REDES NEURAIS ARTIFICIAIS

PROFA. MARIANA KLEINA

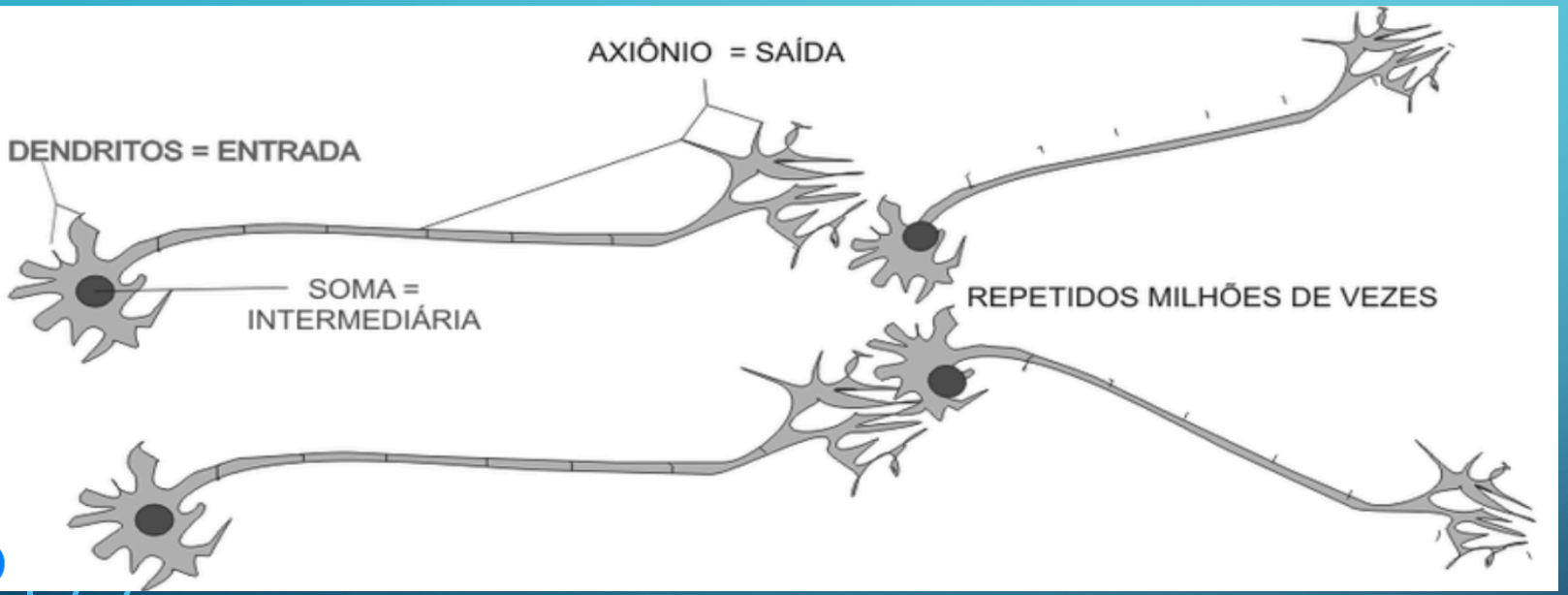
REDES NEURAIS ARTIFICIAIS (RNA)

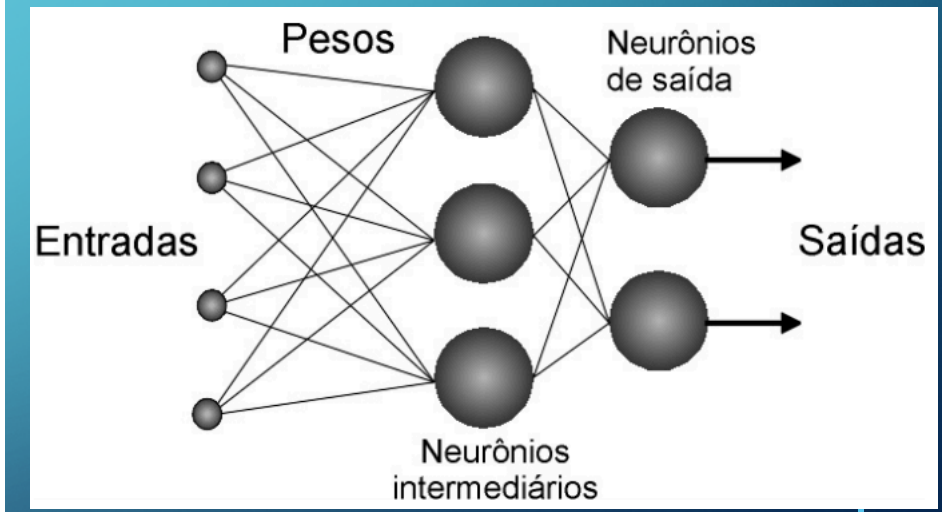
- Tem habilidades de adquirir e armazenar conhecimento para realizar uma tarefa.
- Consistem em modelos inspirados no cérebro humano.
- O neurônio artificial imita o neurônio biológico.

SÃO CAPAZES DE:

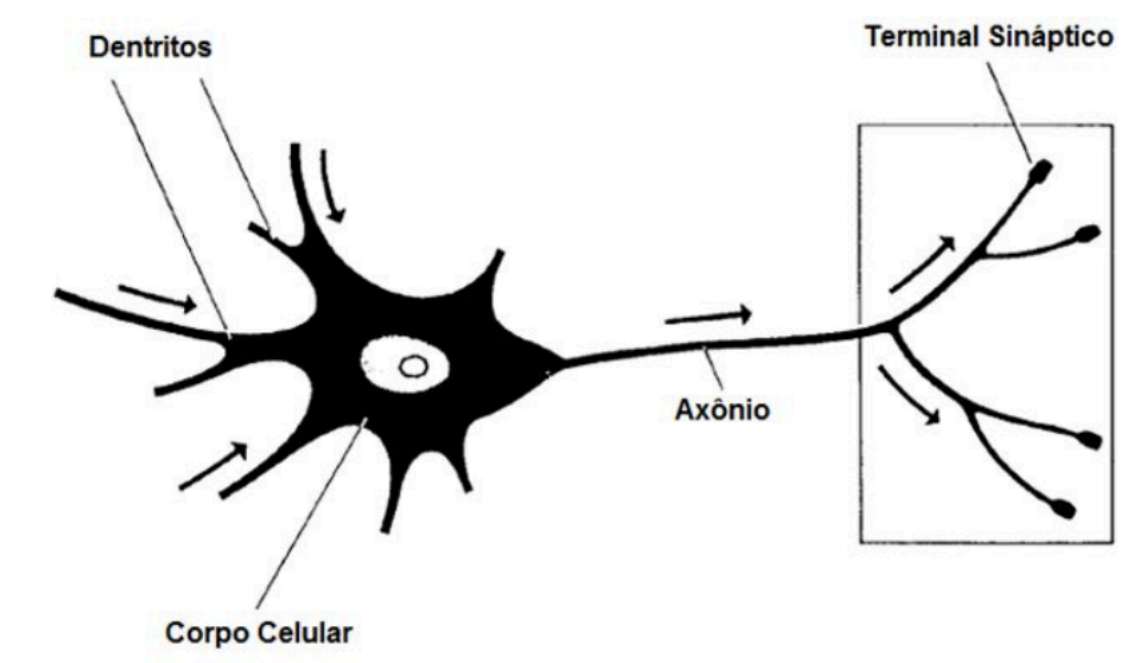
- APRENDER a partir de amostras de treinamento.
- GENERALIZAR a partir do conhecimento adquirido.
- SE ADAPTAR ajustando-se a uma nova realidade.

NEURÔNIO BIOLÓGICO X REDE NEURAL ARTIFICIAL





NEURÔNIO BIOLÓGICO



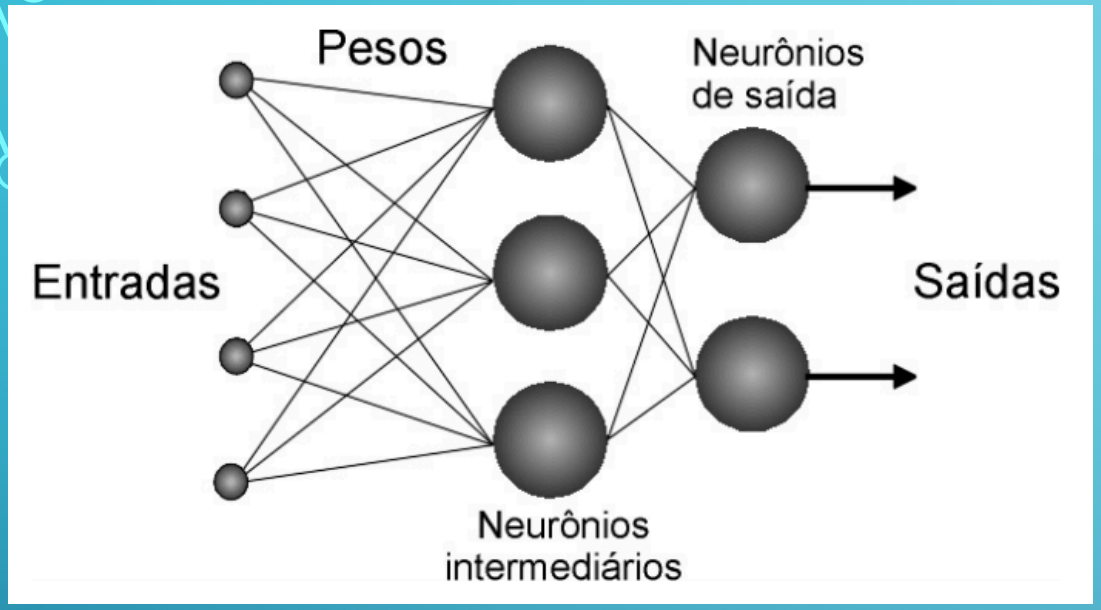
Os **dentritos** são responsáveis pelo recebimento e condução das informações vindas de outros neurônios ou do meio externo onde podem estar em contato.

O **corpo celular** é encarregado de coletar e processar as informações enviadas pelos dentritos.

Um potencial de ativação é produzido e indicará se os impulsos nervosos serão ou não conduzidos para outros neurônios por meio do **axônio**.

A terminação dos axônios é ramificada e recebe a denominação de **terminações sinápticas**, que se conectam, mesmo que sem contato físico com os dentritos de outros neurônios. Esse contato que se encarrega de transferir os impulsos nervosos de um neurônio para outro é chamado de sinapse.

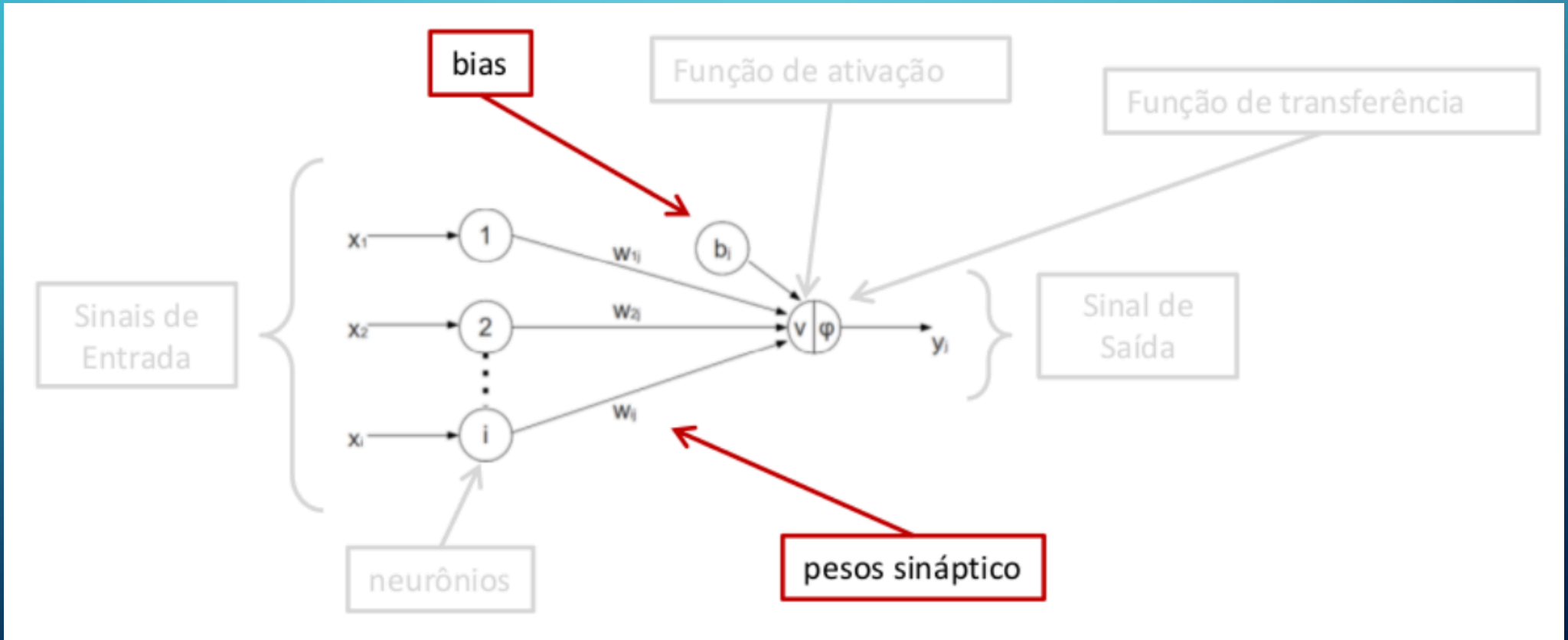
NEURÔNIO ARTIFICIAL



Os neurônios recebem os sinais de entrada. O corpo do neurônio realiza a soma da multiplicação termo a termo entre os sinais de entrada (representando os impulsos elétricos captados pelos dentritos) e os pesos sinápticos que informam o quão relevante é cada uma das entradas no neurônio.

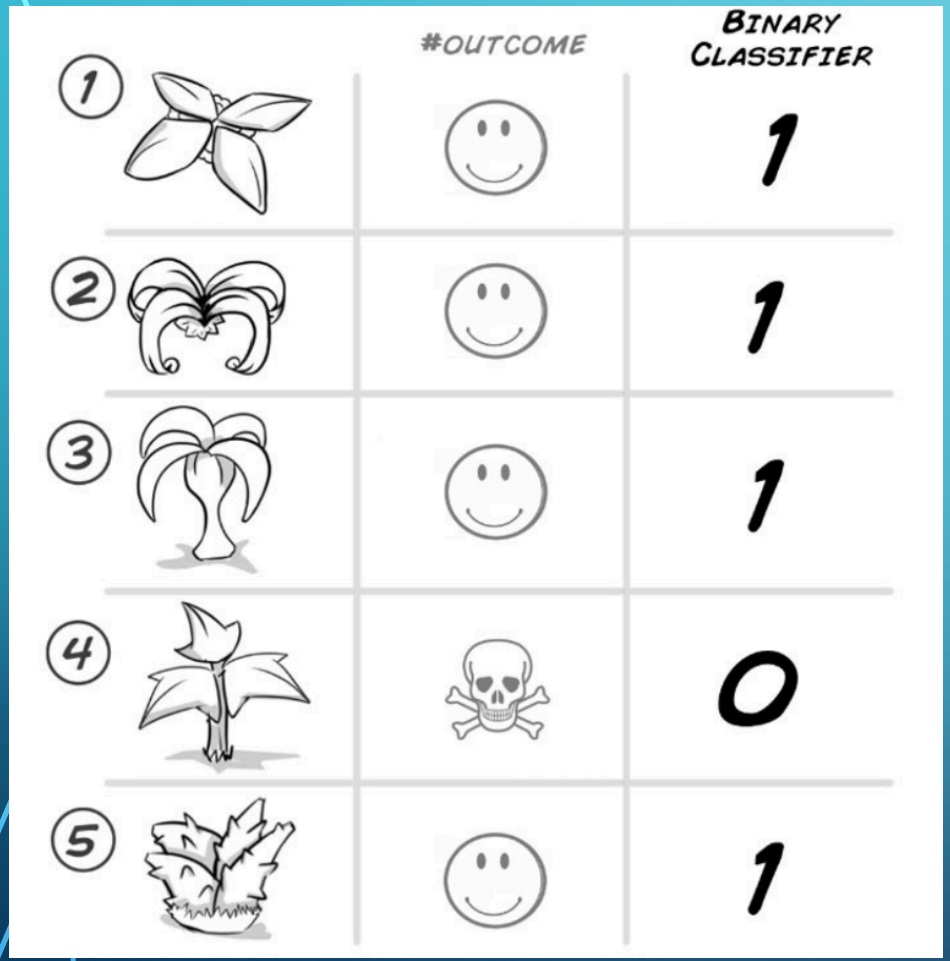
O resultado da soma da multiplicação (+ um fator corretivo chamado *bias*) representa a saída do corpo celular artificial. Tal valor é apresentado a uma função de ativação (representando o axônio) responsável por limitar a saída do neurônio (terminações sinápticas), evitando o acréscimo progressivo dos valores de saída ao longo das camadas da rede.

