

Inteligência Artificial

Redes Neurais Artificiais

Prof. Dr^a. Andreza Sartori

asartori@furb.br

Documentos Consultados/Recomendados

- ARTERO, Almir Olivette. **Inteligência artificial: teórica e prática**. 1. ed. São Paulo: Livraria da Física, 2008.
- Carvalho, André P. **Redes Neurais Artificiais**. USP. <http://www.din.uem.br/ia/neurais/>
- Carvalho, Cedric Luiz de. **Redes Neurais Artificiais**. UFG, 2006.
- COPPIN, Ben. **Inteligência artificial**. Rio de Janeiro: LTC, 2013.
- KLEIN, Dan; ABBEEL, Pieter. **Intro to AI**. UC Berkeley. Disponível em: <http://ai.berkeley.edu>
- LIMA, Edirlei Soares. **Inteligência Artificial**. PUC-Rio, 2015.
- NG, Andrew. **Machine Learning**. Stanford University.
<https://www.coursera.org/learn/machine-learning>
<http://cs229.stanford.edu/materials.html>
- RUSSELL, Stuart J. (Stuart Jonathan); NORVIG, Peter. **Inteligência artificial**. Rio de Janeiro: Campus, 2013. 1021p.

Conteúdo Programático – Inteligência Artificial

Unidade 1: Fundamentos de Inteligência Artificial

Unidade 2: Busca

Unidade 3: Sistemas Baseados em Conhecimento

Unidade 4: Redes Neurais Artificiais

Unidade 5: Aplicações de Inteligência Artificial



Conteúdo Programático

Unidade 1: Fundamentos de Inteligência Artificial

Unidade 2: Busca

Unidade 3: Sistemas Baseados em Conhecimento

Unidade 4: Redes Neurais Artificiais

Unidade 5: Aplicações de Inteligência Artificial



Conteúdo Programático

Unidade 1: Fundamentos de Inteligência Artificial

Unidade 2: Busca

Unidade 3: Sistemas Baseados em Conhecimento

Unidade 4: Redes Neurais Artificiais

4.1. Fundamentos biológicos

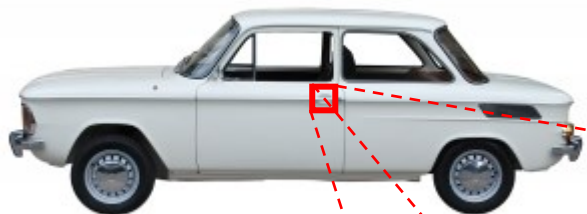
4.2. Estrutura das RNAs

4.3. Rede Perceptron



Problema de Visão Computacional

O que você vê....



O que a máquina enxerga:

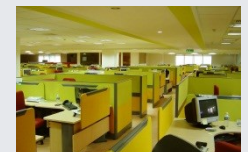
194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Visão Computacional: Reconhecimento de Carros

Treinamento



Carros



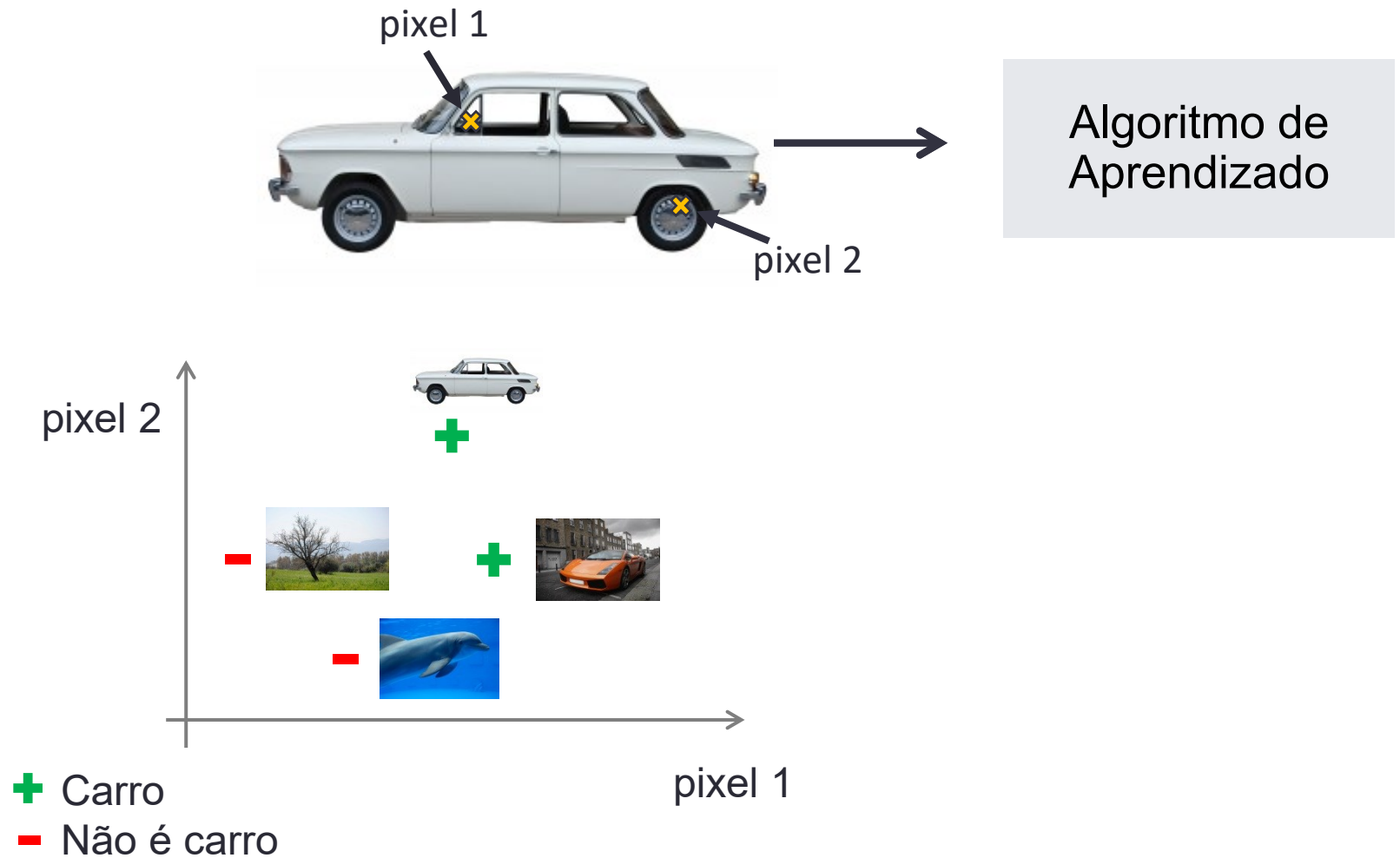
Não é carro

Teste:

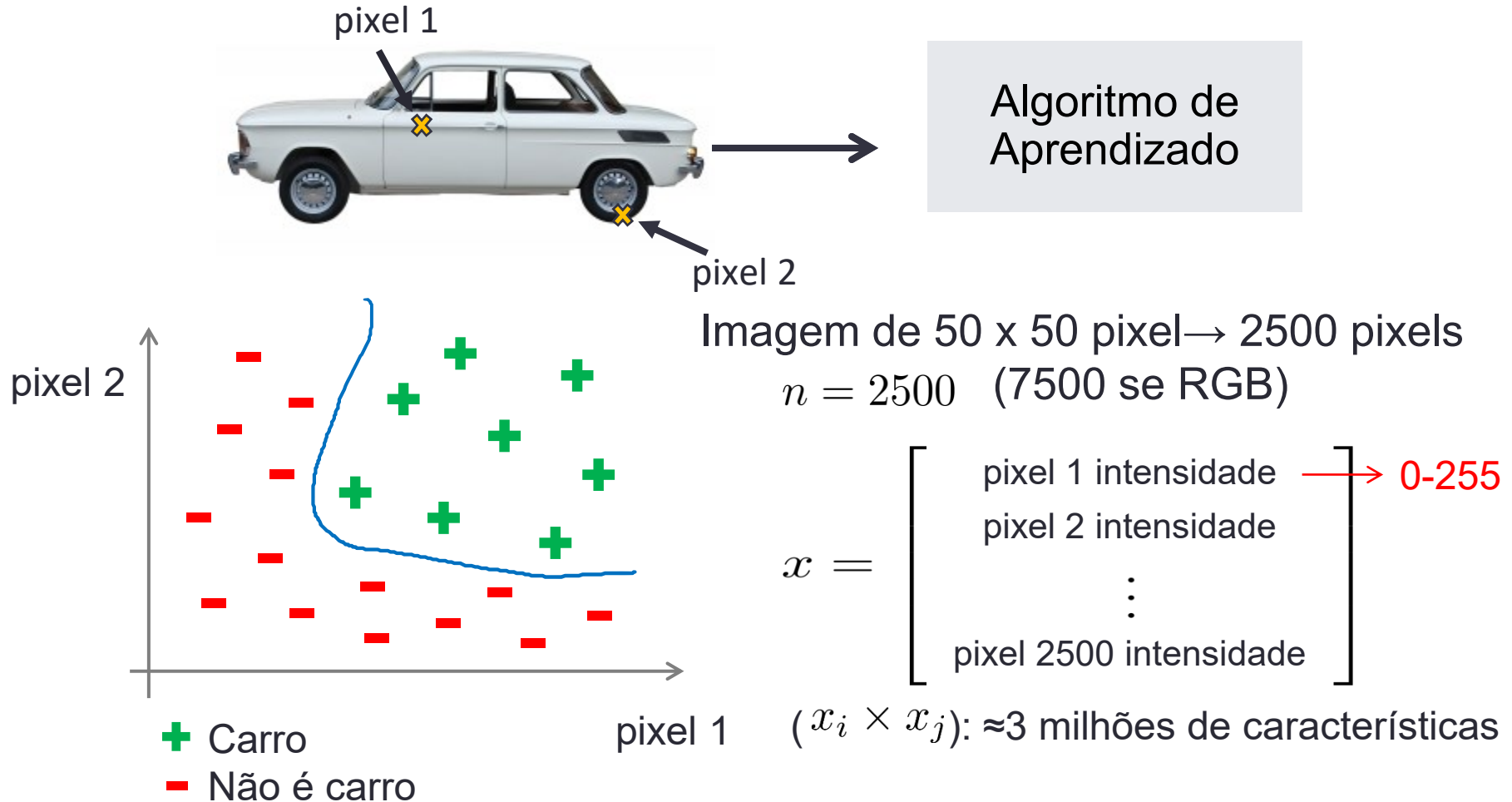


O que é?

Visão Computacional: Reconhecimento de Carros

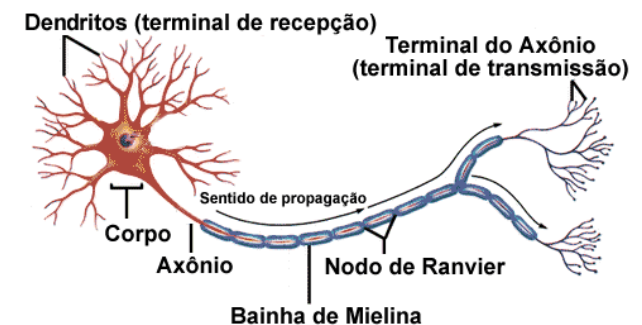


Visão Computacional: Reconhecimento de Carros



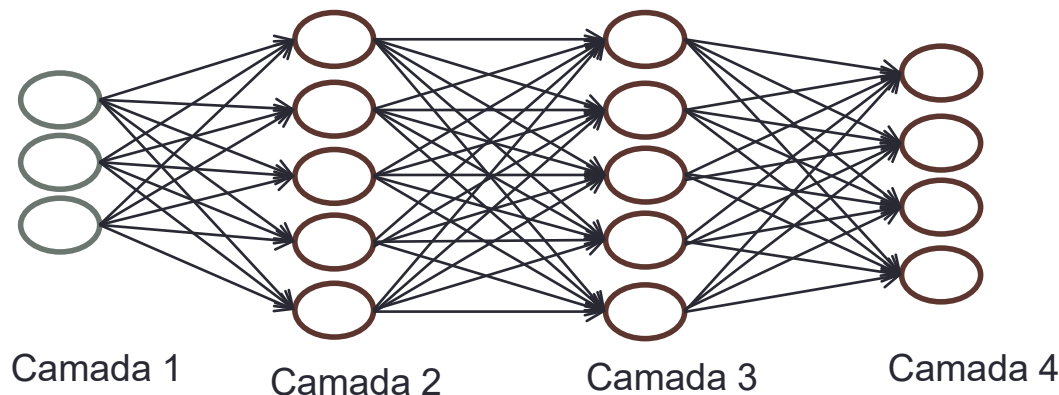
Redes Neurais Artificiais (RNA)

- Modelos computacionais inspirados no cérebro humano.
 - Tentam “imitar” o cérebro humano.
- Amplamente utilizada na década de 80 e início dos anos 90, mas a popularidade diminuiu no final dos anos 90.
- Com computadores mais robustos, com maior capacidade de raciocínio, houve um resurgimento de modelos de redes neurais para resolver problemas da ciência cognitiva:
 - processamento de linguagem natural,
 - processamento de fala e visão.
 - reconhecimento de padrões,
 - voz, imagens e objetos.



