

*Universidade Federal do Rio Grande do Sul*

*Escola de Engenharia*

*Programa de Pós-graduação em Engenharia Elétrica*

*Laboratório de Instrumentação Eletro-Eletrônica & Biomédica*



*Inteligência Computacional I*

*Apresentador Convidado: Dr. Vinícius Horn Cene*

*NA 15 - Extreme Learning Machines (ELM)*

# TÓPICOS ABORDADOS

MOTIVAÇÃO: METODOS ITERATIVOS X NÃO-ITERATIVOS

ELM (EXTREME LEARNING MACHINES)

RESOLUÇÃO POR MÍNIMOS QUADRADOS

SVD (SINGULAR VALUE DECOMPOSITION)

DECOMPOSIÇÃO MATRICIAL

APLICAÇÕES DO SVD

O SVD EM PROCESSAMENTO DE IMAGENS

ELM - RESOLUÇÃO DO SISTEMA

MATRIZ PSEUDOINVERSA

REGULARIZAÇÃO

VALOR DE ARG MAX

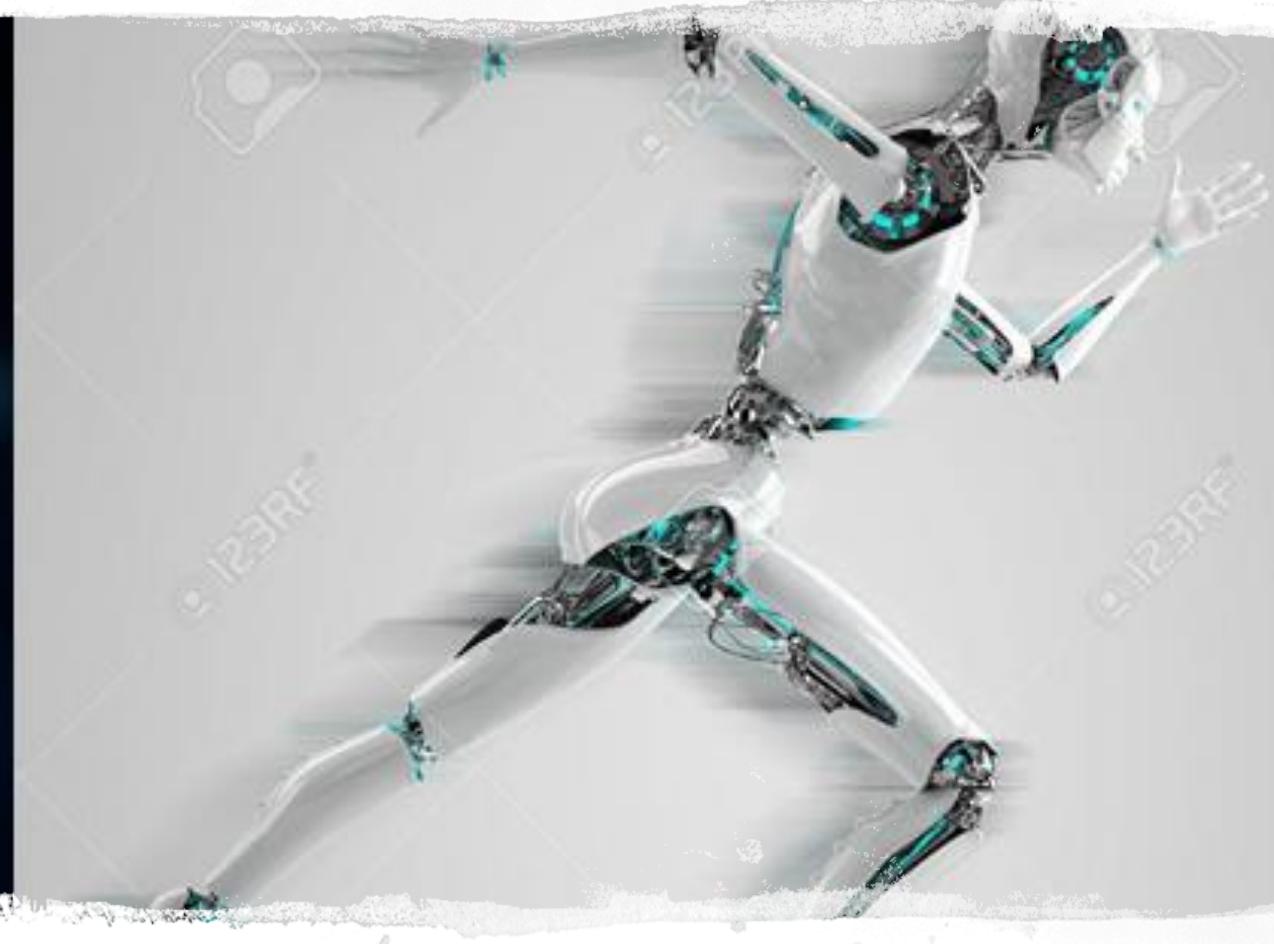
CLASSIFICAÇÃO

ELM PASSO A PASSO

ELM DE ARG MAX SUAVIZADO

EXERCÍCIOS DE FIXAÇÃO



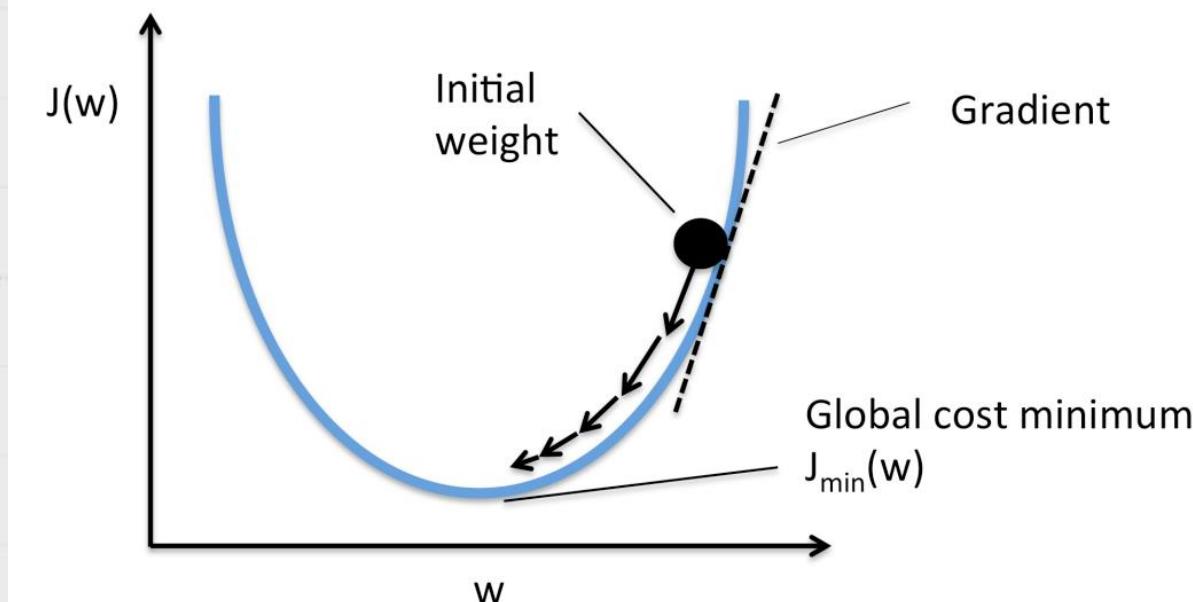
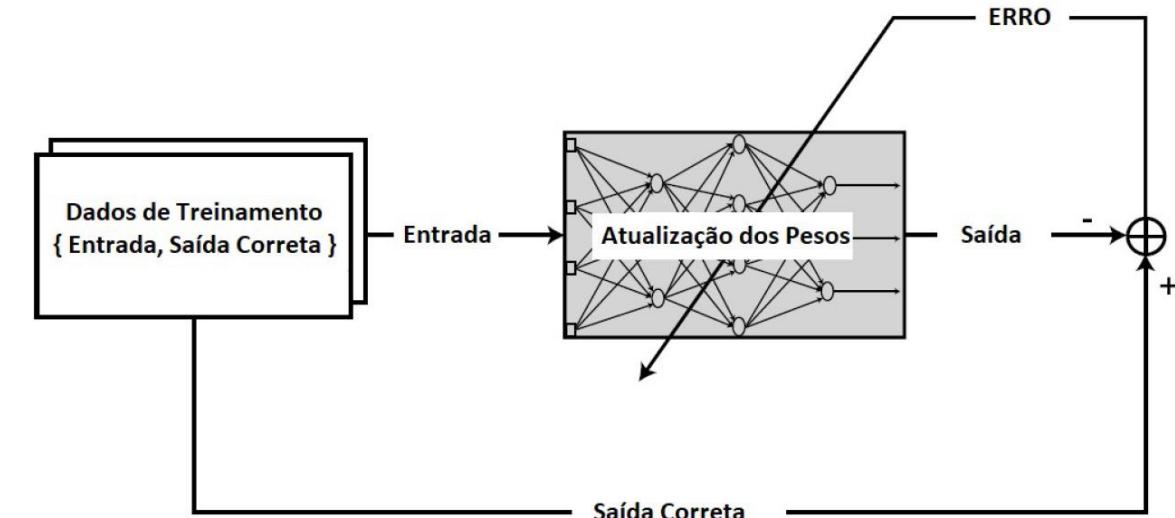
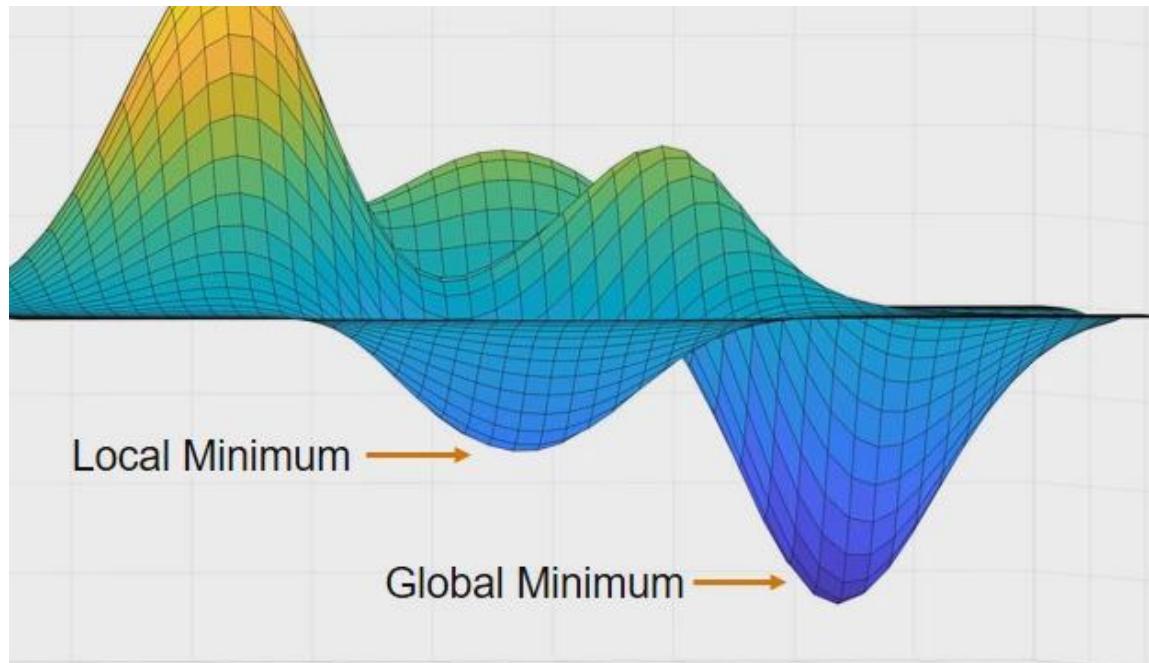


# MÉTODOS ITERATIVOS E NÃO-ITERATIVOS



# MÉTODOS ITERATIVOS

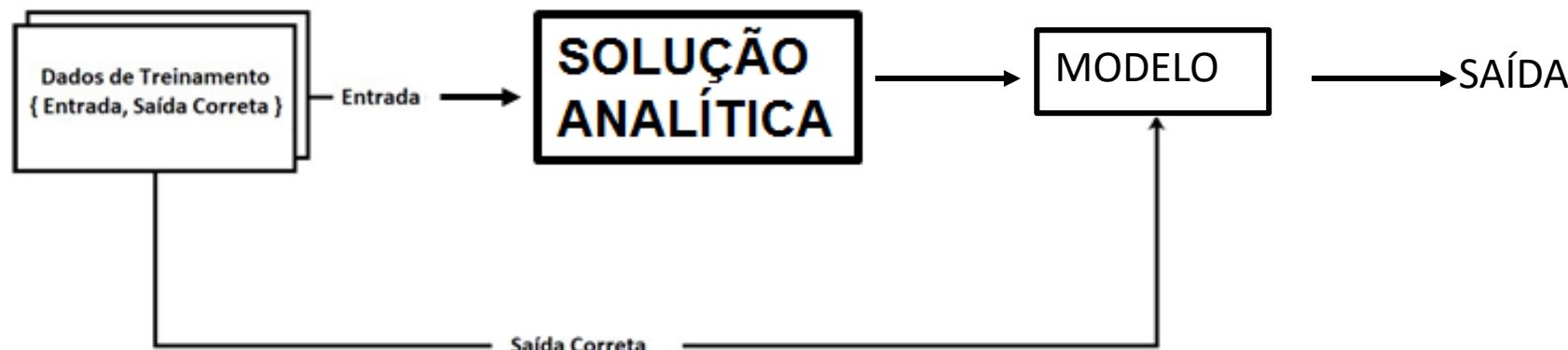
- MODELOS TRADICIONAIS: OTIMIZAÇÃO
- PROCESSO CUSTOSO
- BASEADO EM HEURÍSTICAS
- HIPERPARAMETROS:
  - CRITÉRIOS DE PARADA
  - LAYERS
  - MÉTODO DE OTIMIZAÇÃO



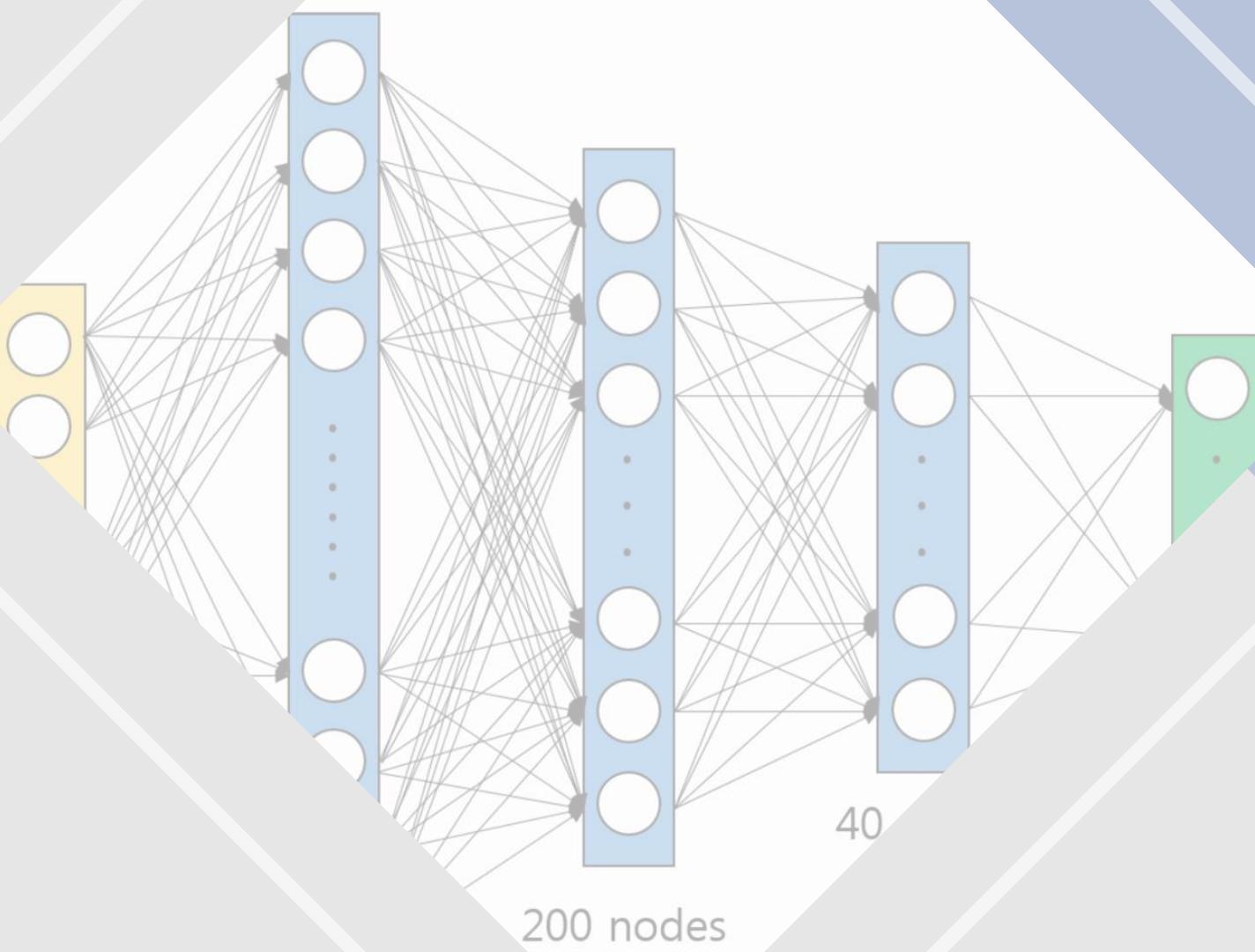


# MÉTODOS NÃO-ITERATIVOS

- NÃO UTILIZAM MÉTODOS DE OTIMIZAÇÃO ITERATIVA
- RESOLUÇÃO ANALÍTICA DO SISTEMA
- MODELO INDEPENDENTE DAS ENTRADAS
- 1 HIPERPARAMETRO (NÚMERO DE *HIDDEN NEURONS*)
- VELOZ:
  - GANHO DE 5 (MLP) – 6 (SVM) ORDENS DE MAGNITUDE
  - REQUER GRANDE QUANTIDADE DE MEMÓRIA PARA OPERAÇÕES MATRICIAIS (PROPORCIONAL AO NÚMERO DE *HN*)

