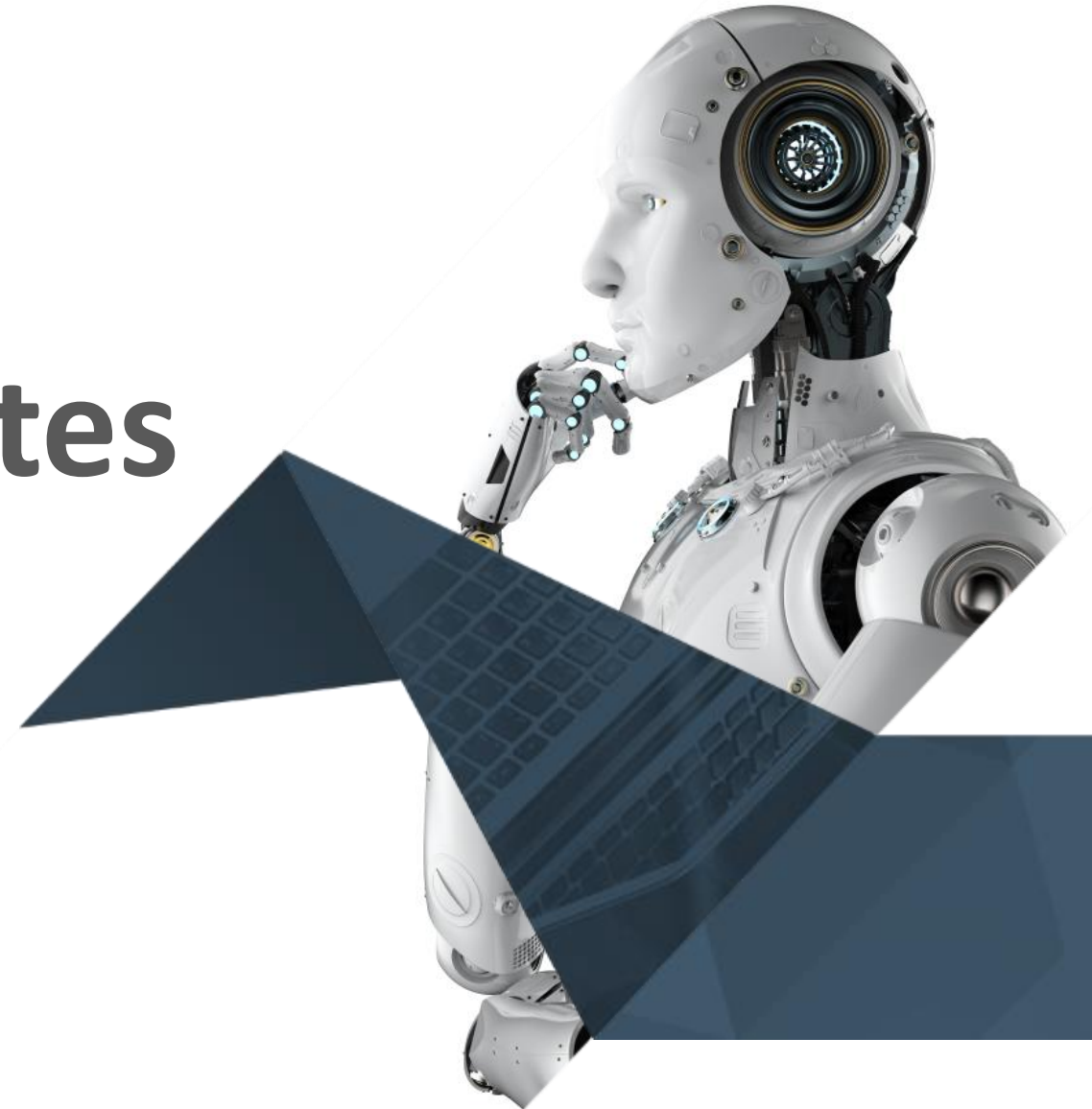
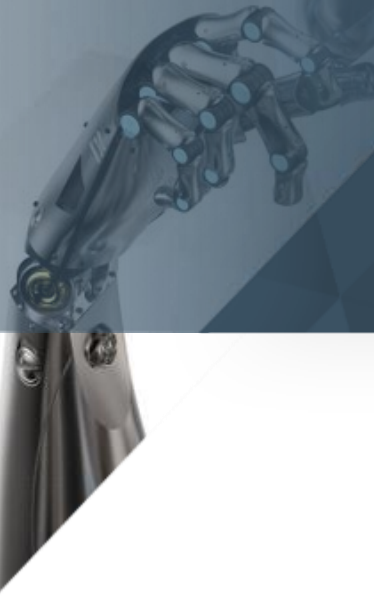


Introdução aos Sistemas Inteligentes

Prof. Sandro Jerônimo



Neurônio Artificial



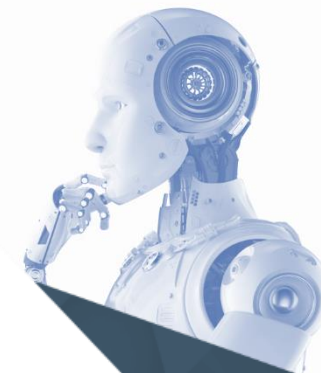
Fontes das Imagens: www.shutterstock.com ou próprio autor



PUC Minas Virtual

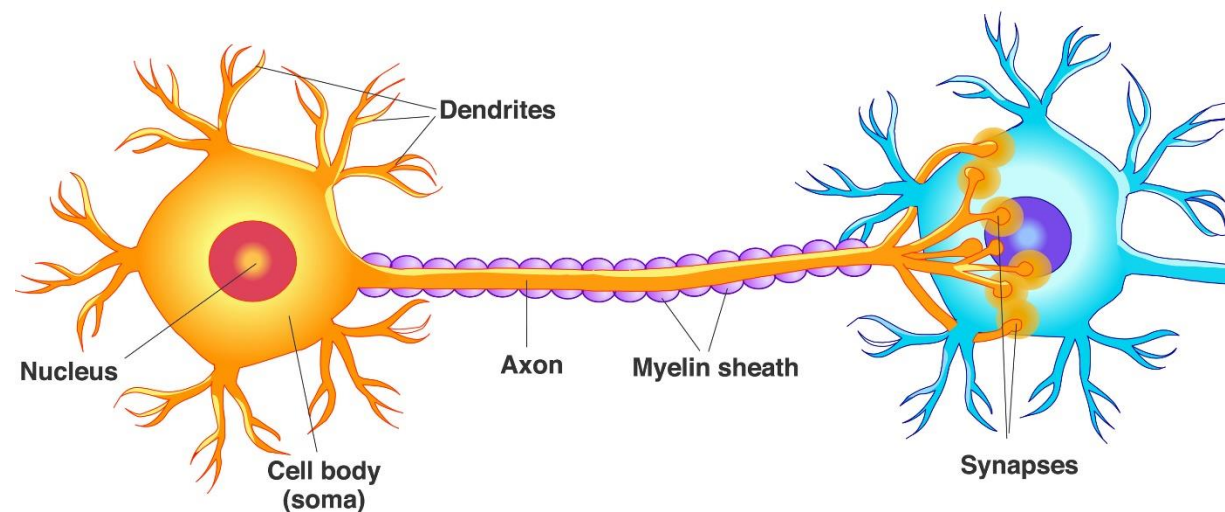


Neurônio biológico



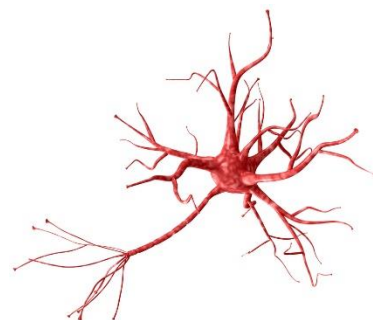
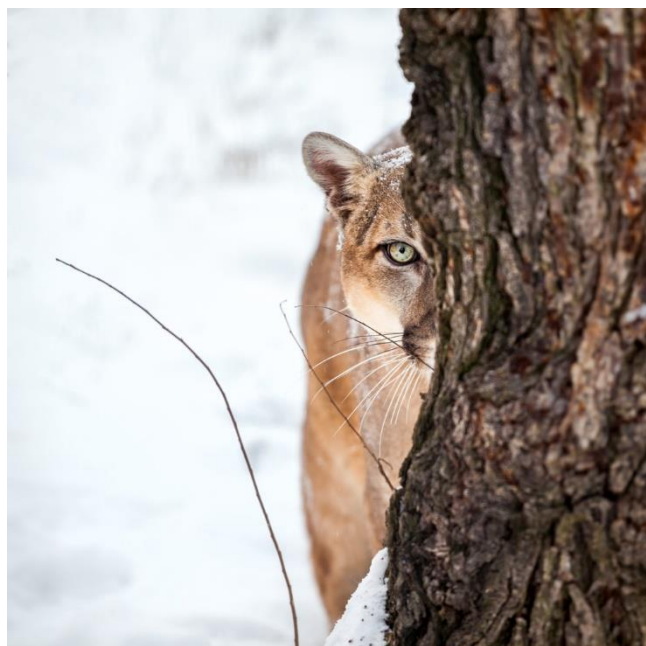
Modelo biológico

- Corpo humano possui aproximadamente 100 bilhões de neurônios. Todos interconectados através das sinapses/dendritos.

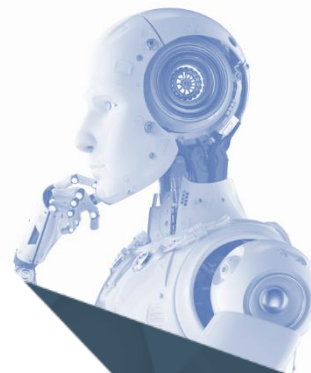
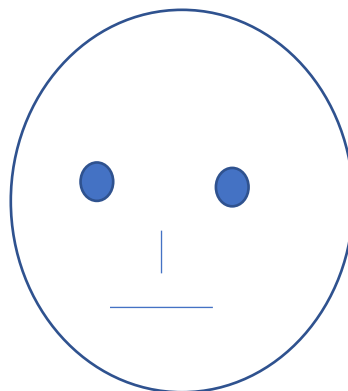


Ideia geral

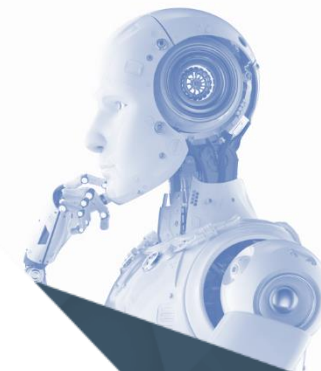
Um neurônio se adapta de forma a dar uma resposta ao que se “espera”



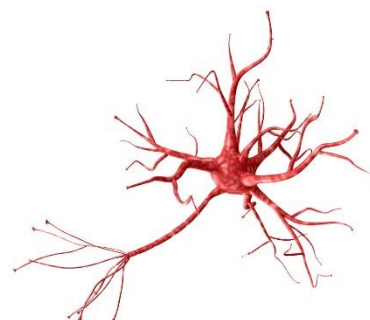
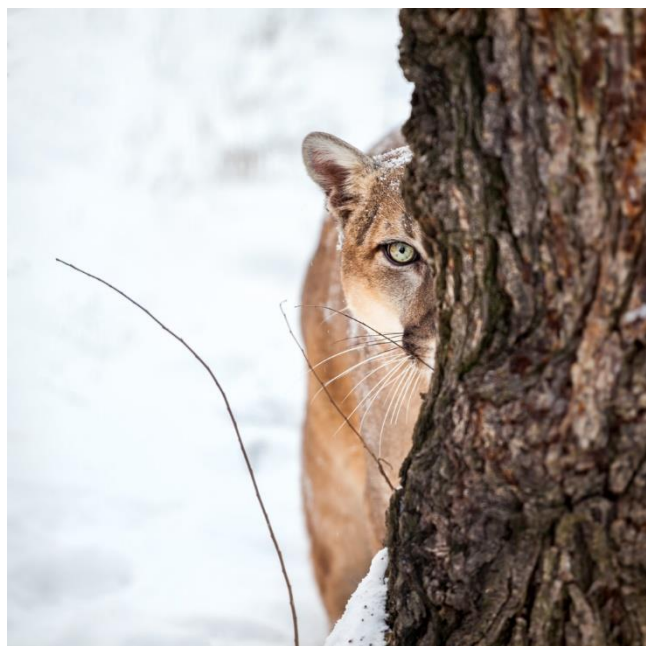
Neurônios



Aprendizado



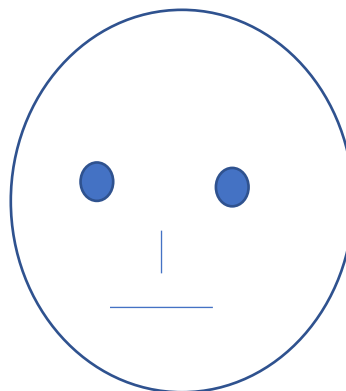
E se o neurônio não estiver treinado?



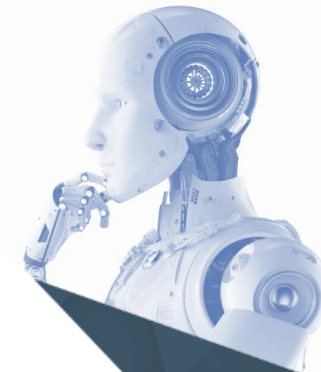
Neurônios



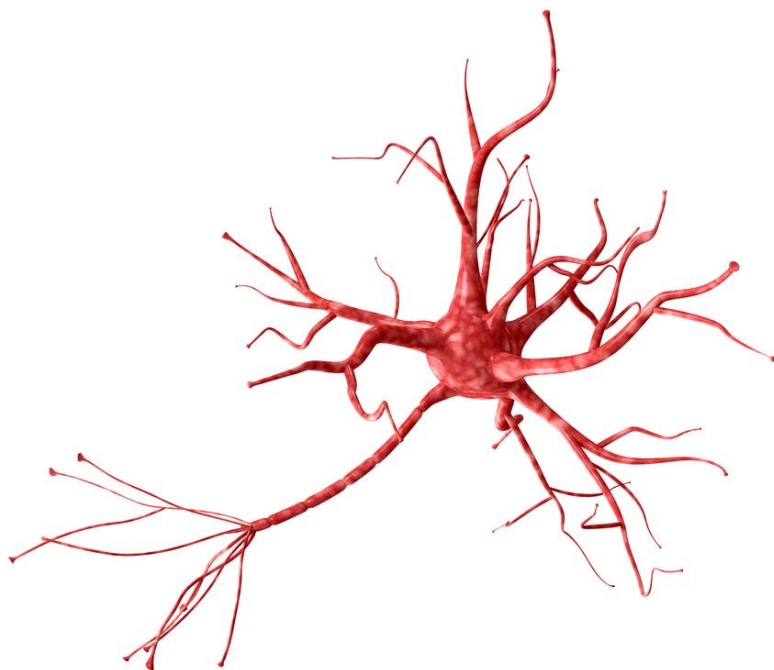
?



Aprendizado



- Se o neurônio não estiver treinado, vamos ter que ensiná-lo. O aprendizado consiste em “reconfigurar o neurônio”.



- As sinapses transmitem estímulos através de diferentes concentrações de **Na⁺ (Sódio)** e **K⁺ (Potássio)**



Motivação com o Neurônio Artificial

O melhor de dois mundos!

