



PODE TECH

FIAP + alura

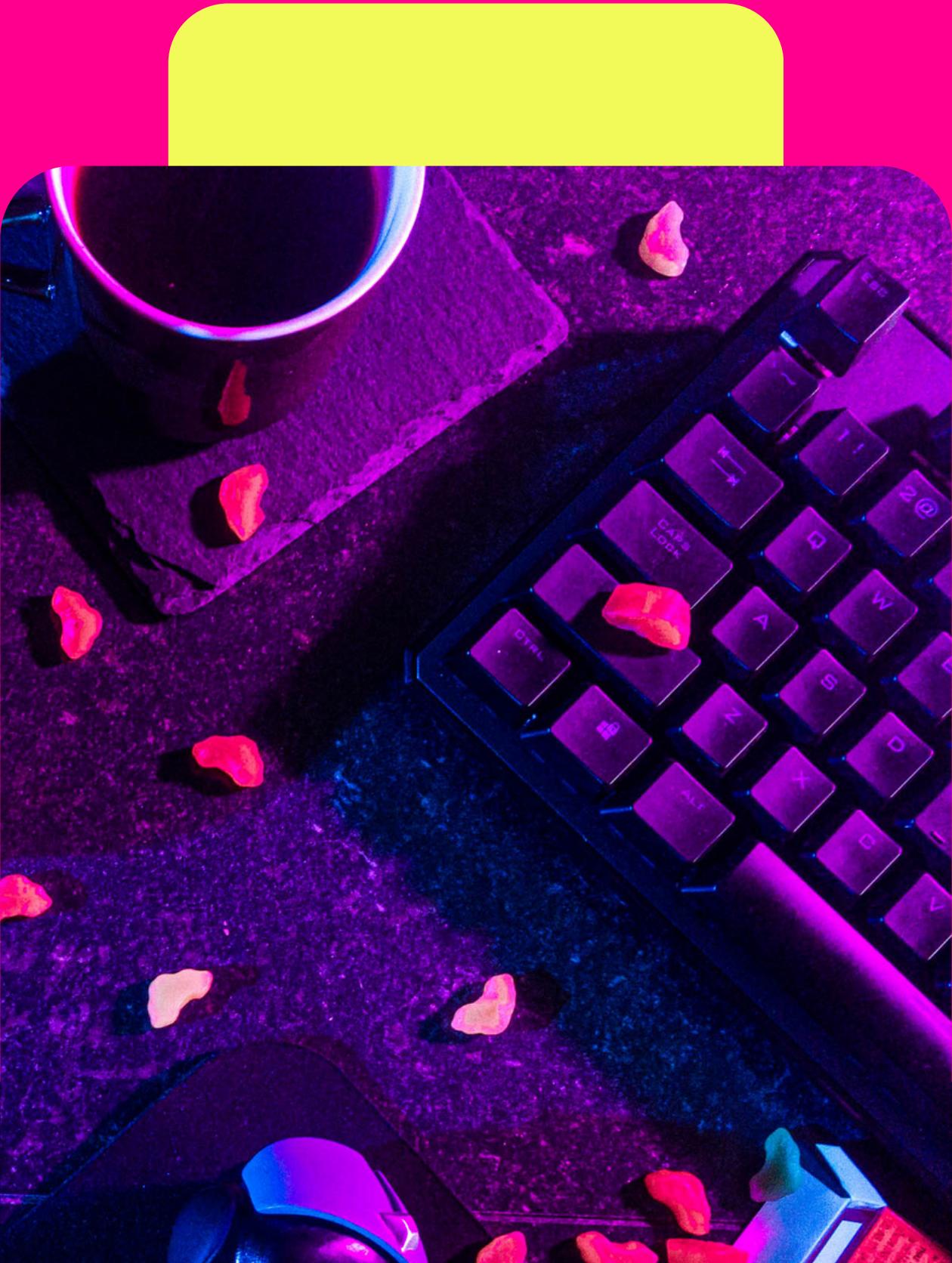
# ANA RAQUEL

## PROFESSORA

- Tecnólogo em banco de dados pela faculdade FIAP.
- MBA em inteligência artificial pela FIAP.
- Mais de 8 anos de experiência como profissional na área de dados tendo atuado em diversos projetos de Banco de Dados, BI, Analytics e Data Science.
- Cientista de dados na FIAP.

