Redes Neurais

Prof. Tiago Buarque A. de Carvalho

Neurônio

Prof. Tiago Buarque A. de Carvalho

Inspiração

Ações simples

caminhar, dar um nó em um cadarço, reconhecer uma pessoa...

Não tão simples para um robô

Cérebro humano como inspiração para algoritmos de aprendizagem de máquina

Bioinspirado

O começo das RNs

McCulloch e Pitts (1943)

- Neurônio LTU: Logic Threshold Unit

Hebb (1949)

- Aprendizado Hebbiano

Rosenblatt (1958)

- Perceptron

Sistema nervoso biológico

Seres vertebrados

Células especiais: NEURÔNIOS

- Responde a estímulos internos e externos
- Transmissão de impulsos a outro neurônios e células

Cérebro humano: 10 a 500 bilhões de neurônios

Um modelo de neurônio

Dendritos

recebem o sinal

Corpo celular

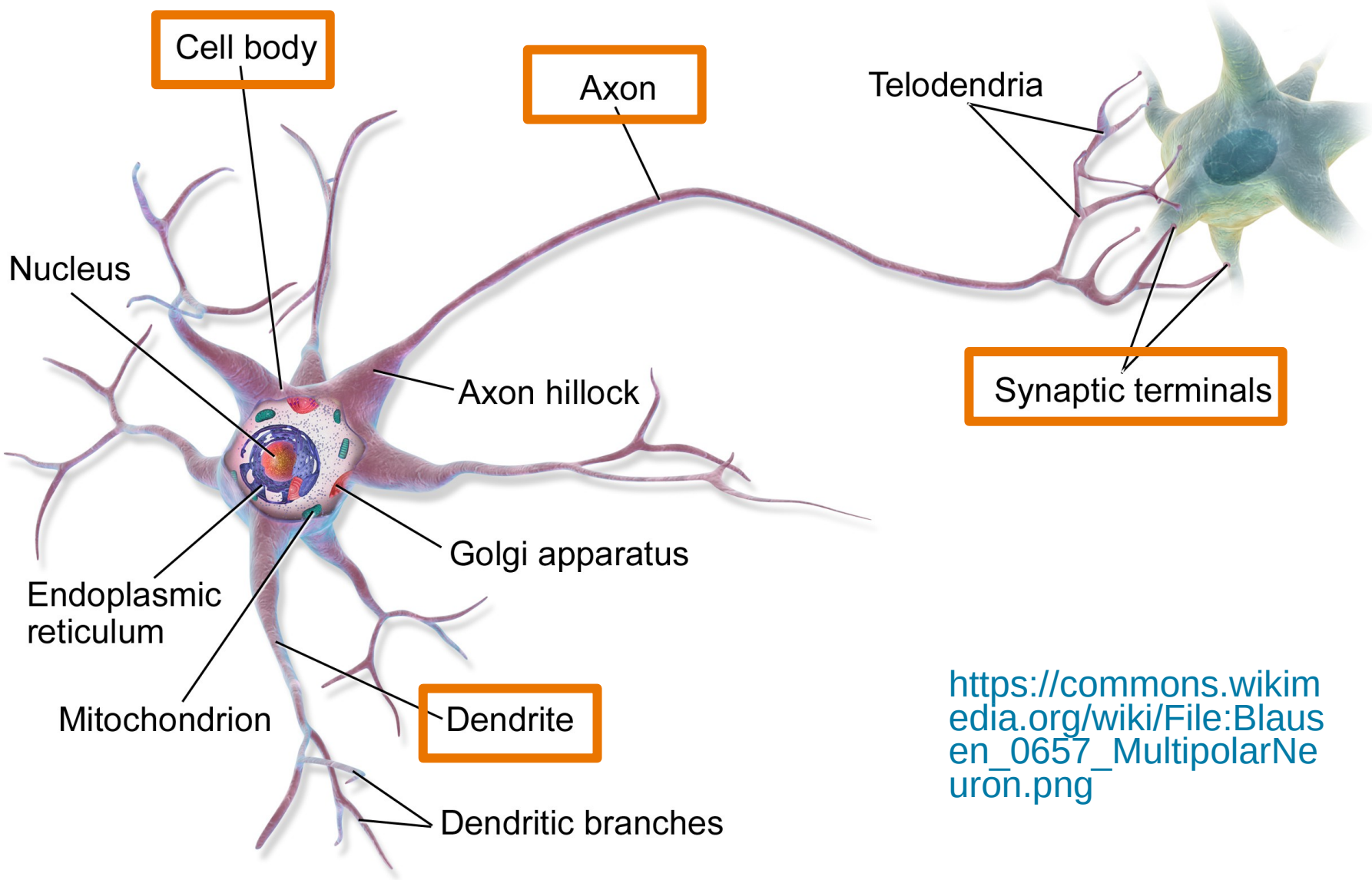
acumula o sinal, dispara um impulso para o axônio quando ultrapassa o limiar de ativação

Axônio

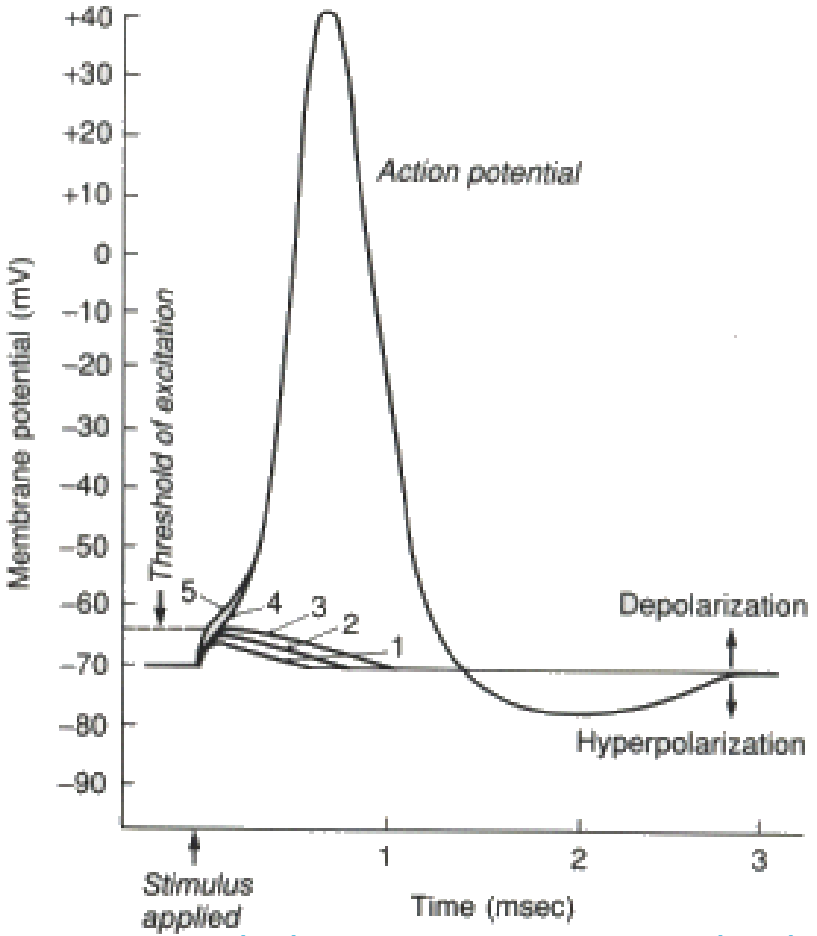
transmite o impulso para dendritos de outros neurônios

Sinapses

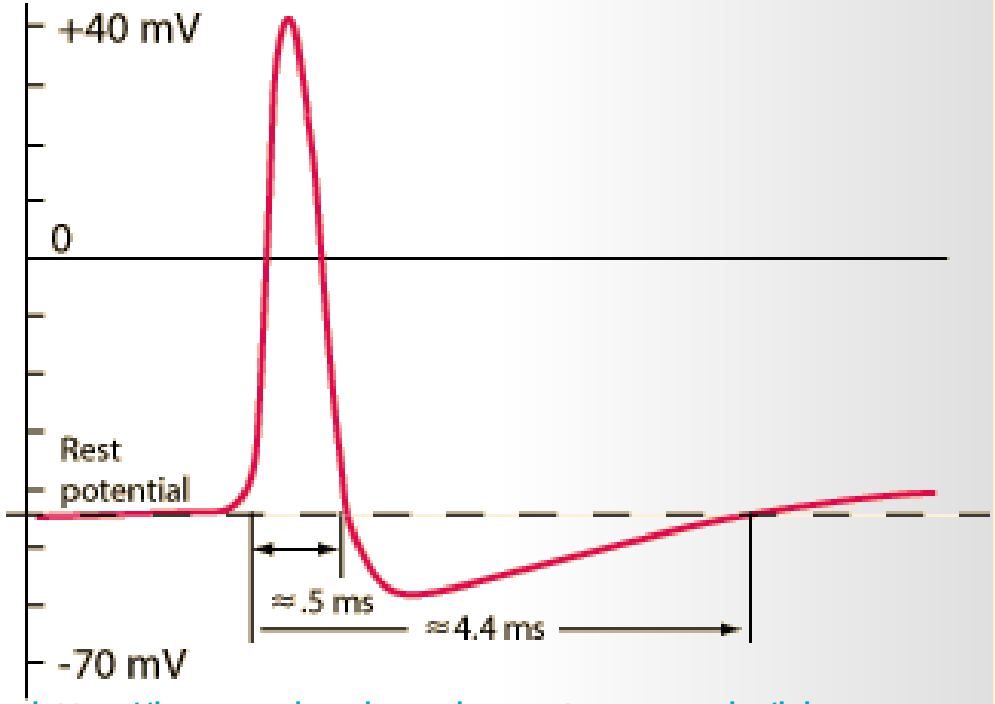
junção entre axônio e dendritos, pode ser excitatória ou inibitória



https://commons.wikimedia.org/wiki/File:Blausen_0657_MultipolarNeuron.png



http://www.mindcreators.com/neuronbasics.htm



http://hyperphysics.phy-astr.gsu.edu/hbase/biology/actpot.html

Ativação

Neurônio McCulloch-Pitts

O corpo do neurônio

- acumulador = somatório

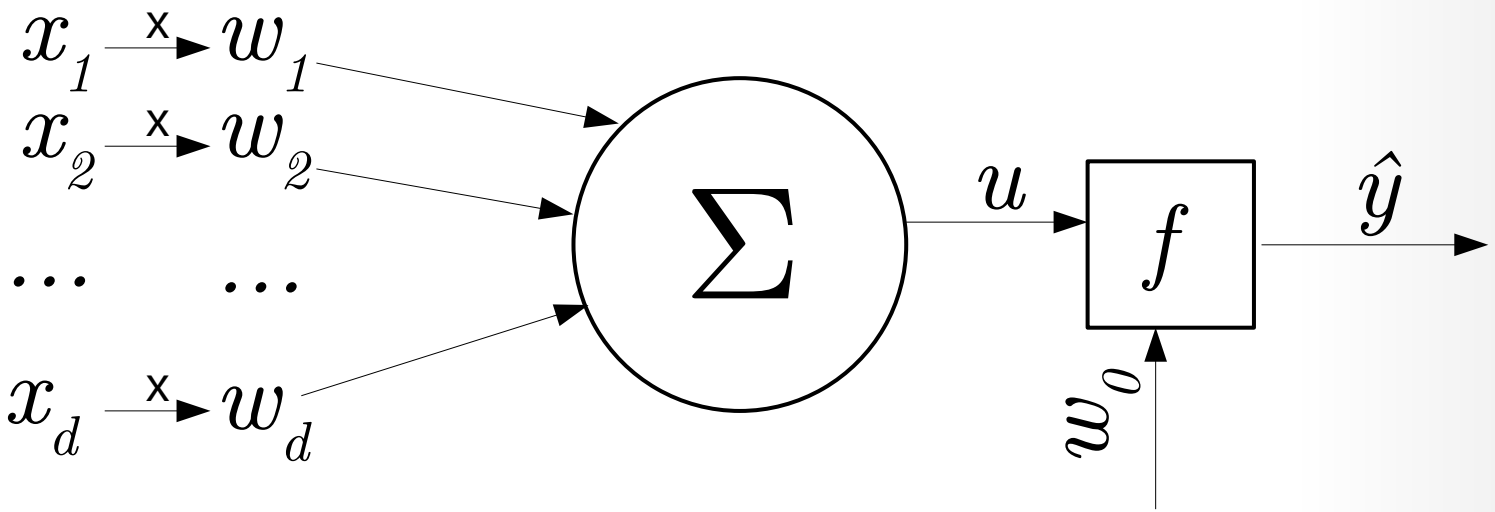
Sinapses excitatórias ou inibitórias

- pesos positivos ou negativos

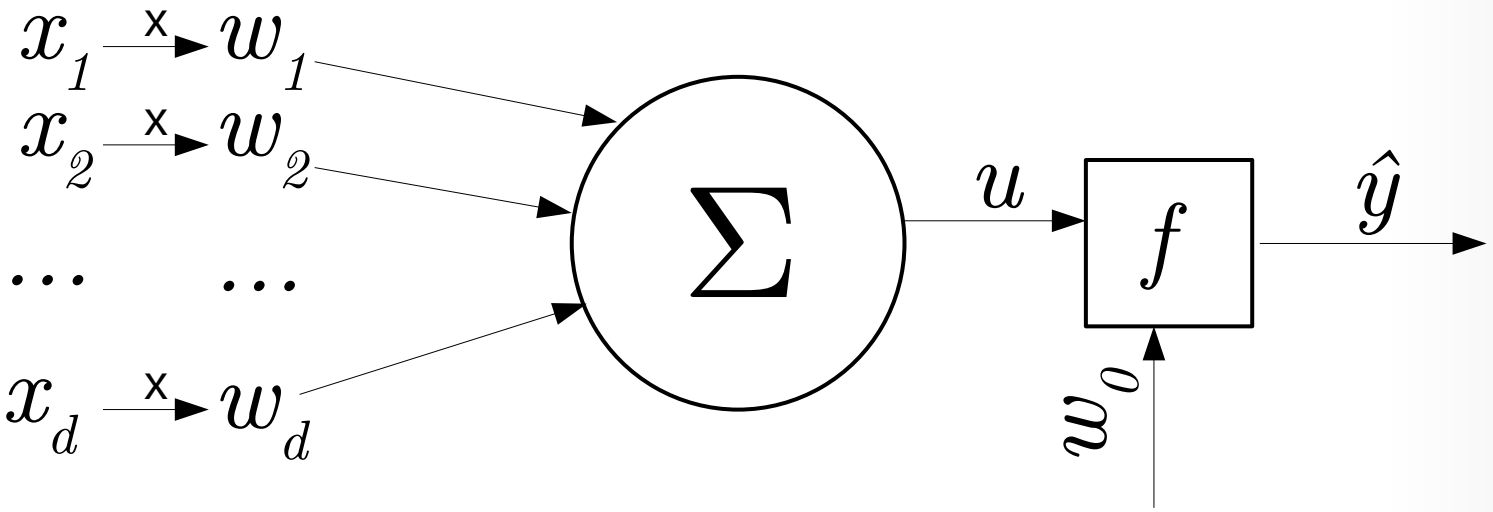
Ativação

- função limiar

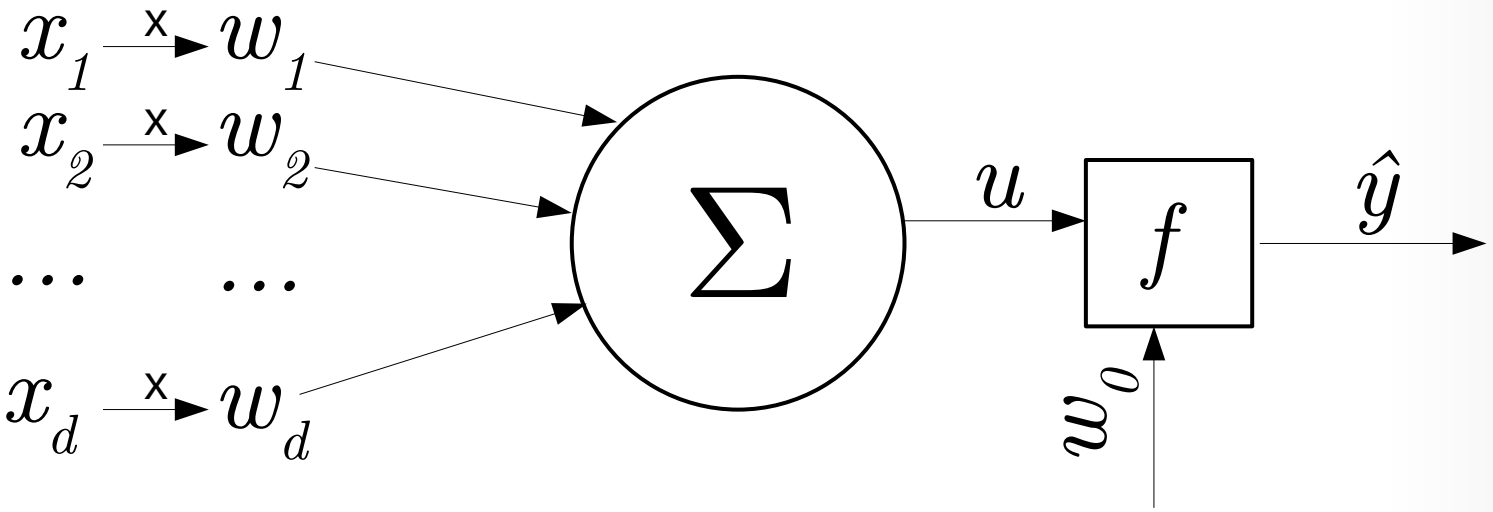
Neurônio McCulloch-Pitts



$$u = \sum_{i=1}^{d} x_i w_i$$



$$f(u, w_0) = \begin{cases} 1, u > w_0 \\ 0, u \leq w_0 \end{cases}$$

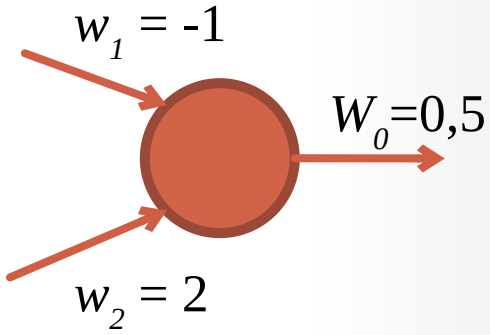


Qual a saída do neurônio?