- Os neurônios biológicos, trabalhando em rede, proporcionando uma fabulosa capacidade de processamento e armazenamento de informação
- Um computador faz uma conta simples, cerca de um milhão de vezes mais rápido que um ser humano

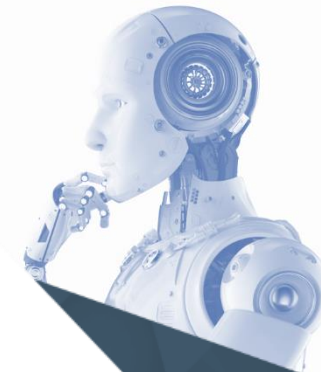


Redes Neurais Artificiais

Definição

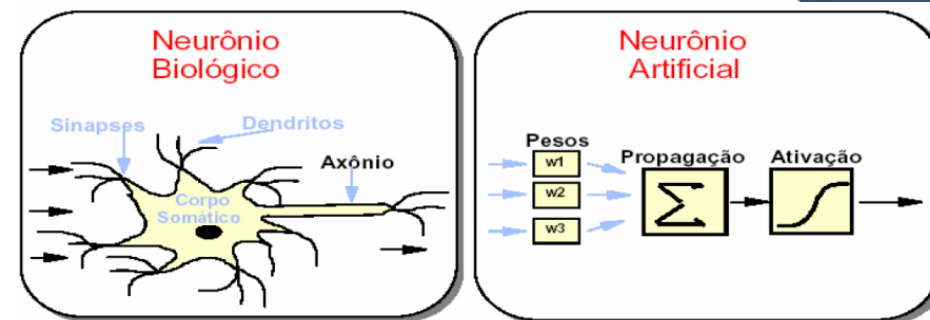
- Redes Neurais Artificiais são sistemas inspirados nos neurônios biológicos e na estrutura massivamente paralela do cérebro, com capacidade de adquirir, armazenar e utilizar conhecimento experimental

Neurônio: biológico x artificial



Neurônio Biológico

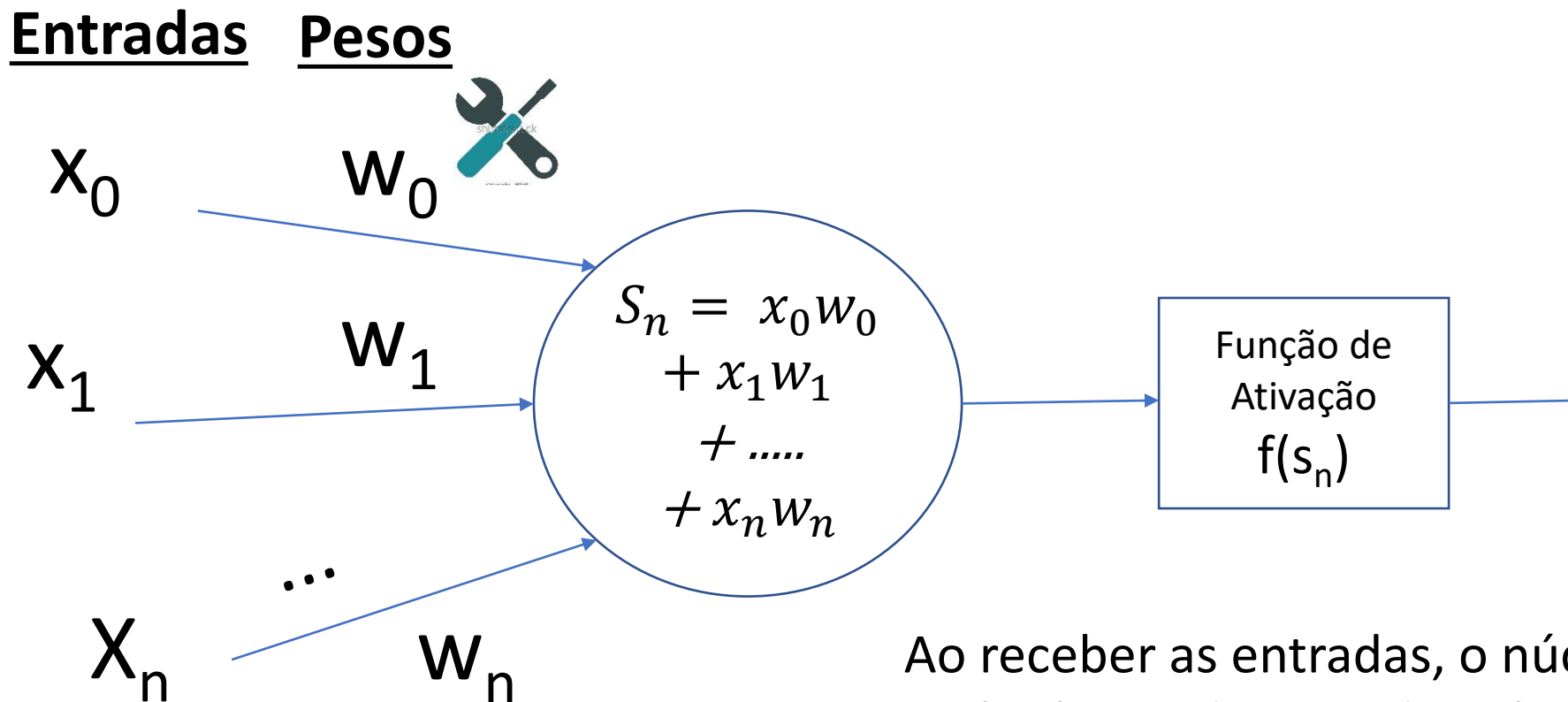
- Dendrito e sinapses: entradas de sinais
- Corpo somático: processa os sinais de entrada
- Axônio: saída do neurônio (ativo ou não)



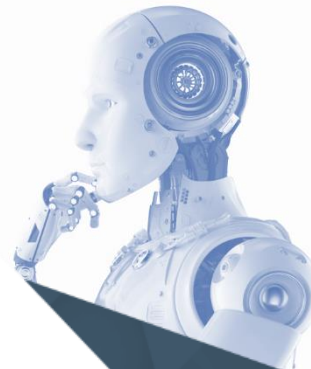
Neurônio Artificial

- Entradas e Pesos: entradas de dados com pesos para cada entrada
- Propagação: somatório da multiplicação dos pesos pelas entradas
- Saída: o somatório passa por uma função de ativação

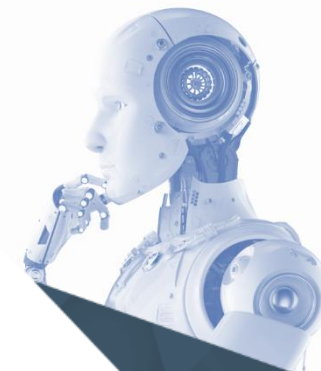
Neurônio Artificial



Ao receber as entradas, o núcleo do neurônio multiplica cada entrada pelo peso correspondente e realiza um somatório. Depois o resultado passa por uma função de ativação, que retorna valores (normalmente 0 ou 1).



Função de Ativação



Propósito

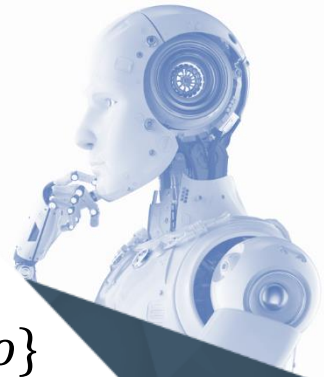
- Recebe o valor da saída do neurônio e coloca em um formato desejado

$$f(\text{saida_neuronio}) = \{\text{coloca o valor da saida_neuronio em um formato desejado}\}$$

Exemplo

- Se o neurônio indica que uma ação deve ou não ser tomada, a função de ativação deveria dar respostas no formato: 0 (não faz) ou 1 (faz).
- Se o neurônio calcula a probabilidade de uma pessoa ser boa pagadora, a função deveria dar respostas entre 0% e 100%

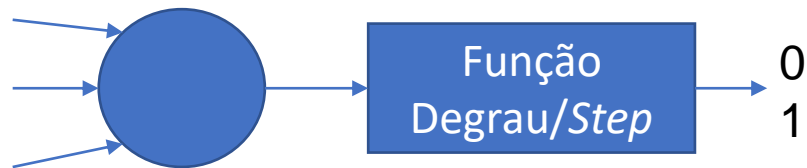
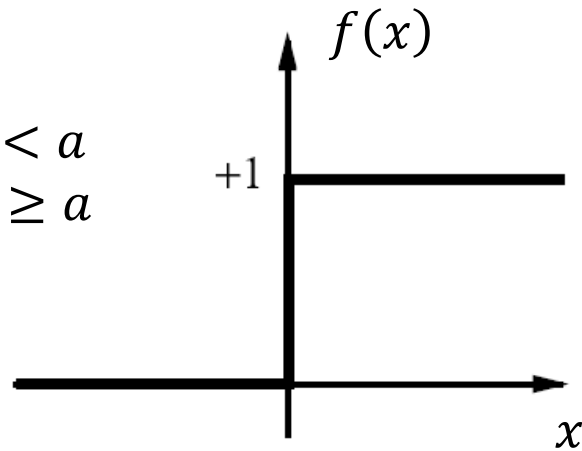
Função de Ativação



$f(x) = \{\text{coloca o valor } x \text{ da saída de neurônio em um formato desejado}\}$

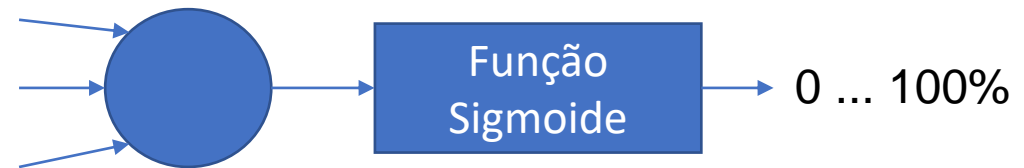
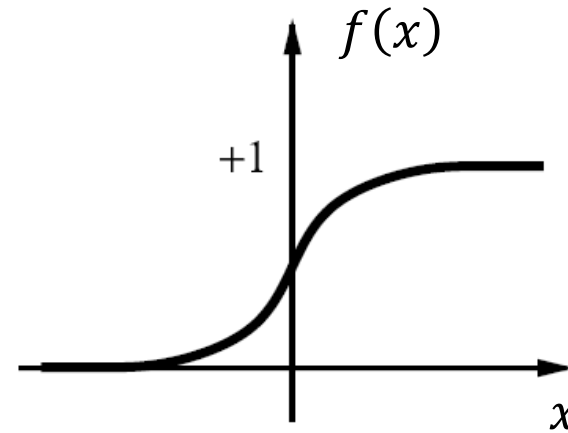
Função Degrau/Step

$$f(x) = \begin{cases} 0, & \text{se } x < a \\ 1, & \text{se } x \geq a \end{cases}$$



Função Sigmoide

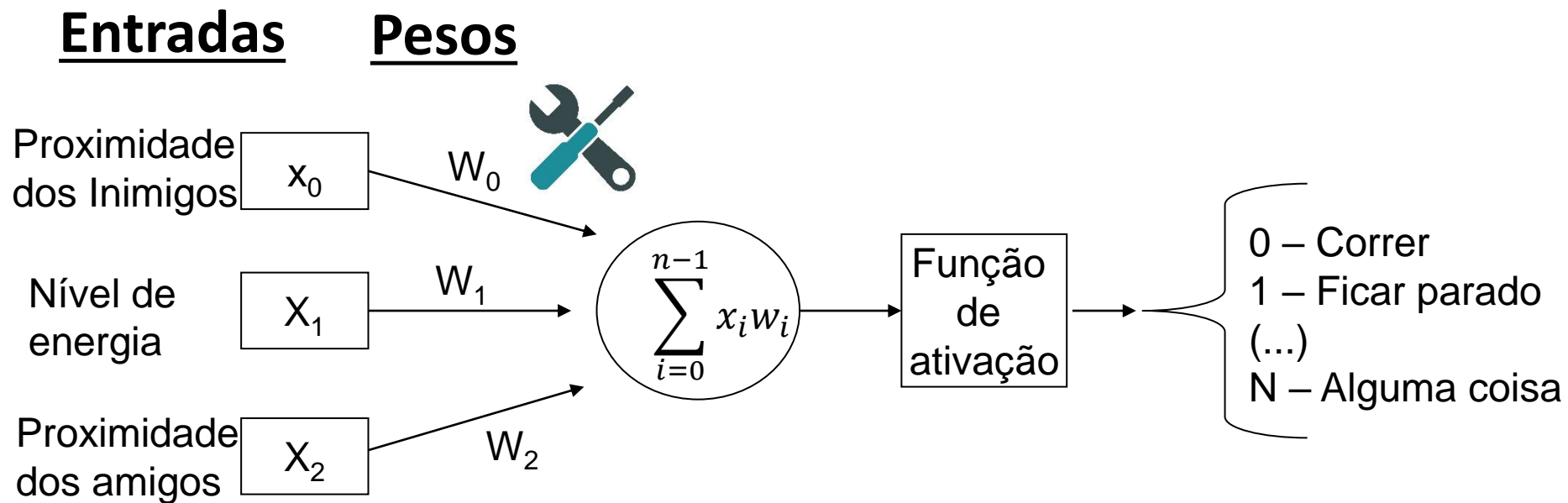
$$f(x) = \frac{1}{1 + e^{-x}}$$



Neurônio Artificial de um NPC (Jogos)



- Como um NPC poderia aprender a escolher uma melhor decisão usando sua energia e a posição de outros jogadores?





PUC Minas
Virtual



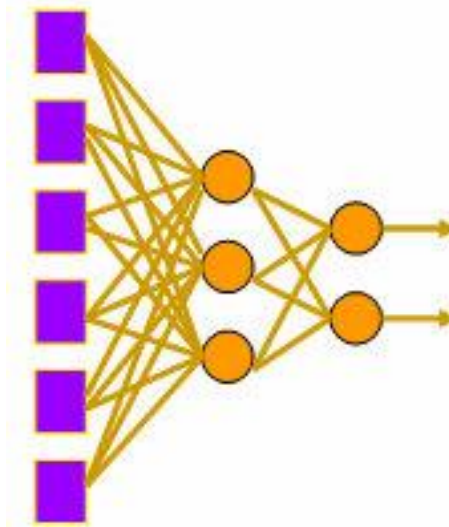
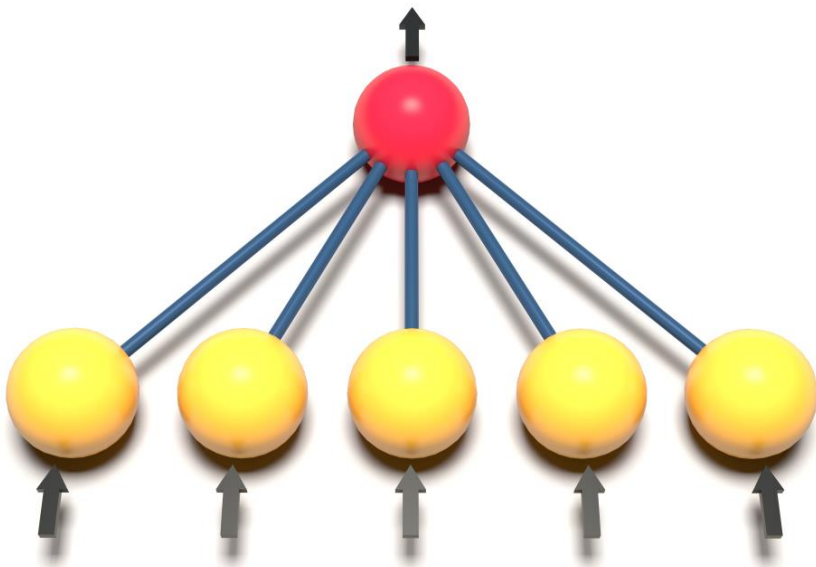
Rede Neural Perceptron



PUC Minas Virtual

Rede Neural Perceptron



- Uma rede neural *perceptron* tradicional possui uma ou mais camadas com vários neurônios artificiais





Rede Neural Perceptron

A utilização da rede neural *perceptron* se divide em:

- 
- **Treinamento da Rede:** após definir um conjunto de dado e parâmetros de treinamento, então o treinamento começa
 - **Uso efetivo da rede** (treinada previamente) para obter respostas
- 



Perceptron - Treinamento

- **Taxa de Aprendizado:** velocidade na qual o sistema está caminhando para encontrar a solução (o tamanho do passo).
 - Taxa muito baixa vai necessitar de muitas épocas de treinamento.
 - Taxa alta pode tornar o algoritmo oscilante, demora para convergir.
- **Erro Global:** média dos erros para todo o conjunto de dados de teste.
- **Época:** cada período de treinamento, no qual todo o conjunto de dados já foi usado.



Perceptron - Treinamento

Estratégia simples de treinamento de um neurônio

- Dado um conjunto de dados (entradas e saída), seguir o seguinte passos:
- (1) Iniciar os valores dos pesos aleatoriamente
- (2) Para cada conjunto de dados, calcular o erro na saída no neurônio
(erro = saída desejada - saída do neurônio)
- (3) Baseado no erro, atualizar o valor de cada peso segundo a seguinte equação:
$$Wx = Wx + (\text{Erro} \times \text{TaxaAprendizado} \times \text{Entrada})$$
- (4) Repetir a partir do passo (2), até que se complete um número de épocas ou o erro global esteja abaixo de um limiar.



Perceptron - Treinamento

Vamos ensinar um neurônio a somar 2 números!