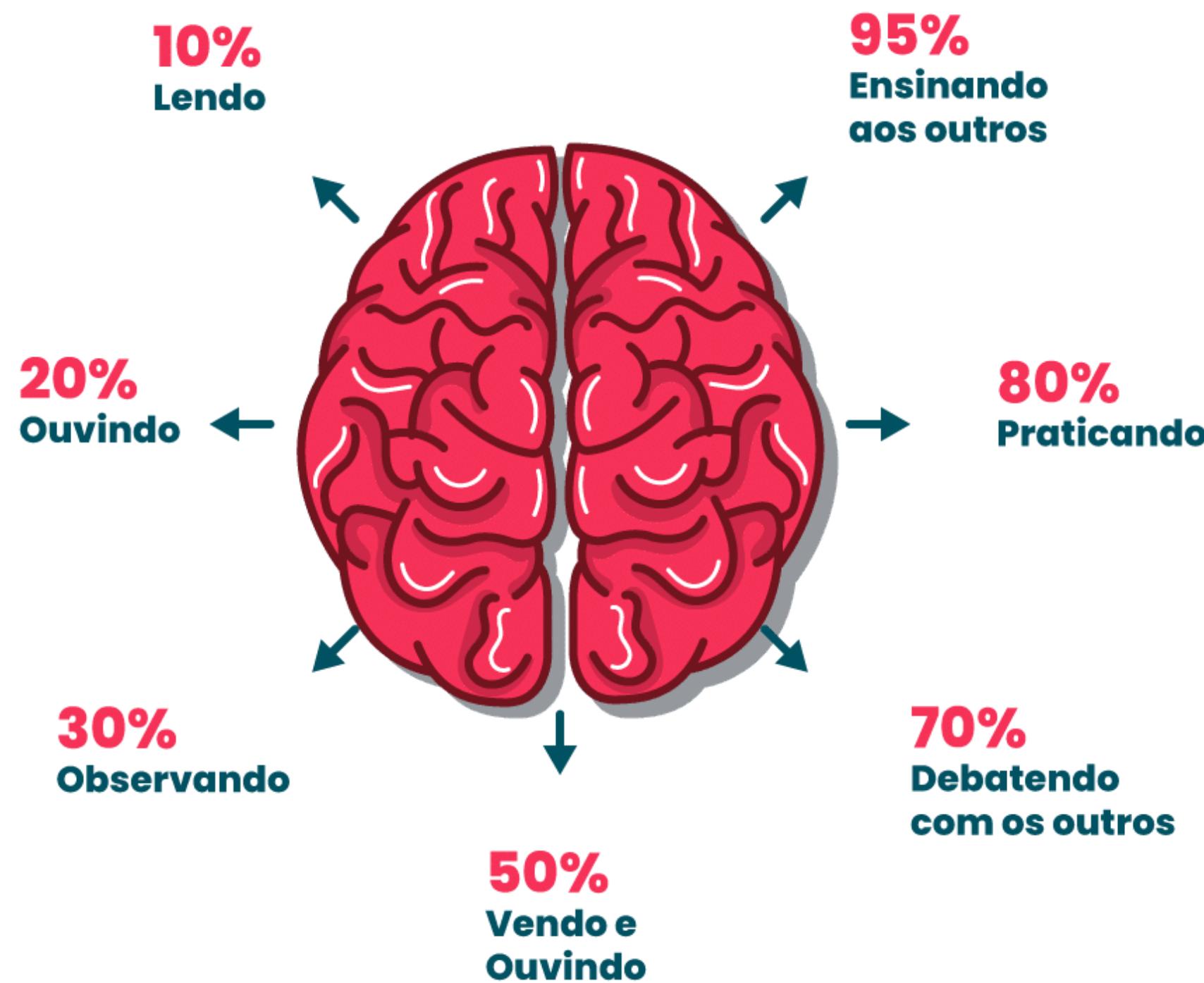


# DEEP LEARNING



# DISCUSSÃO: COMO O SER HUMANO APRENDE?

# COMO NOSO CÉREBRO APRENDE

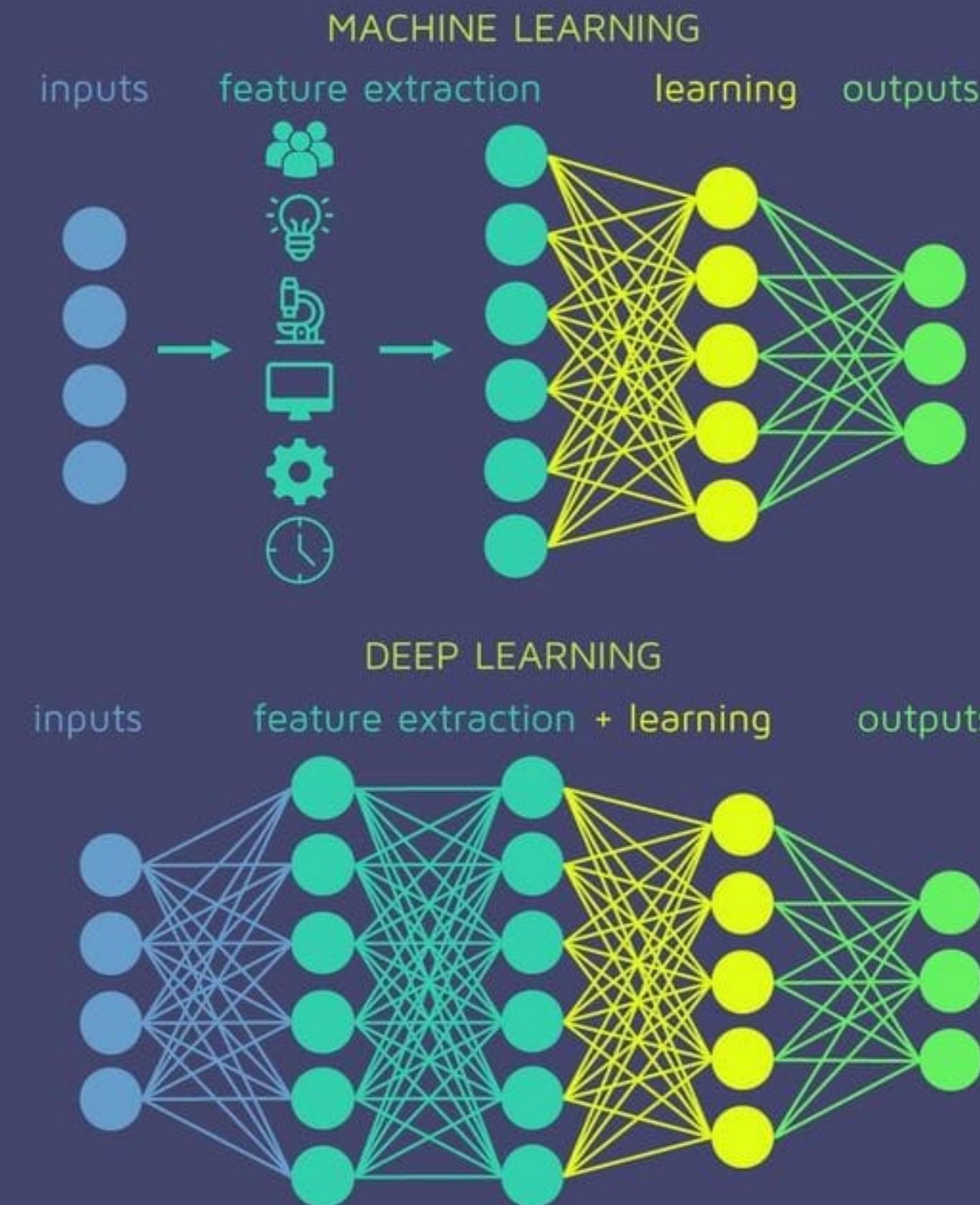


## TENTATIVAS DE ERRO E ACERTO

### DEDICAÇÃO

### PRÁTICAS

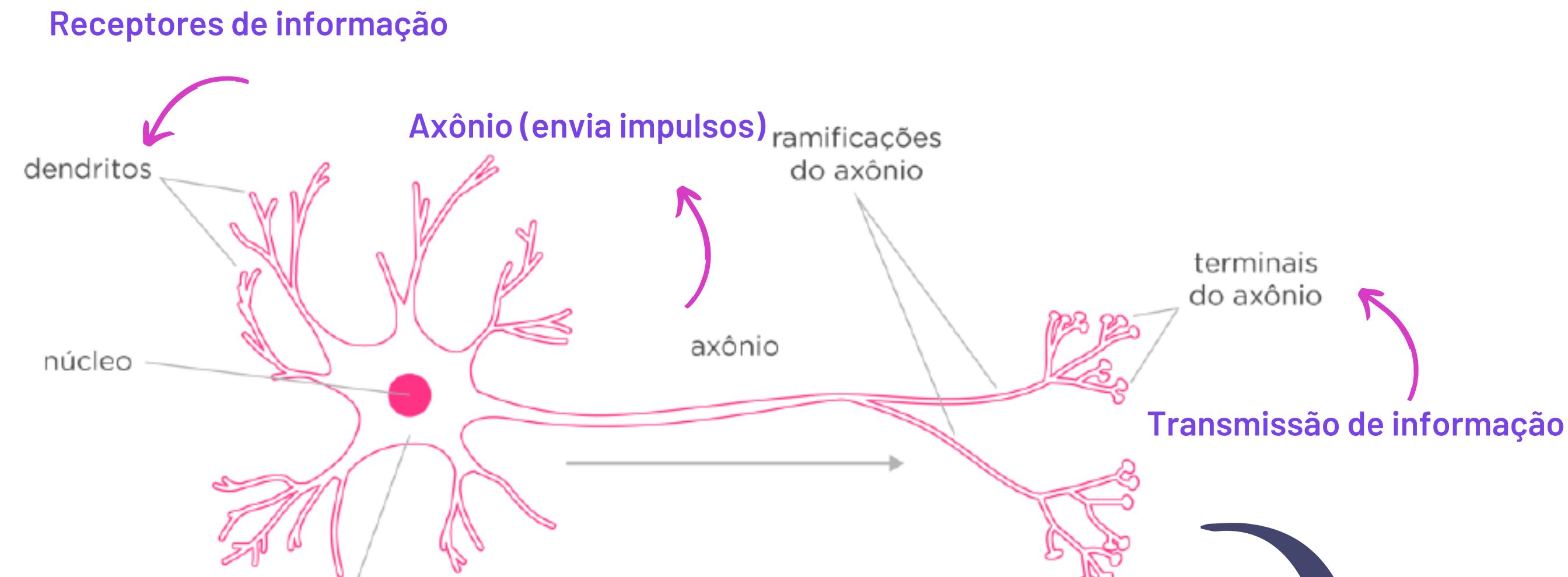
# QUAL É A DIFERENÇA ENTRE MACHINE LEARNING E DEEP LEARNING?



# INSPIRAÇÃO DAS REDES NEURAIS ARTIFICIAIS

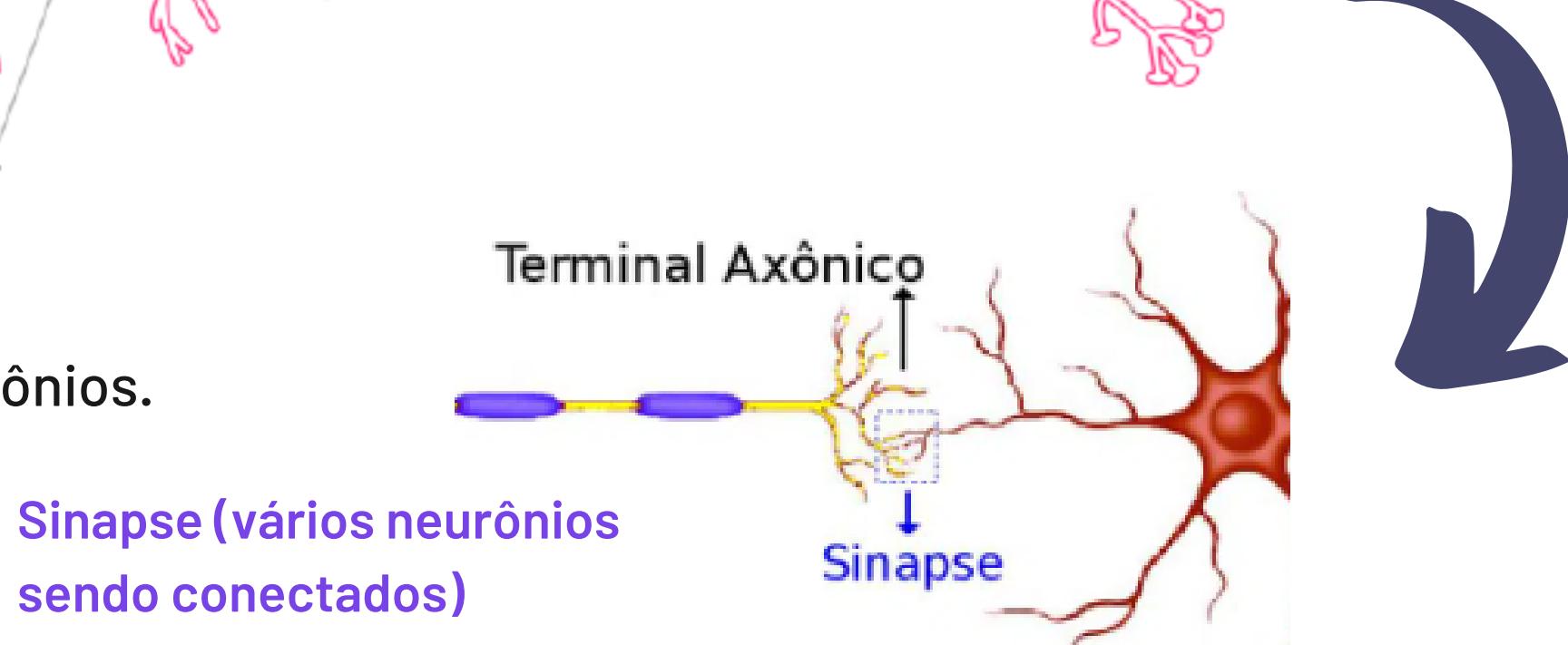
## Cérebro humano

O cérebro humano utiliza milhões de neurônios para realizar uma tarefa, tornando assim as tomadas de decisões mais eficientes.