Entrada:
(x1 = 0,6; x2 = 0,5)

U = 0,6*(-1) + 0,5*2
= -0,6 + 1
= 0,4

f(0,4; 0,5) = 0

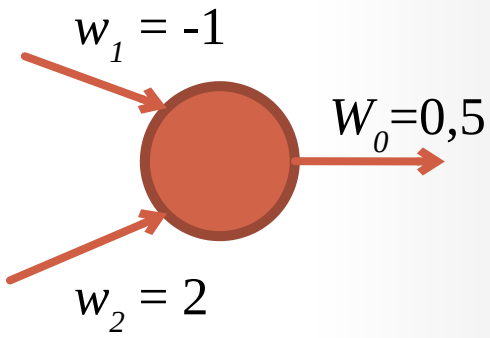


$$u = \sum_{i=1}^{d} x_i w_i$$

$$f(u, w_0) = \begin{cases} 1, u > w_0 \\ 0, u \leq w_0 \end{cases}$$

Qual a saída do neurônio?

Entrada:
(0,6, 0,7)



u = 0,6*(-1) + 0,7*2
= -0,6 + 1,4
= 0,8

f(0,8, 0,5) = 1

$$u = \sum_{i=1}^{d} x_i w_i$$

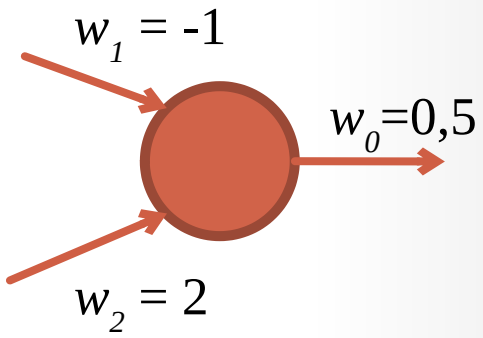
$$f(u, w_0) = \begin{cases} 1, u > w_0 \\ 0, u \leq w_0 \end{cases}$$

Qual a saída do neurônio?

Entrada:
(1,0; 0,5)

u = 1*(-1) + 0,5*2
= -1 + 1
= 0

f(0,0; 0,5) = 0

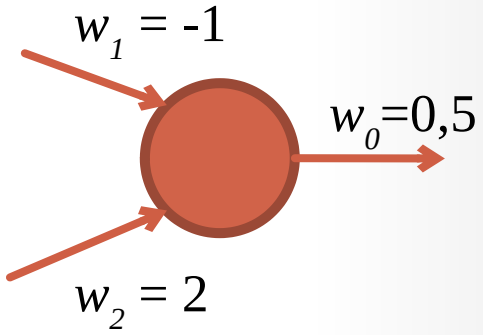


$$u = \sum_{i=1}^{d} x_i w_i$$

$$f(u, w_0) = \begin{cases} 1, u > w_0 \\ 0, u \leq w_0 \end{cases}$$

Qual a saída do neurônio?

Entrada:
(a, b)



u = a*(-1) + b*2
= -a + 2b
= 0

$$u = \sum_{i=1}^{d} x_i w_i$$

f(-a+2b; 0,5) = 1, se -a+2b > 0,5
0, caso contrário

$$f(u, w_0) = \begin{cases} 1, u > w_0 \\ 0, u \leq w_0 \end{cases}$$

Treinamento de Neurônio

Prof. Tiago Buarque A. de Carvalho

Superfície de Decisão

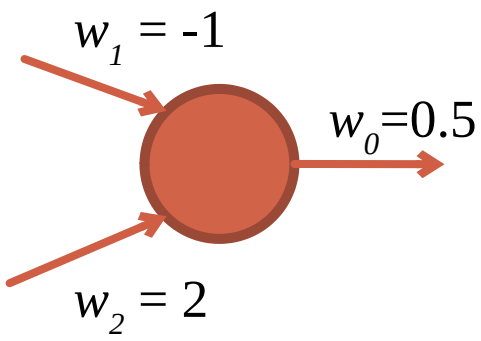
Os neurônios resolvem problemas

“LINEARMENTE SEPARÁVEIS”

A superfície de decisão é um hiperplano

- Em 2D é uma reta
- Em 3D é um plano

Qual a superfície de decisão deste neurônio?



Entrada:
(a, b)

u = a*(-1) + b*2
= -a + 2b
= 0

f(-a+2b; 0,5) = 1, se -a+2b > 0,5
0, caso contrário

$$f(-a+2b;\ 0{,}5) = \begin{cases} 1, & \text{se } -a+2b > 0{,}5 \\ 0, & \text{se } -a+2b \leq 0{,}5 \end{cases}$$

Qual a linha que separa?

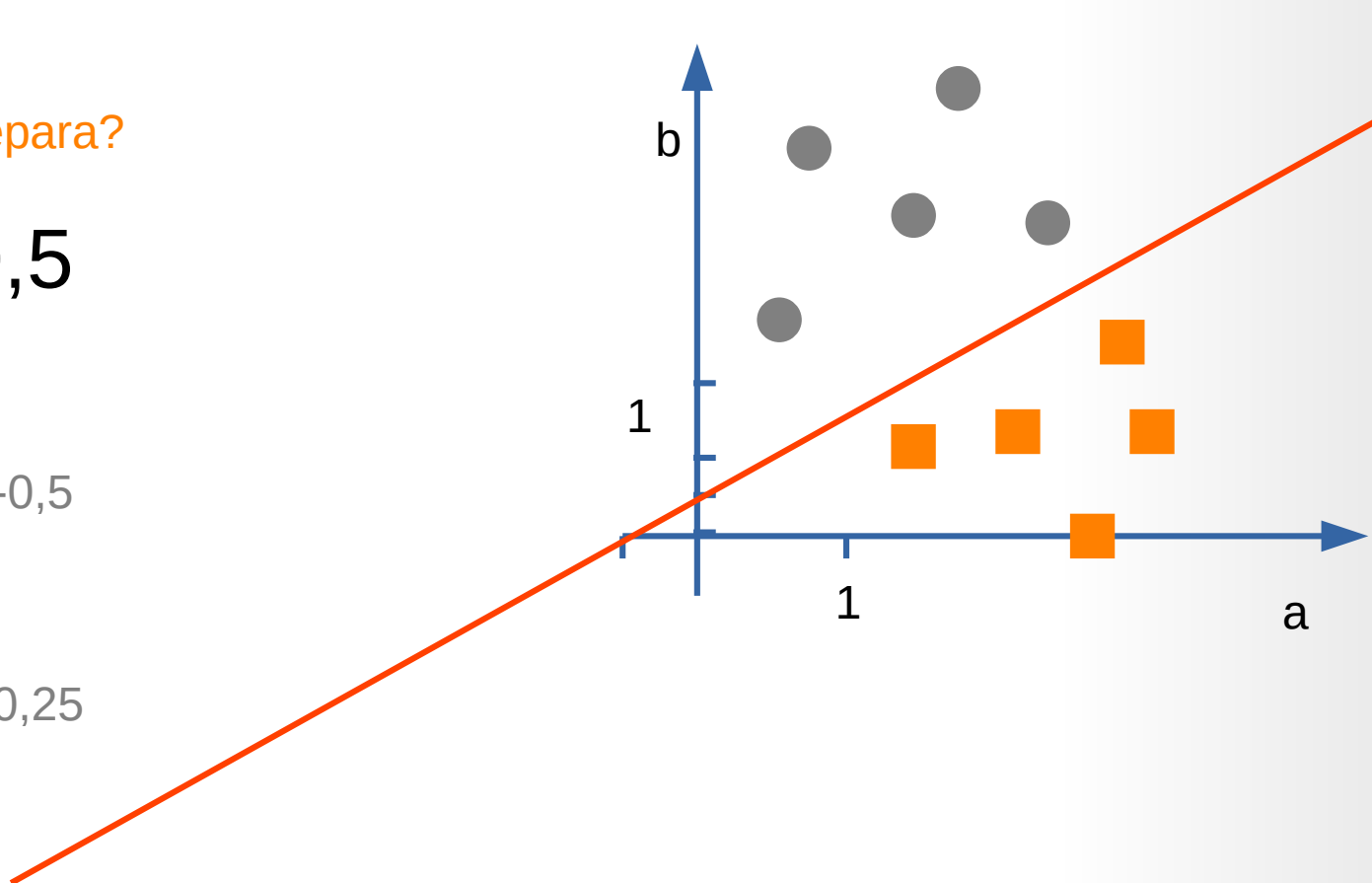
$$-a + 2b = 0{,}5$$

$a = 2b - 0{,}5$

se $b = 0$ então $a = -0{,}5$

$b = (0{,}5 + a)/2$

se $a = 0$ então $b = 0{,}25$



Bias = viés

É o limiar da função de ativação do neurônio

A função de ativação pode ser simplificada se o bias for inserido no somatório

Como?

Fazendo $x_0 = -1$

$$f(u, w_0) = \begin{cases} 1, u > w_0 \\ 0, u \leq w_0 \end{cases} = \begin{cases} 1, u - w_0 > 0 \\ 0, u - w_0 \leq 0 \end{cases}$$

$$u - w_0 = \sum_{i=1}^{d} x_i w_i - w_0 = \sum_{i=0}^{d} x_i w_i, x_0 = -1$$

$$v = u - w_0$$

$$x_0 = -1$$

$$v = \sum_{i=0}^{d} x_i w_i$$

$$u = \sum_{i=1}^{d} x_i w_i$$

$$f(v) = \begin{cases} 1, v > 0 \\ 0, v \leq 0 \end{cases}$$

$$f(u, w_0) = \begin{cases} 1, u > w_0 \\ 0, u \leq w_0 \end{cases}$$

Algoritmo de treinamento

Entrada: *n* exemplos de treino $\{(\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_n, y_n)\}$; taxa de aprendizagem η

Saída: neurônio treinado, $\mathbf{w} = (w_0, \ldots, w_d)$

1. Inicializa o neurônio com valores aleatórios no intervalo [0;1]
2. erroTotal = ε+1
3. Enquanto (erroTotal > ε)
   A. Para cada exemplo x_i, com i = 1, …, n, faça
      I. erro = $(y_i - f(u_i))^2$
      II. Se erro > 0
          Para cada peso w_j, com j = 1, … d, faça atualização dos pesos $\mathbf{w_j[t+1] = w_j[t] + \eta x_{ij}(y_i - f(u_i))}$
   B. erroTotal = 0
   C. Para cada exemplo x_i, com i = 1, …, n, faça
      I. erro = $(y_i - f(u_i))^2$
      II. erroTotal = erroTotal + erro

$$w_j[t+1] = w[t] + \eta x_{ij}(y_i - f(u_i))$$ equação de atualização dos pesos

$w_j[t+1]$ peso APÓS a atualização

$w_j[t]$ peso ANTES da atualização

η taxa de aprendizagem

$x_{ij}, \mathbf{x}_i = (x_{i1}, \ldots, x_{id})$ valor da característica j do exemplo i, exemplo i

y_i classe do exemplo i

$f(u_i)$ saída do neurônio para o exemplo i

```
static double saidaNeuronio(double w[], double exemplo[]) {

double v = 0;

for (int i = 0; i < exemplo.length; i++) {
    v = v + w[i] * exemplo[i];
}

if (v > 0)
    return 1;
else
    return 0;

}
```

```
static void treinamento(double[] y, double[][] x, double eta, int n, int d, double[] w, double erroMax) {
        double erroTotal = erroMax;
        int epoca = 0;
        int atualizacao = 0;
        printSuperficieDecisao(300, -1, 2, -1, 2, w, x, y, 10, "Início");
        while (erroTotal >= erroMax) {
            epoca++;
            System.out.println("\nÉpoca #" + epoca);
            for (int i = 0; i < n; i++) {
                double f = saidaNeuronio(w, x[i]);
                double delta = y[i] - f;
                double erro = delta * delta;
                erroTotal = erroTotal + erro;
                System.out.print("y[" + i + "]=" + y[i] + ", f=" + f);
                if (erro > 0) {
                    atualizacao++;
                    for (int j = 0; j < d; j++) {
                        w[j] = w[j] + eta * x[i][j] * delta;
                        System.out.print(", w[" + j + "]=" + w[j]);
                    }
                    printSuperficieDecisao(300, -1, 2, -1, 2, w, x, y, 10, "Atualização #" + atualizacao);
                }
                System.out.println();
            }

            erroTotal = 0;
            for (int i = 0; i < n; i++) {
                double f = saidaNeuronio(w, x[i]);
                double delta = y[i] - f;
                double erro = delta * delta;
                erroTotal = erroTotal + erro;
            }
            System.out.println("Erro Total = " + erroTotal);
        }
    }
```

Atualiza os pesos

Calcula o erro do neurônio

Exemplo de Treinamento

X1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1
OU		

x0 = -1

x0	x1	x2	y
-1	0	0	0
-1	0	1	1
-1	1	0	1
-1	1	1	1

Pesos iniciais

w0 = 0
w1 = 0
w2 = 0

η = 0,5

x0	x1	x2	y
-1	0	0	0
-1	0	1	1
-1	1	0	1
-1	1	1	1

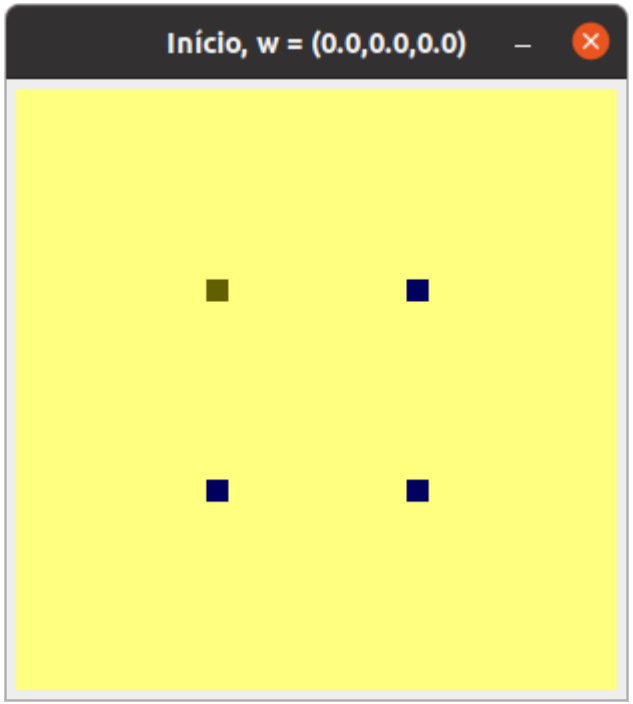
```
Época #1
y[0]=0.0, f=0.0
y[1]=1.0, f=0.0, w[0]=-0.5, w[1]=0.0, w[2]=0.5
y[2]=1.0, f=1.0
y[3]=1.0, f=1.0
Erro Total = 1.0
```

