Inteligência Artificial

Redes Neuronais Artificiais (Parte IV)

Paulo Moura Oliveira

Departamento de Engenharias

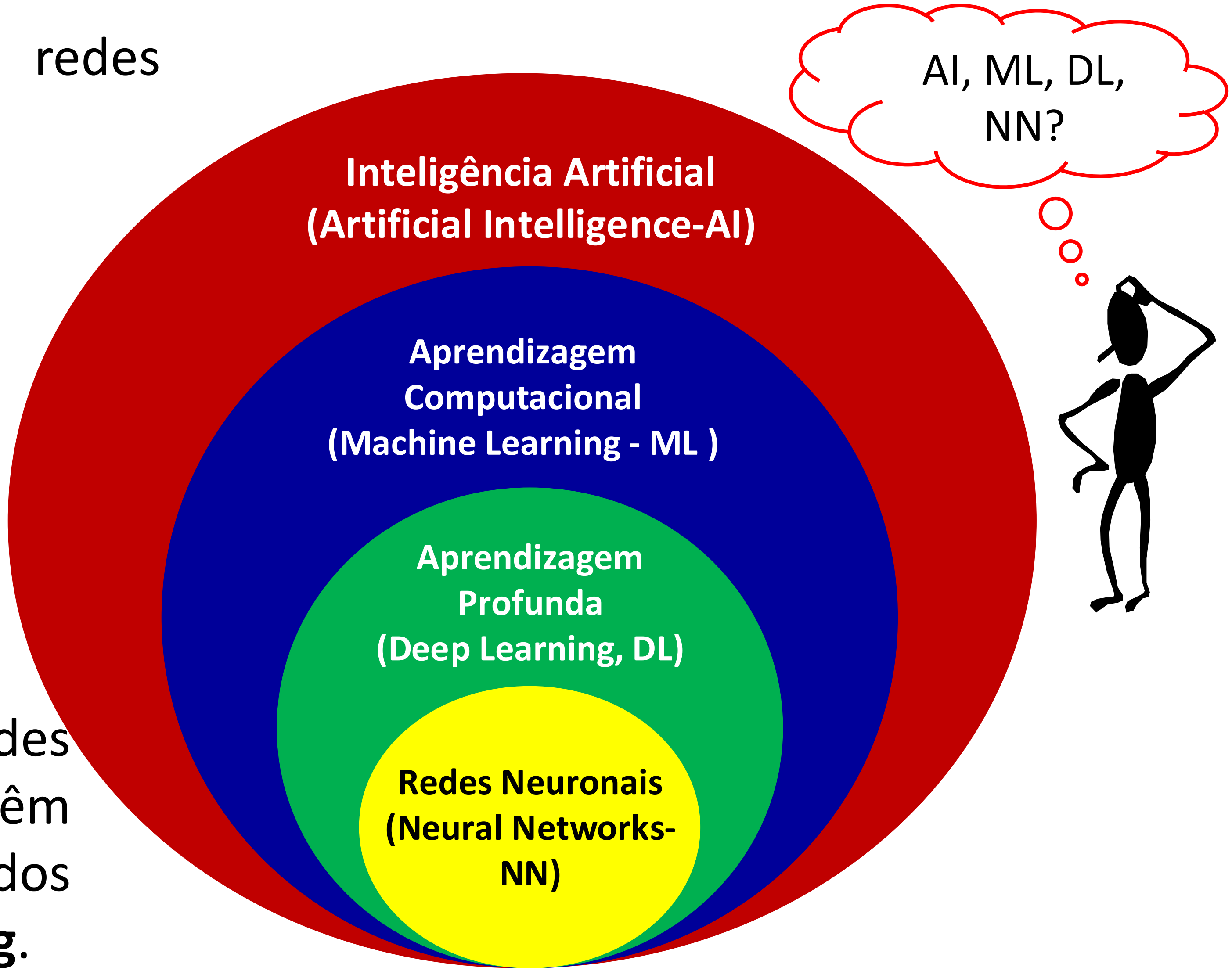
Gabinete F2.15, ECT-1

UTAD

email: oliveira@utad.pt

AI-ML-DL-NN

- Introdução às redes neuronais artificiais.
- A saga continua…



- Há conceitos das redes neuronais que têm vindo a ser adaptados para o **Deep Learning**.

Funções de Custo

- ✓ Se consideramos a minimização do quadrado do erro:

$$E = \frac{1}{2}(t-y)^2 = \frac{1}{2}e^2$$

SSE- Sum of Square Errors

$$\boxed{SSE = \frac{1}{2}\sum_{i=1}^{N} e_i^2}$$

MSE- Mean Squared Errors

$$\boxed{MSE = \frac{1}{N}\sum_{i=1}^{N} e_i^2}$$

CEE- Crossed Entropy Error

$$\boxed{CEE = -\sum_{i=1}^{N}[t_i \log y_i + (1-t_i)\log y_i]}$$

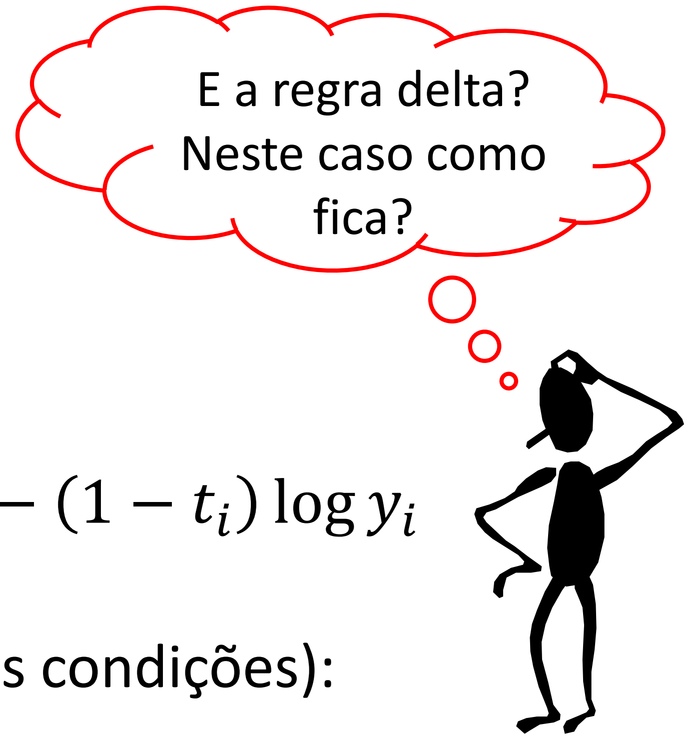
- ✓ Pode-se mostrar que todos este índices são proporcionais ao erro.

Funções de Custo

- Há vantagens em utilizar o CCE no processo de aprendizagem das RN.

- Como a regra delta foi obtida considerando:

$$E = \frac{1}{2}(t-y)^2 = \frac{1}{2}e^2$$



- Neste caso a regra delta fica: $E = -t_i \log y_i - (1-t_i) \log y_i$

- Considerando o mesmo exemplo XOR (mesmas condições):

$$\delta_s = y \times (1-y) e \omega_j$$

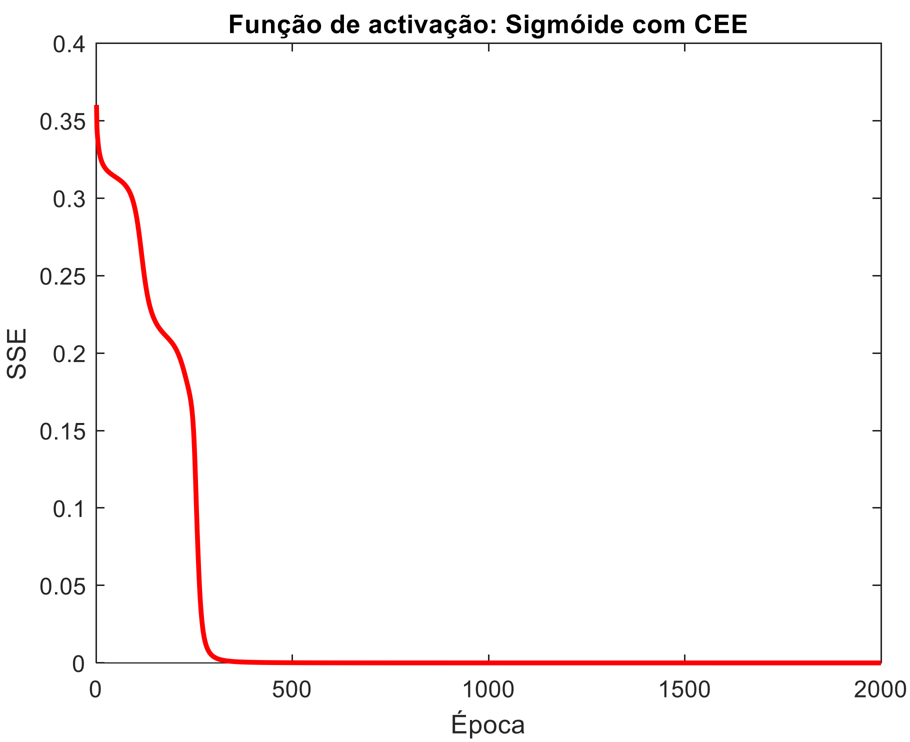
$$\Delta\omega_{hi} = \alpha \times o_{hi} \times \delta_s$$