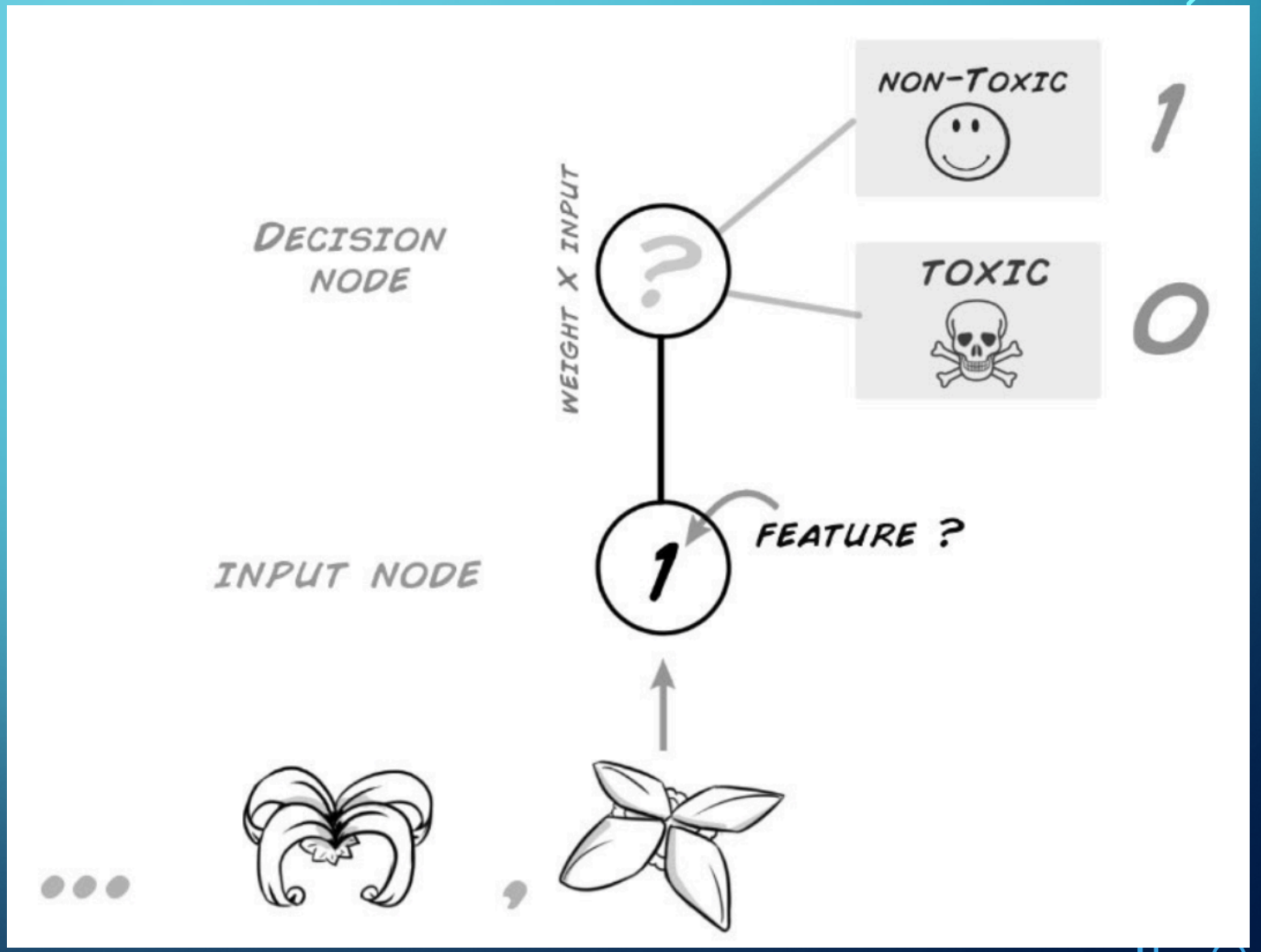
Treinar uma rede neural artificial significa encontrar valores ideais para os pesos sinápticos e *bias*.



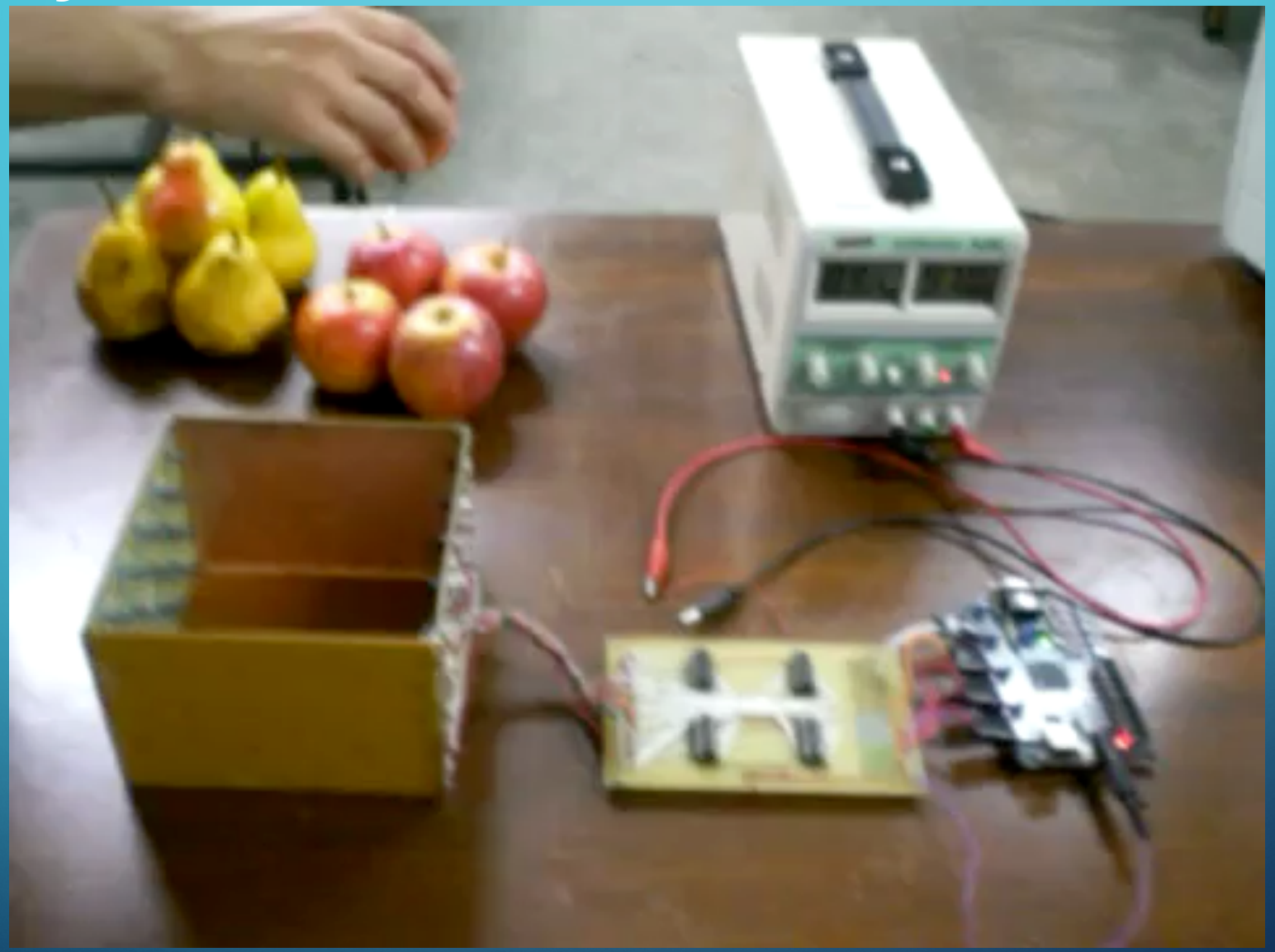
APLICAÇÕES DE RNA

- Classificação





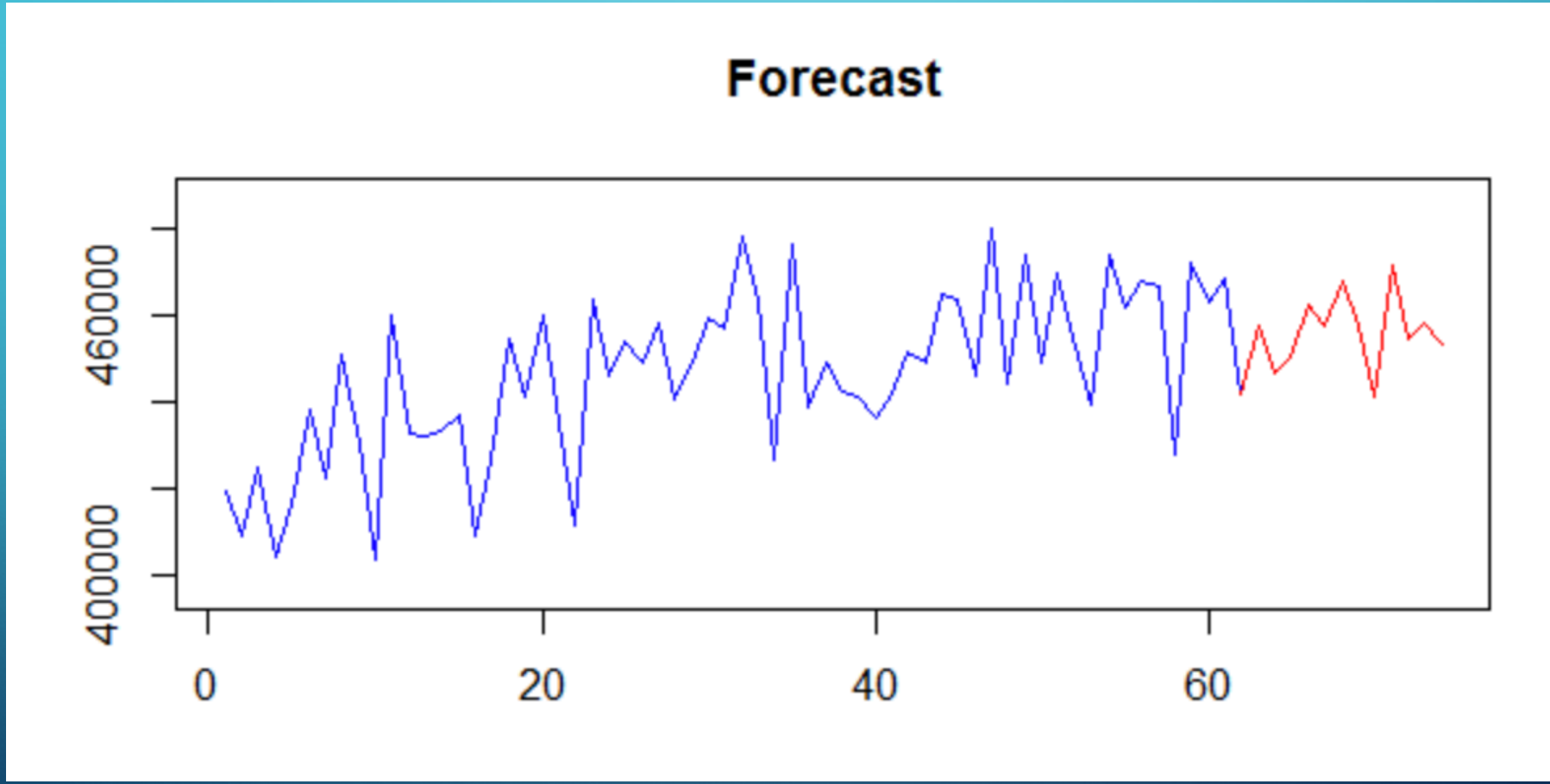
APLICAÇÕES DE RNA



https://www.youtube.com/watch?v=chNXBcd2QNY

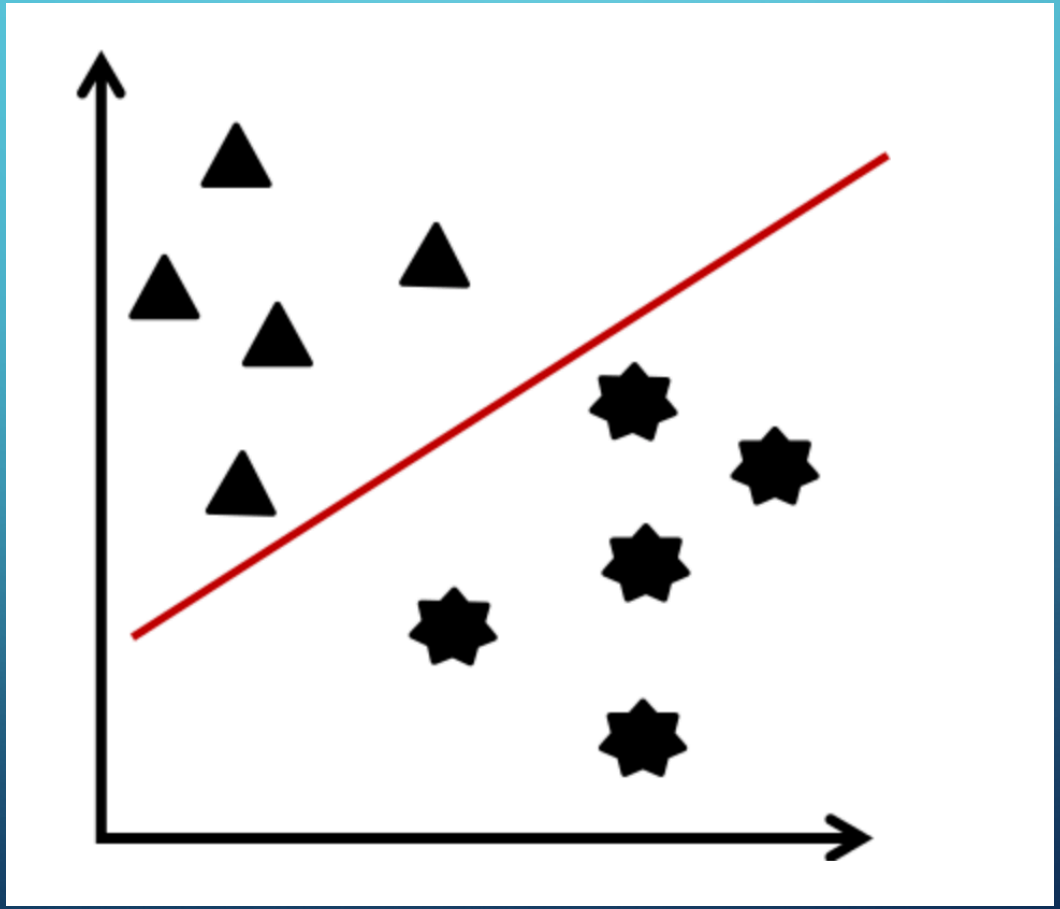
APLICAÇÕES DE RNA

- Previsão



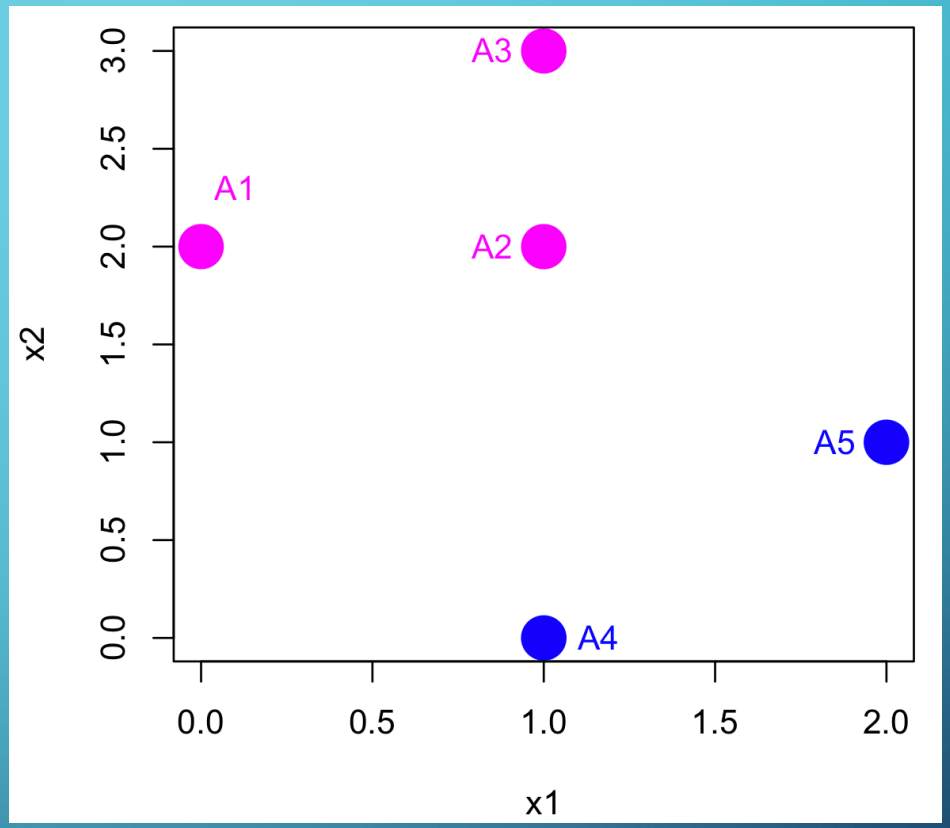
PERCEPTRON

Rede neural artificial mais simples que consegue resolver qualquer problema de classificação de dados linearmente separáveis.



