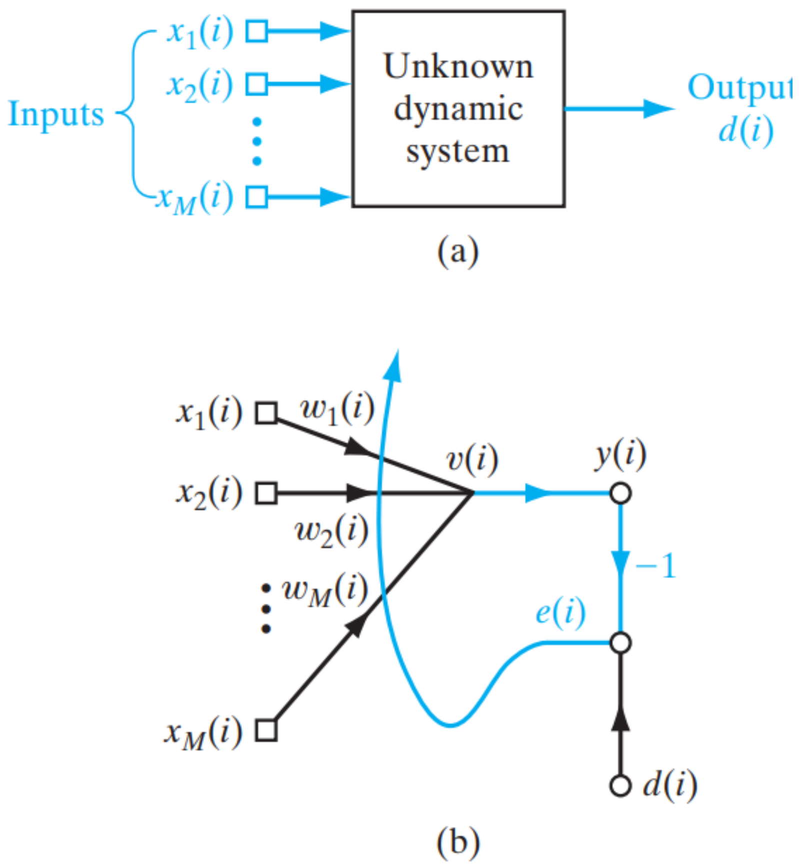
Universidade Presbiteriana Mackenzie

Perceptron de Única Camada

Prof. Dr. Leandro Augusto da Silva
leandroaugusto.silva@mackenzie.br
Laboratório de Big Data e Métodos Analíticos Aplicados
Faculdade de Computação e Informática
Programa de Pós-Graduação em Engenharia Elétrica e Computação

O Problema de Filtragem Adaptativa

FIGURE 3.1 (a) Unknown dynamic system. (b) Signal-flow graph of adaptive model for the system; the graph embodies a feedback loop set in color.

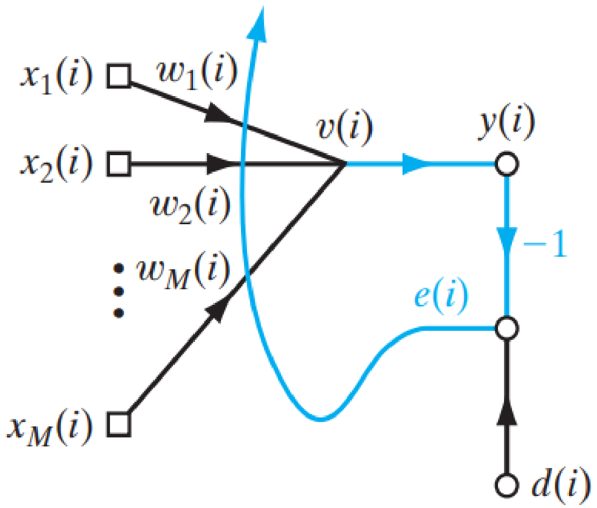


O Problema de Filtragem Adaptativa

- O problema é o de como projetar um modelo de múltiplas entradas-única saída do sistema dinâmico desconhecido.
- O modelo neuronal opera sob a influência de um algoritmo que controla os ajustes necessários dos pesos sinápticos do neurônio, considerando:
  - O algoritmo inicia com uma configuração arbitrária de pesos
  - Os ajustes são feitos de forma contínua
  - Os cálculos dos ajustes são completados dentro de um intervalo de tempo