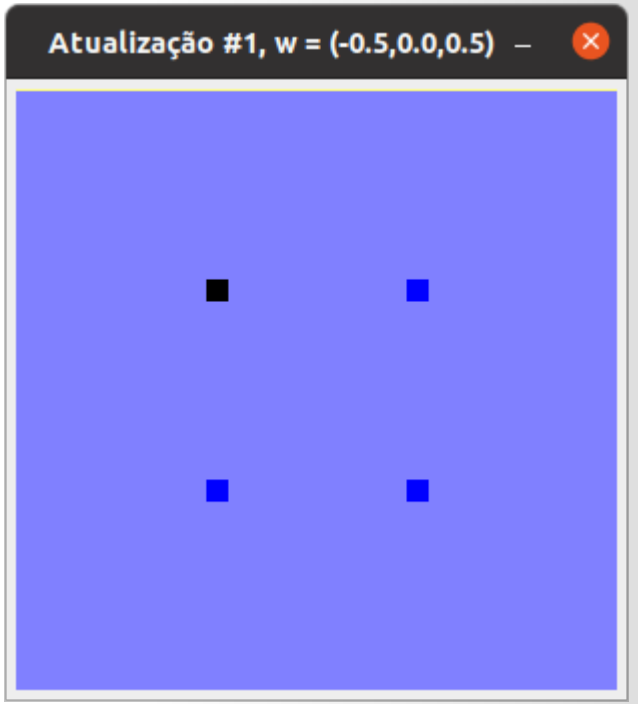
Pesos iniciais

w0 = 0
w1 = 0
w2 = 0

η = 0,5

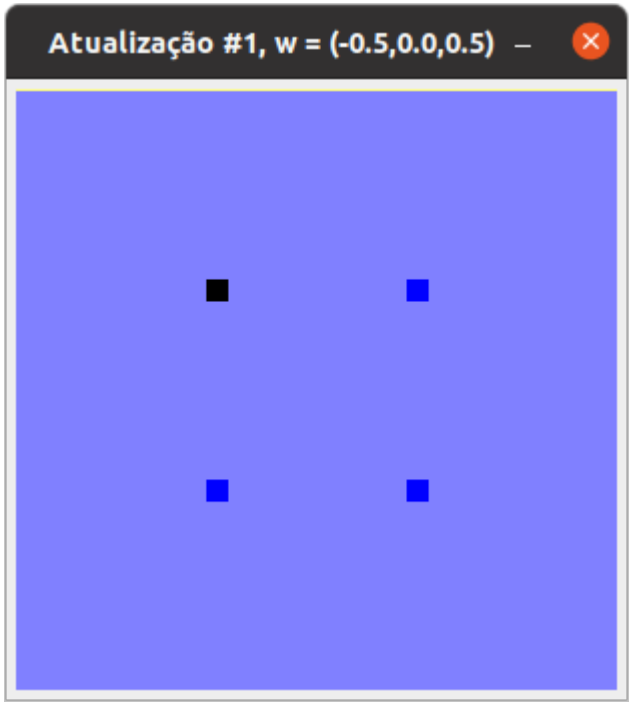
Erro Máximo = 0.001

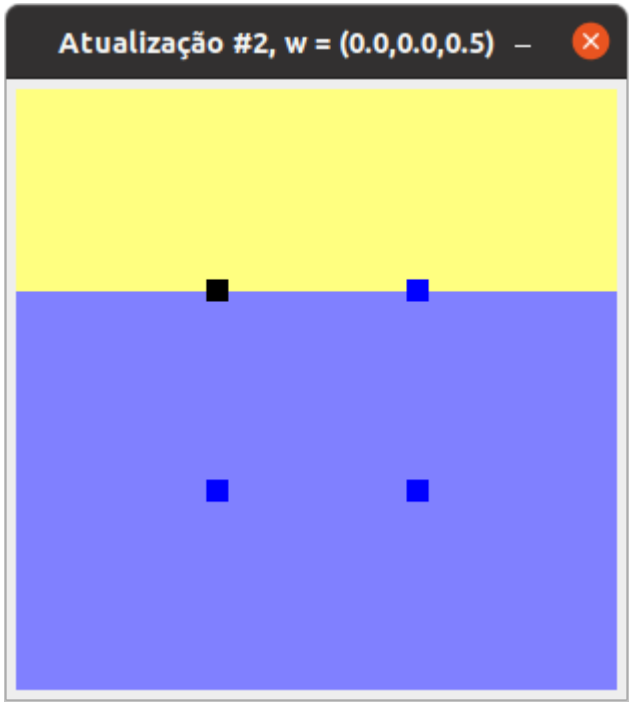


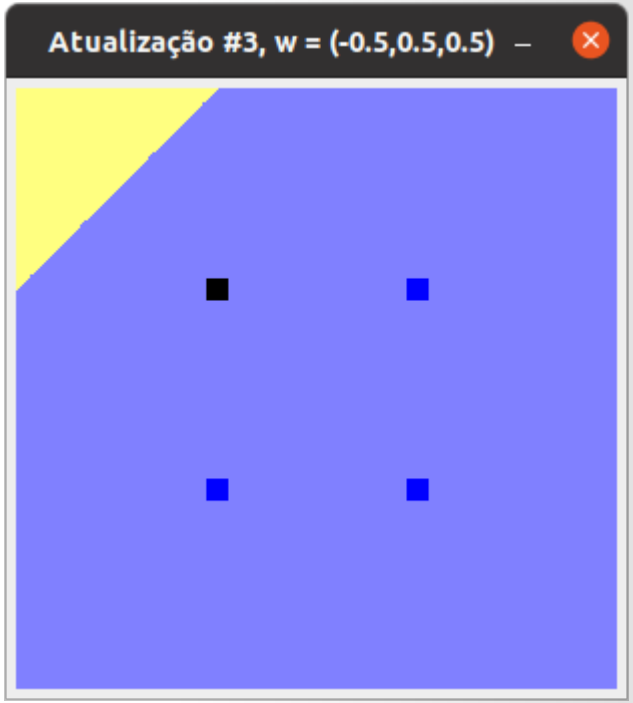


x0	x1	x2	y
-1	0	0	0
-1	0	1	1
-1	1	0	1
-1	1	1	1

```
Época #2
y[0]=0.0, f=1.0, w[0]=0.0, w[1]=0.0, w[2]=0.5
y[1]=1.0, f=1.0
y[2]=1.0, f=0.0, w[0]=-0.5, w[1]=0.5, w[2]=0.5
y[3]=1.0, f=1.0
Erro Total = 2.0
```



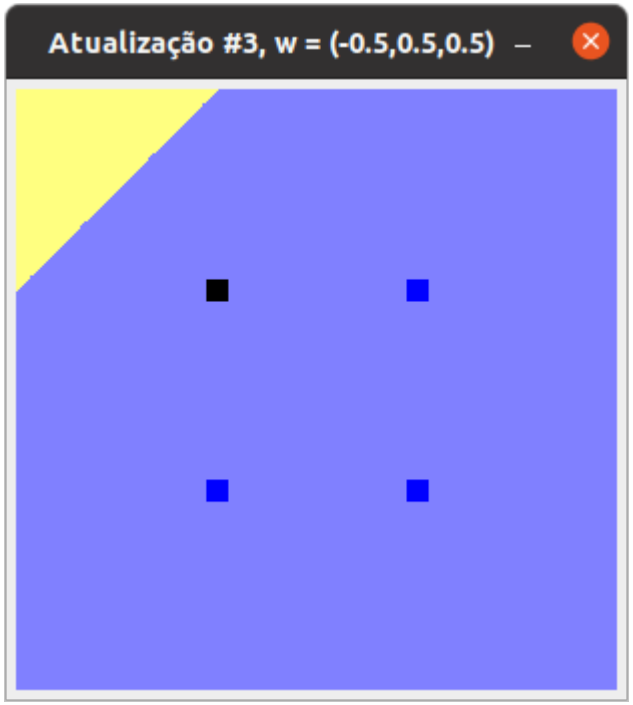


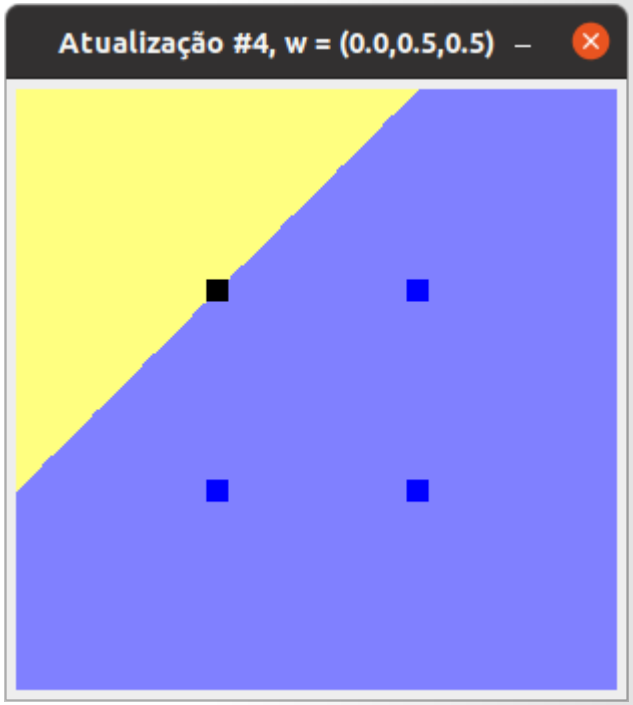


x0	x1	x2	y
-1	0	0	0
-1	0	1	1
-1	1	0	1
-1	1	1	1

Época #3

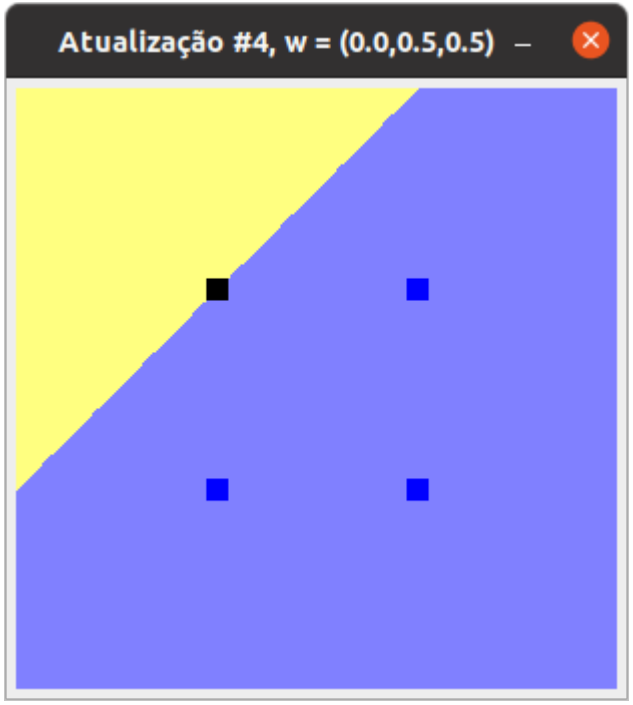
```
y[0]=0.0, f=1.0, w[0]=0.0, w[1]=0.5, w[2]=0.5
y[1]=1.0, f=1.0
y[2]=1.0, f=1.0
y[3]=1.0, f=1.0
Erro Total = 1.0
```





x0	x1	x2	y
-1	0	0	0
-1	0	1	1
-1	1	0	1
-1	1	1	1

```
Época #4
y[0]=0.0, f=0.0
y[1]=1.0, f=1.0
y[2]=1.0, f=1.0
y[3]=1.0, f=1.0
Erro Total = 0.0
```



Pesos iniciais

w0 = 0
w1 = 0
w2 = 0

η = 0,5

Erro Máximo = 0.001

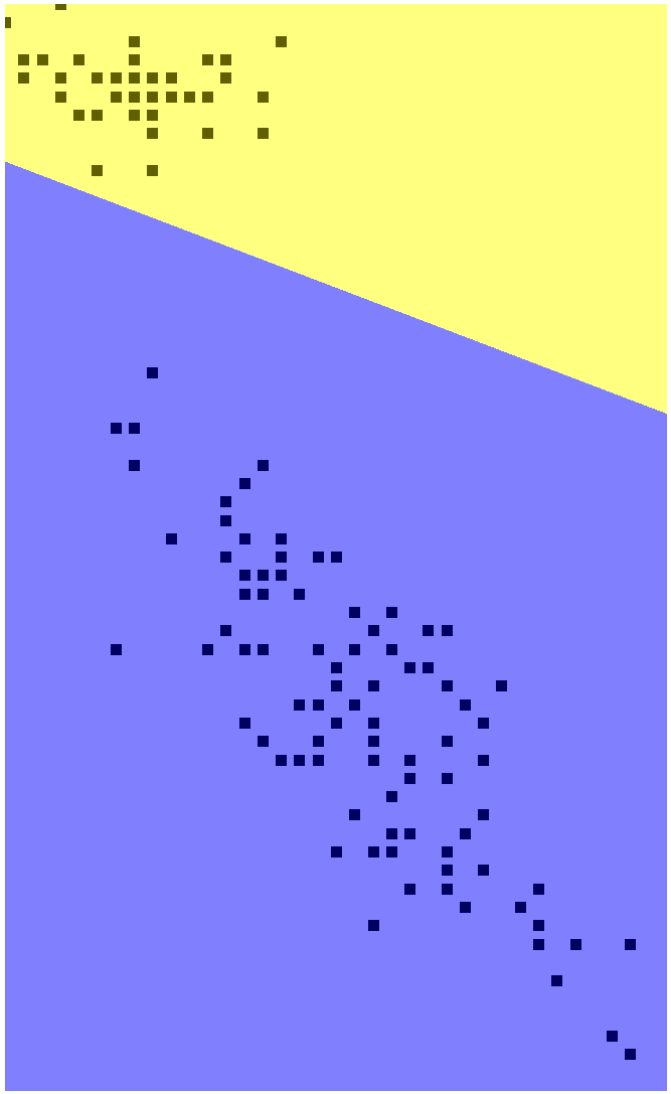
Exemplo na Base Iris

Dados

- “Iris-setosa” é a classe 0
- Os demais exemplos são da classe 1
- Utilizando a 1ª e 3ª colunas

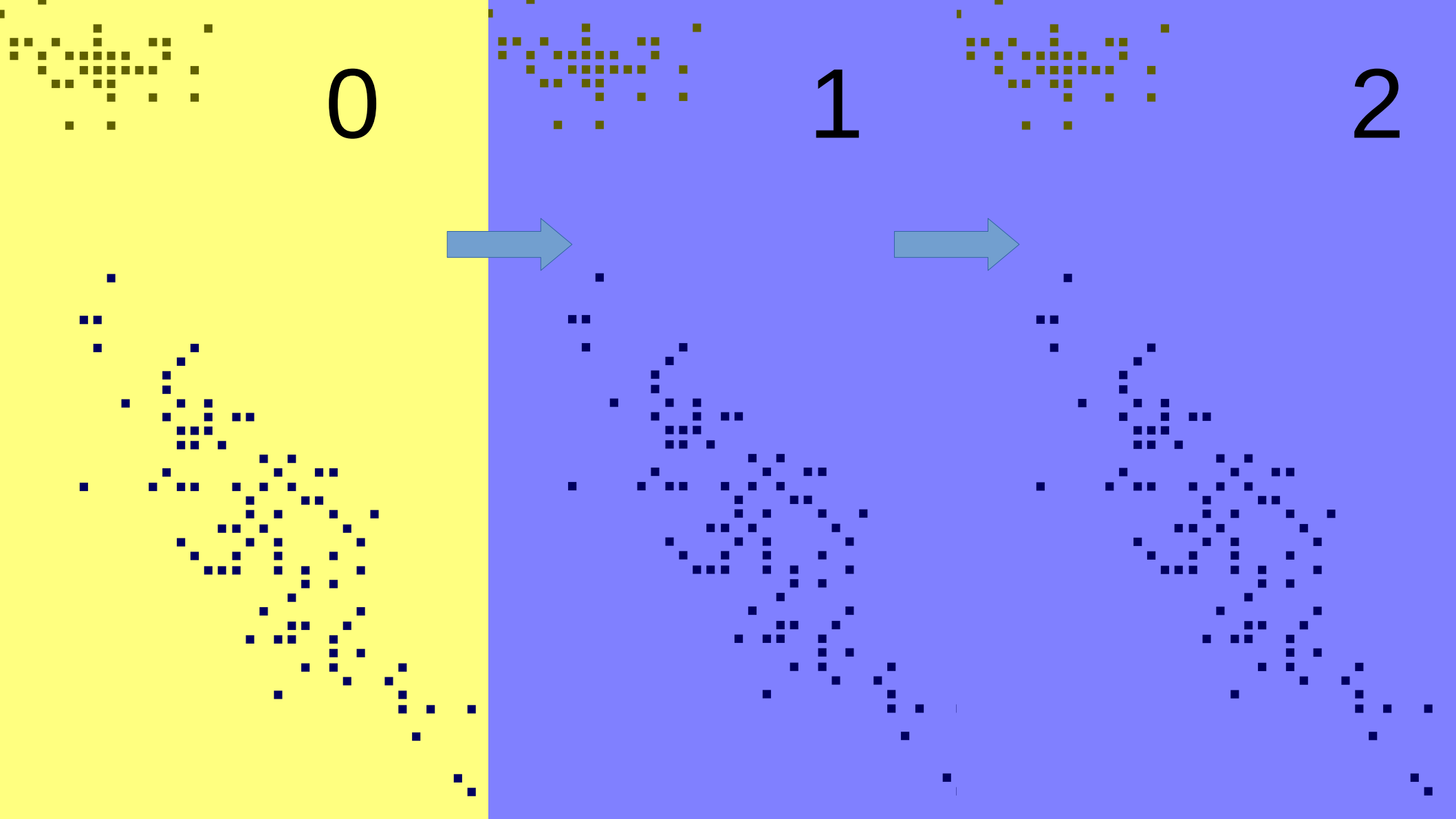
Parâmetros

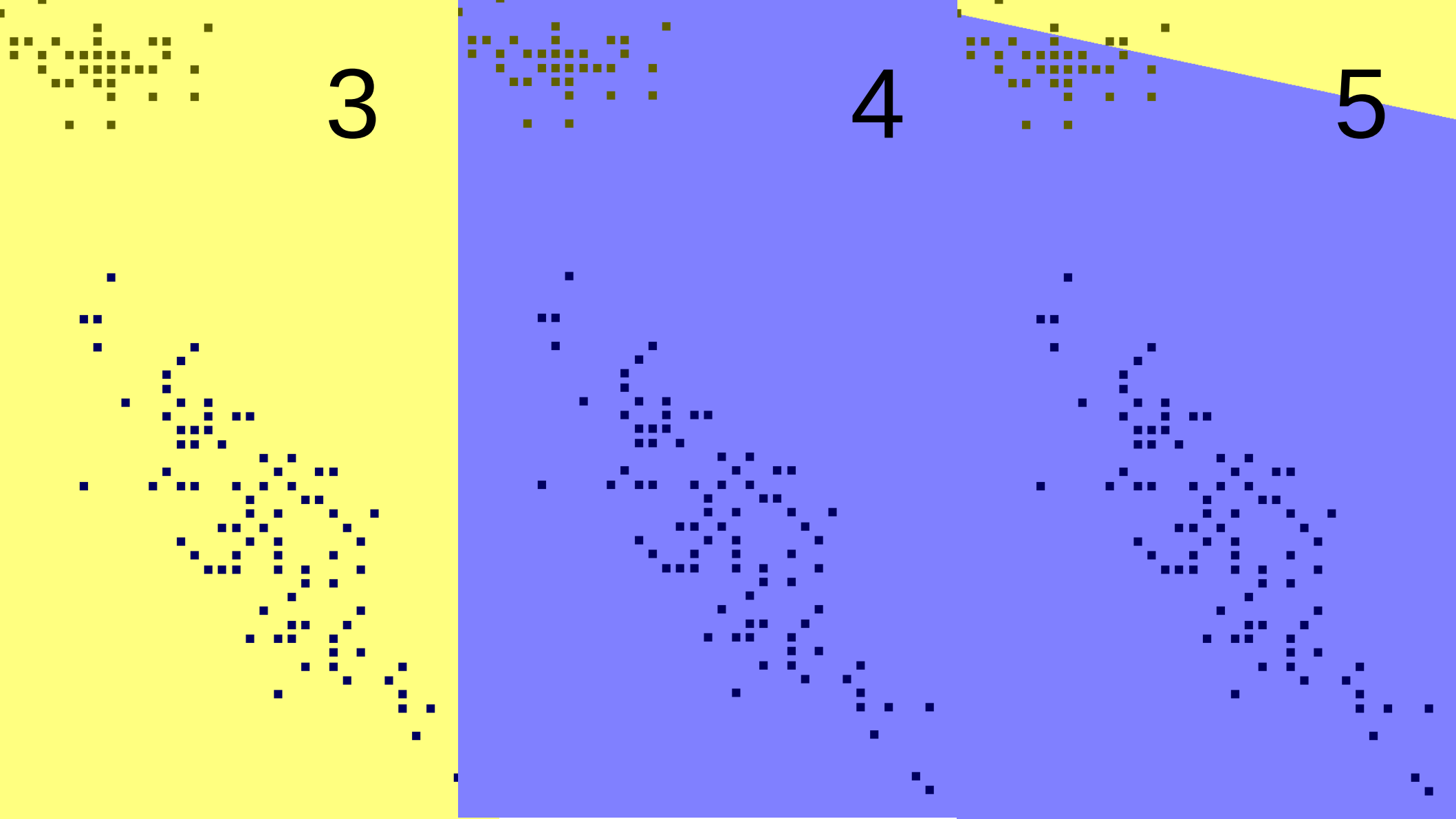
- Taxa de aprendizagem = 1,0
- Erro máximo = 1
- Pesos iniciais = 0