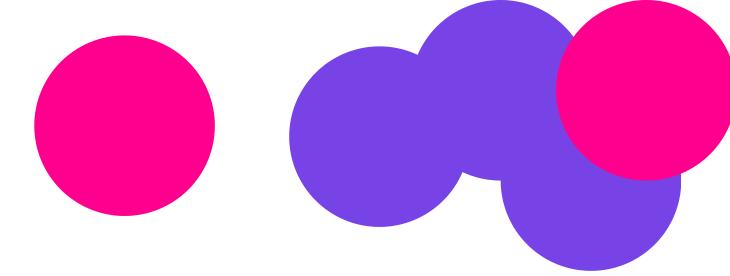


## Curiosidades:

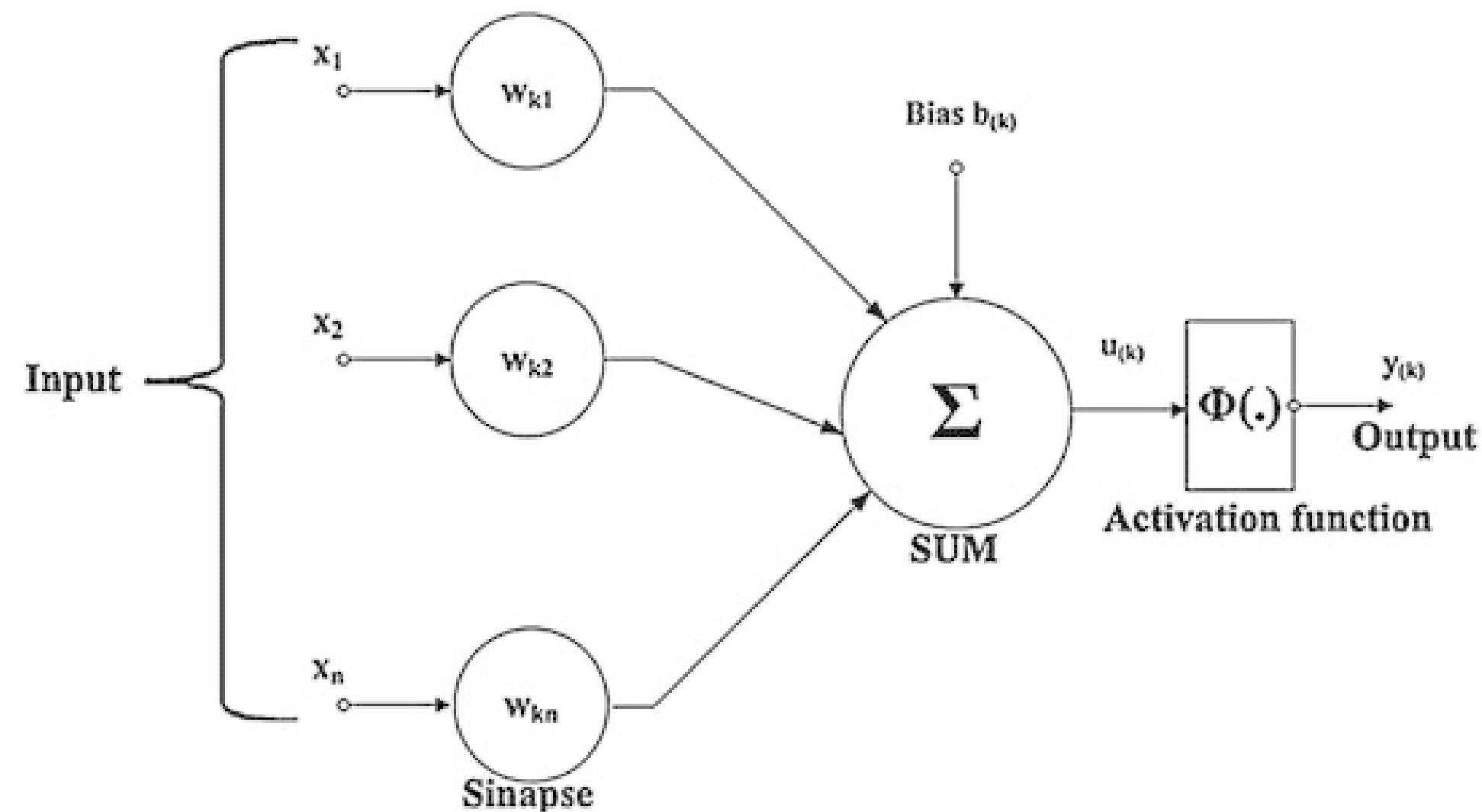
- Nascemos com cerca de 100 bilhões de neurônios.
- Um neurônio pode se conectar a até 100.000 outros neurônios.



# NEUROÔNIO MATEMÁTICO

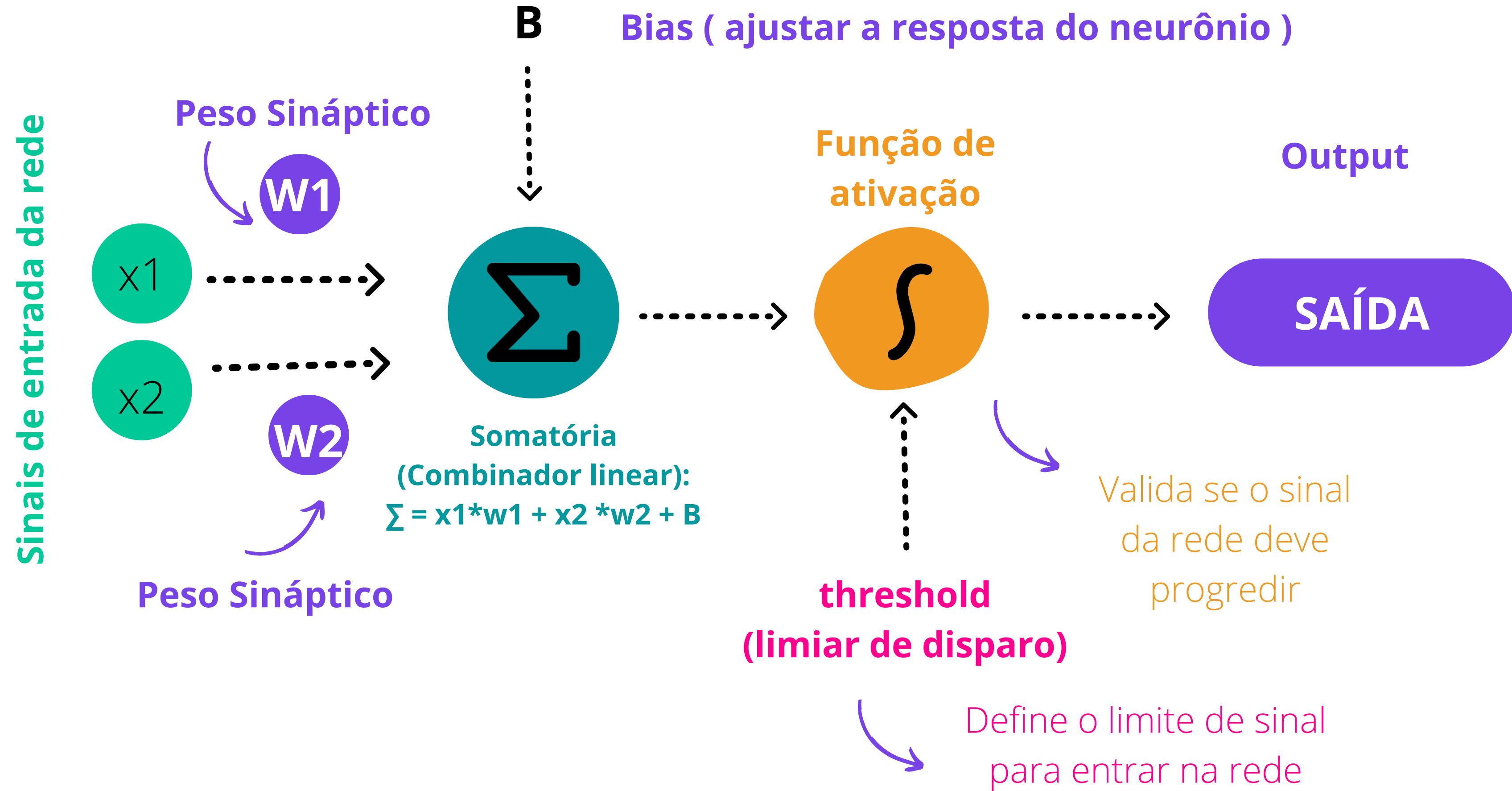


O modelo mais bem aceito foi proposto por **Warren McCulloch** e **Walter Pitts** em **1943**, o qual implementa de maneira simplificada os componentes e o funcionamento de um neurônio biológico.

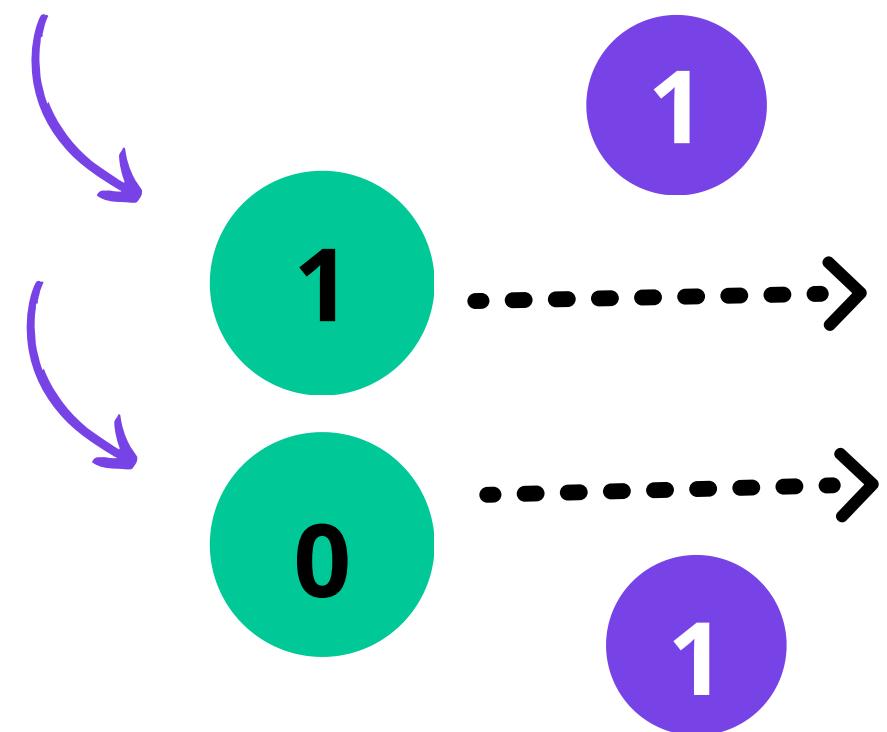


Vamos entender essa fórmula...