O Problema de Filtragem Adaptativa

- O modelo neuronal descrito é conhecido como um filtro adaptativo, cuja operação é constituída de dois processos:
  1) Processo de Filtragem
     - Uma saída: y(i)
     - Um sinal de erro
  2) Processo adaptativo
     - Ajuste dos pesos de acordo com o sinal de erro



Técnicas de otimização irrestritas

- A maneira pela qual o sinal de erro é usado para controlar os ajustes dos pesos é determinada pela *função de custo* ($\mathcal{E}(\mathbf{w})$)
- Considere a função diferenciável de um vetor de peso (parâmetro) desconhecido $\mathbf{w}$. O objetivo é encontrar a solução ótima $\mathbf{w}^*$

$$\mathcal{E}(\mathbf{w}^*) \le \mathcal{E}(\mathbf{w})$$

- Ou seja, deseja-se minimizar $\mathcal{E}(\mathbf{w})$ em relação a $\mathbf{w}$

Técnicas de otimização irrestritas

A condição necessária para a otimização é

$$\nabla \mathscr{E}(\mathbf{w}^*) = \mathbf{0} \tag{3.7}$$

onde ∇ é o *operador gradiente*:

$$\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \dots, \frac{\partial}{\partial w_m}\right]^T \tag{3.8}$$

e $\nabla \mathscr{E}(\mathbf{w})$ é o *vetor gradiente* da função de custo:

$$\nabla \mathscr{E}(\mathbf{w}) = \left[\frac{\partial \mathscr{E}}{\partial w_1}, \frac{\partial \mathscr{E}}{\partial w_2}, \dots, \frac{\partial \mathscr{E}}{\partial w_m}\right]^T \tag{3.9}$$

Uma classe de algoritmos de otimização irrestritos que é particularmente adequada para o projeto de filtros adaptativos é baseada na idéia da *descida iterativa* local:

> *Iniciando com uma suposição inicial representada por* $\mathbf{w}(0)$, *gere uma seqüência de vetores de peso* $\mathbf{w}(1), \mathbf{w}(2), \dots,$ *de modo que a função de custo* $\mathscr{E}(\mathbf{w})$ *seja reduzida a cada iteração do algoritmo, como mostrado por*
>
> $$\mathscr{E}(\mathbf{w}(n+1)) < \mathscr{E}(\mathbf{w}(n)) \tag{3.10}$$
>
> *onde* $\mathbf{w}(n)$ *é o valor antigo do vetor de peso e* $\mathbf{w}(n+1)$ *é o seu valor atualizado.*

Esperamos que este algoritmo eventualmente convirja para a solução ótima $\mathbf{w}^*$. Dizemos "esperamos" porque há uma nítida possibilidade de o algoritmo divergir (i.e., se tornar instável) a menos que sejam tomadas precauções especiais.

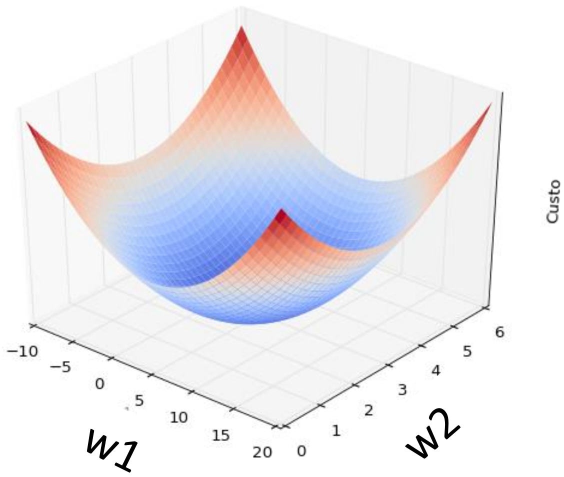
Haykin (2001_, pg 147)

Gradiente descendente*

- Para visualização da função de custo, considere um neurônio com duas entradas, portanto dois pesos plotados com a função de custo

- O ponto de mínimo está onde w1=5 e w2=3. Portanto:

$$\nabla = \left[\frac{\partial}{\partial w1}, \frac{\partial}{\partial w2}\right]$$



  - O gradiente é simplesmente um vetor de derivadas parciais que dão a inclinação da "tigela" em cada ponto e em cada direção
  - Portanto, a direção oposta ao gradiente leva ao ponto de mínimo