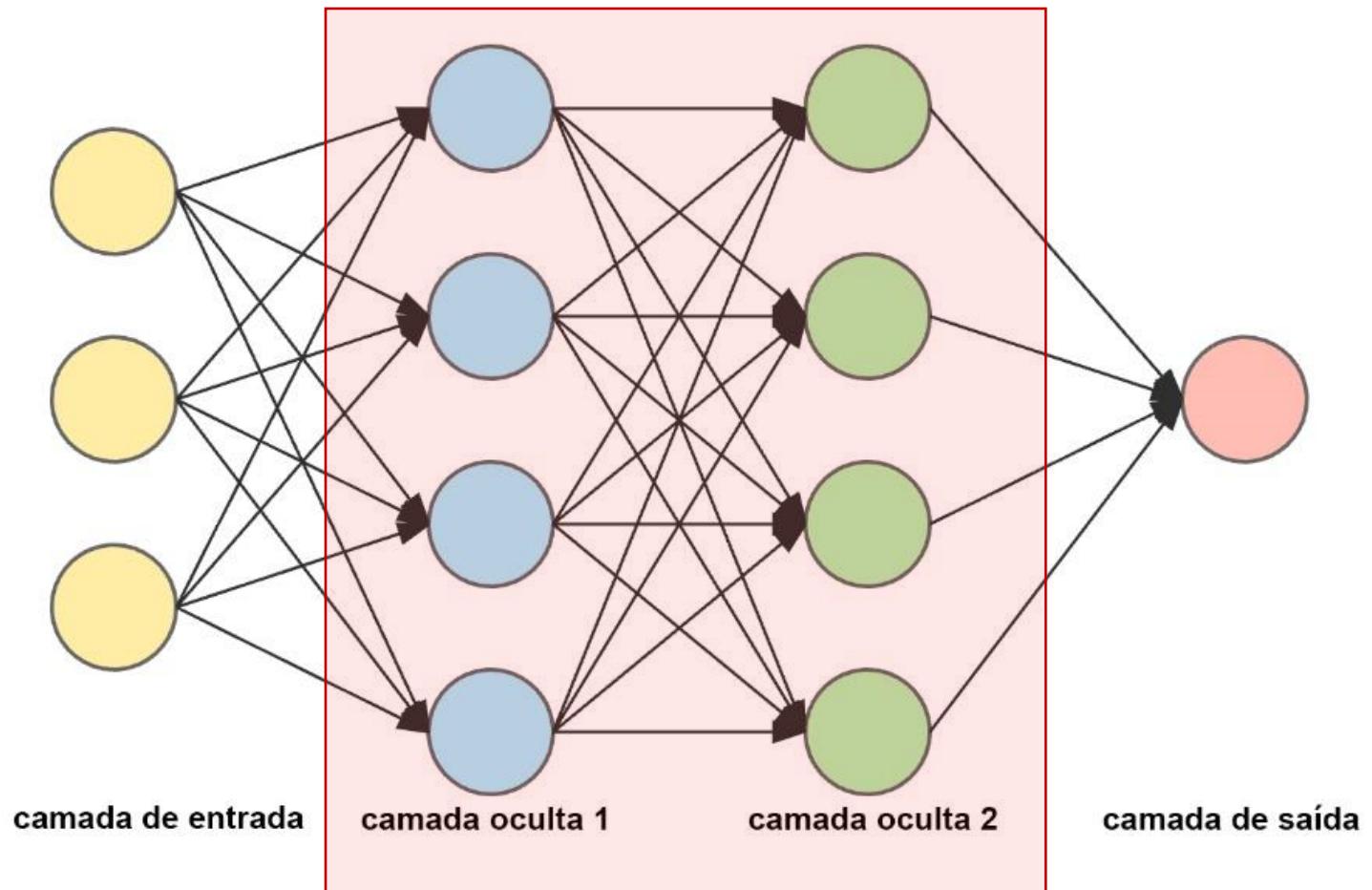


# *COMPARAÇÃO ENTRE DIFERENTES ARQUITETURAS DE REDES NEURAIS*



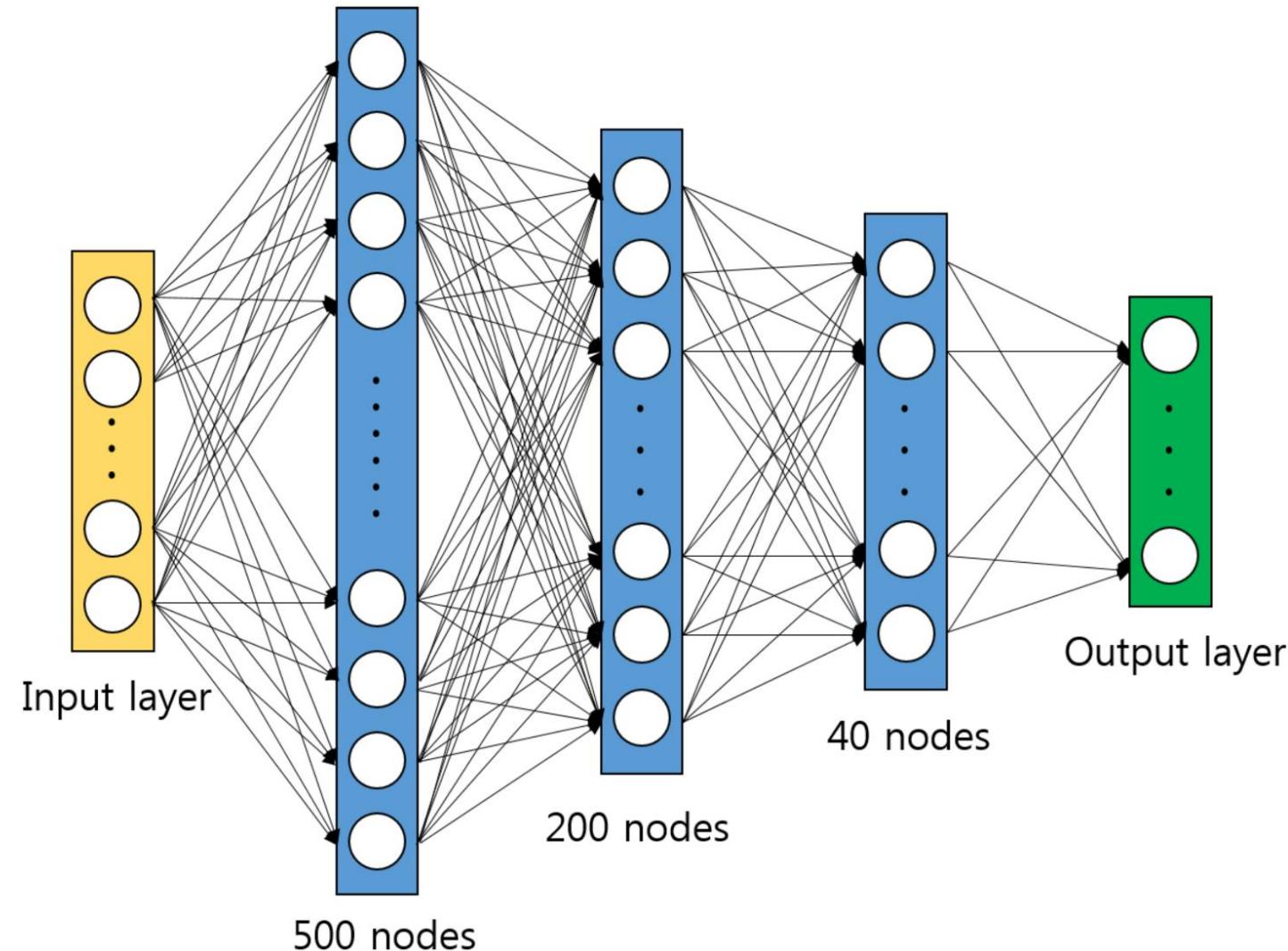
# MULTIPLE LAYER PERCEPTRON

COMPARAÇÃO ENTRE  
SINGLE LAYER  
FEEDFORWARD NEURAL  
NETWORKS (SLFN) E  
OUTRAS  
ARQUITETURAS



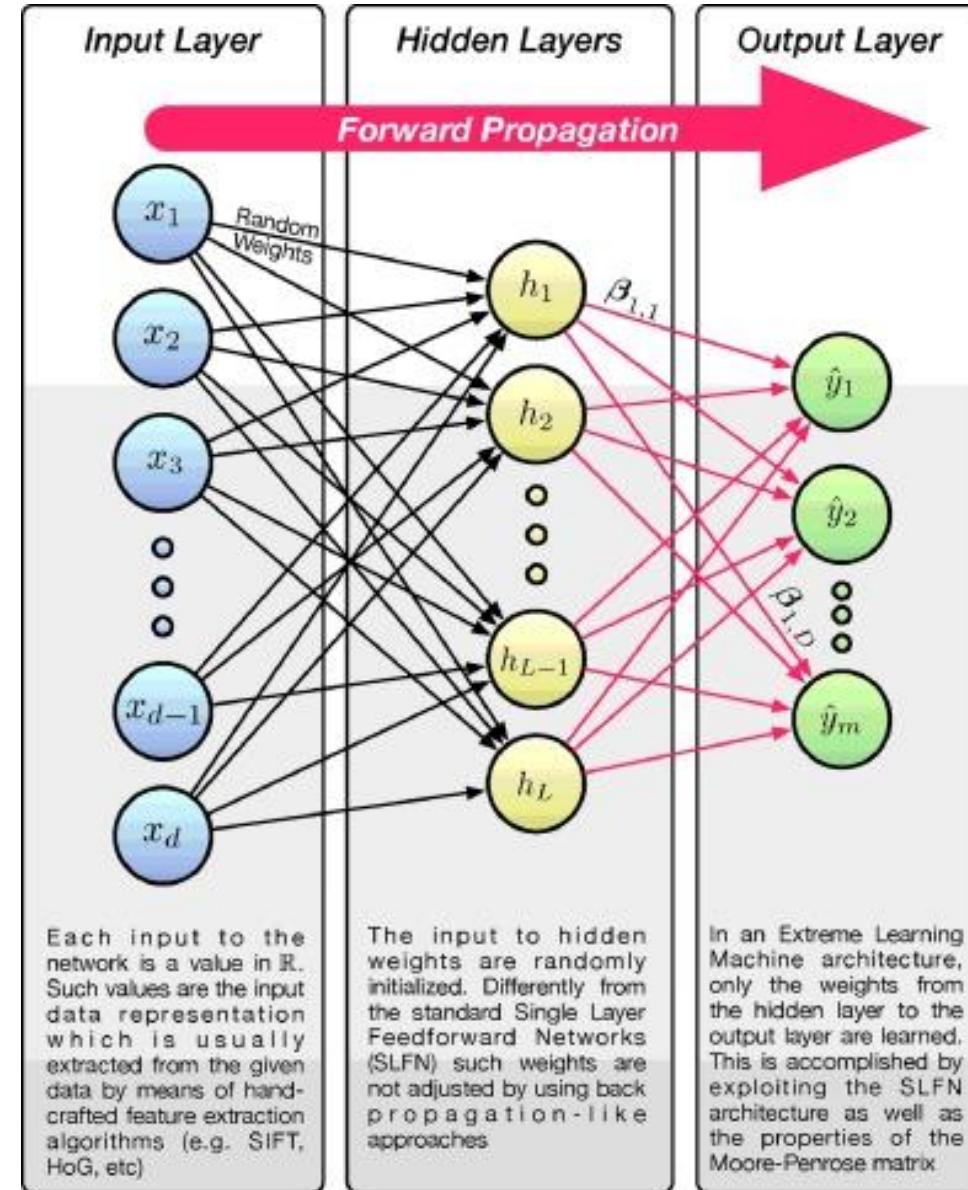
# DEEP NEURAL NETWORK

COMPARAÇÃO ENTRE  
*SINGLE LAYER*  
*FEEDFORWARD NEURAL*  
*NETWORKS (SLFN) E*  
OUTRAS  
ARQUITETURAS



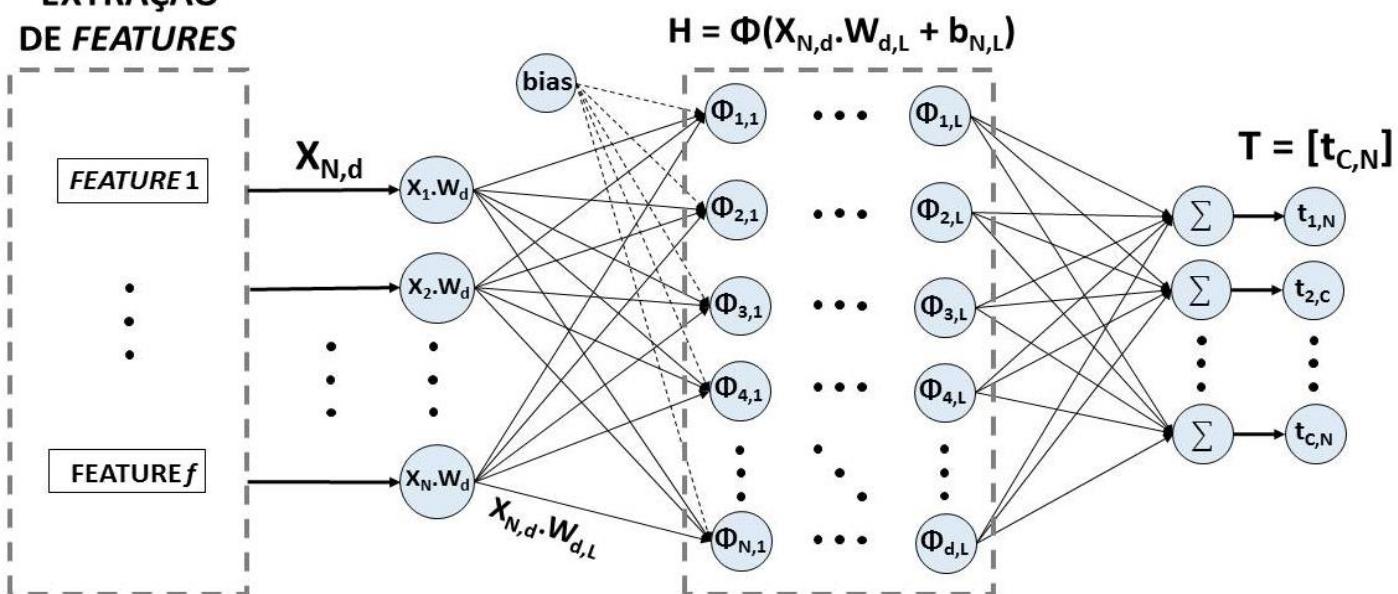
# SINGLE LAYER FEEDFORWARD NEURAL NETWORKS

## COMPARAÇÃO ENTRE SINGLE LAYER FEEDFORWARD NEURAL NETWORKS (SLFN) E OUTRAS ARQUITETURAS



# EXTREME LEARNING MACHINES (ELM)

EXTRAÇÃO  
DE FEATURES



CLASSIFICADOR EMERGENTE NA  
ÁREA DE **BIG DATA**;

REDE NEURAL SUPERVISIONADA  
DE TOPOLOGIA *SINGLE LAYER  
FEEDFORWARD NETWORK (SLFN)*;

CAMADA ÚNICA DE NEURÔNIOS  
**HIDDEN**;

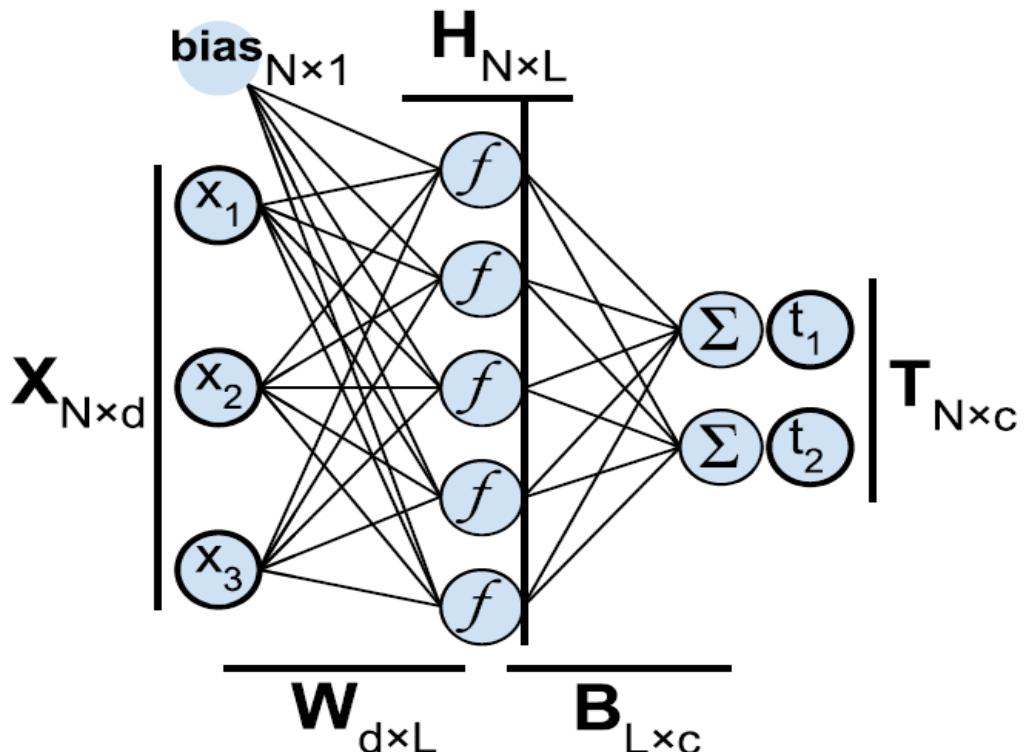
DEFINIÇÃO ALEATÓRIA DE  
VALORES PARA A CAMADA DE  
ENTRADA E NEURÔNIOS **HIDDEN**;

NÃO UTILIZA OTIMIZAÇÃO;

GANHO EM VELOCIDADE NA  
CRIAÇÃO E TESTE DO MODELO;

# ELM – PARÂMETROS DO MODELO

$$\mathbf{H} = \begin{bmatrix} \phi(\mathbf{w}_1 \mathbf{x}_1 + b_1) & \cdots & \phi(\mathbf{w}_L \mathbf{x}_1 + b_L) \\ \vdots & \ddots & \vdots \\ \phi(\mathbf{w}_1 \mathbf{x}_N + b_1) & \cdots & \phi(\mathbf{w}_L \mathbf{x}_N + b_L) \end{bmatrix},$$
$$\boldsymbol{\beta} = (\beta_1^T \cdots \beta_L^T)^T, \quad \mathbf{T} = (\mathbf{y}_1^T \cdots \mathbf{y}_N^T)^T.$$



- sendo:
  - $\mathbf{x}$ : vetor de entradas
  - $\mathbf{w}$ : vetor de pesos de entrada
  - $\mathbf{b}$ : matriz de bias
  - $\Phi$ : kernel
  - $d$ : # features

$N$ : n° de amostras

$L$ : n° de neurônios hidden

$\boldsymbol{\beta}$ : Modelo

$\mathbf{T}$ : Labels

$\mathbf{H}$ : Neurônios  
hidden

Transformada  
Linear

$H\boldsymbol{\beta} = \mathbf{T}$

# RESOLUÇÃO DO SISTEMA – MÍNIMOS QUADRADOS

## DIFERENTES SISTEMAS:

- Sub-determinados ( $N < L$ )
- Determinados ( $N = L$ )
- Sobre-determinados ( $N > L$ )

## Sub-determinados (infinitas soluções):

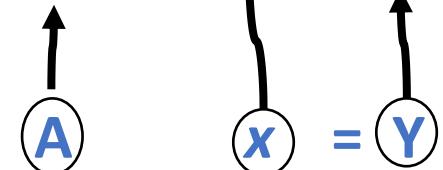
$$\begin{aligned} a_1X_1 + b_1X_2 + c_1X_3 &= Y_1 \\ a_2X_1 + b_2X_2 + c_2X_3 &= Y_2 \end{aligned} \rightarrow \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}$$

## Determinados (inversa existe):

$$\begin{aligned} a_1X_1 + b_1X_2 &= Y_1 \\ a_2X_1 + b_2X_2 &= Y_2 \end{aligned} \rightarrow \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}$$

## Sobre-determinados (zero soluções):

$$\begin{aligned} a_1X_1 + b_1X_2 &= Y_1 \\ a_2X_1 + b_2X_2 &= Y_2 \\ a_3X_1 + b_3X_2 &= Y_3 \end{aligned} \rightarrow \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix}$$



# RESOLUÇÃO DO SISTEMA – MÍNIMOS QUADRADOS

## Linear Regression

$$\{(X_1, y_1), (X_2, y_2), \dots, (X_n, y_n)\}$$

Least Squares Criteria

$$\widehat{\beta}_0 = \bar{y} - \widehat{\beta}_1 \bar{X}$$
$$\widehat{\beta}_1 = \frac{\sum_{i=1}^n X_i y_i - n \bar{X} \bar{y}}{\sum_{i=1}^n X_i^2 - n \bar{X}^2}$$
$$\hat{y} = \widehat{\beta}_0 + \widehat{\beta}_1 X$$