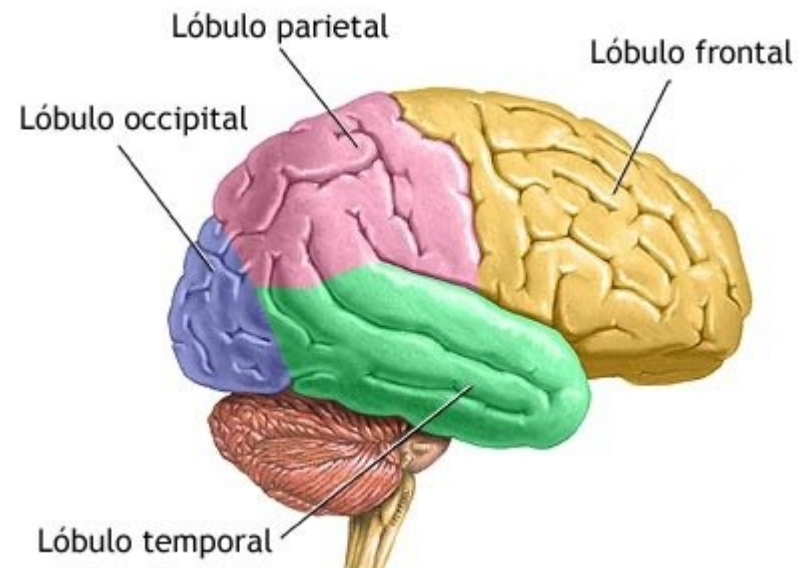
Redes Neurais Artificiais (RNA)

- Modelos versáteis que podem ser aplicadas a quase todas as tarefas de aprendizagem:
 - classificação, previsão numérica e como também reconhecimento não supervisionado de padrões.
- São melhor aplicadas a problemas onde os dados de entrada e os dados de saída são bem definidos ou, pelo menos, bastante simples, mas o processo que relaciona a entrada com a saída é extremamente complexo



Cérebro Humano

- Somos capazes aprender novas línguas, aprender a fazer cálculos, criar novas habilidades como tocar um instrumento musical, pilotar maquinários, etc.
- O cérebro humano é um sistema complexo e tem sido extensamente estudado.
- Porém ainda não somos capazes de entender completamente o seu funcionamento.

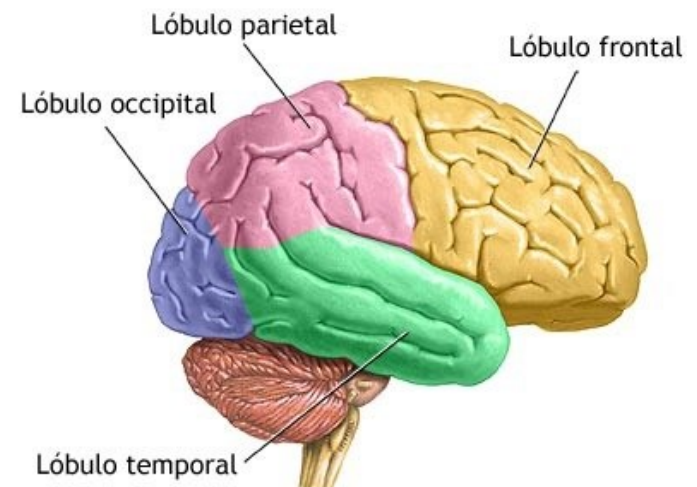


Cérebro Humano

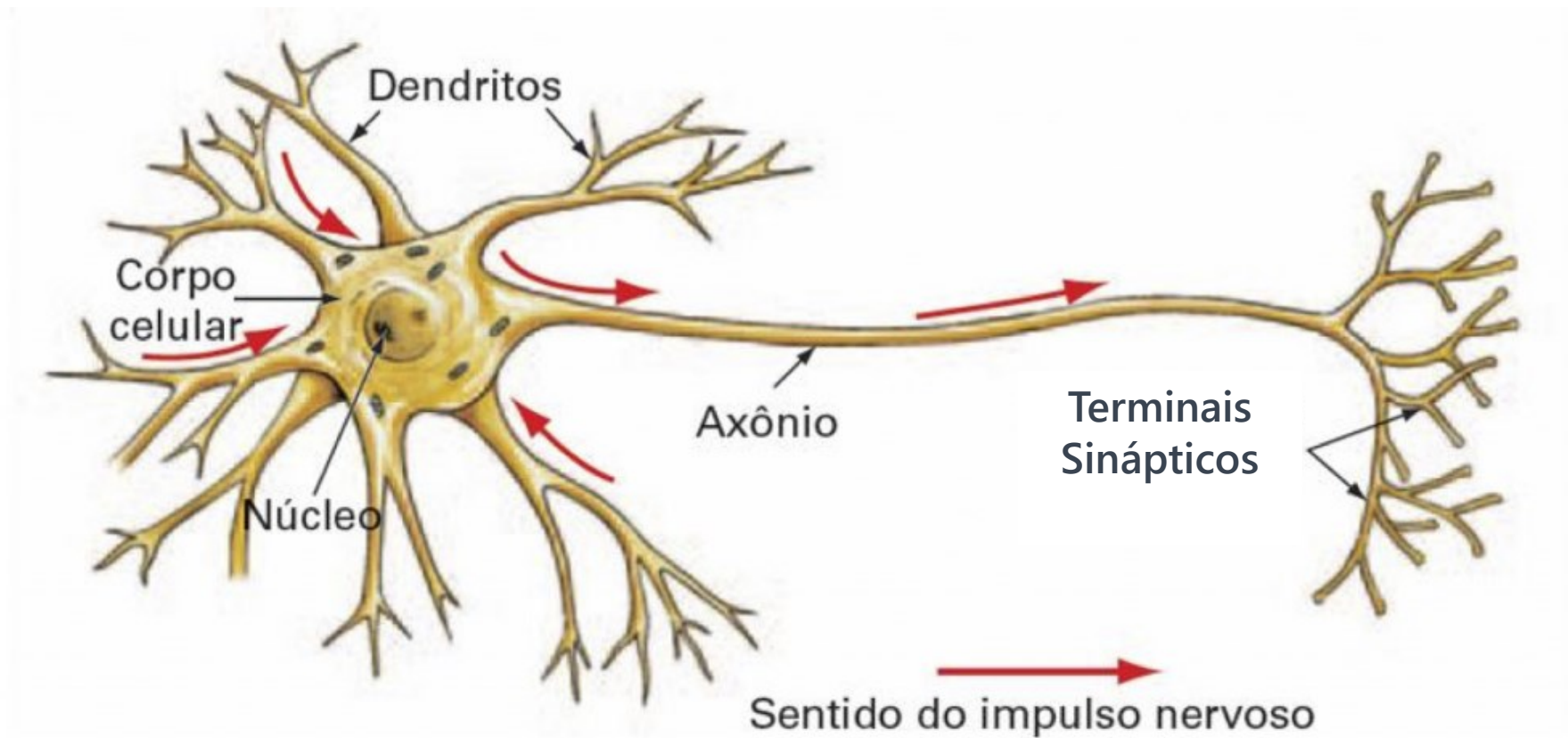
- Faz milhares de operações por segundo.
- Possui cerca de 86 bilhões de neurônios.
- Seu número de conexões, conhecido como **sinapse**, ultrapassa 60 trilhões, possibilitando a formação de uma rede muito complexa.
- As células neuronais controlam todos os sinais e ações dos seres vivos.

Cérebro de um cão ~ 530 milhões de neurônios

Algumas Redes Neurais Artificiais –
algumas centenas de neurônios

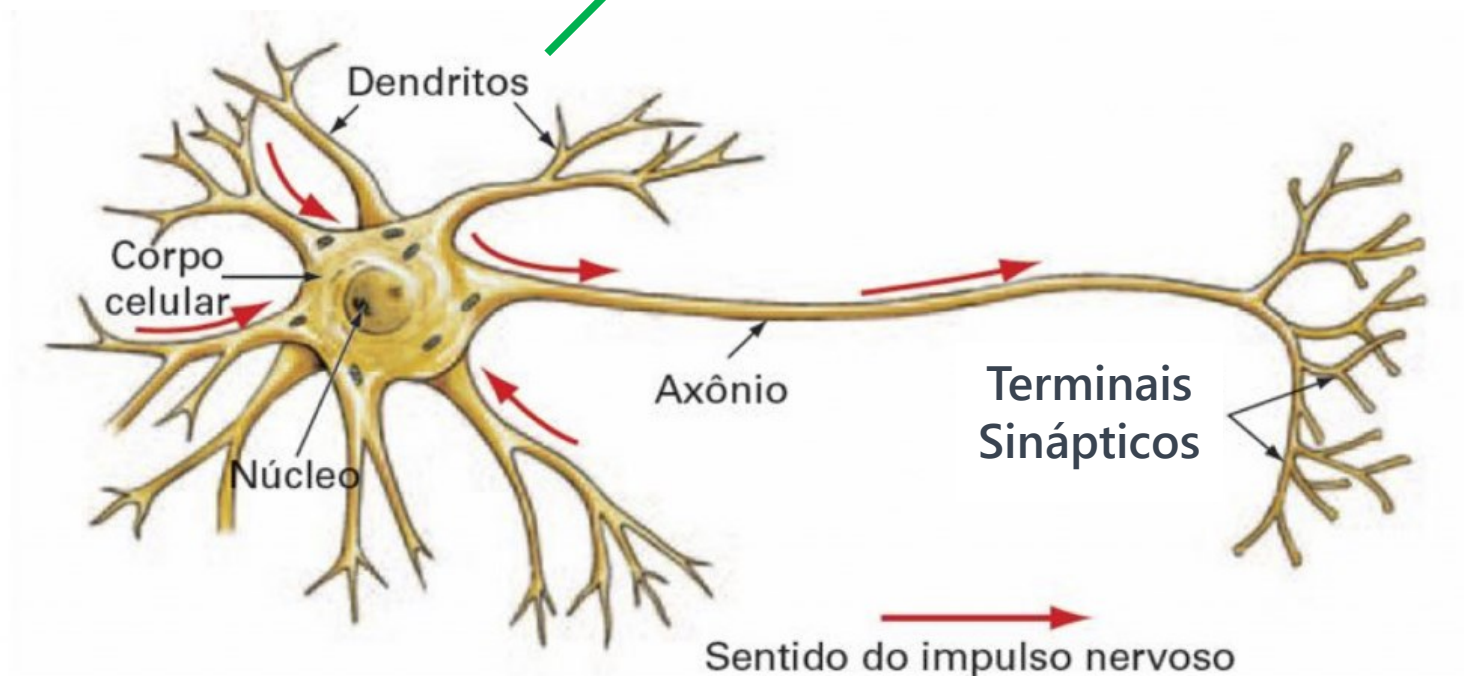


Neurônio Biológico



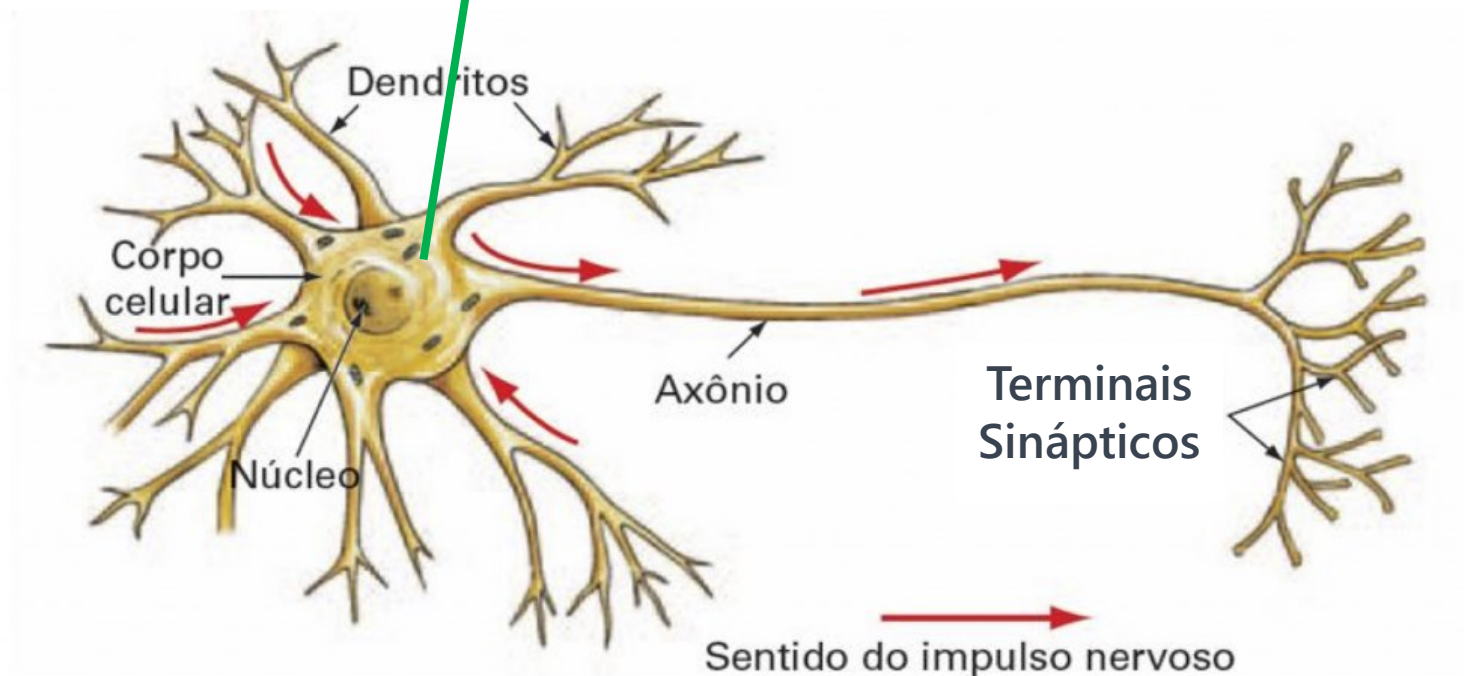
Neurônio Biológico

O neurônio recebe sinais de outros neurônios através dos **dendritos**, a ele conectados por meio das **sinapses** (região onde dois neurônios se interligam).