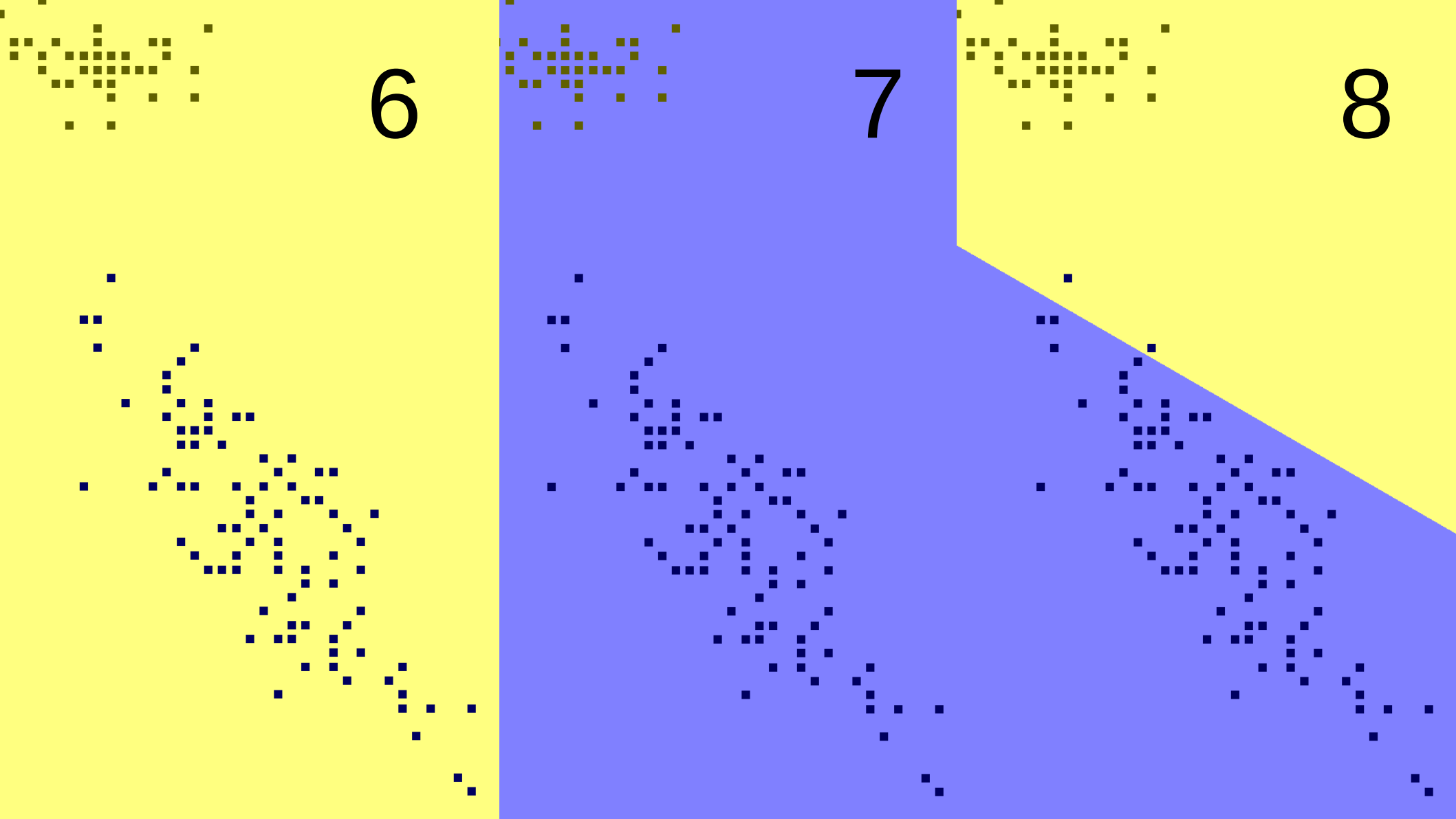


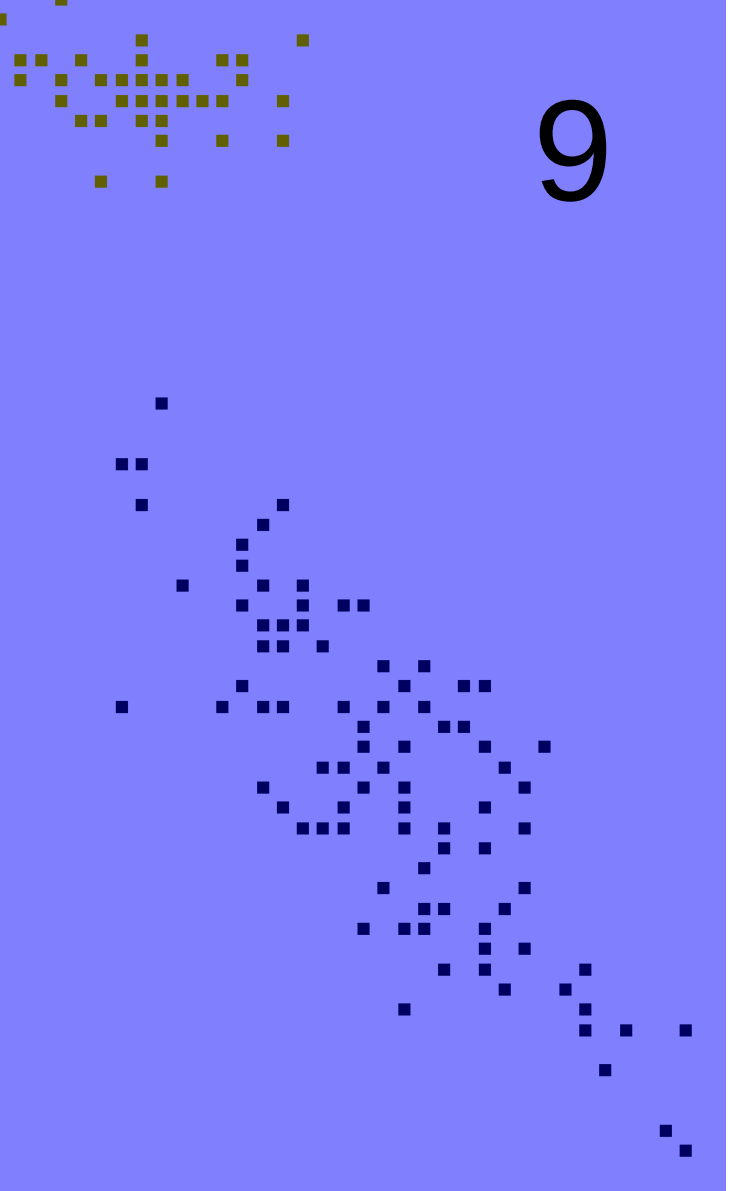
Exemplo na Base Iris

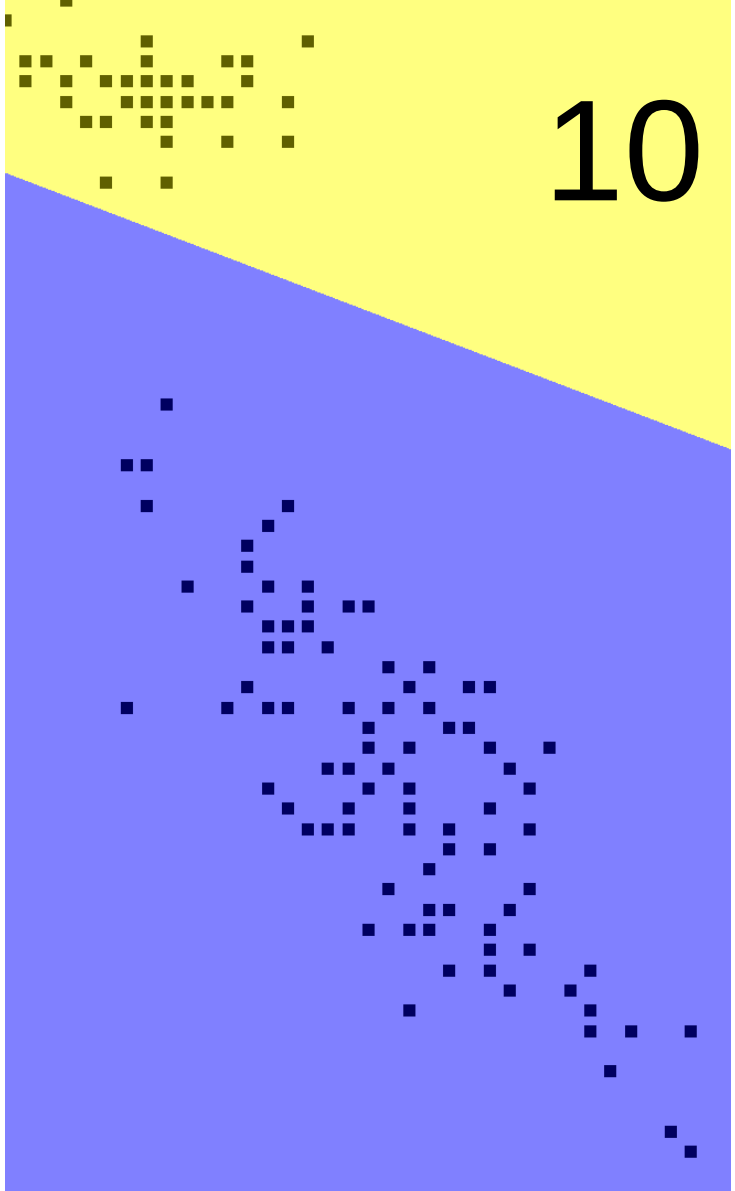
```
Época #1
Erro Total = 50.0

Época #2
Erro Total = 50.0

Época #3
Erro Total = 50.0

Época #4
Erro Total = 50.0

Época #5
Erro Total = 0.0
```


0
1
2


3
4
5


6
7
8


9


10

```
Época #1
Atualização 1
y[50]=1.0, f=0.0, w[0]=-1.0, w[1]=7.0, w[2]=4.7
Erro Total = 50.0

Época #2
Atualização 2
y[0]=0.0, f=1.0, w[0]=0.0, w[1]=1.9000000000000004, w[2]=3.3000000000000003
Atualização 3
y[1]=0.0, f=1.0, w[0]=1.0, w[1]=-3.0, w[2]=1.9000000000000004
Atualização 4
y[50]=1.0, f=0.0, w[0]=0.0, w[1]=4.0, w[2]=6.6000000000000005
Erro Total = 50.0

Época #3
Atualização 5
y[0]=0.0, f=1.0, w[0]=1.0, w[1]=-1.0999999999999996, w[2]=5.200000000000001
Atualização 6
y[1]=0.0, f=1.0, w[0]=2.0, w[1]=-6.0, w[2]=3.800000000000001
Atualização 7
y[50]=1.0, f=0.0, w[0]=1.0, w[1]=1.0, w[2]=8.500000000000002
Erro Total = 50.0

Época #4
Atualização 8
y[0]=0.0, f=1.0, w[0]=2.0, w[1]=-4.1, w[2]=7.100000000000001
Atualização 9
y[79]=1.0, f=0.0, w[0]=1.0, w[1]=1.6000000000000005, w[2]=10.600000000000001
Erro Total = 50.0

Época #5
Atualização 10
y[0]=0.0, f=1.0, w[0]=2.0, w[1]=-3.499999999999999, w[2]=9.200000000000001
Erro Total = 0.0
```

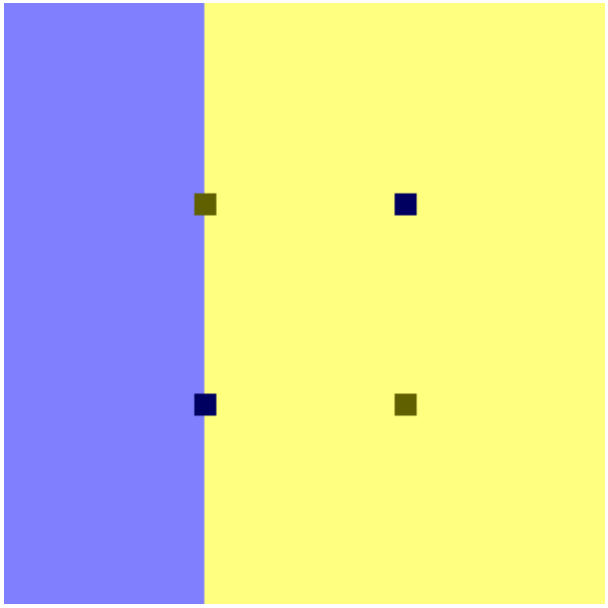
Limitações do Neurônio

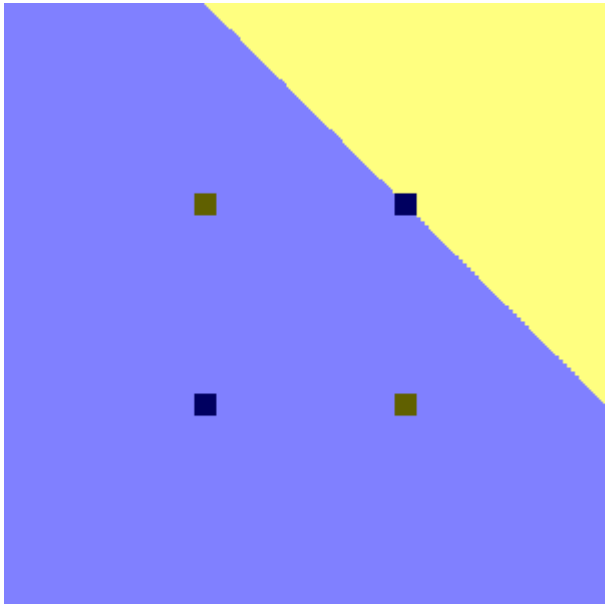
Prof. Tiago Buarque A. de Carvalho

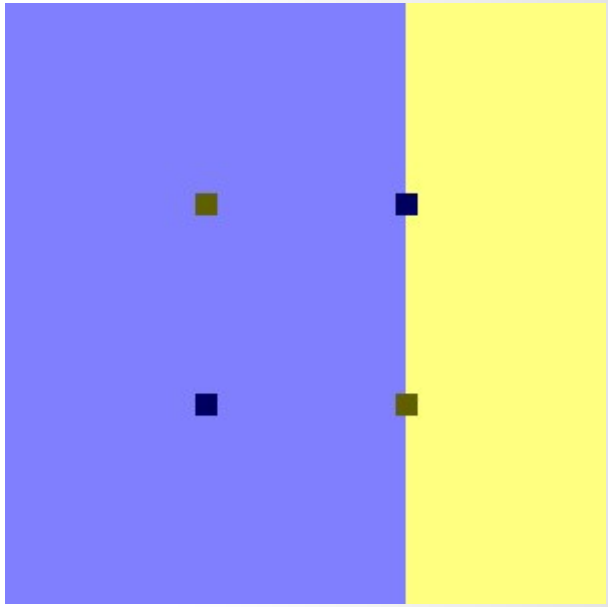
Limitações do neurônio simples

- Resolve apenas problemas lineares
  - XOR: ou exclusivo
  - Iris-versicolor
- Pode não alcançar erro zero
  - Para alguns problemas
- Pode Ter erro sempre alto
  - Não há solução linear aproximada