## Multiple Regression

General Parametric Equation:

$$y = f(\underline{X}) + \epsilon$$

Depends on Statistical Method

$$f(\underline{X}) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$$

$$\hat{y} = \widehat{\beta}_0 + \widehat{\beta}_1 X_1 + \dots + \widehat{\beta}_p X_p$$

# RESOLUÇÃO DO SISTEMA – MÍNIMOS QUADRADOS

## Multiple Regression

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

$$\mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} y_1 - \hat{y}_1 \\ y_2 - \hat{y}_2 \\ y_3 - \hat{y}_3 \\ \vdots \\ \vdots \\ y_n - \hat{y}_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \vdots \\ \hat{y}_n \end{bmatrix} = \mathbf{y} - \hat{\mathbf{y}}$$

$$RSS = \sum_{i=1}^n e_i^2 \quad \longrightarrow \quad RSS = \mathbf{e}^T \mathbf{e}$$

$$RSS = \mathbf{e}^T \mathbf{e}$$

$$RSS = (\mathbf{y} - \hat{\mathbf{y}})^T (\mathbf{y} - \hat{\mathbf{y}})$$

$$RSS = (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})^T (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})$$

$$RSS = (\mathbf{y}^T - \hat{\boldsymbol{\beta}}^T \mathbf{X}^T)(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})$$

$$RSS = \mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}^T \mathbf{X}^T \mathbf{y} + \hat{\boldsymbol{\beta}}^T \mathbf{X}^T \mathbf{X}\hat{\boldsymbol{\beta}}$$

# Multiple Regression

$$RSS = \mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{X} \hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}} \mathbf{X}^T \mathbf{y} + \hat{\boldsymbol{\beta}}^T \mathbf{X}^T \mathbf{X} \hat{\boldsymbol{\beta}}$$

## Matrix Differentiation

$x = m \times 1$  matrix

$A = n \times m$  matrix;  $A \perp x$

$$\mathbf{y} = A \rightarrow \frac{\delta \mathbf{y}}{\delta x} = \mathbf{0}$$

$$\mathbf{y} = Ax \rightarrow \frac{\delta \mathbf{y}}{\delta x} = A$$

$$\mathbf{y} = xA \rightarrow \frac{\delta \mathbf{y}}{\delta x} = A^T$$

$$\mathbf{y} = x^T Ax \rightarrow \frac{\delta \mathbf{y}}{\delta x} = 2x^T A$$

$$\frac{\delta(RSS)}{\delta \hat{\boldsymbol{\beta}}} = \frac{\delta(\mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{X} \hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}} \mathbf{X}^T \mathbf{y} + \hat{\boldsymbol{\beta}}^T \mathbf{X}^T \mathbf{X} \hat{\boldsymbol{\beta}})}{\delta \hat{\boldsymbol{\beta}}} = 0$$

$$\frac{\delta(\mathbf{y}^T \mathbf{y})}{\delta \hat{\boldsymbol{\beta}}} - \frac{\delta(\mathbf{y}^T \mathbf{X} \hat{\boldsymbol{\beta}})}{\delta \hat{\boldsymbol{\beta}}} - \frac{\delta(\hat{\boldsymbol{\beta}} \mathbf{X}^T \mathbf{y})}{\delta \hat{\boldsymbol{\beta}}} + \frac{\delta(\hat{\boldsymbol{\beta}}^T \mathbf{X}^T \mathbf{X} \hat{\boldsymbol{\beta}})}{\delta \hat{\boldsymbol{\beta}}} = 0$$

$$0 - \mathbf{y}^T \mathbf{X} - (\mathbf{X}^T \mathbf{y})^T + 2\hat{\boldsymbol{\beta}}^T \mathbf{X}^T \mathbf{X} = 0$$

$$0 - \mathbf{y}^T \mathbf{X} - \mathbf{y}^T \mathbf{X} + 2\hat{\boldsymbol{\beta}}^T \mathbf{X}^T \mathbf{X} = 0$$

$$2\hat{\boldsymbol{\beta}}^T \mathbf{X}^T \mathbf{X} = 2\mathbf{y}^T \mathbf{X} \quad \hat{\boldsymbol{\beta}}^T \mathbf{X}^T \mathbf{X} = \mathbf{y}^T \mathbf{X}$$

$$\hat{\boldsymbol{\beta}}^T = \mathbf{y}^T \mathbf{X} (\mathbf{X}^T \mathbf{X})^{-1} \quad \underline{\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}}$$

# Multiple Regression

$$\{ (X_1, y_1), (X_2, y_2), \dots, (X_n, y_n) \}$$



Least Squares Criteria

$$\hat{\beta} = (X^T X)^{-1} X^T y \xrightarrow{\text{SVD}} \text{PSEUDOINVERSA}$$



$$\hat{y} = \widehat{f(X)} = X\hat{\beta}$$

# SINGULAR VALUE DECOMPOSITION (SVD)

$$A = U D V^T$$

Left singular vectors

Singular values

Right singular vectors

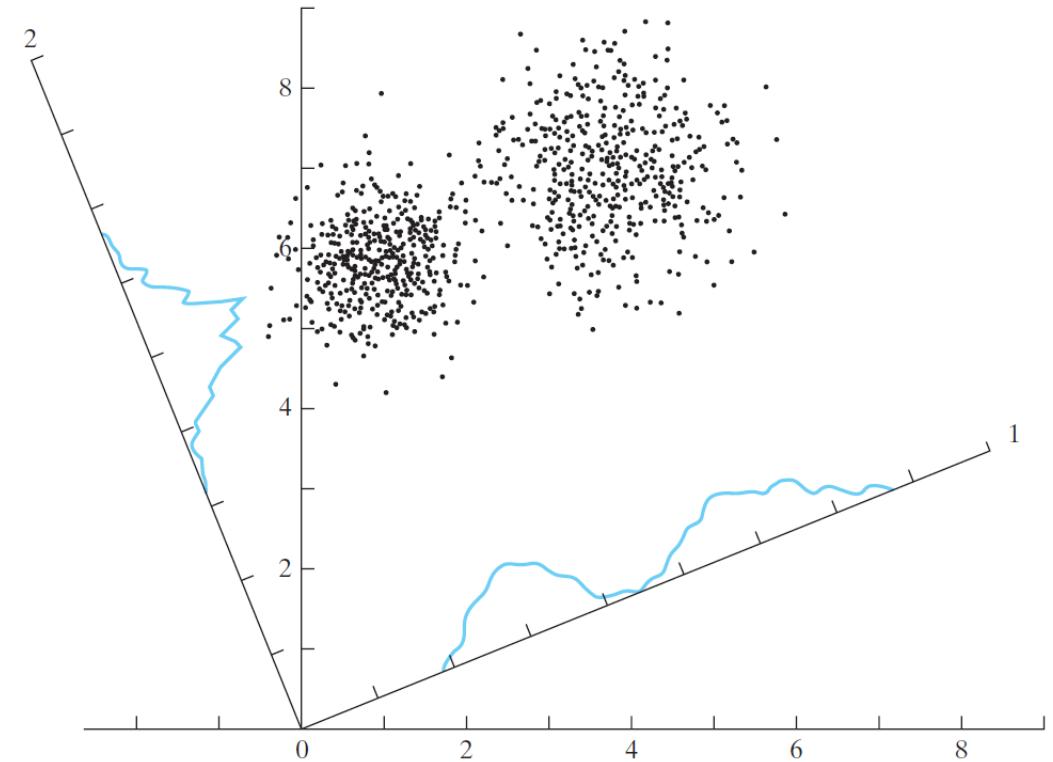
## IDENTIDADES:

$$A = U \Sigma V^T$$

$$A^T = V \Sigma U^T$$

$$A^T A = V \Sigma (U^T U) \Sigma V^T = V \Sigma^2 V^T$$

$$A A^T = U \Sigma (V^T V) \Sigma U^T = U \Sigma^2 U^T$$



## CÁLCULO PARÂMETROS:

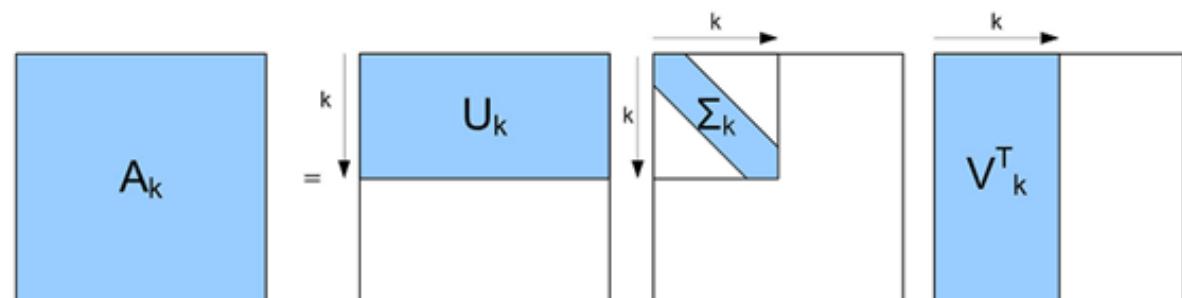
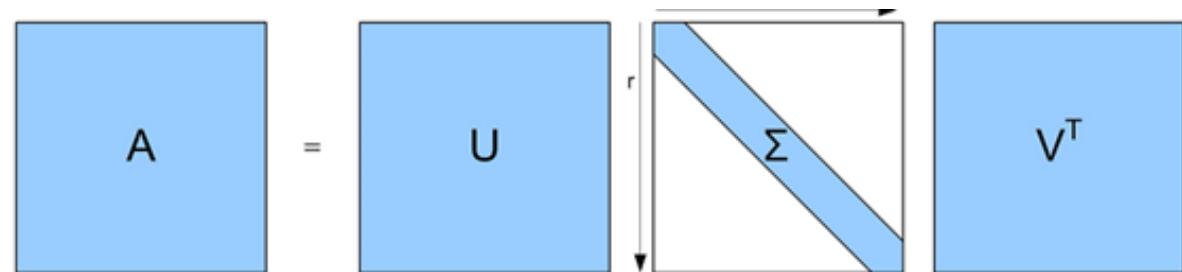
$\Sigma$  = Raíz dos autovalores de  $A A^T$

$U, V$  = Autovetores de  $A A^T$  e  $A^T A$

# SINGULAR VALUE DECOMPOSITION (SVD)

## TEOREMA DE ECKART-YOUNG:

Dado o SVD de uma matriz  $A_{m,n}$  existe uma aproximação  $k$  que minimiza o erro da diagonal ( $L^2$  ou Frobenius Norm)



$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \dots + \sigma_k u_k v_k^T$$

# SINGULAR VALUE DECOMPOSITION (SVD)

PODE-SE DIMINUIR O TAMANHO DE UMA IMAGEM, POR EXEMPLO, EM FUNÇÃO DO NÚMERO DE VALORES SINGULARES UTILIZADOS (RANK DA MATRIZ)

The Singular Value Decomposition

$$A = U \Sigma V^T$$

$m \times n$

$m \times m$

$m \times n$

$n \times n$

$r =$  the rank of  $A$   
 $=$  number of linearly independent  
columns/rows

# SINGULAR VALUE DECOMPOSITION (SVD)

## EXEMPLO 1: PROCESSAMENTO DIGITAL DE IMAGENS (PDI)

- Redução Do Número De Valores Singulares (Rank)
- Pode Ser Inclusive Utilizado Como Filtro
- Métricas utilizadas:

$$\text{Norma } L_2 = \frac{\|A - A_k\|_2}{\|A\|_2} = \frac{\sigma_{k+1}}{\sigma_1}$$

$$RMSE = \sqrt{\frac{\sum_{m,n} (A_{ij} - A_{kij})^2}{mn}}$$

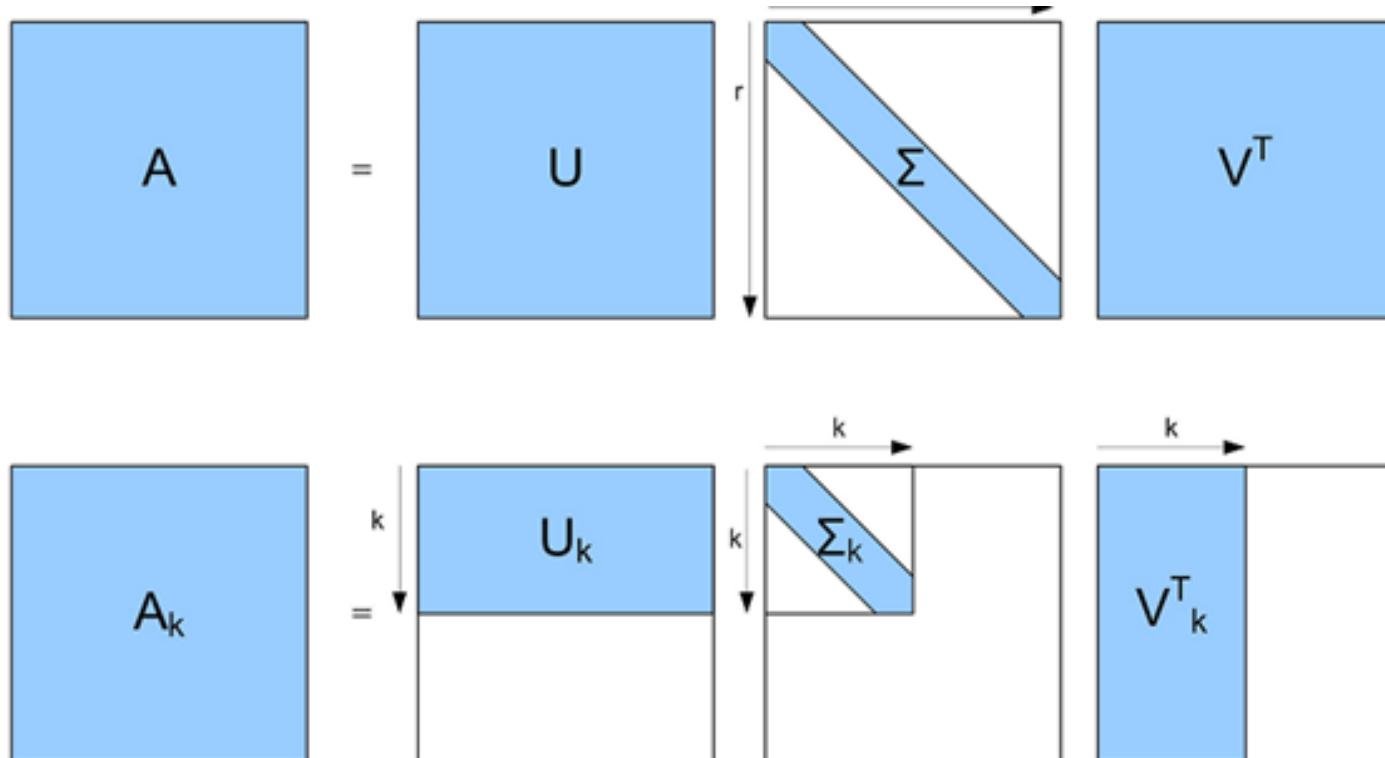
$$CR = \frac{mn}{k(m + n + 1)}$$

ONDE:

CR = *Compression Ratio*

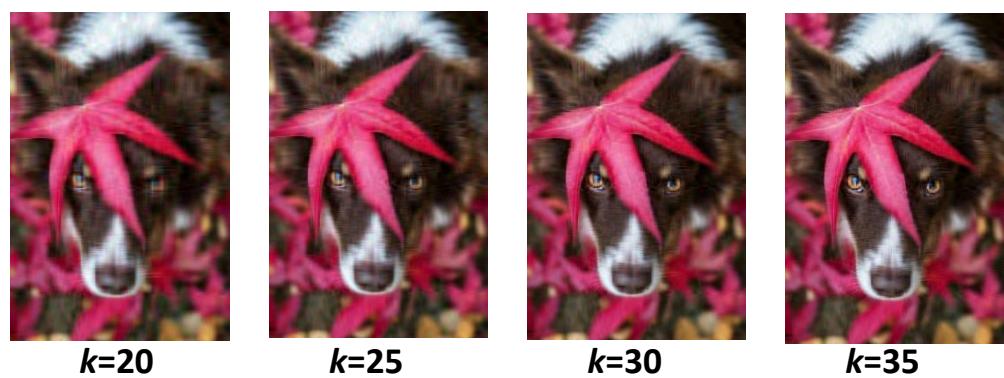
m = # linhas; n = # colunas

k varia de 1 ao total de valores Singulares > 0



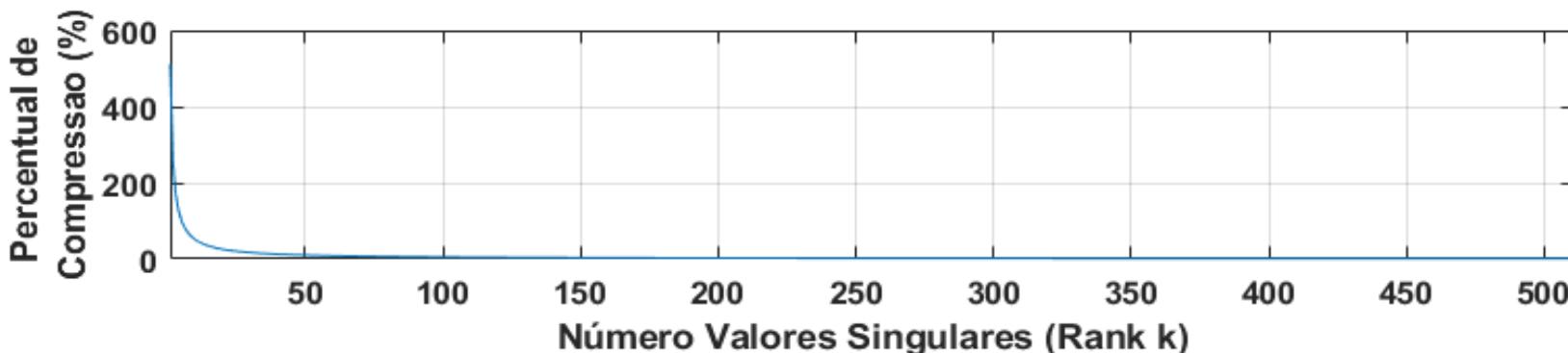
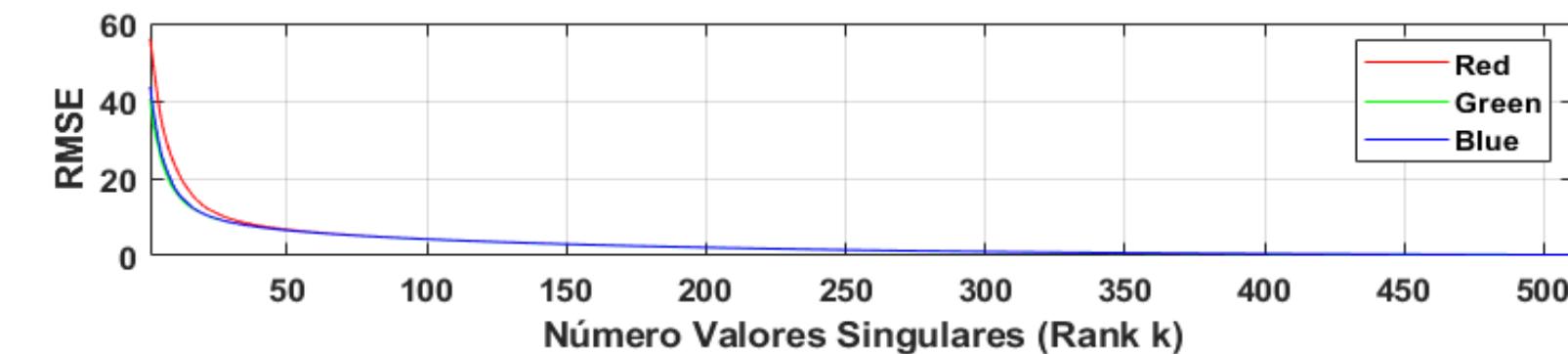
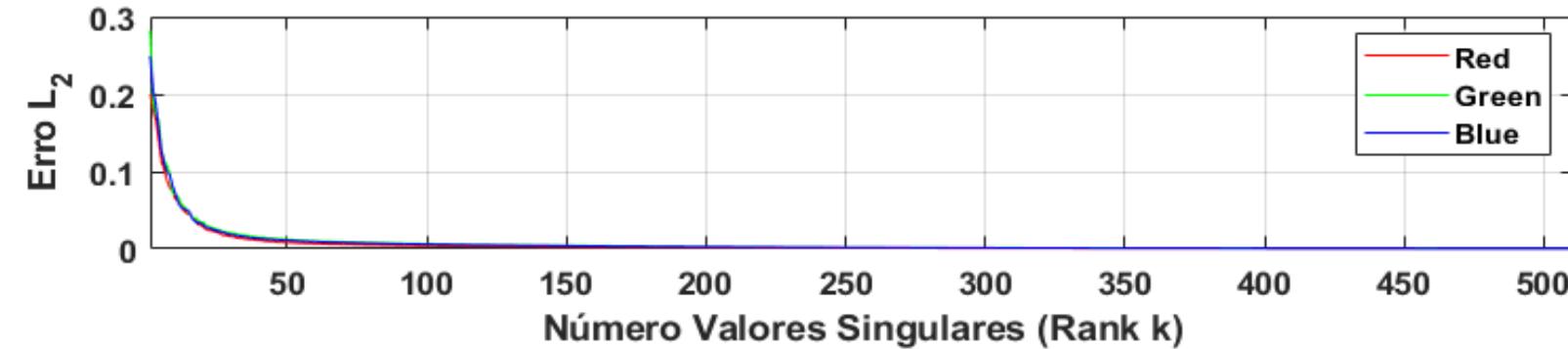
# SINGULAR VALUE DECOMPOSITION (SVD)