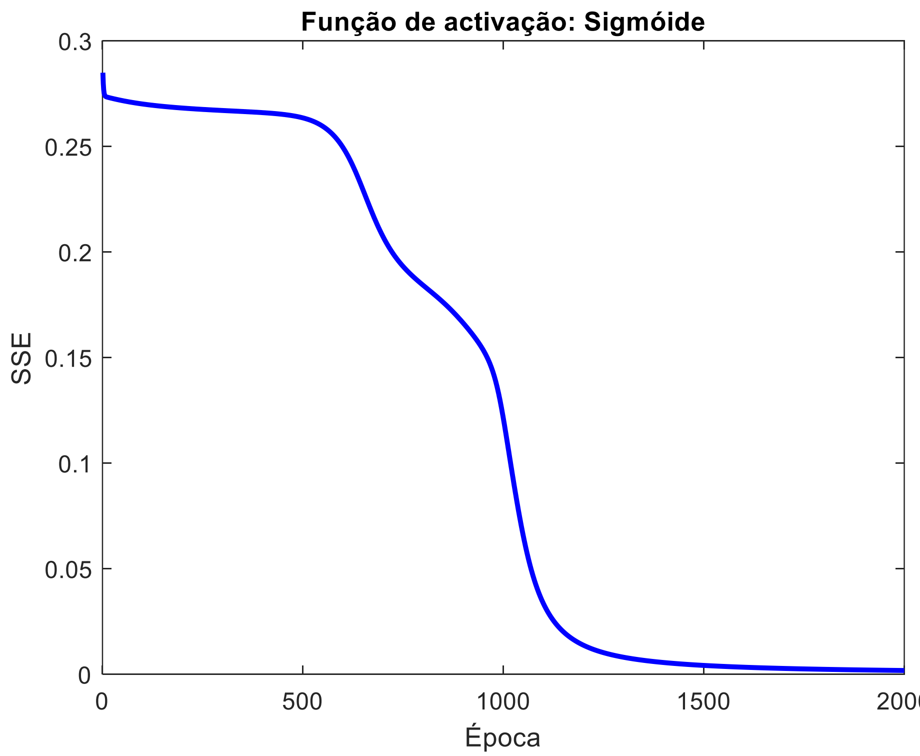
$$\delta_s = e \omega_j$$

$$\Delta\omega_{hi} = \alpha \times o_{hi} \times \delta_s$$

Funções de Custo

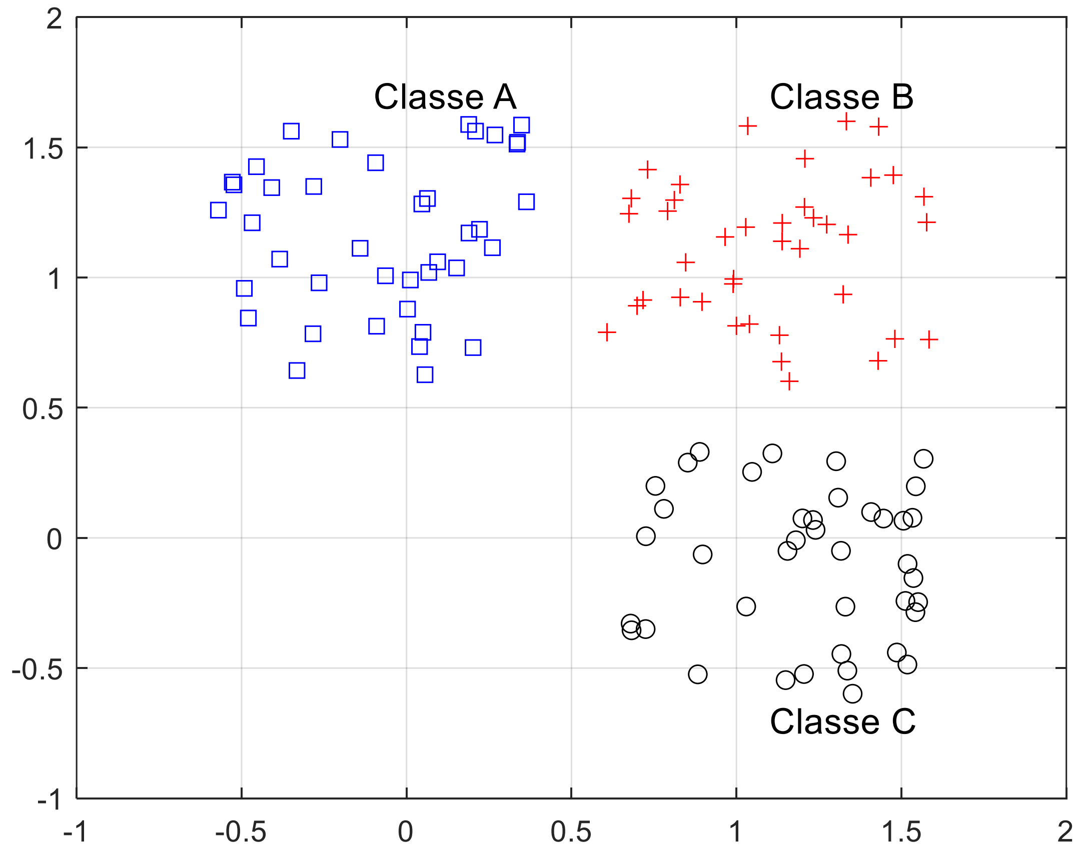
- Considerando o mesmo exemplo XOR (mesmas condições do exemplo do final da Parte III):





Classificação Multi-Classe

- ✓ Até este ponto só consideramos RN com uma saída:
- ✓ Grande parte das aplicações práticas implicam classificar várias classes de dados:
- ✓ Vamos considerar o seguinte exemplo com 3 classes designadas como A, B e C (40 amostras geradas aleatoriamente por classe):



Como utilizar uma RN para classificar pontos nestas três classes?

Classificação Multi-Classe

Como utilizar uma RN para classificar pontos nestas três classes?

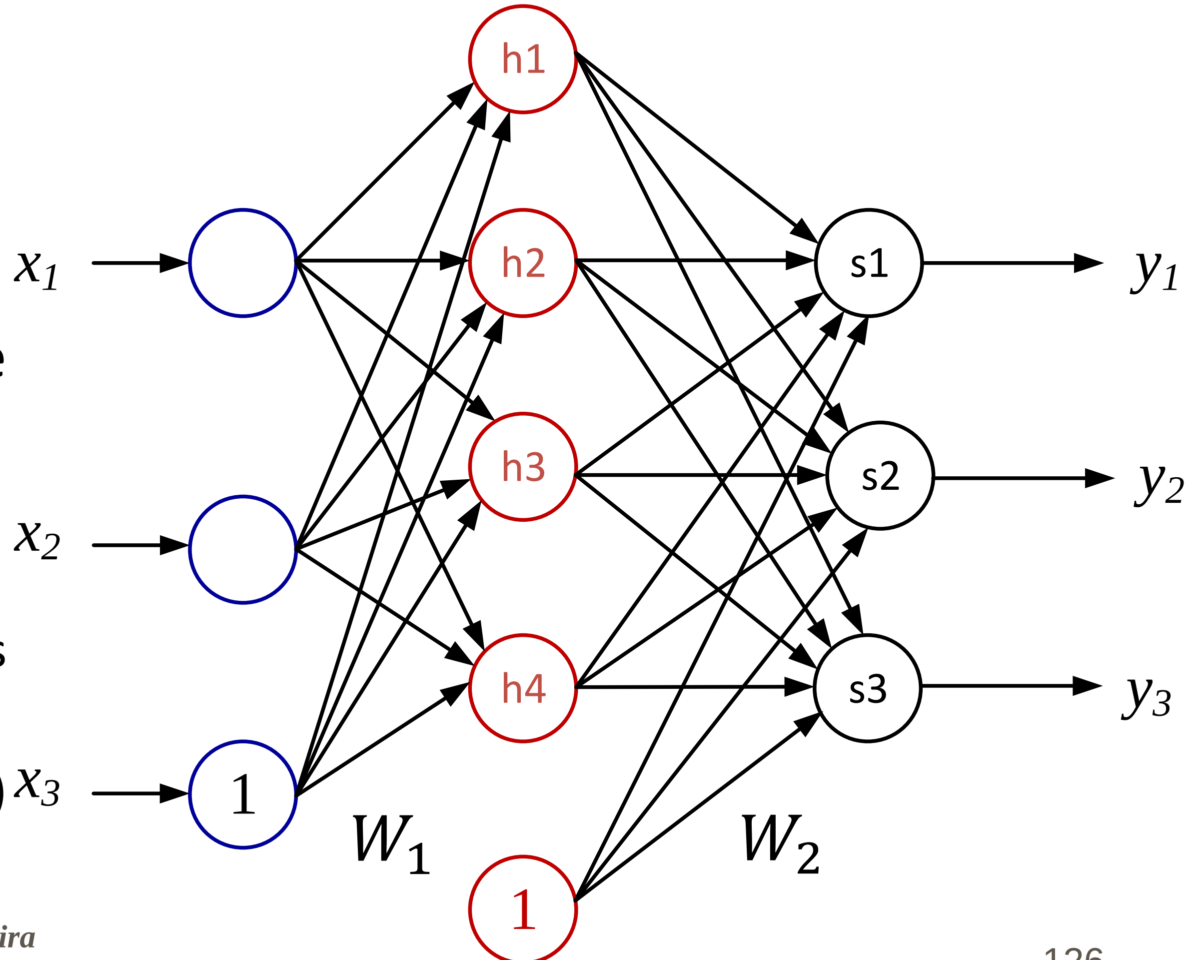
✓ Vamos assumir que os valores desejados (target) para cada uma das três classes é representado usando valores binários:

- **Classe A – [1 0 0]**
- **Classe B – [0 1 0]**
- **Classe C – [0 0 1]**

✓ Vamos utilizar a seguinte topologia da RN.

✓ Notar que:

- poderíamos utilizar outras topologias.
- este exemplo é (suposto ser) didático.