- Exemplos a serem passado para o neurônio (base de treinamento)

X_0	X_1	Saída desejada
1	1	2
2	2	4
3	3	6
4	4	8
5	5	10

- Vamos ignorar a função de ativação neste exemplo
- Além disso, vamos utilizar uma taxa de aprendizado igual a 0,05 e apenas uma época (5 iterações)

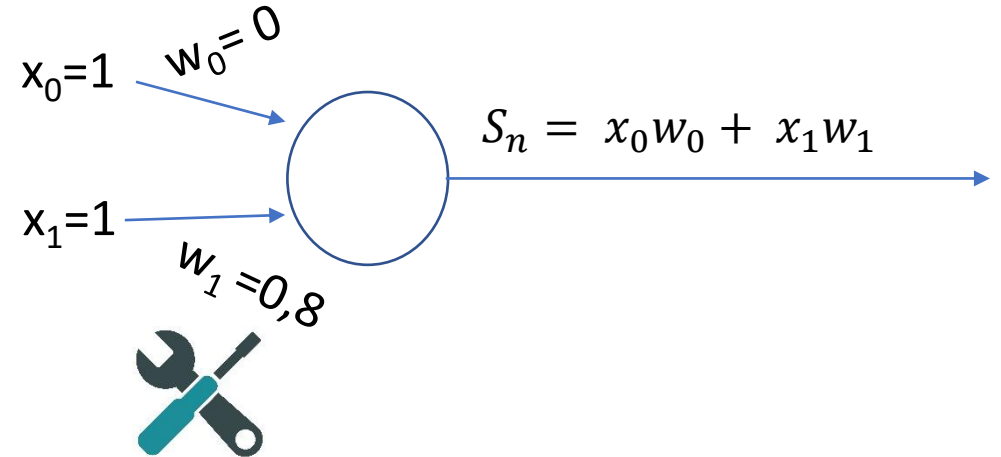
Ensinando um neurônio a somar

- Iniciando o algoritmo...

Pesos aleatórios

- $W_0 = 0,000000$

- $W_1 = 0,800000$



- Treinando com a linha 1

$$S_n = (x_0 \times w_0) + (x_1 \times w_1)$$
$$= (1 \times 0) + (1 \times 0,8) = 0,8$$

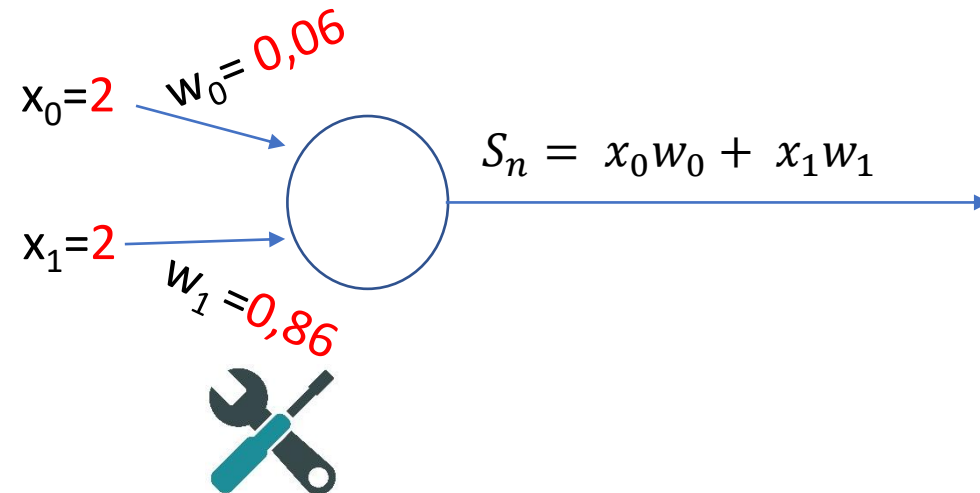
$$\text{Erro} = S_d - S_n = 2 - 0,8 = 1,2$$

$$W_0 = W_0 + (\text{TaxaApr} \times \text{Erro} \times \text{Entrada})$$
$$= 0 + (0,05 \times 1,2 \times 1) = 0,06$$

$$W_1 = 0,86$$

X_0	X_1	S_d
1	1	2
2	2	4
3	3	6
4	4	8
5	5	10

Ensinando um neurônio a somar




- Treinando com a linha 2

$$S_n = (x_0 \times w_0) + (x_1 \times w_1) \\ = (2 \times 0,06) + (2 \times 0,86) = 1,84$$

$$\text{Erro} = S_d - S_n = 4 - 1,84 = 2,16$$

$$W_0 = W_0 + (\text{TaxaApr} \times \text{Erro} \times \text{Entrada}) \\ = 0,06 + (0,05 \times 2,16 \times 2) = 2,276$$

$$W_1 = 1,076$$



X_0	X_1	S_d
1	1	2
2	2	4
3	3	6
4	4	8
5	5	10

Ensinando um neurônio a somar

- Treinando com a linha 3

Erro = 1,944

$W_0 = 0,5676$

$W_1 = 1,3676$

- Treinando com a linha 4


Erro = 0,2592

$W_0 = 0,61944$

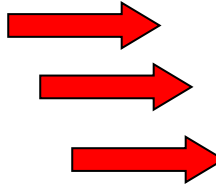
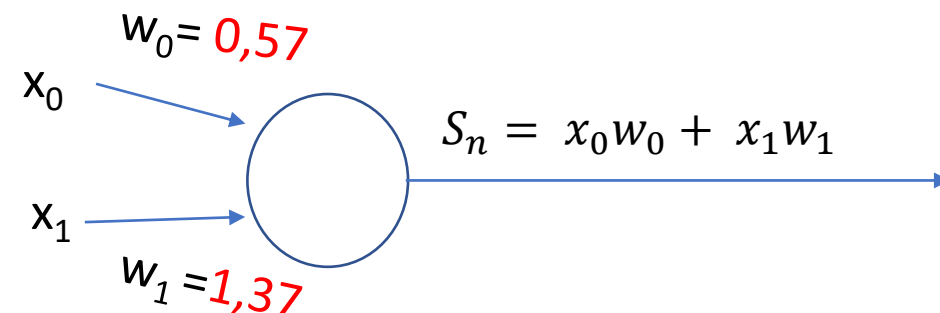
$W_1 = 1,41944$

- Treinando com a linha 5

Erro = -0,1944

 $W_0 = 0,57084$

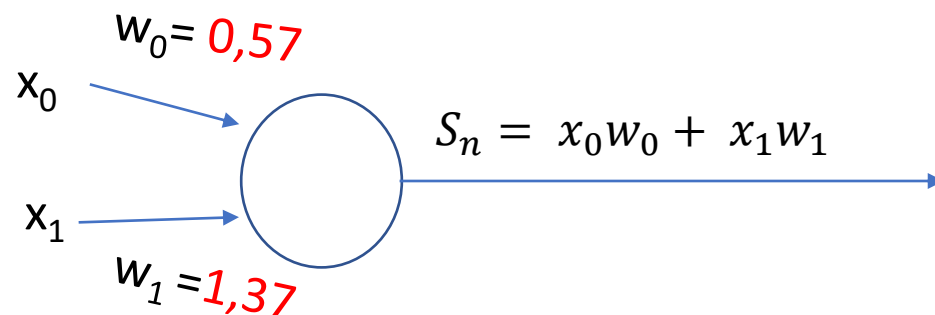
$W_1 = 1,37084$



X_0	X_1	Sd
1	1	2
2	2	4
3	3	6
4	4	8
5	5	10

Afinal...

O neurônio aprendeu a somar?



Um teste simples - Quanto é 1+1?

$$\begin{aligned} S_n &= (x_0 \times w_0) + (x_1 \times w_1) \\ &= (1 \times 0,57) + (1 \times 0,37) = 1,94 \end{aligned}$$

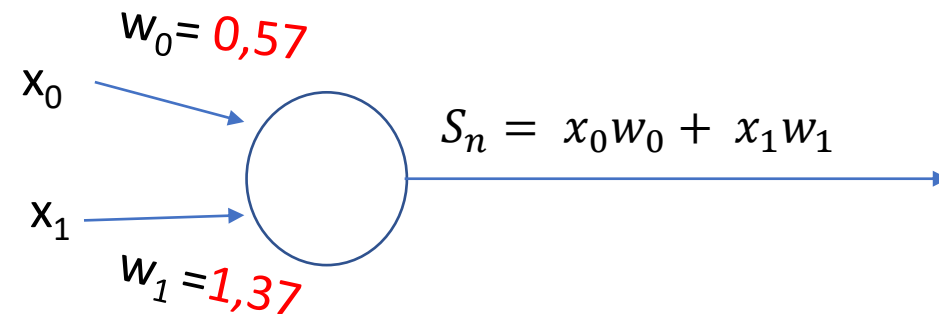
Precisão: $(1,94 / 2) = 0,97 \lll 97\%$

Afinal...

O neurônio aprendeu a somar?

Testando mais valores

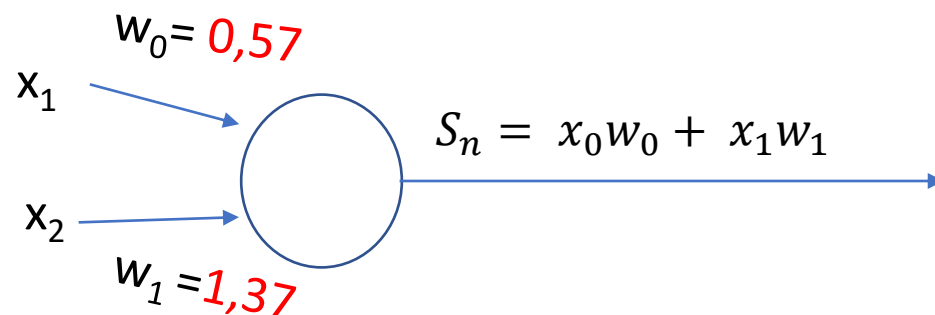
- Na tabela notamos que a rede aprendeu a somar dois números iguais com uma margem de erro de **0,02**
- O erro poderia ser menor se tivéssemos a treinado por mais épocas.
- Ela consegue generalizar, somando números não treinados anteriormente (ex:10+10)



x_0	x_1	Saída
1	1	1,94
2	2	3,88
3	3	5,8
4	4	7,76
5	5	9,7
6	6	11,65
7	7	13,59
10	10	19,41

Afinal...

O neurônio aprendeu a somar?



Um teste fora dos padrões: quanto é $1000 + 1$?

$$\begin{aligned} S_n &= (x_0 \times w_0) + (x_1 \times w_1) \\ &= (1.000 \times 0,57) + (1 \times 0,37) = 570,37 \end{aligned}$$

Precisão: $(570,37 / 1001) = 0,57 \lll 57\%$

Por que a precisão caiu? Resposta: base de treino é incompleta



PUC Minas
Virtual



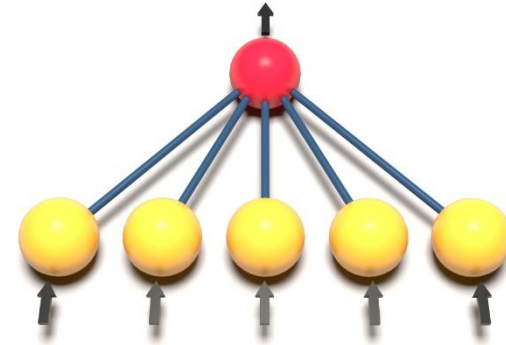
Rede Neural Multicamadas Perceptron



PUC Minas Virtual

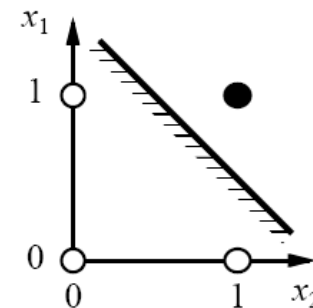
Perceptron de camada única

Limitações



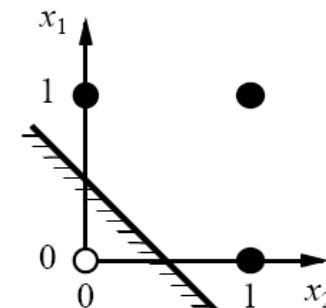
- Uma rede neural *perceptron* de uma única camada tem dificuldade de representar funções não-lineares ou modelos que separem linearmente os dados.

X1	X2	E
1	1	1
1	0	0
0	1	0
0	0	0



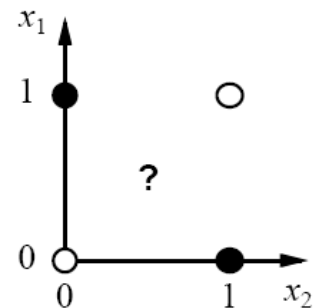
(a) x_1 and x_2

X1	X2	OU
1	1	1
1	0	1
0	1	1
0	0	0



(b) x_1 or x_2

X1	X2	XOR
1	1	0
1	0	1
0	1	1
0	0	0



(c) x_1 xor x_2



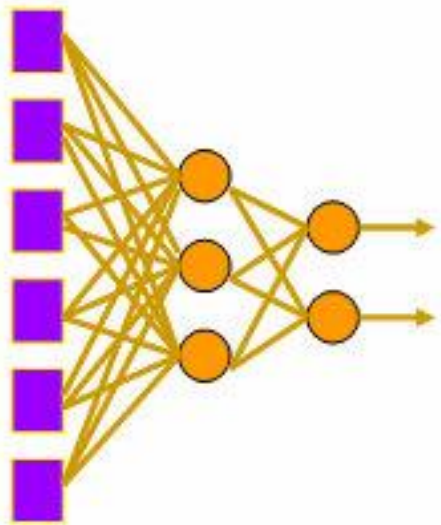
Histórico das Redes Neurais

- McCulloch & Pitts (1943): modelo computacional para o neurônio artificial. Não possuía capacidade de aprendizado
- Hebb (1949): modelo de aprendizado (Hebbian Learning Rule)
- Rosenblatt (1957): Perceptron, com grande sucesso em certas aplicações de problemas em outras aparentemente similares
- Minsky & Papert (Perceptrons 1969): prova matemática de que as redes Perceptron são incapazes de solucionar problemas simples tipo OU-EXCLUSIVO
- Rumelhart (início da década de 80): novos modelos que superaram os problemas dos Perceptrons. Ex: redes perceptron multicamadas etc.
- Schmidhuber & Hochreiter (1997): modelo LSTM é proposto possibilitando avanços teóricos para *deep learning*
- Anos 2000.... : avança o volume de bases de dados transacionais (*Big Data*), o desenvolvimentos de hardware, metodologias, *frameworks* e plataformas de serviços de computação em nuvem (Microsoft, Google, Amazon, IBM... *AI as a Services, MLOps*)

Perceptron Multicamadas

Motivação

- Superar as limitações das redes de camada única
- Realiza aproximações para funções contínuas
- Todo conjunto de dados pode ser separado por uma rede de 3 camadas.
- Quase todo conjunto de dados pode ser separado por uma rede de 2 camadas.

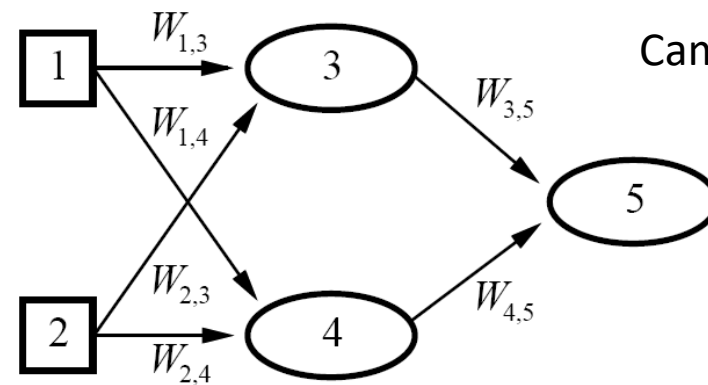


Exemplo com 3 camadas

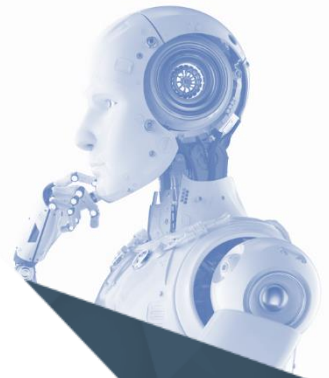
Camada de Entrada

Camada oculta

Camada de Saída



Perceptron Multicamadas: estruturas



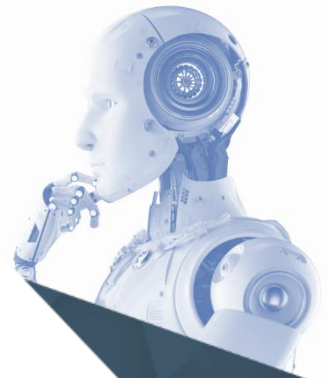
Redes feed-forward

- Single-layer ou multi-layer
- Implementam funções – não possuem estado interno

Redes Recorrentes

- Possuem ciclos direcionados com atrasos – possuem estado interno.
- Redes de *Hopfield*: implementam memória associativa.
- Máquinas de Boltzmann: usa funções estocásticas de ativação.