XOR (ou exclusivo): NÃO CONVERGE

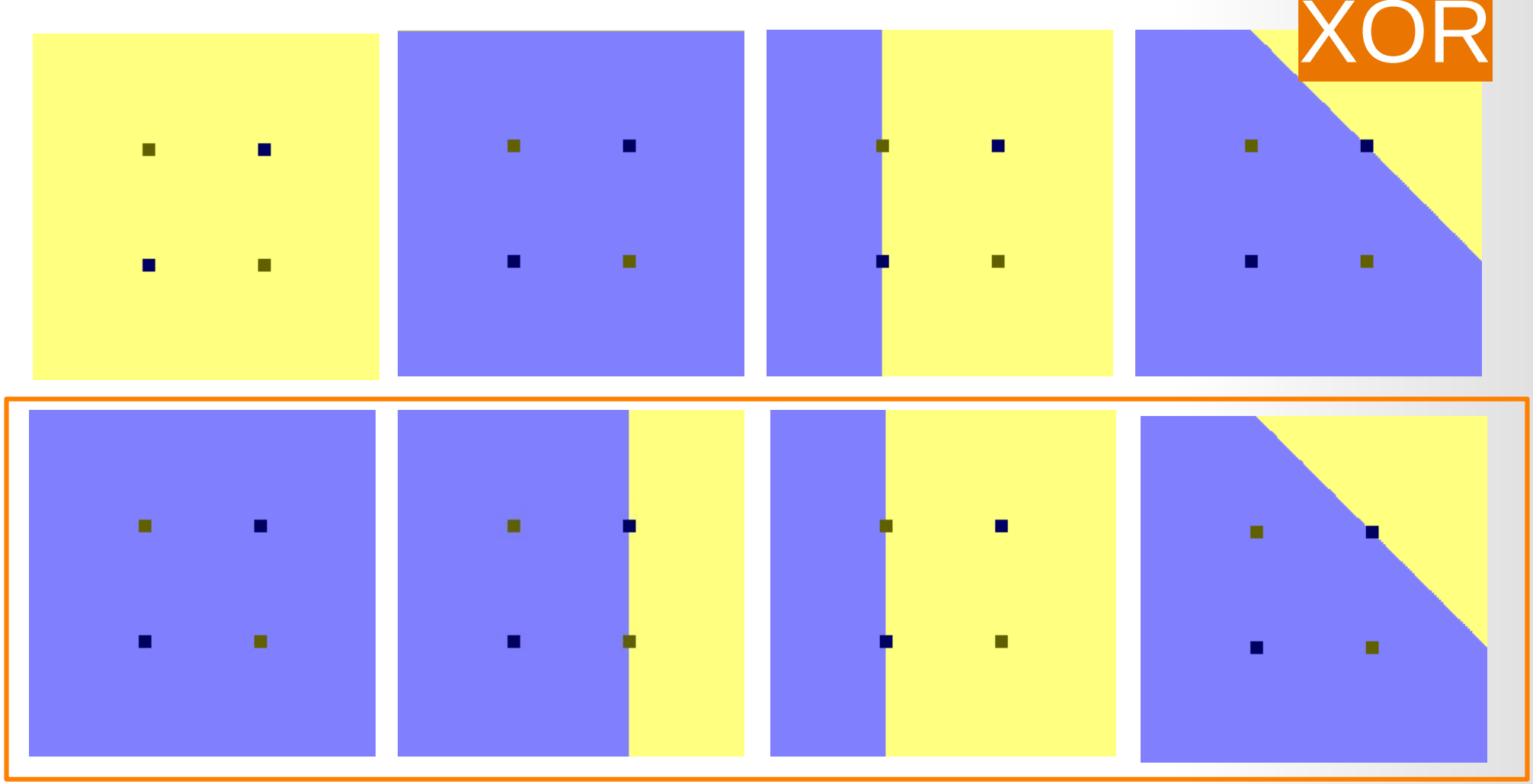




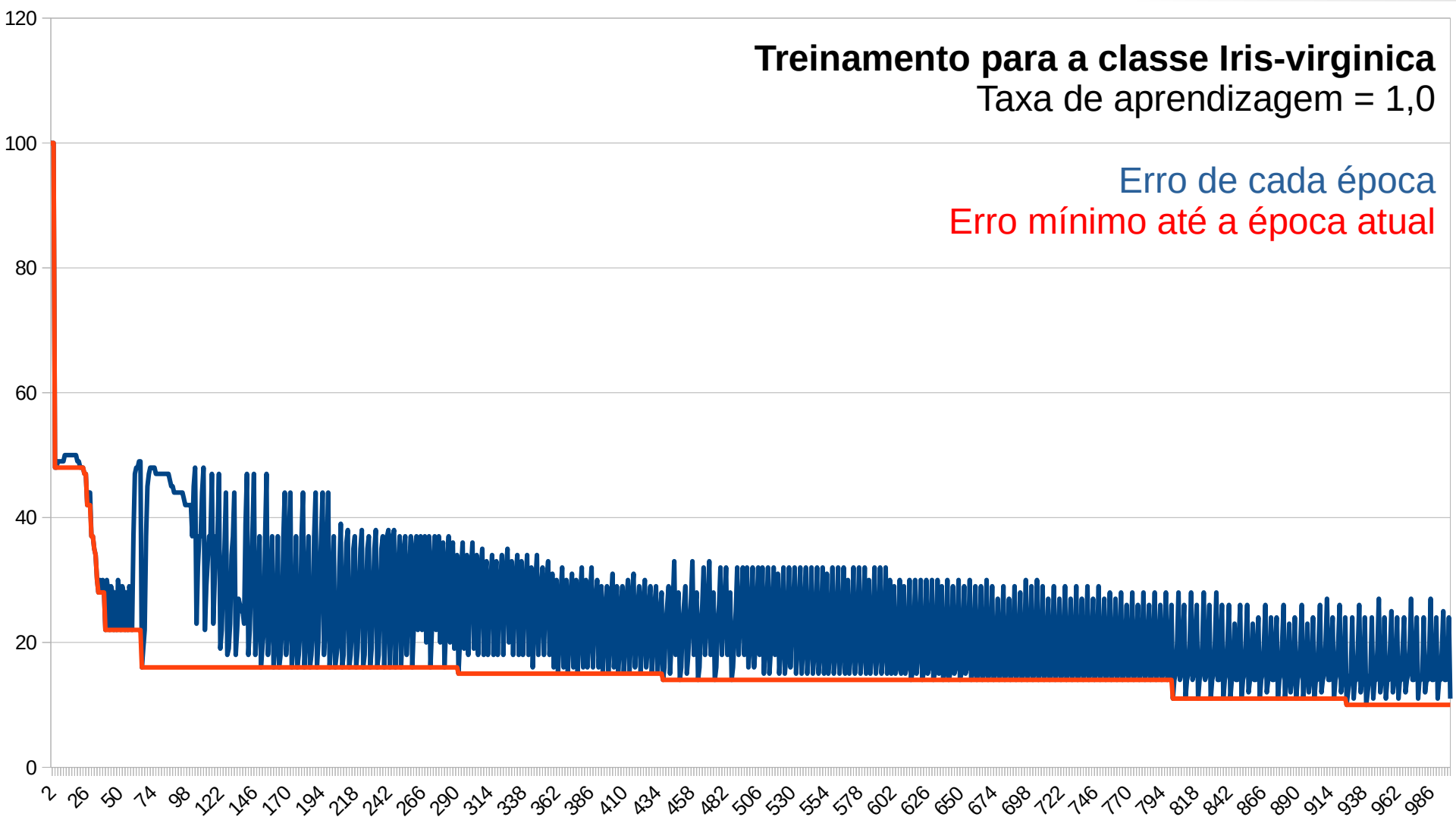


x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

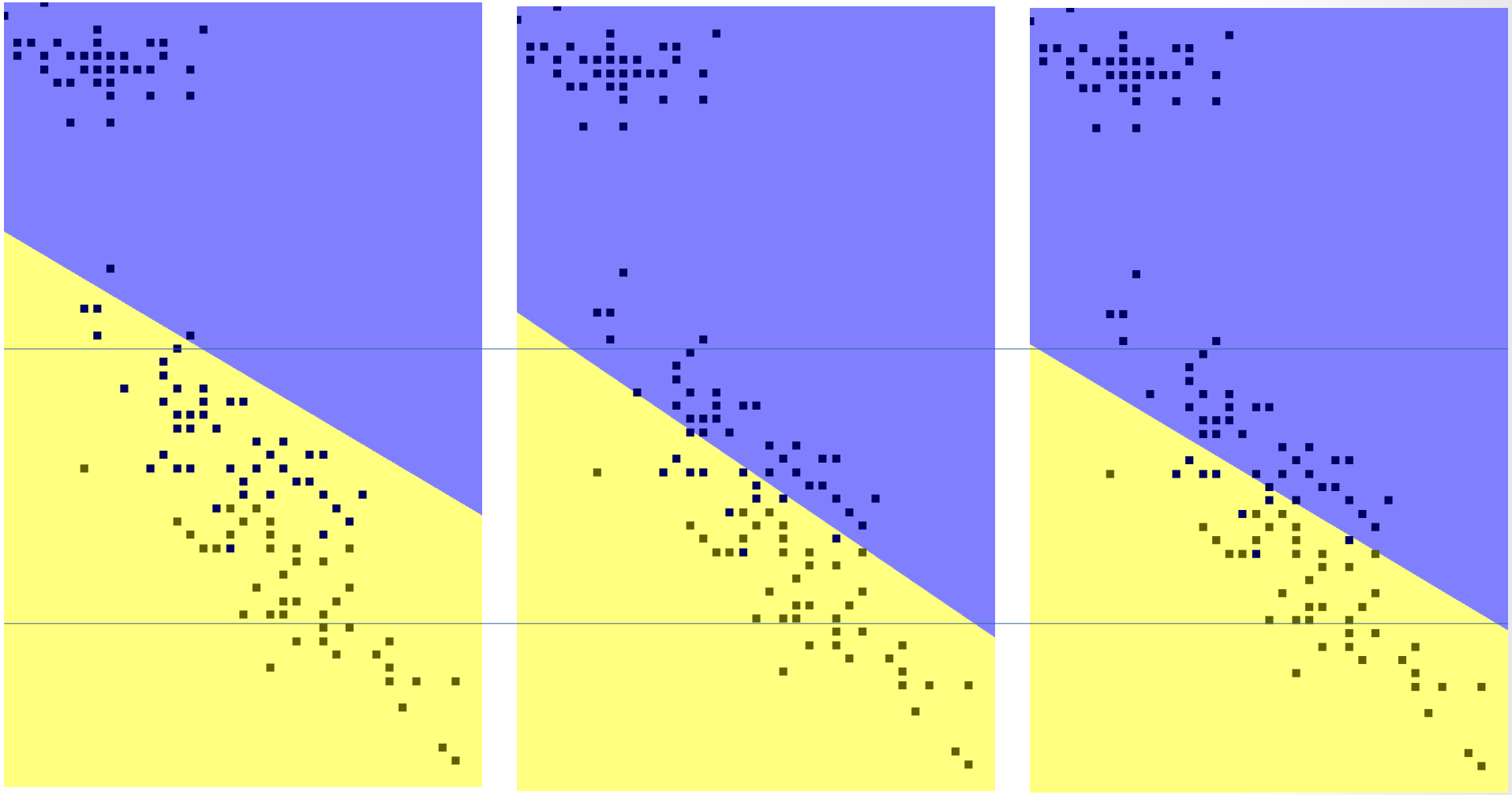
Pesos iniciais =(0,0,0)
Taxa de aprendizagem = 0.5



REPETE


Treinamento para a classe Iris-virginica
Taxa de aprendizagem = 1,0
Erro de cada época
Erro mínimo até a época atual
120
100
80
60
40
20
0
2
26
50
74
98
122
146
170
194
218
242
266
290
314
338
362
386
410
434
458
482
506
530
554
578
602
626
650
674
698
722
746
770
794
818
842
866
890
914
938
962
986

Iris-virginica



Base Iris

Setosa

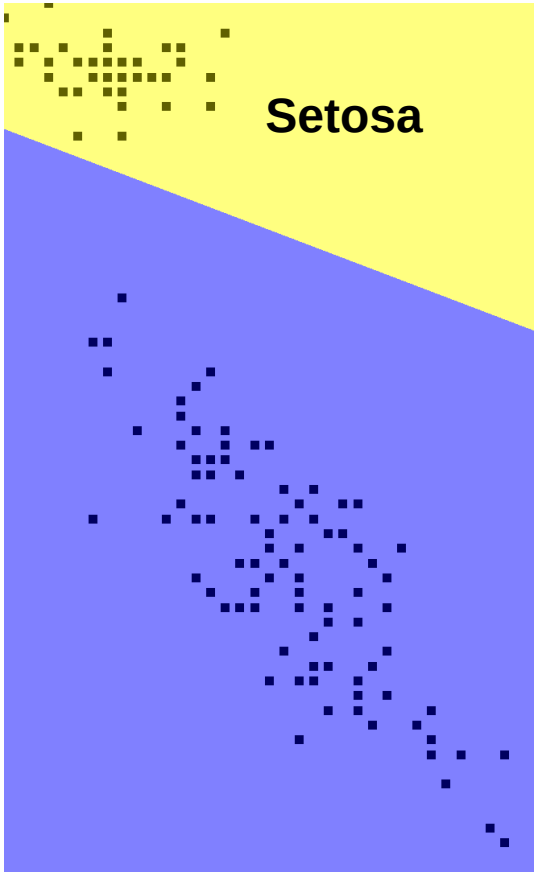
Erro = 0

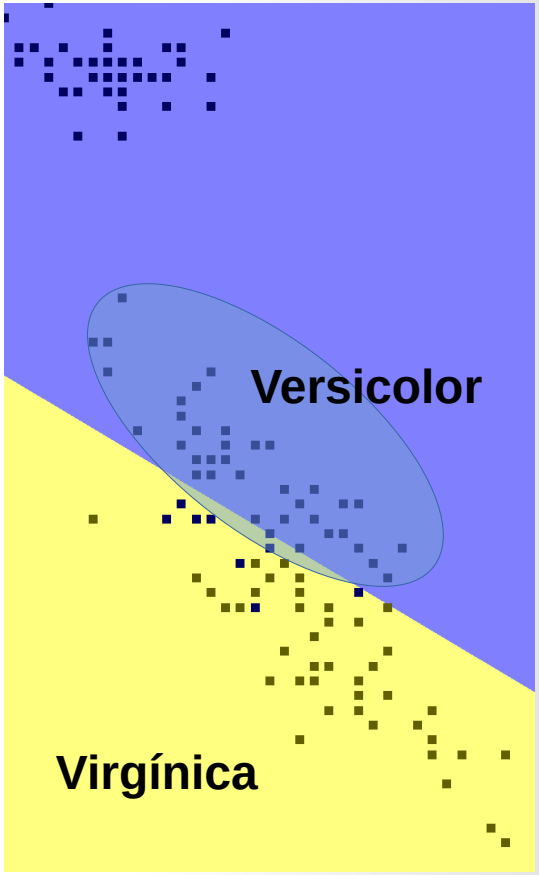
Versicolor

Erro = 100%

Virgínica

Erro = 20%





Como classificar Iris-Versicolor?

Dois neurônios

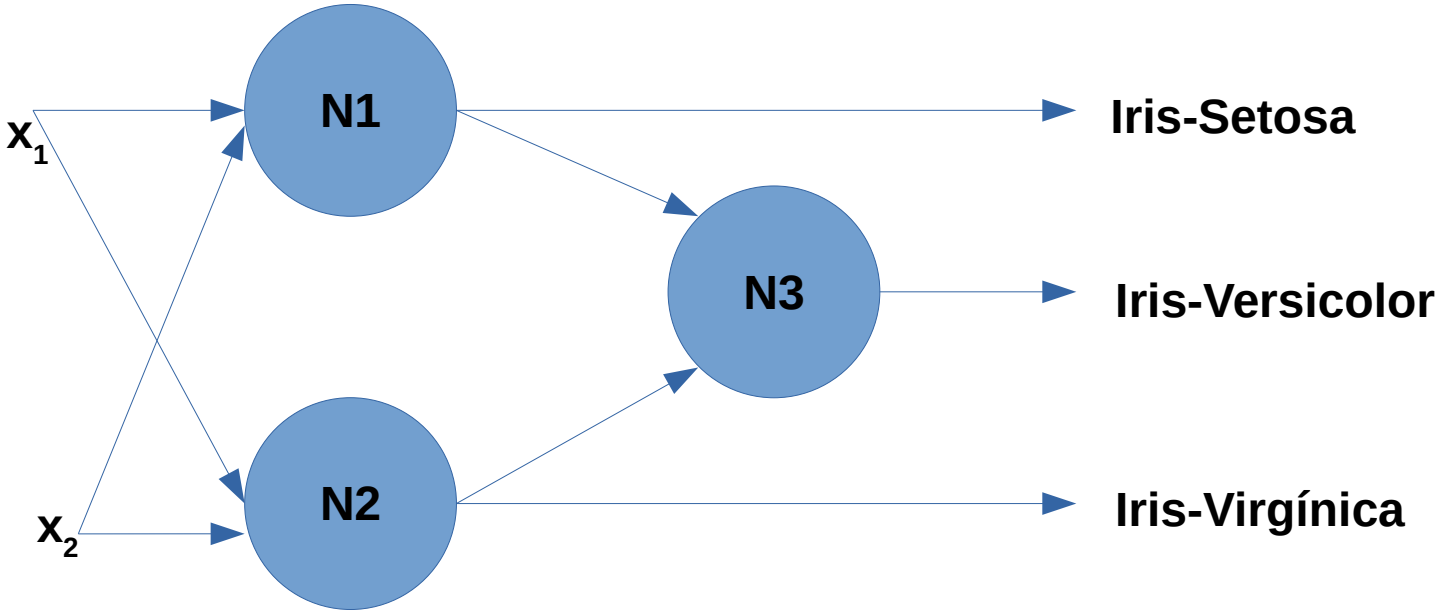
- o primeiro diz se é Setosa
- o segundo se é Virgínica

Se não for

- nem setosa nem Setosa nem Virgínica
- é Versicolor

Será que pode aprender Versicolor sozinha?