EXEMPLO 1: RECONHECIMENTO DE PADRÕES

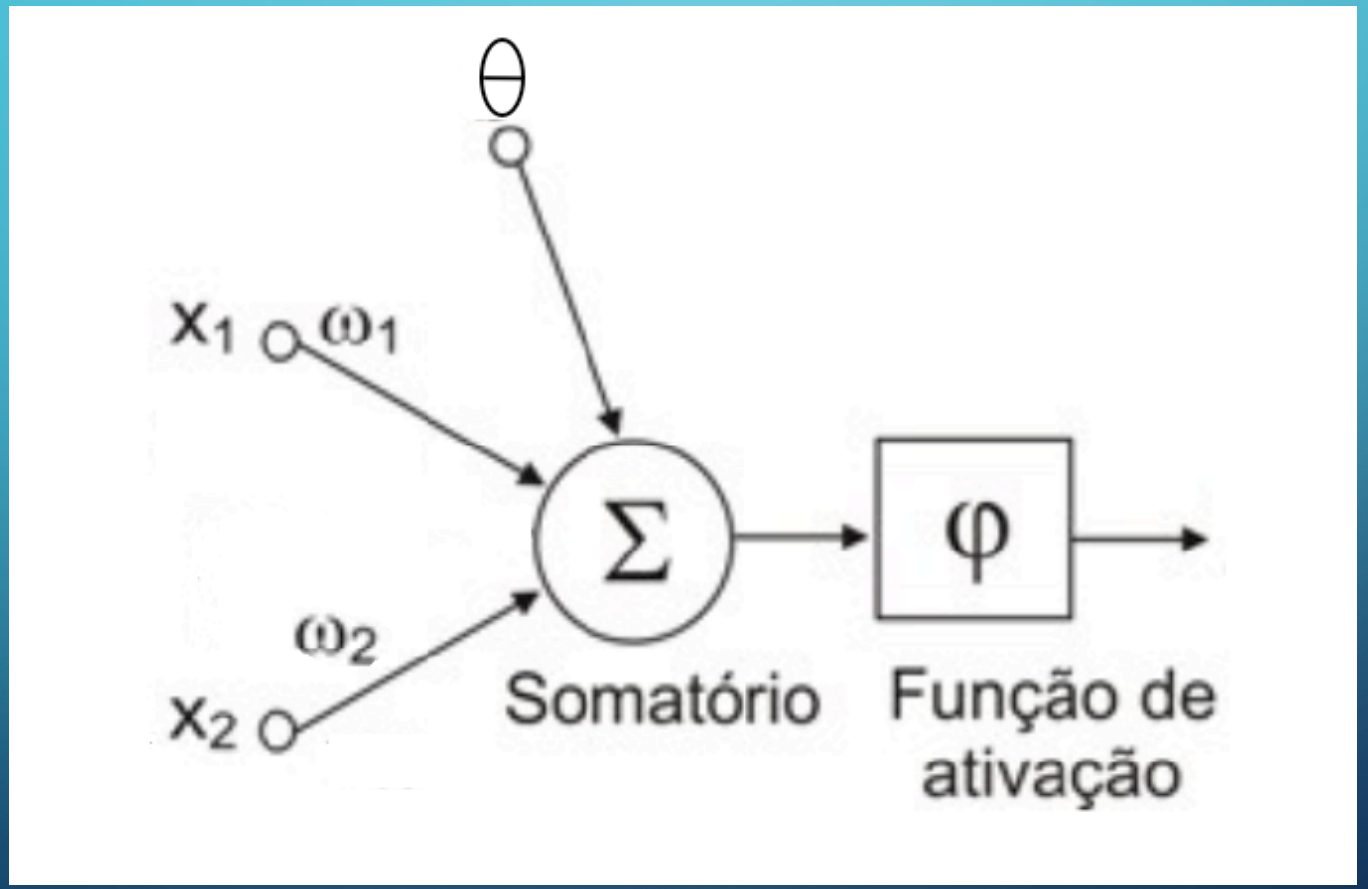
PONTOS	RESPOSTA DESEJADA
A1=(0,2)	1
A2=(1,2)	1
A3=(1,3)	1
A4=(1,0)	0
A5=(2,1)	0



Podemos perceber que existe um "padrão de comportamento" nos pontos apresentados. Podemos encontrar uma reta que separa os dois padrões. Usaremos a rede **Perceptron** que é usada para problemas linearmente separáveis.

PERCEPTRON

Para o exemplo:



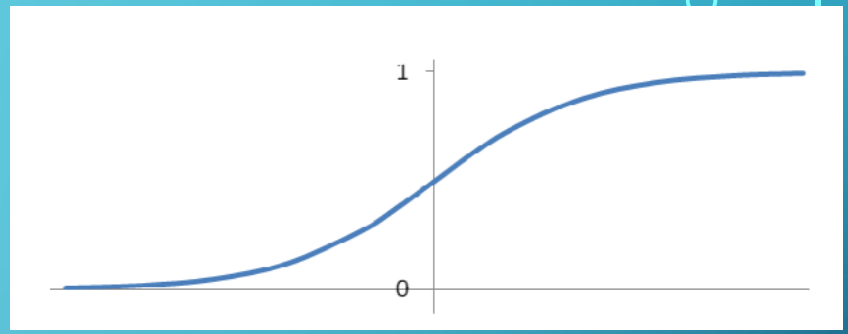
ALGORITMO PERCEPTRON

- PROPAGAÇÃO *FORWARD* (PARA FRENTE)

$$v^p = \sum_j w_j x_j^p + \theta$$

$$\varphi(v^p) = \frac{1}{1 + e^{-v^p}}$$

Função de ativação sigmoide logística



- PROPAGAÇÃO *BACKWARD* (PARA TRÁS)

$$\delta^p = d^p - \varphi(v^p)$$

- δ é o gradiente local e d^p a resposta desejada

$$\Delta w_j = \gamma \times x_j \times \delta^p$$

- γ é a taxa de aprendizagem, $0 < \gamma < 1$

$$w_j(novo) = w_j(velho) + \Delta w_j$$

O *bias* também é um peso e atualiza-se da mesma forma:

$$\Delta\theta = \gamma \times \textcircled{1} \times \delta^p$$

Como não se tem a resposta desejada, coloca-se o elemento neutro da multiplicação

$$\theta(novo) = \theta(velho) + \Delta\theta$$

ALGORITMO PERCEPTRON

- PROPAGAÇÃO *FORWARD* (PARA FRENTE)

$$v^p = \sum_j w_j x_j^p + \theta$$

$$\varphi(v^p) = \frac{1}{1 + e^{-v^p}}$$

- PROPAGAÇÃO *BACKWARD* (PARA TRÁS)

$$\delta^p = d^p - \varphi(v^p)$$

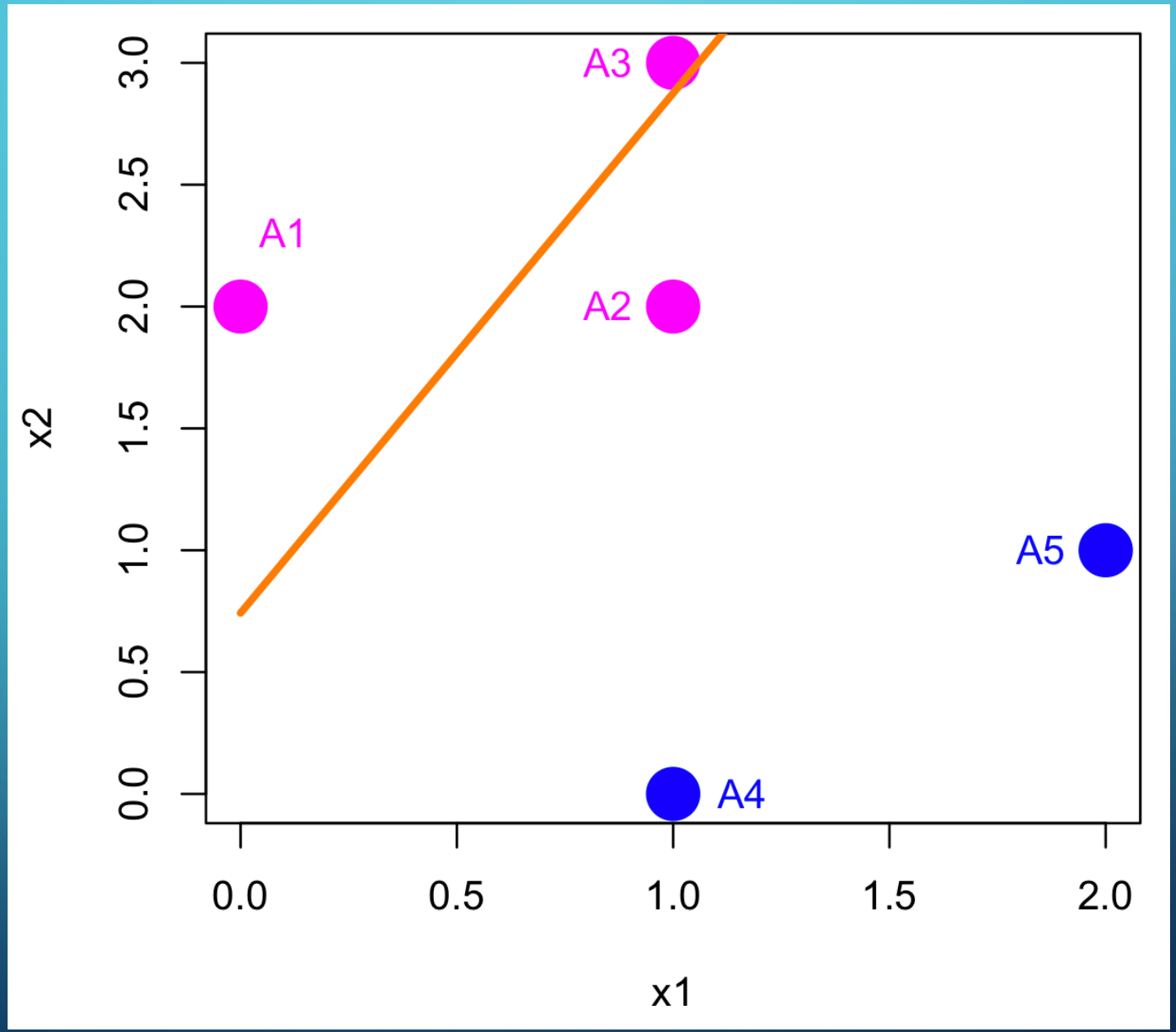
$$\Delta w_j = \gamma\, x_j \delta^p$$

$$w_j(novo) = w_j(velho) + \Delta w_j$$

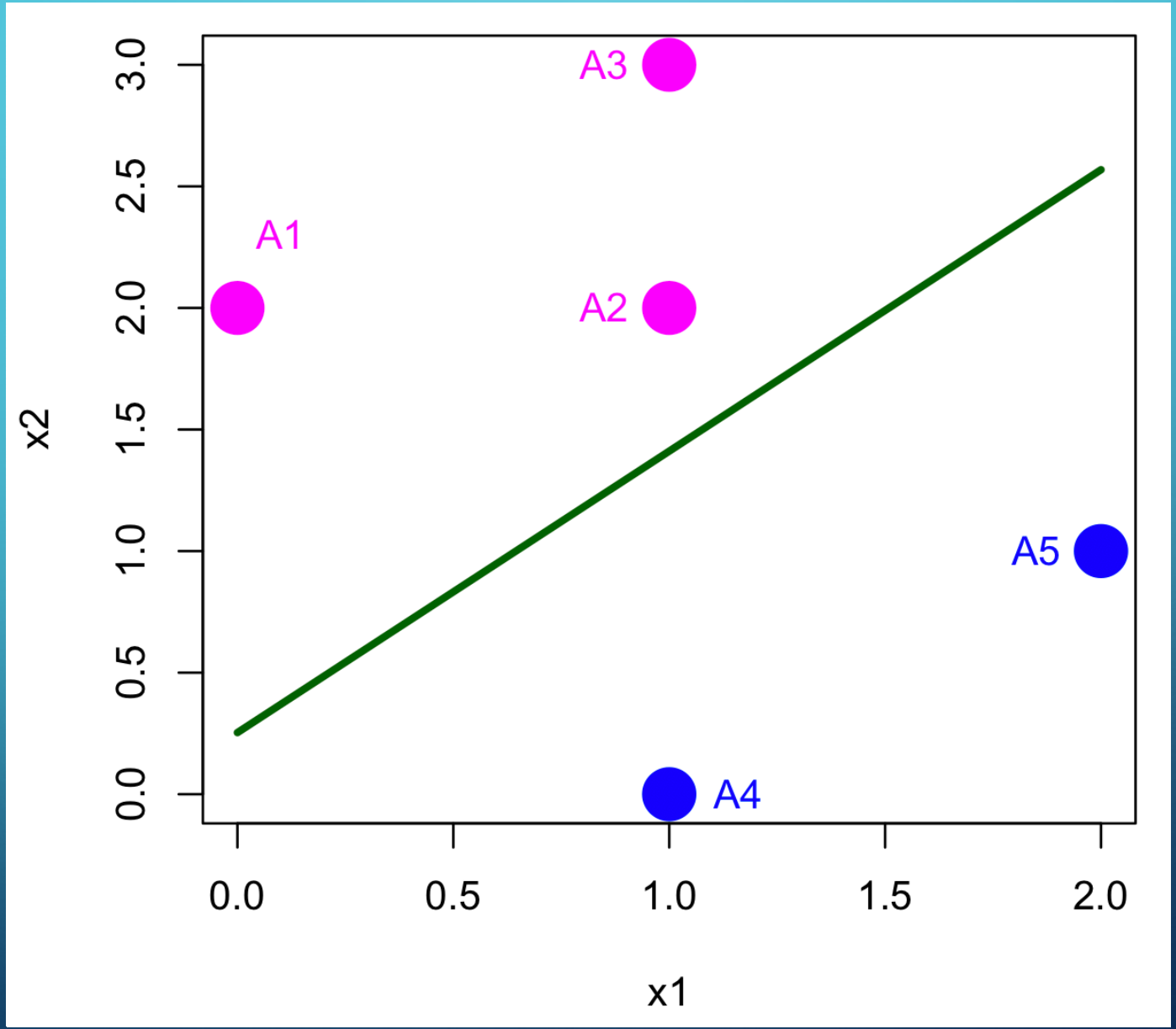
$$\Delta\theta = \gamma \delta^p$$

$$\theta(novo) = \theta(velho) + \Delta\theta$$

SEPARAÇÃO APÓS 1^a ITERAÇÃO



SEPARAÇÃO APÓS 2^a ITERAÇÃO

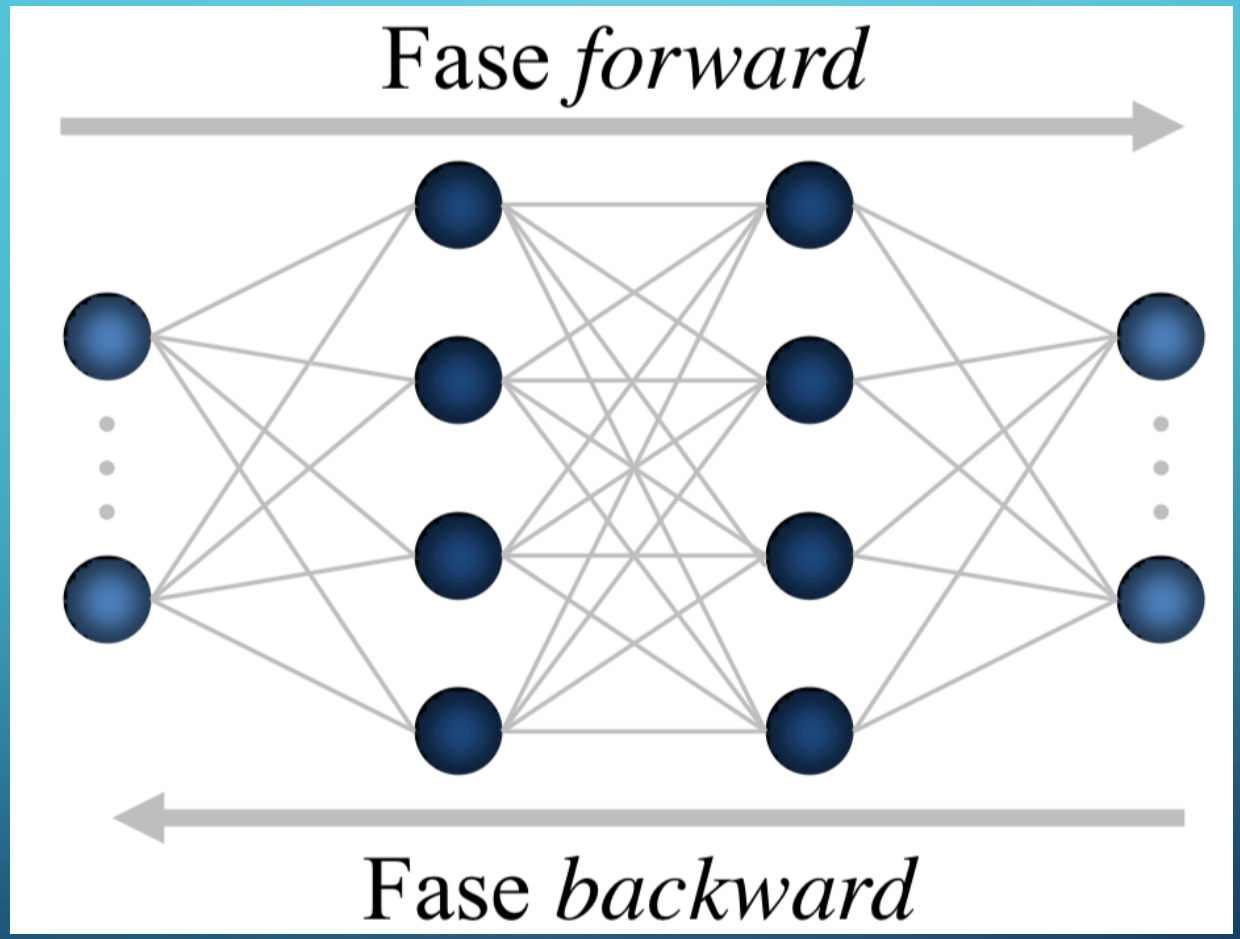


Com este primeiro exemplo, podemos entender alguns pontos:

- O aprendizado é **supervisionado**, isto é, as respostas são conhecidas pois a rede precisa aprender com os resultados conhecidos.
- O ***bias*** fornece maior liberdade na ponderação dos dados de entrada, além de uma maior capacidade de aproximação da rede. Também permite que um neurônio apresente uma saída não nula, caso todos os dados de entrada sejam nulos.
- O treinamento é realizado por meio do ***backpropagation***, que usa o erro entre a saída apresentada e a saída calculada para atualizar os pesos sinápticos e *bias*.