## EXEMPLO 1: PROCESSAMENTO DIGITAL DE IMAGENS (PDI)



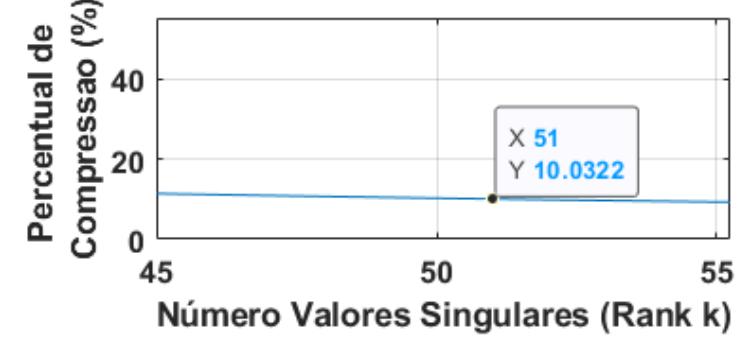
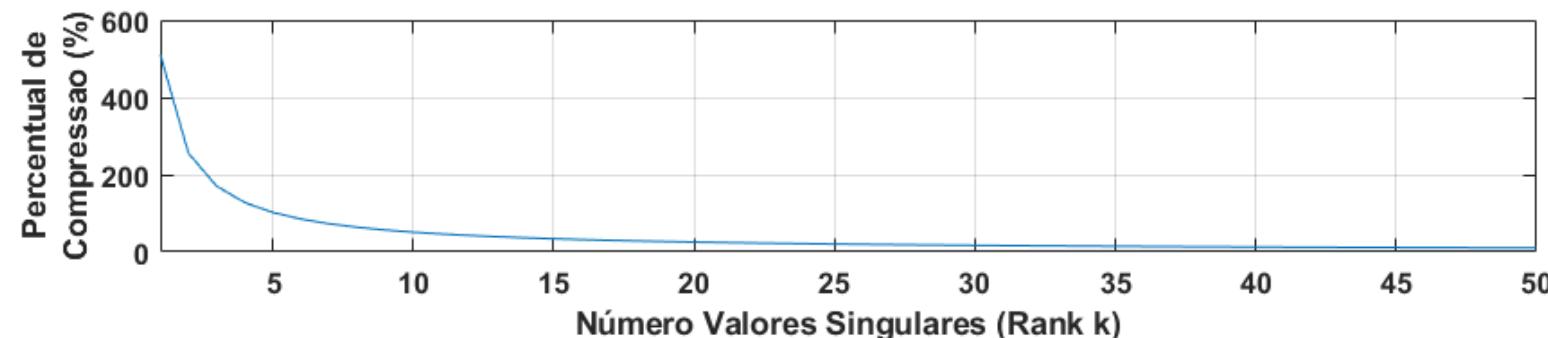
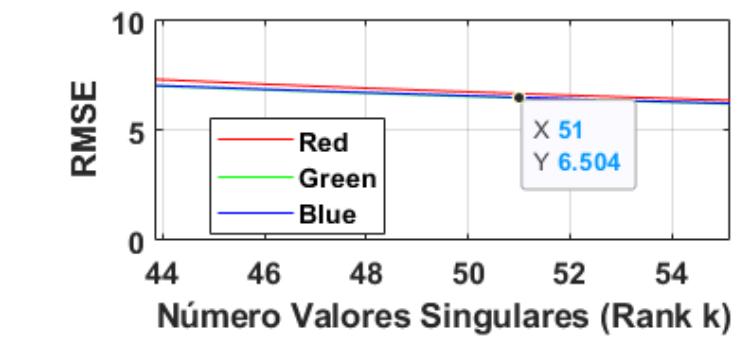
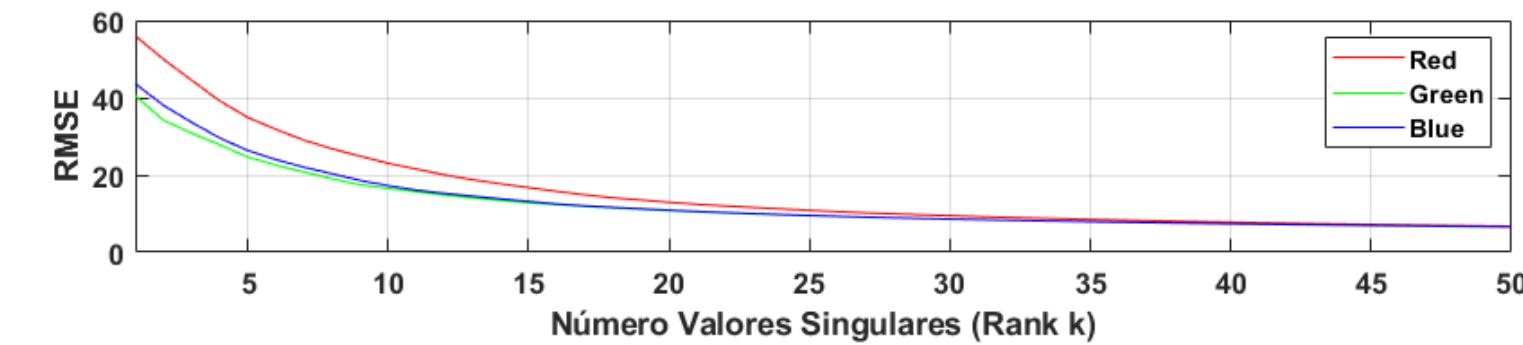
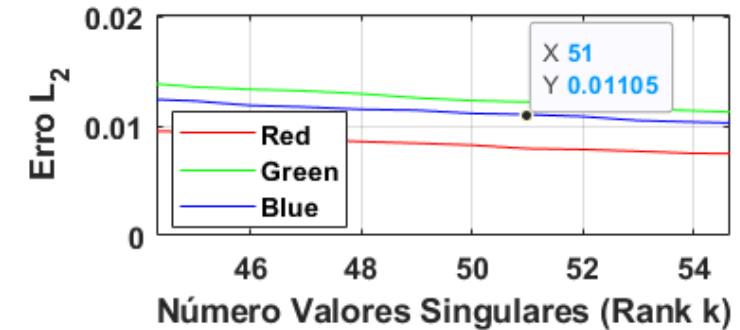
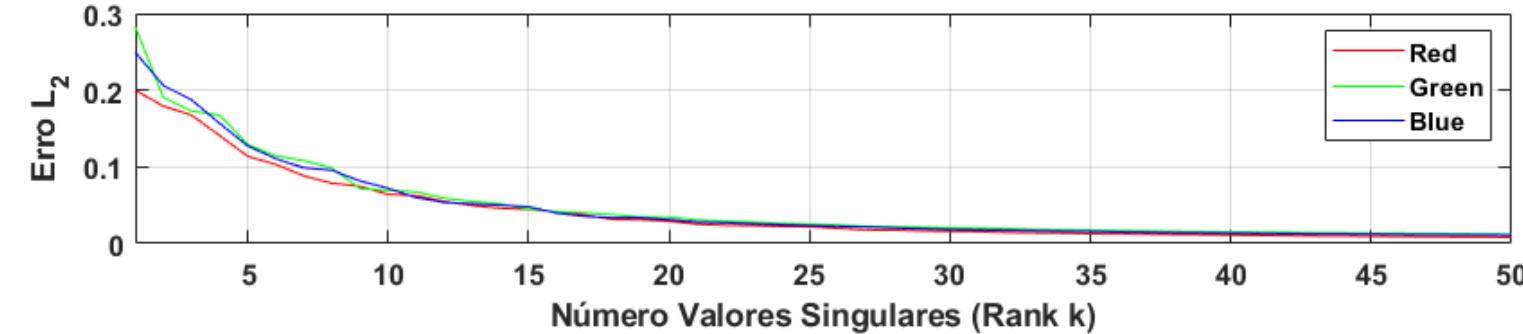
# SINGULAR VALUE DECOMPOSITION (SVD)

## EXEMPLO 1: PROCESSAMENTO DIGITAL DE IMAGENS (PDI)



# SINGULAR VALUE DECOMPOSITION (SVD)

## EXEMPLO 1: PROCESSAMENTO DIGITAL DE IMAGENS (PDI)



# SINGULAR VALUE DECOMPOSITION (SVD)

O SVD POSSUI OUTRAS APLICAÇÕES INTERESSANTES COMO SELEÇÃO DE FEATURES OU SISTEMAS DE RECOMENDAÇÃO:

Item x subject matrix  
(ISM)

	S1	S2	S3	S4	S5
dog	1	1	1	1	1
cat	1	1	0	1	0
cow	0	0	1	0	1
lion	0	0	1	1	0
tiger	1	1	0	0	1

Singular decomposition analysis (SVD)

$$C_{m \times n} = U_{m \times r} \times \Sigma_{r \times r} \times V'_{r \times n}$$



Reducing dimensions from  $r$  to  $k$

$$\tilde{C}_{m \times n} = U_{m \times k} \times \Sigma_{k \times k} \times V'_{k \times n}$$



**RECOMENDAÇÃO: PESQUISAR O USO DO SVD COMO RECOMMENDATION SYSTEM:**

<https://www.youtube.com/watch?v=E8aMcwmqsTg>

<https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75>

<https://pt.coursera.org/lecture/machine-learning-applications-big-data/recsys-svd-i-bakO5>

# PSEUDOINVERSA DE MOORE-PENROSE:

## ROGER PENROSE:

Físico, matemático e filósofo científico inglês. Prêmio Nobel de física em 2020 juntamente a Reinhard Genzel e Andrea Ghez por sua pesquisa relacionada a formação de buracos negros.

Consolidou a teoria da matriz inversa generalizada, comumente chamada de Matriz Pseudoinversa de Moore-Penrose.

$$Ax = Y$$

$$x = (A^T A)^{-1} A^T Y$$

$$x = A^\dagger Y$$

## UNINDO SUA TEORIA AO SVD:

$$A = U \Sigma^T V \therefore A^\dagger = V \Sigma^{-1} U^T$$

$$x = V \Sigma^{-1} U^T Y$$



# SINGULAR VALUE DECOMPOSITION (SVD)

**EXEMPLO 1: CONSIDERANDO A MATRIZ  $A = [1, 2]$ :**

- i) Utilize SVD para a decomposição da matriz  $A$  em  $[U, \Sigma, V]$ ;
- ii) Compare os resultados com a função *svd* do Matlab;
- iii) Com as matrizes  $[U, \Sigma, V]$ , calcule a pseudoinversa  $(A^T)$  de  $A$ ;
- iv) Compare o resultado com a função *pinv* do Matlab;
- v) Confirme as identidades a seguir considerando  $B = A^T$ :

$$B \cdot A = I_n$$

$$A \cdot B \cdot A = A$$

$$B \cdot A \cdot B = B$$

$$(AB)^T = AB$$

$$(BA)^T = BA$$

- vi) Para uma entrada  $x$  no espaço nulo ( $N(A)$ ) de  $A$ , calcule  $A^T Ax$ ;
- vii) Para um  $x$  no espaço de colunas ( $C(A)$ ) de  $A$ , calcule  $A^T Ax$ ;

# *SINGULAR VALUE DECOMPOSITION (SVD)*

**EXEMPLO 1: CONSIDERANDO A MATRIZ  $A = [1, 2]$ :**

- i) Utilize SVD para a decomposição da matriz  $A$  em  $[U, \Sigma, V]$ ;

```
%% DEFINE MATRIZ A
A = [1 2]

%% i) Utilize SVD para a decomposição da matriz A em [U, s, V];
% Calculando Componentes U, S e V:
% COMO A TEM TAMANHO 1x2, S TERÁ O MESMO TAMANHO
% Base da pseudoinversa que é uma matriz quadrada que possibilita as operações
% SVD(A) = U*S*V'

Base = A.*A'
```

# SINGULAR VALUE DECOMPOSITION (SVD)

## EXEMPLO 1: CONSIDERANDO A MATRIZ $A = [1, 2]$ :

- i) Utilize SVD para a decomposição da matriz  $A$  em  $[U, \Sigma, V]$ ;

% S terá dimensão 1x2 e que  $U$   $V$  são bases ortonormais e matrizes quadradas  
% Logo, para possibilitar  $U^*S^*V'$ ,  $U$  é 1x1 e  $V$  é 2x2  
% O calculo de  $S$  é um problema de auto valor da Matriz Base, então  
% Cauculo de  $U$  e  $V$  é um prolema de Auto Vetor (substitui valores de  $S$  em  $M$ )

```
U = 1                                % Como U tem que ser ortonormal e é 1x1, U=1
[V, S] = eig(Base)                    % Calcula Auto Valores e Vetores para A.A'
                                         % Realinhando S em ordem decrescente
S_aux = [S(2,2), S(2,1); ...          % Singular Value = Raíz Quadrada(Auto Valor)
         S(1,2), S(1,1)]
s = sqrt(S_aux)
                                         % O que nos obriga a realinhar V também
V_aux = [V(:,2), V(:,1)]
```



U =	S =	V =
1	2.2361	0      0.4472    -0.8944
	0      0	0.8944    0.4472

# SINGULAR VALUE DECOMPOSITION (SVD)

**EXEMPLO 1: CONSIDERANDO A MATRIZ  $A = [1, 2]$ :**

ii) Compare os resultados com a função *svd* do Matlab;

```
% ii) Compare os resultados com a função svd do Matlab;  
% Temos então nossa matriz  $A \cdot A'$  decomposta em  $[U, S, V]$ ,  
% conferindo coma função do Malab, temos:
```

```
[U_, s_, V_] = svd(A)
```

```
% Notem que a diferença numérica é relacionada com questões  
% numéricas dos métodos, sendo os valores iguais, na prática.
```

```
Diff_U = U - U_
```

```
Diff_S = s - s_
```

```
Diff_V = V - V_
```

```
% Reconstruindo A:
```

```
A_Matlab = U_*s_*V_'
```

```
A_SVD = U*s*V'
```



```
Diff_V =  
  
1.0e-15 *  
  
0.0555 -0.1110  
0.1110 -0.1665  
  
A_Matlab =  
  
1.0000 2.0000  
  
A_SVD =  
  
1 2  
0 0
```

# SINGULAR VALUE DECOMPOSITION (SVD)

```
%% iii) Com as matrizes [U, s, V], calcule a pseudoinversa (dagger) de A:  
% Sendo A = U*s*V; Adagger = V*([1./s(1) 0])'*U'  
% OBS: O que fazemos aqui é obter a pseudo inversa de s, o que é obtido  
%       invertendo cada valor de s ~= 0 e fazendo a transposta da matriz  
Adagger_SVD = V*([1./s(1) 0])'*U'
```

%% iv) Compare o resultado com a função pinv do Matlab:

```
% 10E-9 corresponde ao termo de relaxação utilizado pelo Matlab e refere-se  
% ao # de valores singulares utilizados na aproximação da matriz.
```

```
Adagger_Matlab = pinv(A)
```

```
Adagger_Matlab = pinv(A, 10E-9)
```



Adagger\_SVD =

0.2000

0.4000

Adagger\_Matlab =

0.2000

0.4000

Adagger\_Matlab =

0.2000

0.4000

# SINGULAR VALUE DECOMPOSITION (SVD)

%% v) Confirme as identidades a seguir considerando  $B = \text{dagger}$ :

$\text{Adagger} = \text{Adagger}_{\text{Matlab}}$

$M_{\text{Identidade}} = A^* \text{Adagger}$        $\circ = I$

$M_{\text{Original}} = A^* \text{Adagger} * A$        $\circ = A$

$\text{Pseudoinversa} = \text{Adagger} * A^* \text{Adagger}$        $\circ = \text{Adagger}$

$AB = (A^* \text{Adagger})'$        $\circ = A^* \text{Adagger}$

$BA = (\text{Adagger} * A)'$        $\circ = \text{Adagger} * A$



$M_{\text{Identidade}} =$	$ABt =$	$BAt =$		
1.0000	1.0000	0.2000	0.4000	
		0.4000	0.8000	
$M_{\text{Original}} =$	$AB =$	$BA =$		
1.0000	2.0000	1.0000	0.2000	0.4000
			0.4000	0.8000
$\text{Pseudoinversa} =$				
0.2000				
0.4000				

# SINGULAR VALUE DECOMPOSITION (SVD)

%% vi) Para uma entrada  $x$  no espaço nulo  $N(A)$  de  $A$ , calcule  $Adagger * A * x$ :

Os Vetores Singulares associados aos Valores Singulares nulos (0) representam os espaços nulos  $N(A)$  da matriz. De forma análoga, todos os Vetores Singulares não-nulos representam o Espaço de Colunas  $C(A)$  onde diversas soluções triviais podem ser encontradas, dependendo do número de Autovalores

% Considerando s:

s

% Percebe-se que só há um Autovalor definido positivo, como  $U$  é 1, o modelo % é projetado por  $V$ , que tem como descritor do Espaço Nulo a 2a Coluna:

$V(:,2)$  % Simplificadamente [-2; 1]

% Portanto, para qualquer constante  $k$  multiplicada por  $V(:,2)$  deve levar a % um resultado no  $N(A)$ . Abaixo alguns valores são testados:

$x = [-2; 1]$

```
for k=1:10
    N_A(:,k) = Adagger*A*x*(k-5); % testa valores de -5 < k < 5
end
```

% Como esperado, todos os resultados encontram-se em  $N(A)$

# SINGULAR VALUE DECOMPOSITION (SVD)

%% vi) Para uma entrada x no espaço nulo N(A) de A, calcule Adagger\*A\*x:

```
s =                               Vet_EspacoNulo =      x =
2.2361          0      -0.8944          -2
0          0      0.4472           1
```

```
for k=1:10
    N_A(:,k) = Adagger*A*x*(k-5);           % testa valores de -5 < k < 5
end
```

```
N_A =
0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0
```

% Como esperado, todos os resultados encontram-se em N(A)

# SINGULAR VALUE DECOMPOSITION (SVD)

%% vii) Para um  $x$  no espaço de colunas ( $C(A)$ ) de  $A$ , calcule  $\text{dagger}^*A^*x$ :

% Como  $s$  tem dimensão  $2 \times 2$  e a 2a coluna é nula (representando o espaço nulo)

% o descritor do Espaço Coluna encontra-se na 1a Coluna:

Vet\_EspacoColunas = V(:,1) % Simplificadamente [1; 2]

% Portanto, qualquer constante  $k$  multiplicada por  $V(:,1)$  deve levar a  
% recuperação do valor de  $x$ . Abaixo alguns valores são testados:

$x = [1; 2]$

```
for k=1:10
    N_C(:,k) = Adagger*A*x*(k-5); % testa valores de -5 < k < 5
end
```

% Como esperado, todos os resultados são  $x$  escalonado proporcionalmente.

