LISTA 1 - EEC1505: REDES NEURAIS

Aluno: Pedro Victor Andrade Alves
Matrícula: 20211022916

1º)

Considere o problema de estimativa da matriz de covariância de uma dada distribuição de vetores aleatórios dados por $\mathbf{x_1} = \begin{bmatrix} 7 \\ 3 \\ 9 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 4 \\ 6 \\ 11 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 4 \\ 2 \\ 5 \end{bmatrix}, \mathbf{x}_4 = \begin{bmatrix} 5 \\ 5 \\ 7 \end{bmatrix}$. Estime o vetor média (média amostral) e a matriz de covariância amostral $\mathbf{S} = \{s_{ij}\}$ assumindo que a matriz é simétrica. Observando que a média amostral é dada por $\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$ e a variância amostral é dada por $s_{ij} = \frac{\sum_{l=1}^{N}(x_{il} - \hat{\mu}_i)(x_{jl} - \hat{\mu}_j)}{N-1}$. Calcule também a partir da matriz de covariância obtida:

Resposta:

$$S = \begin{bmatrix} S_{11} & S_{12} & S_{13} \\ S_{21} & S_{22} & S_{23} \\ S_{31} & S_{32} & S_{33} \end{bmatrix}$$

$$\vec{u} = \frac{1}{4} \times \sum_{i=1}^{4} \vec{x_i}$$

$$\vec{u} = \frac{1}{4} \times (\vec{x_1} + \vec{x_2} + \vec{x_3} + \vec{x_4})$$

$$\vec{u} = \frac{1}{4} \times (\begin{bmatrix} 7 \\ 3 \\ 9 \end{bmatrix} + \begin{bmatrix} 4 \\ 6 \\ 11 \end{bmatrix} + \begin{bmatrix} 4 \\ 2 \\ 5 \end{bmatrix} + \begin{bmatrix} 5 \\ 5 \\ 7 \end{bmatrix})$$

$$\vec{u} = \begin{bmatrix} 5 \\ 4 \\ 8 \end{bmatrix}$$

$$x = \begin{bmatrix} 7 & 4 & 4 & 5 \\ 3 & 6 & 2 & 5 \\ 9 & 11 & 5 & 7 \end{bmatrix}$$

Para S_{11}:

$$S_{11} = \frac{1}{3} \times \sum_{l=1}^{4} (\vec{x_{1l}} - \vec{u_1})(\vec{x_{1l}} - \vec{u_1})$$

$$S_{11} = \frac{1}{3} \times [(7-5) \times (7-5) + (4-5) \times (4-5) + (4-5) \times (4-5) + (5-5) \times (5-5)]$$

$$S_{11} = 2$$

Para S_{12}:

$$S_{12} = \frac{1}{3} \times \sum_{l=1}^{4} (\vec{x_{1l}} - \vec{u_1})(\vec{x_{2l}} - \vec{u_2})$$

$$S_{12} = \frac{1}{3} \times [(7-5)\times(3-4)+(4-5)\times(6-4)+(4-5)\times(2-4)+(5-5)\times(5-4)]$$

$$S_{12} = \frac{-2}{3}$$

Para S_{13}:

$$S_{13} = \frac{1}{3} \times \sum_{l=1}^{4}(\vec{x_{1l}} - \vec{u_1})(\vec{x_{3l}} - \vec{u_3})$$

$$S_{13} = \frac{1}{3} \times [(7-5)\times(9-8)+(4-5)\times(11-8)+(4-5)\times(5-8)+(5-5)\times(7-8)]$$

$$S_{13} = \frac{2}{3}$$

Para S_{21}:

$$S_{21} = \frac{1}{3} \times \sum_{l=1}^{4}(\vec{x_{2l}} - \vec{u_2})(\vec{x_{1l}} - \vec{u_1})$$

$$S_{21} = \frac{1}{3} \times [(3-4)\times(7-5)+(6-4)\times(4-5)+(2-4)\times(4-5)+(5-4)\times(5-5)]$$

$$S_{21} = \frac{-2}{3}$$

Para S_{22}:

$$S_{22} = \frac{1}{3} \times \sum_{l=1}^{4}(\vec{x_{2l}} - \vec{u_2})(\vec{x_{2l}} - \vec{u_2})$$

$$S_{22} = \frac{1}{3} \times [(3-4)\times(3-4)+(6-4)\times(6-4)+(2-4)\times(2-4)+(5-4)\times(5-4)]$$

$$S_{22} = \frac{10}{3}$$

Para S_{23}:

$$S_{23} = \frac{1}{3} \times \sum_{l=1}^{4} (\vec{x_{2l}} - \vec{u_2})(\vec{x_{3l}} - \vec{u_3})$$

$$S_{23} = \frac{1}{3} \times [(3-4)\times(9-8)+(6-4)\times(11-8)+(2-4)\times(5-8)+(5-4)\times(7-8)]$$

$$S_{23} = \frac{10}{3}$$

Para S_{31}:

$$S_{31} = \frac{1}{3} \times \sum_{l=1}^{4} (\vec{x_{3l}} - \vec{u_3})(\vec{x_{1l}} - \vec{u_1})$$

$$S_{31} = \frac{1}{3} \times [(9-8)\times(7-5)+(11-8)\times(4-5)+(5-8)\times(4-5)+(7-8)\times(5-5)]$$

$$S_{31} = \frac{2}{3}$$

Para S_{32}:

$$S_{32} = \frac{1}{3} \times \sum_{l=1}^{4} (\vec{x_{3l}} - \vec{u_3})(\vec{x_{2l}} - \vec{u_2})$$

$$S_{32} = \frac{1}{3} \times [(9-8)\times(3-4)+(11-8)\times(6-4)+(5-8)\times(2-4)+(7-8)\times(5-4)]$$

$$S_{32} = \frac{10}{3}$$

Para S_{33}:

$$S_{33} = \frac{1}{3} \times \sum_{l=1}^{4} (\vec{x_{3l}} - \vec{u_3})(\vec{x_{3l}} - \vec{u_3})$$

$$S_{33} = \frac{1}{3} \times [(9-8)\times(9-8)+(11-8)\times(11-8)+(5-8)\times(5-8)+(7-8)\times(7-8)]$$

$$S_{33} = \frac{20}{3}$$

Matriz de covariância:

$$S = \begin{bmatrix} 2 & \frac{-2}{3} & \frac{2}{3} \\ \frac{-2}{3} & \frac{10}{3} & \frac{10}{3} \\ \frac{2}{3} & \frac{10}{3} & \frac{20}{3} \end{bmatrix}$$

a-) Os autovalores e autovetores da matriz e mostre a forma fatorada da matriz isto é $\mathbf{S} = \mathbf{V}^{-1}\mathbf{D}\mathbf{V}$. Onde $\mathbf{D}$ é a matriz diagonal dos autovalores e $\mathbf{V}$ é a matriz dos autovetores correspondentes.

```
-> S = [2 (-2/3) (2/3); (-2/3) 10/3 10/3; 2/3 10/3 20/3]
S  =

  2.         -0.6666667   0.6666667
 -0.6666667   3.3333333   3.3333333
  0.6666667   3.3333333   6.6666667

-> Ava = spec(S)
Ava  =

  0.6478472
  2.6182862
  8.7338667

-> [autoVetor,autoValor] = spec(S)
autoVetor  =

  0.5631157   0.8257309   0.0326983
  0.6949042  -0.4945688   0.5220248
 -0.4472235   0.2712382   0.8523033
autoValor  =

  0.6478472   0.          0.
  0.          2.6182862   0.
  0.          0.          8.7338667
```

```
--> V_inv = inv(autoVetor)
 V_inv  =

   0.5631157   0.6949042  -0.4472235
   0.8257309  -0.4945688   0.2712382
   0.0326983   0.5220248   0.8523033

--> V = autoVetor
 V  =

   0.5631157   0.8257309   0.0326983
   0.6949042  -0.4945688   0.5220248
  -0.4472235   0.2712382   0.8523033

--> D = autoValor
 D  =

   0.6478472   0.          0.
   0.          2.6182862   0.
   0.          0.          8.7338667

--> S = V_inv*D*V
 S  =

   3.2166318  -1.6580637  -2.367358
  -1.6580637   1.7247029   1.3605811
  -2.367358    1.3605811   7.0586652
```

b-) Calcule fazendo uso da forma fatorada $\mathbf{S}^4$

```
--> S = V_inv*D*V
 S  =

   3.2166318  -1.6580637  -2.367358
  -1.6580637   1.7247029   1.3605811
  -2.367358    1.3605811   7.0586652

--> S_4 = S^4
 S_4  =

   1186.5427  -721.90248  -2200.8642
  -721.90248   439.69838   1333.0229
  -2200.8642   1333.0229   4239.6355
```

c-) Calcule a norma Euclidiana da matriz e compare com seu valor estimado pelo traço da matriz.