*https://matheusfacure.github.io/2017/02/20/MQO-Gradiente-Descendente/

Método de Descida mais íngrime

No método da descida mais íngreme, os ajustes sucessivos aplicados ao vetor de peso $\mathbf{w}$ são na direção da descida mais íngreme, isto é, em uma direção oposta ao *vetor do gradiente* $\nabla\mathscr{E}(\mathbf{w})$. Por conveniência de apresentação, escrevemos

$$\mathbf{g} = \nabla\mathscr{E}(\mathbf{w}) \tag{3.11}$$

Correspondentemente, o algoritmo da descida mais íngreme é descrito formalmente por

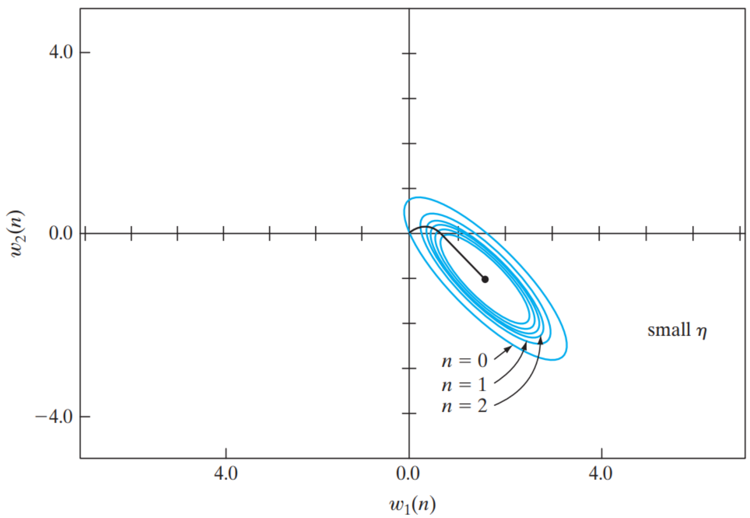
$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta\mathbf{g}(n) \tag{3.12}$$

onde η é uma constante positiva chamada de *tamanho do passo* ou *parâmetro de taxa de aprendizagem*, e $\mathbf{g}(n)$ é o vetor do gradiente calculado no ponto $\mathbf{w}(n)$. Passando da iteração n para $n + 1$, o algoritmo aplica a *correção*

$$\begin{aligned}\Delta\mathbf{w}(n) &= \mathbf{w}(n+1) - \mathbf{w}(n) \\ &= -\eta\mathbf{g}(n)\end{aligned} \tag{3.13}$$

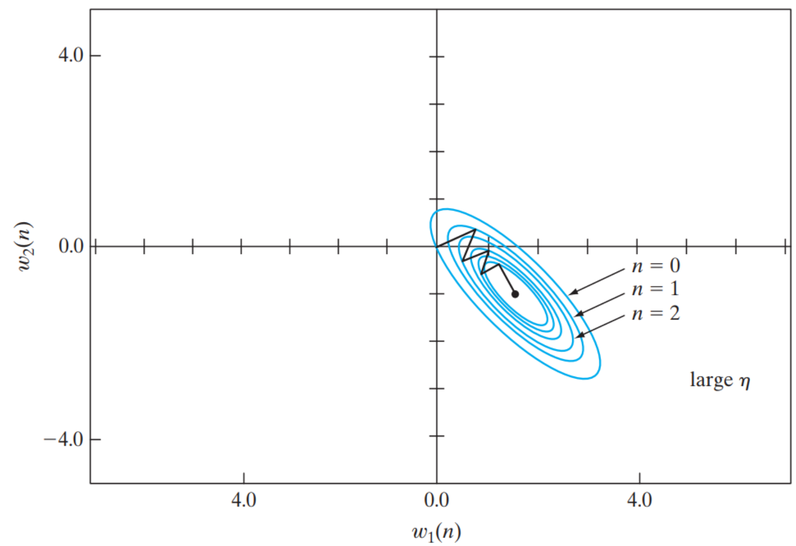
$$\mathbf{g}(n) = -\, e \times \mathbf{x}(n)$$