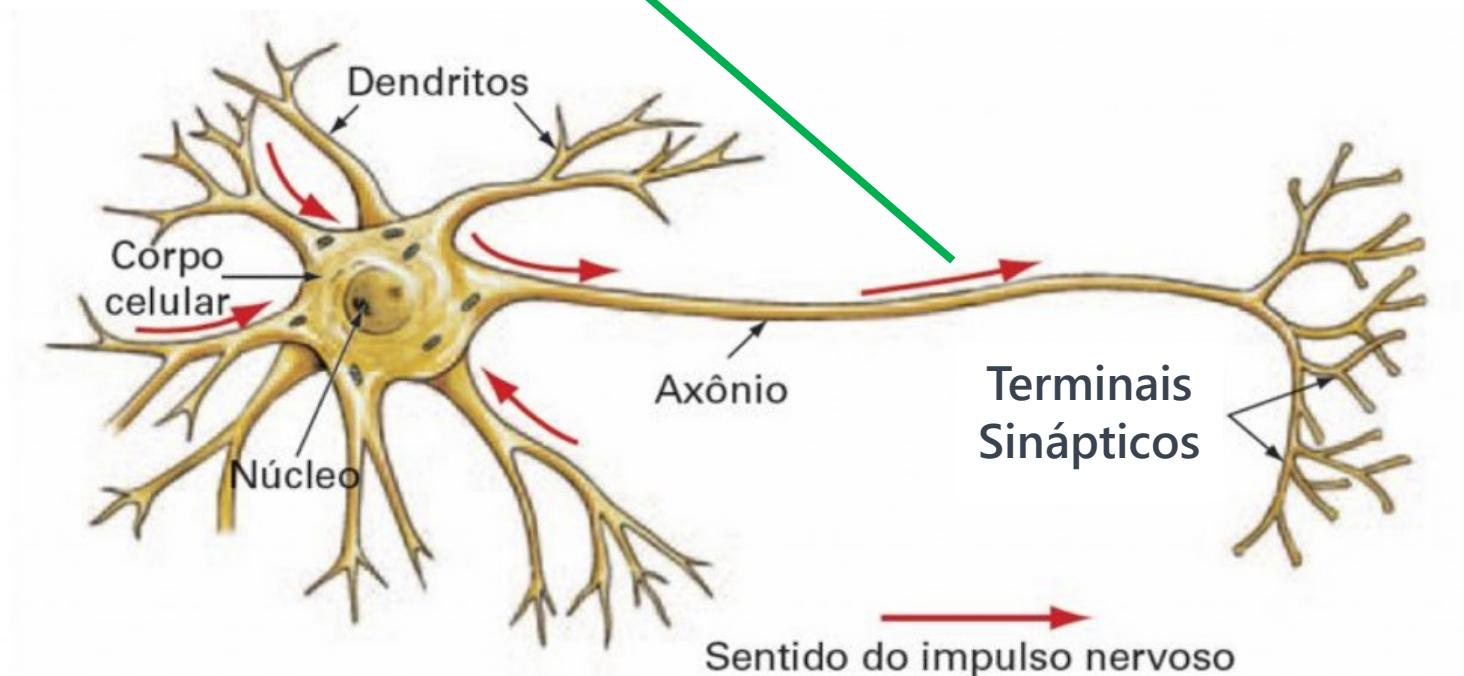
Neurônio Biológico

Os sinais são acumulados no **corpo celular** do neurônio.



Neurônio Biológico

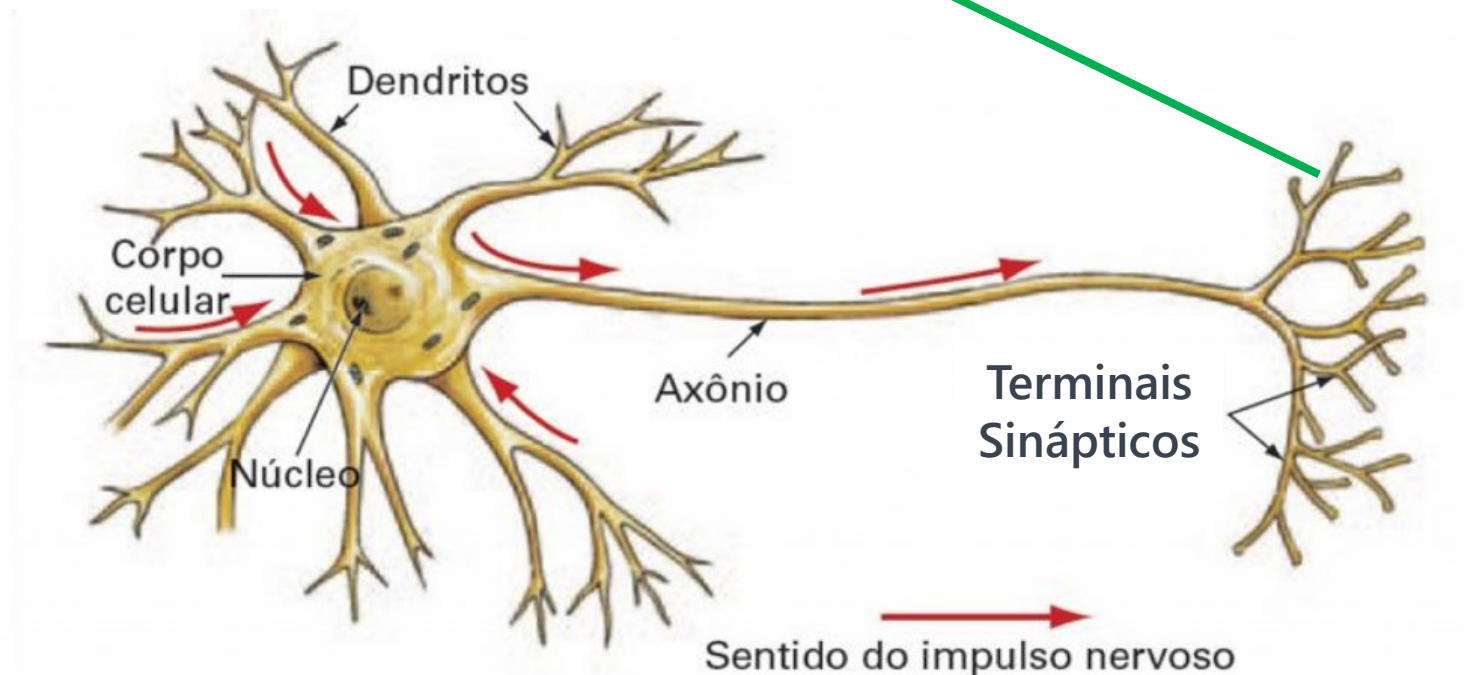
Quando a soma dos sinais excede um limiar, o sinal é enviado para o **axônio**.
Ocorre uma reação química que resulta num pulso elétrico.



Neurônio Biológico

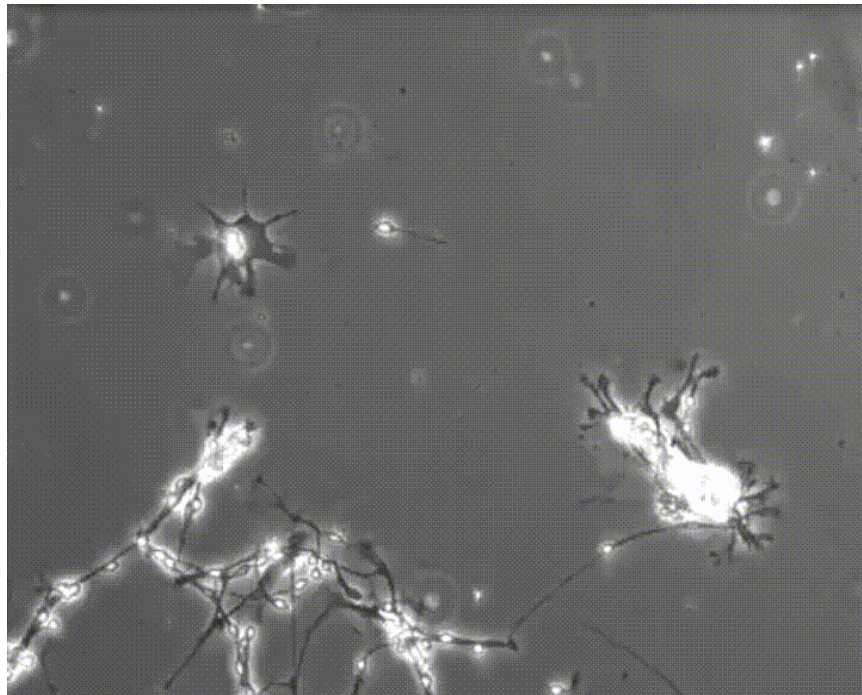
As **sinapses** tem um peso que pode ser:

- excitatório: incrementam a soma dos sinais.
- inibidor: decrementam.



Neurônio Biológico

- Os neurônios podem mudar a natureza e o número de conexões com outros neurônios em resposta a eventos que ocorram.
 - Após vivenciar alguma experiência nova, o cérebro altera as conexões, o que indica um aprendizado.
- Quando a mesma experiência é repetida várias vezes, então as ligações são reforçadas.



Neurônio Artificial

- McCulloch e Pitts em 1943, propuseram o primeiro neurônio artificial, com o objetivo de simular o comportamento do neurônio biológico:



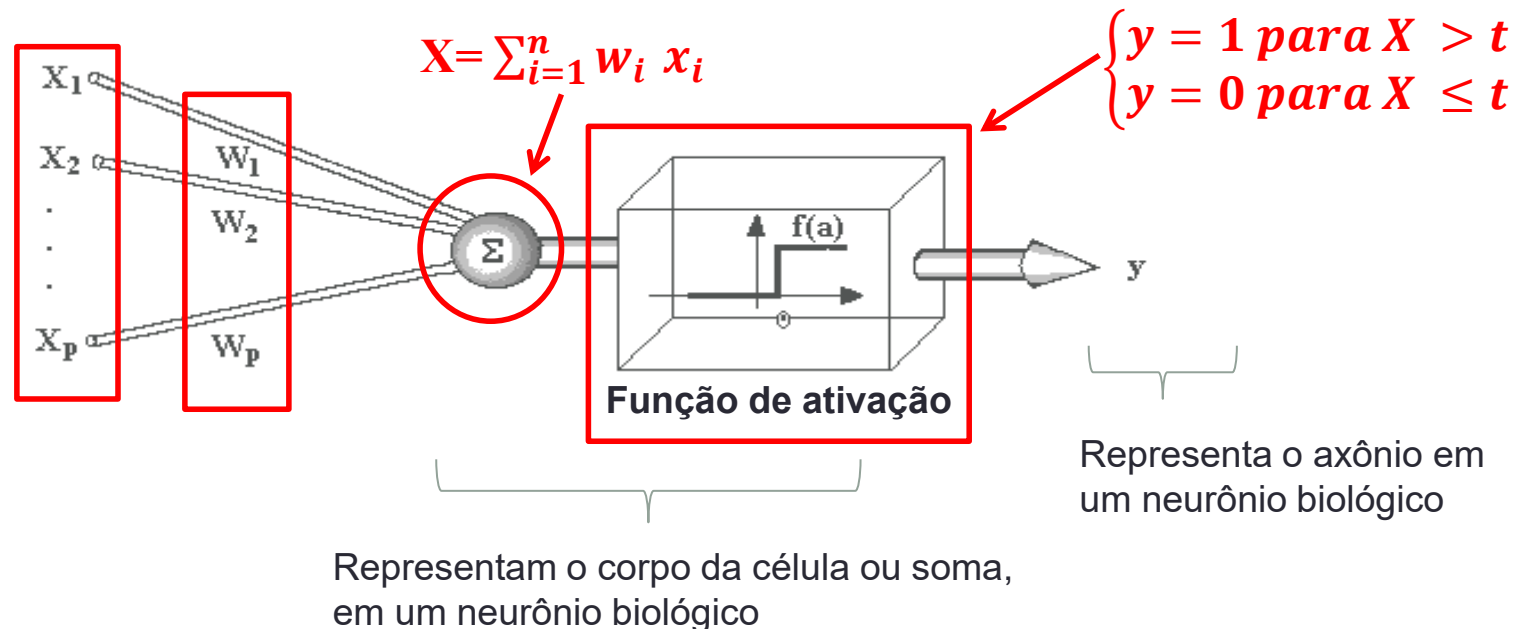
Warren McCulloch
psiquiatra e neuroanatomista