- **Norma de Frobenius (equivalente a norma Euclidiana para vetores)**

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$$

```
--> S = [2 (-2/3) (2/3); (-2/3) 10/3 10/3; 2/3 10/3 20/3]
 S  =

   2.          -0.6666667   0.6666667
  -0.6666667    3.3333333   3.3333333
   0.6666667    3.3333333   6.6666667

--> norm(S,'fro')
 ans  =

   9.1408728
```

- **Norma de Frobenius pelo traço da matriz**

$$||A||_F = \sqrt{\mathrm{Tr}(\boldsymbol{A}\boldsymbol{A}^\top)}.$$

```
--> sqrt(trace(S*S'))
 ans  =

   9.1408728
```

d-) Calcule a distancia Euclidiana dos vetores aleatórios.

$$P = (p_x, p_y, p_z)$$

$$Q = (q_x, q_y, q_z)$$

$$d = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2}$$

$$\mathbf{x}_1 = \begin{bmatrix} 7 \\ 3 \\ 9 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 4 \\ 6 \\ 11 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 4 \\ 2 \\ 5 \end{bmatrix}, \mathbf{x}_4 = \begin{bmatrix} 5 \\ 5 \\ 7 \end{bmatrix}.$$

$$d_{\vec{x_1}\vec{x_2}} = \sqrt{(7-4)^2+(3-6)^2+(9-11)^2}$$

$$d = 4,7$$

$$d_{\vec{x_1}\vec{x_3}} = \sqrt{(7-4)^2+(3-2)^2+(9-2)^2}$$

$$d = 7,7$$

$$d_{\vec{x_1}\vec{x_4}} = \sqrt{(7-5)^2+(3-5)^2+(9-7)^2}$$

$$d = 3,5$$

$$d_{\vec{x_2}\vec{x_3}} = \sqrt{(4-4)^2+(6-2)^2+(11-2)^2}$$

$$d = 9,8$$

$$d_{\vec{x_2}\vec{x_4}} = \sqrt{(4-5)^2+(6-5)^2+(11-7)^2}$$

$$d = 4,2$$

$$d_{\vec{x_3}\vec{x_4}} = \sqrt{(4-5)^2+(2-5)^2+(2-7)^2}$$

$$d = 5,9$$

e-) Calcule a distancia de Mahalanobis dos vetores aleatórios e compare com a distancia Euclidiana

$$d(\vec{x},\vec{y}) = \sqrt{(\vec{x}-\vec{y})^T S^{-1} (\vec{x}-\vec{y})}.$$

Se diferencia da distância Euclidiana, porque considera as correlações (associações estatísticas entre variáveis) do conjunto de dados e não é influenciada pela escala de medição.

```
--> x1 = [7; 3; 9]
 x1  =

   7.
   3.
   9.

--> x2 = [4; 6; 11]
 x2  =

   4.
   6.
   11.

--> x3 = [4; 2; 2]
 x3  =

   4.
   2.
   2.

--> x4 = [5; 5; 7]
 x4  =

   5.
   5.
   7.
```

```
--> S = [2 (-2/3) 2/3; (-2/3) 10/3 10/3; 2/3 10/3 20/3]
 S  =

   2.          -0.6666667   0.6666667
  -0.6666667    3.3333333   3.3333333
   0.6666667    3.3333333   6.6666667

--> S_inv = inv(S)
 S_inv  =

   0.75    0.45  -0.3
   0.45    0.87  -0.48
  -0.3    -0.48   0.42
```

```
--> DM_x1x2 = sqrt((x1-x2)'*S_inv*(x1-x2))
 DM_x1x2  =

   2.4494897

--> DM_x1x3 = sqrt((x1-x3)'*S_inv*(x1-x3))
 DM_x1x3  =

   3.4029399

--> DM_x1x4 = sqrt((x1-x4)'*S_inv*(x1-x4))
 DM_x1x4  =

   2.4494897

--> DM_x2x3 = sqrt((x2-x3)'*S_inv*(x2-x3))
 DM_x2x3  =

   3.6578682

--> DM_x2x4 = sqrt((x2-x4)'*S_inv*(x2-x4))
 DM_x2x4  =

   2.4494897

--> DM_x3x4 = sqrt((x3-x4)'*S_inv*(x3-x4))
 DM_x3x4  =

   2.0928450
```

2º)

Considere a função $E(\mathbf{w})$ onde $\mathbf{w} = [w_1, w_2, \ldots, w_n]^t$ é um vetor com múltiplas variáveis. Usando a expansão em série de Taylor a função pode ser expressa como $E(\mathbf{w}(n) + \Delta\mathbf{w}(n)) = E(\mathbf{w}(n)) + \mathbf{g}^t(\mathbf{w}(n))\Delta\mathbf{w}(n) + \frac{1}{2}\Delta\mathbf{w}^t(n)\mathbf{H}(\mathbf{w}(n))\Delta\mathbf{w}(n) + O(\|\Delta\mathbf{w}\|^3)$, onde $\mathbf{g}(\mathbf{w}(n))$ é o vetor gradiente local definido por $\mathbf{g}(\mathbf{w}) = \dfrac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$ e $\mathbf{H}(\mathbf{w})$ é matriz Hessiana, definida por $\mathbf{H}(\mathbf{w}) = \dfrac{\partial^2 E(\mathbf{w})}{\partial \mathbf{w}^2}$.

Demonstre:

a-) que o método do gradiente da descida mais íngreme é dado por

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta\mathbf{g}(\mathbf{w}(n))$$

$$E\left(w(n)\right) + E'\left(w(n)\right)\Delta w(n) + E''\left(w(n)\right)\frac{\Delta w\left(n\right)^2}{2!} + \ldots + E^{(k)}\left(w(n)\right)\frac{\Delta w\left(n\right)^{(k)}}{k!}$$

$$E\left(w(n) + \Delta w(n)\right) = E\left(w(n)\right) + E'\left(w(n)\right)\Delta w(n)$$

$$\Delta w(n) = -\eta$$

$$E'\left(w(n)\right) = g\left(w(n)\right)$$

$$w\left(n+1\right) = w(n) - \eta g\left(w(n)\right)$$

b-) que o método de Newton é dado por

$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{H}^{-1}(\mathbf{w}(n))\mathbf{g}(\mathbf{w}(n))$

e indique que condição a matriz Hessiana $\mathbf{H}(\mathbf{w}) = \dfrac{\partial^2 E(\mathbf{w})}{\partial \mathbf{w}^2}$ precisa atender para o método poder ser aplicado.

$$E\left(w(n) + \Delta w(n)\right) = E\left(w(n)\right) + g\left(w(n)\right)\Delta w(n) + \frac{1}{2}\Delta w(n)^2 H\left(w(n)\right) + O\left(||\Delta w||^3\right)$$

$$g\left(w(n)\right) = E'\left(w(n)\right)$$

$$H\left(w(n)\right) = E''\left(w(n)\right)$$

$$\frac{d}{\Delta w(n)}\left(E\left(w(n)\right) + g\left(w(n)\right)\Delta w(n) + \frac{1}{2}\Delta w(n)^2 H\left(w(n)\right) + O\left(||\Delta w||^3\right)\right) = 0$$

$$g\left(w(n)\right) + H\left(w(n)\right)\Delta w(n) = 0$$

$$H\left(w(n)\right)\Delta w(n) = -g\left(w(n)\right)$$

$$\Delta w(n) = \frac{-g\left(w(n)\right)}{H\left(w(n)\right)}$$

$$w\left(n+1\right) = w\left(n\right) - \Delta w(n)$$

$$w\left(n+1\right) = w\left(n\right) - \left[-H^{-1}\left(w(n)\right)g\left(w(n)\right)\right]$$

$$w\left(n+1\right) = w\left(n\right) + H^{-1}\left(w(n)\right)g\left(w(n)\right)$$

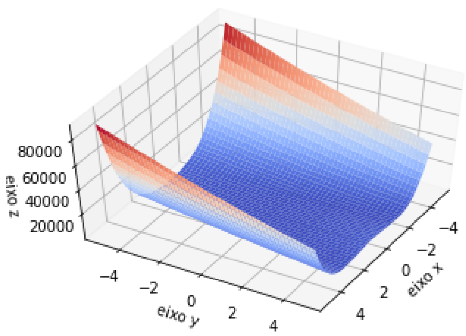
- **Condição da matriz Hessiana para aplicação do método**

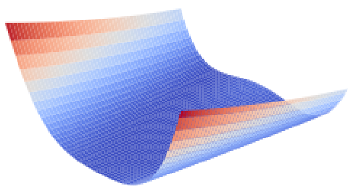
O Método de Newton só é apropriado quando o ponto crítico próximo é um mínimo, ou seja, quando os autovalores da matriz Hessiana são positivos.