Método de Descida mais íngrime



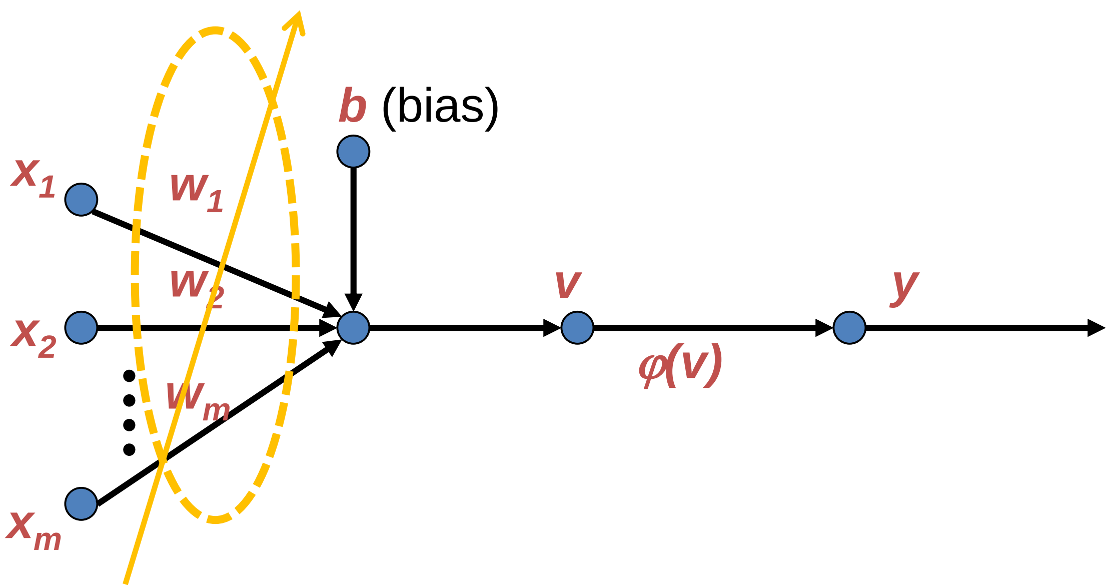
(a)

Quando η é pequeno, a resposta transitória do algoritmo é *sobreamortecida*, sendo que a trajetória traçada por **w**(n) segue um caminho suave no plano W, como ilustrado na Fig. 3.2 a.



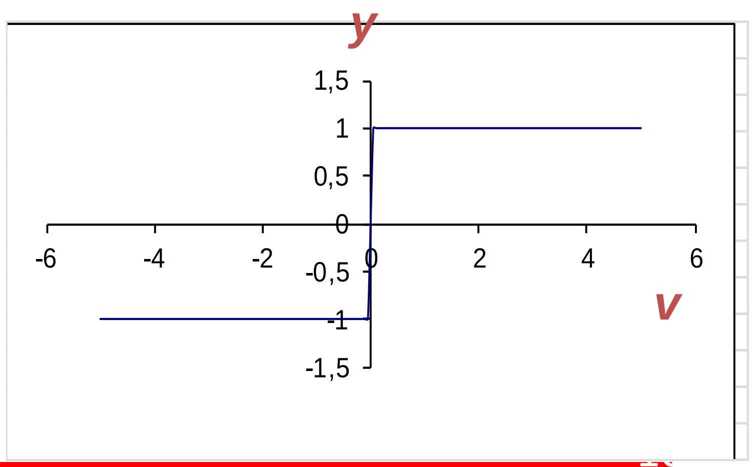
Quando η é grande, a resposta transitória do algoritmo é *subamortecida*, sendo que a trajetória de **w**(n) segue um caminho ziguezagueante (oscilatório), como ilustrado na Fig. 3.2 b.

Perceptron: Modelando o Neurônio



- φ é a função sinal:

$$\varphi(v) = \begin{cases} +1 & \text{IF } v >= 0 \\ -1 & \text{IF } v < 0 \end{cases}$$



O Ajuste sináptico durante o aprendizado

- Quando ocorre erro no reconhecimento / tratamento de uma entrada, um ajuste sináptico é necessário.
- O ajuste sináptico representa o aprendizado em cada neurônio do fato apresentado
- O ajuste sináptico procura corrigir os pesos de modo que se produza a saída desejada diante da respectiva entrada.
- Esse cálculo visa somar ao peso atual, um valor que corresponda a quantidade de erro gerada pela rede, e desta forma corrigir o valor do peso.
- O conhecimento dos neurônios reside nos pesos sinápticos.