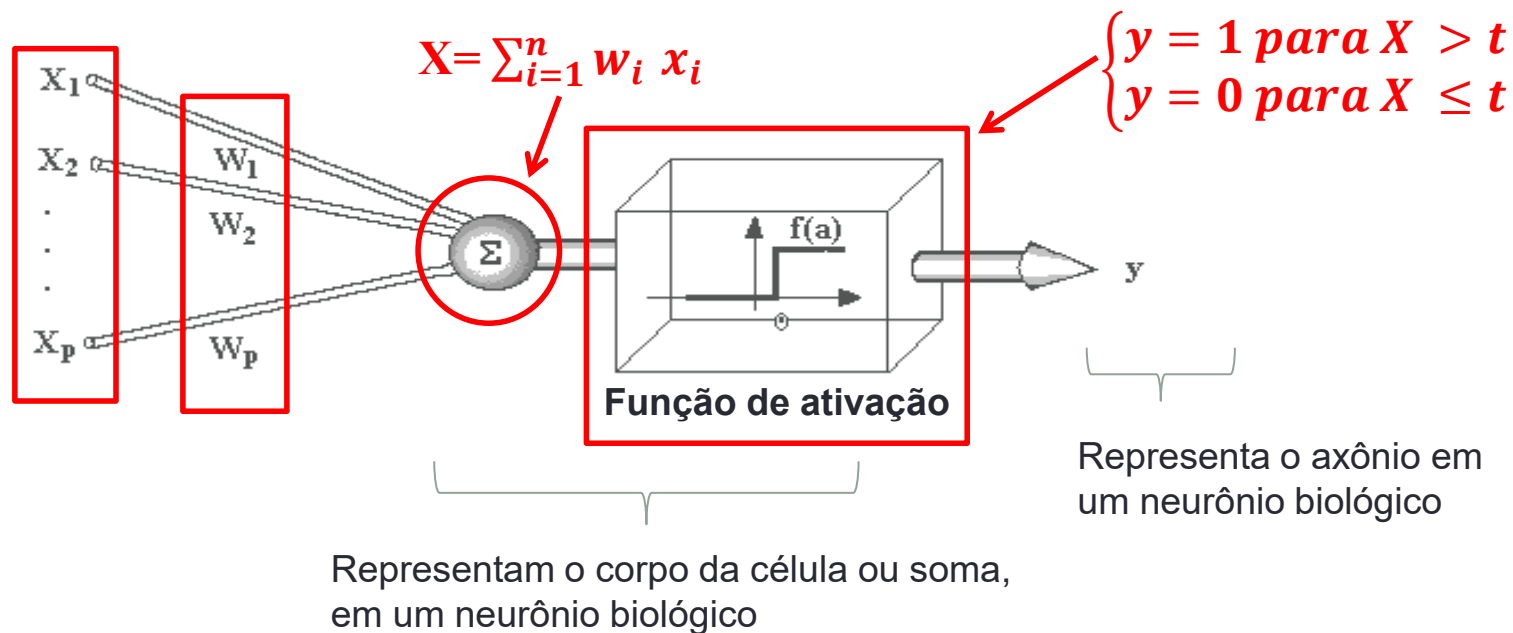
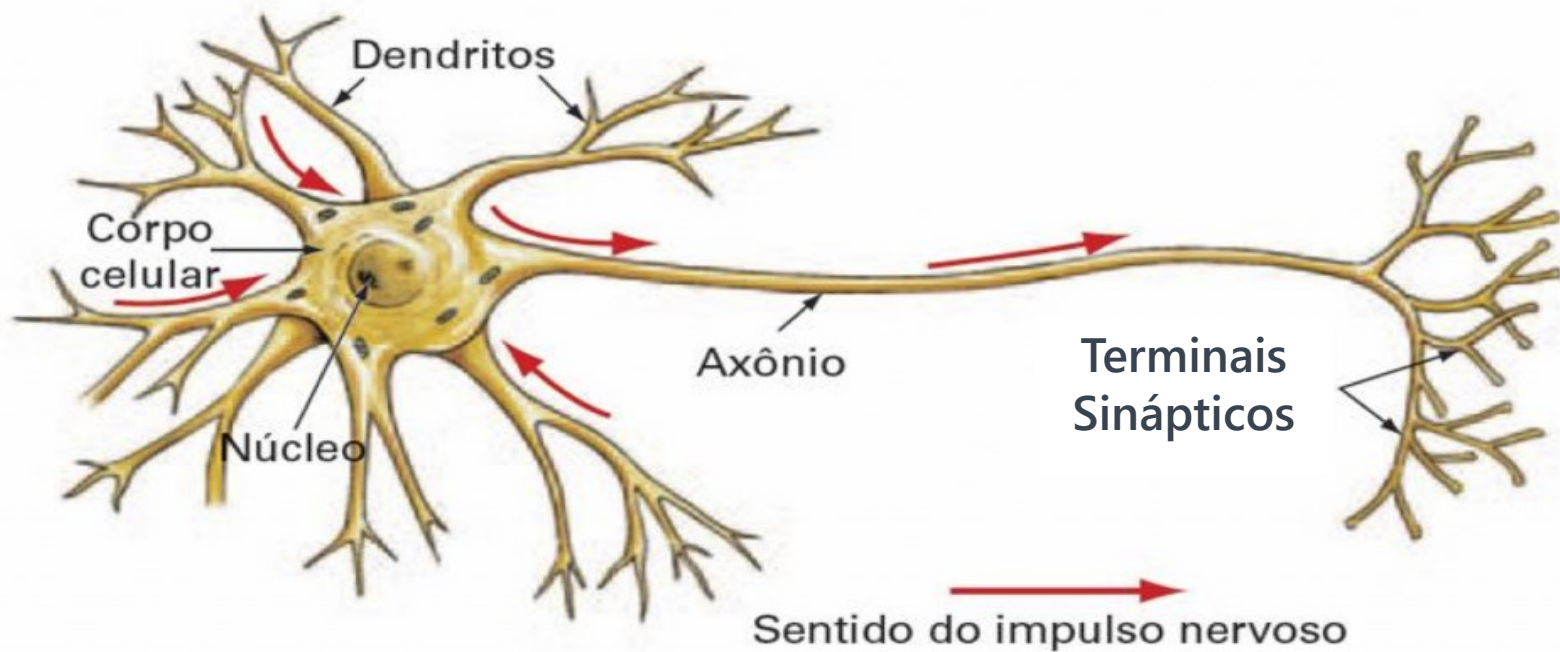


Walter Pitts
matemático

Neurônio Artificial

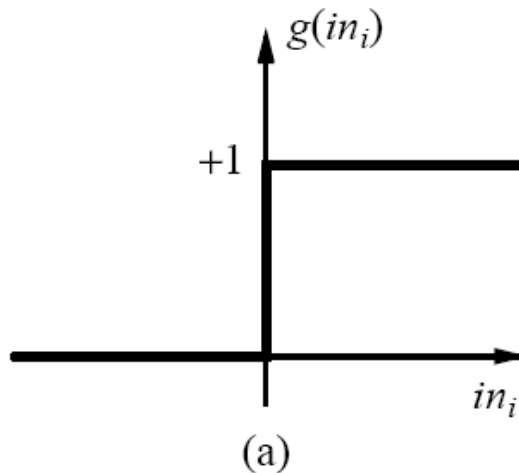
- Sinais são apresentados à entrada (x_1, x_2, \dots, x_p);
- Cada sinal é multiplicado por um peso (w_1, w_2, \dots, w_p), que indica a sua influência na saída da unidade;
- É feita a soma ponderada dos sinais que produz um nível de atividade;
- Se este nível de atividade exceder um certo limite (threshold - t) a unidade produz uma determinada resposta de saída.



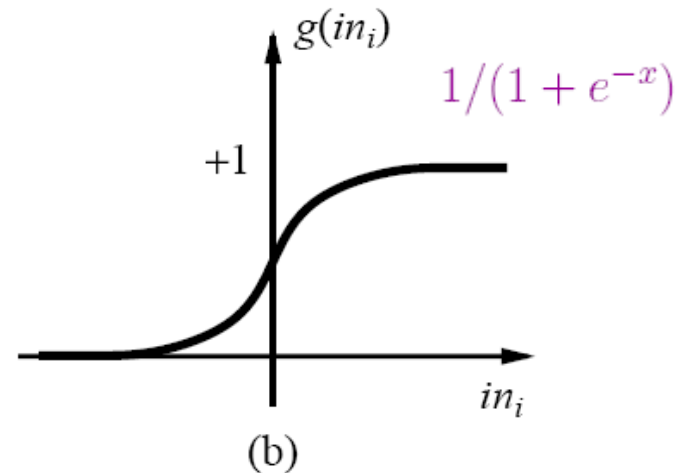


Neurônios Artificiais

- Existem várias funções de ativação, as mais comuns:
 - **Degrau:** usada em modelos como o Perceptron
 - **Sigmoide:** usada para representar funções não-lineares, em modelos como a retropropagação.



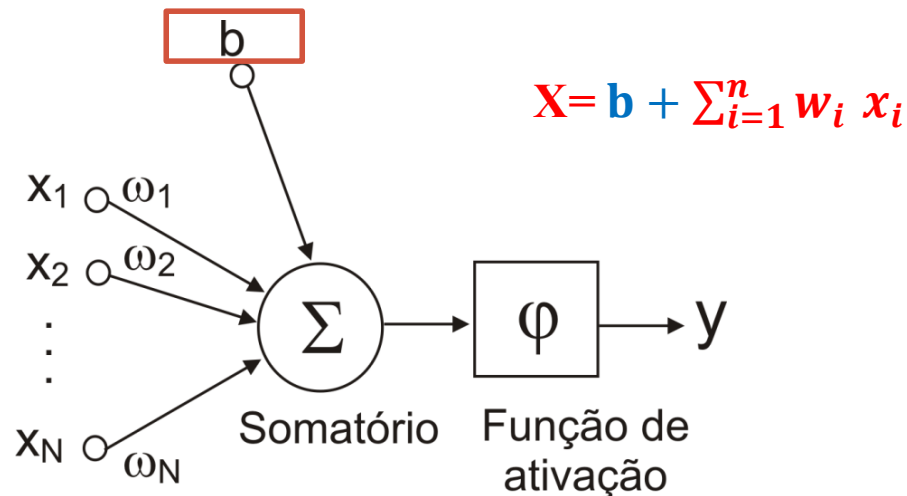
Função Degrau



Função sigmóide

Neurônio Artificial: Bias (viés)

- Também pode incluir uma polarização ou **bias** de entrada.
- Esta variável é incluída ao somatório da função de ativação, com o intuito de aumentar o grau de liberdade desta função e, conseqüentemente, a capacidade de aproximação da rede.
- Possibilita que o neurônio apresente uma saída não nula, mesmo se todas as entrada forem nulas.
- É uma constante (valor positivo (+1) ou negativo (-1)).

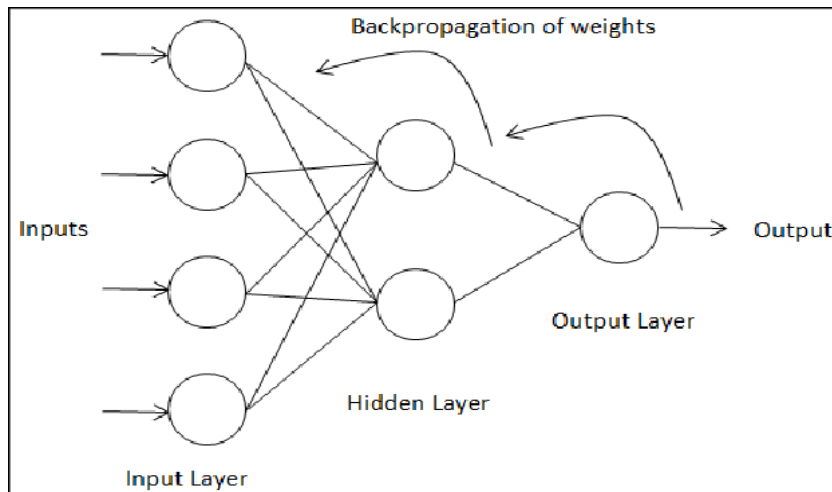


Aprendizagem em Redes Neurais:

1. Perceptron e Multilayer Perceptron:



2. Retropropagação (Backpropagation):



1. Rede Perceptron

- Algoritmo simples (uma camada) desenvolvido nos anos 50 por Frank Rosenblatt.
- A ideia deste algoritmo, e da maioria dos algoritmos de aprendizado de Redes Neurais Artificiais, é ajustar os pesos da rede para minimizar a medida de erro em um conjunto de treinamento.
- O algoritmo faz atualizações iterativamente até chegar aos **pesos corretos**, produzindo assim, a saída correta para cada exemplo.

1. Rede Perceptron

- O aprendizado se dá alterando os valores dos pesos, a partir do **erro** δ da rede neural.
 - É a diferença ente o valor desejado na saída d e o valor que a rede neural fornece Y .

$$\delta = d - Y$$

- O algoritmo faz atualizações iterativamente até chegar aos **pesos corretos**.

$$w_i = w_i + n * \delta * x_i$$

Peso antigo erro dado de entrada

taxa de aprendizado (*learning rate*)

- **Taxa de aprendizado:** Controla a velocidade em que os pesos são ajustados a cada passo do treinamento no intervalo $[0,1]$ (valor baixo - próximo a zero)

1. Rede Perceptron

- Regra de aprendizado:

$$w_i = w_i + n * \delta * x_i$$

onde,

$$\delta = d - Y$$

Valor desejado

Valor obtido

- Se a saída do perceptron não estiver correta ($d \neq Y$):
 - Os pesos w_i são alterados de forma que a saída do perceptron para os novos pesos seja próxima de d .
- O algoritmo vai classificar corretamente se:
 - O conjunto de treinamento é linearmente separável.
 - n é suficientemente pequeno.

Exemplo: Treinando um Neurônio

- Implementação da porta lógica AND

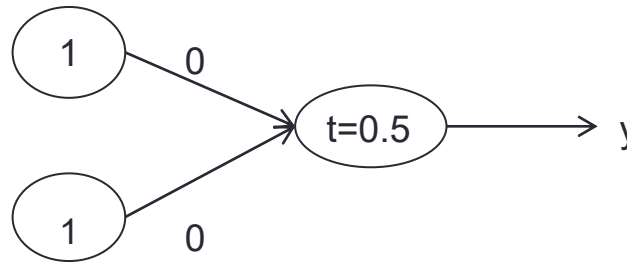
Tabela Verdade - AND

A	B	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Entradas [1,1]
Pesos iniciais [0,0]

Limiar (*Threshold* - t) = 0.5

Taxa de aprendizado
(*Learning Rate*) = 0.5



$$Y = \begin{cases} 1 & \text{para } X > t \\ 0 & \text{para } X \leq t \end{cases}$$

$$X = \sum_{i=1}^n w_i x_i$$

Exemplo: Treinando um Neurônio

- Implementação da porta lógica AND

- Passo 1:** Aplicar a função de Soma (X)

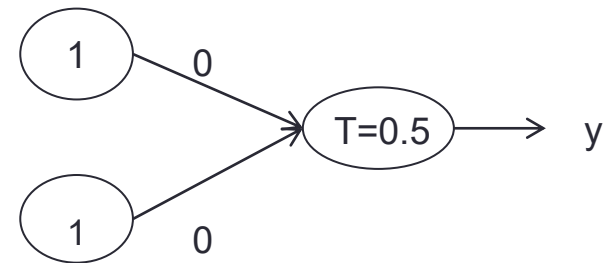
$$X = 1 \cdot 0 + 1 \cdot 0 = 0$$

- Passo 2:** Aplicar a função de Transferência

- $X \leq 0,5 \rightarrow y = 0$
- $X > 0,5 \rightarrow y = 1$

- Transferido 0 para a saída.