3º)

Considere a função Rosenbrock com duas variáveis $f(x, y) = (1 - x)^2 + 100 (y - x^2)^2$ que é uma função comumente utilizada para avaliar o desempenho de um algoritmo de otimização. Seu ponto de mínimo é (1,1). Utilize os métodos do gradiente da descida mais íngreme assim como o método de Newton para solução numérica do cálculo do mínimo. Avalie o desempenho de cada um dos métodos.

O método de Newton convergiu melhor para o mínimo da função do que o método do gradiente da descida mais íngreme. A função Rosenbrock tem um vale estreito e curvo que contém o mínimo (1,1). Por causa do vale, a otimização está ziguezagueando lentamente com pequenos tamanhos de passo em direção ao mínimo. O método de Newton converge, para o mínimo, mais rapidamente, mas o gradiente descendente escapa melhor de pontos de sela.





- **Método do gradiente da descida mais íngreme**

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \mathbf{g}(\mathbf{w}(n))$$

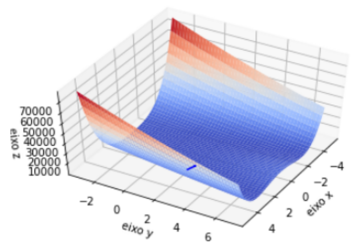
$$\nabla f = \frac{\partial f(x,y)}{\partial(x,y)}$$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

$$\nabla f = \begin{bmatrix} -2 + 2x - 400xy + 400x^3 \\ \\ 200y^2 - 200yx^2 \end{bmatrix}$$

```
xi = np.array([2.0,2.0])
xs = []
zs = []
for i in range(4):
  zi = func3d(xi)
  xs.append(xi)
  zs.append(zi)
  xi=xi-0.0004*grad3d(xi)
xs=np.array(xs)
zs=np.array(zs)
ax=plt.axes(projection='3d')
ax.view_init(50,30)
ax.plot_surface(x[0,:],x[1,:],z,cmap=cm.coolwarm)
ax.plot(xs[:,0],xs[:,1],zs,'-',c='b',zorder=100)
ax.set_xlabel('eixo x')
ax.set_ylabel('eixo y')
ax.set_zlabel('eixo z')
print(xs[0])
print(xs[1])
```

```
[2. 2.]
[1.35116396 2.1596     ]
```



- **Método de Newton**

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{H}^{-1}(\mathbf{w}(n))\mathbf{g}(\mathbf{w}(n))$$

$$H(x,y) = \frac{\partial^2 f(x,y)}{\partial (x,y)^2}$$

$$H(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

$$H(x,y) = \begin{bmatrix} (2 - 400y + 1200x^2) & (-400x) \\ \\ (-400yx) & (400y - 200x^2) \end{bmatrix}$$

```
[16] from scipy import optimize
     import matplotlib.pyplot as plt

[17] def f(x):
         return (1-x)**2+100*(x-x**2)**2

[18] minimo = optimize.newton(f, 2.0)
     print(minimo)

     1.0000000195632512
```

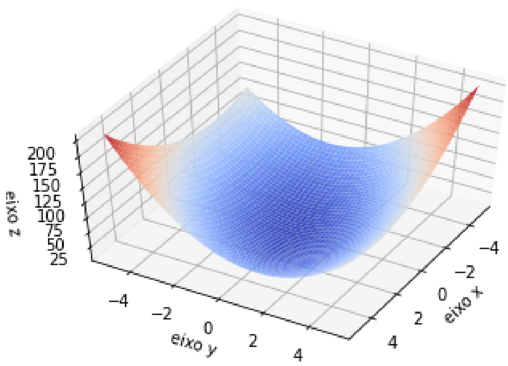
4º)

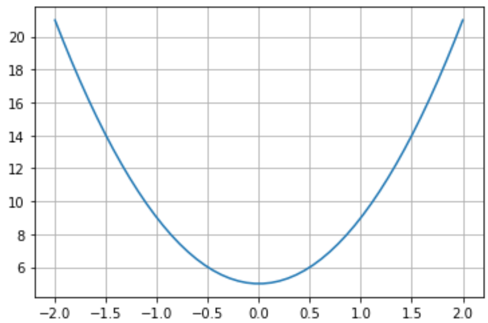
Considere o problema de otimização com restrições:

$$\begin{aligned} &\text{minimize } f(\mathbf{x}) = 2x_1^2 - 2x_1x_2 + 4x_2^2 + 5 \\ &\text{sujeito a:} \\ &\qquad g_1(x_1, x_2) = x_1 + x_2 = 0 \end{aligned}$$

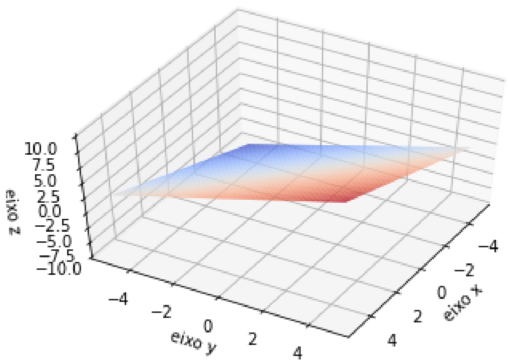
formule o problema usando o funcional de Lagrange e calcule os valores (x_1^*, x_2^*) para mínimo da função.

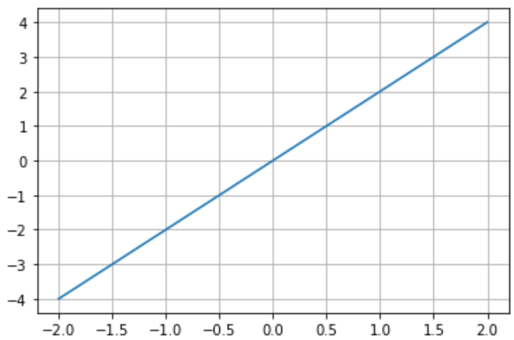
- **Função f(x):**





- **Função de restrição g(x):**





$$\lambda = multiplicador\ de\ lagrange$$
$$\nabla f = \lambda \times \nabla g$$

$$\nabla f = (4x_1 - 2x_2, -2x_1 + 8x_2)$$
$$\nabla g = (1, 1)$$

$$(4x_1 - 2x_2, -2x_1 + 8x_2) = \lambda(1, 1)$$
$$4x_1 - 2x_2 = \lambda \tag{1}$$
$$-2x_1 + 8x_2 = \lambda \tag{2}$$
$$x_1 + x_2 = 0 \tag{3}$$

Igualando as equações (1) e (2):

$$4x_1 - 2x_2 = -2x_1 + 8x_2$$

$$4x_1 + 2x_1 = 8x_2 + 2x_2$$
$$6x_1 = 10x_2$$
$$x_1 = \frac{10}{6}x_2 \tag{4}$$

Atribuindo (4) a (3):

$$\frac{10}{6}x_2 + x_2 = 0$$

$$\frac{16}{6}x_2 = 0$$

$$x_2 = 0 \tag{5}$$

Atribuindo (5) a (4):

$$x_1 = 0$$

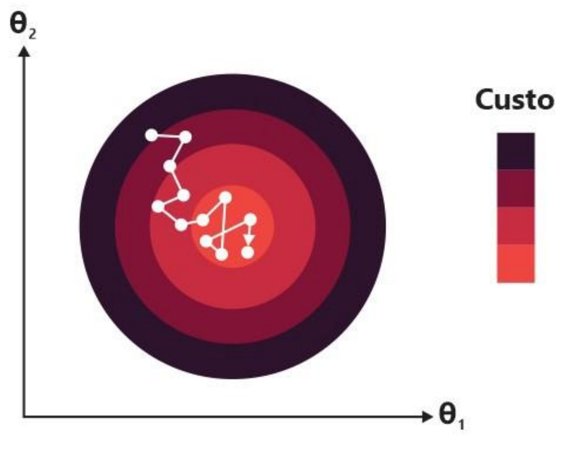
Mínimo da função:

$$(x_1^*, x_2^*) = (0, 0)$$

5º)

Apresente um estudo sobre o método do gradiente estocástico (SGM).