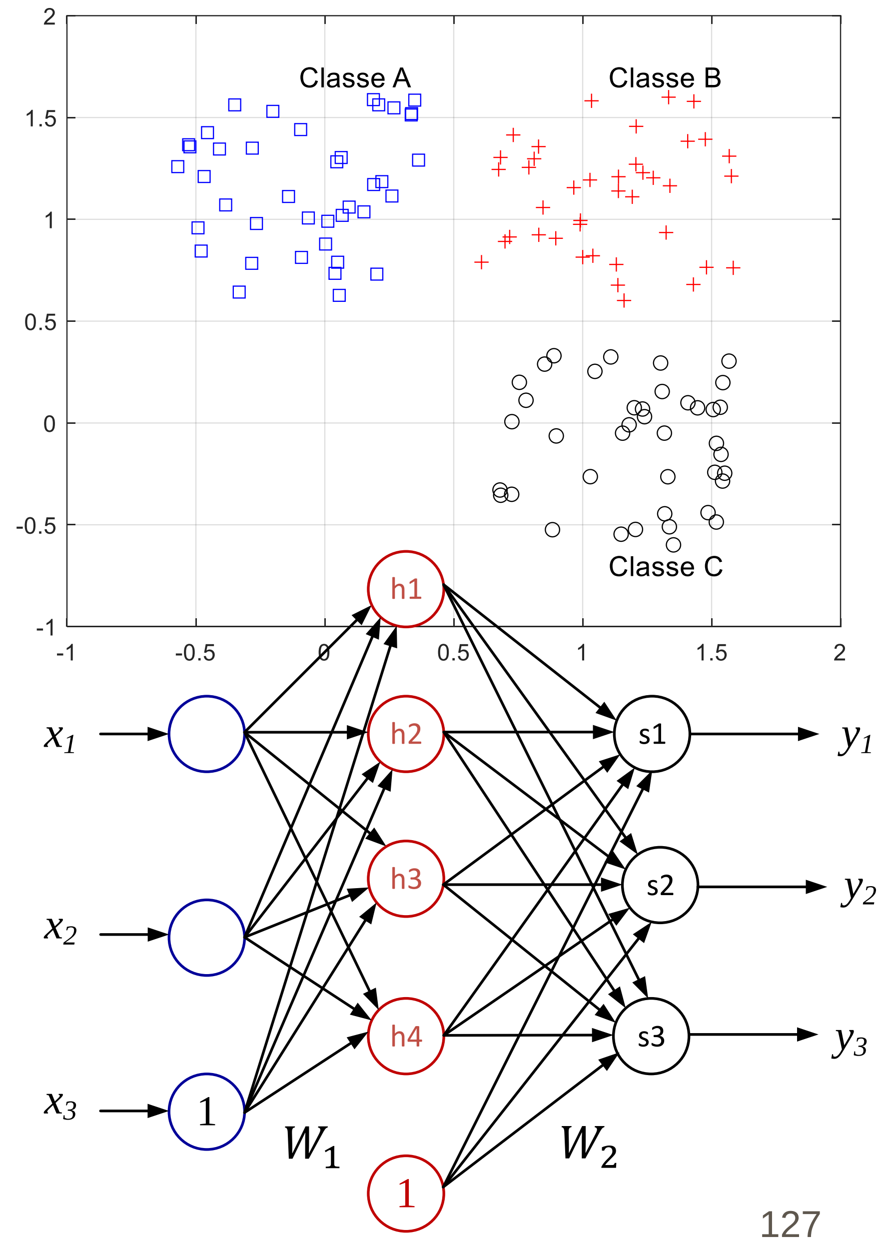


Treino da Rede Neuronal

✓ A título de exemplo considerem-se que as 120 amostras são colocadas nas entradas de forma sequencial:

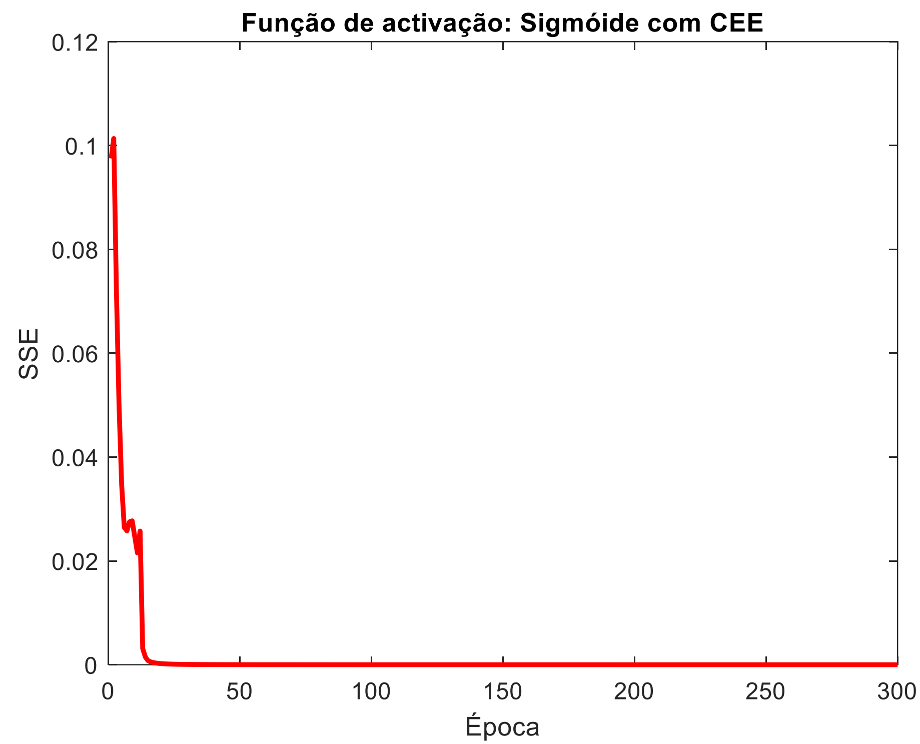
	x_1	x_2	x_3
p_1			
.		A	
.			
p_{40}			
p_{41}			
.		B	
.			
p_{80}			
p_{81}			
.		C	
.			
p_{120}			

1	0	0
1	0	0
	t_A	
1	0	0
0	1	0
0	1	0
	t_B	
0	1	0
0	0	1
0	0	1
	t_C	
0	0	1



Classificação Multi-Classe

Treino da Rede Neuronal

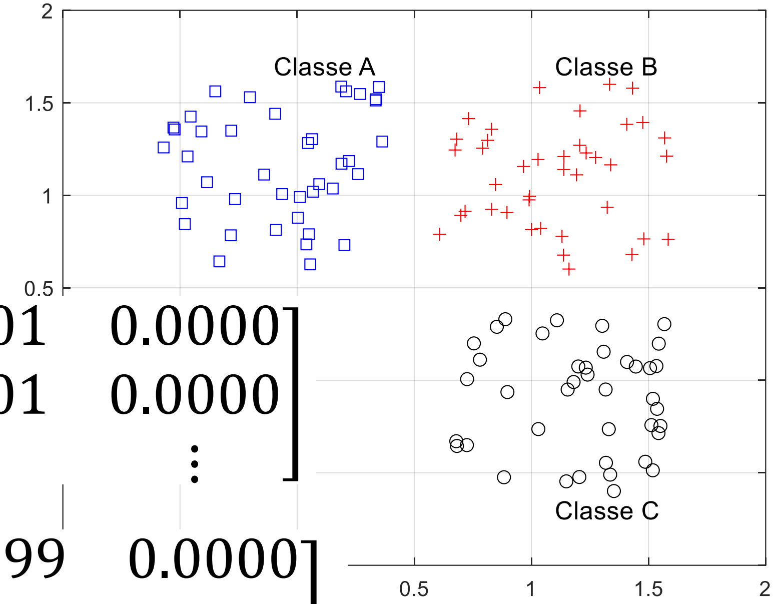


- Considerando um número baixo de épocas(300) para **Treinar a RN** a partir de um conjunto de pesos inicializados aleatoriamente obtiveram-se os seguintes resultados.

$$W_1 = \begin{bmatrix} -12.726 & 2.762 & 2.941 \\ 7.835 & -1.694 & -1.799 \\ -0.514 & 11.219 & -4.700 \\ 0.623 & -10.861 & 4.401 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 11.423 & -8.812 & 2.104 & -2.2341 & -3.292 \\ -14.201 & 4.8924 & 8.286 & -11.335 & -3.639 \\ -2.923 & 1.1944 & -11.250 & 9.136 & 0.1160 \end{bmatrix}$$

Teste da Rede Neuronal



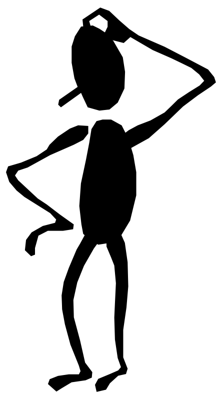
- ✓ Considerando as primeiras duas amostras de cada categoria:

$$X_A = \begin{bmatrix} -0.3493 & 1.562 \\ -0.5231 & 1.3560 \\ \vdots & \vdots \end{bmatrix} \quad Y_A = \begin{bmatrix} 1.000 & 0.0001 & 0.0000 \\ 1.000 & 0.0001 & 0.0000 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

$$X_B = \begin{bmatrix} 1.1386 & 1.1393 \\ 1.3236 & 0.9352 \\ \vdots & \vdots \end{bmatrix} \quad Y_B = \begin{bmatrix} 0.0000 & 0.9999 & 0.0000 \\ 0.0000 & 0.9999 & 0.0001 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

$$X_C = \begin{bmatrix} 1.1801 & -0.0092 \\ 1.3362 & -0.5098 \\ \vdots & \vdots \end{bmatrix} \quad Y_C = \begin{bmatrix} 0.0000 & 0.0000 & 0.9999 \\ 0.0000 & 0.0000 & 1.0000 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

- ✓ Com os pesos apresentados a RN consegue classificar corretamente as 120 amostras usadas no seu treino.