Vet\_EspacoColunas =

0.4472  
0.8944

N\_C =

-4.0000	-3.0000	-2.0000	-1.0000	0	1.0000	2.0000	3.0000	4.0000	5.0000
-8.0000	-6.0000	-4.0000	-2.0000	0	2.0000	4.0000	6.0000	8.0000	10.0000

# SINGULAR VALUE DECOMPOSITION (SVD)

Reparam que esta é a utilidade da utilização da pseudoinversa:

- 1) Quando a inversa existe ela é obtida pela operação que chega em x;
- 2) Quando a inversa não existe ela pode ser aproximada em relação a x através dos vetores em combinação com os Valores Singulares;
- 3) Utilização com o SVD facilita muito a obtenção da solução completa.

s =

2.2361

0

Vet\_EspacoNulo =

0 -0.8944

0 0.4472

x =

-2

1

Vet\_EspacoColunas =

0.4472

0.8944

N\_A =

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

N\_C =

-4.0000	-3.0000	-2.0000	-1.0000	0	1.0000	2.0000	3.0000	4.0000	5.0000
-8.0000	-6.0000	-4.0000	-2.0000	0	2.0000	4.0000	6.0000	8.0000	10.0000

# *SINGULAR VALUE DECOMPOSITION (SVD)*

**RECOMENDO FORTEMENTE ESTAS VIDEO-AULAS SOBRE SVD E PSEUDOINVERSA:**

<https://www.youtube.com/watch?v=N74V4CBO0sk>

<https://www.youtube.com/watch?v=5bxsxM2UTb4>

<https://www.youtube.com/watch?v=J4JZDI05014>

<https://www.youtube.com/watch?v=pv400I6TpSM>

<https://www.youtube.com/watch?v=uQhTuRIWMxw>

[http://web.mit.edu/be.400/www/SVD/Singular\\_Value\\_Decomposition.htm](http://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm)

<https://in.mathworks.com/help/matlab/ref/pinv.html>

**RECOMENDO TAMBÉM ESTAS (EXCELENTES) AULAS DE ÁLGEBRA LINEAR:**

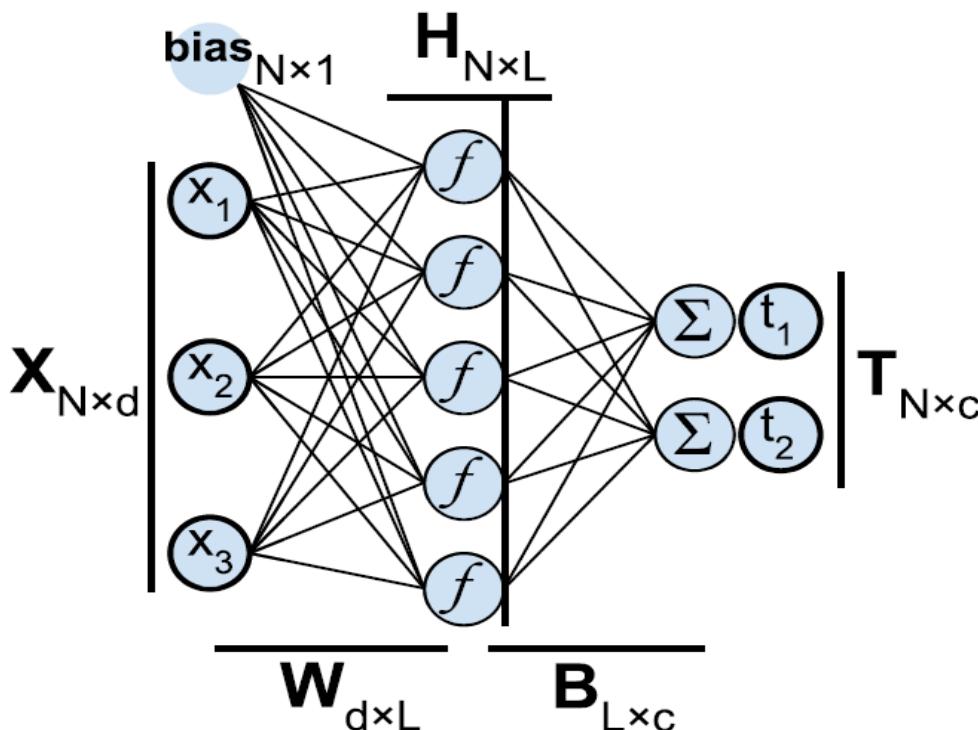
<https://ocw.mit.edu/resources/res-18-009-learn-differential-equations-up-close-with-gilbert-strang-and-cleve-moler-fall-2015/differential-equations-and-linear-algebra/>

# VOLTANDO AO ELM



# ELM – PARÂMETROS DO MODELO

$$\mathbf{H} = \begin{bmatrix} \phi(\mathbf{w}_1 \mathbf{x}_1 + b_1) & \cdots & \phi(\mathbf{w}_L \mathbf{x}_1 + b_L) \\ \vdots & \ddots & \vdots \\ \phi(\mathbf{w}_1 \mathbf{x}_N + b_1) & \cdots & \phi(\mathbf{w}_L \mathbf{x}_N + b_L) \end{bmatrix},$$
$$\boldsymbol{\beta} = (\beta_1^T \cdots \beta_L^T)^T, \quad \mathbf{T} = (\mathbf{y}_1^T \cdots \mathbf{y}_N^T)^T.$$



**SENDO:**

- $\mathbf{x}$ : vetor de entradas
- $\mathbf{w}$ : vetor de pesos de entrada
- $\mathbf{b}$ : matriz de bias
- $\Phi$ : kernel
- $d$ : # features

$H$ : Neurônios  
*hidden*

$N$ : n° de amostras

$L$ : n° de neurônios *hidden*

$\theta$ : Modelo

$T$ : Labels

Transformada  
Linear

$$H\theta = T$$

# ELM – DETERMINAÇÃO DO MODELO

- PARA A RESOLUÇÃO DO SISTEMA:

$$\mathbf{H}\beta = \mathbf{T}$$

$$\beta = \mathbf{H}^\dagger \mathbf{T}$$

$$\mathbf{H}^\dagger = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T$$

- Onde  $\mathbf{H}^\dagger$  é a matriz pseudoinversa generalizada (Moore-Penrose), que possibilita a resolução de sistemas sub e sobre-determinados:

$$\mathbf{H}^\dagger = (\mathbf{H}^T \mathbf{H} + \alpha \mathbf{I})^{-1} \mathbf{H}^T$$

- Onde  $\alpha$  é o termo regularizador que impede inconsistências quando  $\mathbf{H}^T \mathbf{H}$  é muito próximo a uma matriz singular.

# ELM – DETERMINAÇÃO DO MODELO

## RESOLUÇÃO DO SISTEMA ( $Ax = Y$ ):

- *Ridge-Regression:*

$$\min\{\|Y - Ax\|_1^2 + \lambda \|x\|_1^2\}$$

$$\beta^{RR} = (H^T H + \lambda I)^{-1} H^T T$$

Onde  $\lambda I$  é comumente representado por  $\frac{I}{C}$  sendo  $C$  o parâmetro de regularização do modelo.

- *Singular Value Decomposition (SVD):*

$$\begin{aligned} H &= U \Sigma V^T \\ H^\dagger &= V \Sigma^{-1} U^T \end{aligned} \longrightarrow \beta = H^\dagger T$$

- Forma econômica: limitação de coeficientes de vetores singulares para a aproximação da matriz desejada (*precisão x custo computacional*)

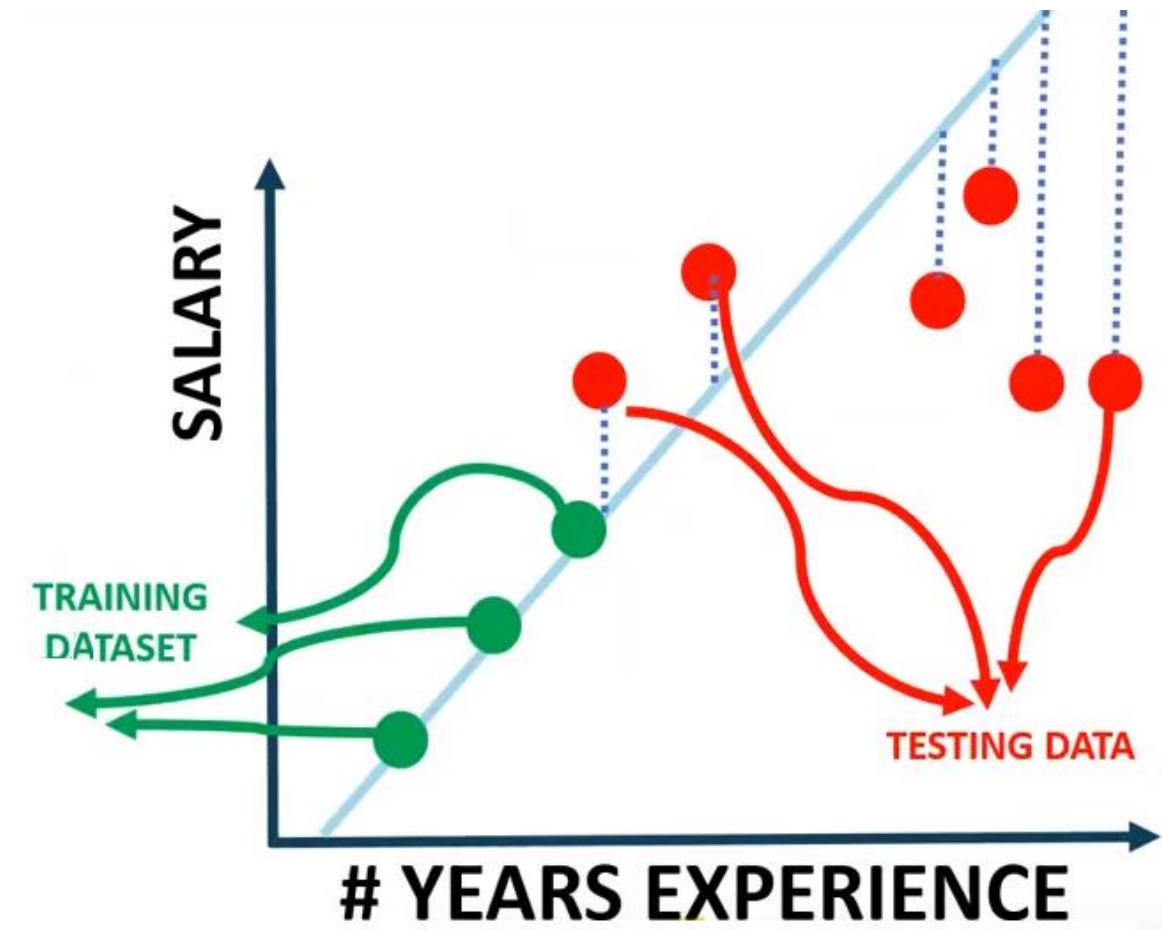
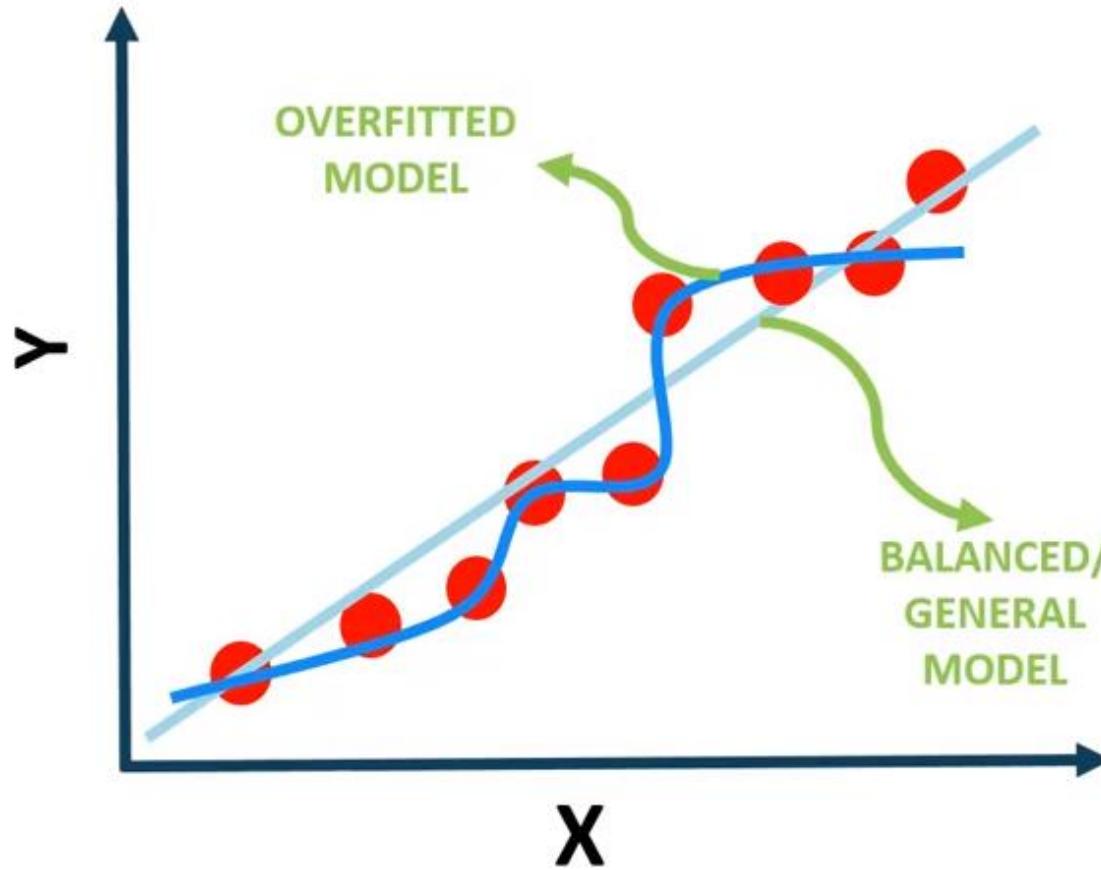
# ELM – CLASSIFICAÇÃO

**CLASSE DE SAÍDA =  $\text{argmax}(C_0, \dots, C_N)$ :**



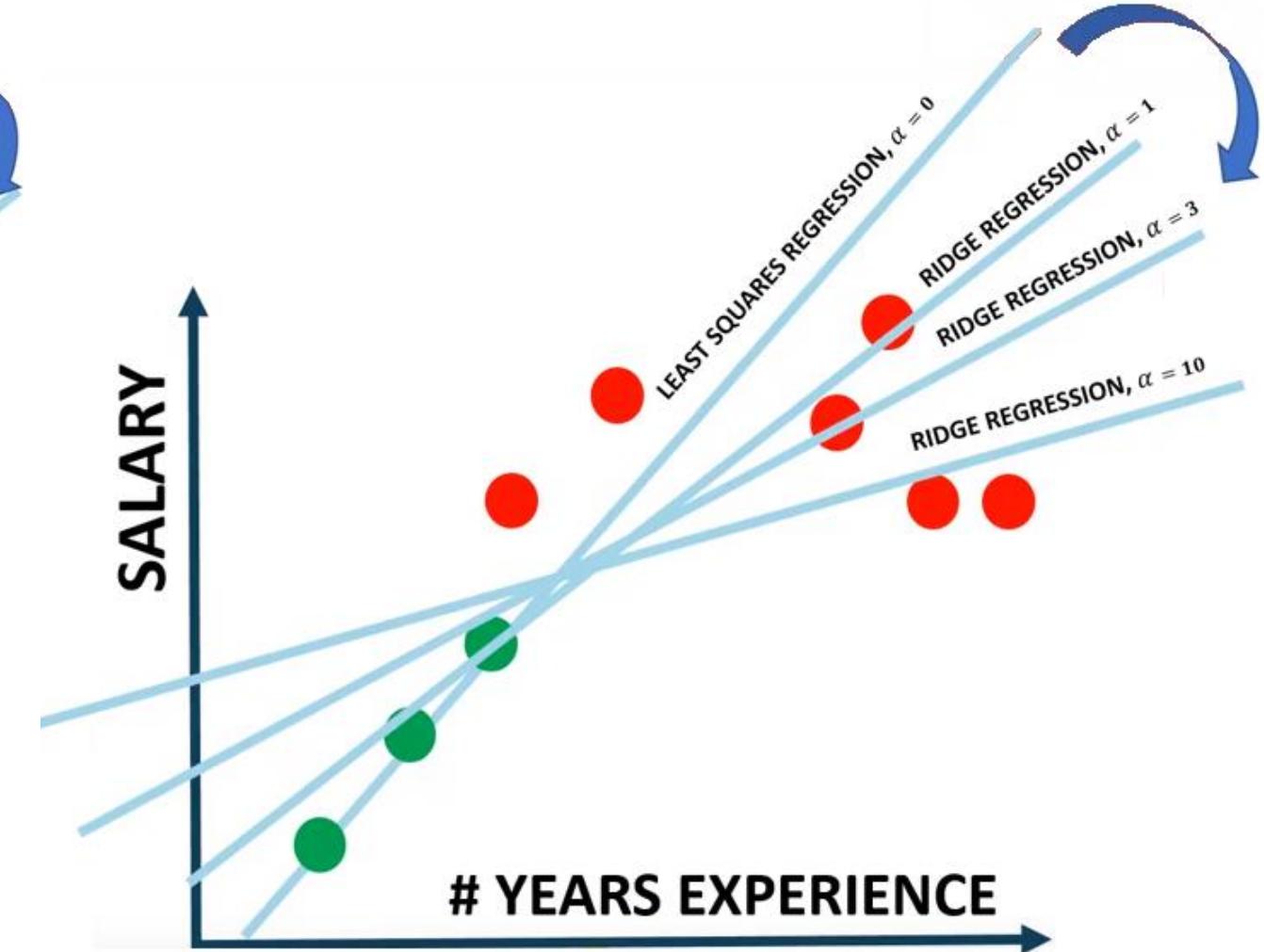
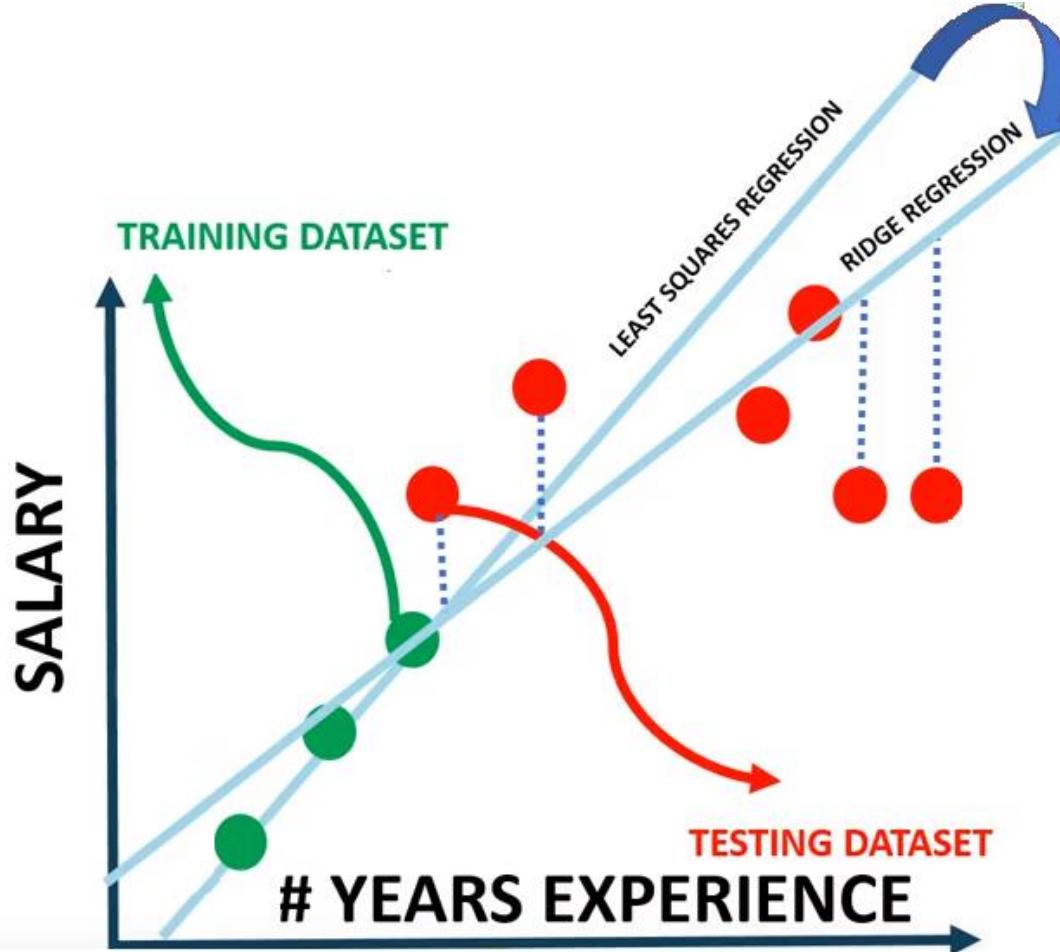
# REGULARIZAÇÃO