O gradiente estocástico (SGM) é um método de otimização utilizado em aprendizado de máquina. Estocástico, nesse contexto, pode ser entendido como aleatório. O SGM, de forma aleatória, escolhe uma instância do conjunto de treinamento com a finalidade de realizar o cálculo do gradiente se baseando nessa instância. Desta forma, o algoritmo se torna rápido devido a pequena quantidade de dados para manipular em cada iteração. O SGM permite realizar o treinamento com grandes volumes de dados. A função custo não diminui suavemente até o mínimo, ela oscila e diminui na média, ou seja, os valores obtidos não são ótimos.



$$w := w - \eta \nabla Q_i(w).$$

A aleatoriedade, desse método, funciona bem para escapar de ótimos locais, mas existe a possibilidade do algoritmo nunca se estabelecer no objetivo, que é o mínimo global. Reduzir gradualmente a taxa de aprendizado, funciona como uma estratégia para fugir desse problema. As etapas começam com uma taxa de aprendizagem alta para fugir dos mínimos locais e diminuem para atingir o mínimo global. Porém, se a taxa for reduzida rapidamente, o algoritmo poderá ficar preso em um mínimo local. Se reduzida lentamente, poderá saltar em torno do mínimo e obter uma solução insuficiente. Cronograma de aprendizado (learning schedule) é a função responsável por definir a taxa de aprendizado para cada iteração.

6º)

O modelo de neurônio artificial de Mc-Culloch-Pitts faz uso da função de ativação para resposta do neurônio artificial. As funções sigmoíde (ou função logística) e tangente hiperbólica (tangsigmoíde) são normalmente utilizadas nas redes neurais perceptrons de múltiplas camadas tradicionais. A função ReLu (retificador linear) é normalmente utilizadas em redes Deep Learning. Segue abaixo as expressões matemáticas de cada uma:

a-) $\varphi(v) = \dfrac{1}{1+\exp(-av)}$ (sigmoíde); b-) $\varphi(v) = \dfrac{1-\exp(-av)}{1+\exp(-av)} = \tanh(\dfrac{av}{2})$ (tangsigmoíde) ;

c-) $\varphi(v) = \max(0, v)$ (Re-Lu).

i) Faça uma análise comparativa de cada uma destas funções apresentando de forma gráfica a variação da função e da sua derivada com relação a v (potencial de ativação)

- **Funções de ativação:**

São elementos, das redes neurais, responsáveis decidir se um neurônio deve ser ativado ou não. Ou seja, se a entrada que o neurônio está recebendo é relevante para a informação fornecida ou deve ser ignorada. Limita a saída de um neurônio em um intervalo de valores.

- **Combinador Linear:**

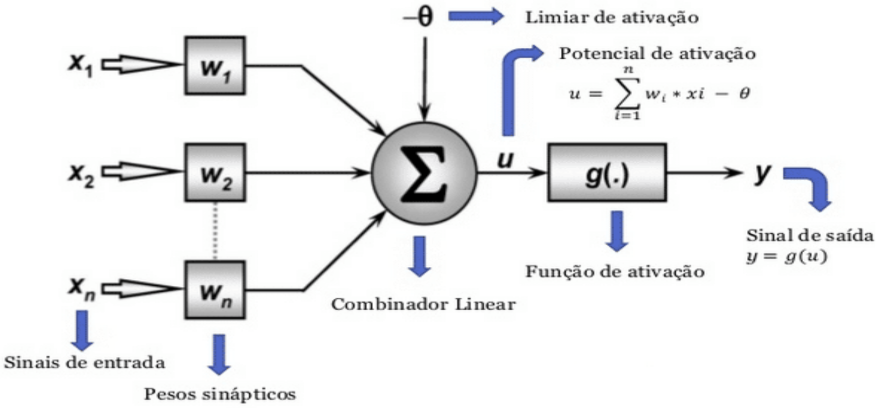
Agregar os sinais de entrada ponderados pelos pesos sinápticos com a finalidade de gerar um potencial de ativação.

- **Limiar de ativação:**

Faz com que o resultado produzido pelo combinador linear possa gerar um valor de disparo de ativação adequado.

- **Potencial de ativação (v):**

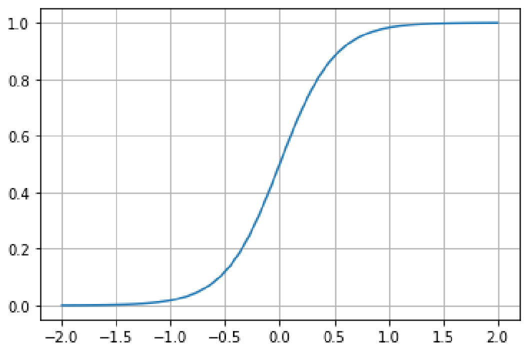
É o resultado obtido pela diferença do valor produzido entre o combinador linear e o limiar de ativação. Se o valor for positivo, ou seja, se v ≥ 0 então o neurônio produz um potencial excitatório; no caso contrário, o potencial se configura como inibitório.



Função Sigmóide

```
[ ] def func_sigmoide(v):
      return 1/(1+np.exp(-4*v))
```

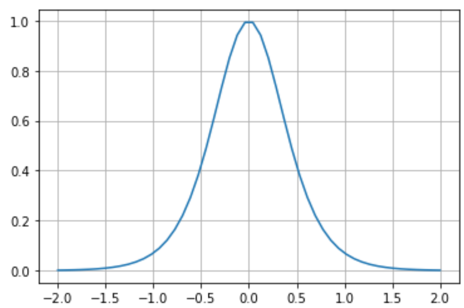
```
[ ] v = np.linspace(-2,2)
    plt.plot(v, func_sigmoide(v))
    plt.grid()
    plt.show()
```

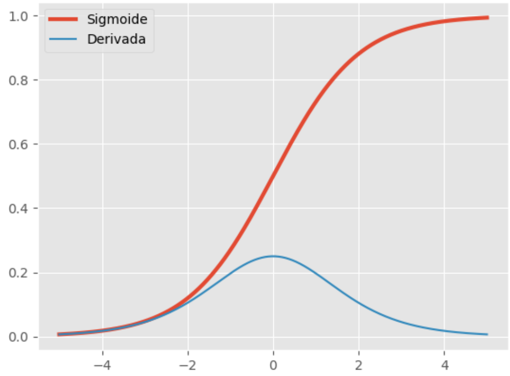


Derivada da Sigmóide

```
[ ] def derivada_sigmoide(v):
      return 4*func_sigmoide(v)*(1-func_sigmoide(v))
```

```
[ ] v = np.linspace(-2,2)
    plt.plot(v, derivada_sigmoide(v))
    plt.grid()
    plt.show()
```

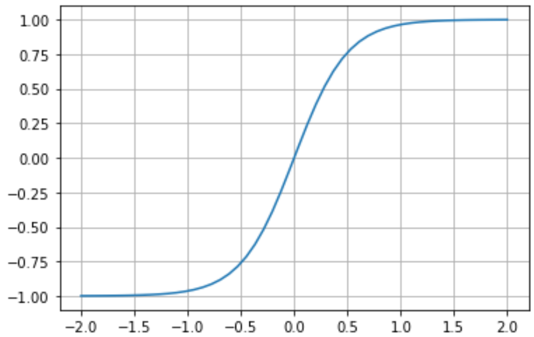




Função Tangsigmóide

```
[14] def func_tangsigmoide(v):
       return ((np.exp(4*v/2)-np.exp(-4*v/2))/
               (np.exp(4*v/2)+np.exp(-4*v/2)))
```

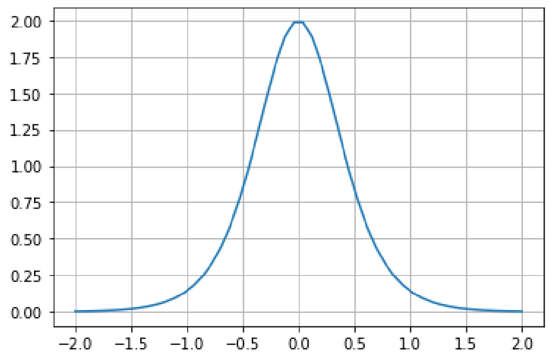
```
[ ]  v = np.linspace(-2,2)
     plt.plot(v, func_tangsigmoide(v))
     plt.grid()
     plt.show()
```

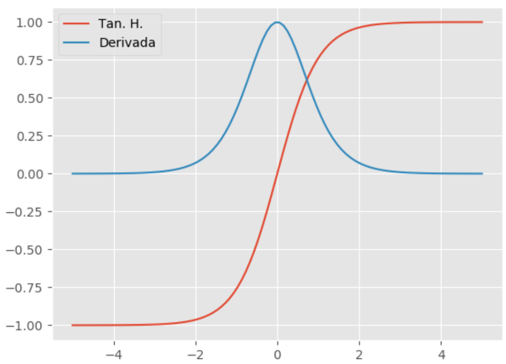


Derivada da Tangsigmóide

```
[ ]  def derivada_tangsigmoide(v):
       return (4/2)*(1-func_tangsigmoide(v)**2)
```

```
[ ]  v = np.linspace(-2,2)
     plt.plot(v, derivada_tangsigmoide(v))
     plt.grid()
     plt.show()
```