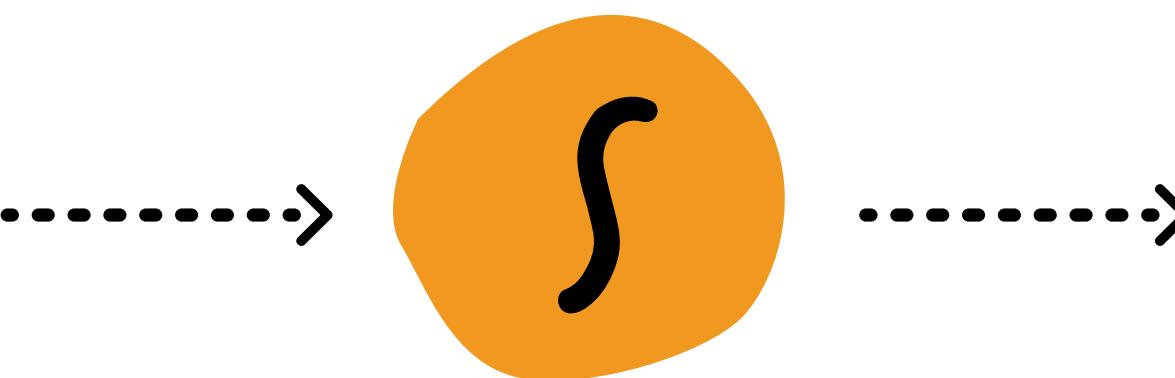
# ARQUITETURA DE UMA REDE NEURAL ARTIFICIAL