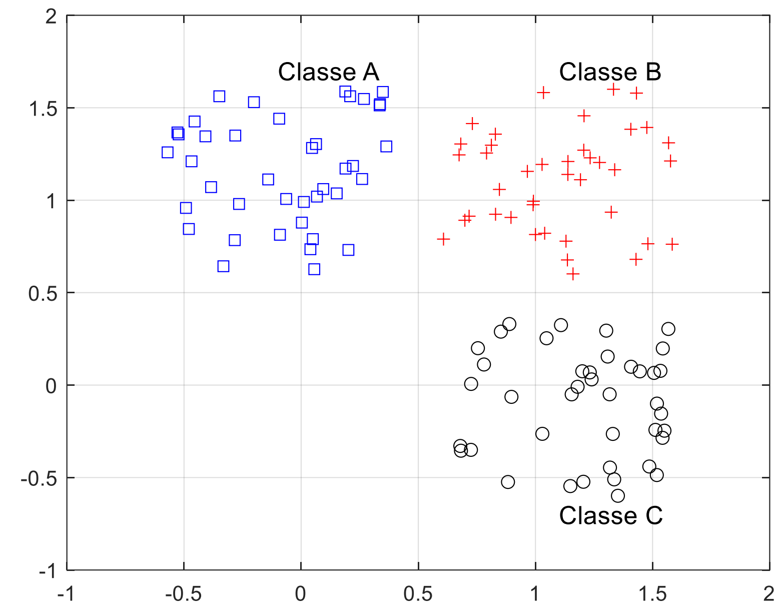


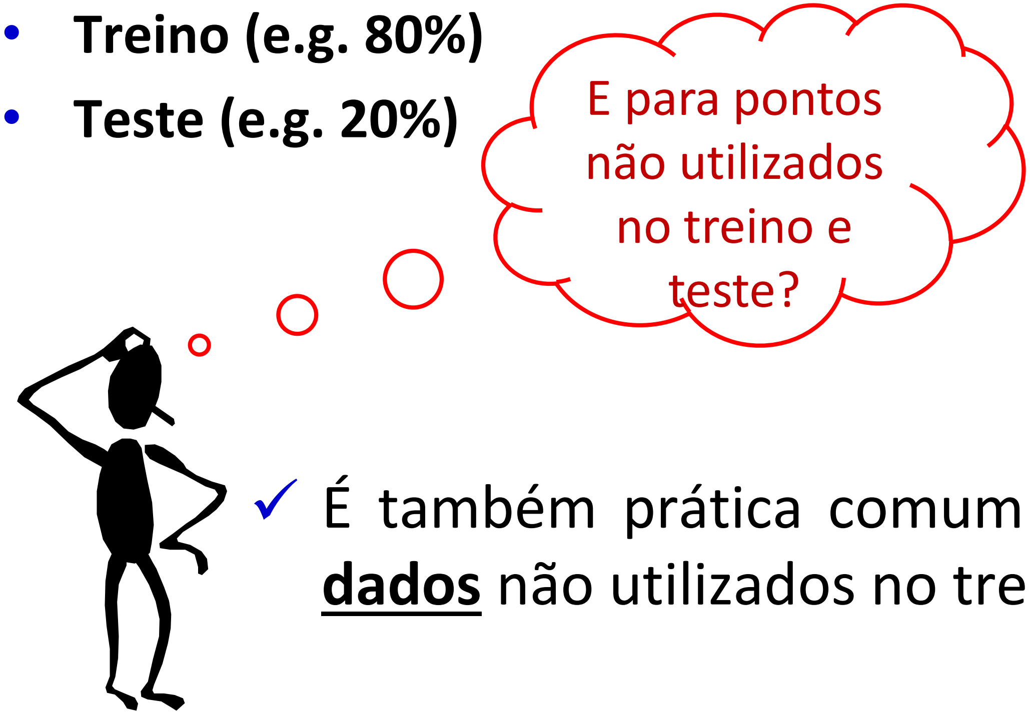
Classificação Multi-Classe

Teste-Validação da Rede Neuronal

✓ É prática comum nunca separar o conjunto de dados em dois sub-conjuntos:

- **Treino (e.g. 80%)**
- **Teste (e.g. 20%)**





✓ É também prática comum **validar** a rede com **<u>outros dados</u>** não utilizados no treino e teste da RN.

Classificação Multi-Classe

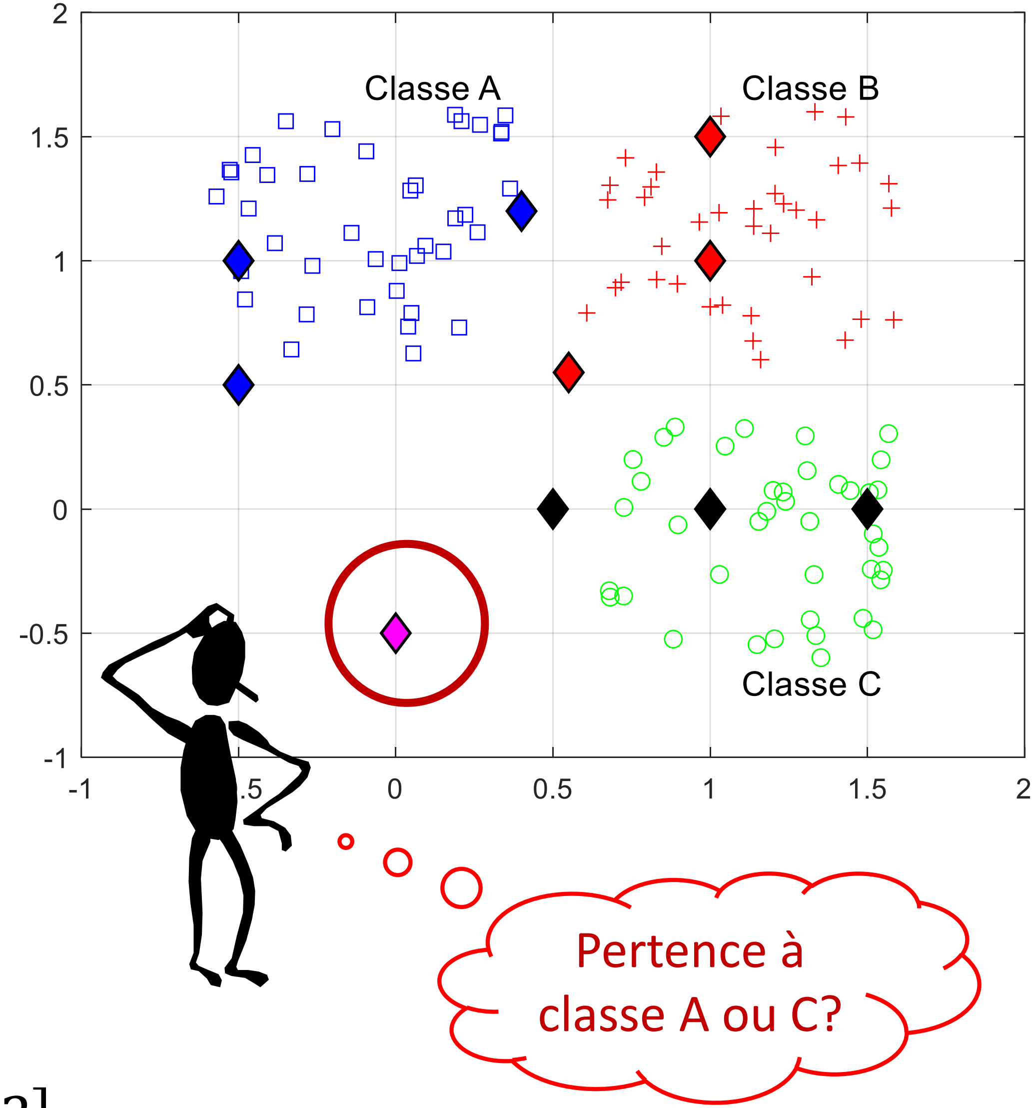
Validação da Rede Neuronal

- ✓ Considerem-se a título ilustrativo os ponto adicionais representados com losangos:

- ✓ Como se pode observar pela cor atribuída a cada losango, 9 das 10 amostras foram classificadas numa classe.

- ✓ No entanto para o 10º ponto o resultado obtido na saída da RN é o seguinte:

$$y_{10} = [0.7954 \quad 0.0000 \quad 0.9992]$$



Função Softmax

- A função de ativação *softmax* oferece vantagens na classificação multi-classe.

- A saída da unidade de saída j é representada por:

$$y_i = f(g_j) = \frac{e^{g_j}}{e^{g_1} + e^{g_2} + \cdots + e^{g_m}} = \frac{e^{g_j}}{\sum_{k=1}^{m} e^{g_k}}$$

- Com:

$$f(g_1) + f(g_2) + \cdots + f(g_n) = 1$$

- Esta função de ativação também funciona na classificação binária.

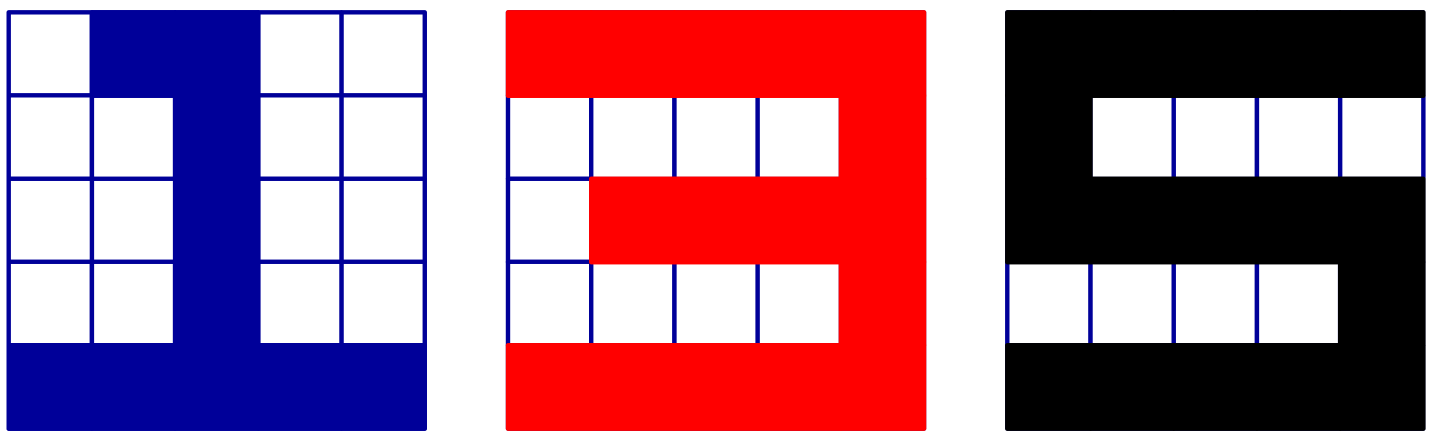
Classificação Multi-Classe

Exemplo:

- Vamos considerar o problema clássico de reconhecimento de uma matrizes representando imagens de dígitos.