Exemplo de ajuste sináptico

Entrada X ⟷ Saída Y (y_i)

resultado esperado (d_i)

Algoritmo simplificado:

- Aplicar entrada X
- Comparar saída Y com a saída desejada (associada a X, d)
- Se saída Y estiver errada então (diferente de d)
    - Calcular ajuste em pesos
- Aplicar nova entrada

Perceptron: Algoritmo de Aprendizado

- Variáveis e parâmetros

$\mathbf{x}(n)$ = vetor de entrada / vetor padrão

$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$

$\mathbf{w}(n)$ = vetor de peso

$= [b(n), w_1(n), w_2(n), \ldots, w_m(n)]^T$

b(n) = bias / polarização

y(n) = resposta atual (ou saída calculada)

d(n) = resposta desejada (faz parte do conjunto de treinamento)

η = taxa de aprendizado ($0 < \eta < 1$)

O algoritmo de aprendizado completo

- Inicialização: set $\mathbf{w}(0) = 0$
- Ativação: perceptron será ativado aplicando um exemplo de entrada (vector $\mathbf{x}(n)$ e resposta desejada $d(n)$)
- Calcula a resposta atual do perceptron:

$$y(n) = \mathrm{sgn}[\mathbf{w}^T(n) \times \mathbf{x}(n)]$$

- Adaptação do vetor peso: se $d(n)$ e $y(n)$ são diferentes

  $e(n) = [d(n)-y(n)]$

  $$\mathbf{w}(n + 1) = \mathbf{w}(n) + \eta \times e \times \mathbf{x}(n) \quad \text{// Eq. Aprendizado}$$

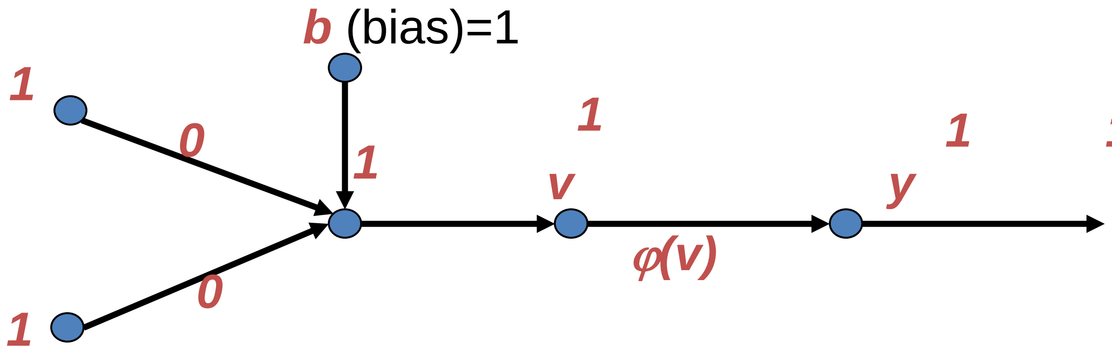
- Continuação: incrementa o passo n e volte ao passo de Ativação