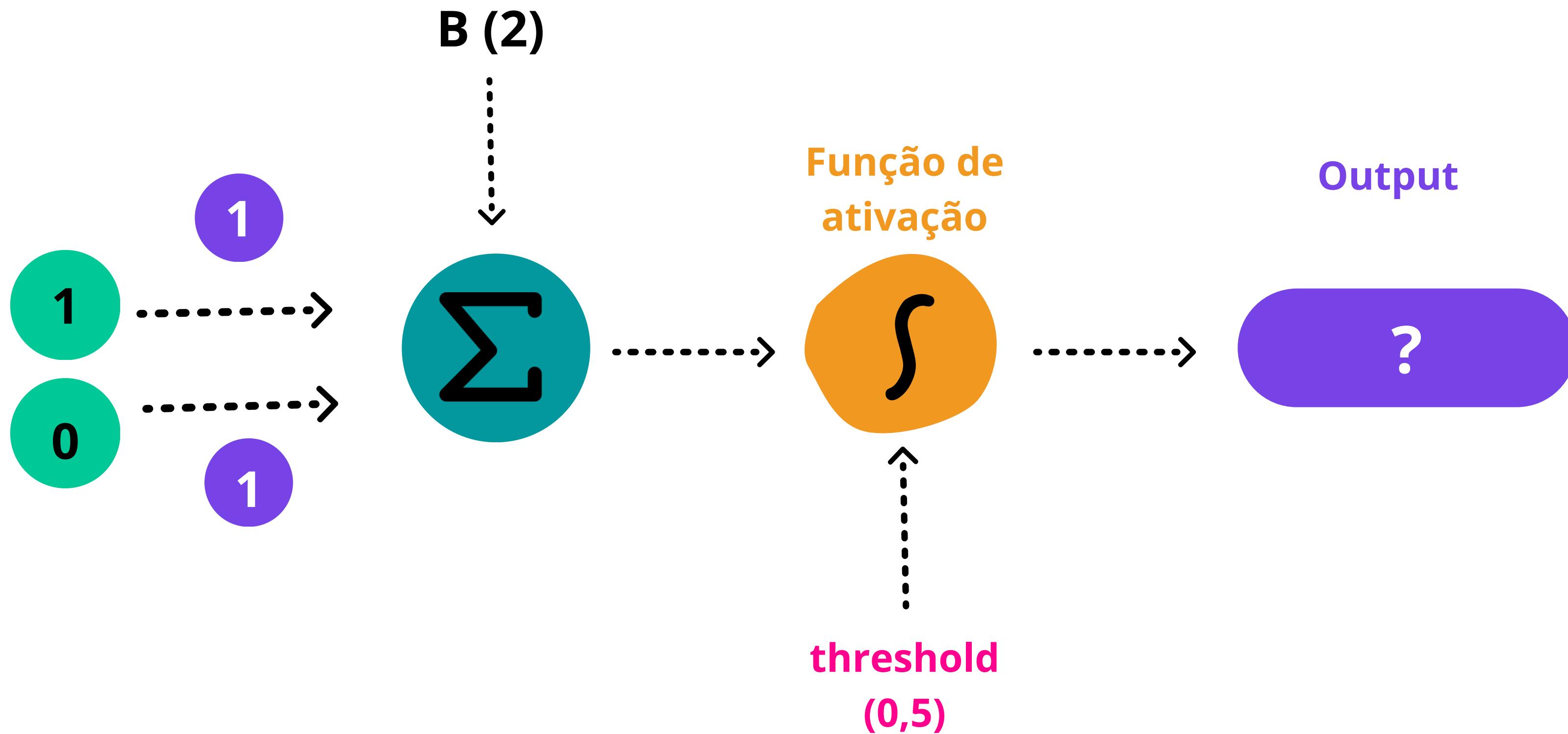
Sinais de  
entrada da rede

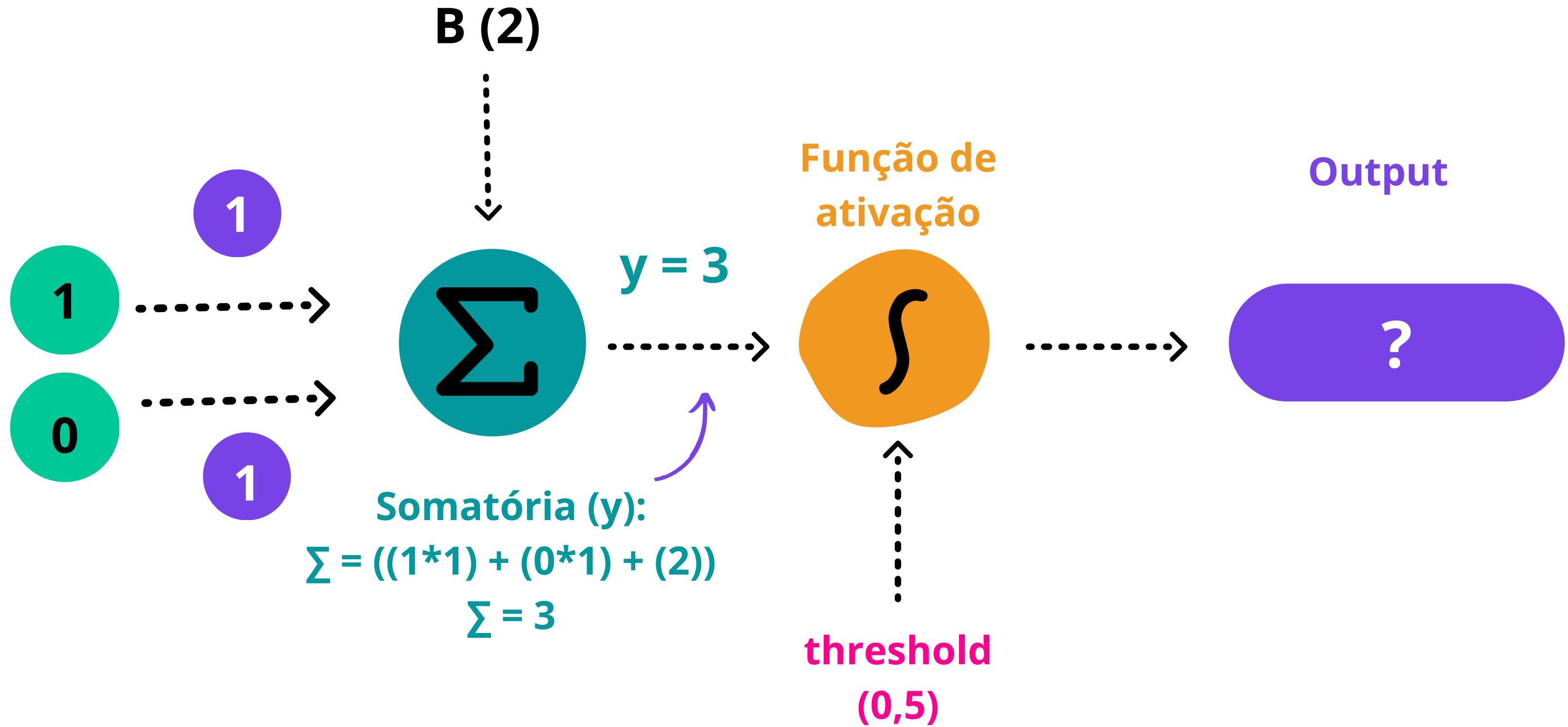


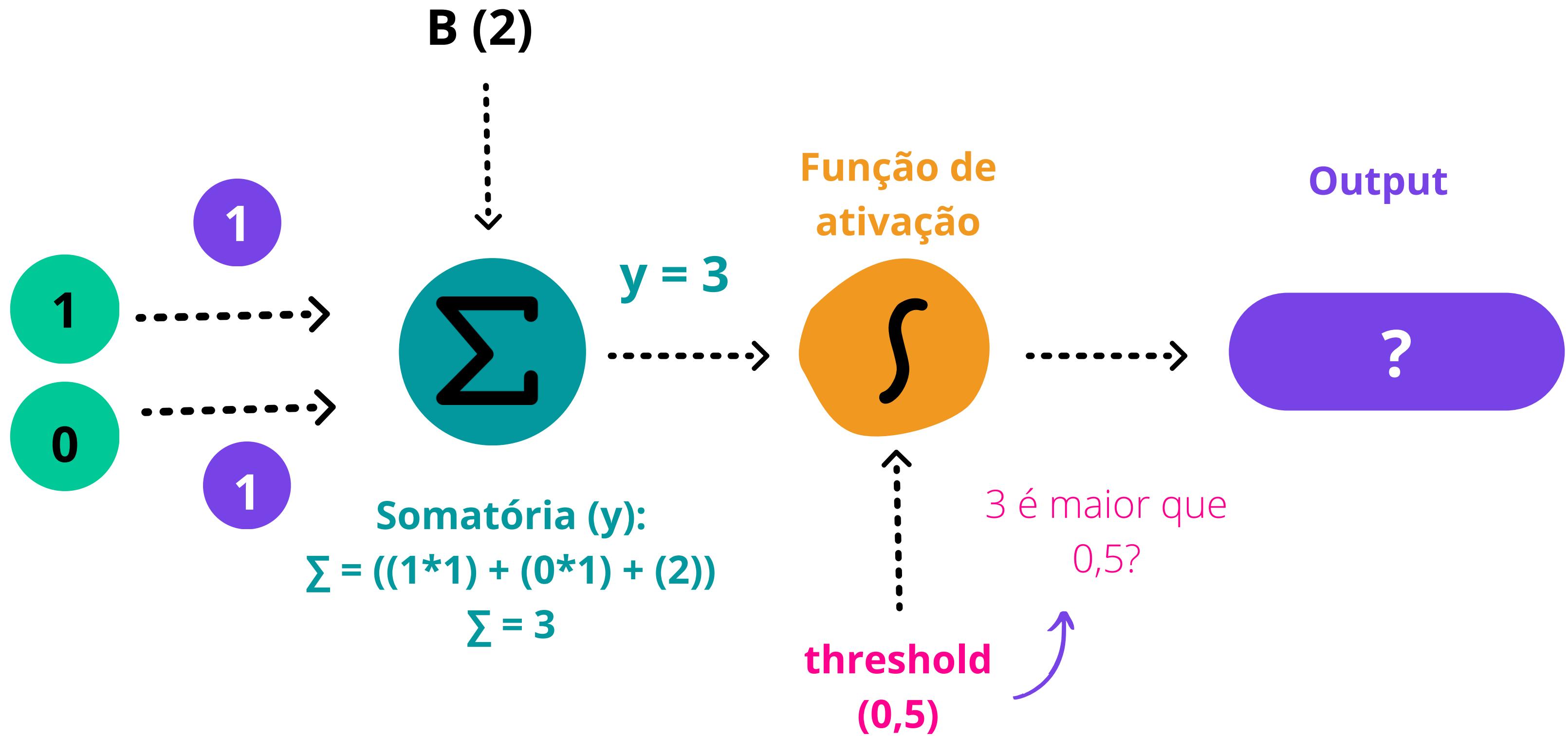
Função de  
ativação

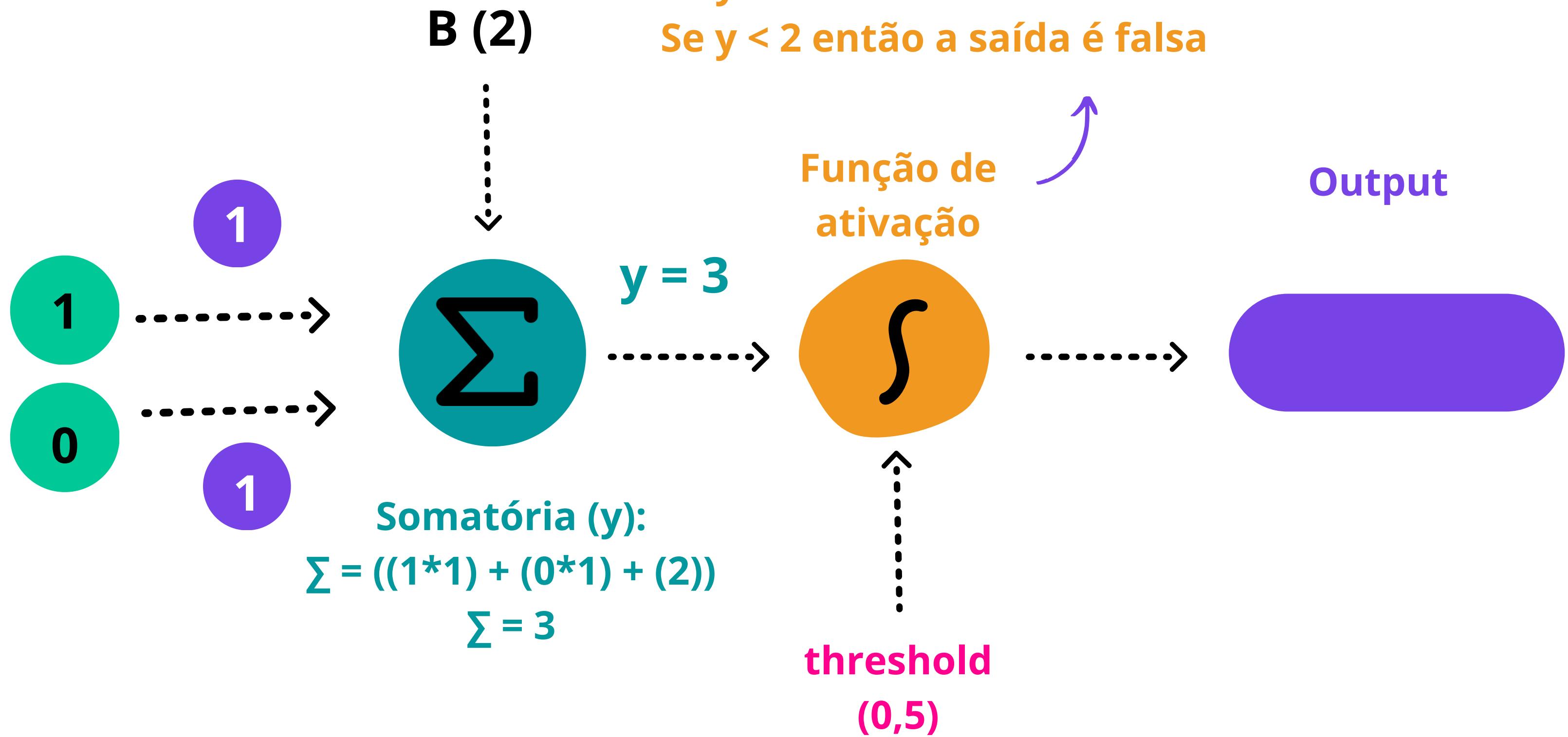


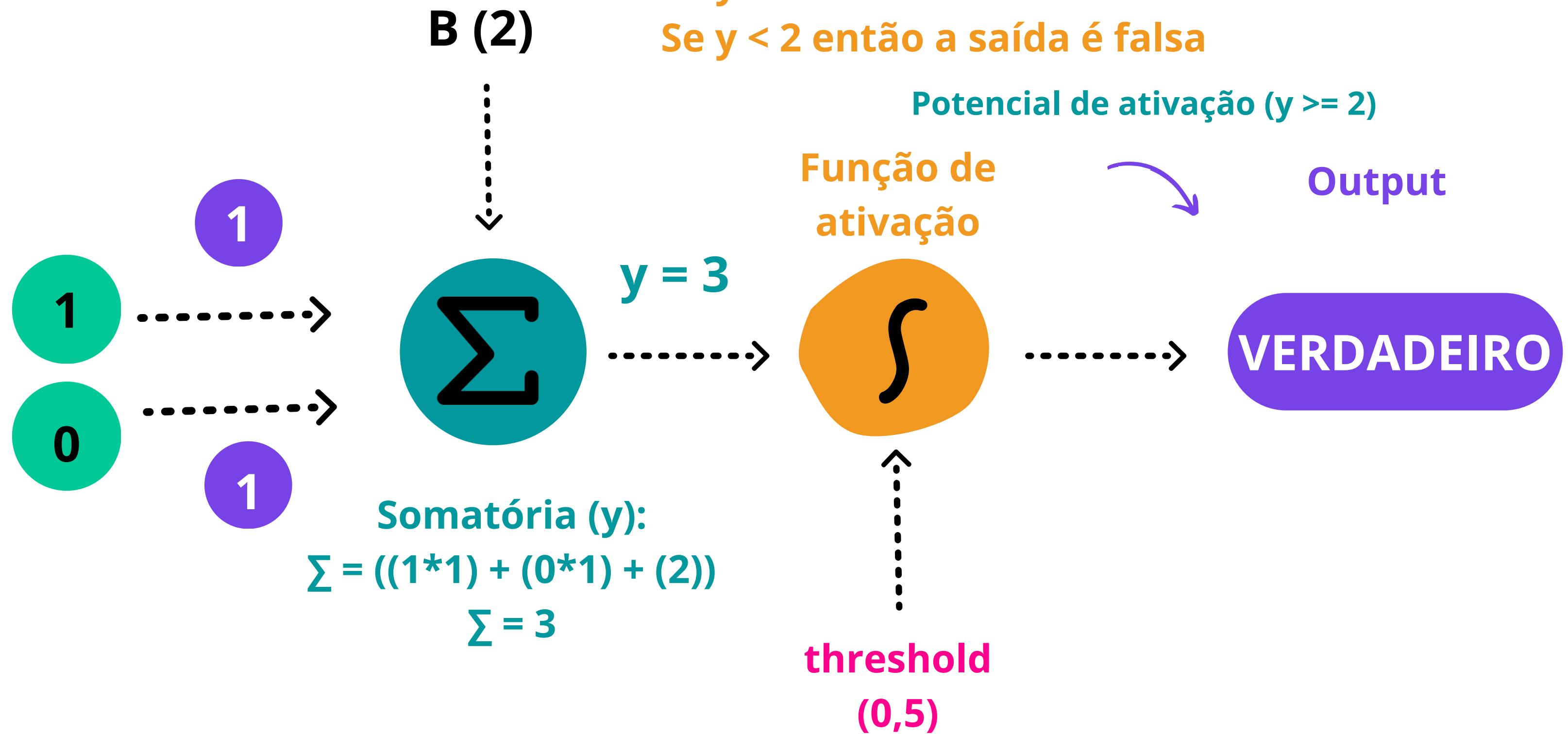
Output











# ALGUMAS OBSERVAÇÕES...

O modelo possui uma natureza **binária (0 e 1)**. Tanto os sinais de entrada quanto a saída, são valores binários.

Os **pesos** da rede neural podem ser **ajustáveis**, inspirados em sinapses, podem ser excitatórias ou inibitórias (positivos ou negativos).

Uma das bases matemáticas para o funcionamento de uma rede neural é a multiplicação de matrizes.

$$\mathbf{X}\mathbf{w} = \mathbf{y}$$
$$\begin{bmatrix} 1 & x_{11} & \dots & x_{1d} \\ 1 & x_{21} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & \dots & x_{nd} \end{bmatrix} \times \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

# COMPONENTES DA REDE NEURAL:

**Sinais de entrada:** São os atributos de entrada, normalmente são as colunas que classificam ou qualificam características aos dados de algum dataframe.