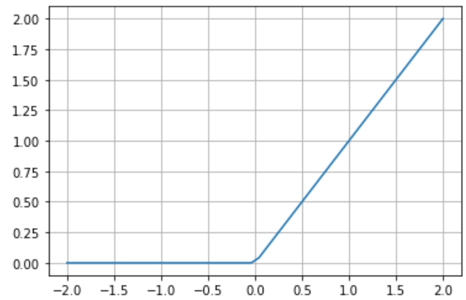


Função ReLU (retificador linear)

```
[ ] from tensorflow.keras.layers import Dense
    Dense(10, activation='relu')
```

```
<keras.layers.core.Dense at 0x7fc1633aacd0>
```

```
[ ] import tensorflow as tf
    from tensorflow.keras.activations import relu
    ## relu = np.maximum(0,v)
    v = np.linspace(-2,2)
    plt.plot(v, relu(v))
    plt.grid()
    plt.show()
```



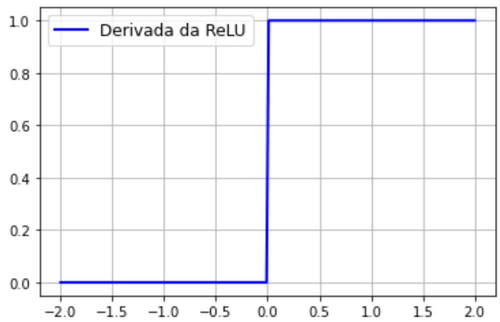
Derivada da ReLU

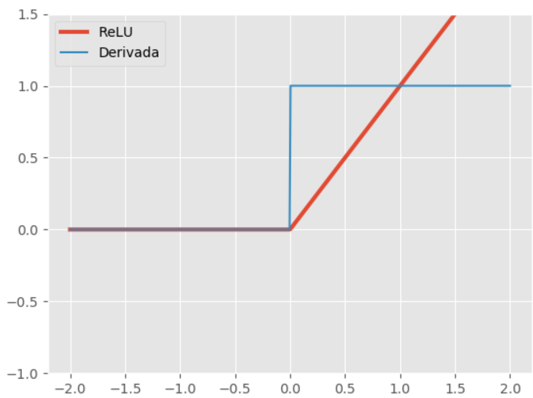
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

def derivative_relu(v):
    return (v >= 0).astype(v.dtype)

v = np.linspace(-2, 2, 200)