- Suponha que tinha de projetar, treinar e validar uma RN para reconhecer os números 1, 3 e 5.

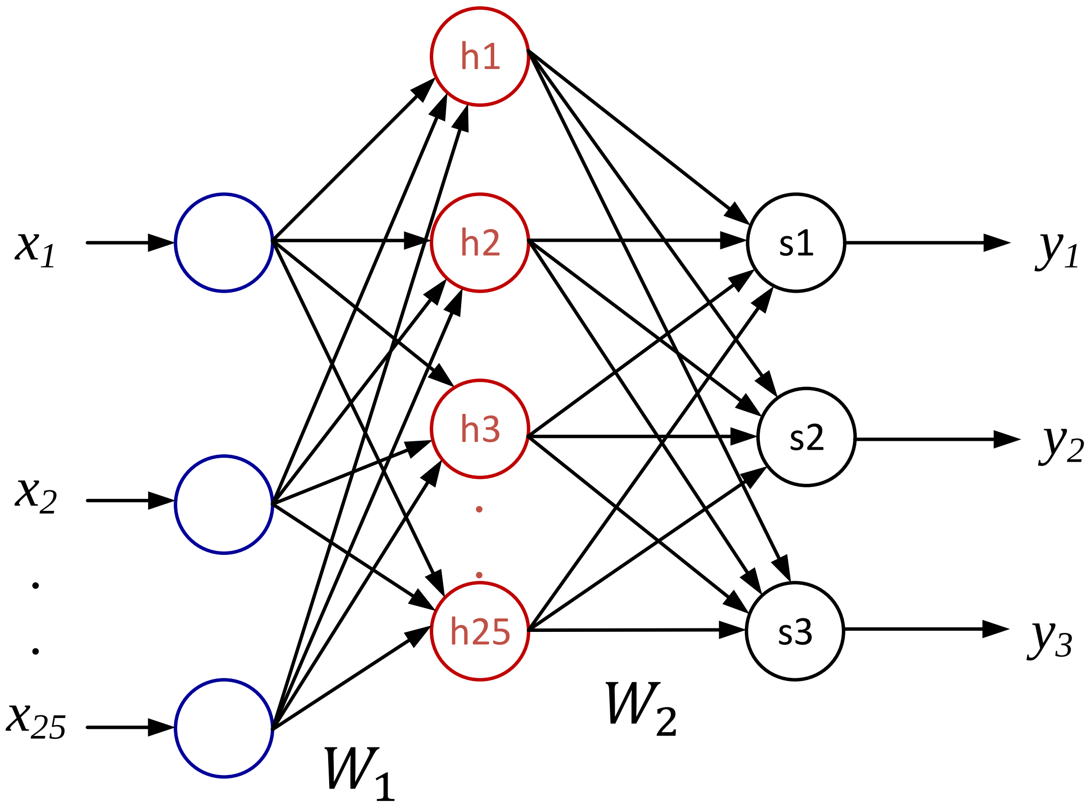
- Assuma a seguinte representação para cada dígito utilizando uma matriz binária de 5x5:



Classificação Multi-Classe

Exemplo:

✓ Considere uma topologia de rede como se apresenta, utilizando a função sigmoide na camada escondida e *softmax* na de saída.

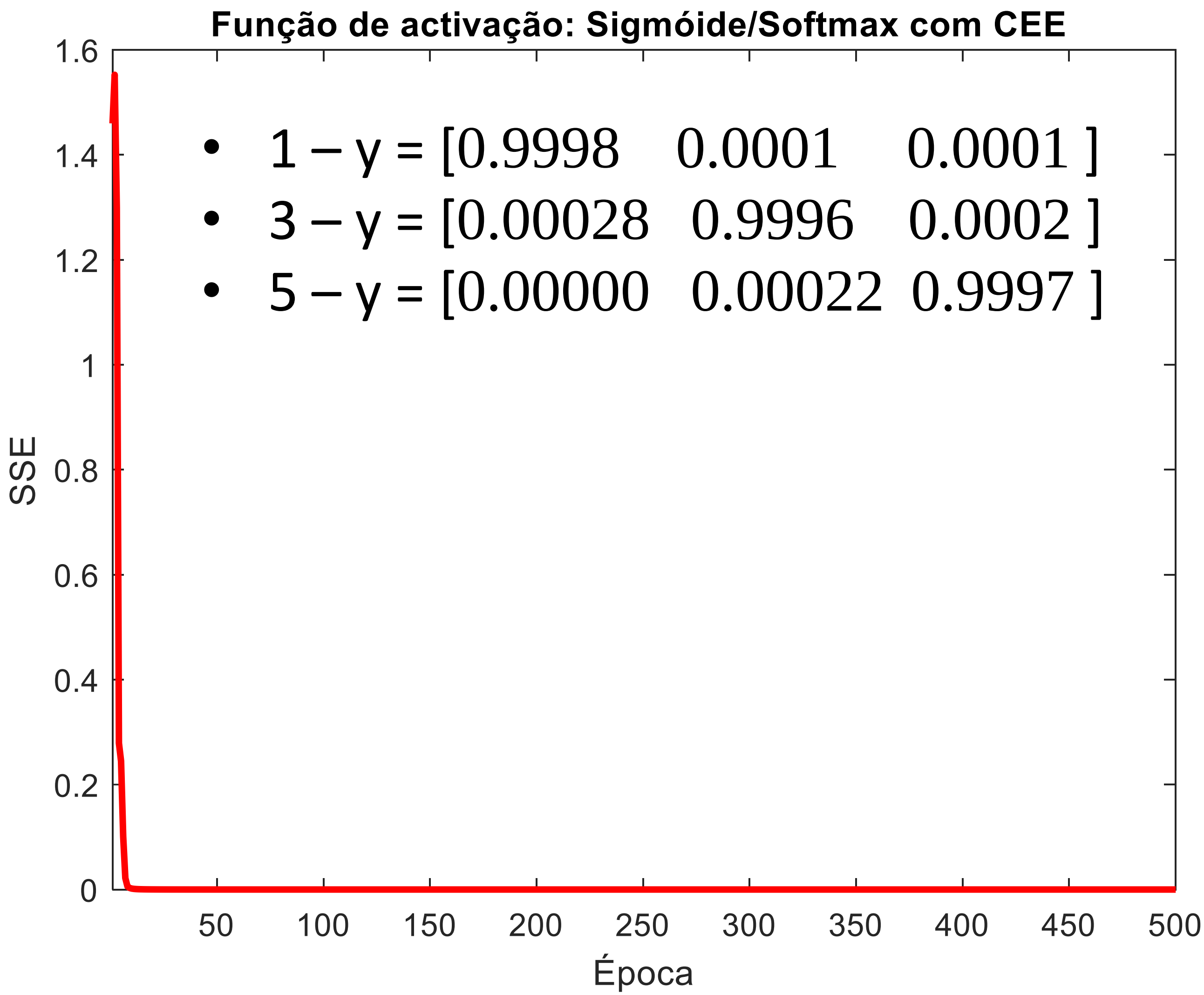


- **1 – y = [1 0 0]**
- **3 – y = [0 1 0]**
- **5 – y = [0 0 1]**

Classificação Multi-Classe

Exemplo:

- **1 – y = [1 0 0]**
- **3 – y = [0 1 0]**
- **5 – y = [0 0 1]**



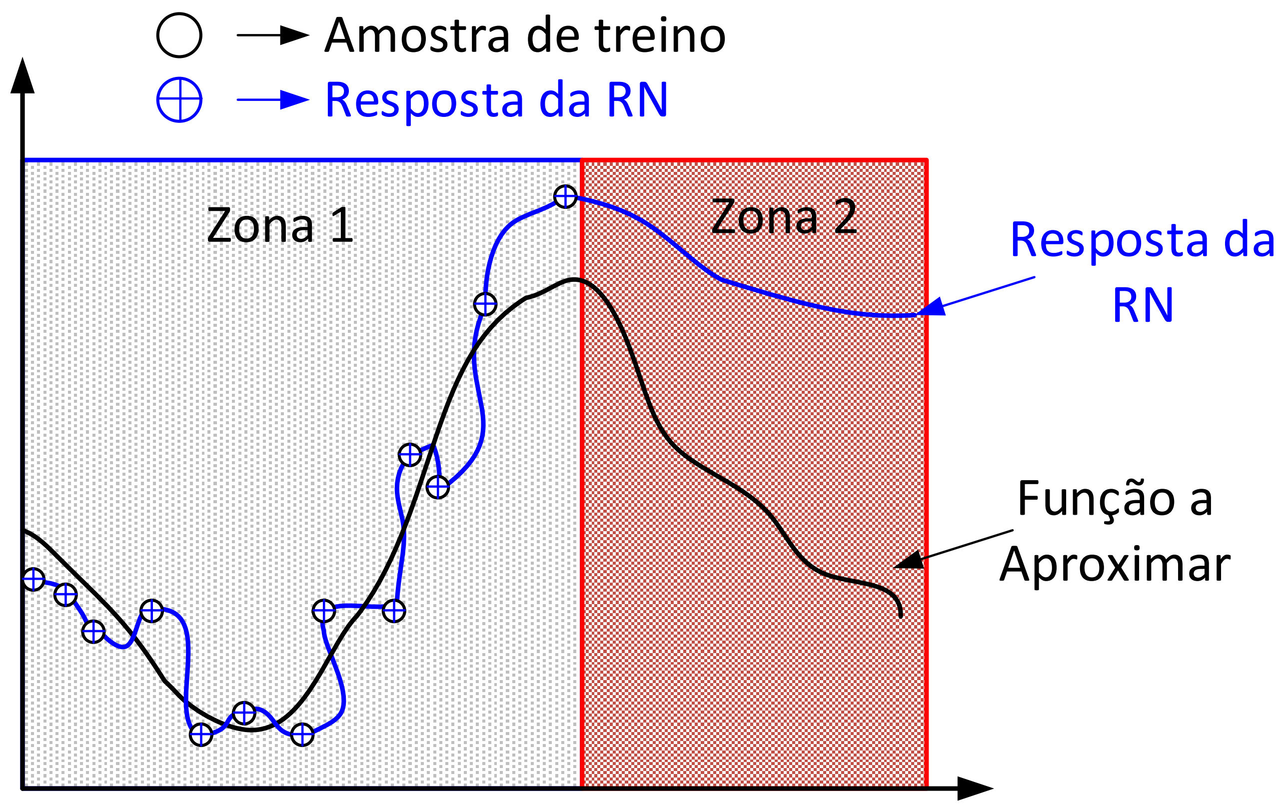
- 1 – y = [0.9998   0.0001   0.0001]
- 3 – y = [0.00028   0.9996   0.0002]
- 5 – y = [0.00000   0.00022 0.9997]

Overfitting

- No seguinte exemplo a RN consegue replicar de forma precisa as amostra de treino. No entanto não aproxima bem a função subjacente. **Ocorreu *Overfitting*** dos dados de treino.