**Pesos sinápticos:** São os pesos que vão ajudar no processo de aprendizagem da rede. Os pesos são responsáveis por ponderar os sinais da rede.

**Combinador linear (somatória):** Tem objetivo de agregar todos sinais de entrada que foram ponderados pelos respectivos pesos sinápticos a fim de produzir um potencial de ativação.

**Limião de ativação:** Especifica a escala apropriada para que o resultado produzido pela a somatória linear possa gerar um valor de disparo de ativação.

**Potencial de ativação:** É o resultado obtido pela diferença do valor entre a somatória e o limiar de ativação. Se o valor for positivo ( $>0$ ) então o neurônio produz um potencial excitatório; caso contrário, o potencial será inibitório.

**Função de ativação:** Objetivo de limitar a saída de um neurônio em um intervalo valores (valida a rede).

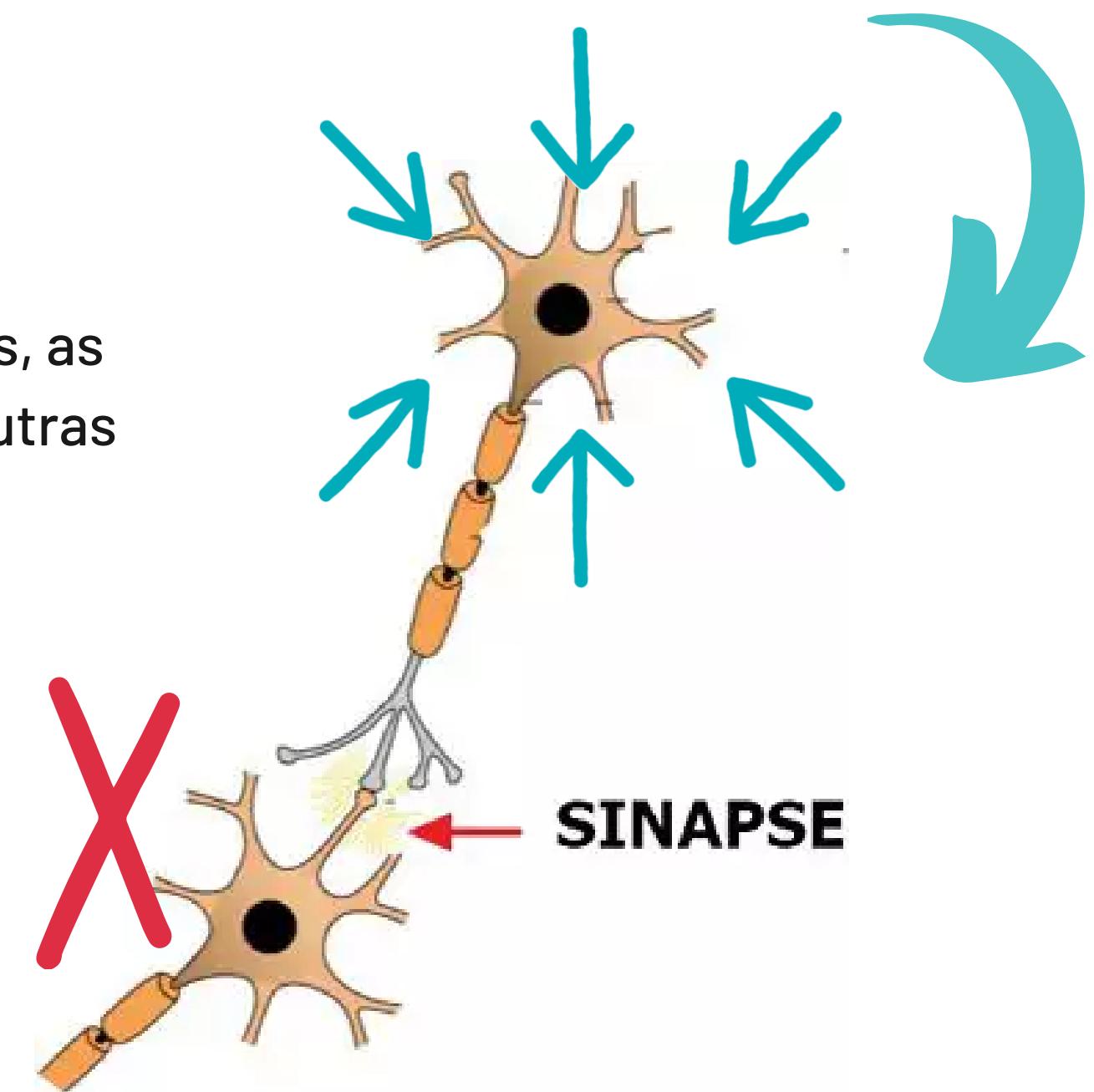
**Saída da rede:** Resultado final após processamento da rede. Pode se tornar entrada para outro neurônio.

# COMO ACONTECE O APRENDIZADO DE UMA REDE NEURAL ARTIFICIAL ?

Esse **ajuste** nas ligações entre os neurônios é uma das características das redes neurais artificiais.

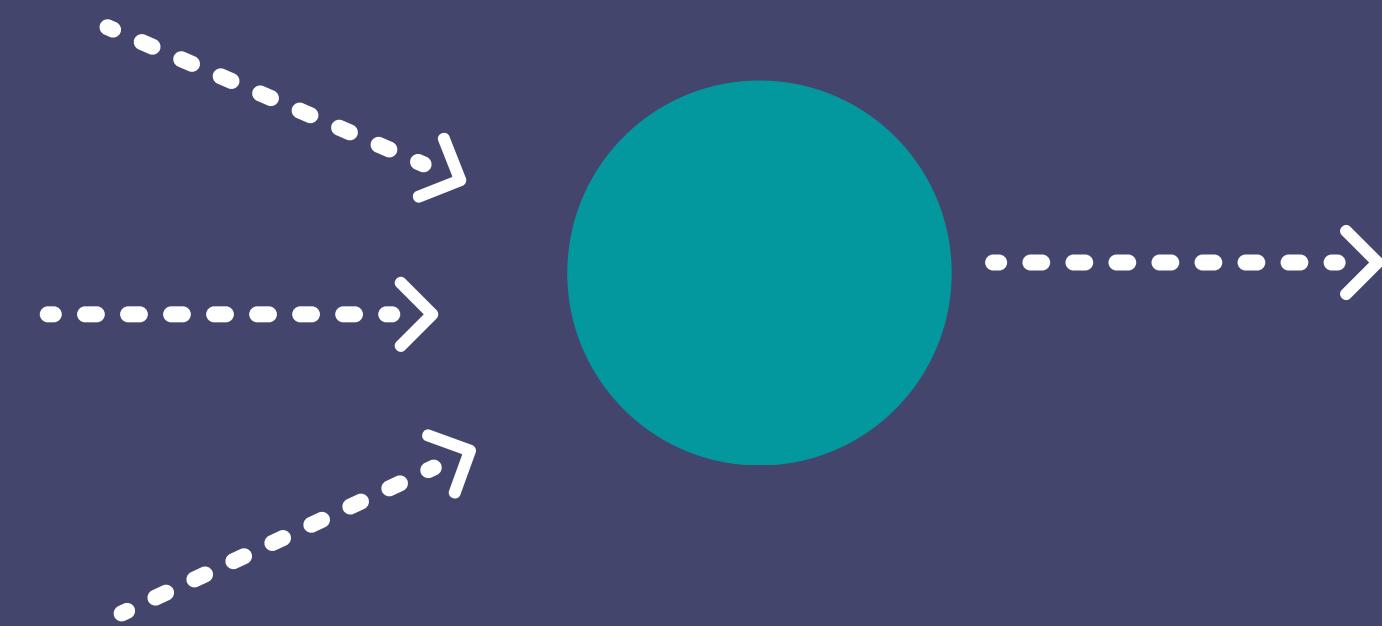
À medida que ocorrem novos eventos, as ligações são reforçadas, enquanto outras são enfraquecidas.

O aprendizado ocorre por sucessivas modificações nas **sinapses** que interconectam em função da maior ou menor liberação de neurotransmissores (mensageiro químico).



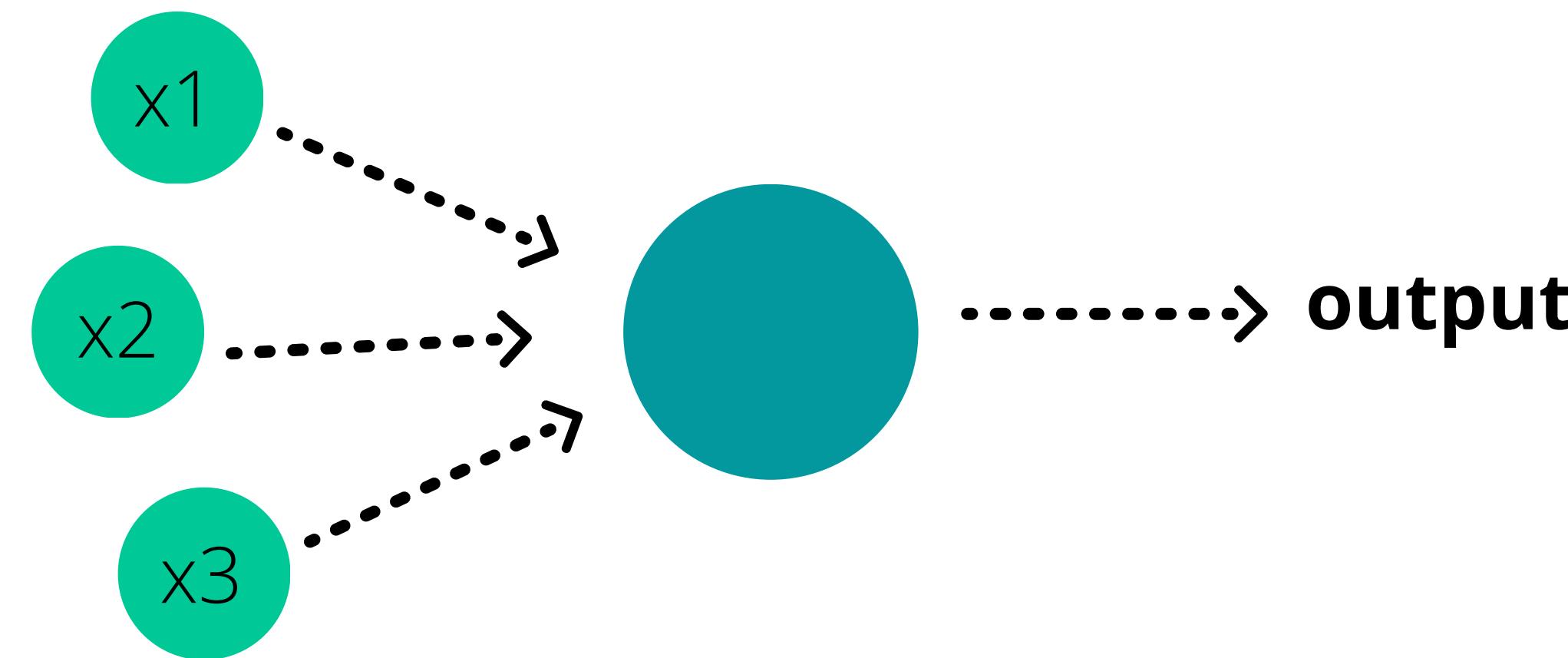
VAMOS APRENDER OS  
PRINCIPAIS TIPOS DE REDES  
NEURAIS ARTIFICIAIS...