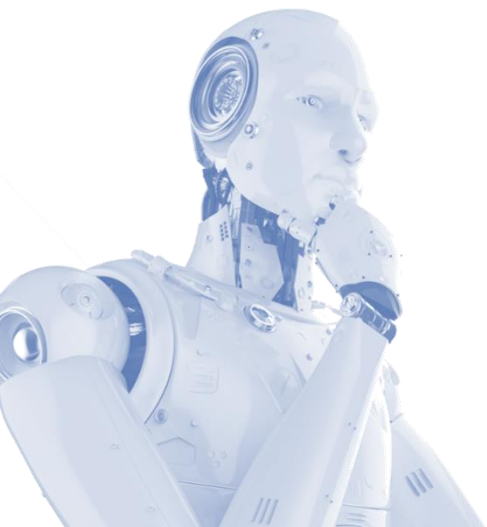
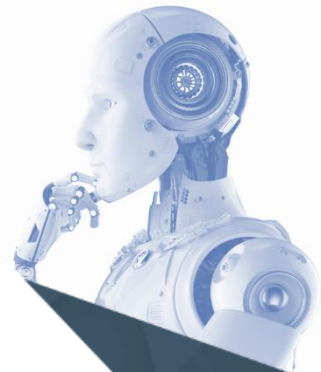
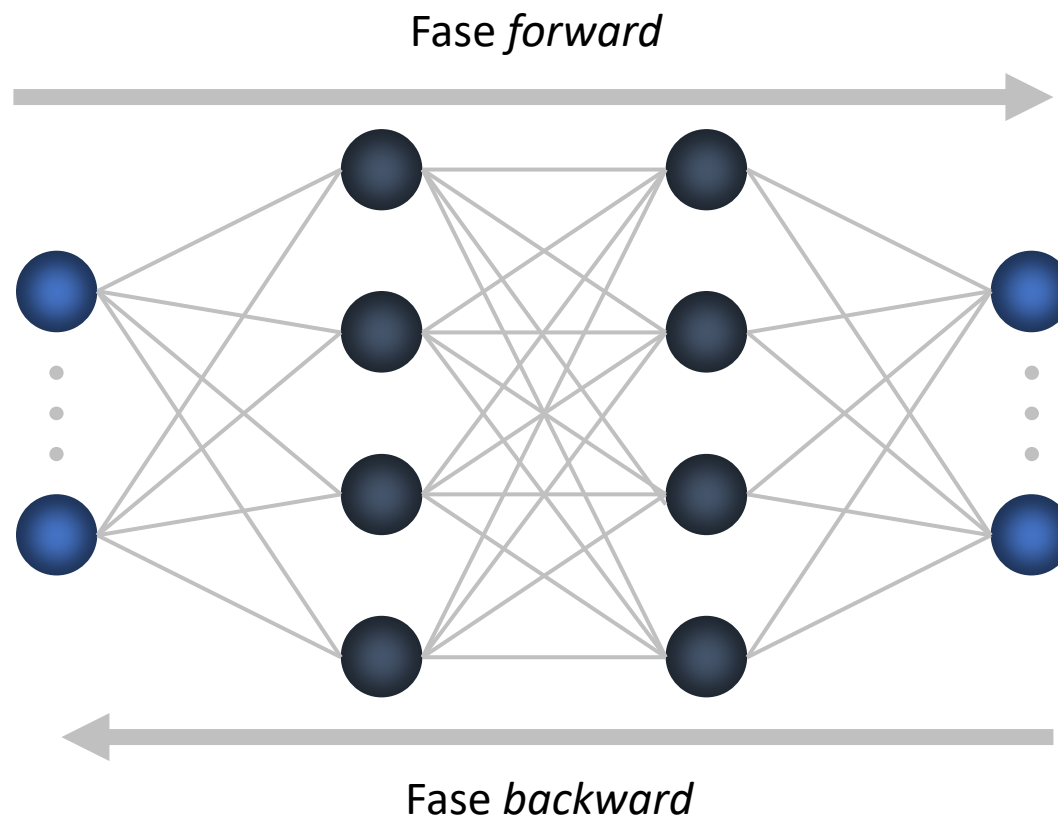
Perceptron Multicamadas



Algoritmo *Back propagation* – Funcionamento Geral

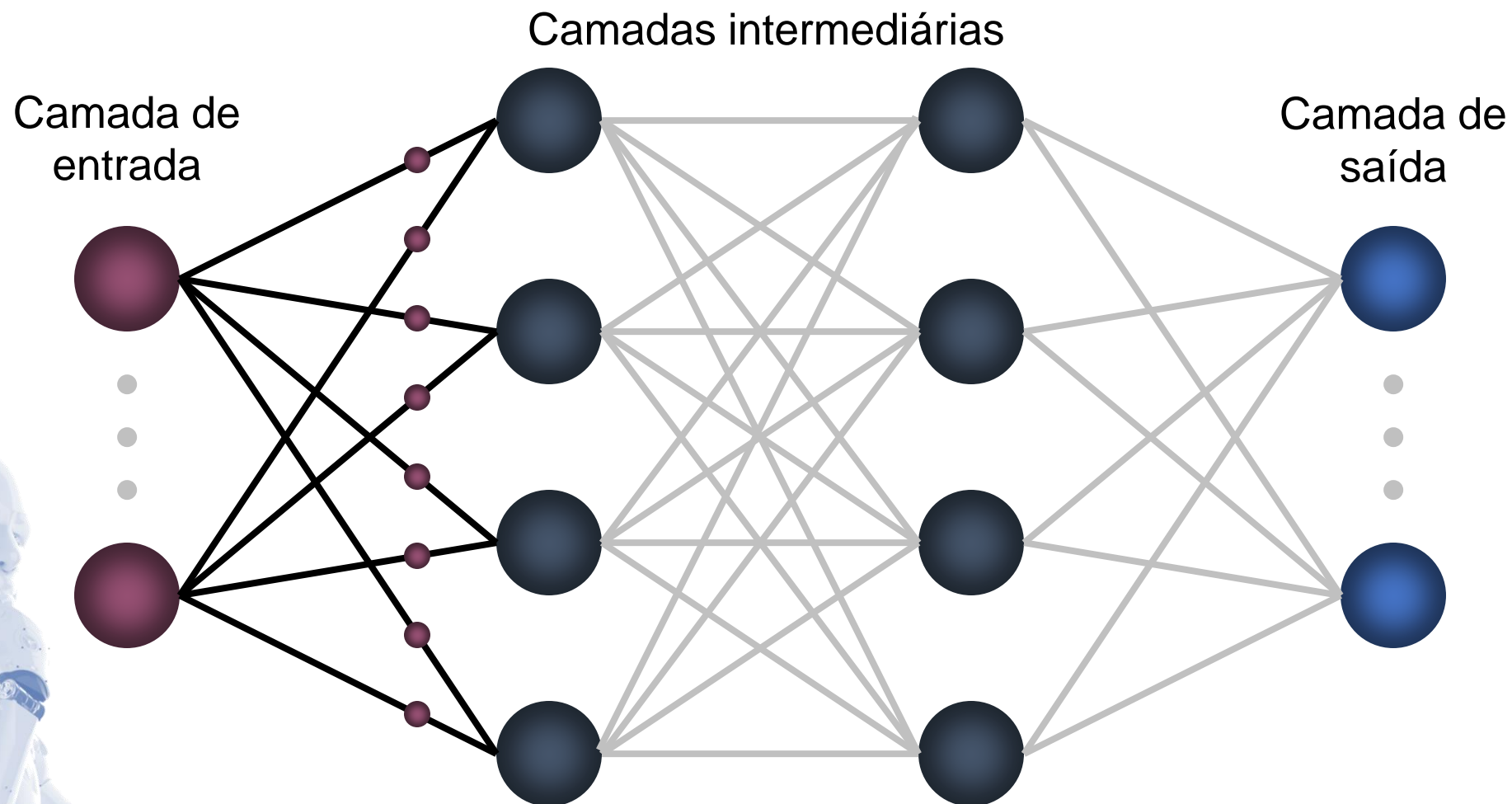
- Inicia-se os pesos com um valor aleatório
- Usando o *training set*, alimenta a rede com os valores de entrada e observa a saída
- Calcula o erro
- Ajusta os pesos para diminuir o erro (*back propagating* na rede), e repete o processo um número fixo de vezes ou até que o erro seja pequeno o suficiente.

Algoritmo *back propagation*



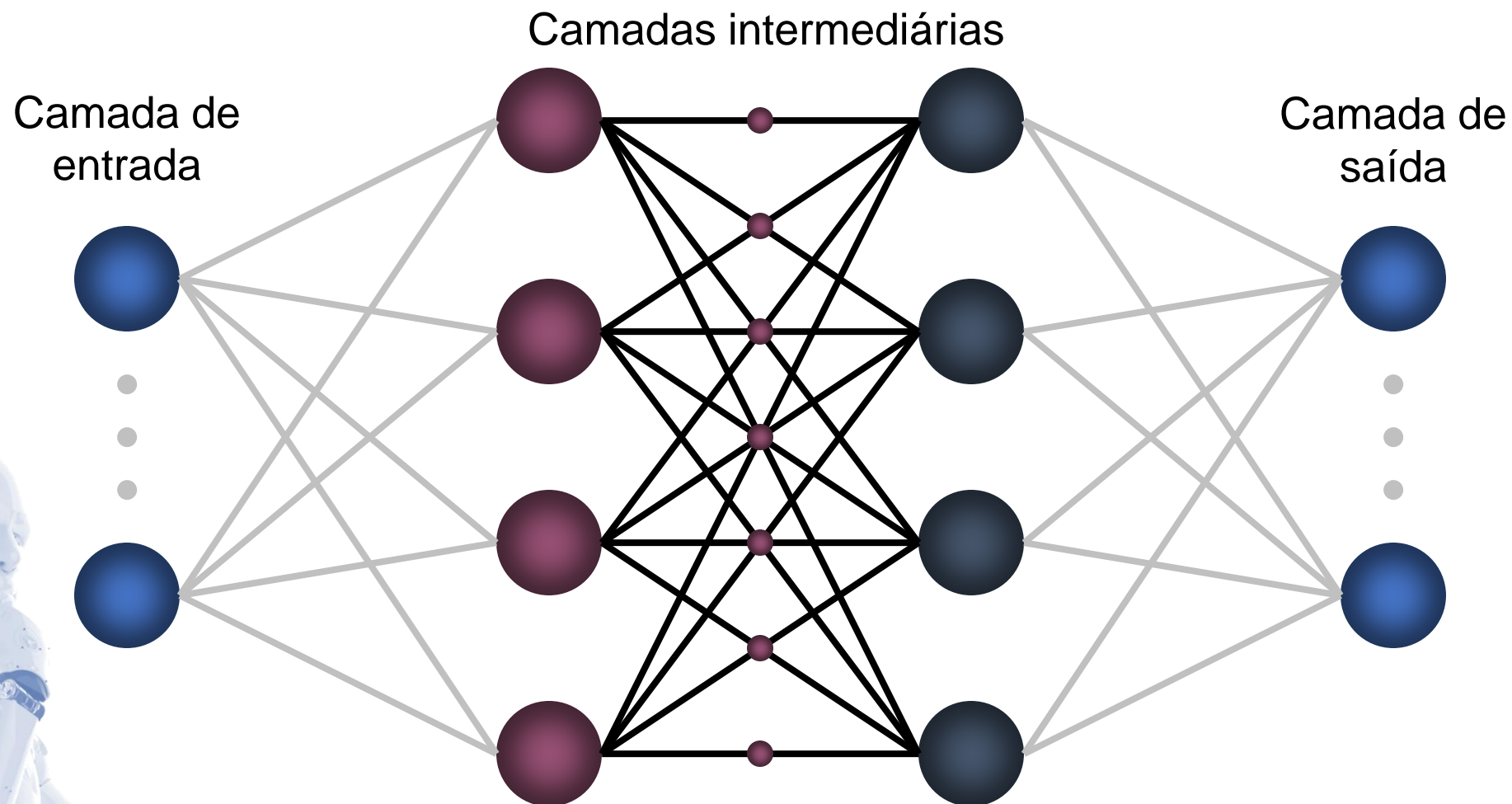
Fase *forward*

Entrada é apresentada à primeira camada da rede e propagado em direção às saídas.



Fase *forward*

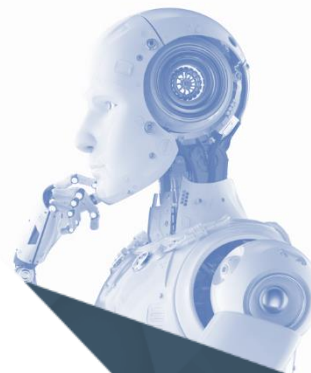
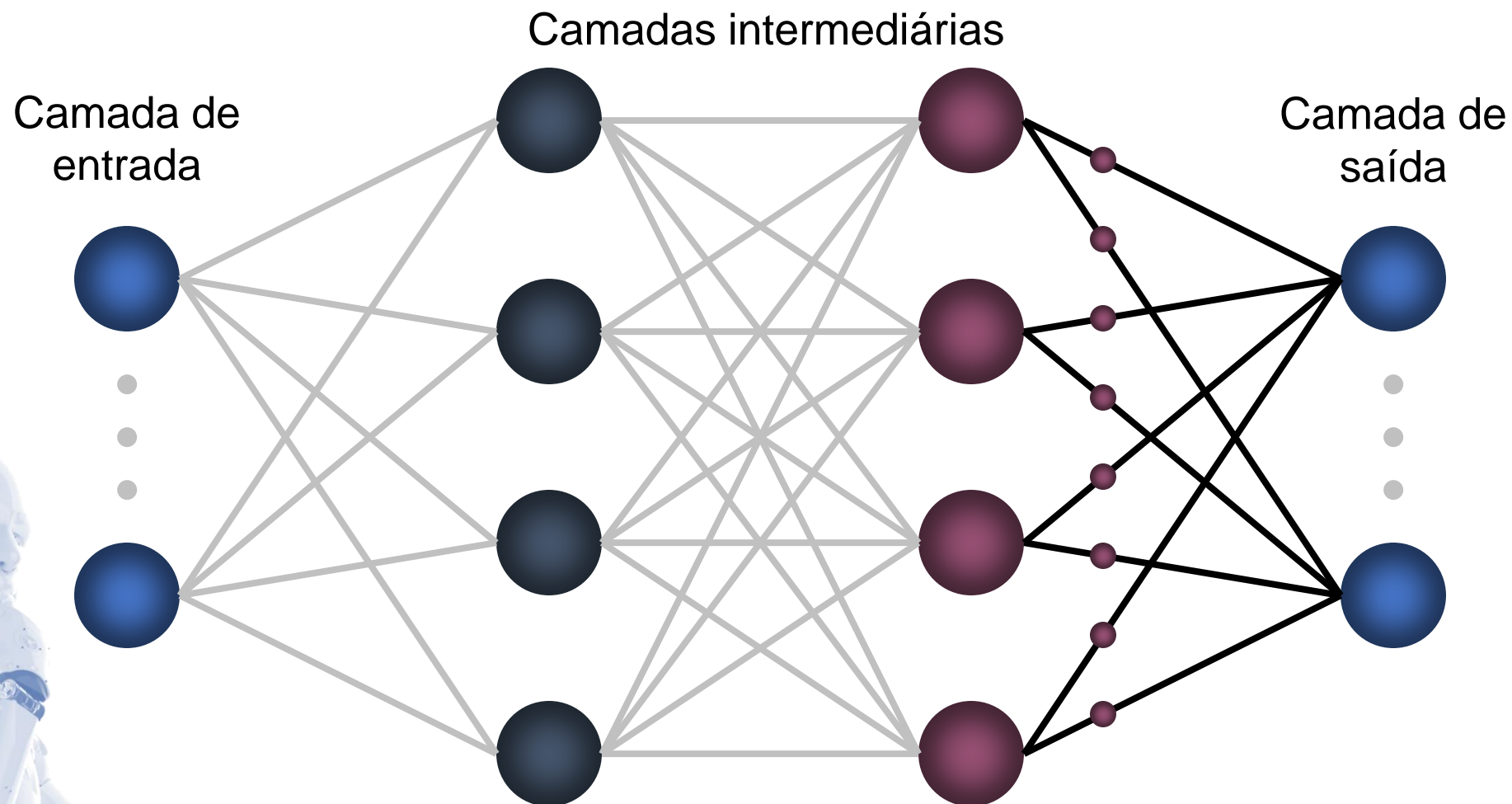
*Os neurônios da camada i calculam
seus sinais de saída e propagam
à camada $i + 1$*