plt.figure()
plt.plot(v, derivative_relu(v),
         "b", linewidth=2, label="Derivada da ReLU")
plt.grid(True)
plt.legend(loc='upper left', fontsize=12)
plt.title("", fontsize=14)
plt.savefig('Derivada_da_ReLU.png')
```





ii) Mostre que $\varphi'(v) = \dfrac{d\varphi(v)}{dv} = a\varphi(v)[1-\varphi(v)]$ para função sigmoíde.

$$\phi(v) = \frac{1}{1+e^{-av}}$$

$$\phi'(v) = \frac{d\phi(v)}{dv}$$

$$\phi'(v) = \frac{0 \times (1+e^{-av}) - 1 \times e^{-av} \times (-a)}{(1+e^{-av})^2}$$

$$\phi'(v) = \frac{ae^{-av}}{(1+e^{-av})^2}$$

$$\phi'(v) = \frac{ae^{-av}}{(1+e^{-av})(1+e^{-av})}$$

$$\phi'(v) = \frac{a}{(1+e^{-av})} \times \frac{e^{-av}}{(1+e^{-av})}$$

$$\phi'(v) = \frac{a}{(1+e^{-av})} \times \frac{e^{-av}+1-1}{(1+e^{-av})}$$

$$\phi'(v) = \frac{a}{(1+e^{-av})} \times \left(\frac{1+e^{-av}}{1+e^{-av}} - \frac{1}{1+e^{-av}}\right)$$

$$\phi'(v) = \frac{a}{(1+e^{-av})} \times \left(1 - \frac{1}{1+e^{-av}}\right)$$

$$\phi'(v) = a\phi(v)\left[1-\phi(v)\right]$$

iii) Mostre que $\varphi'(v) = \dfrac{d\varphi(v)}{dv} = \dfrac{a}{2}[1-\varphi^2(v)]$ para função tangsigmoíde.

$$\phi(v) = \frac{1-e^{-av}}{1+e^{-av}} = tanh\left(\frac{av}{2}\right) = \frac{e^{\frac{av}{2}} - e^{\frac{-av}{2}}}{e^{\frac{av}{2}} + e^{\frac{-av}{2}}}$$

$$\phi'(v) = \frac{d\phi(v)}{dv}$$

$$\phi'(v) = \left(e^{\frac{av}{2}} - e^{\frac{-av}{2}}\right) \times \left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)^{-1}$$

$$\phi'(v) = \left[\left(\frac{a}{2}\right)e^{\frac{av}{2}} - \left(\frac{-a}{2}\right)e^{\frac{-av}{2}}\right] \times \left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)^{-1} + \frac{d}{dv}\left[\left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)^{-1}\right] \times \left(e^{\frac{av}{2}} - e^{\frac{-av}{2}}\right)$$

$$\frac{d}{dv}\left[\left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)^{-1}\right]$$

$$g = \left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)$$

$$\frac{d}{dg}\left(g^{-1}\right) = -1 \times g^{-2}$$

$$\frac{d}{dv}\left[\left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)^{-1}\right] = -1 \times g^{-2} \times \frac{d}{dv}\left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right) = -1 \times g^{-2} \times \left[\left(\frac{a}{2}\right) \times e^{\frac{av}{2}} - \left(\frac{a}{2}\right) \times e^{\frac{-av}{2}}\right]$$

$$\frac{d}{dv}\left[\left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)^{-1}\right] = \left(\frac{-a}{2}\right) \times g^{-2} \times \left[e^{\frac{av}{2}} - e^{\frac{-av}{2}}\right] = \left(\frac{-a}{2}\right) \times \frac{\left(e^{\frac{av}{2}} - e^{\frac{-av}{2}}\right)}{\left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)^{2}}$$

$$\phi'(v) = \frac{\left(\frac{a}{2}\right) \times \left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)}{\left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)} + \left[\left(\frac{-a}{2}\right) \times \frac{\left(e^{\frac{av}{2}} - e^{\frac{-av}{2}}\right)}{\left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)^{2}} \times \left(e^{\frac{av}{2}} - e^{\frac{-av}{2}}\right)\right]$$

$$\phi'(v) = \left(\frac{a}{2}\right) - \left(\frac{a}{2}\right) \times \frac{\left(e^{\frac{av}{2}} - e^{\frac{-av}{2}}\right)^{2}}{\left(e^{\frac{av}{2}} + e^{\frac{-av}{2}}\right)^{2}}$$

$$\phi'(v) = \left(\frac{a}{2}\right) \times \left[1 - \phi^{2}(v)\right]$$

7º)

Dada a função E(n) que corresponde a função custo com base no erro ao quadrado, calculada na saída do neurônio j de uma rede neural perceptron de múltiplas camadas

$\mathrm{E}(\mathrm{n}) = \frac{1}{2}\mathrm{e}_{\mathrm{j}}^{2}(n)$ *com*

$e_j(n) = d_j(n) - y_j(n)$,

$y_j(n) = \varphi(v_j(n));\quad v_j(n) = \sum_{i=0}^{m} w_{ji}(n)\, y_i(n)$

i) Demonstre que $\dfrac{\partial \mathrm{E}(\mathrm{n})}{\partial \mathrm{w}_{\mathrm{ji}}(\mathrm{n})} = -e_j(n)\varphi'(v_j(n))\, y_i(n)$

$$E(n) = \frac{1}{2} \times (d_j(n) - y_j(n))^2$$

$$E(n) = \frac{1}{2} \times (d_j(n) - \phi(v_j(n)))^2$$

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = (d_j(n) - \phi(v_j(n))) \times [-\phi'(v_j(n))] \times y_i(n)$$

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n) \times \phi'(v_j(n)) \times y_i(n)$$

ii) Considerando as funções de ativação da questão anterior apresente a expressão $\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi'(v_j(n))y_i(n)$

- **Para a função sigmóide:**

$$\phi(v_j(n)) = \frac{1}{1 + e^{-av_j(n)}}$$

$$\phi'(v_j(n)) = a\phi(v_j(n))\,[1 - \phi(v_j(n))]$$

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n) \times a\phi(v_j(n))\,[1 - \phi(v_j(n))] \times y_i(n)$$

- **Para a função tangsigmóide:**

$$\phi(v_j(n)) = \frac{e^{\frac{av_j(n)}{2}} - e^{\frac{-av_j(n)}{2}}}{e^{\frac{av_j(n)}{2}} + e^{\frac{-av_j(n)}{2}}}$$

$$\phi'(v_j(n)) = \left(\frac{a}{2}\right)\left[1 - \phi^2(v_j(n))\right]$$

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n) \times \left(\frac{a}{2}\right)\left[1 - \phi^2(v_j(n))\right] \times y_i(n)$$

- **Para a função Re-Lu:**

$$\phi(v_j(n)) = max\left(0, v_j(n)\right)$$

$$\phi'(v_j(n)) = \begin{cases} 0, & v_j(n) < 0 \\ 1, & v_j(n) \geq 0 \end{cases}$$

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n) \times \phi'(v_j(n)) \times y_i(n)$$

8º)

O método da máxima verossimilhança (Max-Likelihood) aplicado na determinação de parâmetros desconhecidos de um modelo de distribuição de probabilidades. O método é bastante utilizado para o desenvolvimento de algoritmos de aprendizagem de máquinas em particular em redes neurais. O problema considerado nesta questão envolve o modelo probabilístico de um comitê de redes especialistas e uma rede que indica qual das redes especialista é a mais adequada para processar uma determinada entrada. O problema consiste em determinar os parâmetros **w** das redes especialistas e **a** da rede que decide qual a rede especialista deve processar um determinada entrada. O modelo probabilístico é denominado de modelo de misturas de gaussianas, dado por

$$f(\mathbf{d}\,|\,\mathbf{x};\mathbf{w},\mathbf{a}) = \sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{K}} g_i f(\mathbf{d}\,|\,\mathbf{x},\mathbf{i};\mathbf{w},\mathbf{a}) = \frac{1}{(2\pi)^{q/2}} \sum_{i=1}^{K} g_i \exp(-\frac{1}{2}\left\|\mathbf{d}-\mathbf{y}_i\right\|^2)$$

$$y_i^{(m)} = \mathbf{x}^{\mathbf{t}}\mathbf{w}_{\mathbf{i}}^{(\mathbf{m})}, \qquad \begin{cases} i = 1,2,\ldots,K \\ m = 1,2\ldots,q \end{cases}$$

$$g_i = \frac{\exp(u_i)}{\sum_{j=1}^{K}\exp(u_j)} \qquad 0 \le g_1 \le 1 \qquad \sum_{i=1}^{K} g_i = 1$$

$$u_i = \mathbf{x}^{\mathbf{t}}\mathbf{a}$$

sendo **d** vetor de resposta desejada, $\mathbf{y_i}$ vetor de saída de cada rede especialista, $\mathbf{w_i}$ o vetor de pesos de cada rede especialista, **a** o vetor de pesos da rede decisora, q a dimensão do vetor de entrada e do vetor de pesos e K o número de redes especialistas . A função $g_i = \frac{\exp(u_i)}{\sum_{j=1}^{K}\exp(u_j)}$ é conhecida como função sofmax e corresponde neste problema a probabilidade a priori da rede i ser escolhida. A função h_i definida abaixo corresponde a probabilidade da posteriori da rede ser escolhida.

Definindo função log de verossimilhança como

$$l(\mathbf{w},\mathbf{a}) = \ln \sum_{i=1}^{K} g_i \exp(-\frac{1}{2}\left\|\mathbf{d}-\mathbf{y}_i\right\|^2)$$

o problema consiste em com relação aos parâmetros **w** e **a. F**azendo uso do método do gradiente da descida mais íngreme para determinação max l(**w,a).**

mostre que

$$\mathbf{w}_{\mathbf{i}}^{(\mathbf{m})}(n+1) = \mathbf{w}_{\mathbf{i}}^{(\mathbf{m})}(n) + \eta h_i(n) e_i^{(m)}(n)\mathbf{x}$$

$$\mathbf{a}_{\mathbf{i}}(n+1) = \mathbf{a}_{\mathbf{i}}(n) + \mu[h_i(n) - g_i(n)]$$

$$e_i^{(m)} = d^{(m)} - y_i^{(m)}(n)$$

$$h_i(n) = \frac{g_i(n)\exp(-\frac{1}{2}\left\|\mathbf{d}-\mathbf{y}_{\mathbf{i}}(n)\right\|^2)}{\sum_{j=1}^{K} g_j(n)\exp(-\frac{1}{2}\left\|\mathbf{d}-\mathbf{y}_j(n)\right\|^2)} \qquad 1 \le h_i(n) \le 1 \qquad \sum_{i=1}^{K} h_i(n) = 1$$

$$l\left(w,a\right)=ln\left(\sum_{i=1}^{k}g_{i}e^{\left(\frac{-1}{2}||e_{i}||^{2}\right)}\right)$$

$$u=\sum_{i=1}^{k}g_{i}e^{\left(\frac{-1}{2}||e_{i}||^{2}\right)}$$

$$\frac{\partial l(w,a)}{\partial w}=\frac{d}{du}ln\left(u\right)\times g_{i}e^{\left(\frac{-1}{2}||e_{i}||^{2}\right)}\times\left(-e_{i}\right)\times\left(-x\right)$$

$$\frac{\partial l(w,a)}{\partial w}=\frac{1}{\sum_{i=1}^{k}g_{i}e^{\left(\frac{-1}{2}||e_{i}||^{2}\right)}}\times g_{i}e^{\left(\frac{-1}{2}||e_{i}||^{2}\right)}\times\left(-e_{i}\right)\times\left(-x\right)$$

$$\frac{\partial l(w,a)}{\partial w}=\frac{g_{i}e^{\left(\frac{-1}{2}||e_{i}||^{2}\right)}}{\sum_{i=1}^{k}g_{i}e^{\left(\frac{-1}{2}||e_{i}||^{2}\right)}}\times e_{i}x$$

$$\frac{\partial l(w,a)}{\partial w}=\nabla l\left(w,a\right)$$

$$w_{i}\left(n+1\right)=w_{i}\left(n\right)+\eta\nabla l\left(w,a\right)$$

$$w_{i}\left(n+1\right)=w_{i}\left(n\right)+\eta\times\frac{g_{i}e^{\left(\frac{-1}{2}||e_{i}||^{2}\right)}}{\sum_{i=1}^{k}g_{i}e^{\left(\frac{-1}{2}||e_{i}||^{2}\right)}}e_{i}x$$

$$h_i(n) = \frac{g_i e^{\left(\frac{-1}{2}||e_i||^2\right)}}{\sum_{i=1}^{k} g_i e^{\left(\frac{-1}{2}||e_i||^2\right)}}$$

$$w_i\left(n+1\right) = w_i\left(n\right) + \eta \times h_i(n) \times e_i \times x \tag{1}$$

$$\frac{\partial l(w,a)}{\partial a} = \frac{d}{du} ln\left(u\right) \times g_i\left[1-g_i\right] e^{\left(\frac{-1}{2}||e_i||^2\right)}$$

$$\frac{\partial l(w,a)}{\partial a} = \frac{g_i e^{\left(\frac{-1}{2}||e_i||^2\right)} \times \left[1-g_i\right]}{\sum_{i=1}^{k} g_i e^{\left(\frac{-1}{2}||e_i||^2\right)}}$$

$$\frac{\partial l(w,a)}{\partial a} = \nabla l\left(w,a\right)$$

$$a_i\left(n+1\right) = a_i\left(n\right) + \mu \nabla l\left(w,a\right)$$

$$a_i\left(n+1\right) = a_i\left(n\right) + \mu \times h_i(n) \times \left[1-g_i\right] \tag{2}$$

9º)

Implementações Computacionais de Redes Neurais.

Para cada uma das aplicações abaixo apresente a curva do erro de treinamento e de validação:

9.1-) Defina a estrutura de uma rede perceptron de múltiplas camadas para aproximar as funções abaixo. Defina o conjunto de treinamento e de validação. Compare para os itens b e c a aproximação da função obtida pela rede neural com as curvas exatas. Apresente para cada caso a curva do erro médio de treinamento com relação ao número de épocas e a curva do erro médio com o conjunto de validação.

a) a função lógica $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$

b) a função real

$$f(x_1, x_2) = \left(\frac{\cos(2\pi x_1)}{1-(4x_1)^2} sen(\pi x_1)/\pi x_1 \right)\left(\frac{\cos(2\pi x_2)}{1-(4x_2)^2} sen(\pi x_2)/\pi x_2 \right) \qquad -4\pi \le x_1 \le 4\pi \qquad -4\pi \le x_2 \le 4\pi$$

Épocas: número de vezes que um algoritmo de machine learning analisa um conjunto de dados completo. No caso do uso de minilotes o número de épocas não é igual ao número de interações.

Função de perda (loss function): calcula a diferença entre a saída desejada e a saída atual. Informa o nível de precisão da rede neural fazendo previsões para as entradas.

Activation function: Sigmóide

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

Loss function: Entropia cruzada

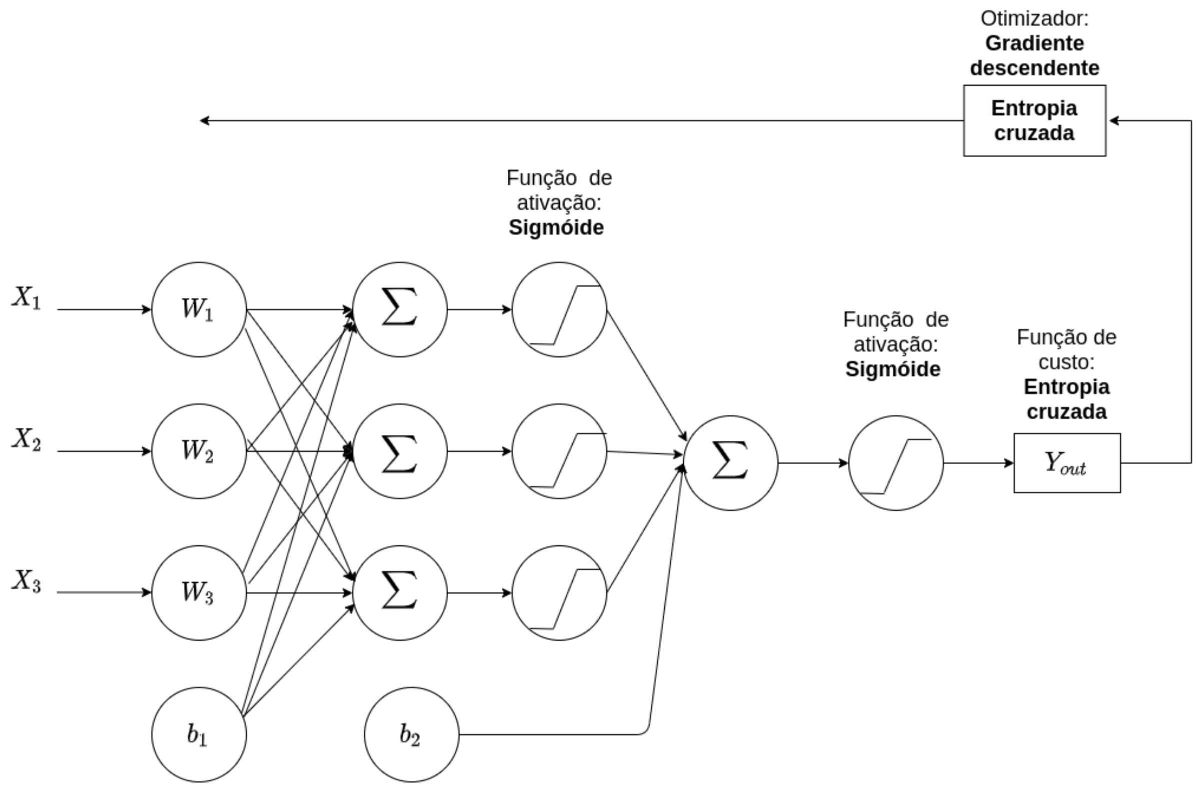
$$L = -\sum_{i=1}^{N} y^{(i)} * \log \hat{y}^{(i)}$$

Optimizer: Gradient Descent

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \mathbf{g}(\mathbf{w}(n))$$

$$f(X_1, X_2, X_3) = X_1 \oplus X_2 \oplus X_3$$

X1	X2	X3	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
from matplotlib import pyplot as plt
```

```
X = tf.placeholder(tf.float32, shape=[8,3])
Y = tf.placeholder(tf.float32, shape=[8,1])
```