Rede Neural para a base Iris



Redes MLP

Prof. Tiago Buarque A. de Carvalho

Perceptron multicamadas

MLP

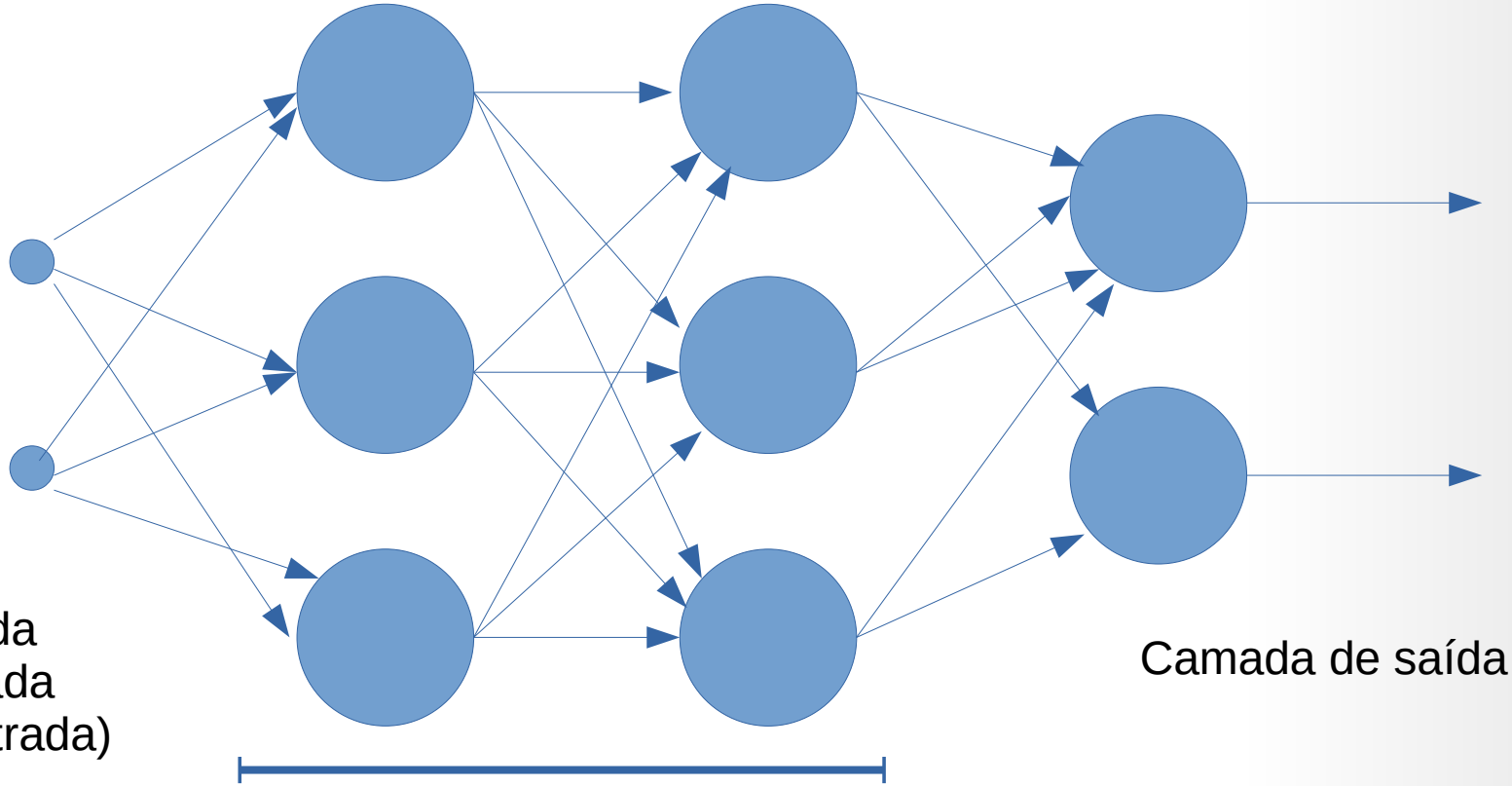
- Multilayer Perceptron (MLP)

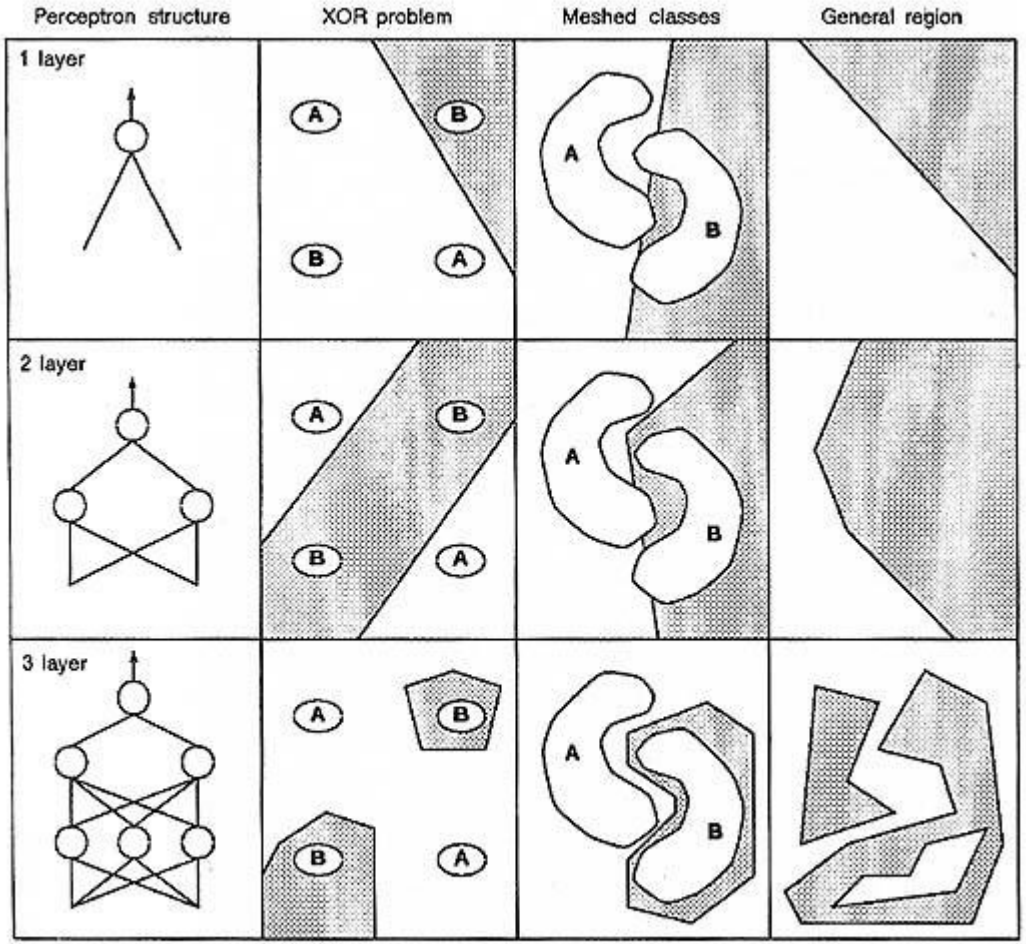
Camadas

- Geralmente Completamente Conectadas
- Cada neurônio de uma camada conecta-se a todos os outros da camada seguinte

A saída de uma camada é entrada da outra

MLP completamente conectada





(after Lippmann, IEEE ASSP April 1987)

Efeito do número de camadas

Pré-processamento para a saída

Um neurônio na camada de saída para cada classe

- Pré-processamento dos dados
- Forma mais comum mas não única

Pode fazer ajustes para evitar “saturar a saída”

Classe
Setosa
Virgínica
Setosa
Versicolor

Classe Setosa	Classe Versicolor	Classe Virgínica
1	0	0
0	0	1
1	0	0
0	1	0

Classe Setosa	Classe Versicolor	Classe Virgínica
0,9	0,1	0,1
0,1	0,1	0,9
0,9	0,1	0,1
0,1	0,9	0,1

Funções de Ativação

Prof. Tiago Buarque A. de Carvalho

Funções de Ativação

Necessariamente não-lineares

- A combinação de camadas lineares pode ser representada por uma única camada linear

Diferenciável (contínua, monotonicamente crescente)

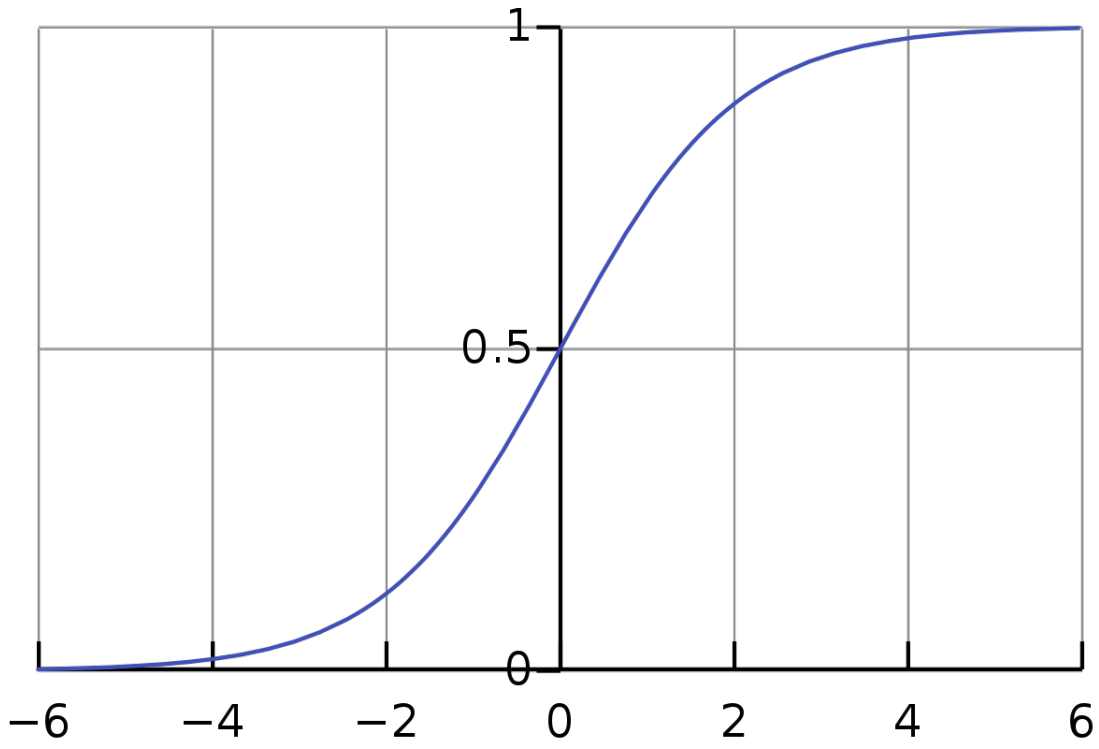
- Gradiente descendente (treinamento da rede)
- Utiliza-se a derivada da função

Várias

- Sigmóide, Tangente Hiperbólica, RELU, …

A derivada podem atrasar o treinamento

FunçãoSigmóide ou Função Logística



Saída ϵ [0, 1]

Assemelha-se ao degrau

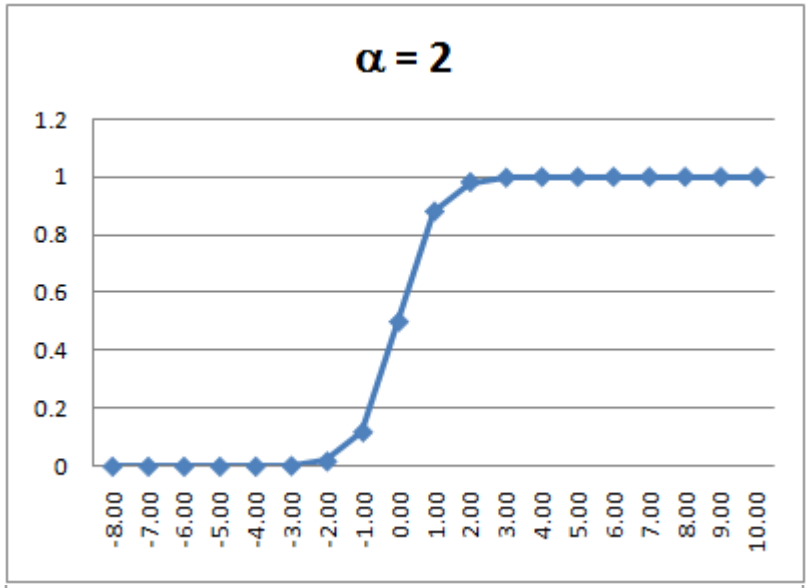
Derivada em [0; 0,25]

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

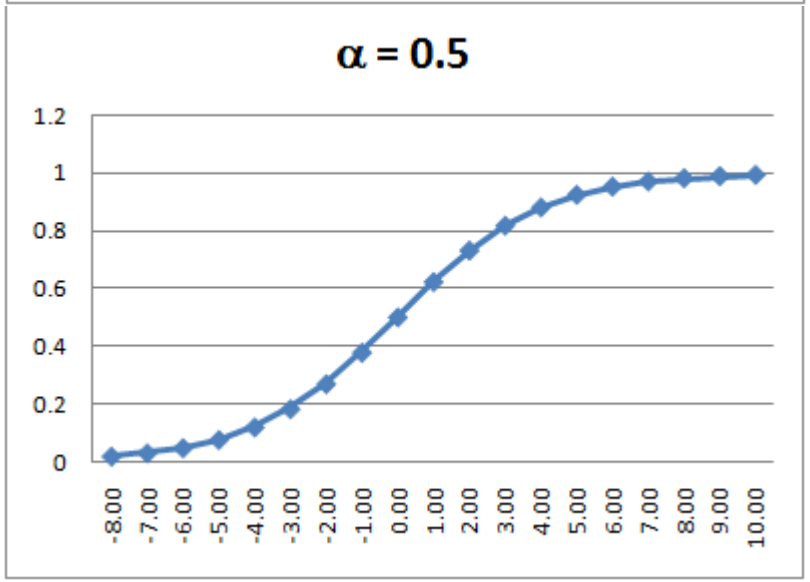
$$f'(x) = f(x)(1 - f(x))$$

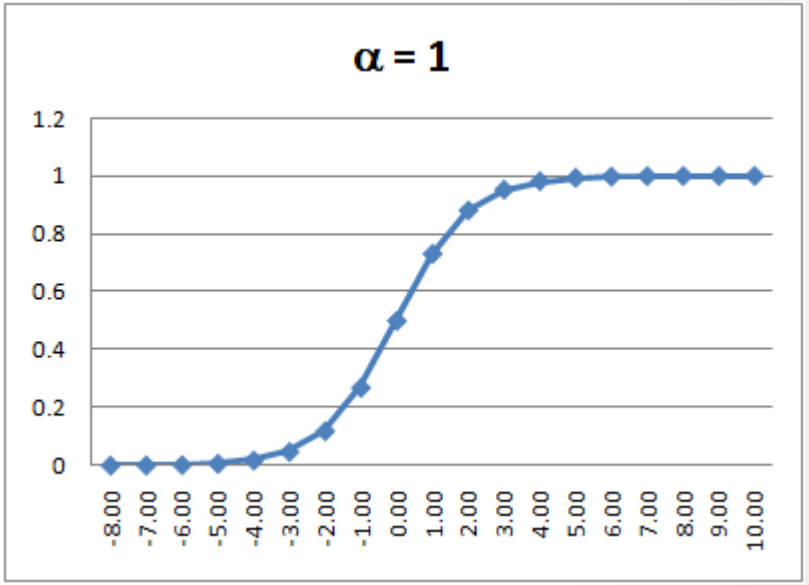
https://en.wikipedia.org/wiki/File:Logistic-curve.svg

Sigmóide



$$f(u) = \frac{1}{1 + \exp(-u\alpha)}$$





Tangente Hiperbólica

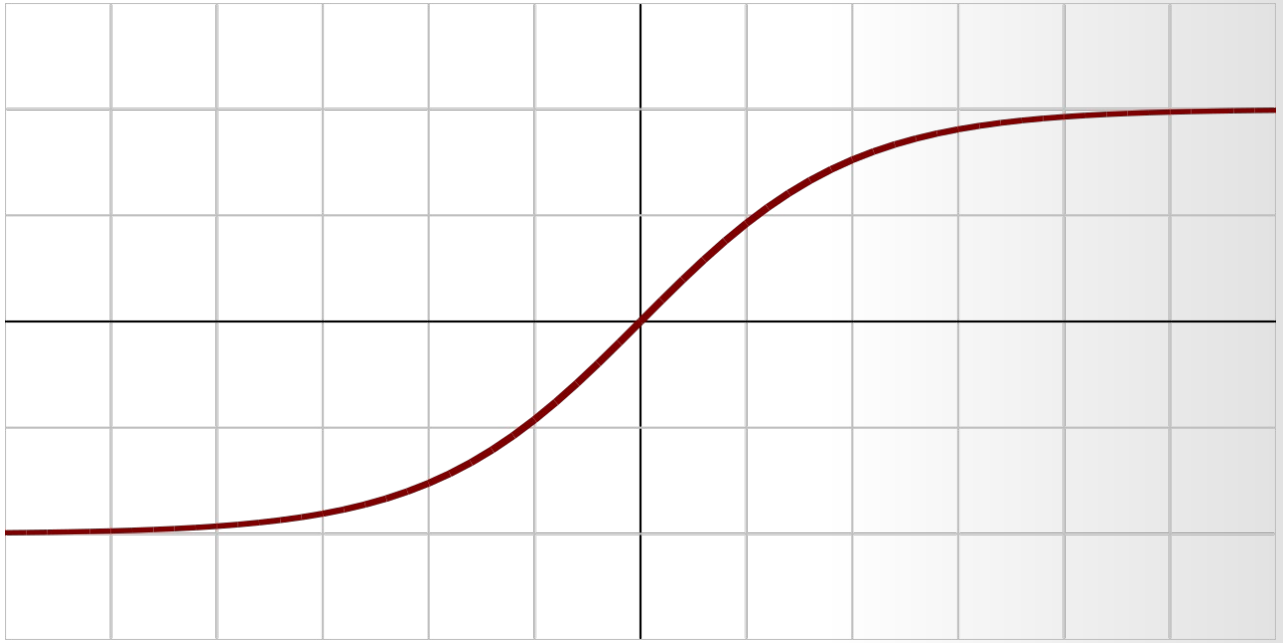
Saída no intervalo [-1, 1]

Evita multiplicação por zero

Derivada em [0; 1]

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

$$f'(x) = 1 - f(x)^2$$



https://en.wikipedia.org/wiki/File:Activation_tanh.svg

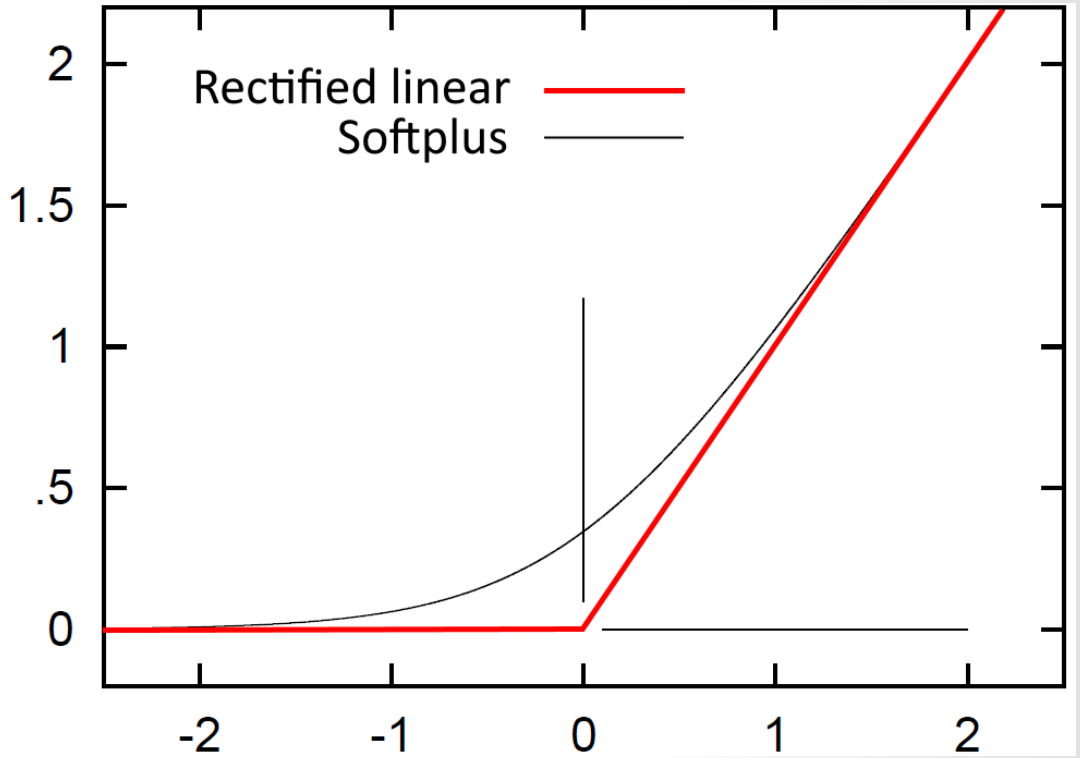
Rectified linear unit (ReLU)

Saída >= 0

Derivada em [0; 1]

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$$



https://en.wikipedia.org/wiki/File:Rectified_linear_and_Softplus_activation_functions.png

Treinamento de MLP

Prof. Tiago Buarque A. de Carvalho

Back-propagation

Algoritmo de treinamento de uma rede MLP

Restringe que cada neurônio deve ter uma função de ativação **diferenciável** (contínua? monotonicamente crescente?)