Problema da Zona 1

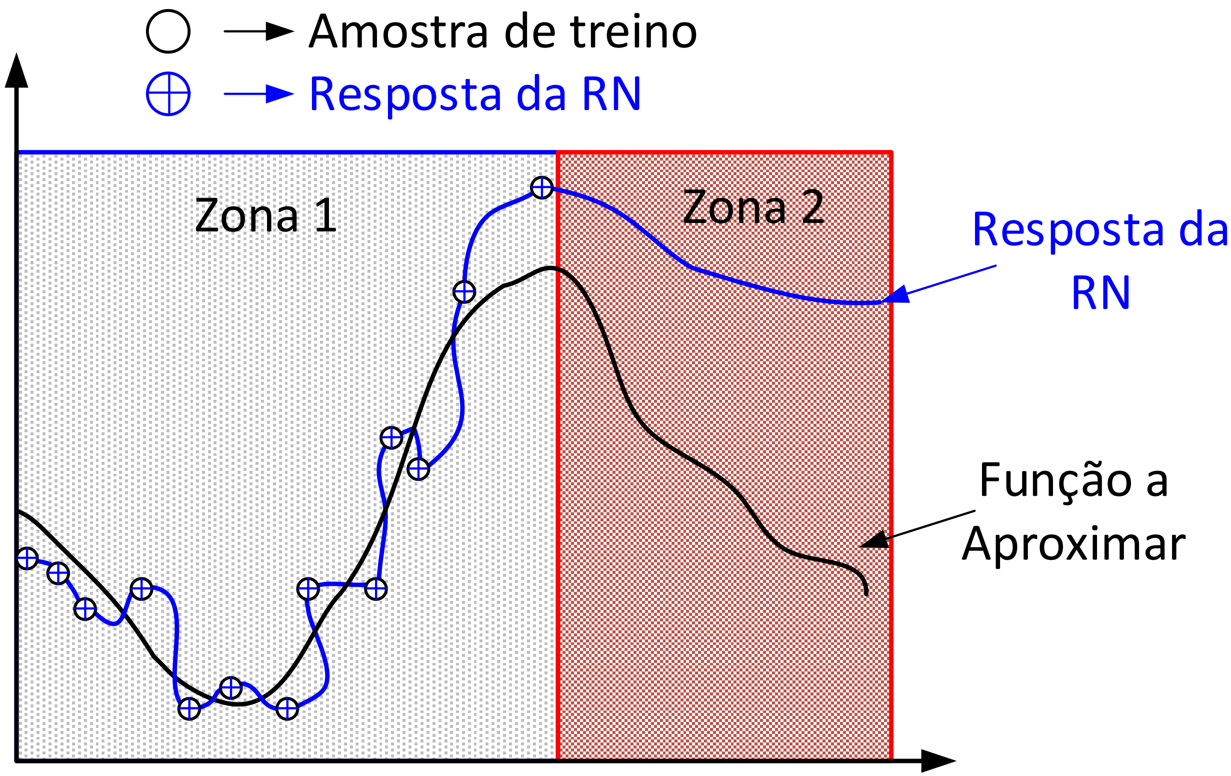
- Esta RN vai ter um **mau desempenho** para **amostras que não constem no conjunto de treino** (fraca interpolação)



Overfitting

Problema da Zona 2

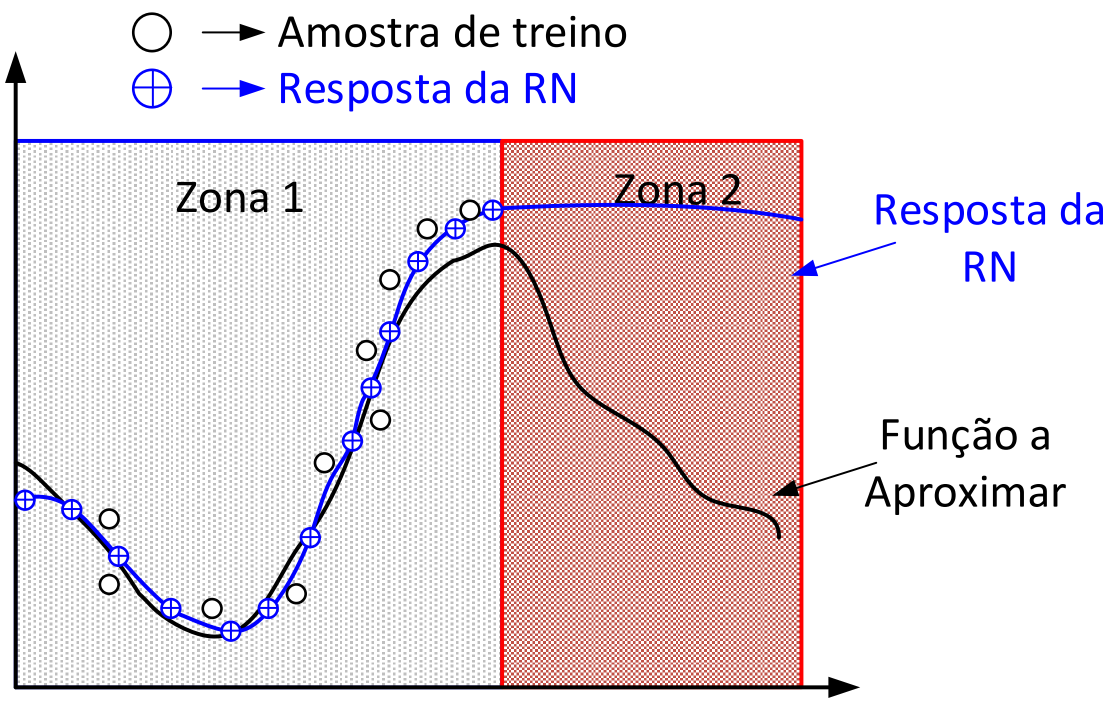
- ✓ Como não existem amostras de treino na Zona 2, **não é devido ao overfitting** que a rede não consegue **extrapolar**.



Generalização

- No seguinte exemplo a RN pode-se afirmar que houve um bom *fitting,* no entanto é impossível para a rede fazer previsões na zona 2.

- Para redes com múltiplas entradas pode ser uma tarefa difícil evitar overfitting e aumentar o **poder de generalização da RN**.



- Duas das técnica mais utilizadas para melhora a generalização consistem em:

1. **Reduzir o número de pesos da** RN (ou número de neurónios
2. **Restringir a amplitude dos pesos.**

Generalização

- Em principio só existem problemas de Overfitting quando o número de amostras de treino, N, é limitada.

- Uma regra de uso comum é a seguinte:

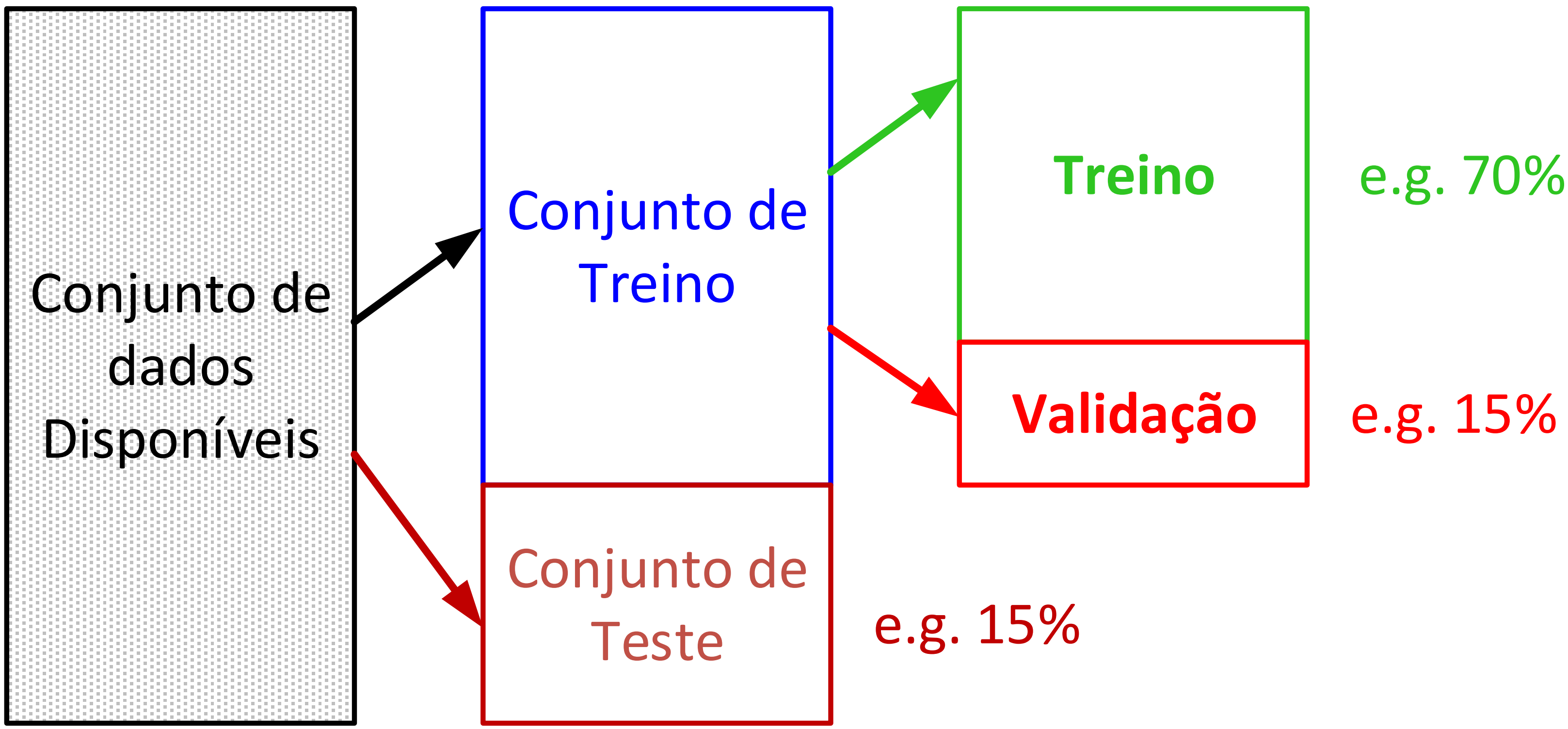
$$N = 10 \times número\ total\ de\ pesos\ da\ RN$$

- Se o conjunto de dados de treino for representativo de todas as condições (situações) utilizadas no conjunto de teste, é expectável classificar os dados de teste com 90% de acurácia.