First Layer

```
[34] # First layer has three neurons taking three input values
     W1 = tf.Variable(tf.random_uniform([3,3]))
     # each neuron has one bias
     B1 = tf.Variable(tf.zeros([3]))
     # First Layer's output is Z which is the sigmoid(W1 * X + B1)
     Z = tf.sigmoid(tf.matmul(X, W1) + B1)
```

Second Layer

```
# Second layer has one neurons taking three input values.
W2 = tf.Variable(tf.random_uniform([3,1]))
# one neuron has one bias.
B2 = tf.Variable(tf.zeros([1]))
# Second Layer's output is Y_out which is the sigmoid(W2 * Z + B2)
Y_out = tf.sigmoid(tf.matmul(Z, W2) + B2)
```

Loss Function

```
[36] # cross entropy
     loss = tf.reduce_mean(-1*((Y*tf.log(Y_out))+((1-Y)*tf.log(1.0-Y_out))))
```

Optimizer

```
[37] # Gradient Descent
     train_step = tf.train.GradientDescentOptimizer(0.05).minimize(loss)
```

Train

```
[38] train_X = [[0,0,0],[0,0,1],[0,1,0],[0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1]]
     train_Y = [[0],[1],[1],[1],[1],[1],[1],[0]]
```

```
# initialize
init = tf.global_variables_initializer()
# Start training
with tf.Session() as sess:
    # Run the initializer
    sess.run(init)
    print("train data: "+str(train_X))
    for i in range(20000):
        sess.run(train_step, feed_dict={X: train_X, Y: train_Y})
        if i % 5000 == 0:
            print('Epoch : ', i)
            print('Output : ', sess.run(Y_out, feed_dict={X: train_X, Y: train_Y}))
            print('Error : ', sess.run(Y-Y_out, feed_dict={X: train_X, Y: train_Y}))
    print('Final Output : ', sess.run(Y_out, feed_dict={X: train_X, Y: train_Y}))
    print('Error : ', sess.run(Y-Y_out, feed_dict={X: train_X, Y: train_Y}))
```

```
train data: [[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]
Epoch :  0
Output :  [[0.6253079 ]
 [0.6569054 ]
 [0.6342983 ]
 [0.6647425 ]
 [0.62845767]
 [0.6596839 ]
 [0.6370884 ]
 [0.6670307 ]]
Error :  [[-0.6253079 ]
 [ 0.3430946 ]
 [ 0.36570168]
 [ 0.33525747]
 [ 0.37154233]
 [ 0.34031612]
 [ 0.36291158]
 [-0.6670307 ]]
```

```
Epoch :  5000
Output :  [[0.5775588 ]
 [0.7487359 ]
 [0.7496636 ]
 [0.803478  ]
 [0.75763285]
 [0.8058163 ]
 [0.80465615]
 [0.81578493]]
Error :  [[-0.5775588 ]
 [ 0.2512641 ]
 [ 0.2503364 ]
 [ 0.196522  ]
 [ 0.24236715]
 [ 0.1941837 ]
 [ 0.19534385]
 [-0.81578493]]
```

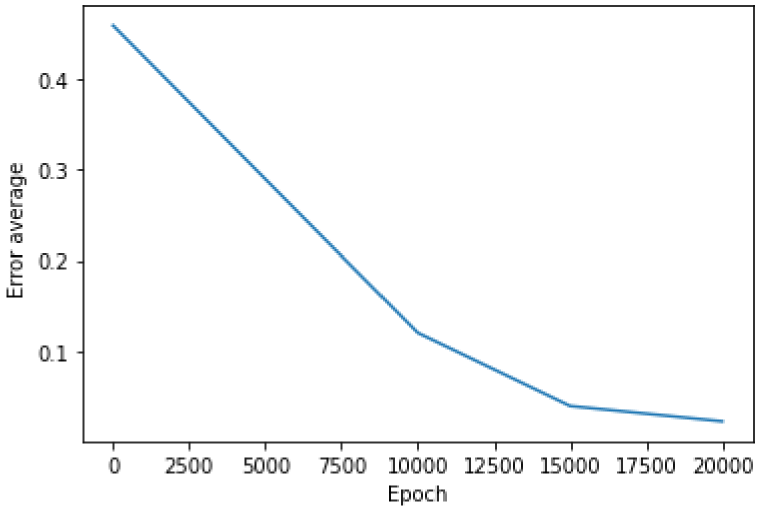
```
Epoch :  10000
Output :  [[0.2107355 ]
 [0.8980821 ]
 [0.896824  ]
 [0.8304689 ]
 [0.89930034]
 [0.83938766]
 [0.8330858 ]
 [0.6406274 ]]
Error :  [[-0.2107355 ]
 [ 0.10191792]
 [ 0.103176  ]
 [ 0.1695311 ]
 [ 0.10069966]
 [ 0.16061234]
 [ 0.16691422]
 [-0.6406274 ]]
```

```
Epoch :  15000
Output :  [[0.08434626]
 [0.9813998 ]
 [0.98161453]
 [0.9481945 ]
 [0.9813032 ]
 [0.94822025]
 [0.94817185]
 [0.15094462]]
Error :  [[-0.08434626]
 [ 0.01860023]
 [ 0.01838547]
 [ 0.0518055 ]
 [ 0.01869678]
 [ 0.05177975]
 [ 0.05182815]
 [-0.15094462]]
```

```
Final Output :  [[0.046799   ]
 [0.99084175]
 [0.9908956 ]
 [0.97722423]
 [0.9907503 ]
 [0.97722197]
 [0.9772068 ]
 [0.06419009]]
```

```
Error :  [[-0.046799   ]
 [ 0.00915825]
 [ 0.00910437]
 [ 0.02277577]
 [ 0.00924969]
 [ 0.02277803]
 [ 0.02279317]
 [-0.06419009]]
```