Os pesos devem obrigatoriamente serem inicializados com **valores aleatórios** para os neurônios não ficarem iguais

Duas fases

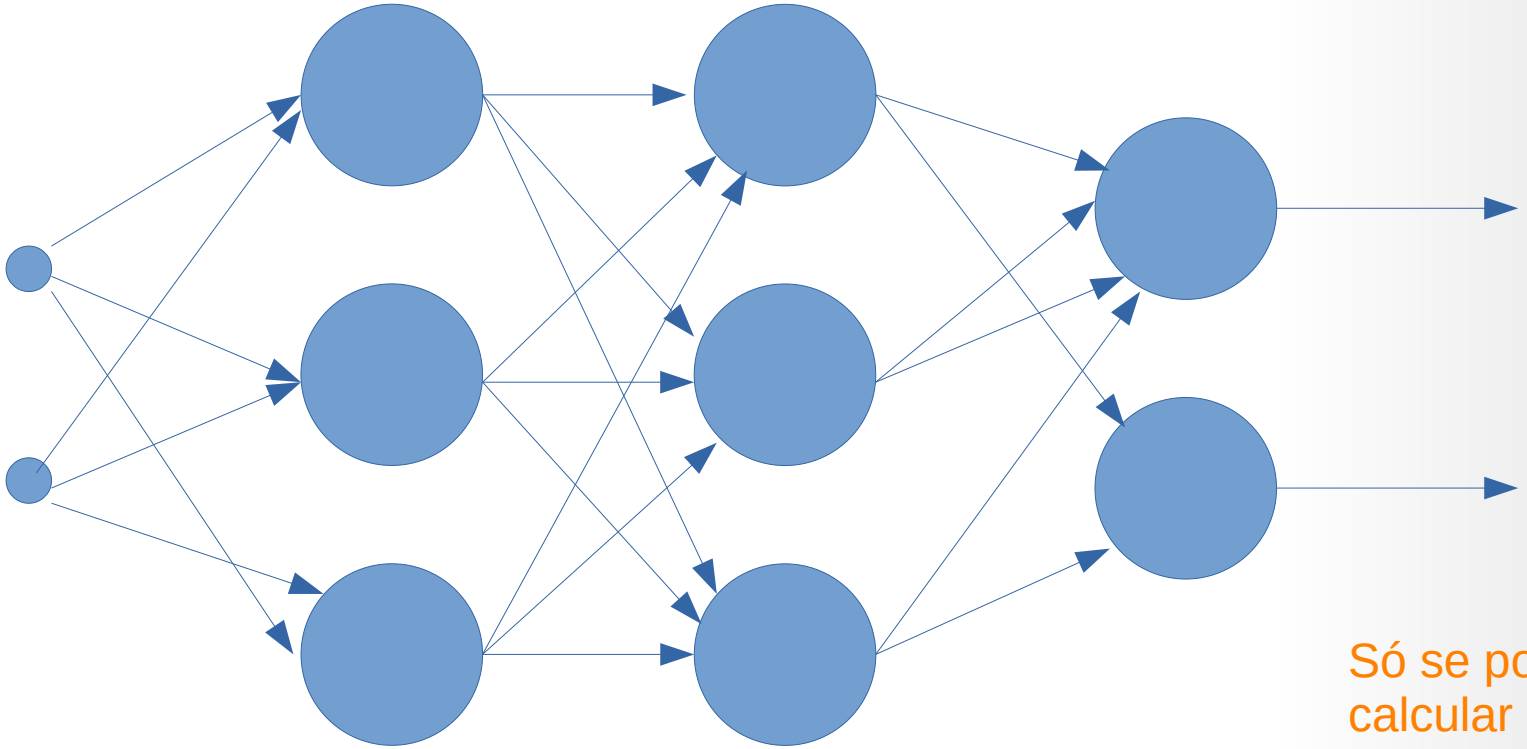
Para frente: calcula a saída da rede

Para trás: atualiza os pesos

- Calcula o erro na camada de saída e propaga para as outras camadas
- Só se pode calcular o erro na camada de saída!
- A atualização das camadas escondidas depende do erro na cama seguinte

Pra frente: calcula a saída

Pra trás: atualiza os peso



Só se pode calcular o erro na camada de saída!

Atualização dos Pesos

A atualização dos pesos é diferente na camada de saída em relação às camadas intermediárias

O peso da entrada j do neurônio k é atualizado em função do delta para aquele neurônio, da taxa de aprendizagem e do valor x_j do vetor **x**

$$w_{kj}[t+1] = w_{kj}[t] + \eta x_j \delta_k$$

Delta

O valor do delta na entrada é o produto entre a derivada da saída e o erro – o erro é a saída desejada menos a saída do neurônio (*para minimizar o erro quadrático*)

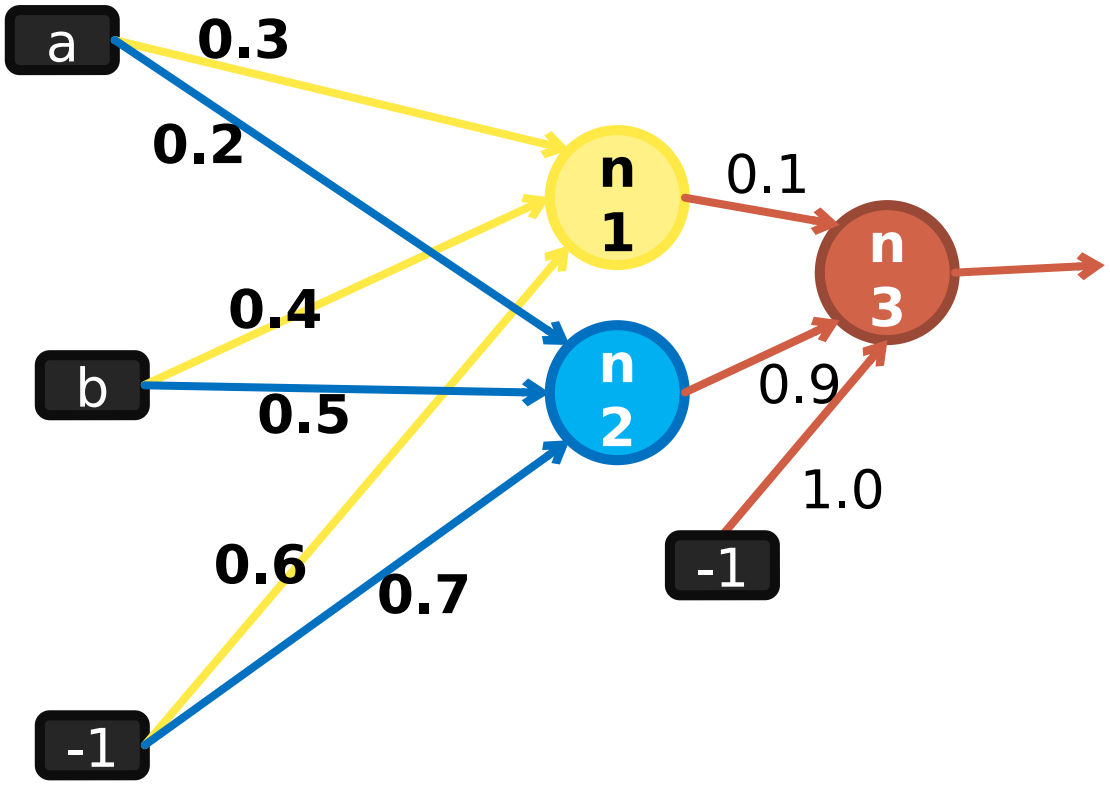
O delta em um neurônio de camada escondida é uma combinação dos deltas dos neurônios para quem ele transmite multiplicado pelo peso da sua saída para o receptor

$$\delta_k = \begin{cases} f'(v_k)(y_i - f(v_k)), \text{ na saída} \\ f'(v_k)\sum_j w_{kj}\delta_j, \text{ internamente} \end{cases}$$

$$w_{kj}[t+1] = w_{kj}[t] + \eta x_j \delta_k$$

$$f(v) = \frac{1}{1 + exp(-v)}$$

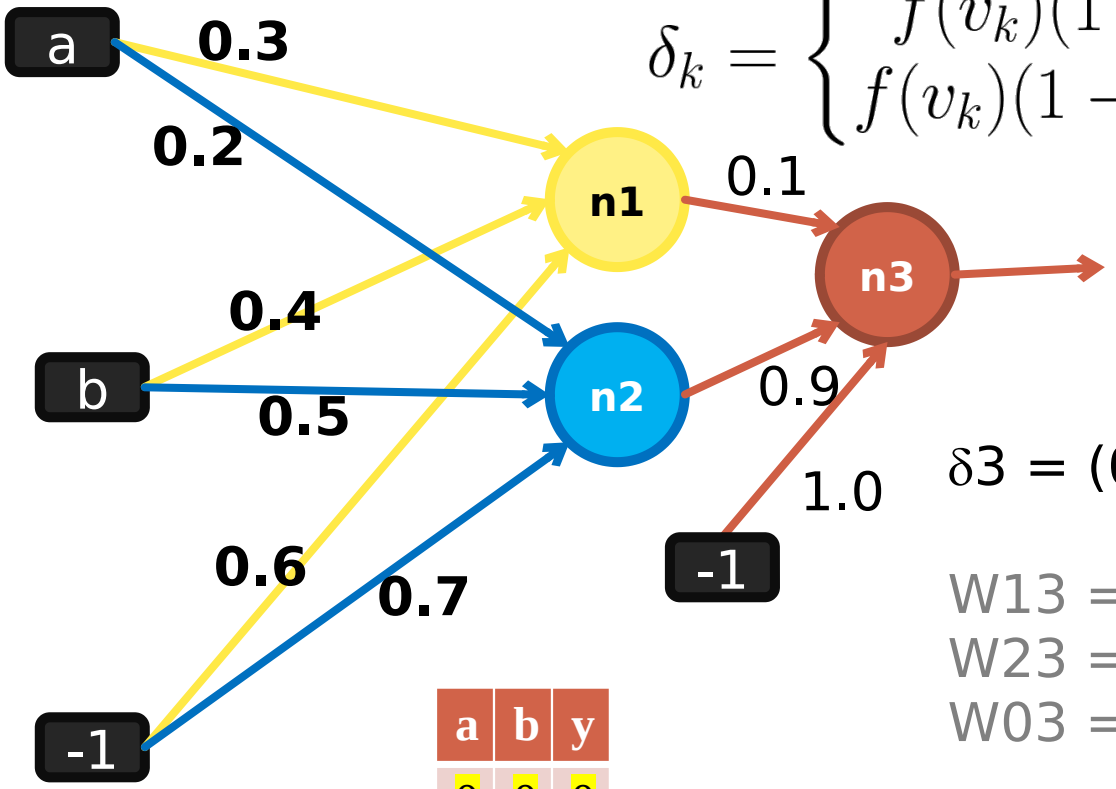
$$\delta_k = \begin{cases} f(v_k)(1 - f(v_k))(y_i - f(v_k)), \text{ na saída} \\ f(v_k)(1 - f(v_k)) \sum_j w_{kj}\delta_j, \text{ internamente} \end{cases}$$



a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

$\eta = 20$

u	-8	-4	-2	-1	0	1	2	4	8
f(u)	0.00	0.02	0.12	0.27	0.50	0.73	0.88	0.98	1.00



$$\delta_k = \begin{cases} f(v_k)(1 - f(v_k))(y_i - f(v_k)), \text{ na saída} \\ f(v_k)(1 - f(v_k)) \sum_j w_{kj}\delta_j, \text{ internamente} \end{cases}$$

u1 = -0.6, f(u1)=0.27
u2=-0.7, f(u2)=0.27
u3=-0.73, f(u3)= 0.27

δ3 = (0.27 *(1-0.27)*(0-0.27)) = -0.05

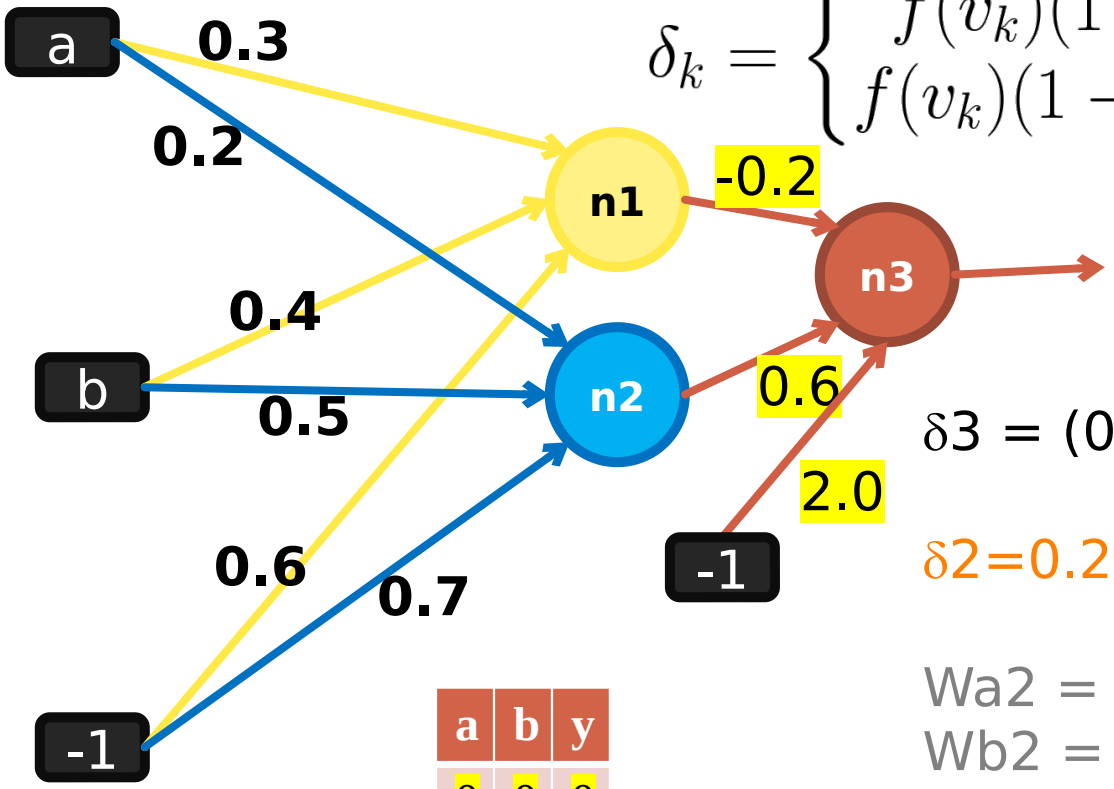
W13 = 0.1 +2 *0.27 * (-0.05)= -0.2
W23 = 0.9 +20 *0.27 * (-0.05) = 0.6
W03 = 1 +20 *(-1)* (-0.05) = 2

$$w_{kj}[t+1] = w_{kj}[t] + \eta x_j \delta_k$$

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

η= 20

u	-8	-4	-2	-1	0	1	2	4	8
f(u)	0.00	0.02	0.12	0.27	0.50	0.73	0.88	0.98	1.00



$$\delta_k = \begin{cases} f(v_k)(1-f(v_k))(y_i - f(v_k)), \text{ na saída} \\ f(v_k)(1-f(v_k))\sum_j w_{kj}\delta_j, \text{ internamente} \end{cases}$$

u1 = -0.6, f(u1)=0.27
u2=-0.7, f(u2)=0.27
u3=-0.73, f(u3)= 0.27

$\delta 3$ = (0.27 *(1-0.27)*(0-0.27)) = -0.05

$\delta 2$=0.27(1-0.27)(0.6*(-0.05))=-0.0059130

Wa2 = 0.2 + 20*0* $\delta 2$ = 0.2
Wb2 = 0.5 + 20*0* $\delta 2$ = 0.5
W02 = 0.27+ 20*(-1)* (-0.0059130)=0.4

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

η= 20

$$w_{kj}[t+1] = w_{kj}[t] + \eta x_j \delta_k$$

u	-8	-4	-2	-1	0	1	2	4	8
f(u)	0.00	0.02	0.12	0.27	0.50	0.73	0.88	0.98	1.00

Convergência do algoritmo backpropagation

- Lenta
- Requer muitas épocas
- Pode parar em um mínimo local
- Não tem garantia de convergência como o perceptron

Taxa de Aprendizagem

Pequena

- convergência lenta

Grande

- pode oscila demais e não convergir

Diminuir a taxa a cada época de treino

Variações do Backpropagation

Batch Backpropagation

- Atualiza os pesos pelo erro médio para todos os padrões
- **Mini-Batch**

Funções de erros (*loss*)

- Erro médio quadrático

Quickprop, Rprop, ...

- Algoritmos de treinamento mais rápidos

Momentum, Momentum de segunda ordem

- Evita grandes oscilações

Outros métodos de otimização

- Para realizar o treinamento (pesos ótimos)
- Newton, Levenberg-Marquardt, ...