Erro!!!!



$$X = \sum_{i=1}^n w_i x_i$$

$$Y = \begin{cases} 1 & \text{para } X > t \\ 0 & \text{para } X \leq t \end{cases}$$

Limiar (t) = 0.5

Taxa de aprendizado (n) = 0.5

Exemplo: Treinando um Neurônio

- Implementação da porta lógica AND
- **Passo 3:** Ajuste do peso

Equação do erro:

$$\delta = d - Y$$

onde,

d = saída desejada

Y = saída obtida

Fator de correção:

$$F = n * \delta * x_i$$

onde

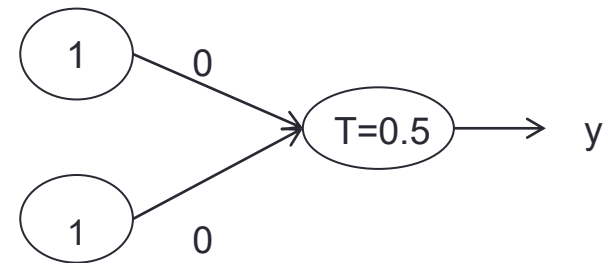
n taxa de aprendizado

x é a entrada

δ é o erro

Equação do ajuste:

$$w_{novo} = w + F$$



$$X = \sum_{i=1}^n w_i x_i$$

$$Y = \begin{cases} 1 & \text{para } X > t \\ 0 & \text{para } X \leq t \end{cases}$$

Limiar (t) = 0.5

Taxa de aprendizado (n) = 0.5

Exemplo: Treinando um Neurônio

- Implementação da porta lógica AND

- **Passo 3:** Ajuste do peso

- Calcular o erro: $\delta = 1 - 0 = 1$

- *Calcular o fator de correção:*

$$F1 = n * \delta * x_1$$

- $F1 = 0,5 * 1 * 1$

- $F1 = 0,5$

$$F2 = n * \delta * x_2$$

- $F2 = 0,5 * 1 * 1$

- $F2 = 0,5$

Equação do erro:

$$\delta = d - Y$$

onde,

d = saída desejada

Y = saída obtida

Fator de correção:

$$F = n * \delta * x_i$$

onde

n taxa de aprendizado

x é a entrada

δ é o erro

Equação do ajuste:

$$w_{novo} = w + F$$

Exemplo: Treinando um Neurônio

- Implementação da porta lógica AND

- **Passo 3:** Ajuste do peso

- Calcular o erro: $\delta = 1 - 0 = 1$

- *Calcular o novo peso:*

$$w1_{novo} = w1 + F1$$

- $w1_{novo} = 0 + 0,5$

- $w1_{novo} = 0,5$

$$w2_{novo} = w1 + F2$$

- $w2_{novo} = 0 + 0,5$

- $w2_{novo} = 0,5$

Equação do erro:

$$\delta = d - Y$$

onde,

d = saída desejada

Y = saída obtida

Fator de correção:

$$F = n * \delta * x_i$$

onde

n taxa de aprendizado

x é a entrada

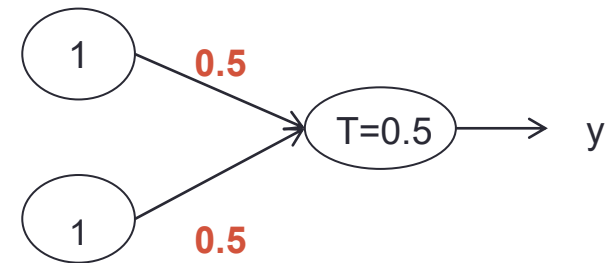
δ é o erro

Equação do ajuste:

$$w_{novo} = w + F$$

Exemplo: Treinando um Neurônio

- Implementação da porta lógica AND
 - Agora pesos iniciais [0,5, 0,5]
- **Passo 1:** Aplicar a função de Soma (X)
 $X = 1 \cdot 0,5 + 1 \cdot 0,5 = 1$
- **Passo 2:** Aplicar a função de Transferência
 - $X \leq 0,5 \rightarrow y = 0$
 - $X > 0,5 \rightarrow y = 1$
- Transferido 1 para a saída.
Correto!!!!



$$X = \sum_{i=1}^n w_i x_i$$

$$Y = \begin{cases} 1 & \text{para } X > t \\ 0 & \text{para } X \leq t \end{cases}$$

Limiar (t) = 0.5

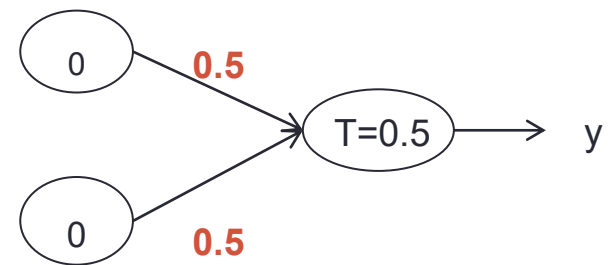
Taxa de aprendizado (n) = 0.5

Exercício

- Continuar treinando a rede
 - Para as entradas
 - [0, 0]
 - [0, 1]
 - [1, 0]
 - e pesos [0,5, 0,5]

Exercício: RESPOSTA

- Implementação da porta lógica AND
 - Agora pesos iniciais [0,5, 0,5]
- **Passo 1:** Aplicar a função de Soma (X)
 $X = 0 \cdot 0.5 + 0 \cdot 0.5 = 0$
- **Passo 2:** Aplicar a função de Transferência
 - $X \leq 0.5 \rightarrow y = 0$
 - $X > 0.5 \rightarrow y = 1$
- Transferido 0 para a saída.
Correto!!!!



$$X = \sum_{i=1}^n w_i x_i$$

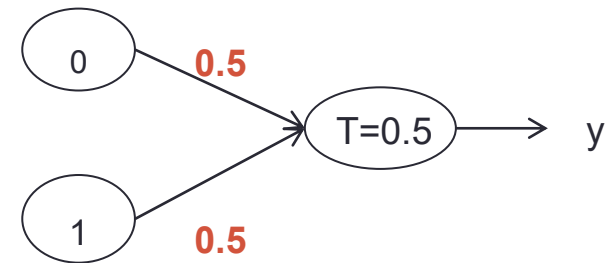
$$Y = \begin{cases} 1 & \text{para } X > t \\ 0 & \text{para } X \leq t \end{cases}$$

Limiar (t) = 0.5

Taxa de aprendizado (n) = 0.5

Exercício: RESPOSTA

- Implementação da porta lógica AND
 - Agora pesos iniciais [0,5, 0,5]
- **Passo 1:** Aplicar a função de Soma (X)
 $X = 0 \cdot 0,5 + 1 \cdot 0,5 = 0,5$
- **Passo 2:** Aplicar a função de Transferência
 - $X \leq 0,5 \rightarrow y = 0$
 - $X > 0,5 \rightarrow y = 1$
- Transferido 0 para a saída.
Correto!!!!



$$X = \sum_{i=1}^n w_i x_i$$

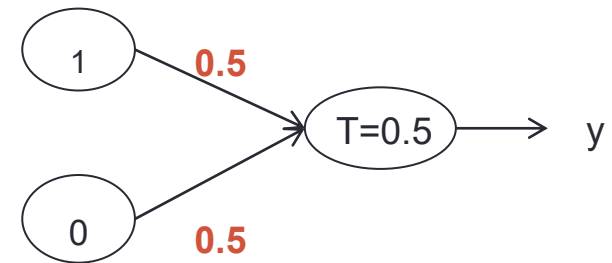
$$Y = \begin{cases} 1 & \text{para } X > t \\ 0 & \text{para } X \leq t \end{cases}$$

Limiar (t) = 0.5

Taxa de aprendizado (n) = 0.5

Exercício: RESPOSTA

- Implementação da porta lógica AND
 - Agora pesos iniciais [0,5, 0,5]
- **Passo 1:** Aplicar a função de Soma (X)
 $X = 1 \cdot 0.5 + 0 \cdot 0.5 = 0$
- **Passo 2:** Aplicar a função de Transferência
 - $X \leq 0.5 \rightarrow y = 0$
 - $X > 0.5 \rightarrow y = 1$
- Transferido 0 para a saída.
Correto!!!!



$$X = \sum_{i=1}^n w_i x_i$$

$$Y = \begin{cases} 1 & \text{para } X > t \\ 0 & \text{para } X \leq t \end{cases}$$

Limiar (t) = 0.5

Taxa de aprendizado (n) = 0.5

1. Rede Perceptron: Algoritmo

Treinamento

Iniciar todas as conexões com $w_i = 0$ (ou aleatórios)

Repita

Para cada padrão de treinamento (X, d)

faça

Calcular a saída y

Se $(d \neq y)$

então atualizar pesos

$$w_i = w_i + n * \delta * x_i$$

até o erro ser aceitável

1. Rede Perceptron: Algoritmo

Teste

Para cada padrão de 1 a p
faça

Apresentar X_p à entrada da rede
Calcular a saída y

Se $y \geq t$

então $X_p \in \text{Classe 1}$

senão $X_p \in \text{Classe 2}$

Exercício

1) Treinar uma rede *Perceptron* para classificar os seguintes padrões:

Entrada:



-1 -1 -1



+1 +1 +1

Saída desejada:



-1



+1

$$X = \sum_{i=1}^n w_i x_i$$

$$Y = \begin{cases} 1 & \text{para } X \geq t \\ -1 & \text{para } X < t \end{cases}$$

Equação do erro:

$$\delta = d - Y$$

onde,

d = saída desejada

Y = saída obtida

Fator de correção:

$$F = n * \delta * x_i$$

onde

n taxa de aprendizado

x é a entrada

δ é o erro

Equação do ajuste:

$$w_{\text{novo}} = w + F$$

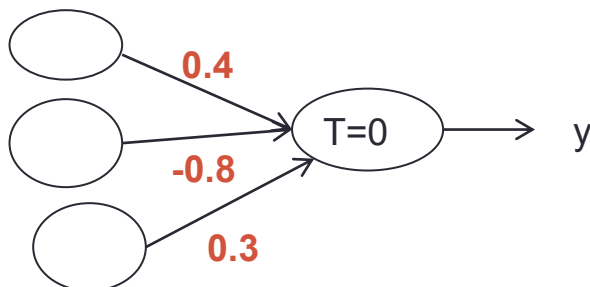
$n = 0.2$

$t = 0$

$w_0 = 0.4$

$w_1 = -0.8$

$w_2 = 0.3$



Exercício: RESPOSTA

1) Treinar uma rede *Perceptron* para classificar os seguintes padrões:

Entrada:



-1 -1 -1

Saída desejada:



-1

$$(-1)*(0.4) + (-1)*(-0.8) + (-1)*(0.3) = 0.1$$

Y = +1 (uma vez que $0.1 \geq 0$)

como ($d \neq y$), atualizar pesos

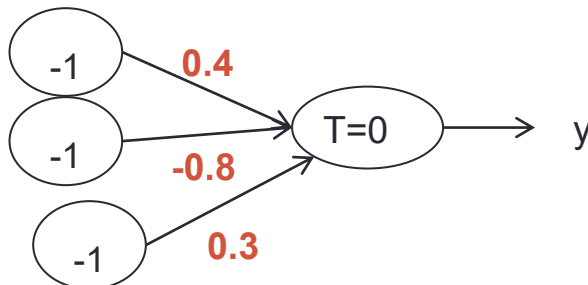
$$n = 0.2$$

$$t = 0$$

$$w_0 = 0.4$$

$$w_1 = -0.8$$

$$w_2 = 0.3$$



$$X = \sum_{i=1}^n w_i x_i$$

$$Y = \begin{cases} 1 & \text{para } X \geq t \\ -1 & \text{para } X < t \end{cases}$$

Equação do erro:

$$\delta = d - Y$$

onde,

d = saída desejada

Y = saída obtida

Fator de correção:

$$F = n * \delta * x_i$$

onde

n taxa de aprendizado

x é a entrada

δ é o erro

Equação do ajuste:

$$w_{\text{novo}} = w + F$$

Exercício: RESPOSTA

1) Treinar uma rede *Perceptron* para classificar os seguintes padrões:

Entrada:



-1 -1 -1

Saída desejada:



-1

$$(-1)*(0.4) + (-1)*(-0,8) + (-1)*(0.3) = 0.1$$

$Y = +1$, logo ($d \neq y$)

$$\delta = (-1 - (+1)) = -2$$

$$F = 0.2 * (-2) * (-1) = 0.4$$

PESO NOVO

$$w_0 = 0.4 + 0.4 = 0.8$$

$$w_1 = -0.8 + 0.4 = -0.4$$

$$w_2 = 0.3 + 0.4 = 0.7$$

$$n = 0.2$$

$$t = 0$$

PESO ANTIGO

$$w_0 = 0.4$$

$$w_1 = -0.8$$

$$w_2 = 0.3$$

$$X = \sum_{i=1}^n w_i x_i$$

$$Y = \begin{cases} 1 & \text{para } X \geq t \\ -1 & \text{para } X < t \end{cases}$$

Equação do erro:

$$\delta = d - Y$$

onde,

d = saída desejada

Y = saída obtida

Fator de correção:

$$F = n * \delta * x_i$$

onde

n taxa de aprendizado

x é a entrada

δ é o erro

Equação do ajuste:

$$w_{\text{novo}} = w + F$$

Exercício: RESPOSTA

1) Treinar uma rede *Perceptron* para classificar os seguintes padrões:

Entrada:



-1 -1 -1

Saída desejada:



-1

$$(-1)*(0.8) + (-1)*(-0.4) + (-1)*(0.7) = -1.1$$

Y = -1 (uma vez que $-1.1 < 0$)

como ($d = y$), não necessita atualizar pesos

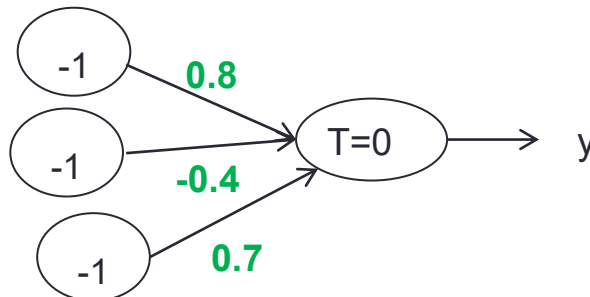
$$n = 0.2$$

$$t = 0$$

$$w_0 = 0.8$$

$$w_1 = -0.4$$

$$w_2 = 0.7$$



$$X = \sum_{i=1}^n w_i x_i$$

$$Y = \begin{cases} 1 & \text{para } X \geq t \\ -1 & \text{para } X < t \end{cases}$$

Equação do erro:

$$\delta = d - Y$$

onde,

d = saída desejada

Y = saída obtida

Fator de correção:

$$F = n * \delta * x_i$$

onde

n taxa de aprendizado

x é a entrada

δ é o erro

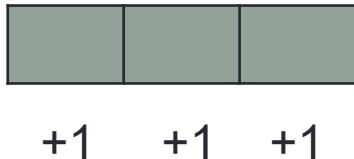
Equação do ajuste:

$$w_{\text{novo}} = w + F$$

Exercício: RESPOSTA

1) Treinar uma rede *Perceptron* para classificar os seguintes padrões:

Entrada:



Saída desejada:



$$(+1)*(0.8) + (+1)*(-0.4) + (+1)*(0.7) = 1.1$$

Y = +1 (uma vez que $1.1 \geq 0$)

como ($d = y$), não necessita atualizar pesos

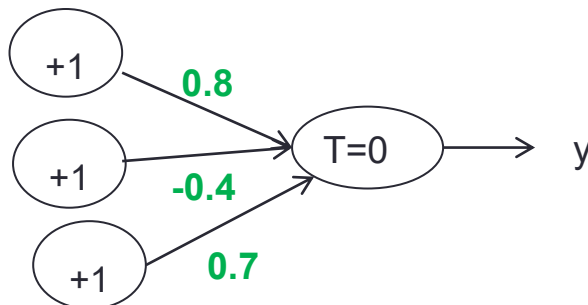
$$n = 0.2$$

$$t = 0$$

$$w_0 = 0.8$$

$$w_1 = -0.4$$

$$w_2 = 0.7$$



$$X = \sum_{i=1}^n w_i x_i$$

$$Y = \begin{cases} 1 & \text{para } X \geq t \\ -1 & \text{para } X < t \end{cases}$$

Equação do erro:

$$\delta = d - Y$$

onde,

d = saída desejada

Y = saída obtida

Fator de correção:

$$F = n * \delta * x_i$$

onde

n taxa de aprendizado

x é a entrada

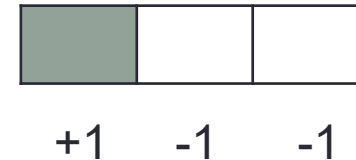
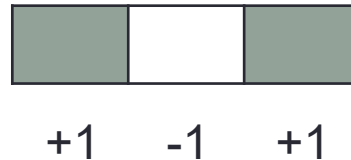
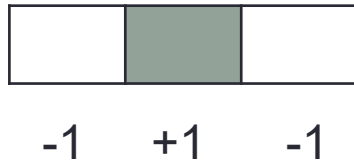
δ é o erro

Equação do ajuste:

$$w_{\text{novo}} = w + F$$

Exercício

2) Utilizar a rede treinada para classificar os padrões: **Testar**



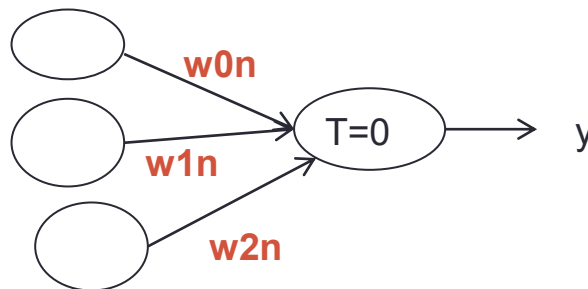
$n = 0.2$

$t = 0$

w_0 novo

w_1 novo

w_2 novo

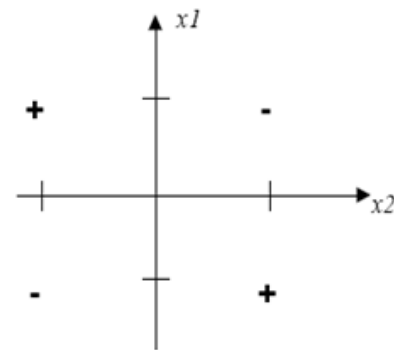
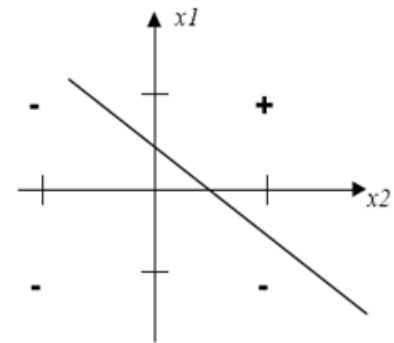


$$X = \sum_{i=1}^n w_i x_i$$

$$Y = \begin{cases} 1 & \text{para } X \geq t \\ -1 & \text{para } X < t \end{cases}$$

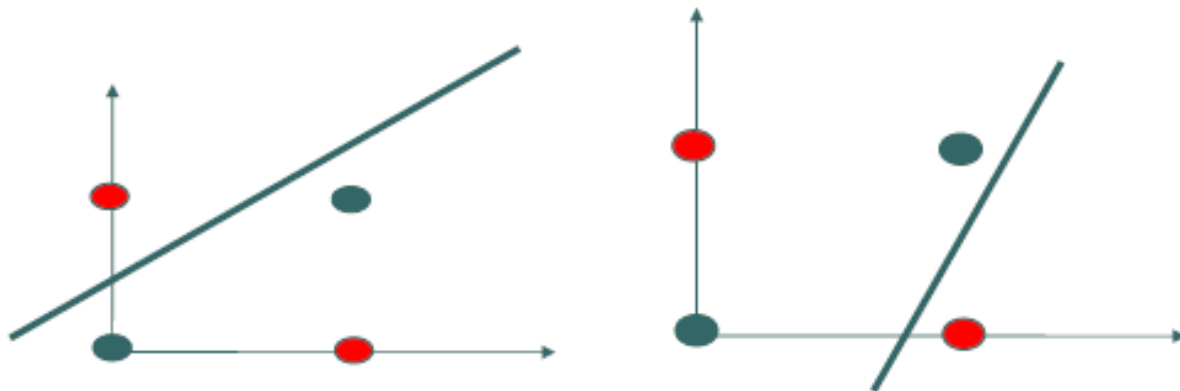
1. Rede Perceptron: Limitações

- Um único Perceptron consegue resolver somente funções linearmente separáveis.
- Em funções não linearmente separáveis o perceptron não consegue gerar um hiperplano para separar os dados.



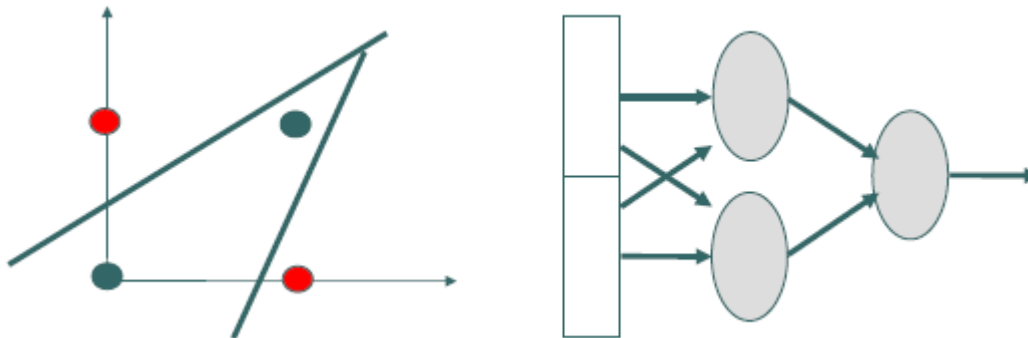
Redes Multicamadas

- Perceptrons expressam somente superfícies de decisão linear.
- Entretanto, é possível combinar vários perceptrons lineares para gerar superfícies de decisão mais complexas.
 - **Camada 1:** uma rede *Perceptron* para cada grupo de entradas linearmente separáveis.



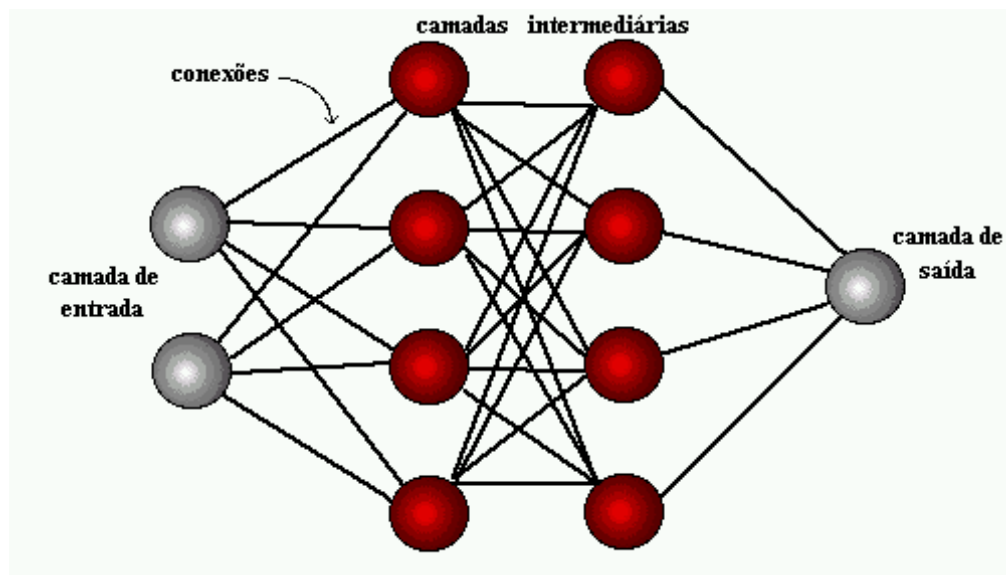
Redes Multicamadas

- Perceptrons expressam somente superfícies de decisão linear.
- Entretanto, é possível combinar vários perceptrons lineares para gerar superfícies de decisão mais complexas.
- **Camada 2:** uma rede combinando as saídas das redes da 1ª camada, produzindo a classificação final



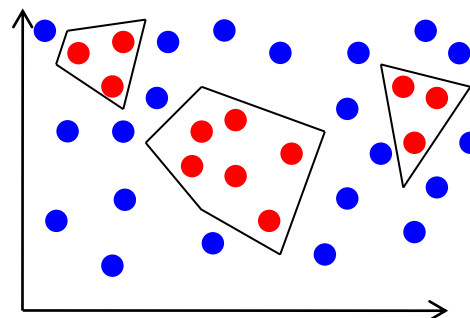
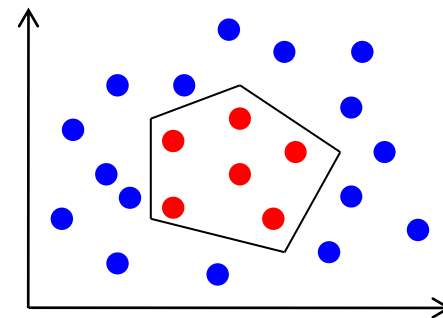
Redes Multicamadas

- **Camada de Entrada:** padrões são apresentados à rede;
- **Camadas Intermediárias ou ocultas:** onde é feita a maior parte do processamento, através das conexões ponderadas.
 - Podem ser consideradas como extratoras de características;
- **Camada de Saída:** onde o resultado final é concluído e apresentado.



Redes Multicamadas

- Adicionar uma camada oculta a rede permite que a rede possa gerar uma função de *convex hull*.
 - *Envoltória Convexa (convex hull)*: o conjunto de todas as combinações convexas destes vetores
- Duas camadas ocultas permite a rede gerar uma função com diferentes *convex hulls*.