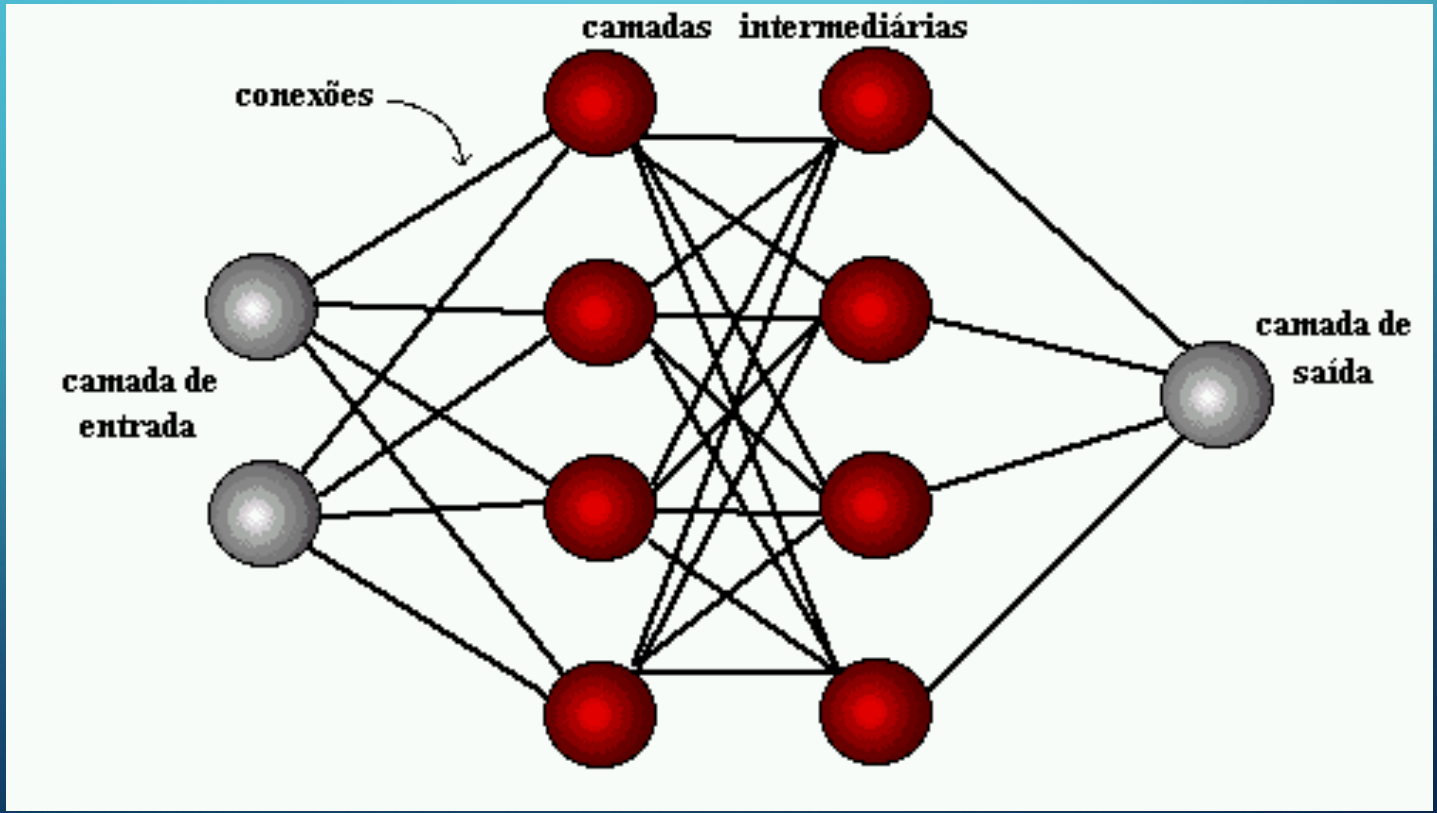
BACKPROPAGATION

O *backpropagation* é composto por duas fases:



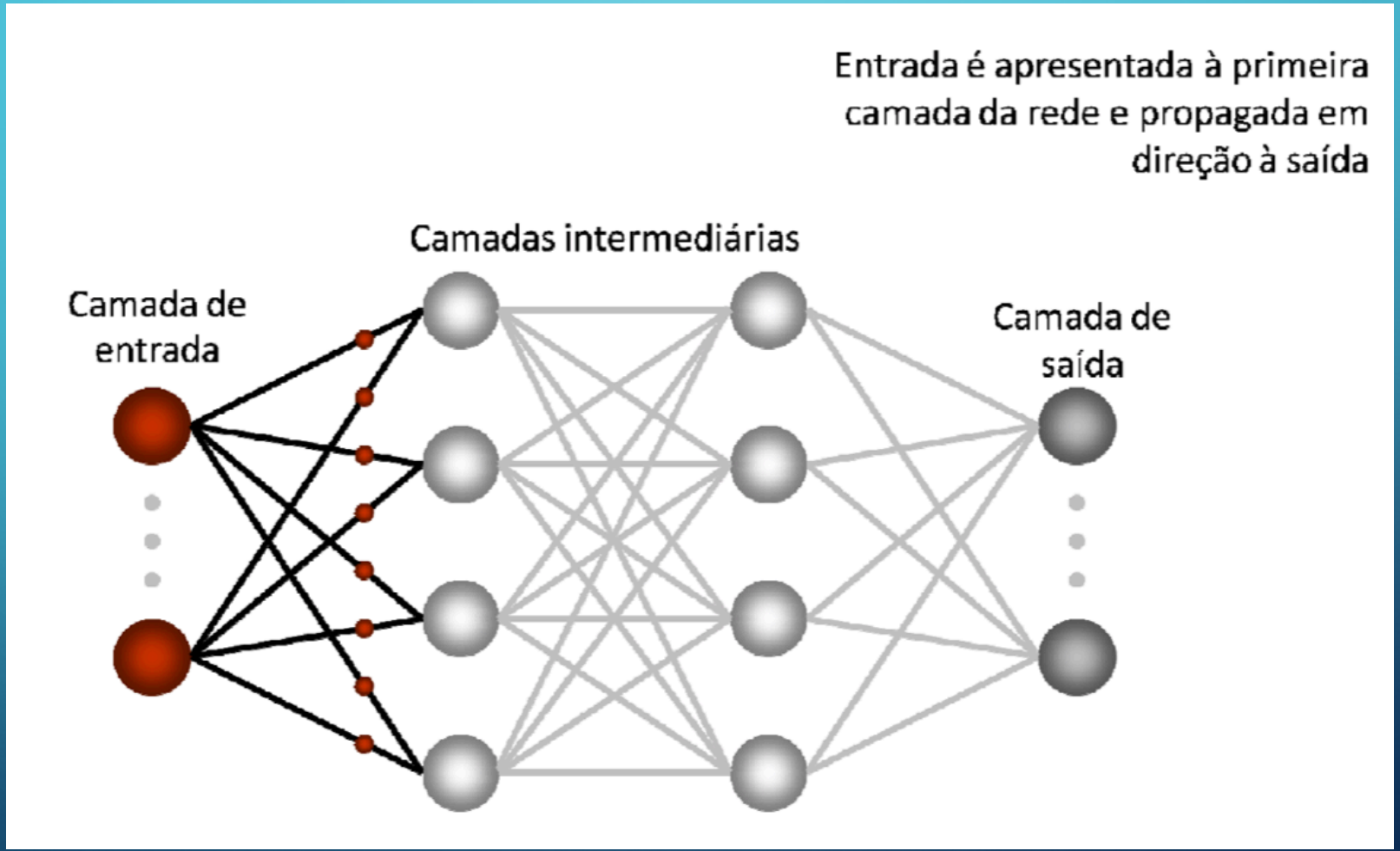
BACKPROPAGATION

- **FASE *FORWARD*** (alimentação para frente) – encarregada de apresentar as saídas calculadas pela rede (os pesos sinápticos e *bias* não são alterados).



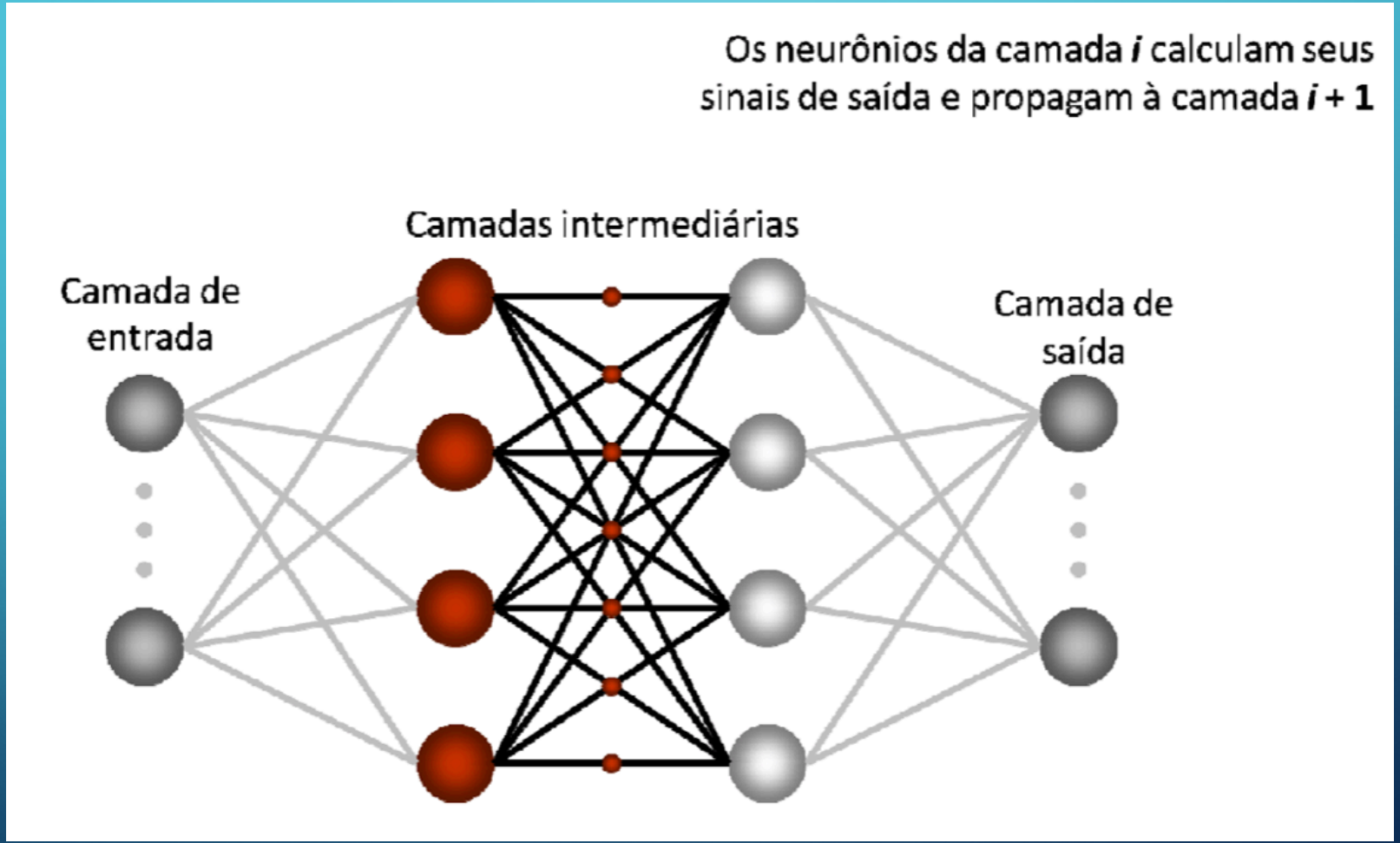
BACKPROPAGATION

- **FASE *FORWARD***



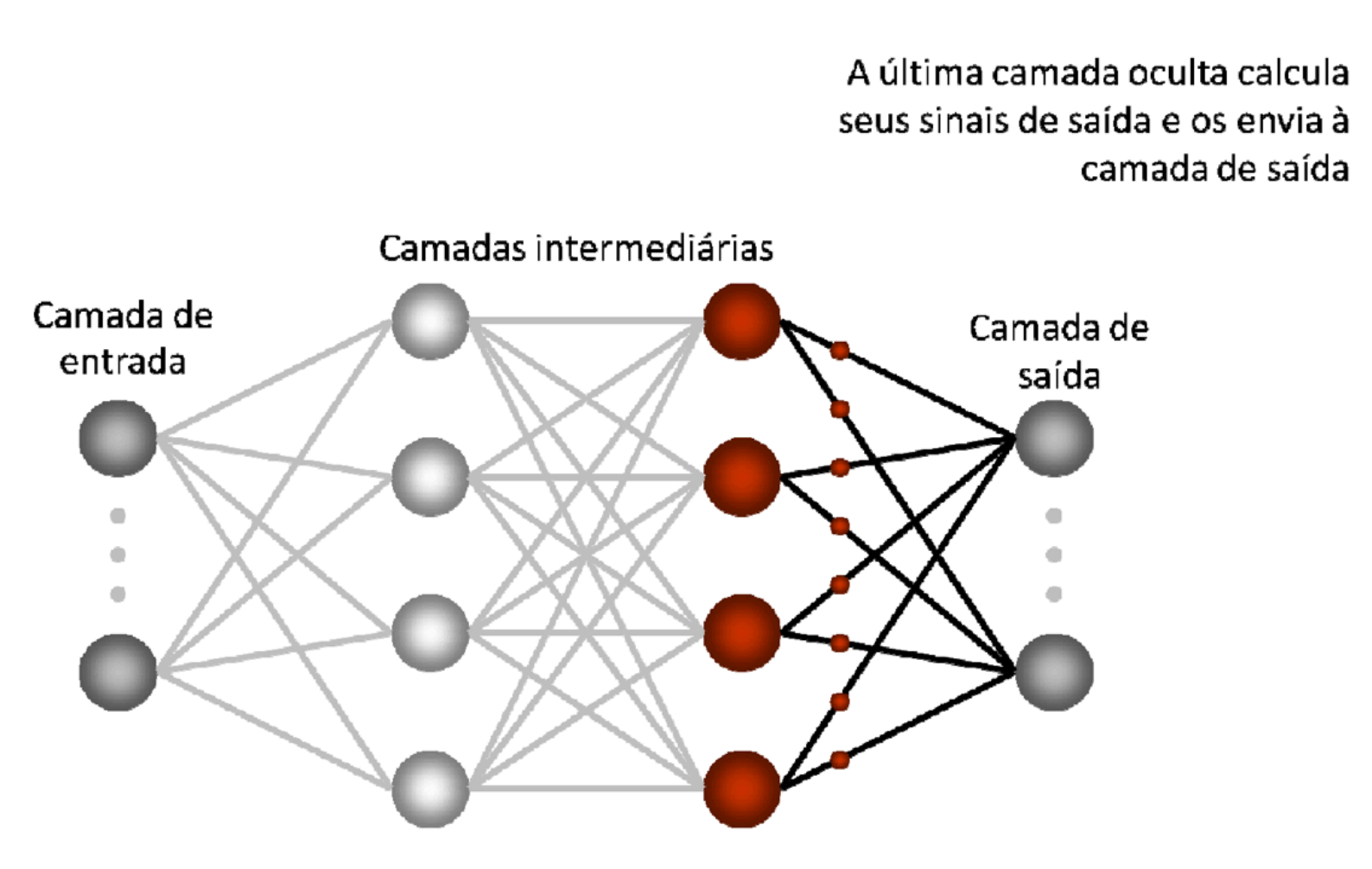
BACKPROPAGATION

- **FASE *FORWARD***



BACKPROPAGATION

- **FASE *FORWARD***

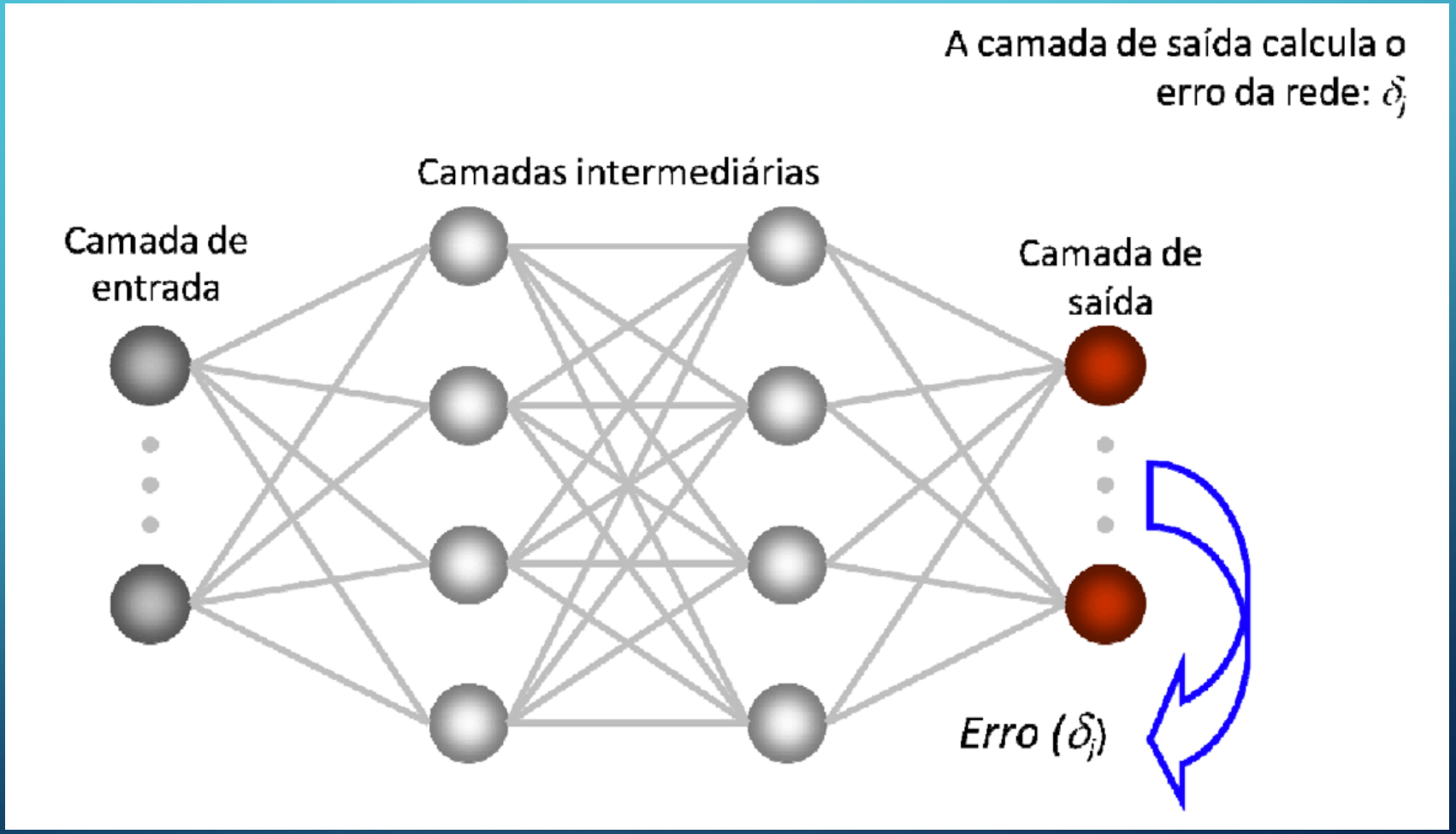


BACKPROPAGATION

- **FASE *BACKWARD*** (alimentação para trás) – responsável, após o cálculo do erro, pelas atualizações dos pesos sinápticos e *bias*, visando a diminuição do erro.