Fase *forward*

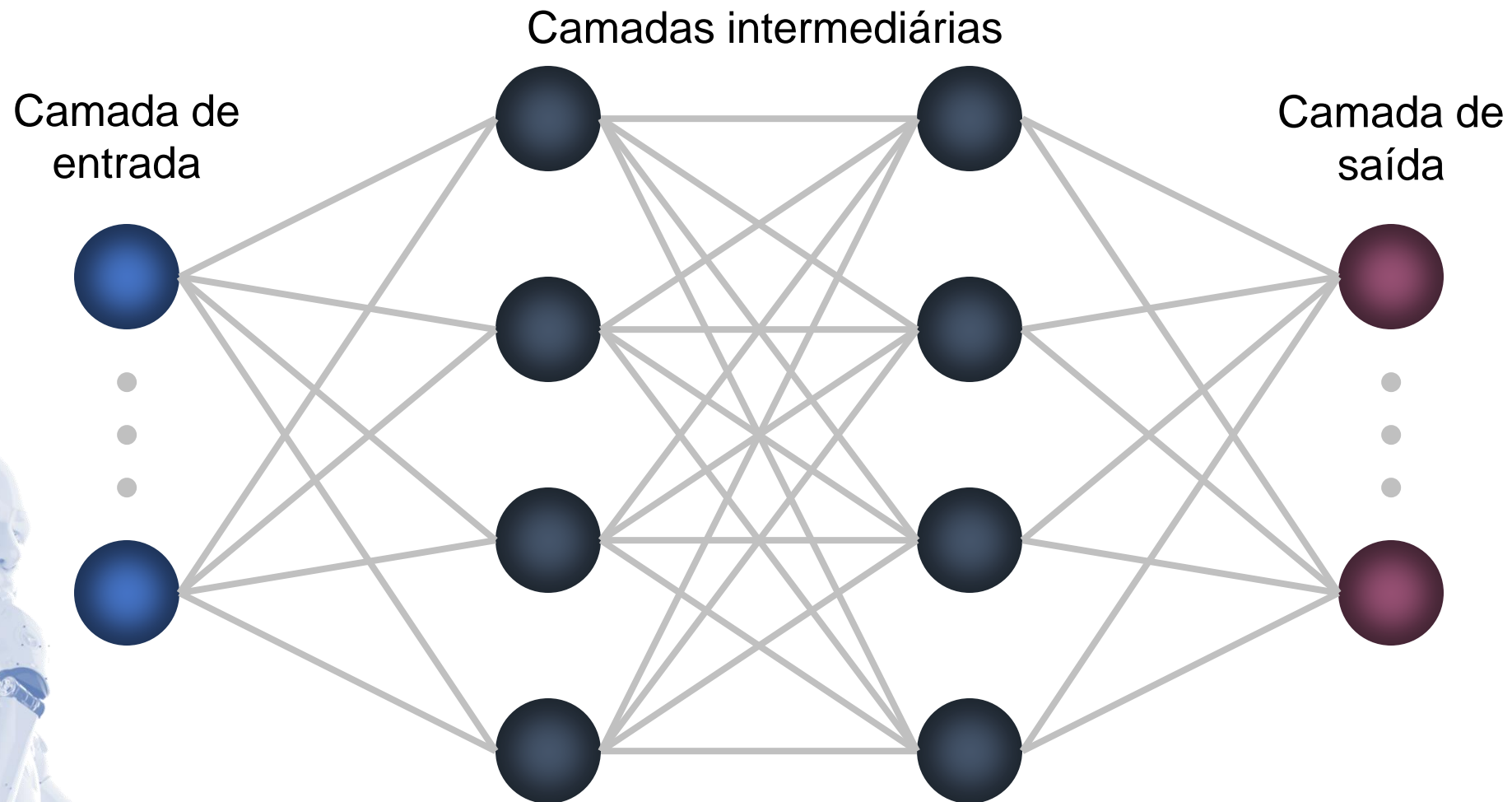
*A última camada oculta calcula
seus sinais de saída e os envia*

à camada de saída

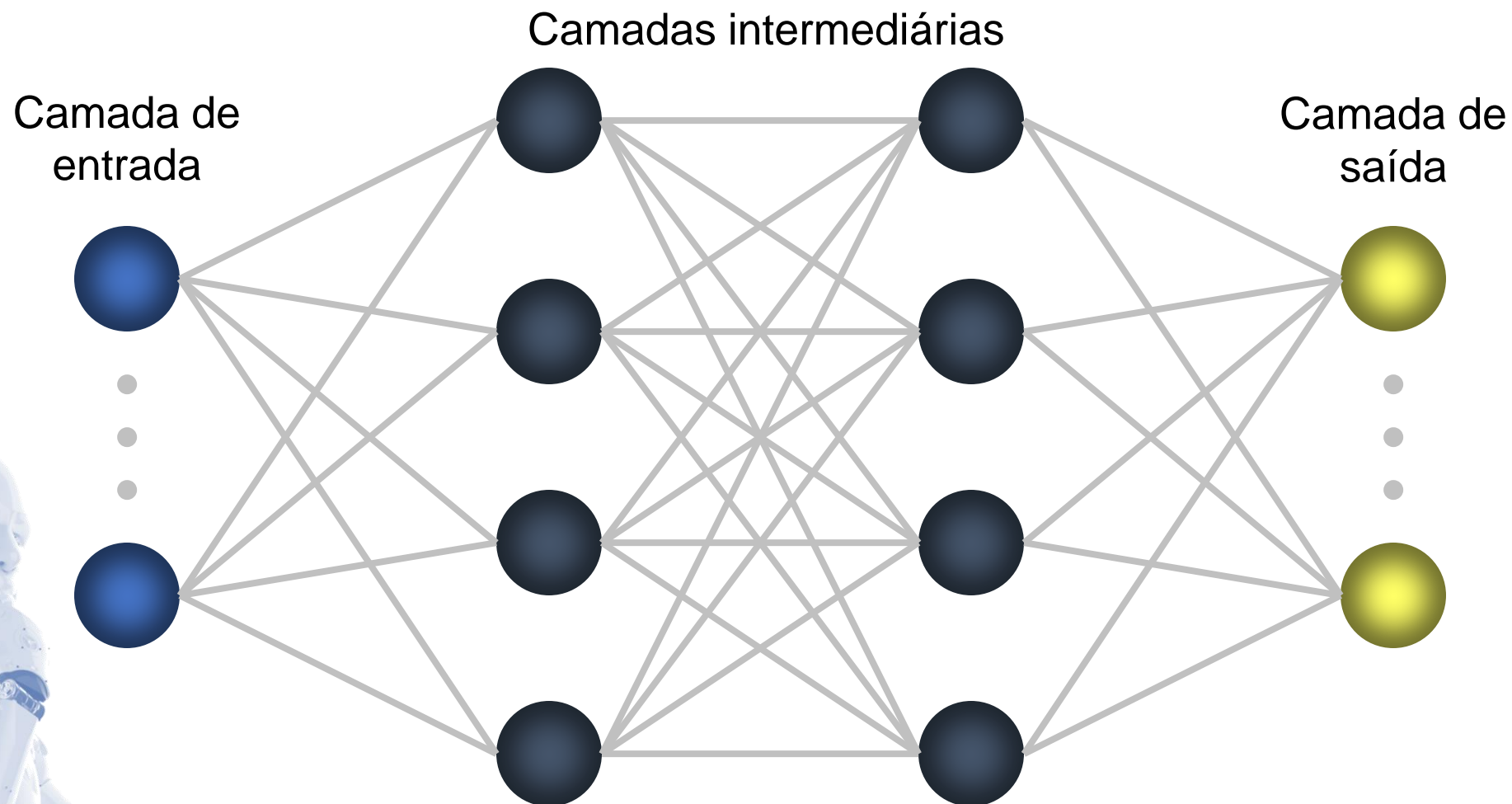


Fase *forward*

A camada de saída calcula os valores de saída da rede.

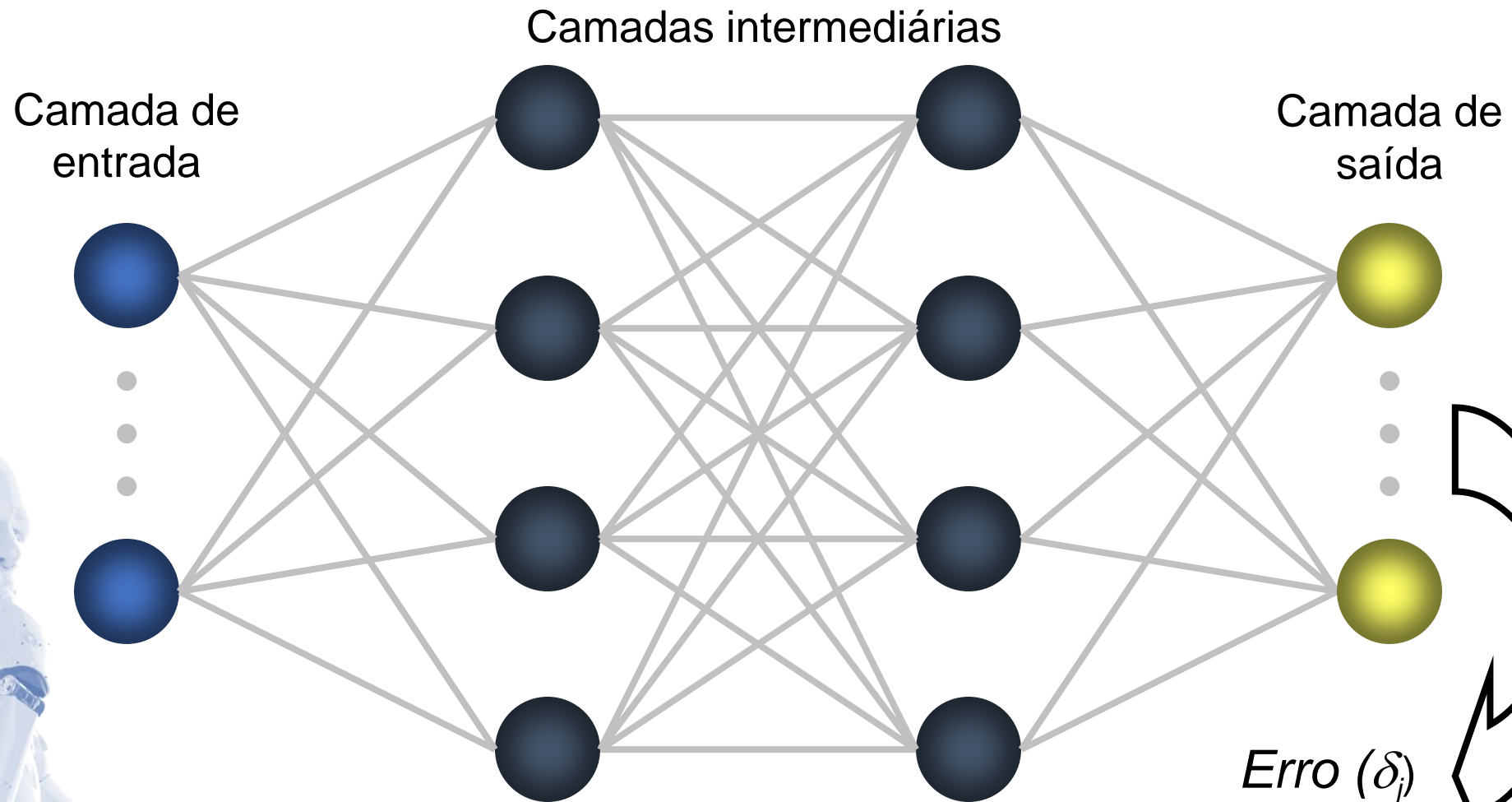


Fase *backward*



Fase *backward*

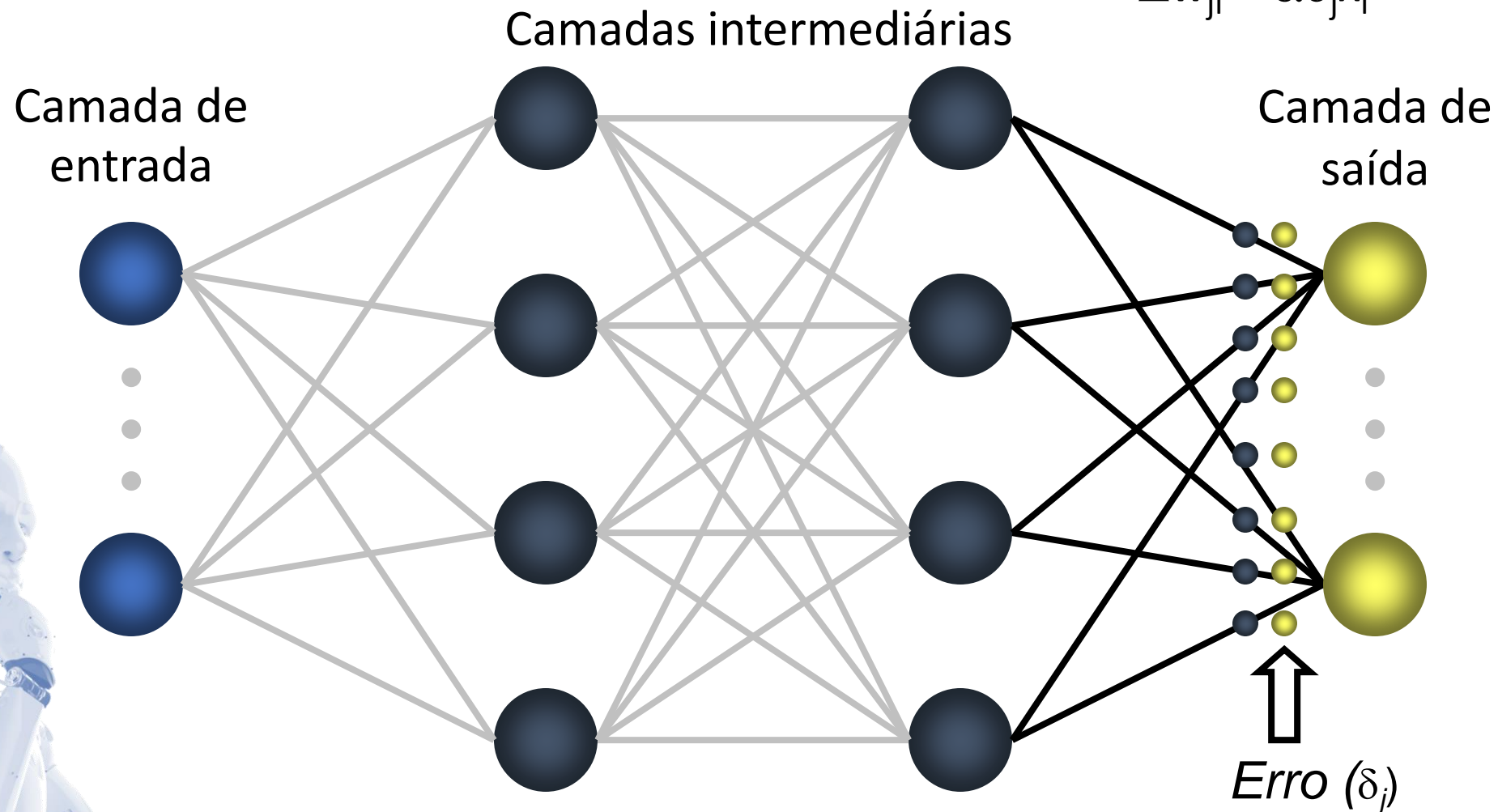
A camada de saída
calcula o erro da rede: δ_j