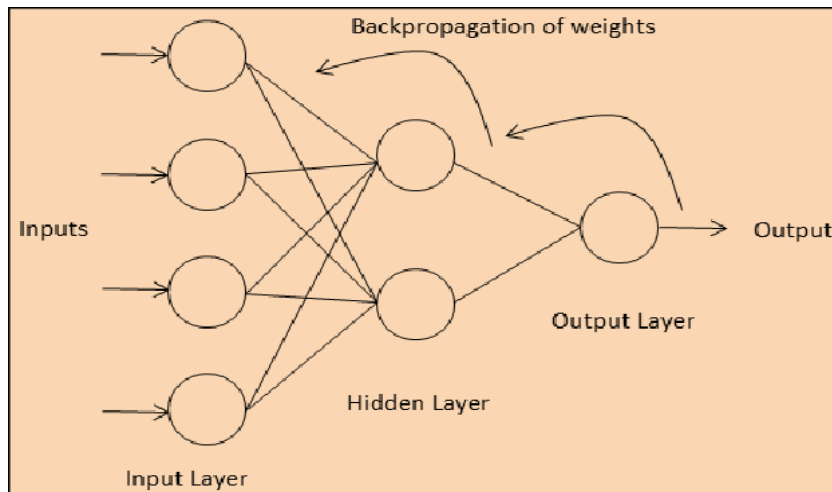


Aprendizagem em Redes Neurais:

1. Perceptron e Multilayer Perceptron:

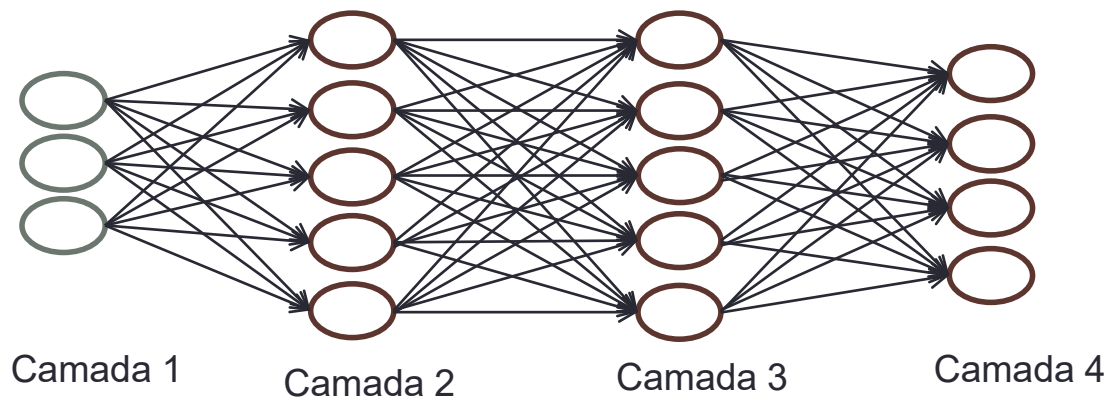


2. Retropropagação (Backpropagation):



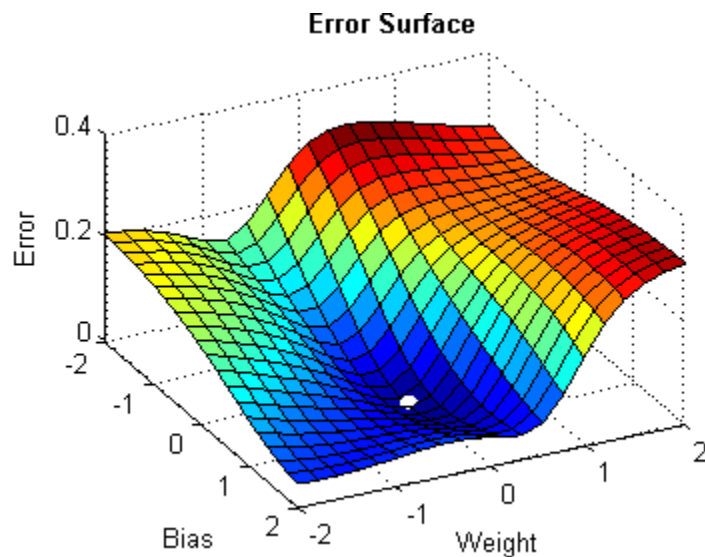
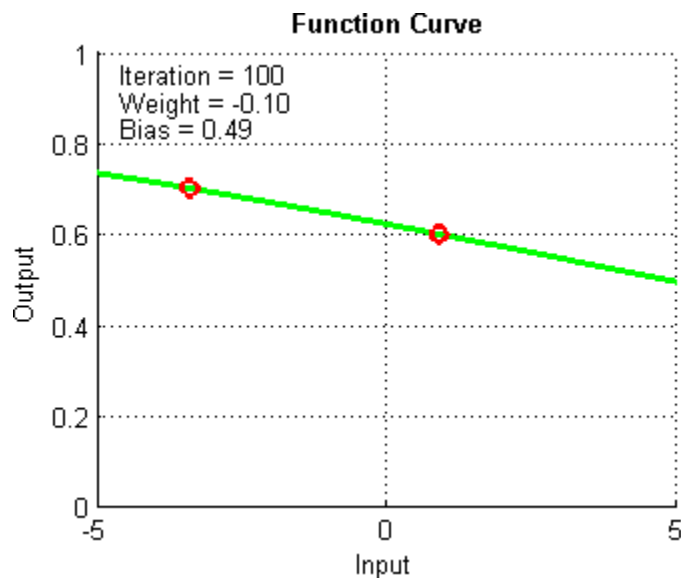
2. Retropropagação (*Backpropagation*)

- Algoritmo mais complexo utilizando múltiplas camadas - desenvolvido nos anos 80.
- **Aprende os pesos para uma rede multicamadas**, dada uma rede com um número fixo de unidades e interconexões.
- Emprega a **descida do gradiente** para minimizar o erro quadrático entre o valor da saída da rede e os valores esperados para estas saídas.

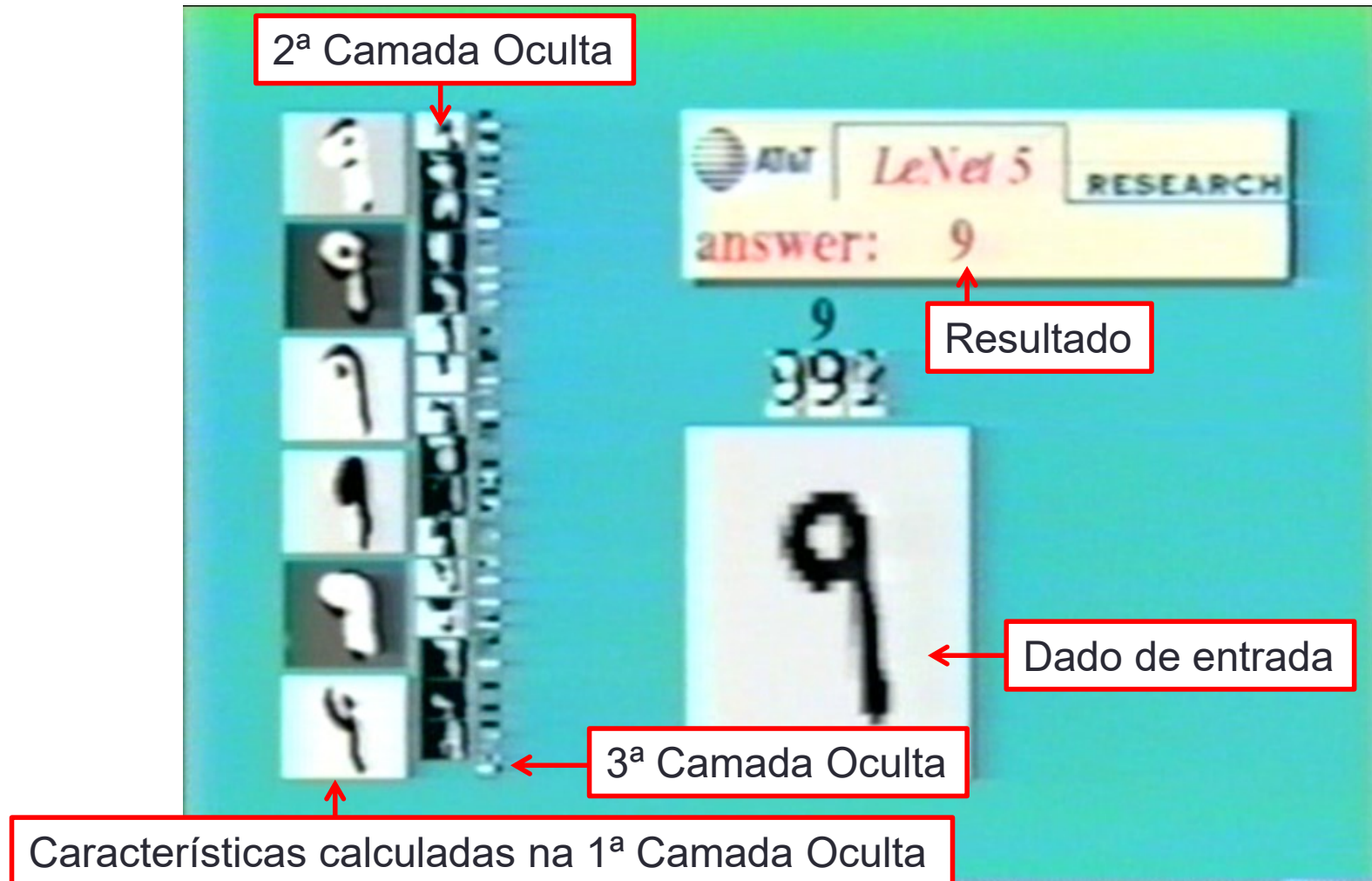


Descida do Gradiente

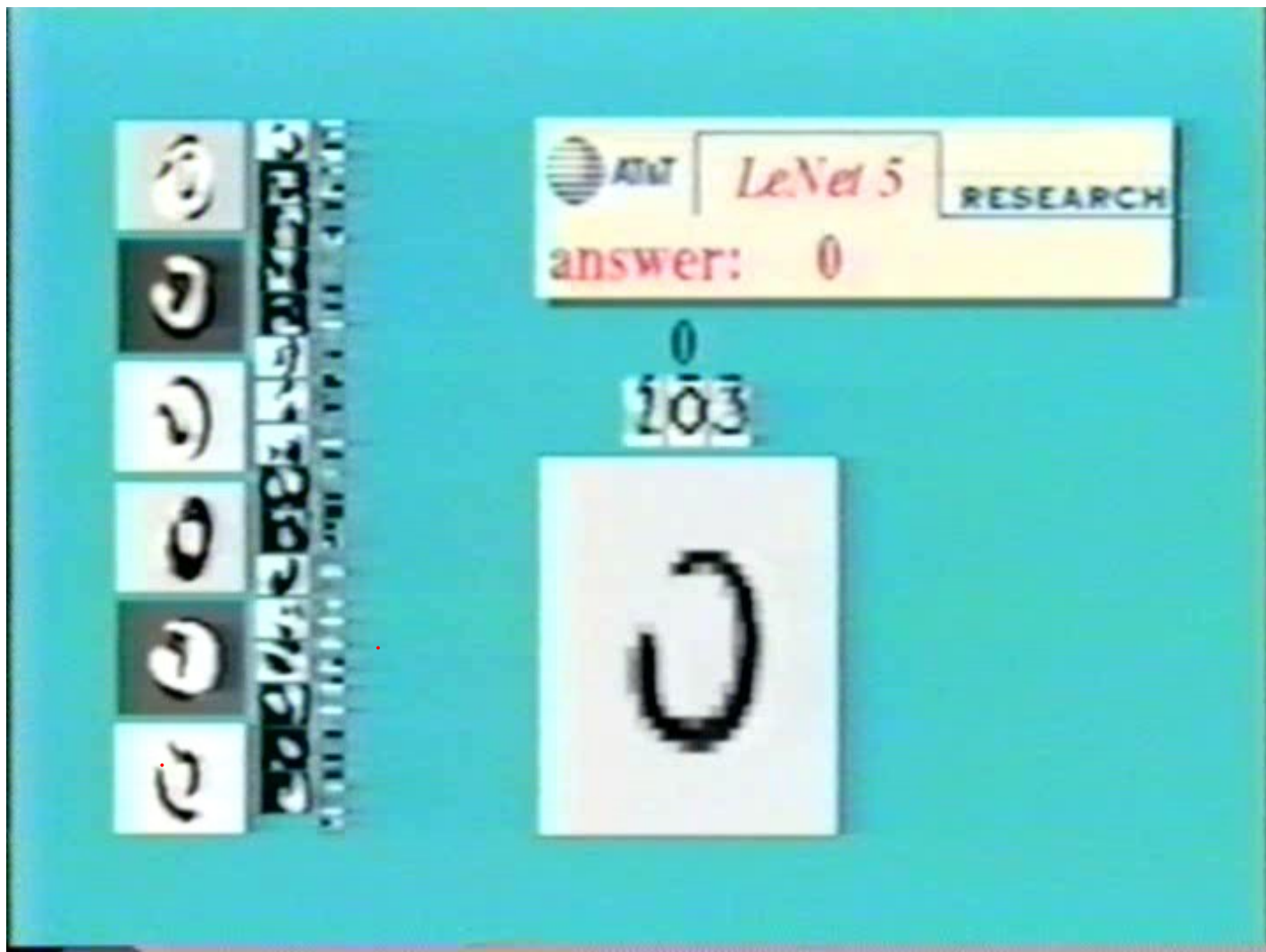
- Busca determinar um vetor de pesos que minimiza o erro.
- Começa com um vetor inicial de **pesos arbitrário** e o modifica repetidamente em pequenos passos.
- A cada passo, o vetor de pesos é alterado na direção que produz a maior queda ao longo da superfície de erro.