Early Stopping Criterion:

Um dos métodos mais conhecidos para aumentar a capacidade de generalização da RN consiste em parar o treino no ponto de **máxima generalização da RN.**

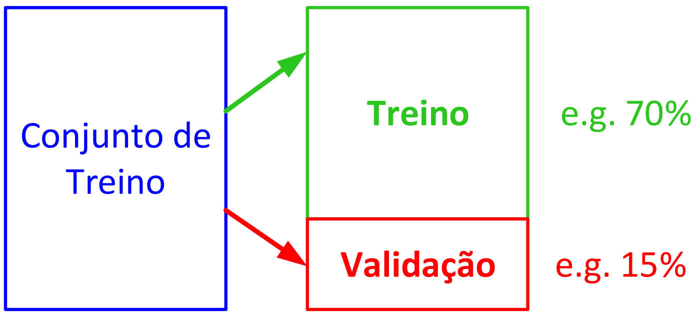
✓ É importante frisar mais uma vez que o **conjunto de dados que é utilizado para testar a rede a RN não pode ser utilizado no treino e aprendizagem da mesma**.

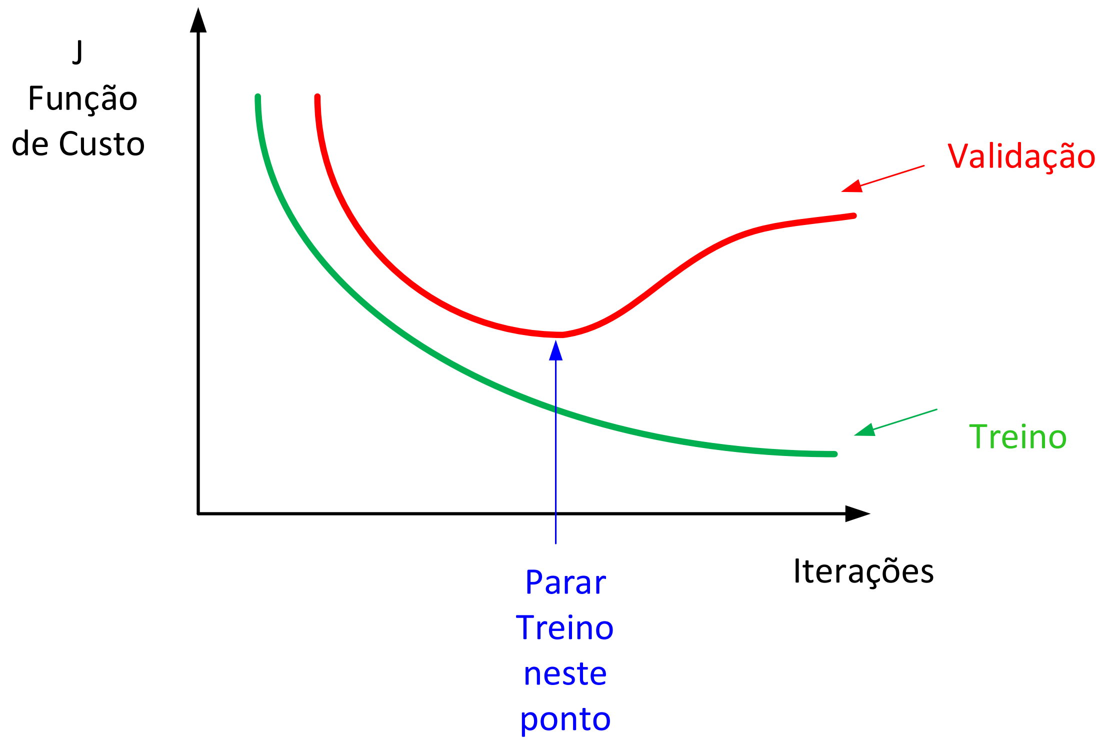


Generalização

Early Stopping Criterion and Cross Validation

- O conjunto de treino é sub-dividido em dois conjuntos: treino e validação.





- Existem vária formas de ir testando a RN usando dados do conjunto de validação. Técnicas de ***Cross-Validation (e.g. K-fold)***.

Regularização

- A regularização é uma técnica com o fim de prevenir o Overfitting. De uma forma simplificada consiste em alterar a função de custo adicionando uma **soma dos pesos**.

$$J = \frac{1}{2}\sum_{i=1}^{N}(t-y)^2$$

$$J = \frac{1}{2}\sum_{i=1}^{N}(t-y)^2 + \lambda\frac{1}{2}\|\omega\|^2$$

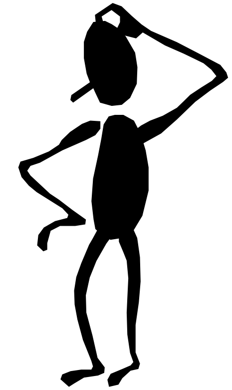
$$J = -\sum_{i=1}^{N}[t_i \log y_i + (1-t_i)\log y_i]$$

- λ - Constante de Regularização

$$J = -\sum_{i=1}^{N}[t_i \log y_i + (1-t_i)\log y_i] + \lambda\frac{1}{2}\|\omega\|^2$$

utad

Regularização



✓ Como escolher o valor da constante de regularização?

$$J = \frac{1}{2}\sum_{i=1}^{N}(t-y)^2 + \lambda\frac{1}{2}\|\omega\|^2$$

✓ Quando os pesos atingem valores suficientemente baixos, os neurónios tendem a ficar mais desconetados.

λ - Elevado → Menos ligações → Redes mais simples

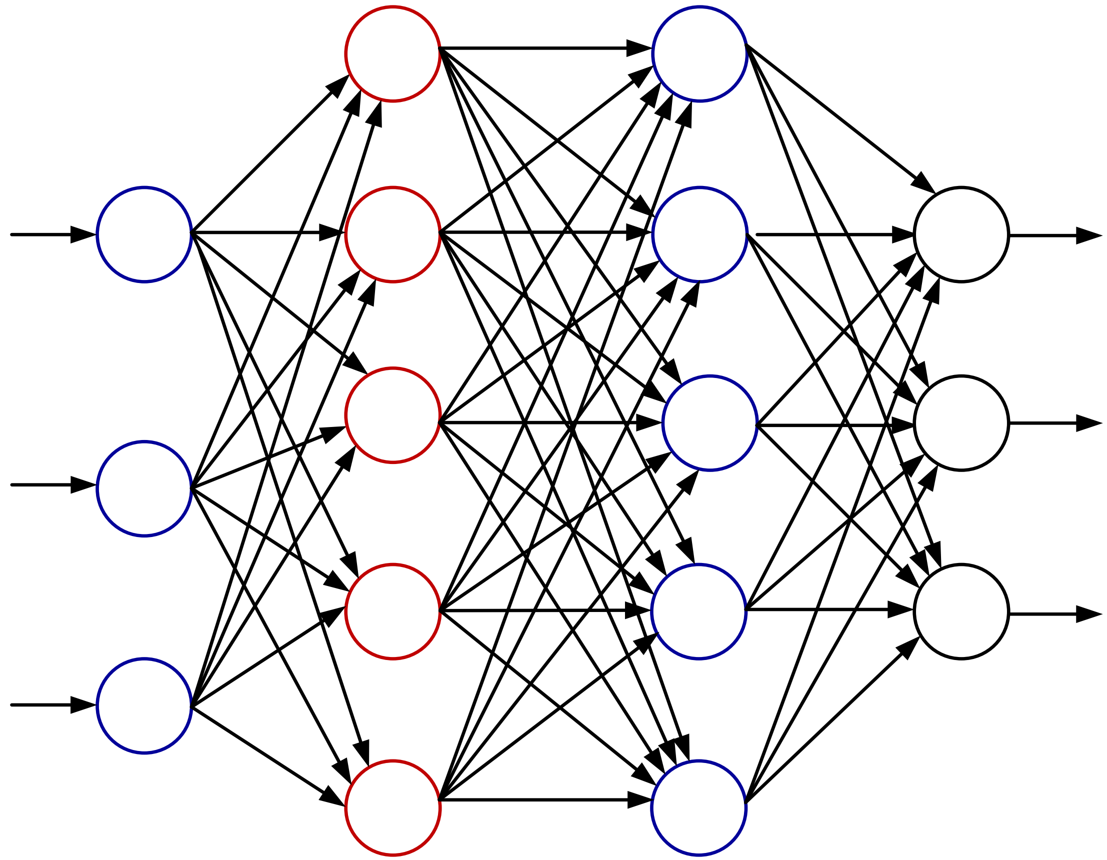
λ - Baixo → Mais ligações → Redes mais complexas

Vanishing Gradient Problem – Problema do Desaparecimento do Gradiente

- As aplicações de RN no contexto da Aprendizagem Profunda (Deep Learning) utilizam múltiplas camadas.