Exemplo

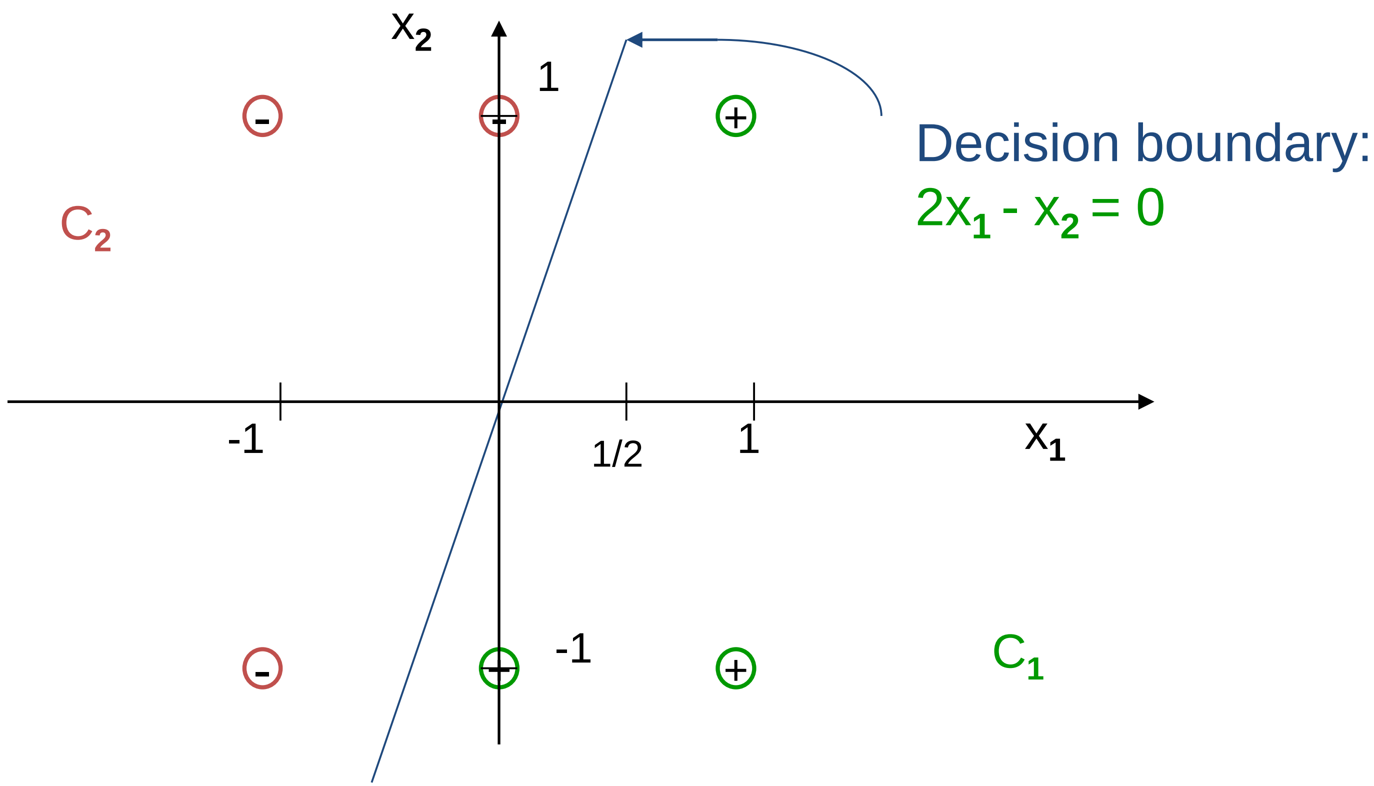
Considere o conjunto de aprendizado C_1 **e** C_2, onde:
$C_1 = \{(1,1), (1, -1), (0, -1)\}$ elementos da classe 1
$C_2 = \{(-1,-1), (-1,1), (0,1)\}$ elementos da classe 0

Use o algotitmo aprendizado do perceptron para classificar estes exemplos.

- $\mathbf{w}(0) = [0, 0]^T$ $\eta = 0{,}5$ $\mathbf{w}(\text{bias}) = [1]^T$



Exemplo



Exercício

- Usando a base de dados da Tabela abaixo, execute uma época para o treinamento do Perceptron, considerado o algoritmo de aprendizado visto em aula. Considere para esse treinamento, os seguintes parâmetros iniciais: $\boldsymbol{w} = [-1\ 0\ 0]^T$, sendo que $w(1)=-1$ é o peso do bias, bias = 1 e eta = $\eta=0,5$.

Após o treinamento, faça:

- Interpretação geométrica do Perceptron.
- Teste da rede com os valores de entrada (x_1 e x_2): 0,5 e 0,5. Qual a classe deste novo objeto?

x_1	x_2	d
0	0	0
0	1	1
1	0	1
1	1	1

Prática para o Perceptron

- Chamaremos as práticas como sendo aplicações do Perceptron em aplicações com duas diferentes naturezas de saída desejada:
  - Discreta: usada para problemas de classificação de dados.
    - Chamaremos esta como sendo prática 1.
    - Aqui utilizaremos o notebook: perceptron_discreto_notebook
  - Contínua: para aplicações cuja saída desejada tem a natureza contínua como é o caso de regressão para problemas de previsão de séries temporais.
    - Chamaremos esta de prática 2 e
    - usaremos o notebook anterior como ponto de partida para solução.

Práticas

- Para ambos os casos seguiremos a mesma metodologia:
  - Discutiremos todos os passos da solução até o exercício
  - Os problemas serão didáticos e de caso real
  - Como desafio, apresenta-se o case com datasets benchmarking com a finalidade de aprender a metodologia de uso de rede neural em problema de classificação e regressão.
  - Estes desafios não contabilizam nota. Servirá de exemplo para ilustrar a metodologia da disciplina.

BigMAAp

Laboratório de BIG DATA e Métodos Analíticos Aplicados

Prof. Dr. Leandro Augusto da Silva

leandroaugusto.silva@mackenzie.br

Faculdade de Computação e Informática

Programa de Pós-Graduação em Engenharia Elétrica e Computação