Fase *backward*

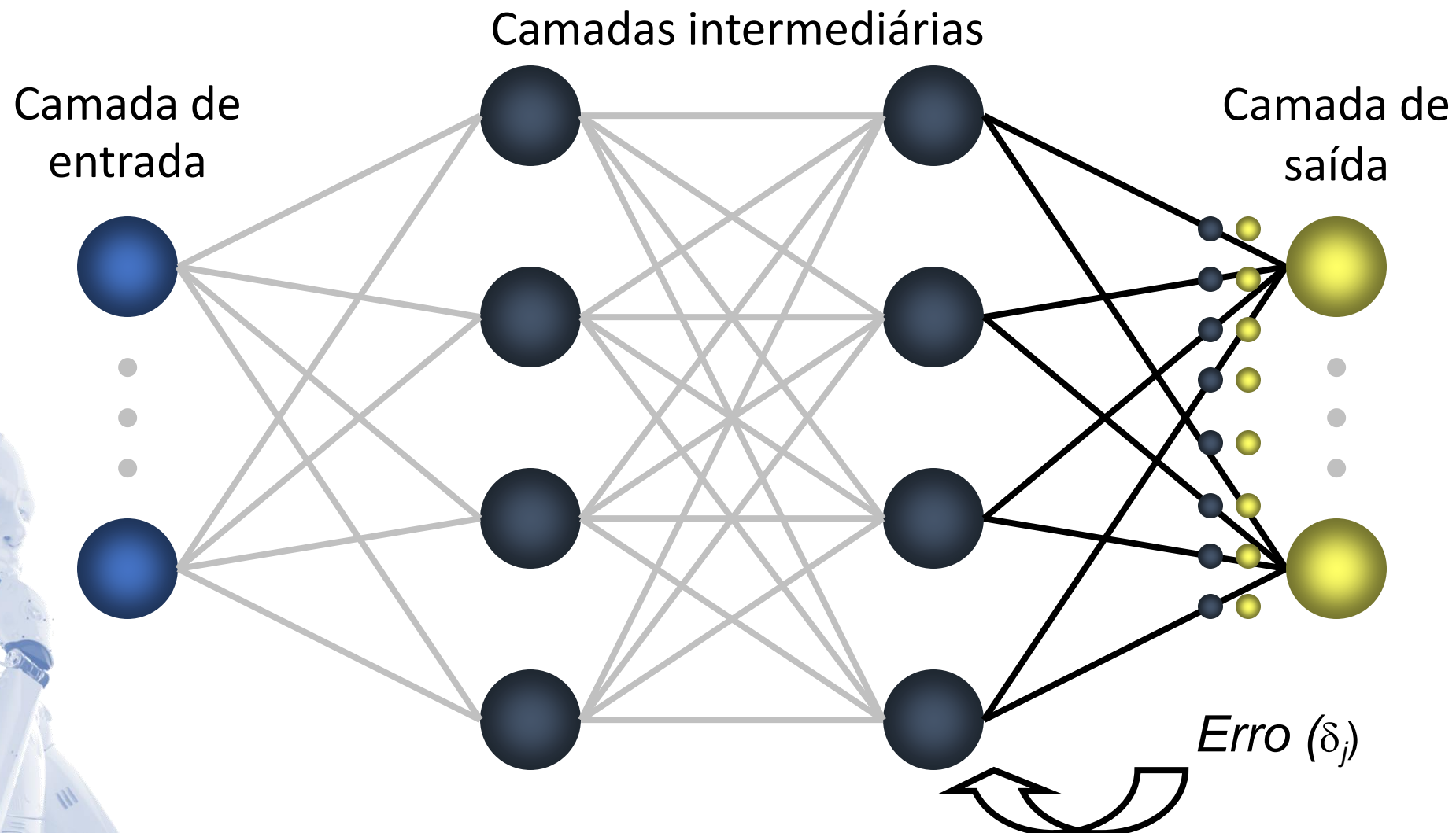
*Calcula o termo de correção dos pesos
(a atualização será feita depois)*

$$\Delta w_{ji} = \alpha \delta_j x_i$$



Fase *backward*

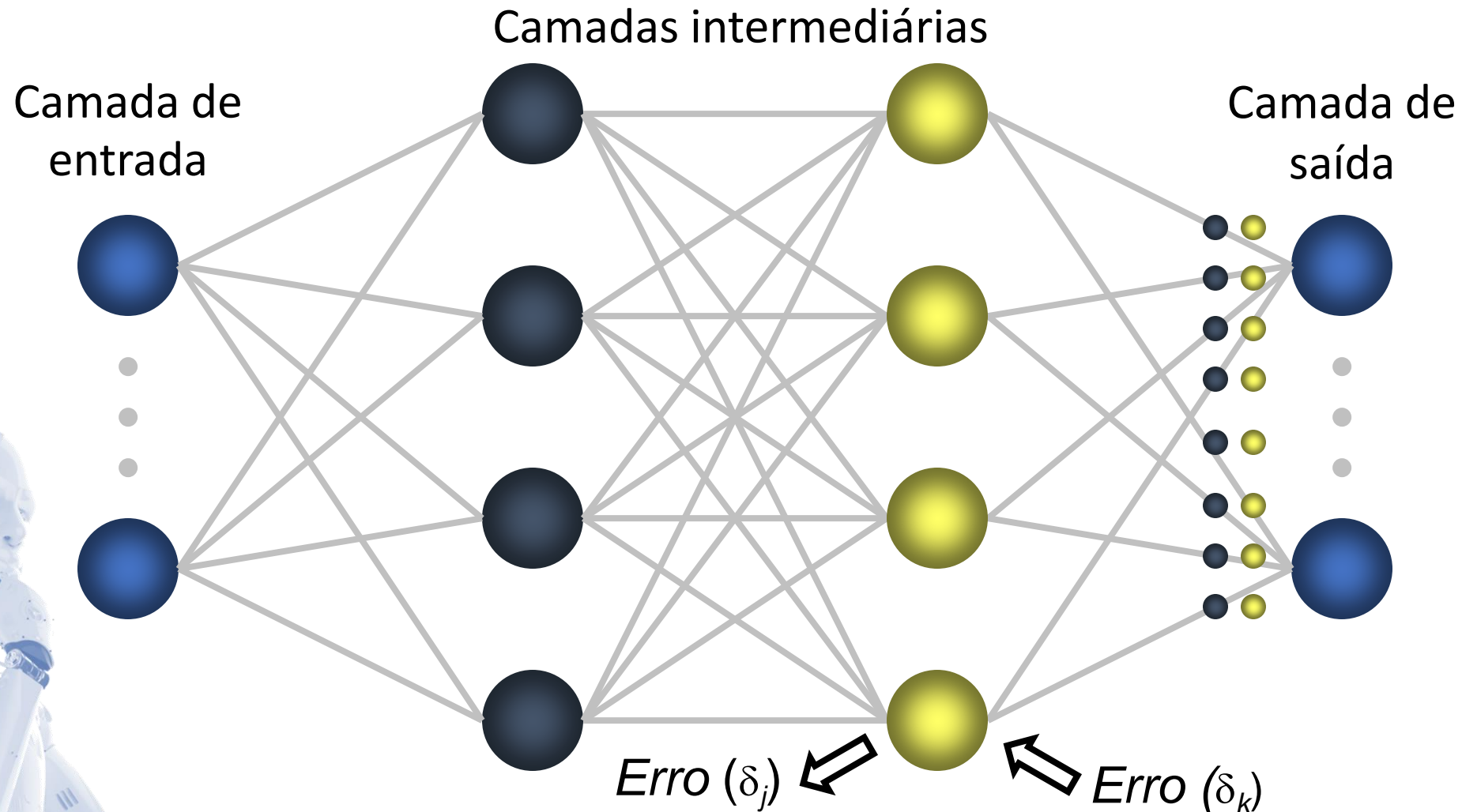
*Envia o erro para a
última camada oculta*



Fase *backward*

A camada oculta calcula o seu erro

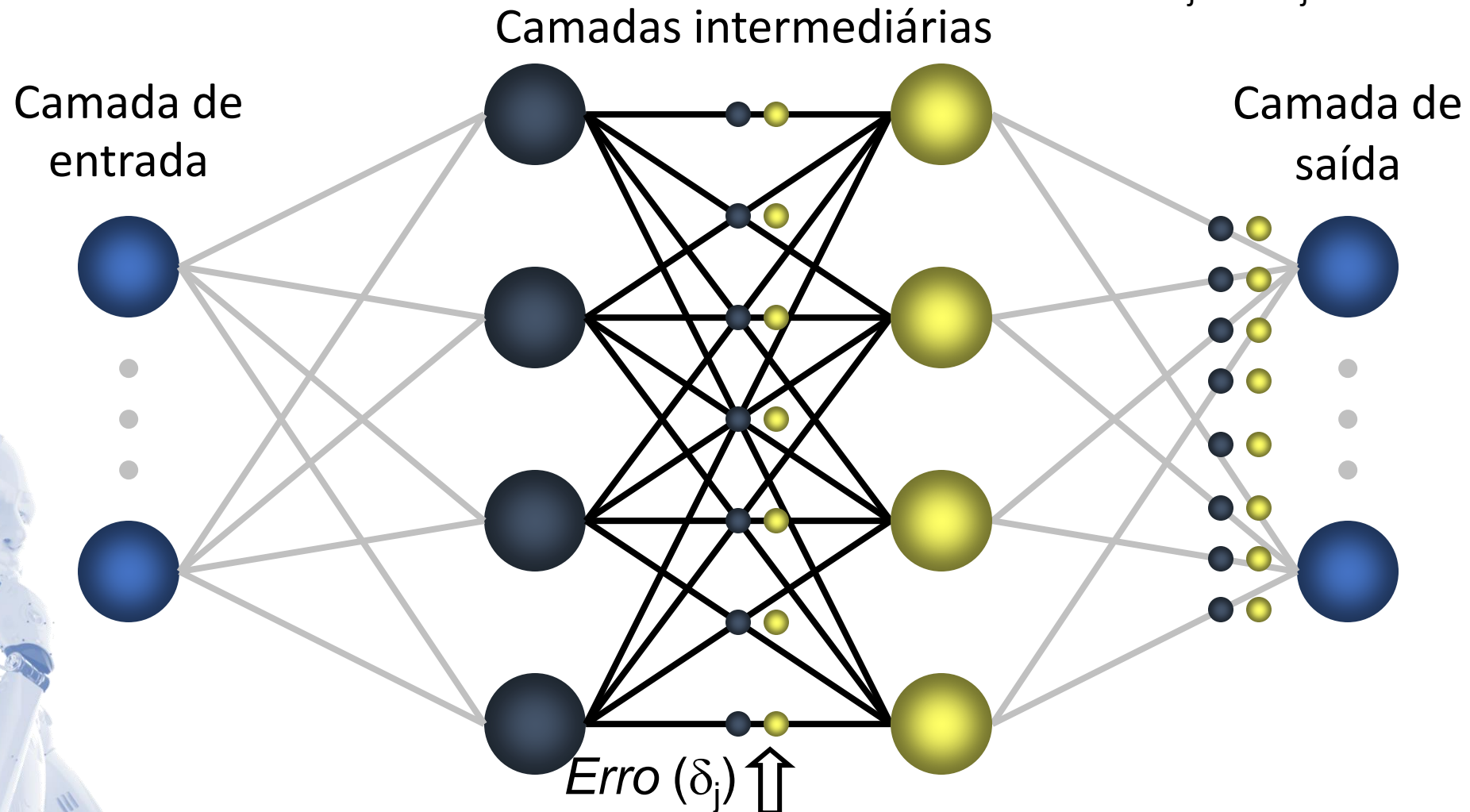
$$\delta_j = f'(u_j) \cdot \sum \delta_k w_{jk}$$



Fase *backward*

*Calcula o termo de correção dos pesos
(a atualização será feita depois)*

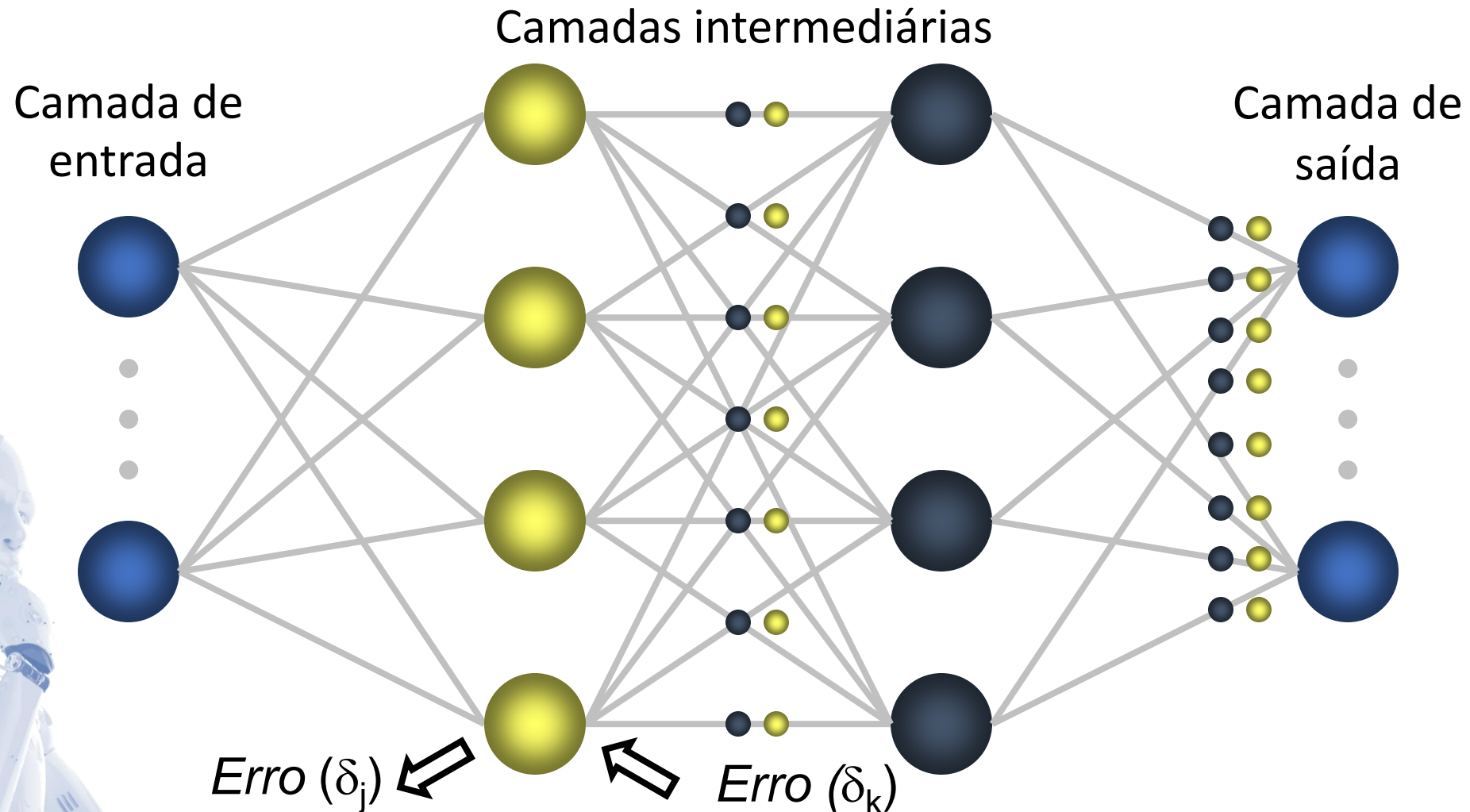
$$\Delta w_{ij} = \alpha \delta_j x_i$$



Fase *backward*

A camada oculta calcula o seu erro

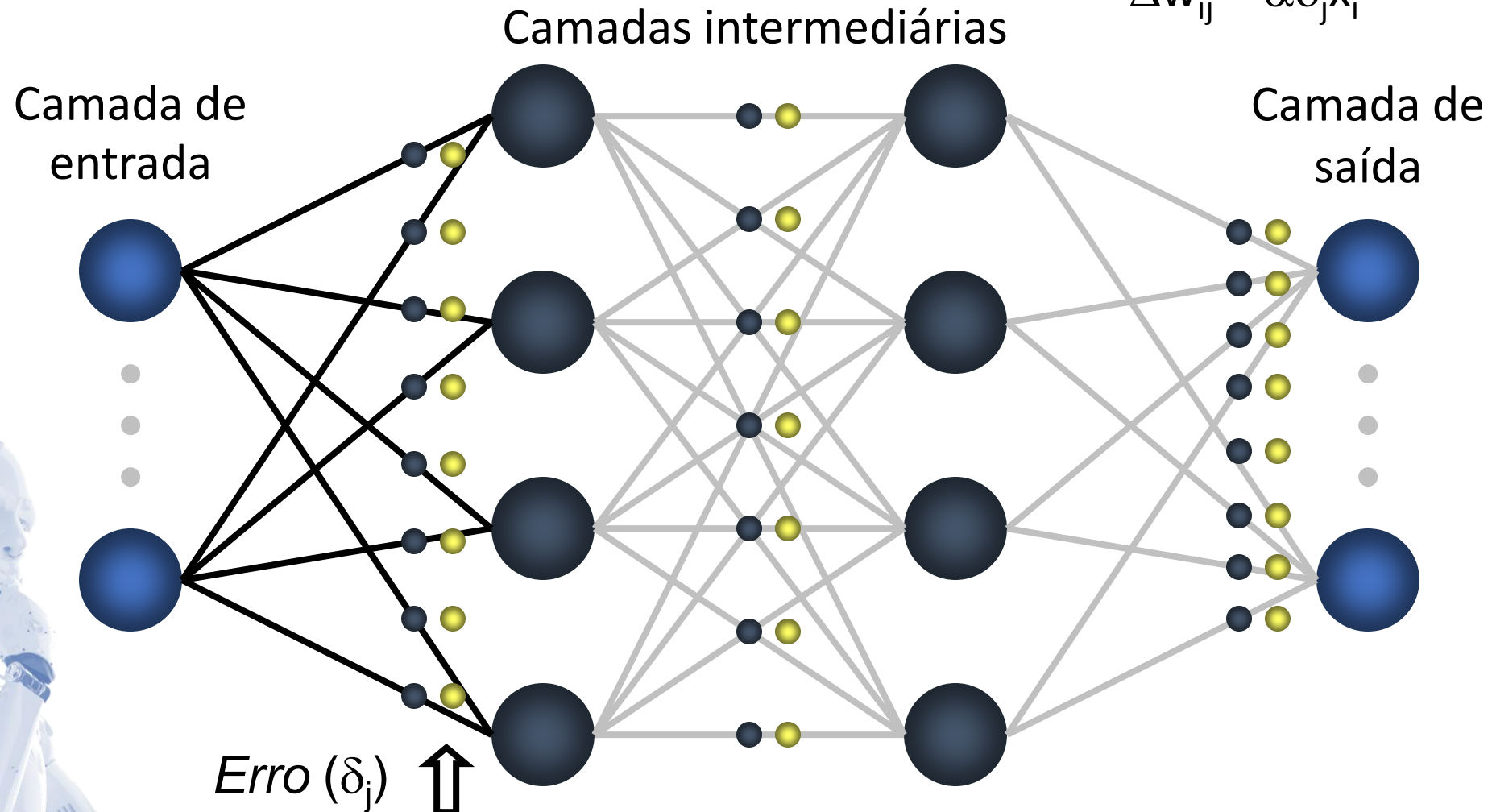
$$\delta_j = f'(u_j) \cdot \sum \delta_k w_{lk}$$