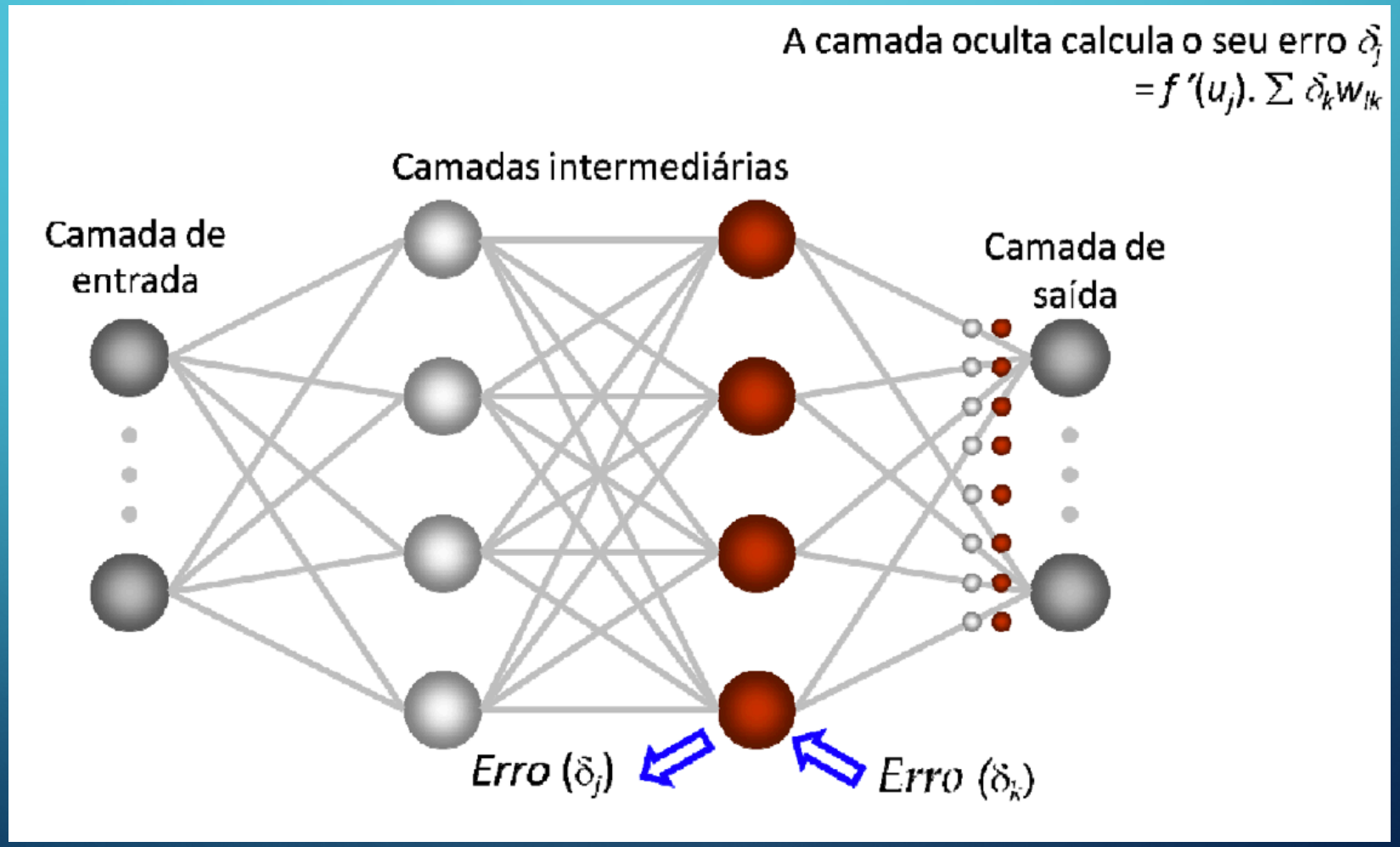
BACKPROPAGATION

- **FASE *BACKWARD***



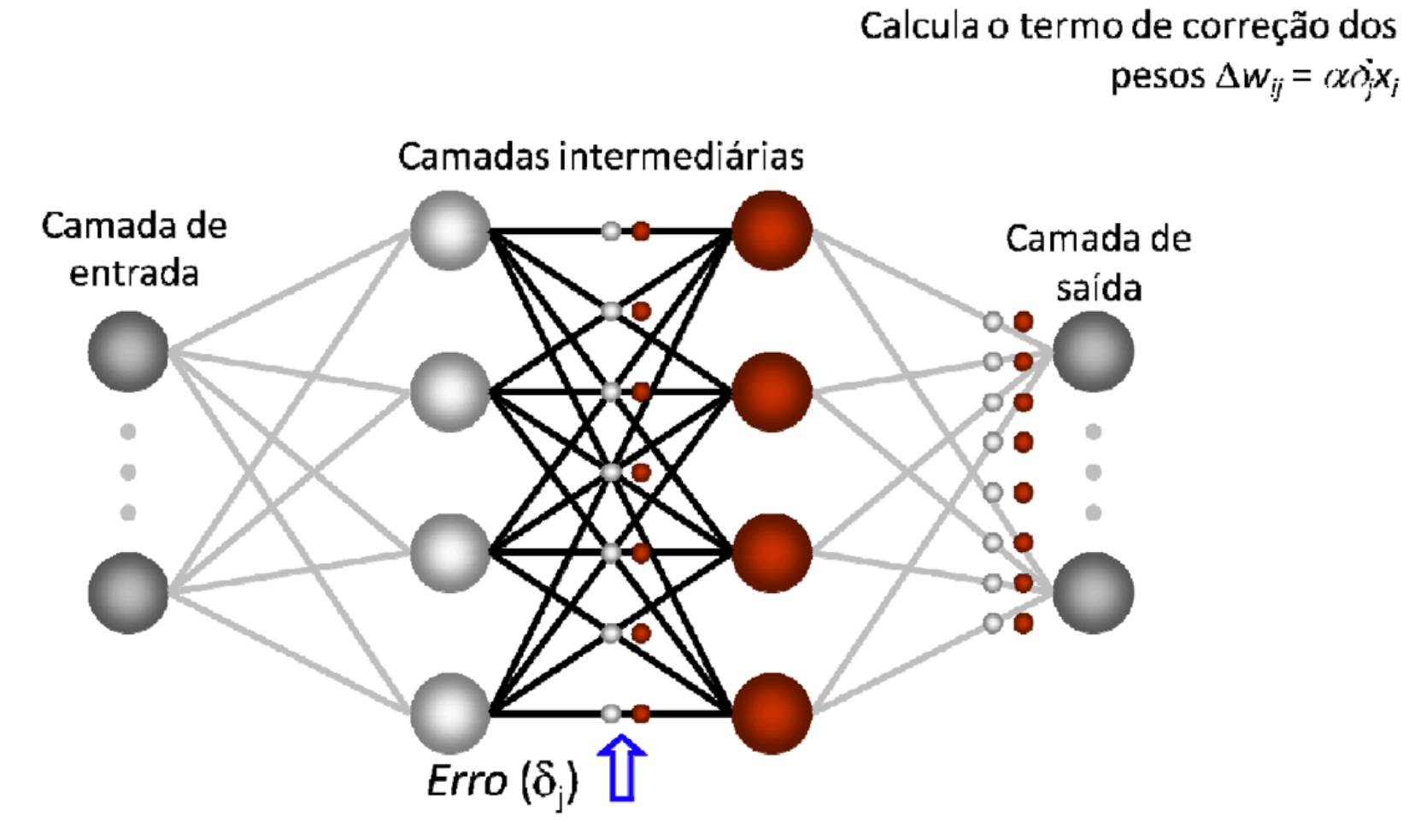
BACKPROPAGATION

- **FASE *BACKWARD***



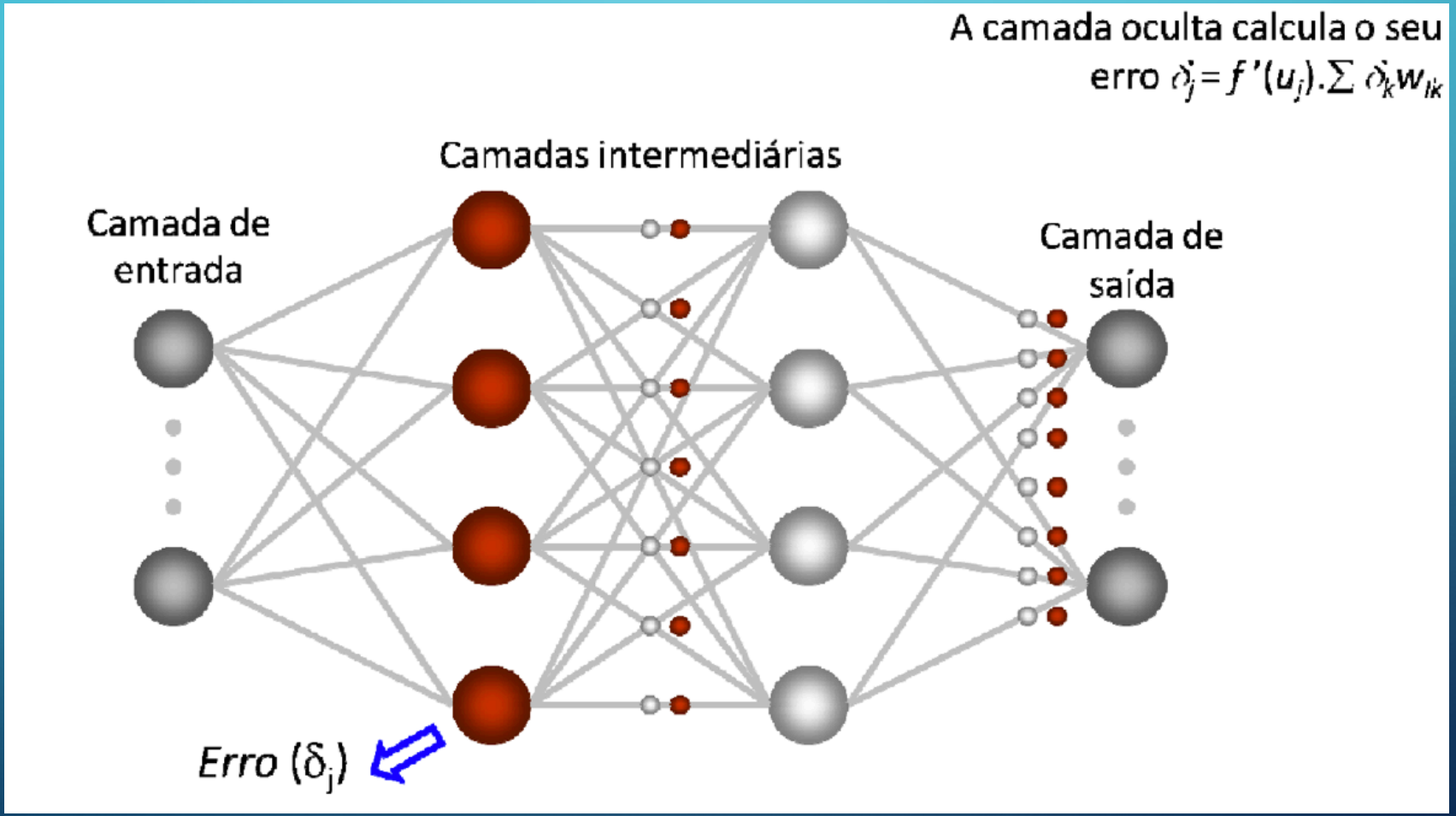
BACKPROPAGATION

- **FASE *BACKWARD***



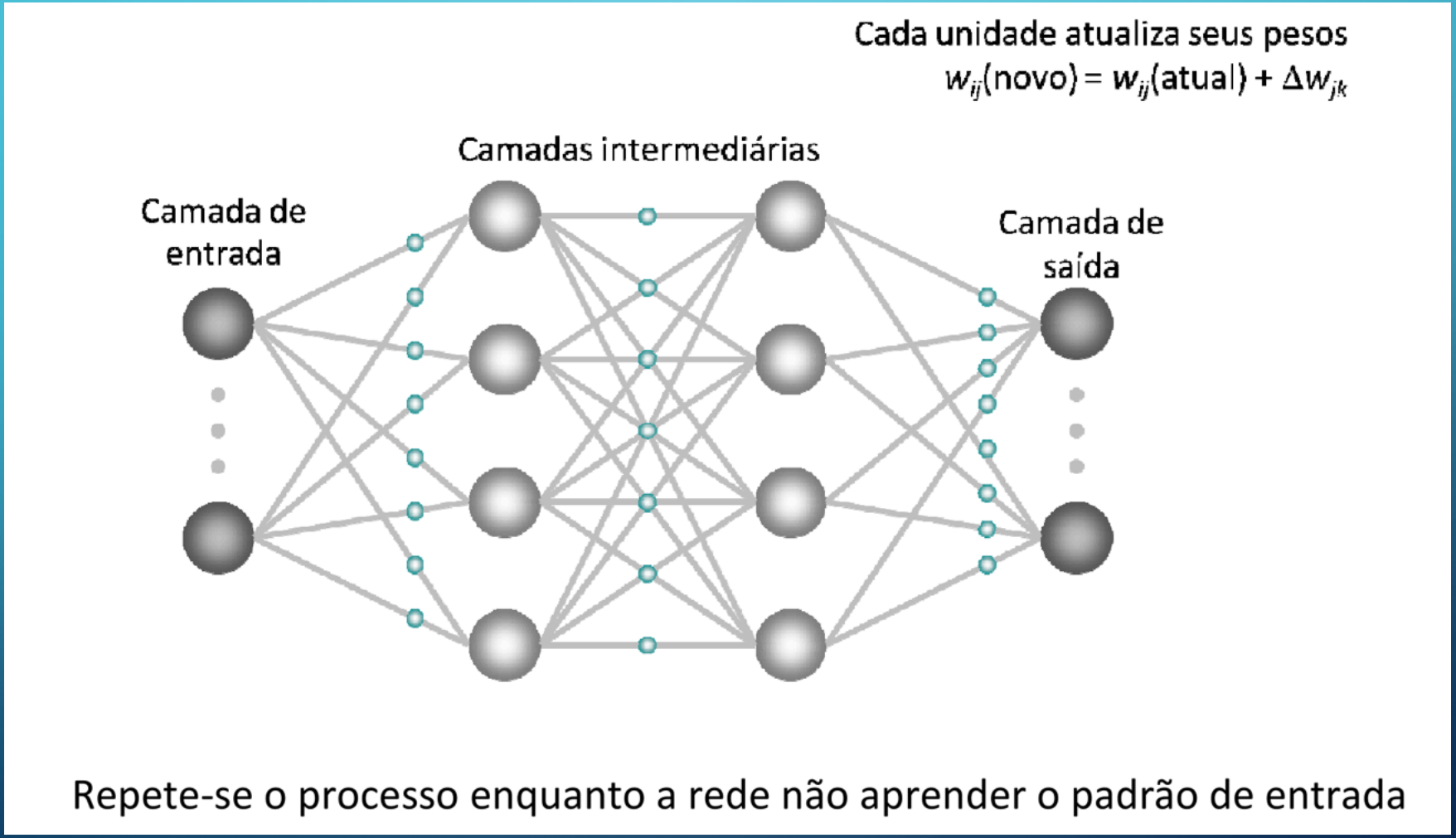
BACKPROPAGATION

- **FASE *BACKWARD***



BACKPROPAGATION

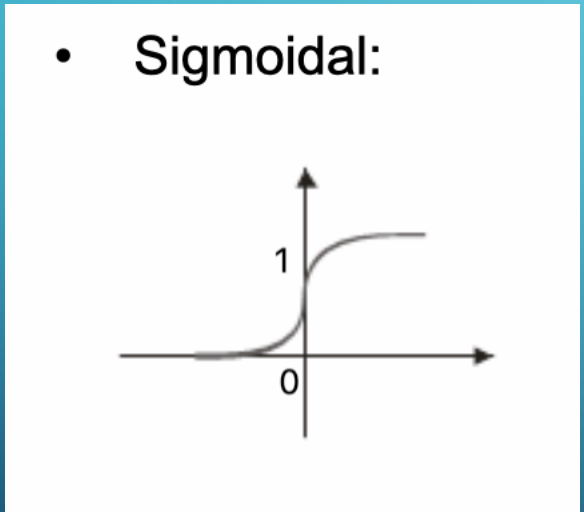
- **FASE *BACKWARD***



FUNÇÃO DE ATIVAÇÃO

Responsável por limitar a saída do neurônio, evitando acréscimos progressivos. As mais usadas são:

- **Sigmoide logística** – saída entre 0 e 1 (valores contínuos)