## PERCEPTRON



# PERCEPTRON

Desenvolvido nas décadas de 50 e 60 pelo cientista Frank Rosenblatt, inspirado em trabalhos anteriores de Warren McCulloch e Walter Pitts



Podemos dizer que o Perceptron pode pesar diferentes tipos de evidências para tomar decisões. Vamos entender um exemplo básico a seguir.

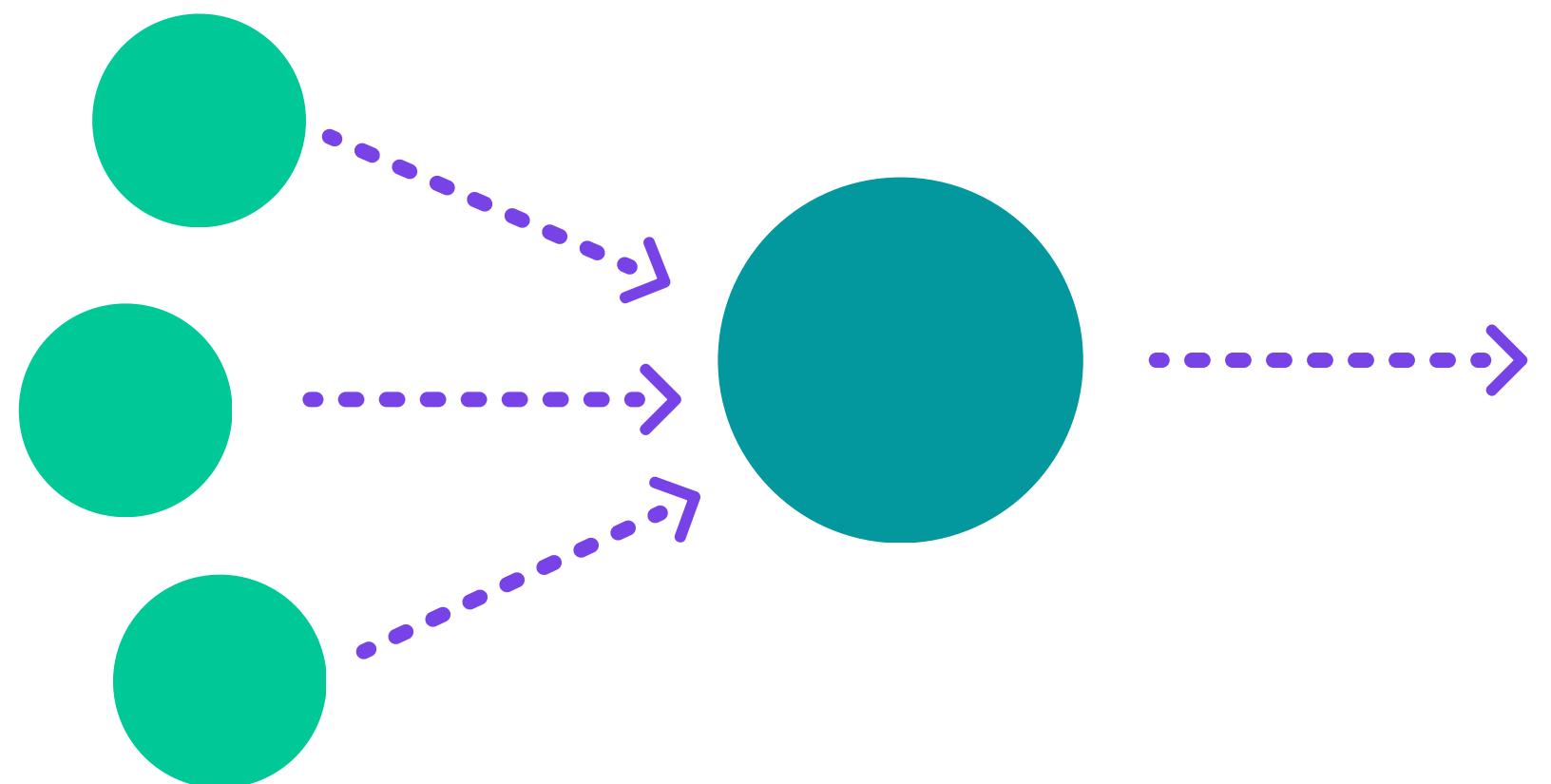
# CARACTERÍSTICAS DO PERCEPTRON

Perceptron é uma rede neural de **camada única** e um **Perceptron de várias camadas é chamado de Rede Neural Artificial.**

O Perceptron é um **classificador linear (binário)**.

Utilizado na **aprendizagem supervisionada** e pode ser usado para classificar os dados de entrada fornecidos.

Classifica a entrada separando duas categorias com uma linha reta.



# REDE NEURAL ARTIFICIAL PERCEPTRON

Esse ano vai ter o show do Guns N' Roses e você quer muito ir ao show!

Vamos supor que você tem 3 fatores para tomar a sua decisão:

- O show será em outro Estado?
- Você vai ter companhia para o show? ( amigo(a), namorado(a), pai, mãe, etc )
- Você tem dinheiro?



Esses fatores são nossas variáveis binárias para a entrada de nossa rede neural artificial perceptron:

**x<sub>1</sub> = O show é em outro Estado?**

**x<sub>2</sub> = Você vai ter companhia para o show? (amigo(a), namorador(a), pai, mãe, etc)**

**x<sub>3</sub> = Você tem dinheiro?**

## Variáveis de entrada da rede

As variáveis X1, X2 e X3 são representadas por valores binários...por exemplo:

**O show é em outro Estado?**

**SIM**

**X1 = 1**

**O show é em outro Estado?**

**NÃO**

**X1 = 0**

Agora vamos supor que você quer muito ver o Slash tocar, não importa se você vá ao show sozinho, tenha que atravessar o país ou se não tem dinheiro para ir...



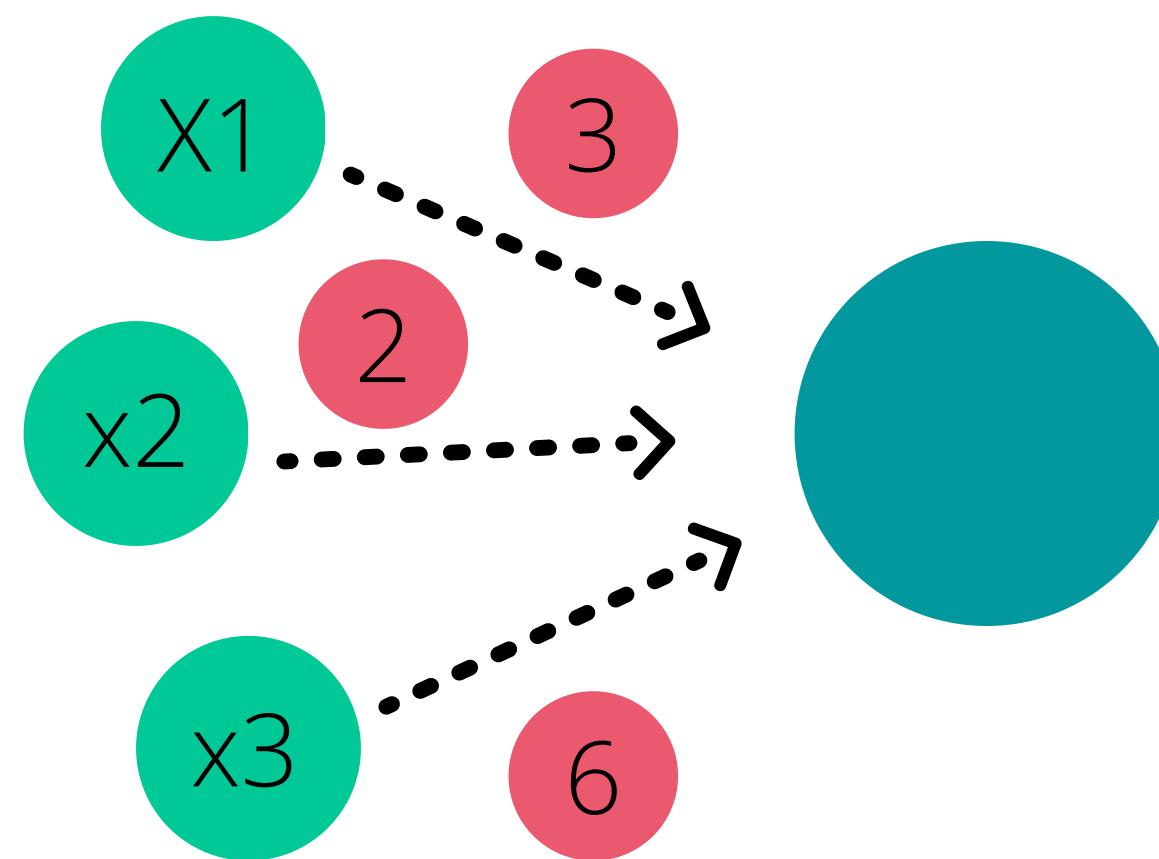
## Dando importância para as variáveis

Como você está economizando para comprar um carro, então o maior peso será atribuído para X3 pois o dinheiro pesa bastante na sua tomada de decisão.