Fase *backward*

*Calcula o termo de correção dos pesos
(a atualização será feita depois)*

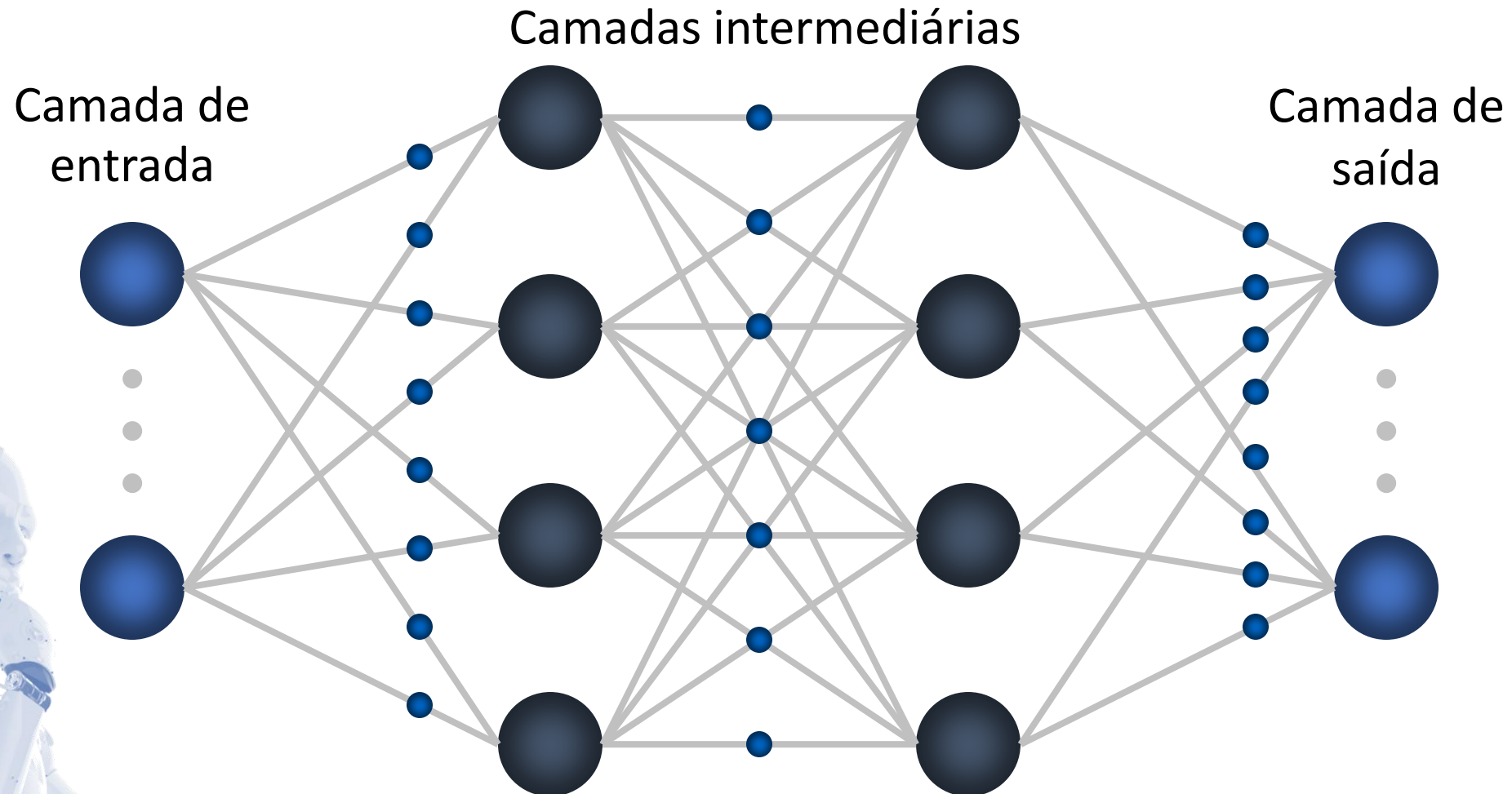
$$\Delta w_{ij} = \alpha \delta_j x_i$$



Fase *backward*

Cada unidade atualiza seus pesos

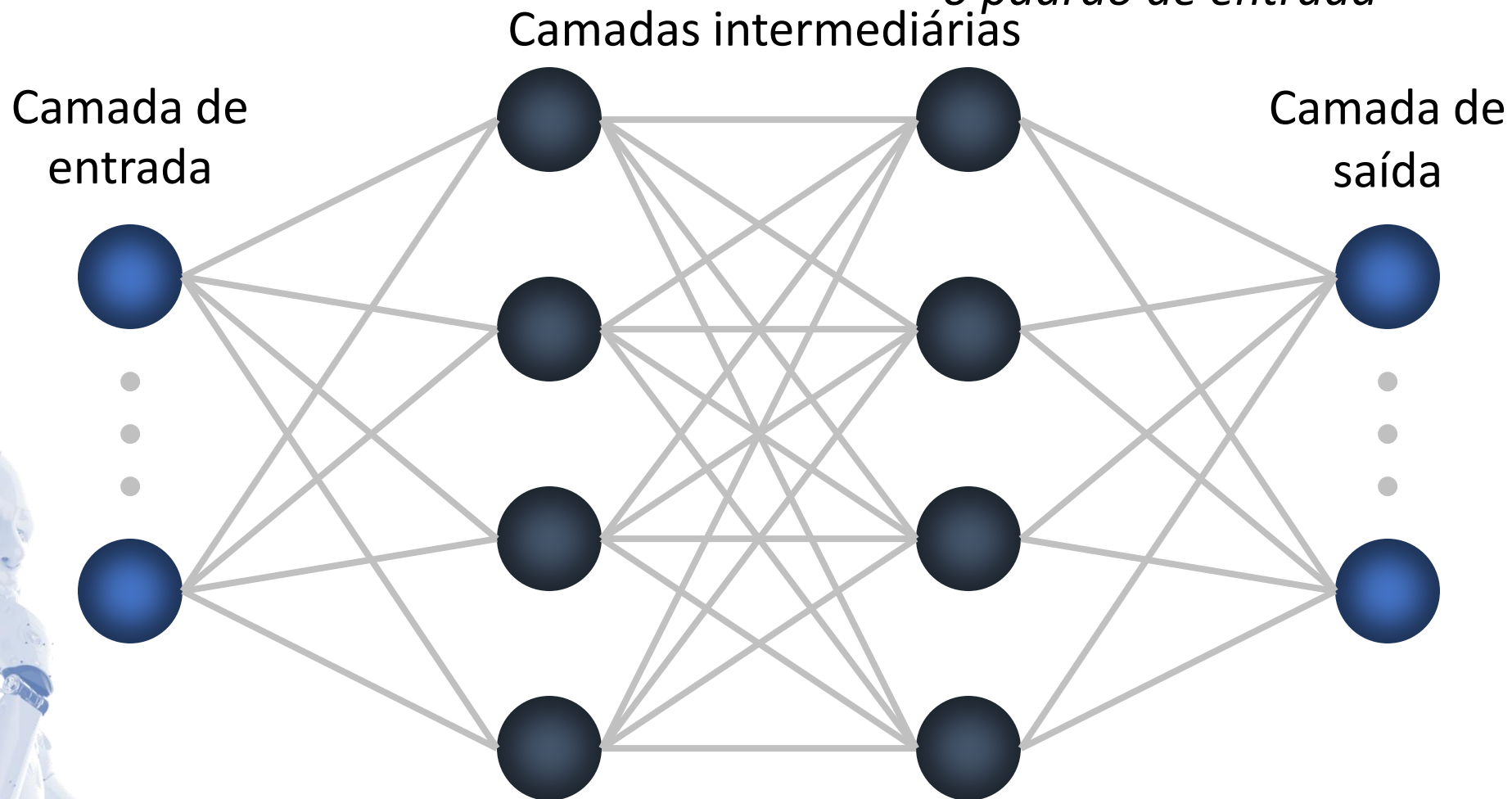
$$w_{ij}(\text{novo}) = w_{ij}(\text{velho}) + \Delta w_{jk}$$



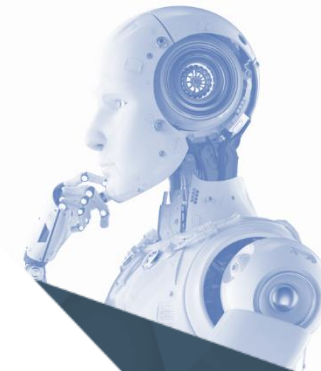
back propagation

*Repete-se o processo enquanto
enquanto a rede não aprender*

o padrão de entrada



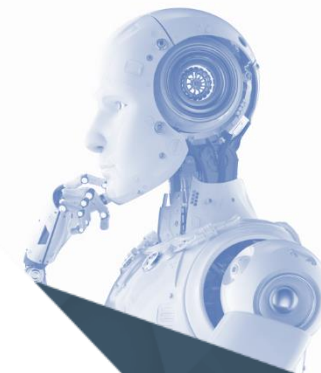
Considerações sobre o treinamento



Cuidados e possibilidades

- Deve-se tomar cuidado para que o *training set* seja representativo do conjunto de entradas e saídas possíveis
- Em caso de mudança significativa nos conjuntos de entrada e saída, deve-se treinar a rede novamente.
- Pode-se fazer a rede adaptativa, de forma que ela continue sendo treinada durante a execução (aprendizado por reforço)

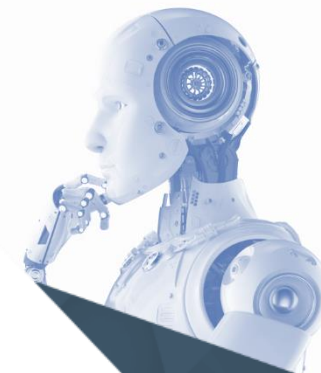
Considerações sobre o treinamento



Treinamento pode ser

- Por padrão (on-line, incremental) - Ajusta pesos a cada padrão de treinamento apresentado.
- Por ciclo (batch) - Ajusta pesos depois de apresentar todos os padrões de treinamento (considera um erro médio para o conjunto de padrões).
- Estático - Topologia fixa (ajusta somente os pesos).
- Dinâmico – Topologia ajustável (ajusta pesos e topologia).

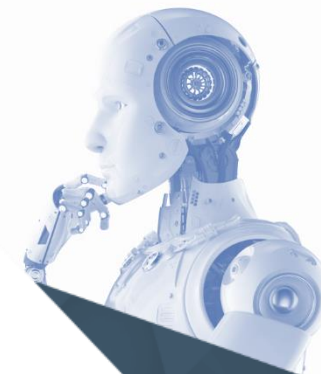
Considerações sobre o treinamento



Algumas dificuldades

- Não existe uma regra eficiente para treinamento e para encontrar uma arquitetura ideal. O treinamento é bastante experimental e as vezes demorado, em função do elevado número de combinação de parâmetros.
- Algumas ferramentas oferecem recursos para ajudar na busca de parâmetros/configurações ideais, mas consome tempo.

Considerações sobre o treinamento



Algumas dificuldades

- *Mínimos locais*: o treinamento pode parar prematuramente.
- *Overfitting*: uma rede hiper treinada (treinou por muito tempo), ou possui mais neurônios do que precisa, acaba se ajustando a um grupo específico de dados, diminuindo a sua generalização.
- Existem diversos algoritmos de treinamentos: *back propagation*, *Rprop*, *Quickprop*, *Levenberg-Marquardt*.

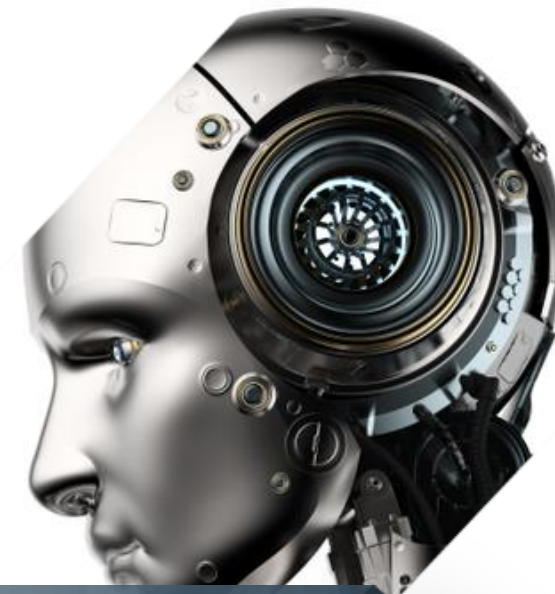


PUC Minas
Virtual



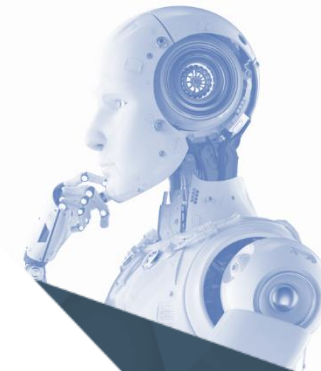
Rede Neural Multicamadas Perceptron

Aplicações



PUC Minas Virtual

Perceptron Multicamadas: Aplicação



Predição de vendas

- Valores médios anuais de vendas (Bilhões R\$)
- Índice do volume de vendas do varejo brasileiro (IBGE)
- Supermercados/Hipermercados

2000	2001	2002	2003	2004	2005
64,41	65,32	65,69	61,57	65,92	67,44

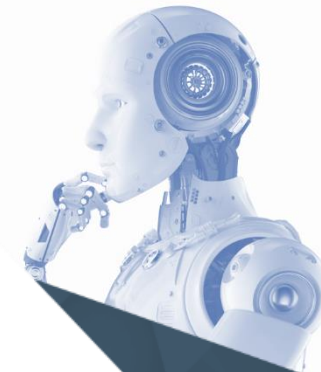
2006	2007	2008	2009	2010	2011
72,78	77,81	81,65	88,46	96,16	88,85

Como serão os próximos anos?

2012	2013	...
?	?	?