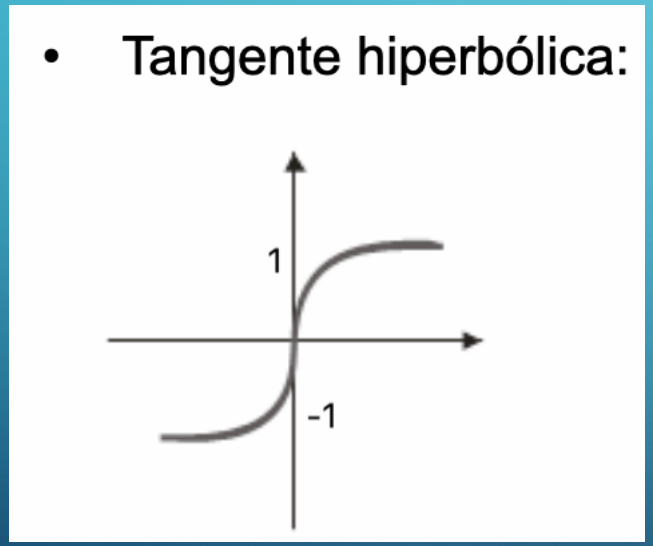


$$\varphi(v) = \frac{1}{1+e^{-v}}$$

FUNÇÃO DE ATIVAÇÃO

Responsável por limitar a saída do neurônio, evitando acréscimos progressivos. As mais usadas são:

- **Tangente hiperbólica** – saída entre -1 e 1 (valores contínuos)

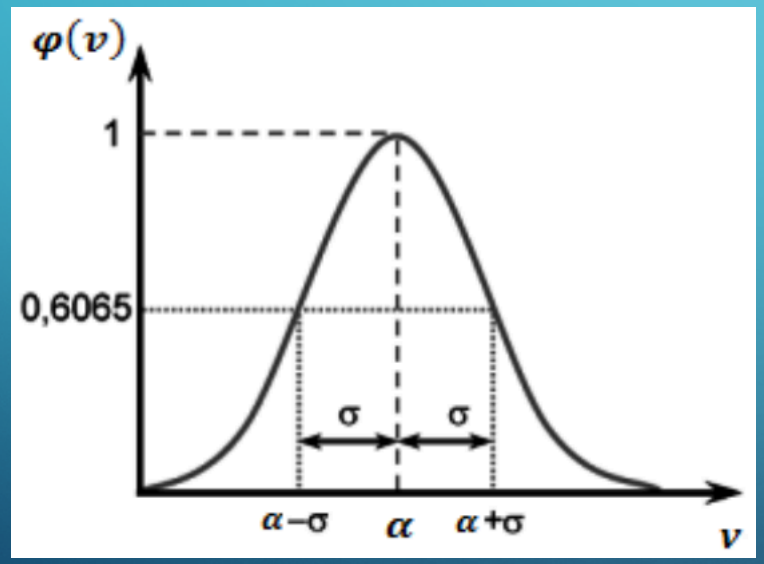


$$\varphi(v) = \tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}}$$

FUNÇÃO DE ATIVAÇÃO

Responsável por limitar a saída do neurônio, evitando acréscimos progressivos. As mais usadas são:

- **Gaussiana**– saída entre 0 e 1 (valores contínuos)



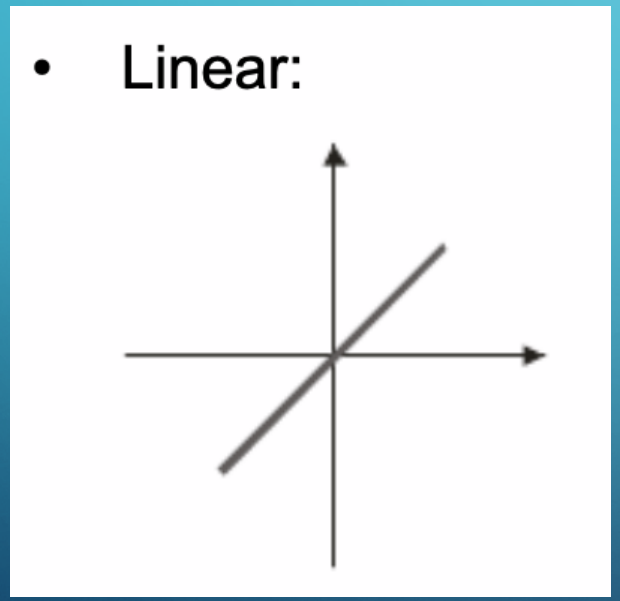
$$\varphi(v) = e^{\frac{-(v-\mu)^2}{2\sigma^2}}$$

μ é o centro da função de ativação e σ é o desvio padrão de v em relação a μ

FUNÇÃO DE ATIVAÇÃO

Responsável por limitar a saída do neurônio, evitando acréscimos progressivos. As mais usadas são:

- **Função linear** – qualquer saída



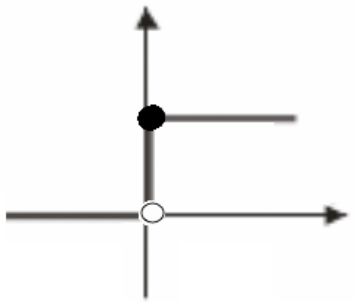
$$\varphi(v) = \alpha v, \qquad \alpha \in \mathbb{R}$$

FUNÇÃO DE ATIVAÇÃO

Responsável por limitar a saída do neurônio, evitando acréscimos progressivos. As mais usadas são:

- **Função degrau** – saída 0 ou 1

- Degrau (ou limiar):



$$\varphi(v) = \begin{cases} 1, se\ v \geq 0 \\ 0, se\ v < 0 \end{cases}$$

FUNÇÃO DE ATIVAÇÃO

Responsável por limitar a saída do neurônio, evitando acréscimos progressivos. As mais usadas são:

- **<u>Função degrau bilopar</u> (adaptação da função degrau)**

Saída 1, 0 ou -1

$$\varphi(v) = \begin{cases} 1, se\ v > 0 \\ 0, se\ v = 0 \\ -1, se\ v < 0 \end{cases}$$

Saída 1 ou -1

$$\varphi(v) = \begin{cases} 1, se\ v \geq 0 \\ -1, se\ v < 0 \end{cases}$$

TAXA DE APRENDIZAGEM (γ)

- Informa o quão rápido uma rede irá identificar e gravar os padrões de entrada.
- Uma taxa alta pode resultar em atualizações oscilatórias, podendo não obter padrões eficientes. Uma taxa pequena tornará o aprendizado mais estável, porém em um processo mais lento.
- O ideal é começar com um valor mais alto (por exemplo 0,8) e a cada iteração reduzir um pouco ($\gamma = 0{,}99\gamma$).
- Para garantir estabilidade e manter alta a taxa de aprendizagem, algumas vezes um termo chamado **momento** é incluído na atualização dos pesos:

$$\Delta w_j = \gamma x_j \delta + \boldsymbol{\alpha} \Delta w_j(anterior)$$

GRADIENTE LOCAL (δ)

- É usado para encontrar a direção em que o erro decresce mais rapidamente.
  - Para redes do tipo perceptron, o gradiente é tomado como sendo o próprio erro ($erro_j = d_j - \varphi(v_j)$).
  - Para redes com camadas ocultas (ou camadas intermediárias) – as Perceptron de Múltiplas Camadas (MLP) – o gradiente local pode ser calculado de duas maneiras, de acordo com a camada que o neurônio j está situado:

GRADIENTE LOCAL (δ)

- j pertence a camada de saída:

Para fins de minimização (facilitará os cálculos), o erro pode ser tomado como: $erro = \frac{1}{2}\left(d - \varphi(v)\right)^2$.

A ideia é minimizar esse erro seguindo na direção $-\frac{\partial erro}{\partial w}$.

$$\frac{\partial erro}{\partial w} = \left(d - \varphi(v)\right)\left(-\varphi'(v)\right)$$

Se $\varphi(v)$ é a sigmoide logística, $\varphi(v) = 1/(1 + e^{-v})$, então:

GRADIENTE LOCAL (δ)

$$\varphi'(v) = \frac{e^{-v}}{(1+e^{-v})^2} = \frac{1}{1+e^{-v}} \times \frac{e^{-v}}{1+e^{-v}}$$

$$= \varphi(v) \times \left(\frac{1+e^{-v}}{1+e^{-v}} - \frac{1}{1+e^{-v}} \right)$$

$$= \varphi(v) \times \left(1 - \frac{1}{1+e^{-v}} \right) = \varphi(v) \times (1 - \varphi(v))$$

Assim $\frac{\partial erro}{\partial w} = \big(d - \varphi(v)\big) \times \big(-\varphi(v)\big(1 - \varphi(v)\big)\big)$ e

$$-\frac{\partial erro}{\partial w} = \big(d - \varphi(v)\big) \times \Big(\varphi(v)\big(1 - \varphi(v)\big)\Big) = \boldsymbol{\delta}$$

GRADIENTE LOCAL (δ)

- j pertence a uma camada oculta:

Usando novamente a função sigmoide logística e minimizando o erro, tem-se:

$$\delta_j = \varphi(v_j)(1 - \varphi(v_j))\delta_h w_{hj}$$

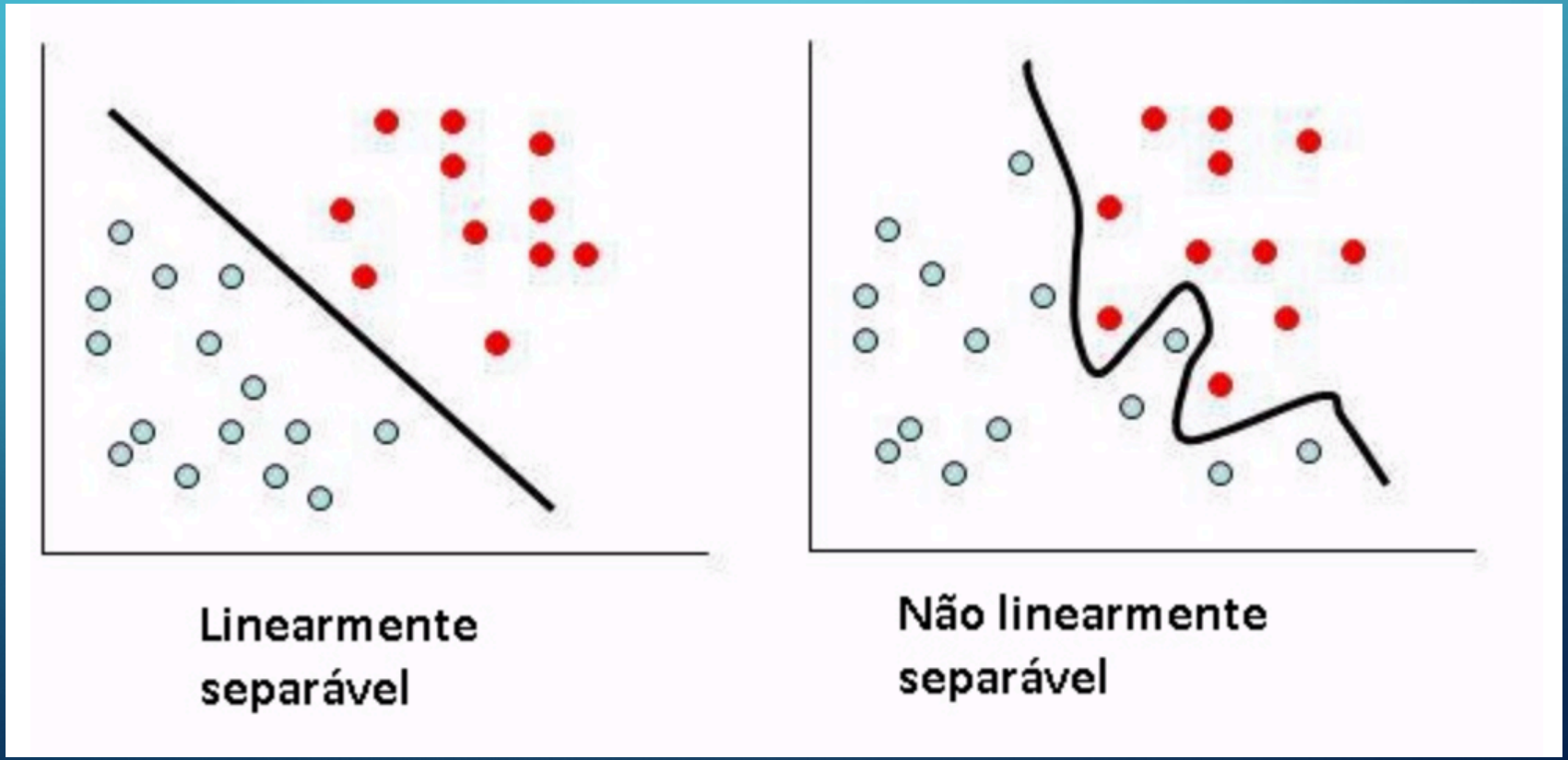
h é a camada consecutiva à camada do neurônio j.

Dedução em: SIQUEIRA [s.a.]

MULTI LAYER PERCEPTRON (MLP) (PERCEPTRON DE MÚLTIPLAS CAMADAS)

MLP

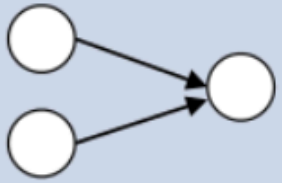
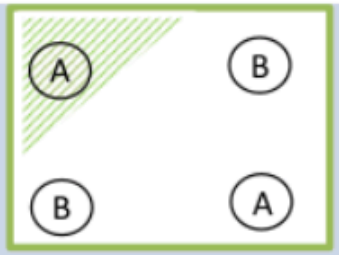
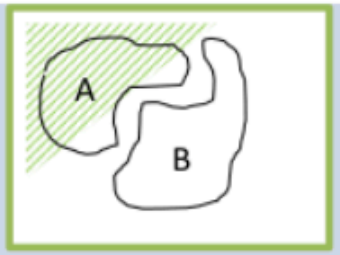
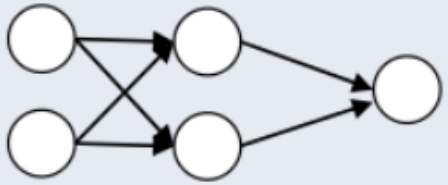
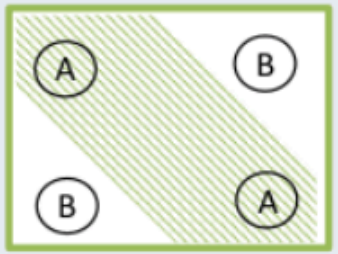
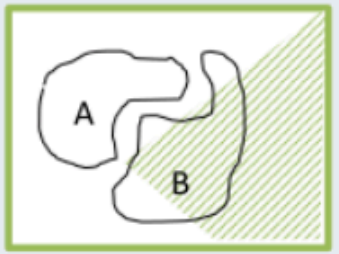
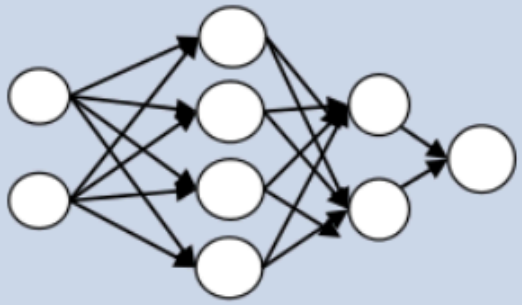
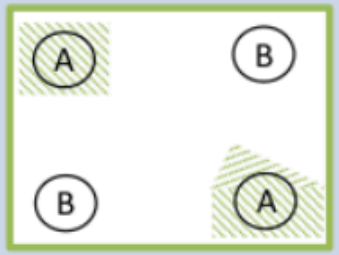
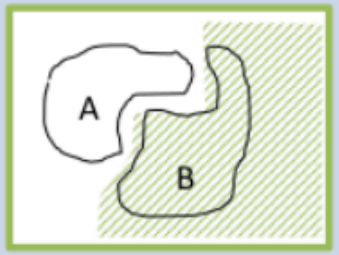
As redes do tipo perceptron só conseguem classificar corretamente todos os dados se estes são linearmente separáveis, isto é, uma reta separa corretamente os dados.



