40; // numero de objetos
    double mx[max_samples],my[max_samples],mz[max_samples],vx[max_samples],vy[max_samples],vz[
        max_samples];

    int Targets[max_samples];


    double pesos_p1[] = {2.981465,-0.469540,0.282919,-3.160719,0.431559,-0.181928,-0.583663};
    double pesos_p2[] = {1.342612,-0.158436,0.073571,-0.221724,-6.347154,-10.661768,19.218855};
    double poses_po[] = {-7.215086,-9.565978,20.784457};

    //========= codigo principal ================
    printf("Loading Features ...\n");
    int n_samples = loadFeature("features000011r.txt",N_FEATURES, mx, my, mz, vx, vy, vz, Targets,
        max_samples);

    // Classificacao
    double previsao[n_samples];
    for(int i =0; i<n_samples;i++){
        double Inputs[6]={mx[i], my[i], mz[i], vx[i], vy[i], vz[i]};
        previsao[i] = rede(Inputs, pesos_p1, pesos_p2, poses_po, N_FEATURES);
        printf("p[i] = %lf y[]=%d\n", previsao[i],Targets[i]);
    }
    // Avaliacao
    eval(previsao,Targets,n_samples);

    //visualizacao
    vizClusters("ClusterClassificationViz.html",n_samples,mx,my,mz,Targets,previsao);

}

//========= funcao eval  ================
void eval(double * previsao, int * target,int n){
    int vp=0, vn=0, fn=0, fp=0;
```

```
    int rotulo =0;
    for(int i=0;i<n;i++){
        // calcule os valore vp, fp, fn vn
        / ...
    }
    printMatrizConfusao(vp,vn,fp,fn);
    float e = exatidao(vp,vn,fp,fn);
    float r = recall(vp,vn,fp,fn);
    float p = precisao(vp,vn,fp,fn);
    float f = f1(vp,vn,fp,fn);
    printEvalMetric(e,p,r,f);
}

//========= funcao vizClusters  ================
void vizClusters(char *ficheiro, int n_samples, double *x, double *y, double *z, int *labels, double
     *y_pred){

    FILE *fp  = fopen (ficheiro, "w");
    if(fp==NULL){
        printf("Nao e' possivel criar o ficheiro\n");
        return;
    }
    int colorR[2],colorG[2],colorB[2];
    colorR[0]=0,colorG[0]=0,colorB[0]=255,colorR[1]=0,colorG[1]=255,colorB[1]=0;

    fprintf(fp,"<head>\n");
    fprintf(fp,"    <script src='https://cdn.plot.ly/plotly-2.16.1.min.js'></script>\n");
    fprintf(fp,"    <script src='https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.17/d3.min.js'></
        script>\n");
    fprintf(fp,"</head><body>\n<div id='myDiv' style='width: 99%%;height:99%%'></div>\n");
    fprintf(fp,"<script>\n");
    for(int j=0;j<n_samples;j++){
        fprintf(fp,"var Cluster%d = { x:[%lf], y:[%lf], z:[%lf],",j,x[j],y[j],z[j]);
        fprintf(fp,"   mode: 'markers',\n");
        fprintf(fp,"    marker: { size: 5,\n");
        int y_p=y_pred[j]>0.5?1:0;
        fprintf(fp,"        color: 'rgba(%d, %d, %d, 0.8)',\n",colorR[y_p],colorG[y_p],colorB[y_p]);
        if(y_p!=labels[j]){
            fprintf(fp,"symbol: 'x',\n");
        }
        fprintf(fp,"        line: { width: 1.5}, opacity: 0.8}, type: 'scatter3d' };\n");
    }
    fprintf(fp,"    var scaleUP = { x: [80], y: [80], z: [80], mode: 'markers',\n");
    fprintf(fp,"        marker: { color: \"rgba(255, 255, 255, 0)\"}, type: 'scatter3d' };\n");
    fprintf(fp,"    var scaleDOWN = { x: [-80], y: [-80], z: [-80], mode: 'markers',\n");
    fprintf(fp,"        marker: {color: \"rgba(255, 255, 255, 0)\"}, type: 'scatter3d' };\n");

    fprintf(fp,"var data = [");
    for(int j=0;j<n_samples;j++){
        fprintf(fp,"Cluster%d ,",j);
    }
    fprintf(fp,"scaleUP,scaleDOWN];\n");
    fprintf(fp,"var layout = {showlegend:false,showgrid: false,hovermode : 'x',margin: {  l: 0, r:
        0, b: 0, t: 0\n");
    fprintf(fp,"  }, xaxis: {autorange: false,scaleratio: 1,range: [-80, 80]}, yaxis: {autorange:
        false,scaleratio: 1,range: [-80, 80]}, zaxis: {autorange: false,scaleratio: 1,range: [-80,
        80]}};\n");
    fprintf(fp,"Plotly.newPlot('myDiv', data, layout);\n</script>\n</body>\n");

    fclose(fp);
}
```

Resultado esperado:

```
      VP | FP
-----------
|  9 |  0 |
-----------
|  2 | 29 |
-----------
      FN | VN

    Metricas:
    exatidao = 0.95
    recall   = 1.00
    precisao = 0.82
    f1 = 0.90
```

9 Exercícios Extra

As funções de ativação são essenciais nas redes neuronais pois introduzem não linearidades essenciais para modelar problemas. Estas não linearidades permitem que os modelos consigam gerar representações mais complexas do que quando se utilizam ativações lineares. Algumas das funções mais utilizadas incluem o Rectified Linear Unit (ReLU), Leaky ReLU, Exponential Linear Unit (ELU), tangente hiperbólica (tanh) e Gaussian Error Linear Units (GELU). Implemente uma função para cada uma das funções de ativação seguintes e teste substituindo a função sigmoide.

9.1 ReLU

$$RELU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{21}$$

9.2 Leaky ReLU

$$LeakyReLU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \tag{22}$$

Considere $\alpha = 0.01$.

9.3 ELU

$$ELU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(exp(x) - 1) & \text{if } x \leq 0 \end{cases} \tag{23}$$

Considere $\alpha = 1$.

9.4 tanh

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{24}$$

9.5 GELU

$$GELU(x) = 0.5x(1 + tanh(\sqrt{}(2/\pi)(x + 0.044715x^3))) \tag{25}$$

Referências

[Est+96] Martin Ester et al. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". Em: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[BB01] Michele Banko e Eric Brill. "Scaling to very very large corpora for natural language disambiguation". Em: *Proceedings of the 39th annual meeting of the Association for Computational Linguistics*. 2001, pp. 26–33.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. New York, NY: Springer, 2006.

[DJ10] Gregory Dudek e Michael Jenkin. *Computational principles of mobile robotics*. Cambridge university press, 2010.

[Kel13] Alonzo Kelly. *Mobile robotics: mathematics, models, and methods*. Cambridge University Press, 2013.

[LBH15] Yann LeCun, Yoshua Bengio e Geoffrey Hinton. "Deep learning". Em: *nature* 521.7553 (2015), pp. 436–444.

[Bar+19] Tiago Barros et al. "Improving Localization by Learning Pole-Like Landmarks Using a Semi-supervised Approach". Em: *Iberian Robotics conference*. Springer. 2019, pp. 255–266.

[Gar+19] Luis Garrote et al. "Mobile robot localization with reinforcement learning map update decision aided by an absolute indoor positioning system". Em: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 1620–1626.

[Gér22] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* "O'Reilly Media, Inc.", 2022.