Perceptron Multicamadas: Aplicação

Predição de vendas – Estratégia da Janela deslizante



Dados Originais

2000	2001	2002	2003	2004	2005
64,41	65,32	65,69	61,57	65,92	67,44
2006	2007	2008	2009	2010	2011
72,78	77,81	81,65	88,46	96,16	88,85

Montagem da base de treinamento

Ano1, Ano2, Ano3, Ano4, Ano5 → Ano6

Ano2, Ano3, Ano4, Ano5, Ano6 → Ano7

Ano3, Ano4, Ano5, Ano6, Ano7 → Ano8

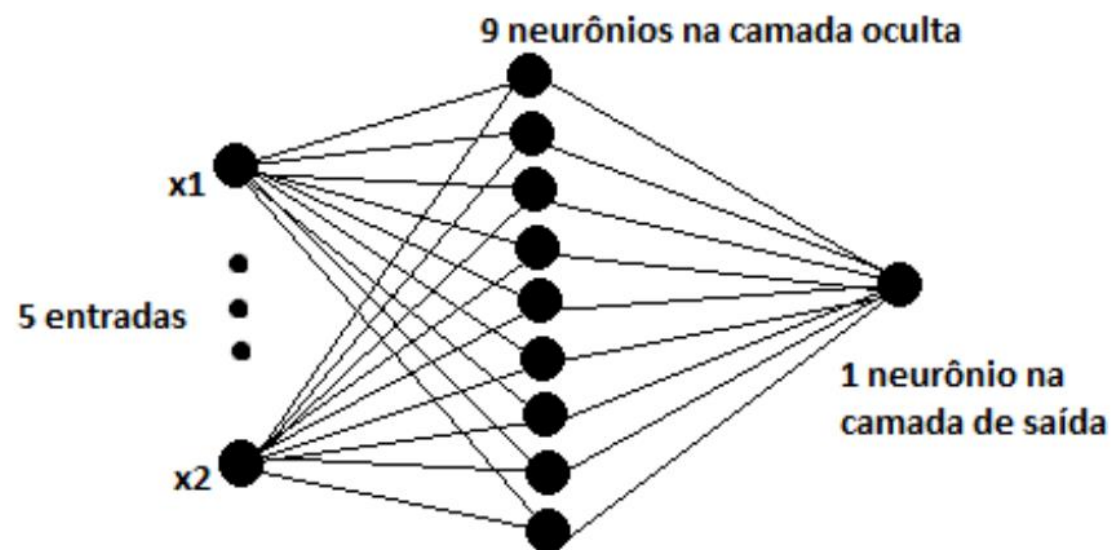
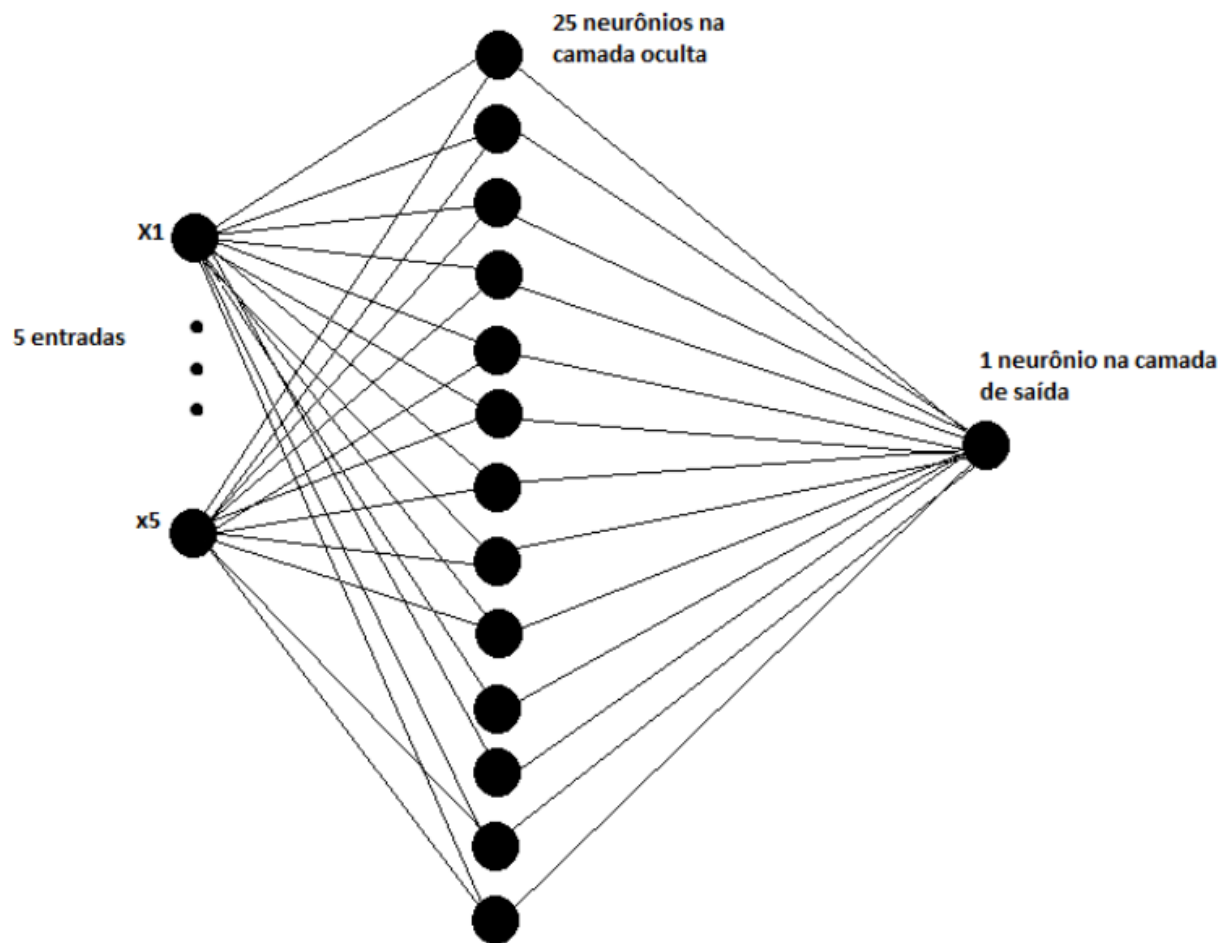
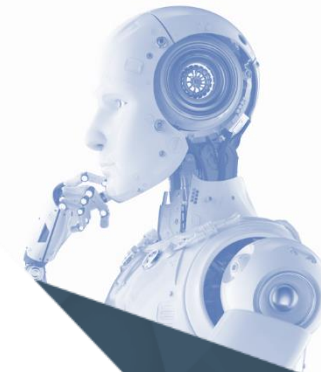
....

Dados de Treinamento

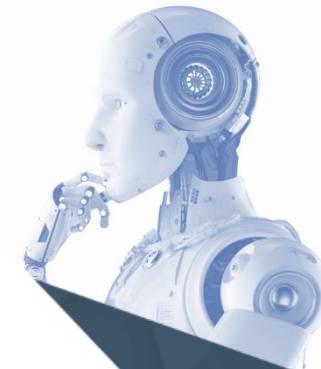
X1	X2	X3	X4	X5	Sd
64,41	65,32	65,69	61,57	65,92	67,44
65,32	65,69	61,57	65,92	67,44	72,78
65,69	61,57	65,92	67,44	72,78	77,81
61,57	65,92	67,44	72,78	77,81	81,65
65,92	67,44	72,78	77,81	81,65	88,46
67,44	72,78	77,81	81,65	88,46	96,16
72,78	77,81	81,65	88,46	96,16	88,85

Perceptron Multicamadas: Aplicação

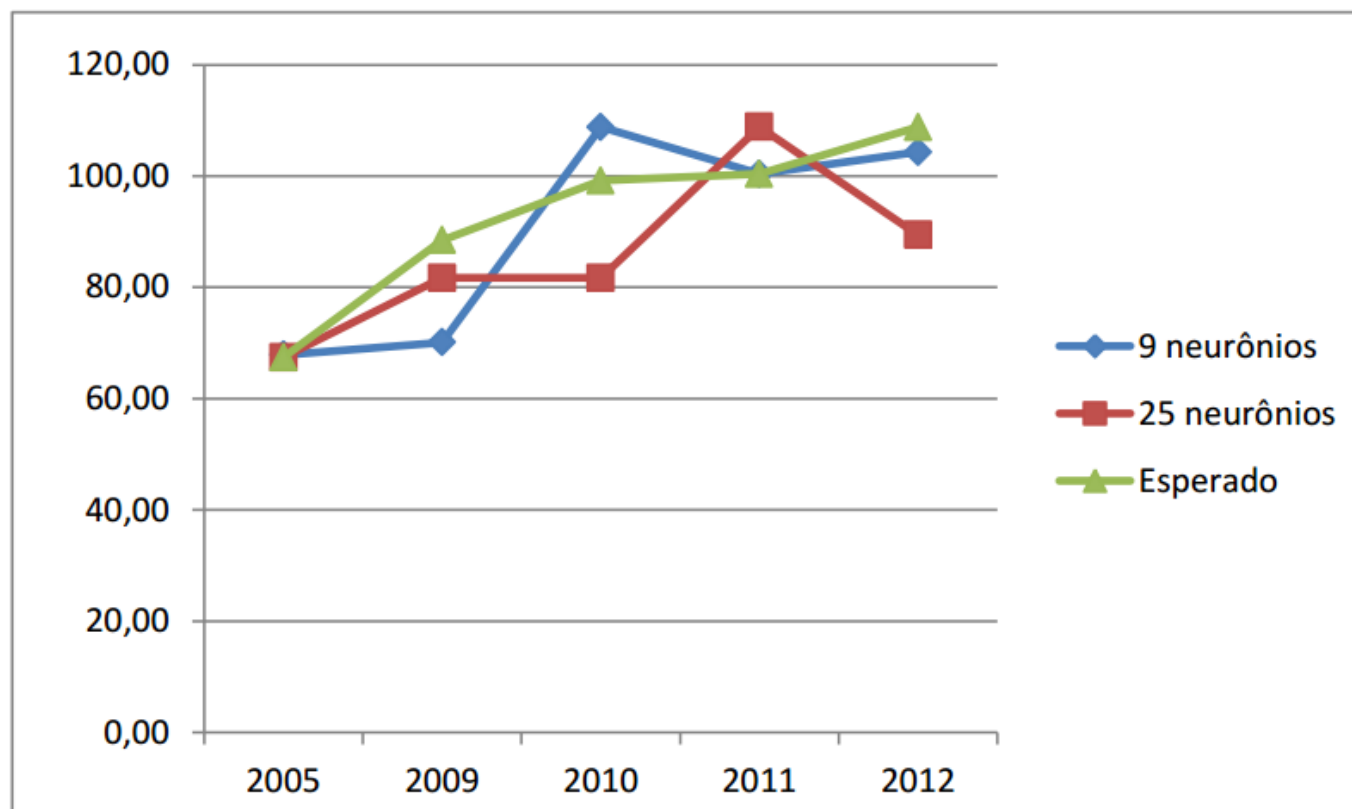
Predição de vendas – Modelos com 25 e 9 neurônios



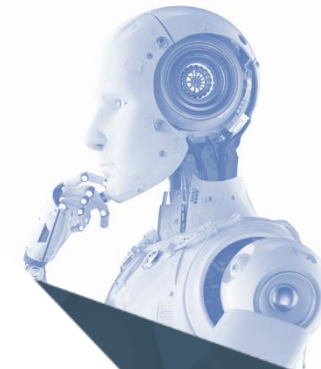
Perceptron Multicamadas: Aplicação



Predição de vendas – Resultados do modelo

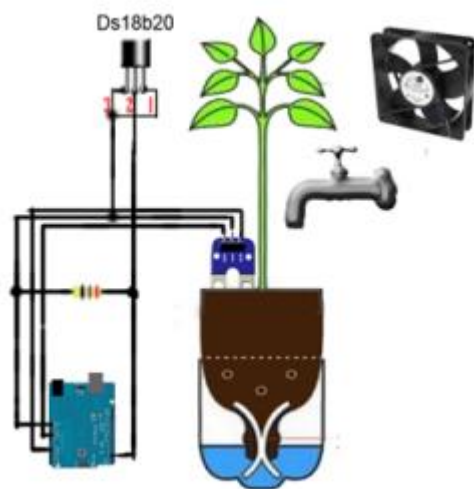


Perceptron Multicamadas: Aplicação

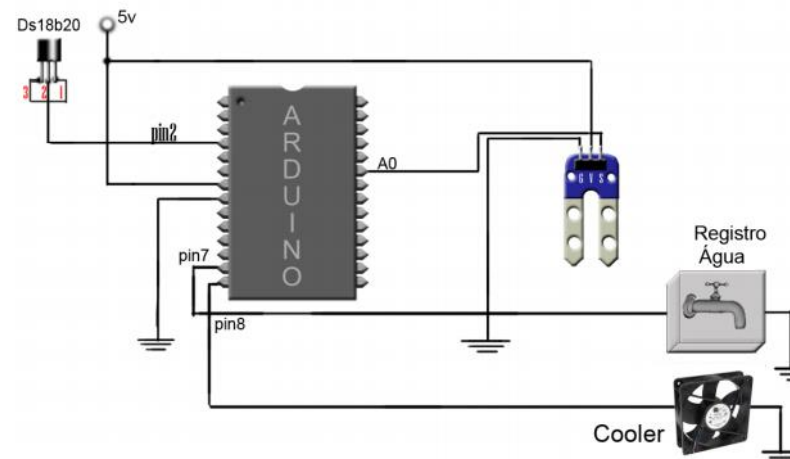


Sistema de controle de irrigação

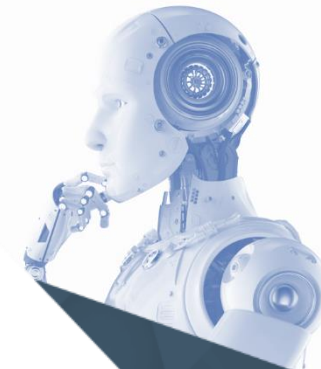
- Ideia: controlar irrigação e ventilação de uma planta com base na temperatura e umidade



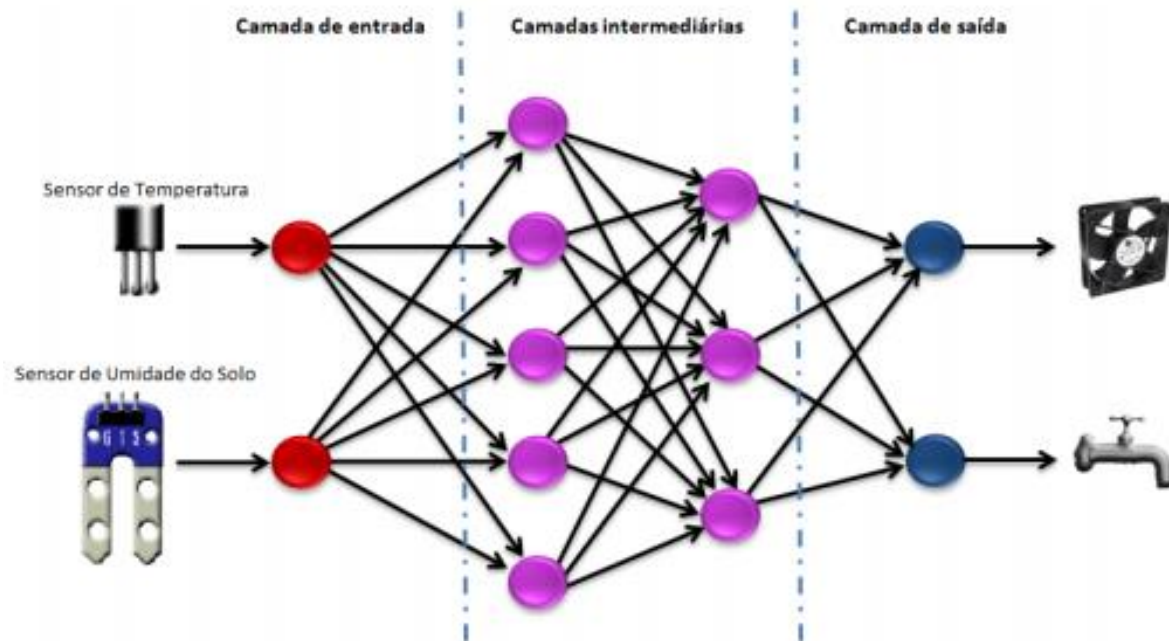
Uso de arduino



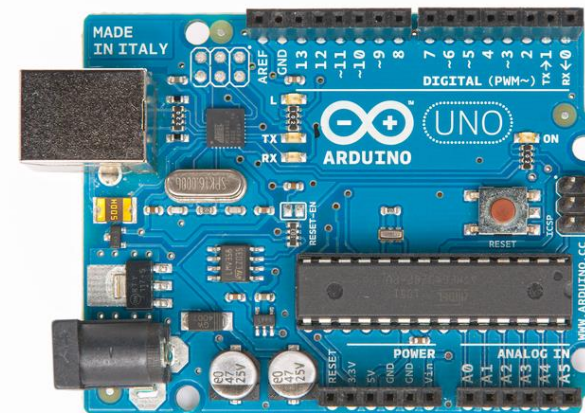
Perceptron Multicamadas: Aplicação



Sistema de controle de irrigação



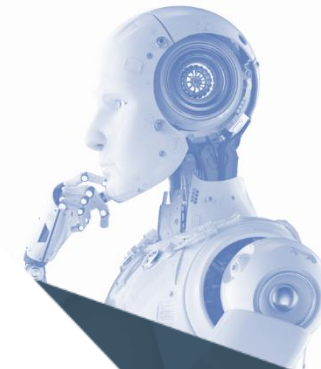
Arduino UNO



Função de ativação (*step*)

$$f(x) = \begin{cases} 0, & \text{se } x < a \\ 1, & \text{se } x \geq a \end{cases}$$

Perceptron Multicamadas: Aplicação



Sistema de controle de irrigação

RESULTADOS OBTIDOS

Entradas		Saídas Esperadas		Saídas da Rede	
Umidade	Temperatura	Registro	Cooler	Registro	Cooler
920	710	0	0	0	0
249	590	1	1	1	1
100	590	1	0	1	0
302	720	1	1	1	1
1022	1020	0	0	0	0
798	897	0	1	0	1
1024	902	0	0	0	0
849	1024	0	1	0	1
603	402	0	0	0	0
499	503	1	0	1	0



PUC Minas
Virtual