Critério de Parada

Época (ciclo) de treinamento

Cada exemplo de treino passou exatamente uma vez a rede

O treinamento leva várias épocas

O algoritmo pára

Após um número fixo de épocas

Taxa máxima de erro sobre **conjunto de validação**

- Conjunto de Validação é uma partição do treino para testa a rede
- A rede é testada a cada 10 ou 100 épocas

O erro sobre a validação começa a crescer por 10 épocas seguidas

Overfitting

Prof. Tiago Buarque A. de Carvalho

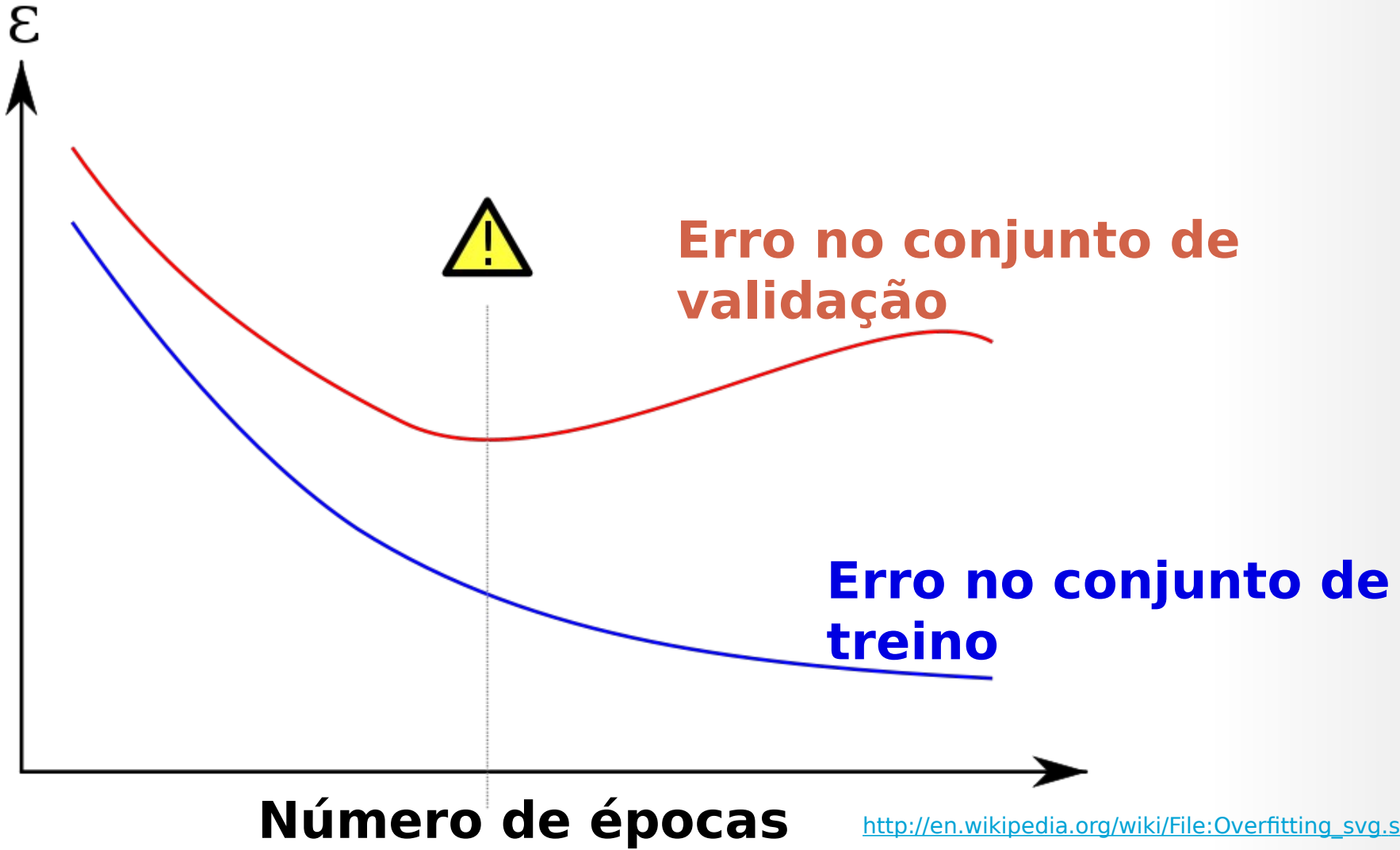
Overfitting

Sobre-ajuste

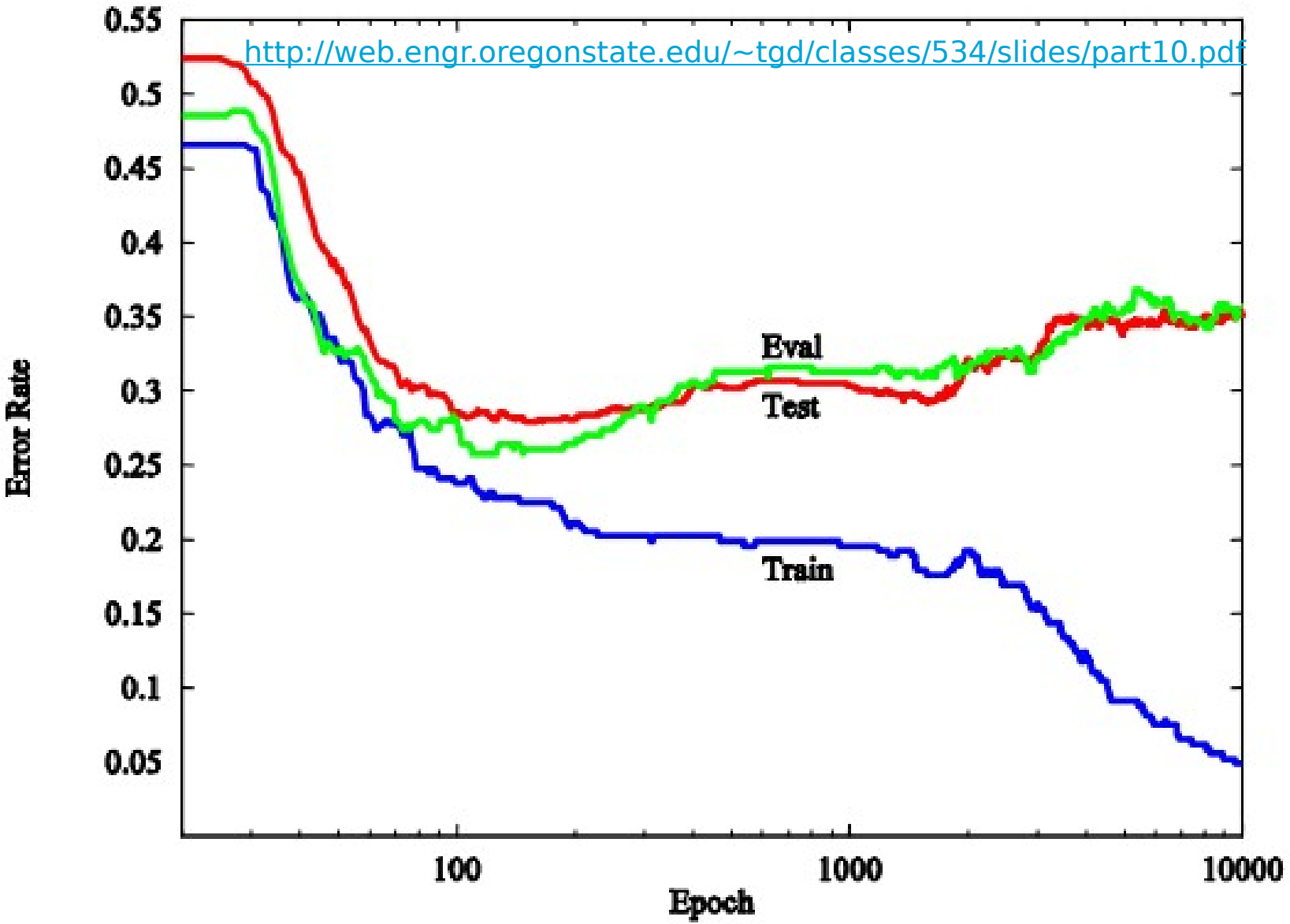
- A rede “decorou o conjunto de treino” e não consegue generalizar

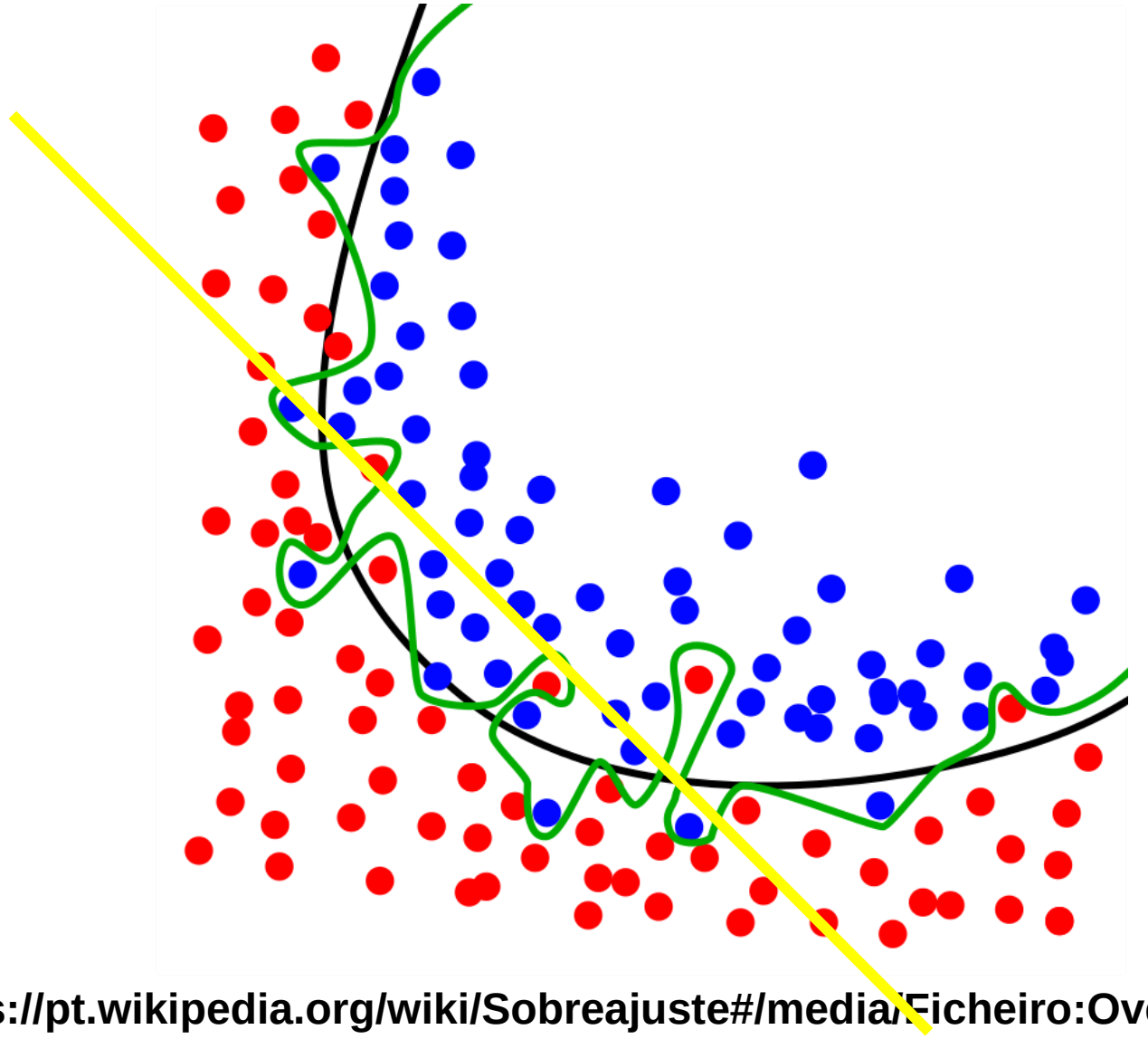
Conjunto de validação

- Diferente do conjunto de treino e teste
- Serve Para monitorar *overfitting*



http://en.wikipedia.org/wiki/File:Overfitting_svg.svg


http://web.engr.oregonstate.edu/~tgd/classes/534/slides/part10.pdf
Error Rate
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
Eval
Test
Train
100
1000
10000
Epoch



https://pt.wikipedia.org/wiki/Sobreajuste#/media/Ficheiro:Overfitting.svg

Underfitting ≠ Overfitting

A rede pode ter menos neurônios que o necessário

- Não consegue generalizar nem no conjunto de treinamento

“Não consigo aprender!” :(

O que mais?

Prof. Tiago Buarque A. de Carvalho

Projetos de arquiteturas

- Função de ativação
- Topologia
  - Número de camadas
  - Neurônios por camadas
- Quanto mais camadas, mais demorado o treinamento
- Estratégia
  - Empírica: começando com apenas uma camada escondida
  - Meta-heurística: gera e testa várias por vez, pode utilizar algoritmos genéticos
  - Poda: elimina nós enquanto o erro de validação permanece aceitável
  - Construtiva: vai inserindo nós nas camadas

Outras Redes

Redes Convolucionais

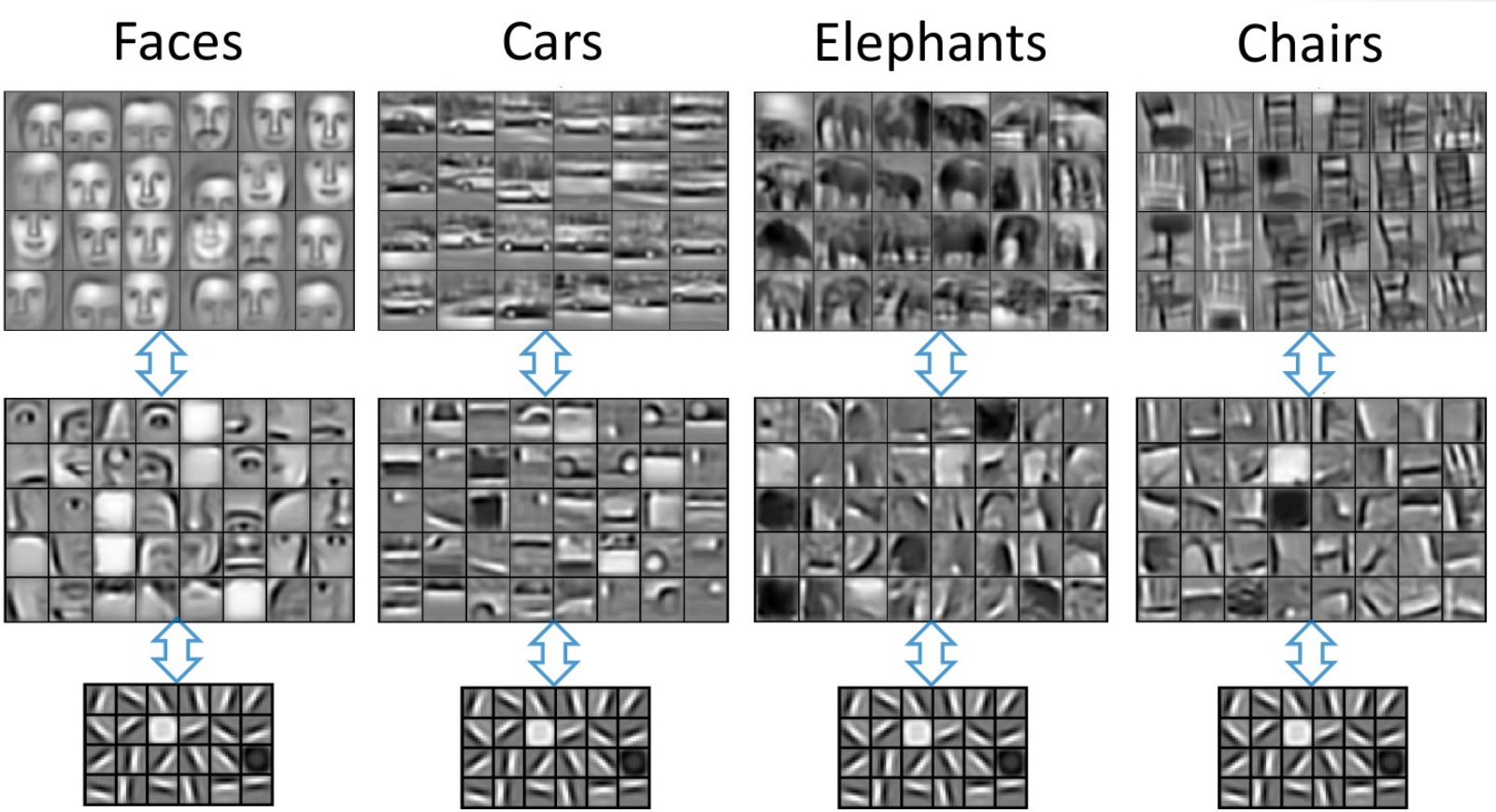
- Aplicada para imagens

LSTM (*Long short-term memory*)

- Redes recorrente
- A saída da rede volta para a entrada
- Problemas de séries temporais, texto, áudio, …

Autoencoders, RBF, SOM, ….

Redes Convolucionais



http://videolectures.net/icml09_lee_cdb/

Vantagens e Desvantagens

- ✔ Generalização
- ✔ Tolerância a falhas e ruídos
- ✔ Degradação graciosa
- ✔ Resolve tarefas de baixo nível, ex. visão computacional
- ✔ Paralela

- ✗ Não fica claro como a rede chega às suas conclusões
- ✗ Dificuldade de escolhas de parâmetros
- ✗ Requer muito tempo e recursos computacionais

Referências

R. Beale, T. Jackson. 1992. **Neural Computing: an introduction**. IOP.

Thomas M. Mitchell. 1997. **Machine Learning** McGraw-Hill.

Katti Faceli, Ana Carolina Lorena, João Gama, André C. P. L. F. de Carvalho. 2011. **Inteligência Artificial – Uma abordagem de Aprendizado de Máquina**. LTC.

Ian Goodfellow and Yoshua Bengio and Aaron Courville. 2016. **Deep Learning**. MIT Press. http://www.deeplearningbook.org

https://www.youtube.com/watch?v=McgxRxi2Jqo

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. 2009. **Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations**. ICML.

Redes Neurais

Prof. Tiago Buarque A. de Carvalho