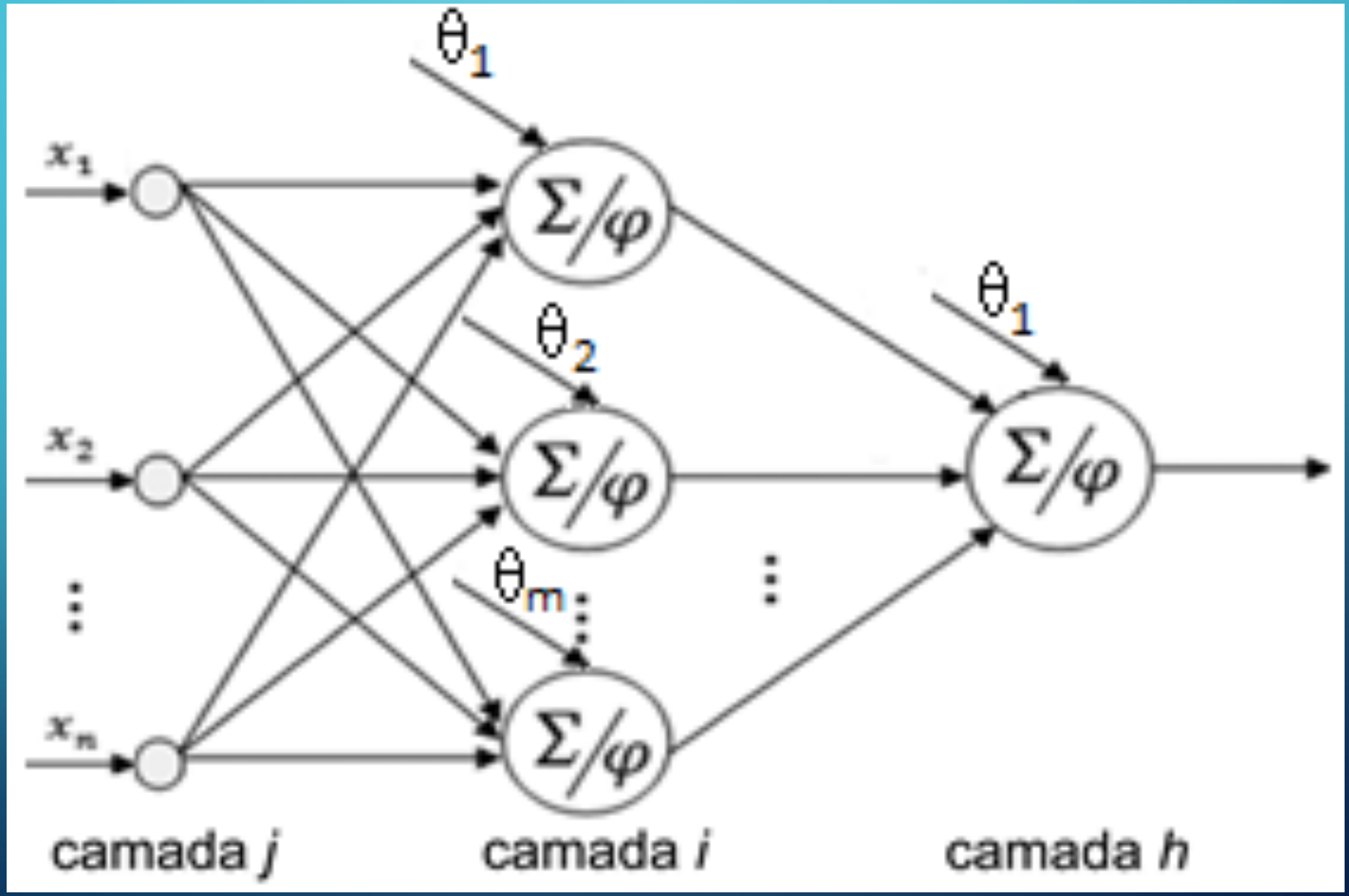
MLP

- Para resolver problemas não-lineares, acrescentam-se camadas (chamadas de camadas ocultas ou intermediárias) entre a camada de entrada e a camada de saída.
- Cada camada tem uma função específica:
  a) A camada de saída recebe os estímulos da camada oculta e constrói o padrão que será a resposta;
  b) As camadas ocultas funcionam como extratoras de características, seus pesos são uma codificação de características apresentadas nos padrões de entrada e permitem que a rede crie sua própria representação, mais rica e complexa do problema.

Como escolher o número de camadas de uma rede?

Arquitetura	Região de Decisão	Exemplo 1: XOR	Exemplo 2: Classificação
• Sem camada oculta 	• Hiperplano		
• Uma camada oculta 	• Regiões polinomiais convexas		
• Duas camada oculta 	• Regiões arbitrárias		

Representação de uma MLP com uma camada oculta



Passos do algoritmo MLP com função de ativação sigmoide logística (notação: $\varphi(v^p) = a^p$)

PROPAGAÇÃO *FORWARD*

$$v_i^{(p)} = \sum_j w_{ij}^{(p)} x_j^{(p)} + \theta_i$$

$$a_i^{(p)} = \frac{1}{1 + e^{-v_i^{(p)}}}$$

$$v_h^{(p)} = \sum_i w_{hi}^{(p)} a_i^{(p)} + \theta_h$$

$$a_h^{(p)} = \frac{1}{1 + e^{-v_h^{(p)}}}$$

PROPAGAÇÃO *BACKWARD*

$$\delta_h^{(p)} = \left(d^{(p)} - a_h^{(p)}\right) a_h^{(p)} (1 - a_h^{(p)})$$

$$\Delta w_{hi}^{(p)}(k) = \gamma a_i^{(p)} \delta_h^{(p)} + \alpha \Delta w_{hi}^{(p)}(k-1)$$

$$\Delta \theta_h^{(p)}(k) = \gamma \delta_h^{(p)} + \alpha \Delta \theta_h^{(p)}(k-1)$$

$$w_{hi}^{(p)}(k) = w_{hi}^{(p)}(k-1) + \Delta w_{hi}^{(p)}(k)$$

$$\theta_h^{(p)}(k) = \theta_h^{(p)}(k-1) + \Delta \theta_h^{p}(k)$$

$$\delta_i^{(p)} = a_i^{(p)} \left(1 - a_i^{(p)}\right) \delta_h^{(p)} w_{hi}^{(p)}$$

$$\Delta w_{ij}^{(p)}(k) = \gamma x_j^{(p)} \delta_i^{(p)} + \alpha \Delta w_{ij}^{(p)}(k-1)$$

$$\Delta \theta_i^{(p)}(k) = \gamma \delta_i^{(p)} + \alpha \Delta \theta_i^{(p)}(k-1)$$

$$w_{ij}^{(p)}(k) = w_{ij}^{(p)}(k-1) + \Delta w_{ij}^{(p)}(k)$$

$$\theta_i^{(p)}(k) = \theta_i^{(p)}(k-1) + \Delta \theta_i^{(p)}(k)$$

Observações:

- **Inicialização dos pesos**: geralmente os pesos são inicializados de forma aleatória, porém pode-se começar com valores vindos de outros algoritmos ou de testes empíricos.
- **Critério de parada**: a rede pode ser considerada treinada quando atingir um erro considerado satisfatório.
- Após todos os dados de entrada terem passado pela rede uma única vez, finaliza-se a 1^a iteração e então calcula-se o erro global dado por:

$$erro(k) = \frac{1}{N}\sum_{j=1}^{N}\left(d_j - a_j\right)^2$$

onde N representa o número de padrões de entrada para o treinamento.

Observações:

- Em geral, o $erro(k)$ representa o quanto as saídas calculadas para um determinado padrão de entrada difere da real resposta.
- O treinamento pode parar quando este erro for suficientemente pequeno, ou seja, $erro(k) \leq \varepsilon$, com ε estipulado pelo usuário.
- Outros critérios de parada que podem (devem) ser usados em conjunto com o critério do erro são: número máximo de iterações e tempo máximo de processamento.

Observações:

- **Separação de conjuntos**: geralmente separam-se os dados em dois conjuntos: treinamento e validação. O conjunto de treinamento consiste entre 70 e 90% do total de dados e é usado para treinar a rede (ajuste dos pesos). Já o conjunto de validação é usado para medir o desempenho da rede em um conjunto de dados nunca visto antes pela rede.

Observações:

- ***Overfitting* (super treinamento)**: a rede memoriza os padrões de treinamento ao invés de extrair as características gerais gravando suas peculiaridades e ruídos. Assim ela é super especializada nos dados de treinamento, mas não obtêm bons resultados para os dados de validação.

Observações:

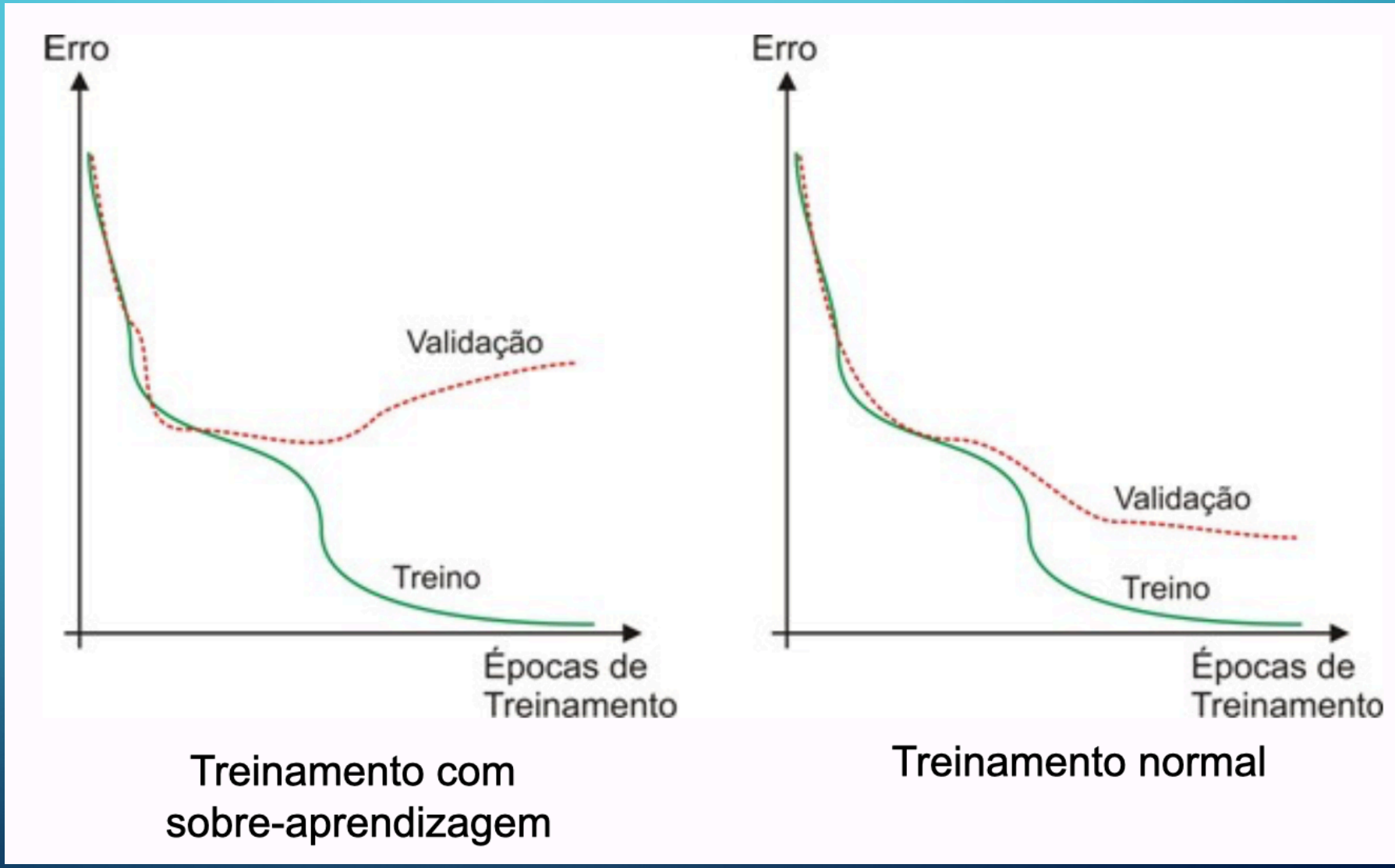
- ***Underfitting* (pouco treinamento)**: é quando a complexidade do problema supera a complexidade da rede e esta não é capaz de descrever e representar o domínio do problema. Geralmente acontece quando a arquitetura da rede está mal definida. Recomenda-se aumentar o número de neurônios na camada oculta.

Observações:

- **Validação cruzada**: é usada para evitar o *overfitting*. Consiste em, com certa frequência (pode ser a cada iteração ou 5 a 10 iterações), verificar o erro para o conjunto de validação. E quando este erro aumentar, para-se o treinamento.

Observações:

Curvas de erro de treinamento



Observações:

- **Embaralhar:** a cada iteração é recomendável fazer um sorteio na sequência de apresentação dos padrões de entrada durante o treinamento.
- **Momento**: é uma estratégia usada para evitar mínimos locais na curva de erro pois essa curva pode conter montanhas e vales e a RNA pode ficar presa em um ponto de mínimo local.

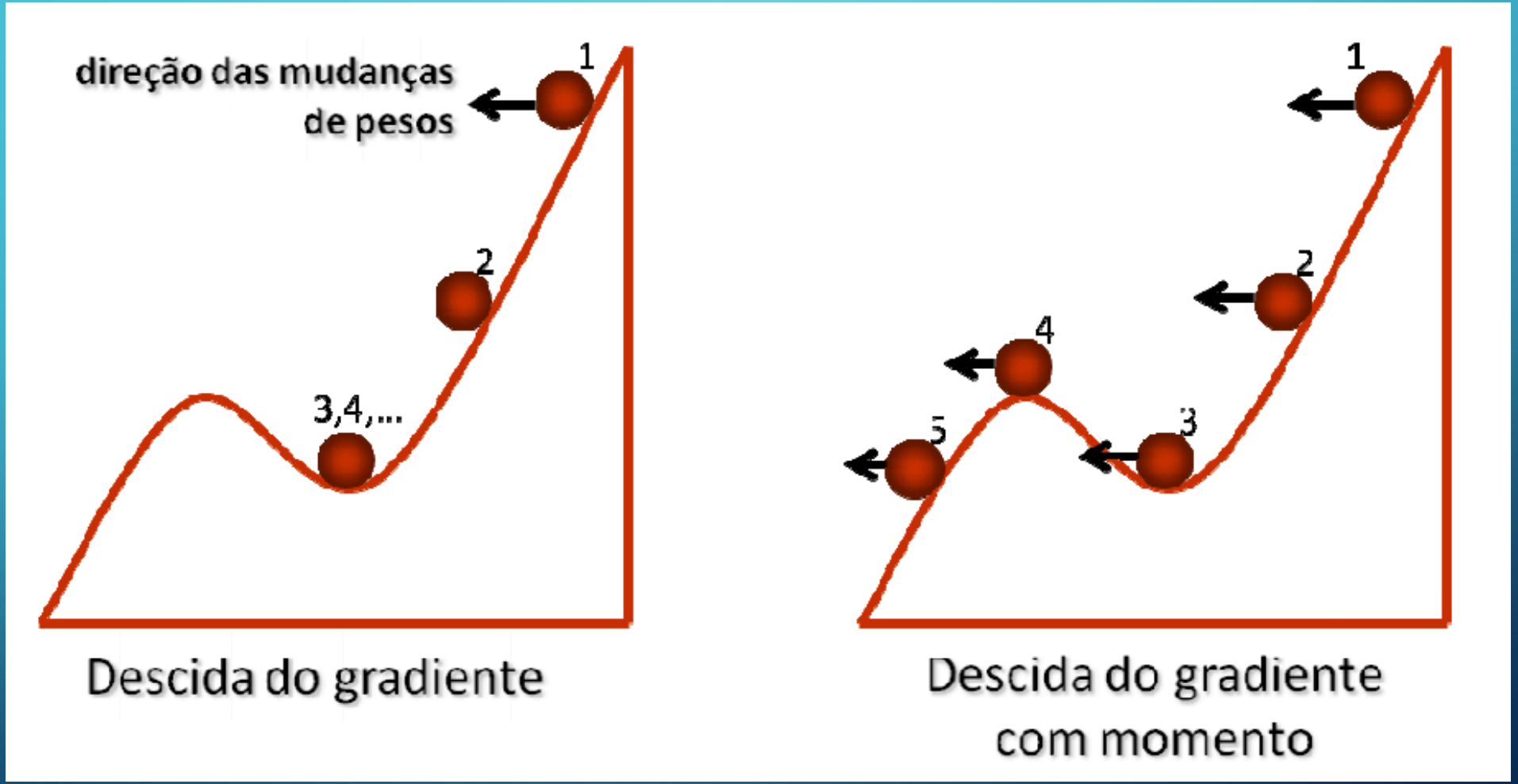
Observações:

- A aprendizagem com momento usa uma memória (incremento anterior) para aumentar a velocidade e estabilizar a convergência.
- Consiste em aplicar uma constante α sobre a mudança no vetor de pesos da última iteração, somando-se a esta a correção de direção da iteração atual.

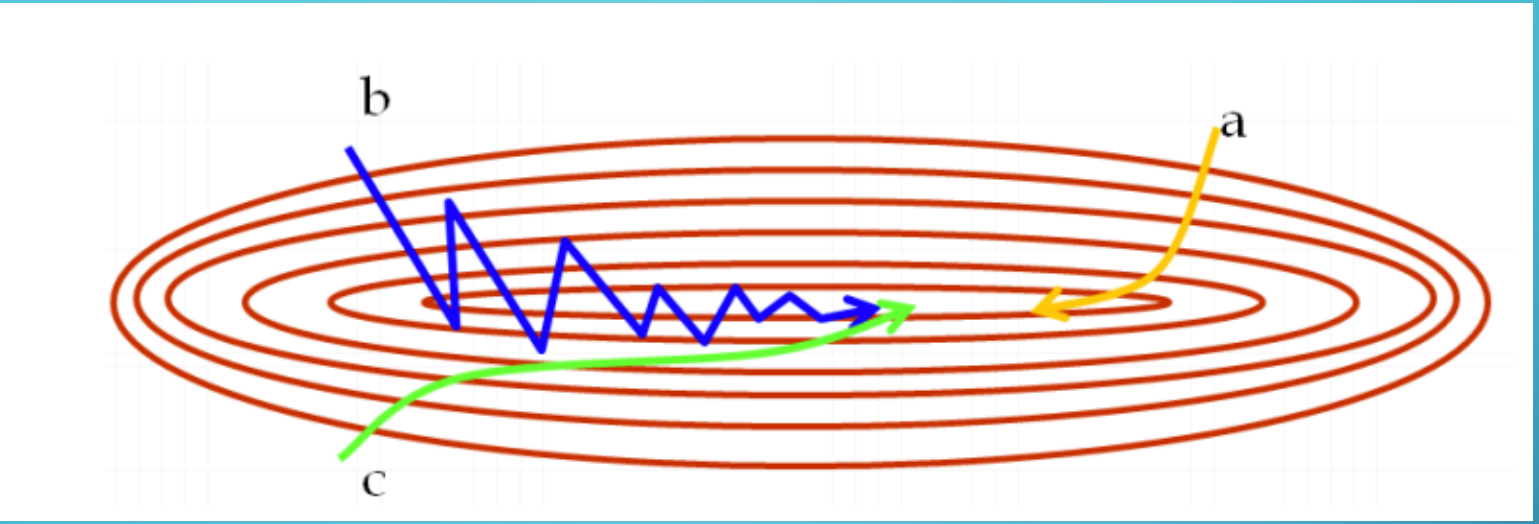
Observações:

- Se a última mudança de pesos foi em uma direção particular, o termo momento faz com que a próxima mudança de peso seja mais ou menos na mesma direção que a anterior.
- Dependendo da situação, este pequeno desvio pode ser suficiente para evitar que a rede caia em um mínimo local.
- Geralmente $0{,}5 \leq \alpha \leq 0{,}9$.