$$\hat{\beta} = (X^T X)^{-1} X^T y$$



# REGULARIZAÇÃO

$$\hat{x} = (A^T A + \lambda I)^{-1} A^T \times Y$$



# ELM PASSO A PASSO

**ENTRADA E SAÍDA DE DADOS PADRÃO DO ELM**

**DEFINIÇÃO DE HIPERPARÂMETROS**

**TREINAMENTO:**

DEFININDO A ENTRADA DE DADOS

DEFININDO A MATRIZ DE *Bias*

DEFININDO A MATRIZ *H*

UTILIZAÇÃO DE *KERNELS* PARA PROJEÇÃO DE *H*

CÁLCULO DOS PESOS DE SAÍDA *B* E A MATRIZ PSEUDO-INVERSA

TAXA DE ACERTO DO TREINAMENTO E VALOR DO *arg max*

**UTILIZAÇÃO DO ELM COMO *REGRESSOR* E *CLASSIFICADOR***

**TESTE:**

DEFININDO A MATRIZ *H*

UTILIZAÇÃO DE *KERNELS* PARA PROJEÇÃO DE *H*

CÁLCULO DO *arg max* USANDO O MODELO TREINADO

CLASSIFICAÇÃO DOS DADOS A PARTIR DO *arg max*

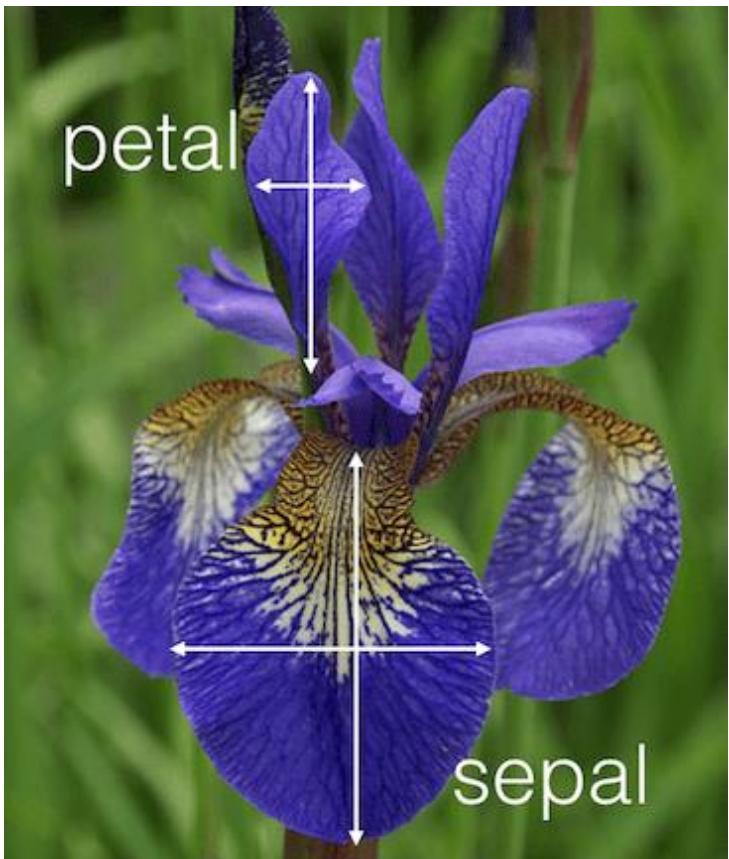
TAXAS DE ACERTO COMO *REGRESSOR* E *CLASSIFICADOR*



EXEMPLO COM A BASE  
“IRIS” DO MATLAB

---

# EXEMPLO COM A BASE “IRIS” DO MATLAB



**CLASSIFICAÇÃO (3 CLASSES):**

- Setosa, Versicolor e Virginica

**4 FEATURES POR CLASSE:**

- 1) Comprimento da pétala;
- 2) Largura da pétala;
- 3) Comprimento da sépala;
- 4) Largura da sépala;

# ENTRADA E SAÍDA DE DADOS PADRÃO DO ELM

LABEL	FEAT. 1	FEAT. 2	FEAT. 3	FEAT. 4
1	5,100	3,500	1,400	0,200
2	7,000	3,200	4,700	1,400
3	6,300	3,300	6,000	2,500

 TREINO

LABEL	FEAT. 1	FEAT. 2	FEAT. 3	FEAT. 4
1	5,000	3,000	1,600	0,200
2	6,600	3,000	4,400	1,400
3	7,200	3,200	6,000	1,800

 TESTE

PARA ESTE EXEMPLO:

- 50% TREINO (SAMPLES #1 - #25 DE CADA CLASSE)
- 50% TESTE (SAMPLES #26 - #50 DE CADA CLASSE)

# DEFINIÇÃO DOS HIPERPARÂMETROS

## ELM PADRÃO:

- # HN (Número De *Hidden Neurons*)

## ELM REGULARIZADO:

- # HN (Número De *Hidden Neurons*)
- Parâmetro regularizador (C)

## PARÂMETROS COMUNS INDEPENDENTES DO MÉTODO:

- **# HN:** Pode ser definido por métodos *post-hoc*
- **FUNÇÃO:** ELM como Regressor ou Classificador
- **KERNEL:** Linear, Sigmoide, Seno, Cosseno, Função de Base Triangular, Função de Base Radial, ...

# PARSING INICIAL

## MACROS PARA A UTILIZAÇÃO DO ELM (função ELM)

```
REGRESSION = 0;  
CLASSIFIER = 1;
```

## LÊ DADOS RELATIVOS AO TREINO

```
labelTreino = Treino(:,1)';  
dadosTreino = Treino(:,2:size(Treino,2))';  
clear Treino;
```

## LÊ DADOS RELATIVOS AO TESTE

```
labelTeste = Teste(:,1)';  
dadosTeste = Teste(:,2:size(Teste,2))';  
clear Teste;
```

## AUTO-DEFINIÇÃO DOS HIPERPARÂMETROS DO MODELO BASEADO NO #HN

```
numAmostrasTR = size(dadosTreino,2);  
numAmostrasST = size(dadosTeste,2);  
NumberofInputNeurons = size(dadosTeste,1);
```

# PARSING INICIAL

```
%% PARSING INICIAL ADICIONAL (SE FUNÇÃO DE CLASSIFICAÇÃO)
% Atribui valores para os labels no intervalo [-1, 1]
if funcaoELM~=REGRESSION

    classes = unique([labelTreino, labelTeste]);
    numClasses = length(classes);
    neoronsSaida = numClasses;
    matrizTreino = zeros(numClasses, length(dadosTreino));
    matrizTeste = zeros(numClasses, length(dadosTreino));

    for Classe=1:numClasses
        matrizTreino(Classe, find(labelTreino == Classe)) = 1;
    end
    matrizTreino=matrizTreino*2-1;

    for Classe=1:numClasses
        matrizTeste(Classe, find(labelTeste == Classe)) = 1;
    end
    matrizTeste=matrizTeste*2-1;
end
```

# TREINAMENTO

- DEFININDO A ENTRADA DE DADOS
- DEFININDO A MATRIZ DE PESOS DE ENTRADA
- DEFININDO A MATRIZ DE BIAS

GERA ALEATORIAMENTE OS PESOS PARA A CAMADA DE ENTRADA E BIAS

% Cria pesos de entrada no intervalo [-1; +1]

```
InputWeight = rand(numHN, NumberofInputNeurons) *2-1;
```

% Cria pesos de entrada no intervalo [-1; +1]

```
BiasofHiddenNeurons=rand(numHN, 1);
```

% Cria bias para 1a Coluna

```
ind=ones(1,numAmostrasTR);
```

```
BiasMatrix=BiasofHiddenNeurons(:,ind);
```

```
tempH=(InputWeight*dadosTreino)+BiasMatrix;
```

```
clear dadosTreino;
```

# TREINAMENTO

- UTILIZAÇÃO DE *KERNELS* PARA PROJEÇÃO DE  $H$

```
%% APLICA O KERNEL EM H PARA O CÁLCULO DA PSEUDOINVERSA
switch lower(kernel)
    case {'sig','sigmoid'}
        %%%%%%%%%
        Sigmoid
        H = 1 ./ (1 + exp(-tempH));
    case {'radbas'}
        %%%%%%%%%
        Radial basis function
        H = radbas(tempH);
    ...
end
clear tempH;
```

**OBS:** VER O CÓDIGO PARA DEMAIS FUNÇÕES (*KERNELS*)

**OBS 2:** SUGIRO QUE VOCÊS EXPERIMENTEM COM ESTE PARÂMETRO TAMBÉM

# TREINAMENTO

- CÁLCULO DOS PESOS DE SAÍDA  $\beta$  E A MATRIZ PSEUDO-INVERSA

CÁLCULO DOS PESOS DE SAÍDA (PSEUDOINVERSA / RIDGE-REGRESSION) :

% PSEUDOINVERSA DE MOORE-PENROSE

```
OutputWeight = pinv(H') * matrizTreino';
```

% RIDGE-REGRESSION (REGULARIZAÇÃO L2) - FORMA #1

```
OutputWeight = inv(eye(size(H,1))/C+H * H') * H * matrizTreino';
```

% RIDGE-REGRESSION (REGULARIZAÇÃO L2) - FORMA #2

```
OutputWeight = (eye(size(H,1))/C+H * H') \ H * matrizTreino';
```

# TREINAMENTO

- CLASSIFICANDO OU FAZENDO REGRESSÃO DOS DADOS DE TREINO PARA CHECAR O **FITTING DO MODELO**:

CÁLCULO DO *arg max* QUE VAI GERAR A CLASSIFICAÇÃO

```
argMaxTreino = (H' * OutputWeight)';
```

**RMSE**

```
if funcaoELM == REGRESSION
    TrainingAccuracy = sqrt(mse(labelTreino - argMaxTreino)) end
clear H;
```

AQUI JÁ É POSSÍVEL OBSERVAR A INFLUÊNCIA DA REGULARIZAÇÃO  
NA REGRESSÃO PARA OS DADOS DE TREINAMENTO

# *TESTE*

- **DEFININDO A MATRIZ *H***
- **UTILIZAÇÃO DE *KERNELS* PARA PROJEÇÃO DE *H***

## **GERA MATRIZ *H* PARA OS DADOS DE TESTE**

```
ind = ones(1,numAmostrasTST);
BiasMatrix = BiasofHiddenNeurons (:,ind);

tempH = (InputWeight*dadosTeste) + BiasMatrix;
clear dadosTeste

switch lower(kernel)
    case {'sig','sigmoid'}
        H_test = 1 ./ (1 + exp(-tempH));
    ...
    case {'radbas'}
        H_test = radbas(tempH);
end
```

# TESTE

- CLASSIFICANDO OU FAZENDO REGRESSÃO DOS DADOS DE TESTE PARA CHECAR O *FITTING* DO MODELO:

## CÁLCULO DO *arg max* E RMSE DO TESTE

```
argMaxTeste = (H_test' * OutputWeight)';  
  
if funcaoELM == REGRESSION  
    TestingAccuracy=sqrt(mse(labelTeste - argMaxTeste))  
end
```

AQUI JÁ É POSSÍVEL OBSERVAR A INFLUÊNCIA DA REGULARIZAÇÃO  
NA REGRESSÃO PARA OS DADOS DE TESTE

# TESTE

- CLASSIFICAÇÃO DOS DADOS A PARTIR DO *arg max*  
**(CLASSE COM O MAIOR VALOR ROTULA A AMOSTRA)**

```
for i = 1 : size(labelTreino, 2)
    [x, label_index_actual] = max(argMaxTreino(:,i));
    if label_index_actual ~= labelTreino(i)
        erroClassificacaoTreino = erroClassificacaoTreino+1;
    end
end
```

TrainAcc = (1-erroClassificacaoTreino/ size(labelTreino,2)\*100

AQUI JÁ É POSSÍVEL OBSERVAR A INFLUÊNCIA DA REGULARIZAÇÃO  
NA CLASSIFICAÇÃO PARA OS DADOS DE TREINAMENTO

# TESTE

- CLASSIFICAÇÃO DOS DADOS A PARTIR DO *arg max*  
**(CLASSE COM O MAIOR VALOR ROTULA A AMOSTRA)**

```
for i = 1 : size(argMaxTeste, 2)
    %[x, label_index_expected]=max(argMaxTeste(:,i));
    [x, label_index_actual]=max(argMaxTeste(:,i));

    labeLEM(i,1) = label_index_actual;

    if label_index_actual~=labelTeste(i)
        erroClassificacaoTeste=erroClassificacaoTeste+1;
    end
end
```

TestAcc = (1-erroClassificacaoTeste/size(argMaxTeste,2))\*100

AQUI JÁ É POSSÍVEL OBSERVAR A INFLUÊNCIA DA REGULARIZAÇÃO  
NA CLASSIFICAÇÃO PARA OS DADOS DE TESTE

# TESTE

- EXEMPLO UTILIZANDO O ELM E O RELM

## LÊ A BASE

```
load('iris_dataset.mat')
```

## SEPARA FEATURES E LABELS DA BASE PARA FORMAR DADOS DE TREINO E TESTE

```
data_tr_c1 = [ones(1,25); irisInputs(:,1:25)];  
data_tst_c1 = [ones(1,25); irisInputs(:,26:50)];  
data_tr_c2 = [ones(1,25)*2; irisInputs(:,51:75)];  
data_tst_c2 = [ones(1,25)*2; irisInputs(:,76:100)];  
data_tr_c3 = [ones(1,25)*3; irisInputs(:,101:125)];  
data_tst_c3 = [ones(1,25)*3; irisInputs(:,126:150)];
```

## TRANSPOE MATRIZES DE ACORDO COM A ENTRADA DO ELM (LABEL + n FEATURES)

```
data_tr_c1=data_tr_c1';  
data_tst_c1=data_tst_c1';  
data_tr_c2=data_tr_c2';  
data_tst_c2=data_tst_c2';  
data_tr_c3=data_tr_c3';  
data_tst_c3=data_tst_c3';
```

# *TESTE*

## **CONCATENA MATRIZES DE TREINO E TESTE**

```
TR = [data_tr_c1; data_tr_c2; data_tr_c3];  
TST = [data_tst_c1; data_tst_c2; data_tst_c3];
```

## **UTILIZAÇÃO DO ELM PADRAO**

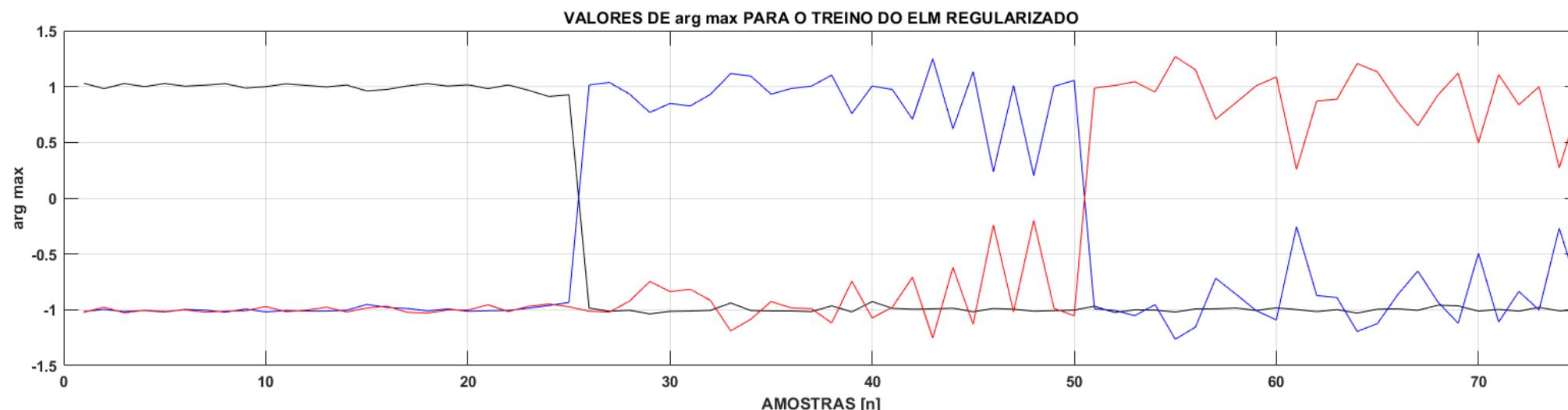
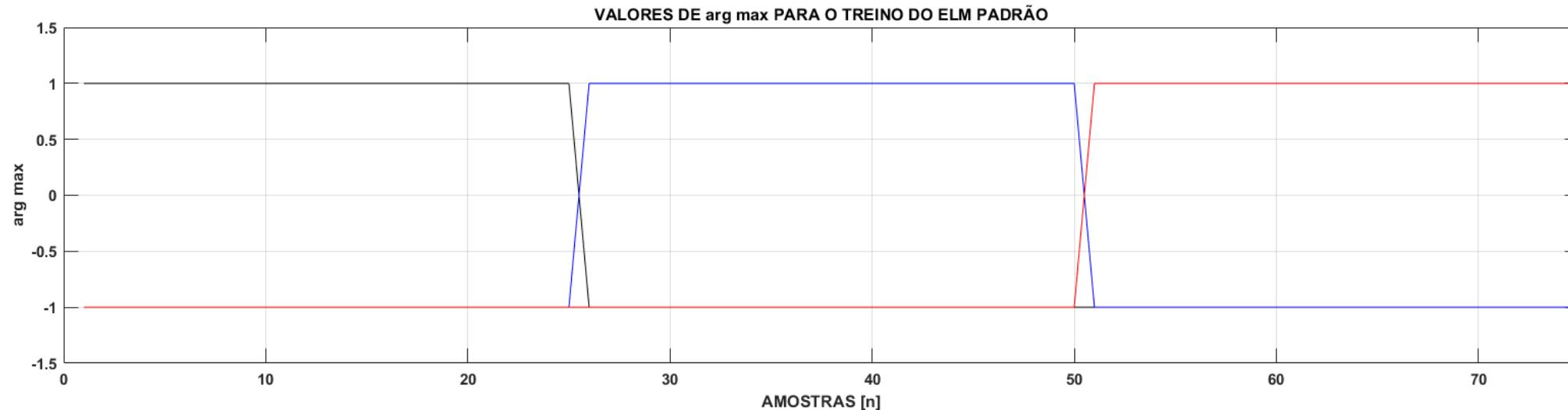
```
Regulariza = 0;  
[TrainingAccuracy, TestingAccuracy, labelPredito, argMaxTreino,  
argMaxTeste] = ELM_Vinicius(TR, TST, 1, 1000, 'radbas', Regulariza,  
ParamRegularizador);
```