- **Curva do erro médio de treinamento com relação ao número de épocas**



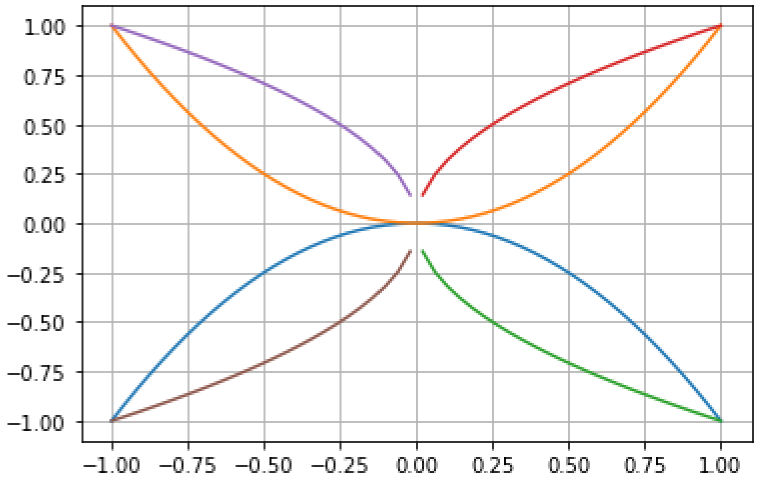
- **Validação (teste):**

```
print('Validation:', sess.run(Y_out, feed_dict={X: [[0,0,0],[0,0,1],[0,1,0]
                                                  ,[0,1,1],[1,0,0],[1,0,1],
                                                  [1,1,0],[1,1,1]]}))
```

```
Validation: [[0.04107541]
 [0.9914405 ]
 [0.99140465]
 [0.9778832 ]
 [0.9912523 ]
 [0.9778877 ]
 [0.97789216]
 [0.06472066]]
```

9.2-) Utilize a rede neural perceptron de múltiplas camadas para fazer a predição de um passo (x^(n+1)) da série temporal $x(n) = 1 + \cos(n + \cos^2(n))$. Avalie o desempenho mostrando a curva a série temporal, a curva de predição o e curva do erro de predição.

9.3-) Considere o problema de classificação de padrões bidimensionais constituído neste caso de 2 padrões. A distribuição dos padrões tem como base um quadrado centrado na origem interceptando os eixos nos pontos +1 e -1 de cada eixo. Os pontos +1 e -1 de cada eixo são centros de quatro semicírculos que se interceptam no interior do quadrado originando uma classe e a outra classe corresponde as regiões de não interseção. Após gerar aleatoriamente os dados que venham formar estas distribuições de dados, selecione um conjunto de treinamento e um conjunto de validação. Solucione este problema considerando uma rede perceptron de múltiplas camada

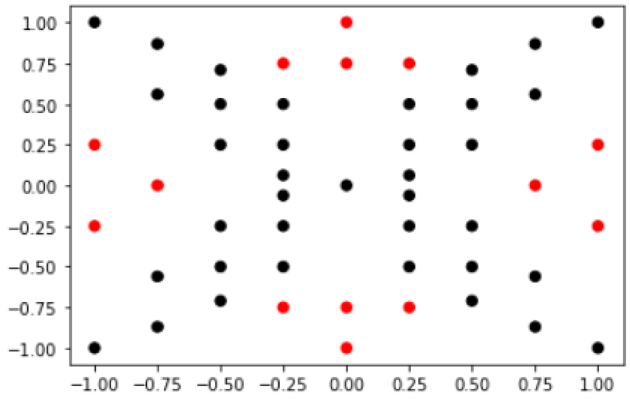


Visualizando os dados:

```
# Seta um arranjo de cores
colormap = np.array(['r', 'k'])

# Plotar os dados em seus respectivos eixos
# Configura o arranjo de cores para os Targets
plt.scatter(inputs.x, inputs.y, c=colormap[inputs.Targets], s=40)
```

```
<matplotlib.collections.PathCollection at 0x7f11b6e11410>
```




```
# MLP
mlpc = MLPClassifier(n_iter_no_change=25, activation='logistic',
                     solver='sgd', hidden_layer_sizes=(4,3,1))

# Treina MLP
mlpc.fit(inputs[['x', 'y']], inputs['Targets'])

# Print the results
classif =  mlpc.predict(inputs[['x', 'y']])
print("Classification", classif)
print("Actual     ", np.array(inputs.Targets))
print("Training accuracy ", mlpc.score(inputs[['x', 'y']],
                                   inputs[['Targets']]) * 100, "%", sep='')
```

```
Classification [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Actual     [1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
 0 0 0 0 0 0 1 1 1 1 1 1 1 1]
Training accuracy 72.54901960784314%
```

10º) Trabalho:

Apresente um estudo com uma análise comparativa dos algoritmos: **AdaGrad, RMSProp e Adam, que tem como base o** método do gradiente estocástico (SGM) e são utilizados no processo de aprendizagem de redes neurais deep learning.

Os algoritmos: AdaGrad, RMSProp e Adam são inseridos na categoria dos algoritmos de taxa de aprendizagem adaptativa. Nesta categoria, a taxa de aprendizagem é modificada de uma forma específica em cada um desses algoritmos.

- **AdaGrad**

O Adagrad ajusta a taxa de aprendizagem conforme a frequência dos parâmetros. A taxa de aprendizagem resultante dos parâmetros mais frequentes são atualizadas mais constantemente, contudo com valores menores. No caso dos parâmetros de ocorrência menos frequentes, a taxa de aprendizagem é atualizada com valores maiores. Esta característica torna o algoritmo Adagrad propício a trabalhar com dados esparsos. O vetor de parâmetros é atualizado de acordo com a fórmula:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t,$$

$$\eta = taxa\ de\ aprendizagem$$

$$g_t = gradiente\ da\ função\ custo\ no\ tempo$$

$$\epsilon = termo\ que\ serve\ para\ evitar\ divisão\ por\ zero$$

$$G_t = matriz\ diagonal$$

Vantagem: Não é necessário ajustar a taxa de aprendizagem manualmente.

Desvantagem: O acúmulo dos gradientes pode causar a estagnação da aprendizagem, diminuindo a acurácia para um número elevado de iterações.

- **RMSProp**

Substitui a soma dos gradientes, presente no algoritmo Adagrad, por uma média móvel dos quadrados dos gradientes.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{E\left[g^2\right]_t + \epsilon}} g_t,$$

$$g = gradiente\ da\ função\ custo$$

$$g_t = gradiente\ da\ função\ custo\ no\ tempo$$

$$E = [g^2]_t = média\ móvel\ dos\ gradientes\ no\ tempo\ x$$

$$\epsilon = termo\ que\ serve\ para\ evitar\ divisão\ por\ zero$$

$$\eta = taxa\ de\ aprendizagem$$

Vantagem: A média móvel diminui o impacto da soma dos gradientes, evitando o problema da estagnação que ocorre no algoritmo Adagrad.

- **Adam (baseado no método do gradiente estocástico SGM)**

O algoritmo Adam usa as médias móveis dos gradientes e dos quadrados dos gradientes com objetivo de ajustar os parâmetros.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \cdot \hat{\mathbf{m}}_t,$$

$$\theta = vetor\ de\ pesos$$

$$\eta = taxa\ de\ aprendizagem$$

$$\vec{m}_t = vetor\ corrigido\ das\ médias\ dos\ gradientes\ no\ tempo\ t$$

$$\vec{v}_t = vetor\ corrigido\ das\ médias\ dos\ quadrados\ dos\ gradientes\ no\ tempo\ t$$

$$\epsilon = termo\ que\ serve\ para\ evitar\ divisão\ por\ zero$$

Vantagem:

- Converge mais rápido que o algoritmo Adagrad para redes neurais convolucionais
- Possui eficiência igual ou superior aos algoritmos Adagrad e RMSProp