- Este problema ocorre quando na retropapagação do erro não chega às primeiras camadas escondidas.

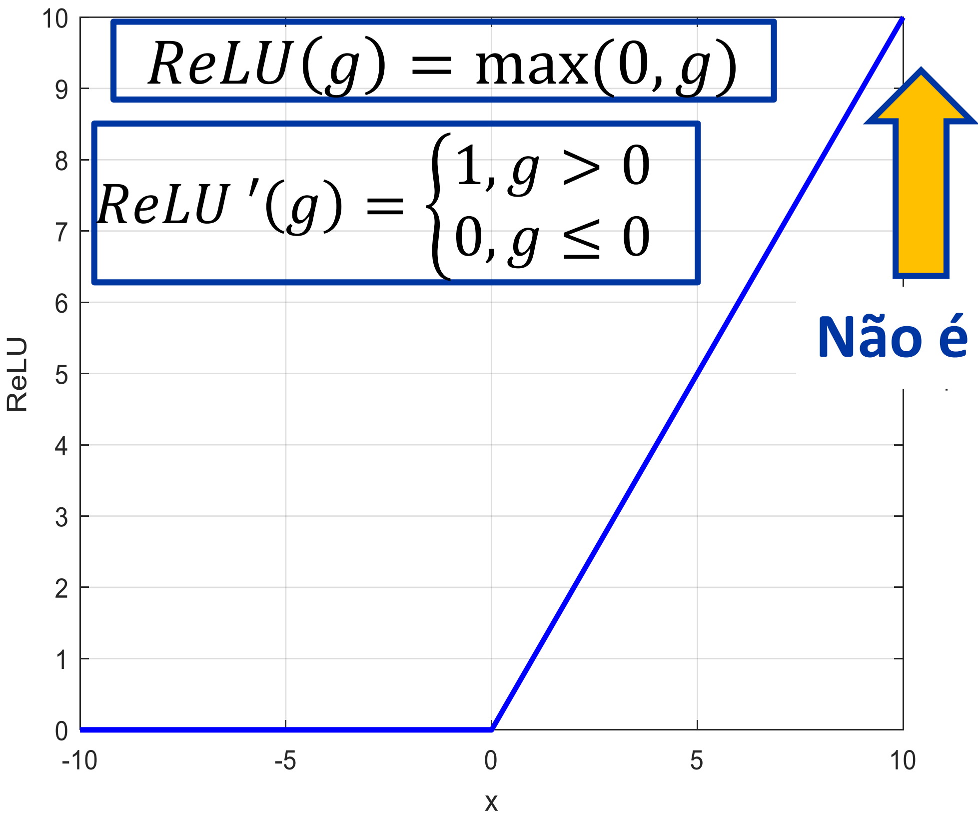
- Não faz sentido aumentar o número de camadas se não se conseguem treinar.

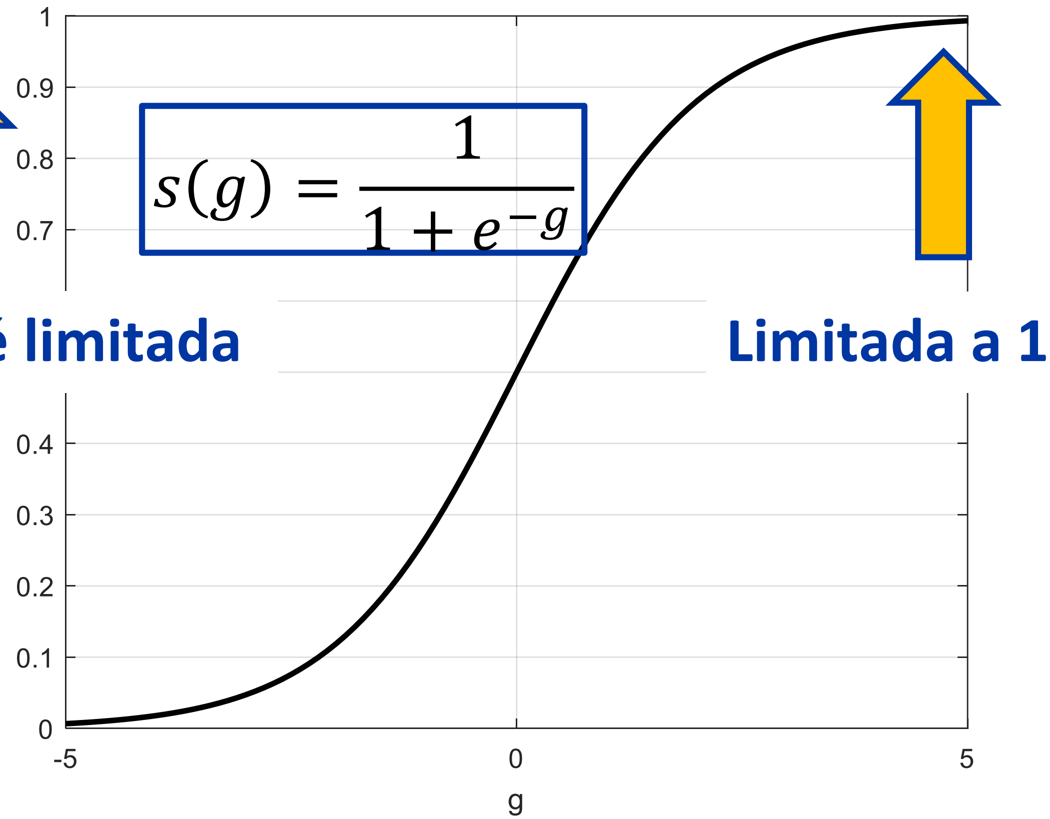


Regularização

Vanishing Gradient Problem – Problema do Desaparecimento do Gradiente

- Uma das soluções para o problema do Desaparecimento do Gradiente passa pela utilização da função **ReLU** (***Rectifier Linear Unit***):



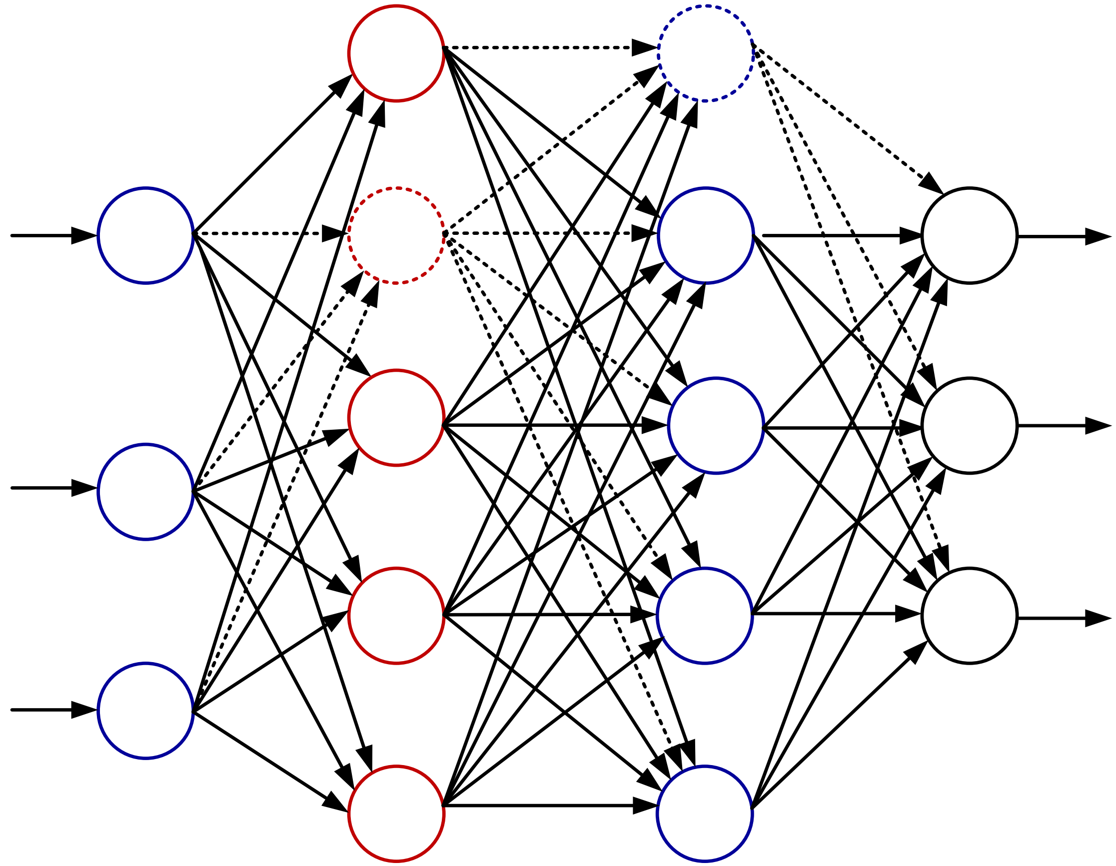


A função ReLU transmite melhor o erro que a função logística.

Regularização- Dropout

- ✓ A técnica de Dropout tem-se mostrado muito eficaz na prevenção do problema de **overfitting**.

- ✓ Esta técnica consiste em selecionar aleatoriamente um conjunto de unidades e colocar a sua saída a zero para desativar essas unidades.



Bibliografia

[1] Johnson J. and Picton P, Concepts on Artificial Intelligence, Designing Intelligent Machines, Part II, Butterworth-Heinemann

[2] Lucci S. e Kopec D., (2016), Artificial Intelligence in the 21st Century, A Living Introduction, Mercury leArning And inforMAtion.

[3] Berwick B., (2020), MIT Classes Notes. http://web.mit.edu/6.034/wwwbob/, acedido em 25-3-2020

[4] Kim P. (2017), MATLAB Deep Learning With Machine Learning, Neural Networks and Artificial Intelligence, APress

[5] Principe J. C., Euliano N. R. e Lefebvre W. C. (1999), Neural and Adaptive Systems, Fundamentals Through Simulations, John Wiley and Sons.