## **UTILIZAÇÃO DO ELM REGULARIZADO**

```
Regulariza = 1; % ATIVA A REGULARIZAÇÃO  
ParamRegularizador=0.5; % VALOR ARBITRÁRIO  
[TrainingAccuracyR, TestingAccuracyR, labelPreditor, argMaxTreinoR,  
argMaxTesteR] = ELM_Vinicius(TR, TST, 1, 1000, 'radbas', Regulariza,  
ParamRegularizador);
```

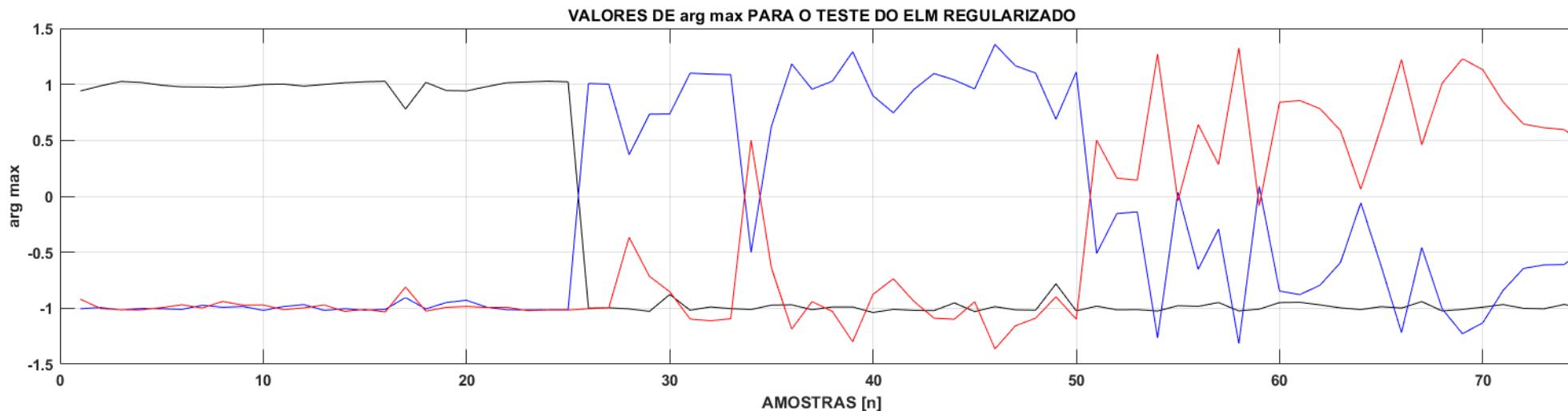
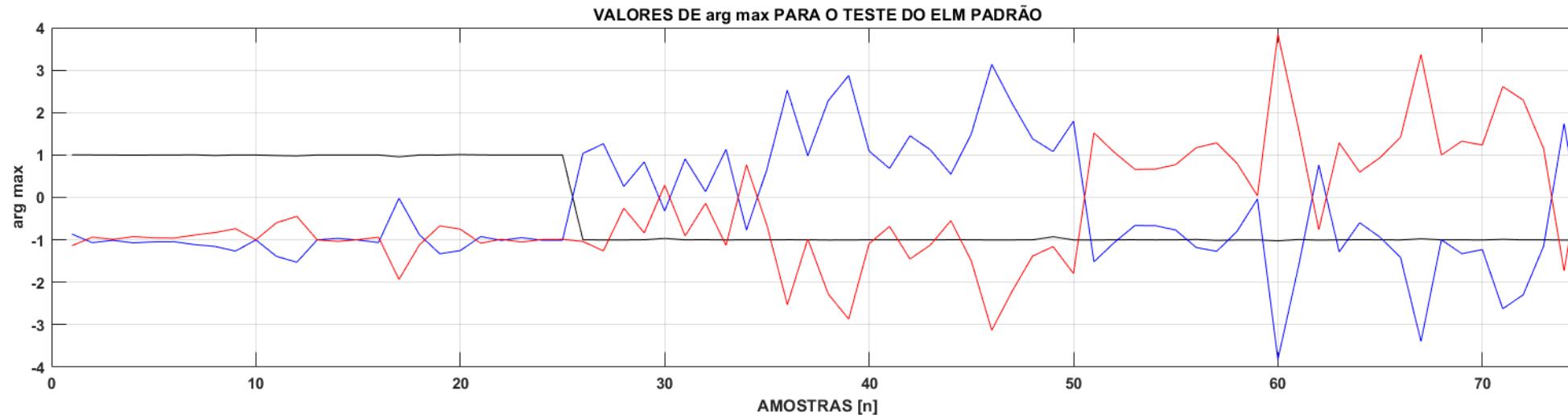
# TESTE

- COMPARAÇÃO ENTRE O MODELO PADRÃO (ELM) E O REGULARIZADO (RELM)



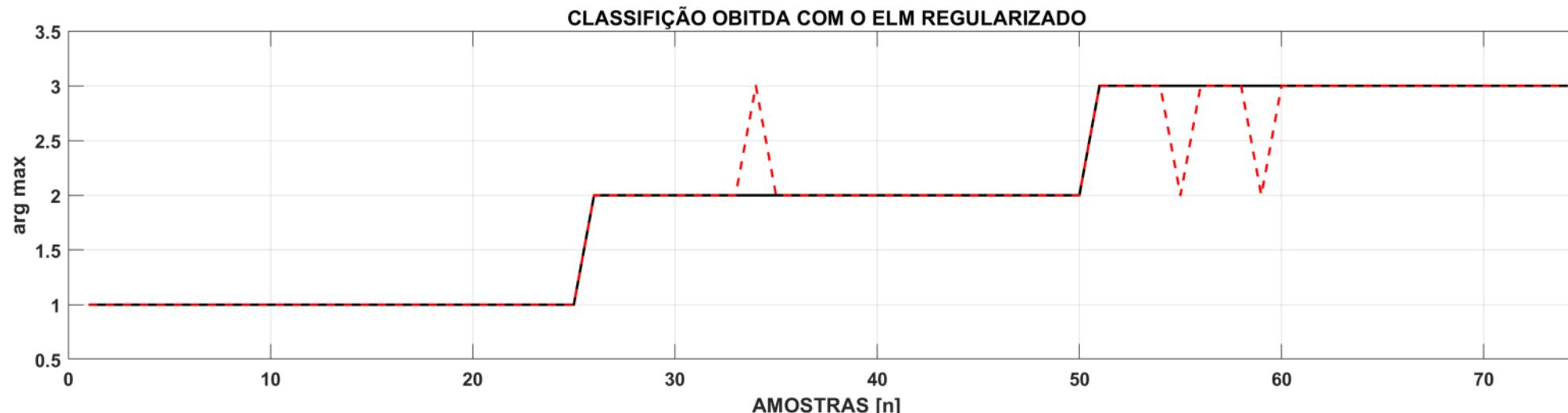
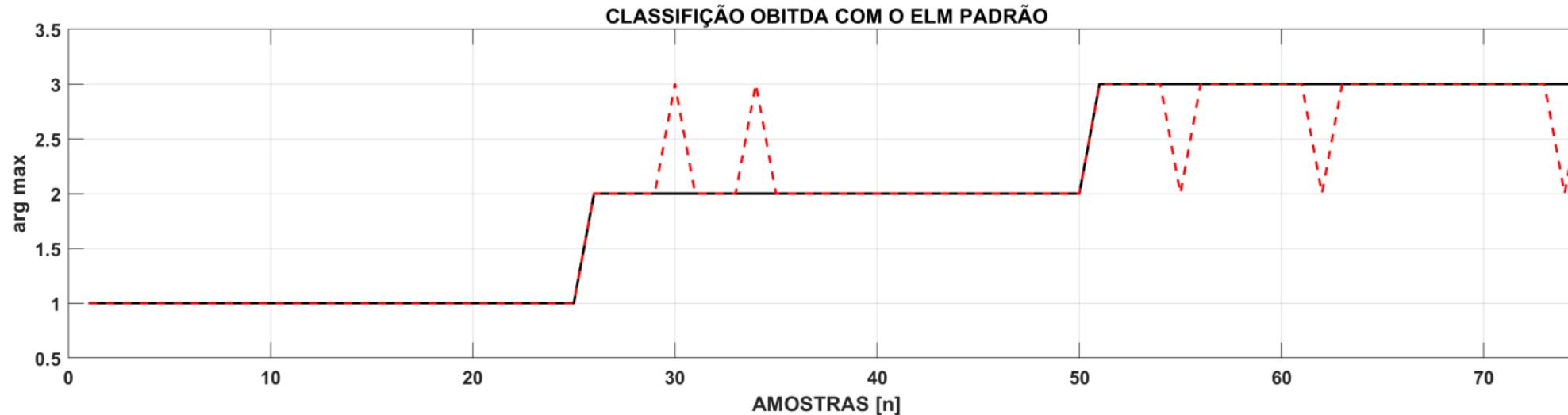
# TESTE

- COMPARAÇÃO ENTRE O MODELO PADRÃO (ELM) E O REGULARIZADO (RELM)



# TESTE

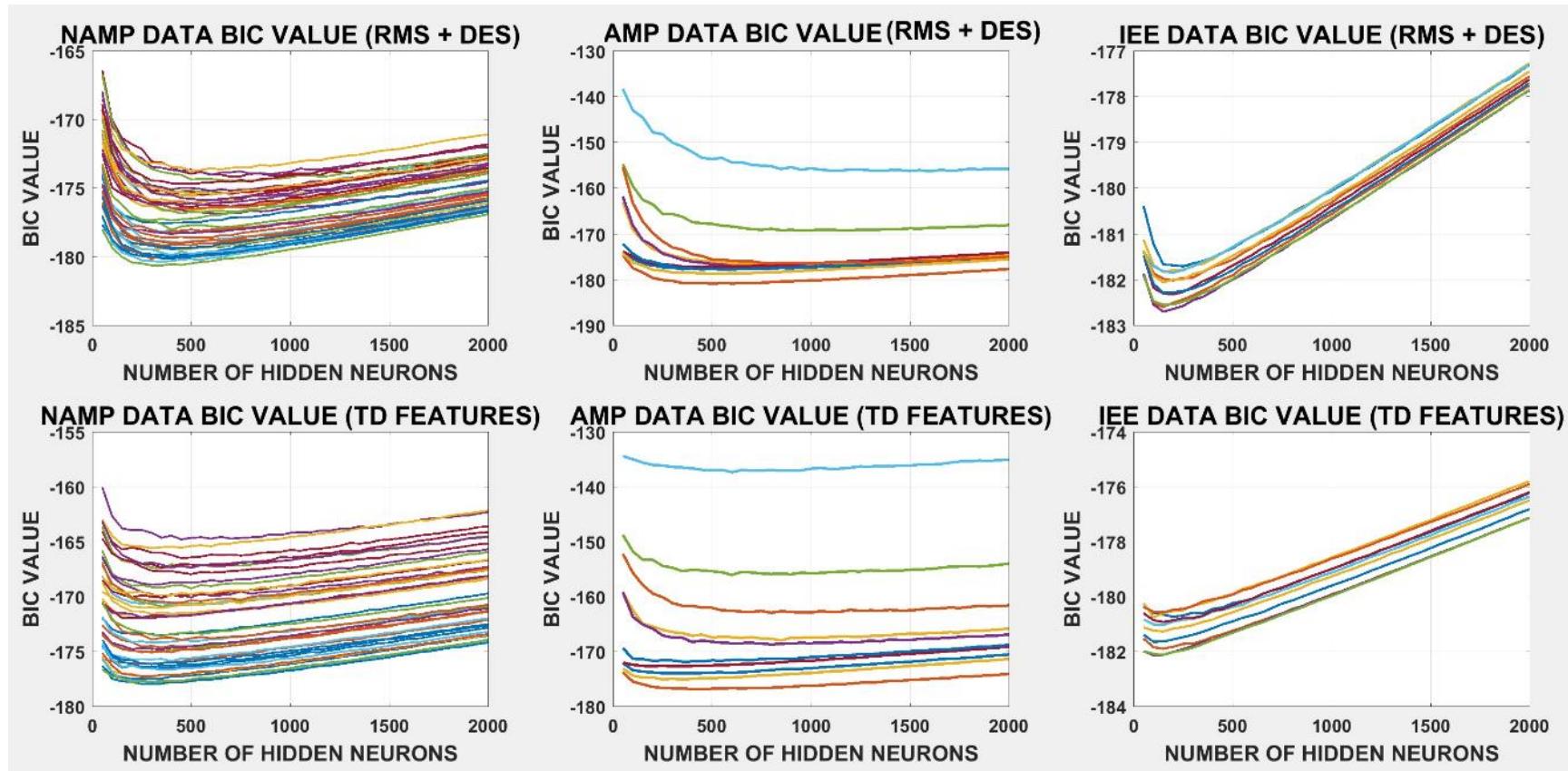
- COMPARAÇÃO ENTRE O MODELO PADRÃO (ELM) E O REGULARIZADO (RELM)



# TÉCNICAS PARA SELEÇÃO DE PARÂMETROS

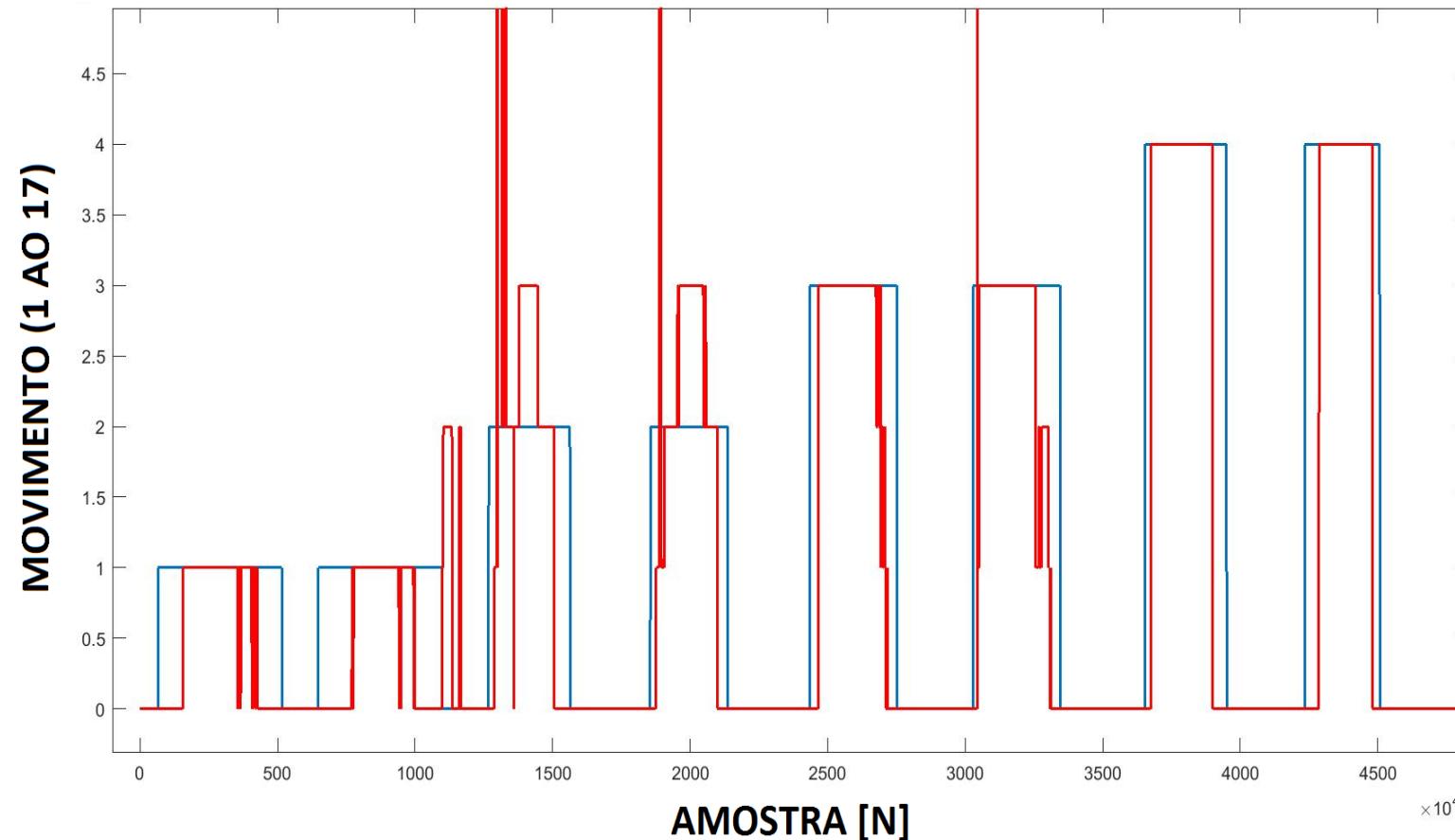
## LEITURA RECOMENDADA:

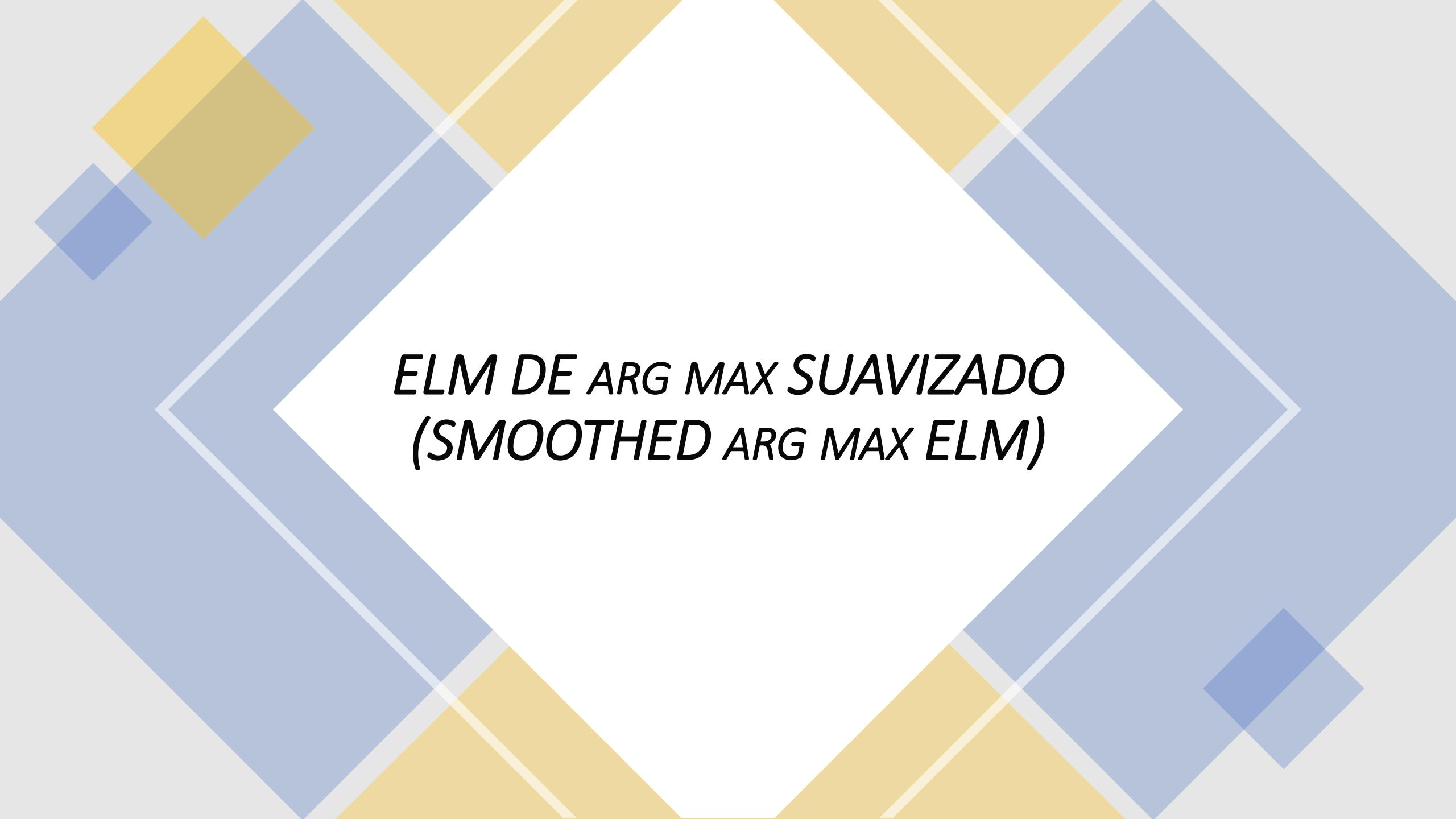
- ALGORITMOS DE PODA (*PRUNING ALGORITHMS*)
- AKAIKE INFORMATION CRITERIA (AIC)
- BAYESIAN INFORMATION CRITERIA (BIC)



# *CONHECIMENTO ESPECÍFICO SOBRE A BASE DE DADOS*

- CONSIDERAR AS CARACTERÍSTICAS DA BASE A SER CLASSIFICADA
- BASES DIFERENTES BENEFICIAM-SE DE ESTRATÉGIAS DIFERENTES
- EXEMPLO: CLASSIFICAÇÃO SINAIS DE sEMG

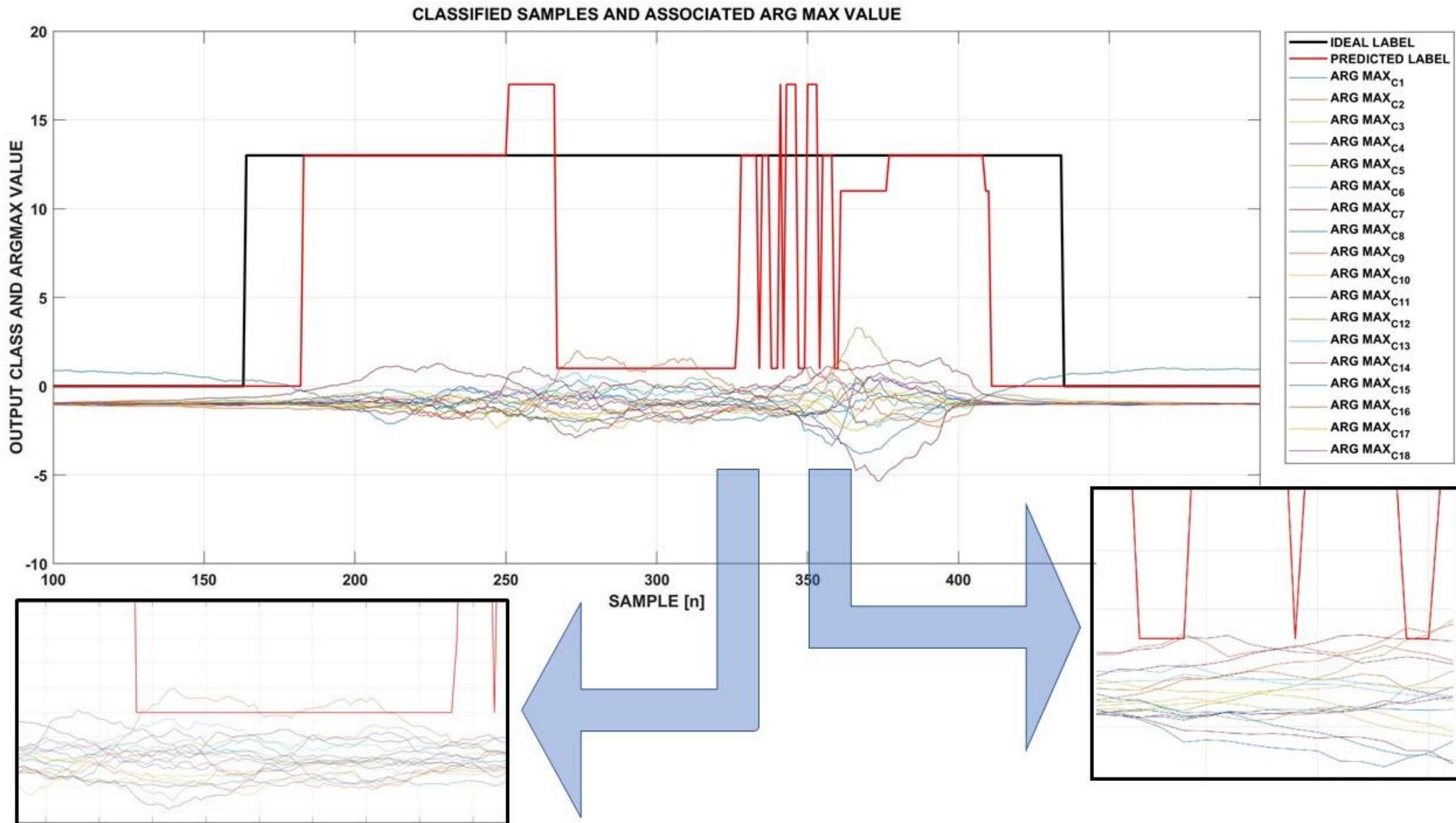




# *ELM DE ARG MAX SUAVIZADO (SMOOTHED ARG MAX ELM)*

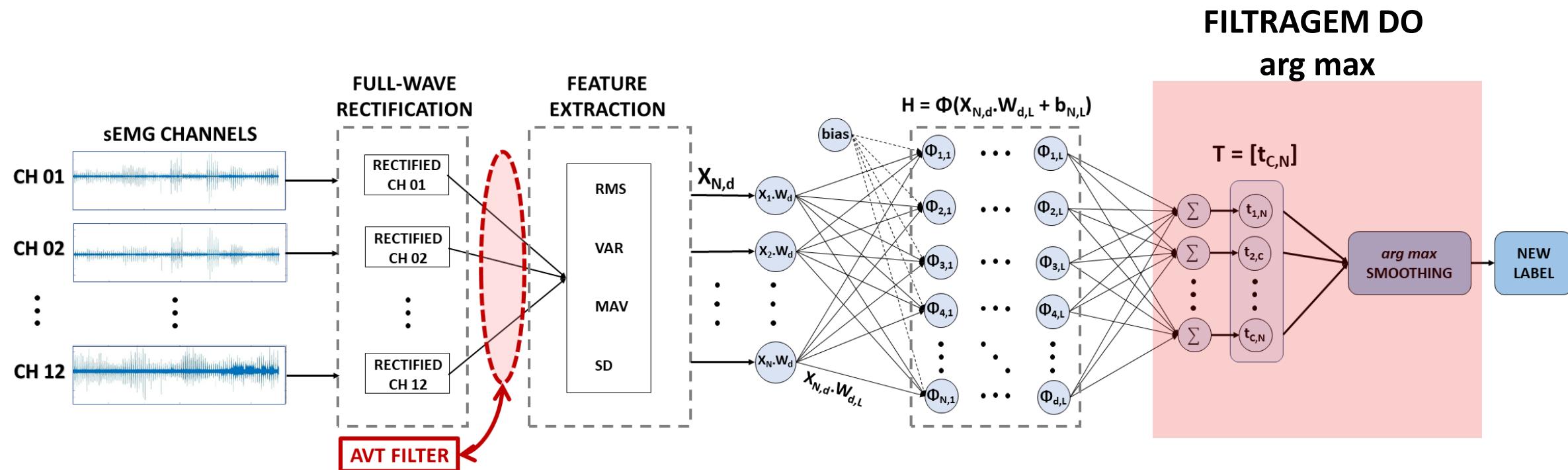
# RUÍDOS DE CLASSIFICAÇÃO

- RUÍDOS INDUZIDOS NO  $\arg \max$  QUE SE PROPAGAM PARA A ATRIBUIÇÃO DE CLASSE



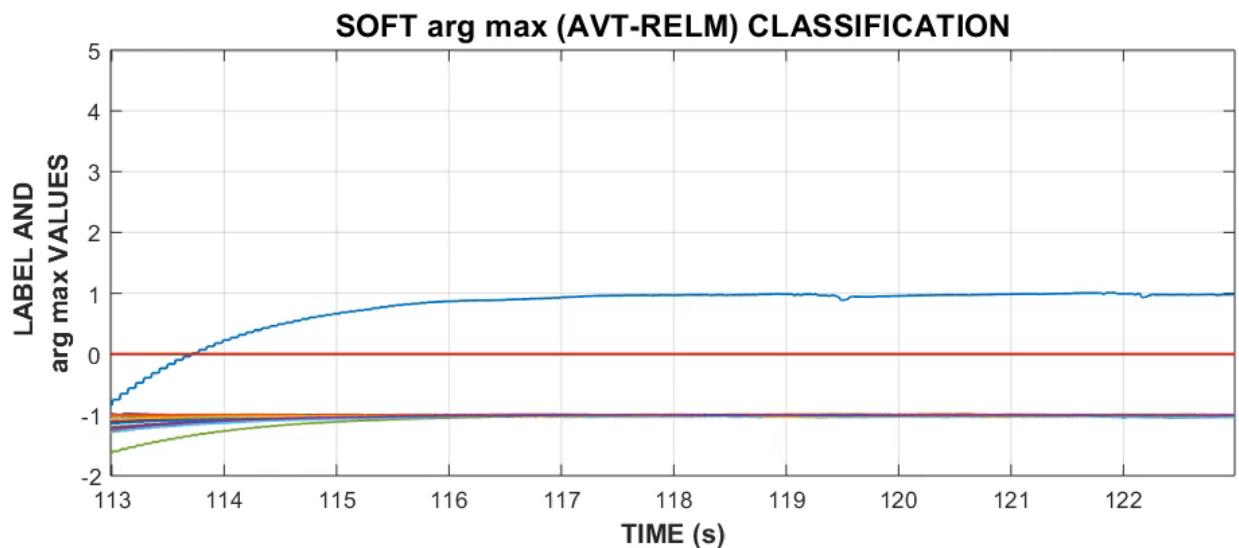
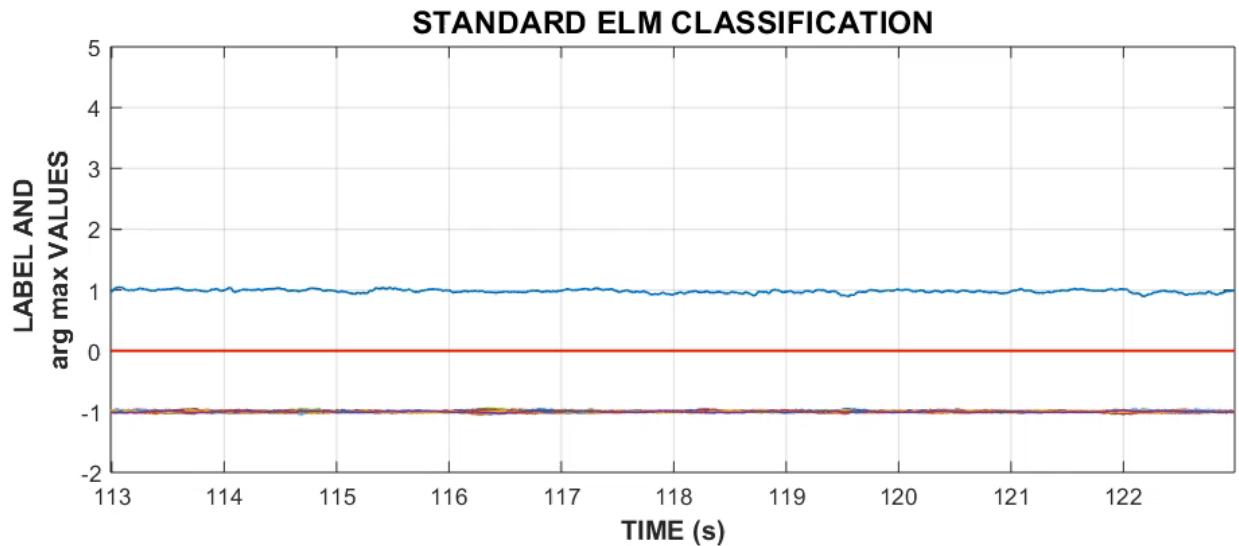
# RUÍDOS DE CLASSIFICAÇÃO

- RUÍDOS INDUZIDOS NO  $\arg \max$  QUE SE PROPAGAM PARA A ATRIBUIÇÃO DE CLASSES



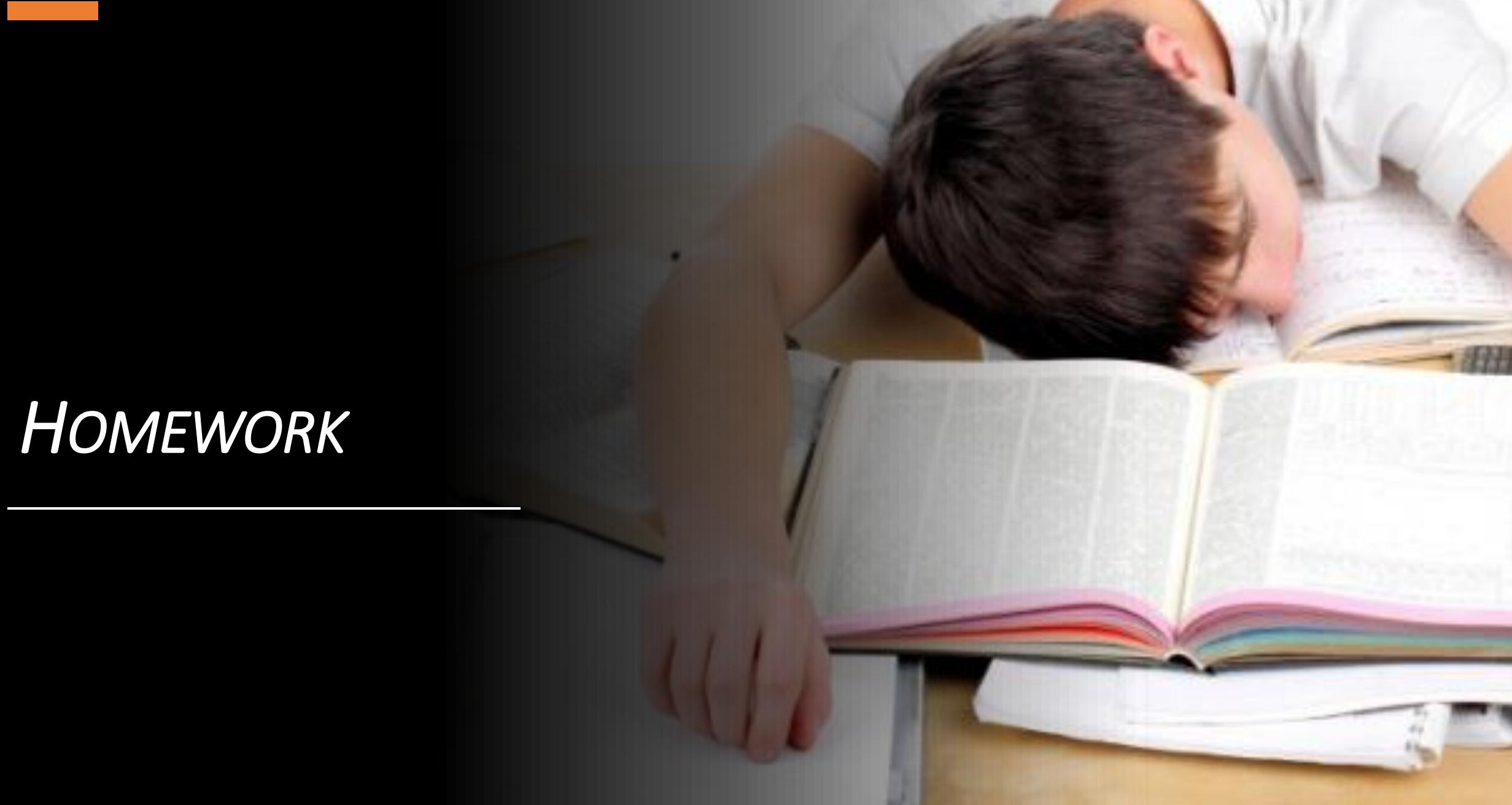
# *RUÍDOS DE CLASSIFICAÇÃO*

FILTRAR O  $\arg \max$  PARA OBTER UMA  
CLASSIFICAÇÃO MAIS CONSISTENTE



# HOMEWORK

---



# ESTUDOS RECOMENDADOS

## 1) SVD:

- 1.1) Pesquisar sobre a aplicação do SVD na compressão de dados e sistemas de recomendação
- 1.2) Aplicar o SVD em uma imagem de sua preferência. Avaliar o efeito para diferentes proporções do rank  $k$  (1%, 5%, ..., 95%, 100%) e components de cor (RGB).
- 1.3) Para cada um dos intervalos de  $k$ , calcule o erro de norma  $L^2$ , o RMSE e a taxa de compressão associada e reflita sobre os resultados.

## 2) ELM:

- 2.1) Utilizando o ELM fornecido e base “iris”, calcule os pesos de saída do modelo. Calcule também o mesmo parâmetro utilizando o método *svd* do Matlab. Calcule a diferença numérica em ambas abordagens.
- 2.2) Varie o **#HN** ([50:50:1000]) para as versões padrão e regularizada do ELM e gere curvas de resultados para as taxas de acerto de treinamento e teste.
- 2.3) Utilizando o **#HN** que forneceu a maior taxa de acertos, varie os *kernels* para a projeção de features do modelo, plote os resultados e disserte sobre eles.

# CONTATO

---

***Dr. Vinícius Horn Cene***

---

*Programa de Pós-Graduação em Engenharia Elétrica*

---

*PPGEE – UFRGS*

---

*Laboratório de Instrumentação*

---

*Eletro-Eletrônica & Biomédica – IEE&Bio*

---

**Prédio do Instituto Eletrotécnico**

---

**DELET, Sala #206-D**

Research Gate: [https://www.researchgate.net/profile/Vinicius\\_Cene](https://www.researchgate.net/profile/Vinicius_Cene)

Linked-In: <https://www.linkedin.com/in/vinicius-cene-17695225/>

GitHub (Aula): <https://github.com/ViniciusCene/Classes>

e-mail: [vinicius.cene@gmail.com](mailto:vinicius.cene@gmail.com)

*Inteligência Computacional I – ELE318 – “NA 15”*