Observações:



Observações:



- Quando α é incluído e γ é pequena: demora para o mínimo ser alcançado (trajetória a).
- Quando α não é incluído e γ é alta: o mínimo nunca é alcançado porque ocorrem oscilações (trajetória b).
- Quando α é incluído e γ é alta: o mínimo é alcançado rapidamente (trajetória c).

Observações:

- **Valores numéricos**: uma rede neural só aceita valores numéricos, assim as entradas precisam ser codificadas.
  - Se um valor é contínuo (ex. temperatura, altura, peso, ...), então ele pode ser usado diretamente, porém recomenda-se uma normalização (transformar os dados para uma mesma escala).

  Possíveis normalizações:

$$x_{normalizado} = \frac{x - média(x)}{desv.p(x)}$$

$$x_{normalizado} = \frac{x - min(x)}{max(x) - min(x)}$$ valores ficam entre [0, 1].

Observações:

- Se um valor é categórico, então se utiliza uma codificação com valores binários.

Exemplo: $\begin{cases} Frio & = & 0 \\ Quente & = & 1 \end{cases}$

Exemplo: $\begin{cases} não & = & 0 \\ sim & = & 1 \end{cases}$

Uma rede que responde frio ou quente tem apenas uma saída com função de ativação sigmoide logística.

Observações:

Atributos com mais de dois valores: número de entradas igual ao número de valores.

Exemplo: $\begin{cases} 1\,0\,0 & = & Frio \\ 0\,1\,0 & = & Morno \\ 0\,0\,1 & = & Quente \end{cases}$

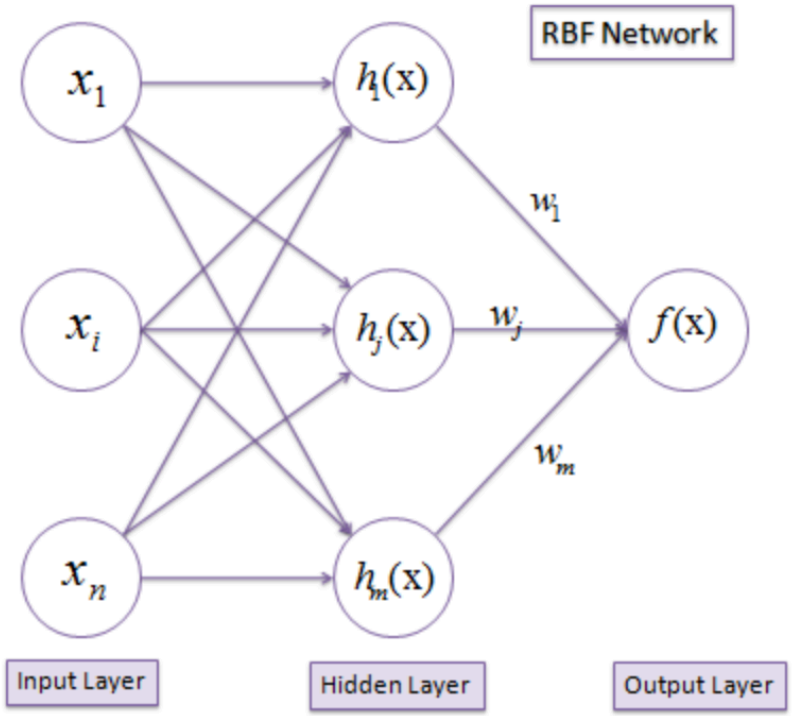
Uma rede que responde quente, morno ou frio possui três saídas com função de ativação logística.

OUTROS TIPOS DE REDES NEURAIS (não abordadas neste curso)

- Redes de Bases Radias (RBF)
- Redes recorrentes
- Redes de Kohonen
- Redes Neurais Convolucionais (https://www.youtube.com/watch?v=1_c_MA1F-vU)

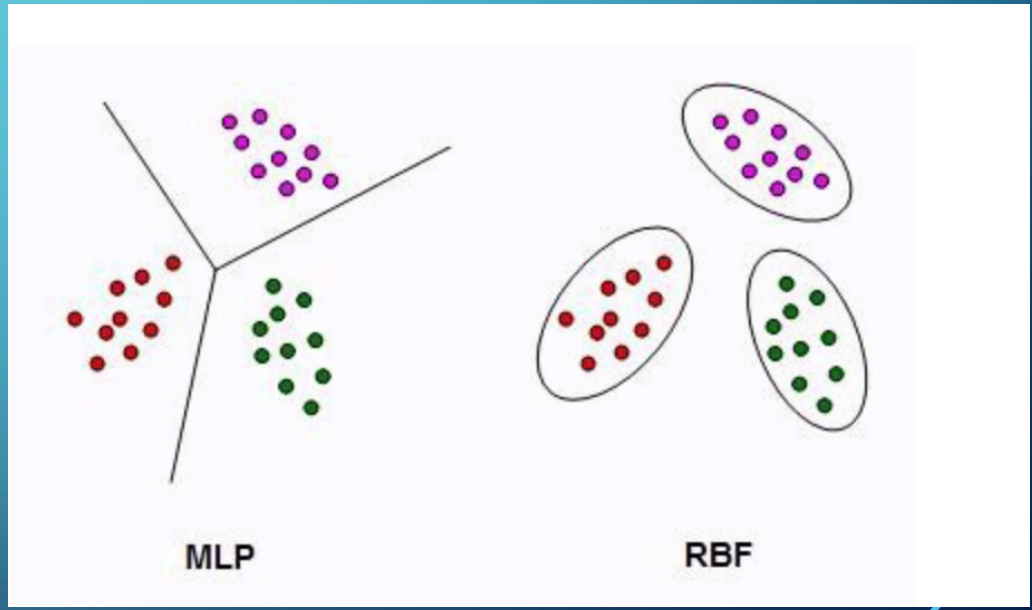
UMA BREVE IDEIA DE OUTRAS REDES NEURAIS

- Redes de Bases Radias (RBF)



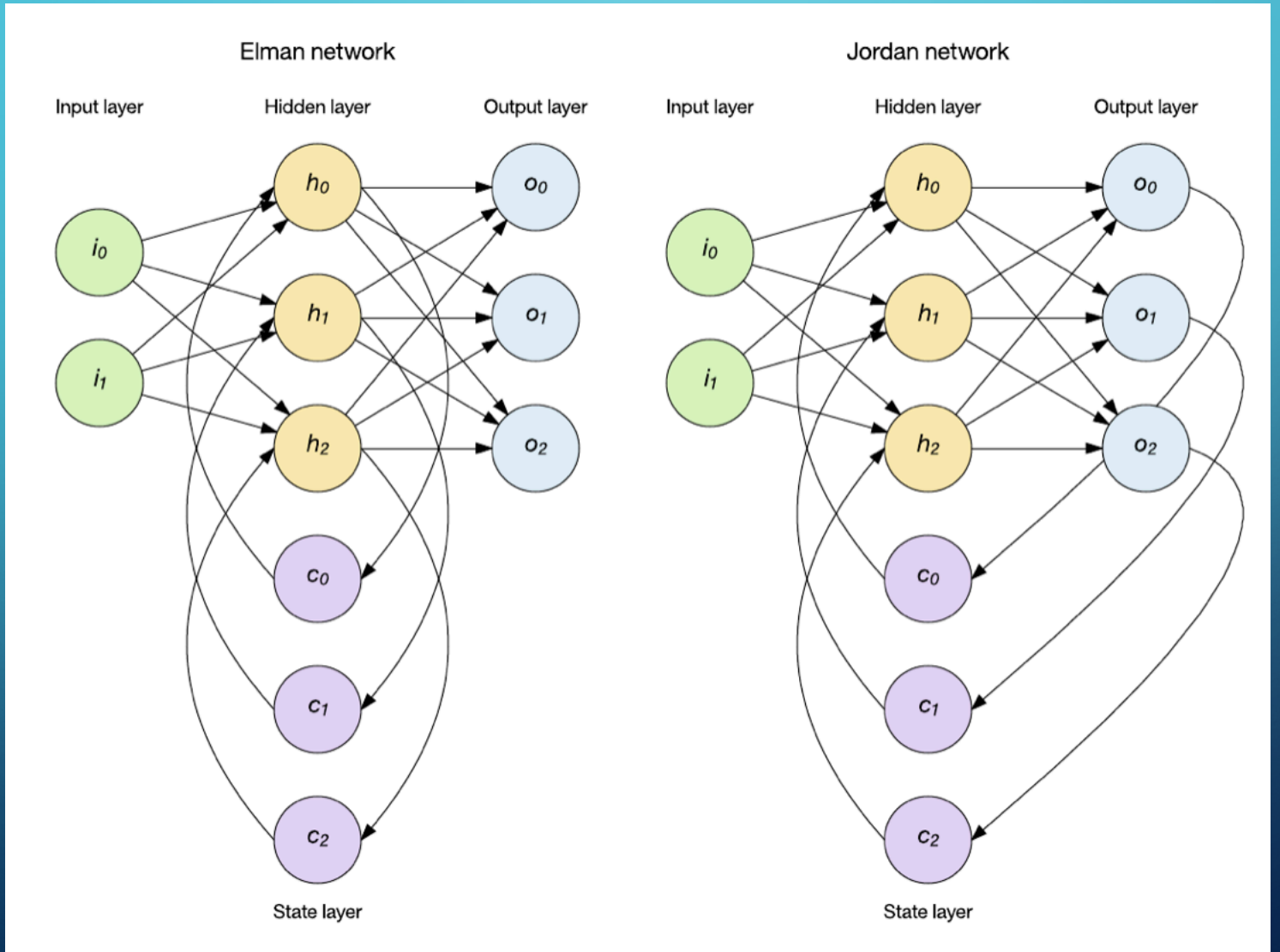
$$f(x) = \sum_{j=1}^{m} w_j h_j(x)$$

$$h(x) = \exp\left(-\frac{(x-c)^2}{r^2}\right)$$



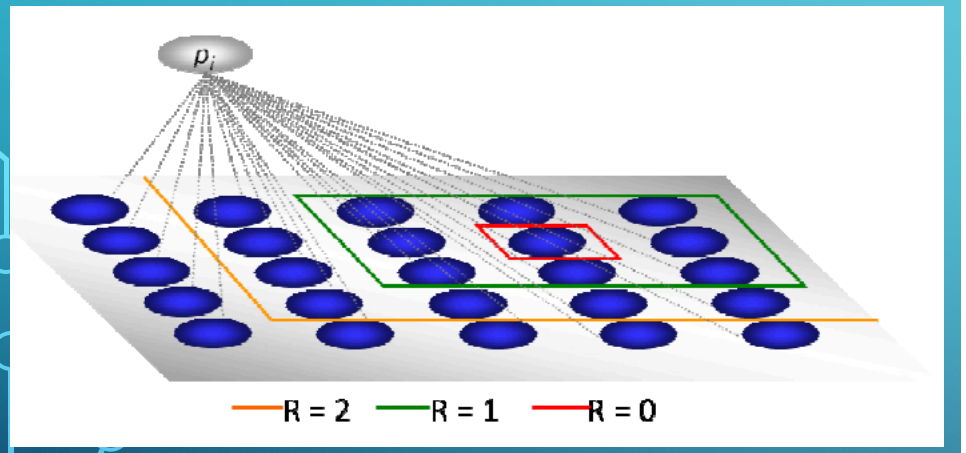
UMA BREVE IDEIA DE OUTRAS REDES NEURAIS

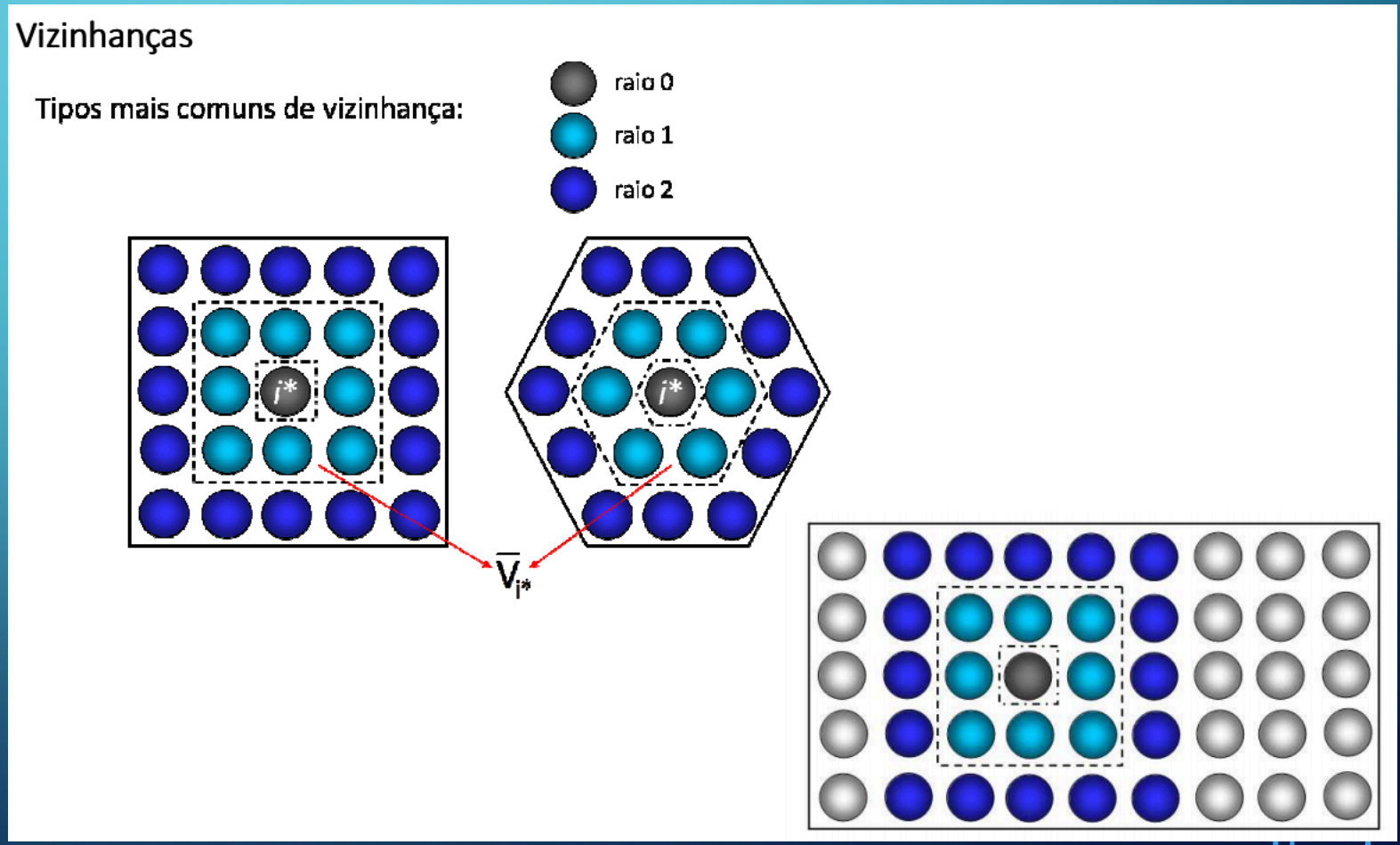
- Redes Recorrentes



UMA BREVE IDEIA DE OUTRAS REDES NEURAIS

- Redes de Kohonen (*Self Organized Maps* – SOM)





VÍDEO-AULAS RECOMENDADAS DE PERCEPTRON/MLP:

- Introdução: https://www.youtube.com/watch?v=9zv56aEdXxs
- Perceptron: https://www.youtube.com/watch?v=7szgyNs9po0
- Perceptron (aula prática): https://www.youtube.com/watch?v=Zflb__rg2As
- MLP: https://www.youtube.com/watch?v=XD5cCPAnQOM
- MLP (aula prática): https://www.youtube.com/watch?v=AcYHsKIIj5E

REFERÊNCIAS

- HAYKIN, S. Redes Neurais: princípios e prática. Bookman, 2003.
- SIQUEIRA, P. H. Metaheurísticas e Aplicações. PPGMNE, UFPR, [s.a.]
- MANICA, R. Aplicação de uma rede neural artificial simplificada para a identificação de gradação de depósitos turbiditíco. Geociências, vol. 32, p. 429-440, 2013.
- SIMAS, E. Processamento Estatístico de Sinais: Introdução às Redes Neurais Artificiais. Notas de aula, UFBA, 2013. Disponível em: https://slideplayer.com.br/slide/292589/1/images/5/Modelos+de+Neurônios+Modelo+de+um+neurônio+biológico
- VOLPI, A. Ferramentas de Python para aprendizado de máquina. Disponível em: https://alexandrevolpi.wordpress.com/2015/10/02/ferramentas-de-python-para-aprendizado-de-maquina/
- LEON, K. Making a Simple Neural Network : Classification, 2017. Disponível em: https://becominghuman.ai/making-a-simple-neural-network-classification-2449da88c77e
- LINARES, K. S. C. Notas de aulas – Sistemas Inteligentes, UTFPR. Disponível em: http://paginapessoal.utfpr.edu.br/kathya/Disciplinas/sistemas_inteligentes/sistemas-inteligentes/aula%2021%20SI.pdf
- USP. Redes Neurais Artificiais. Disponível em: http://conteudo.icmc.usp.br/pessoas/andre/research/neural/
- ROCHA, J. Introdução ao Perceptron passo a passo. Disponível em: https://juliocprocha.wordpress.com/2017/07/27/perceptron-para-classificacao-passo-a-passo/
- KREZANOVSKY, P. Redes Neurais Artificiais aplicadas a problemas de previsão e classificação. Trabalho de conclusão de curso, UFPR, 2016.
- CHANDRADEVAN, R. Radial Basis Functions Neural Networks—All we need to know, 2017. Disponível em: https://towardsdatascience.com/radial-basis-functions-neural-networks-all-we-need-to-know-9a88cc053448
- JONES, M. T. Um mergulho profundo nas redes neurais recorrentes. Disponível em: https://imasters.com.br/data/um-mergulho-profundo-nas-redes-neurais-recorrentes