Aplicação: Classificação de Dígitos Manuscritos



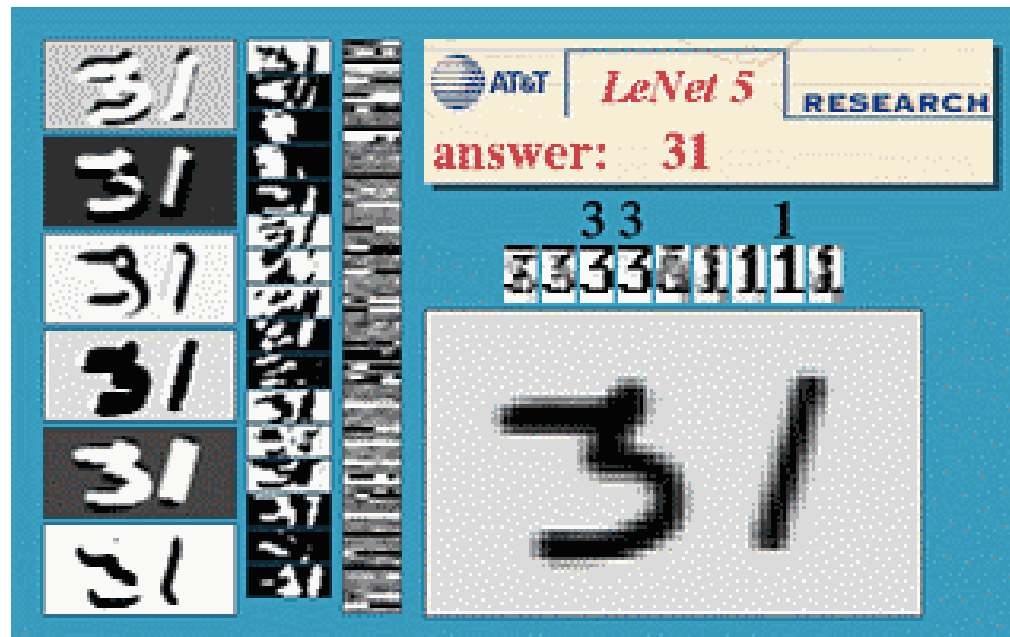
Aplicação: Classificação de Dígitos Manuscritos



Aplicação: Classificação de Dígitos Manuscritos

- Mais informações:

<http://yann.lecun.com/exdb/lenet/index.html>



Outras Aplicações

<https://quickdraw.withgoogle.com/>

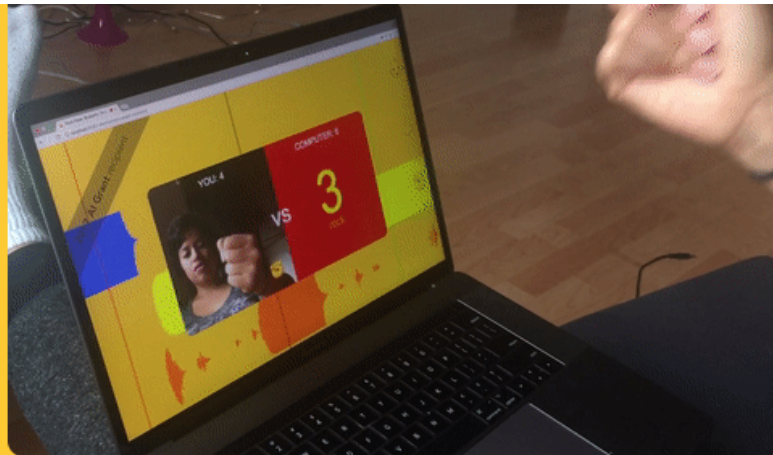
Sobre este jogo

Este jogo foi desenvolvido com aprendizado de máquina. Você desenha e a rede neural tenta adivinhar o que é. É claro que nem sempre ela consegue. Porém, quanto mais você jogar, mais ela aprenderá. Até agora, treinamos a rede para algumas centenas de conceitos e esperamos aumentar esse número com o tempo. Criamos esse jogo como um exemplo de como podemos utilizar o aprendizado de máquina de forma divertida. Assista ao vídeo abaixo para aprender sobre o funcionamento dele e



Outras Aplicações

<https://tenso.rs/demos/rock-paper-scissors/>



Play **Rock Paper Scissors** against your computer!

We're running **neural networks** running entirely **in your browser** to recognize your plays and **keep score**.

Play Rock Paper Scissors

Outras Aplicações



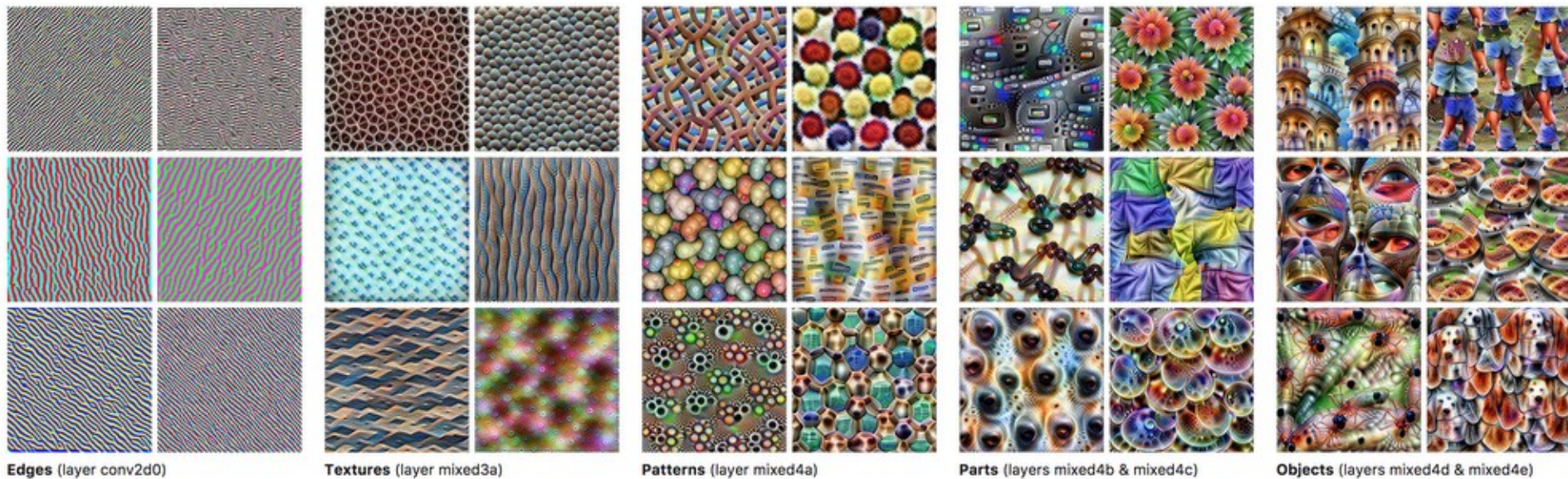
<https://semiconductor.withgoogle.com/>

Deep Learning (Aprendizagem Profunda)

- Rede Neural Artificial com muitas camadas ocultas;
- Capacidade de aprendizagem em grandes quantidades de dados de forma não-supervisionada;
- Dispensam grande parte do pré-processamento das características.
- Para saber mais:
 - <https://www.deeplearning.ai/>
 - Canal no youtube: [DeepLearning.TV](https://www.youtube.com/channel/UC8xp3833133333333333333)
 - Equipe DSA. <http://deeplearningbook.com.br/>
 - Sumit Saha. A Comprehensive Guide to Convolutional Neural Networks <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

O que um neurônio enxerga?

- <https://www.blog.google/technology/ai/understanding-inner-workings-neural-networks/>



1ª camada

2ª camada

4ª camada















... mais avançadas

Camada mais profunda

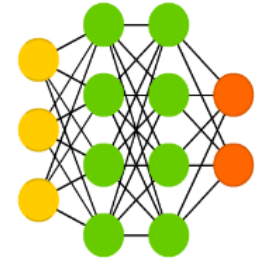
A rede neural detecta primeiro bordas, depois texturas, padrões, partes e objetos.

A mostly complete chart of Neural Networks

©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org

-  Input Cell
-  Backfed Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probabilistic Hidden Cell
-  Spiking Hidden Cell
-  Capsule Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Gated Memory Cell
-  Kernel
-  Convolution or Pool

Deep Feed Forward (DFF)



Perceptron (P)



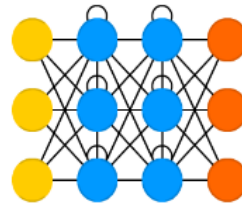
Feed Forward (FF)



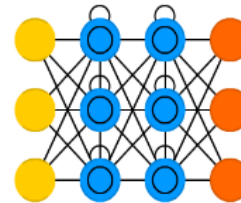
Radial Basis Network (RBF)



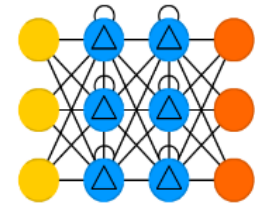
Recurrent Neural Network (RNN)



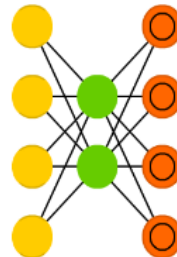
Long / Short Term Memory (LSTM)



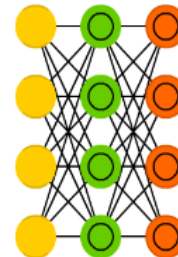
Gated Recurrent Unit (GRU)



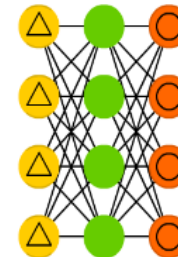
Auto Encoder (AE)



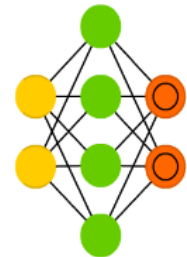
Variational AE (VAE)



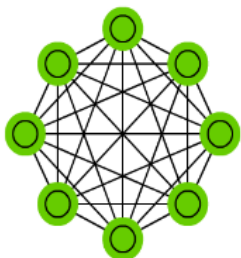
Denoising AE (DAE)



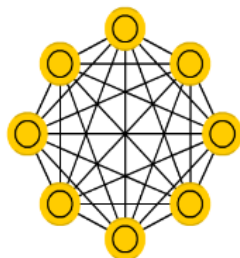
Sparse AE (SAE)



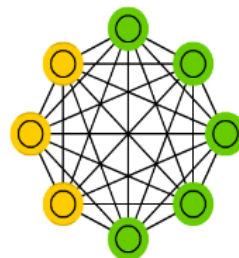
Markov Chain (MC)



Hopfield Network (HN)



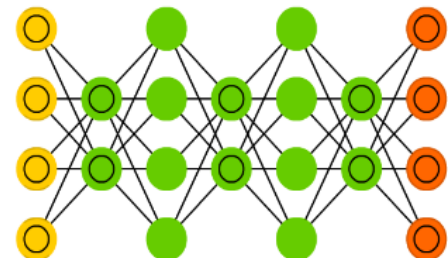
Boltzmann Machine (BM)



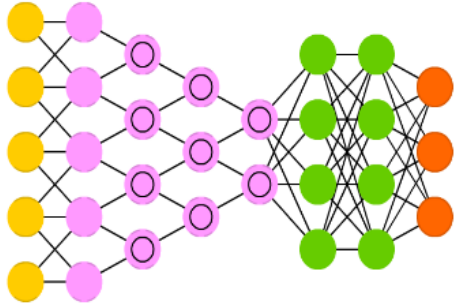
Restricted BM (RBM)



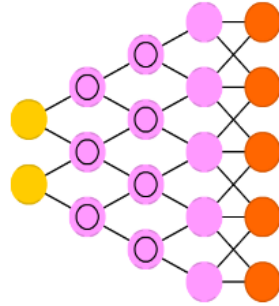
Deep Belief Network (DBN)



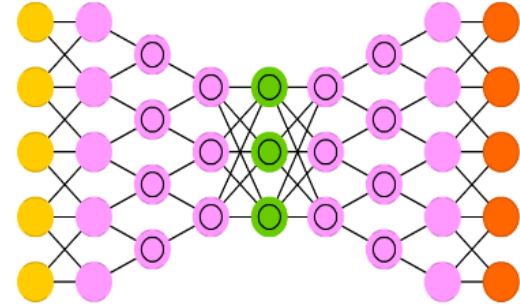
Deep Convolutional Network (DCN)



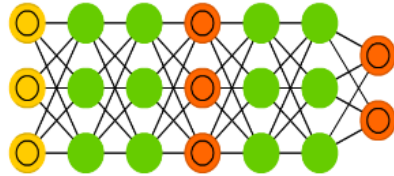
Deconvolutional Network (DN)



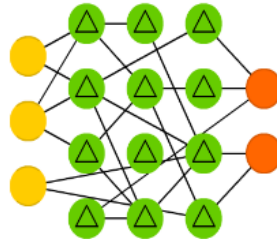
Deep Convolutional Inverse Graphics Network (DCIGN)



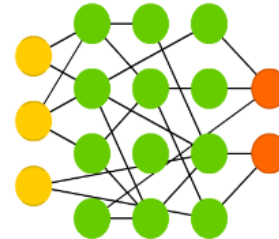
Generative Adversarial Network (GAN)



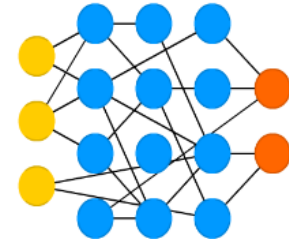
Liquid State Machine (LSM)



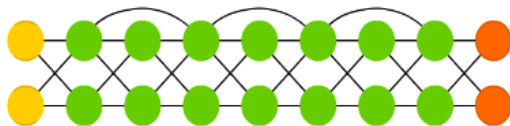
Extreme Learning Machine (ELM)



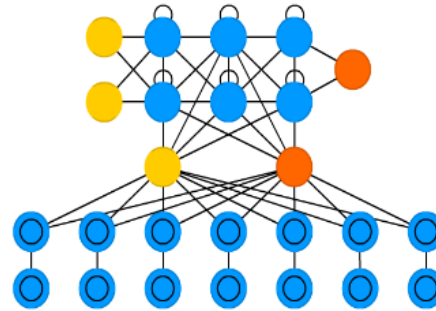
Echo State Network (ESN)



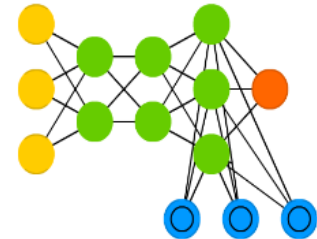
Deep Residual Network (DRN)



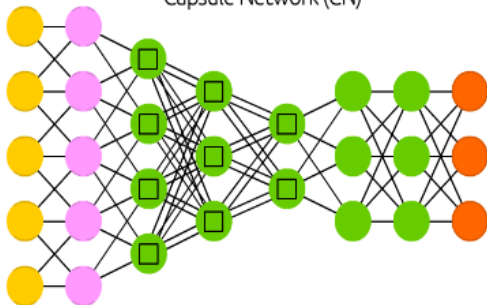
Differentiable Neural Computer (DNC)



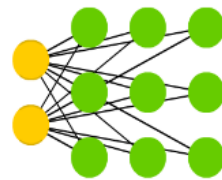
Neural Turing Machine (NTM)



Capsule Network (CN)



Kohonen Network (KN)



Attention Network (AN)