X1 <- 3

X2 <- 2

X3 <- 6

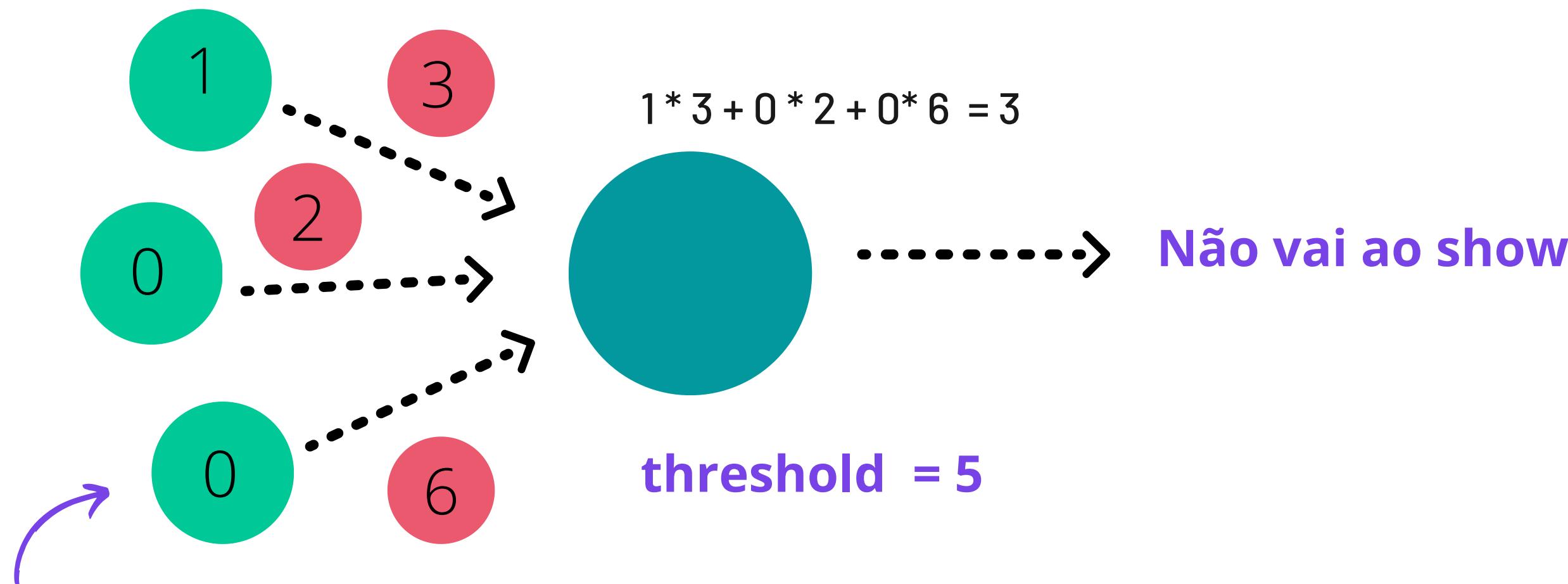


## Definindo o threshold

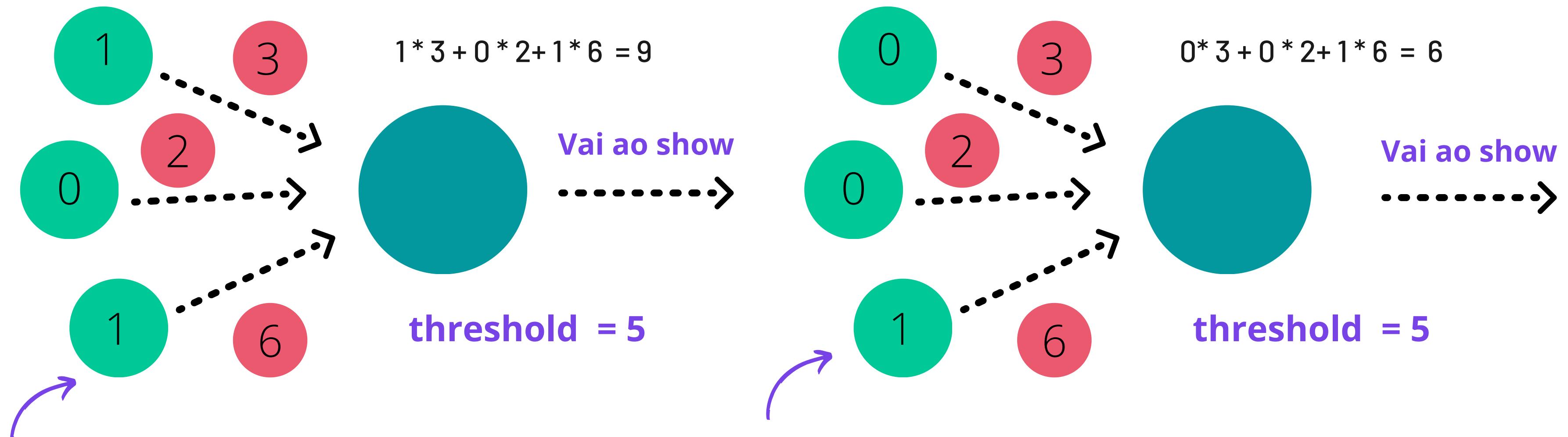
Suponha que você definiu o threshold como 5 para o perceptron.

Com essas escolhas, o Perceptron implementa o modelo de tomada de decisão desejado, produzindo 1 sempre que você tiver dinheiro e 0 sempre que não tiver.

Não faz diferença para você se o show for em outro Estado ou se terá companhia para ir...



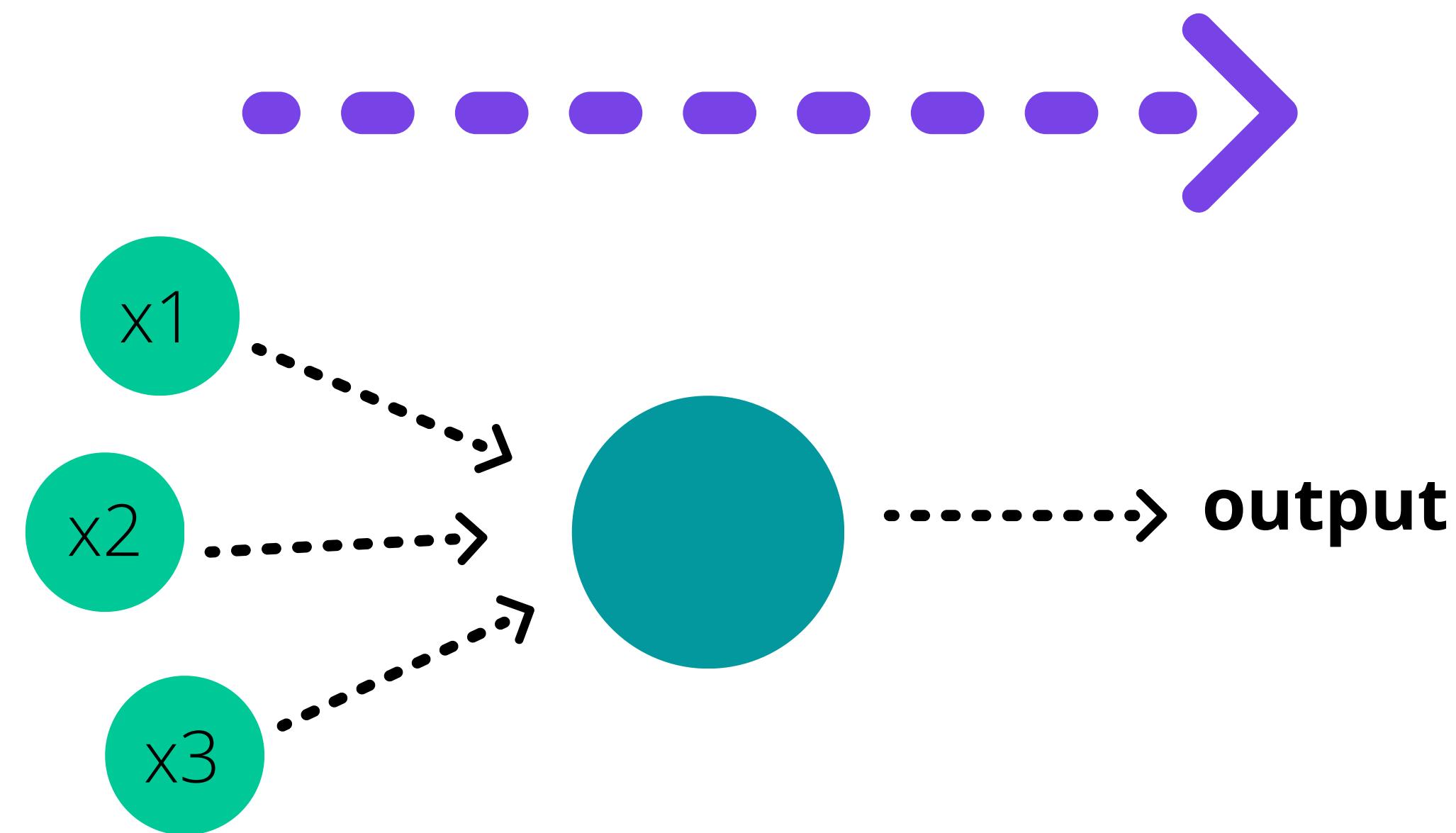
**Variando os pesos e o limiar, podemos obter diferentes modelos de tomada de decisão.**



Quanto mais camadas, mais complexo e abstrato são as tomadas de decisões.

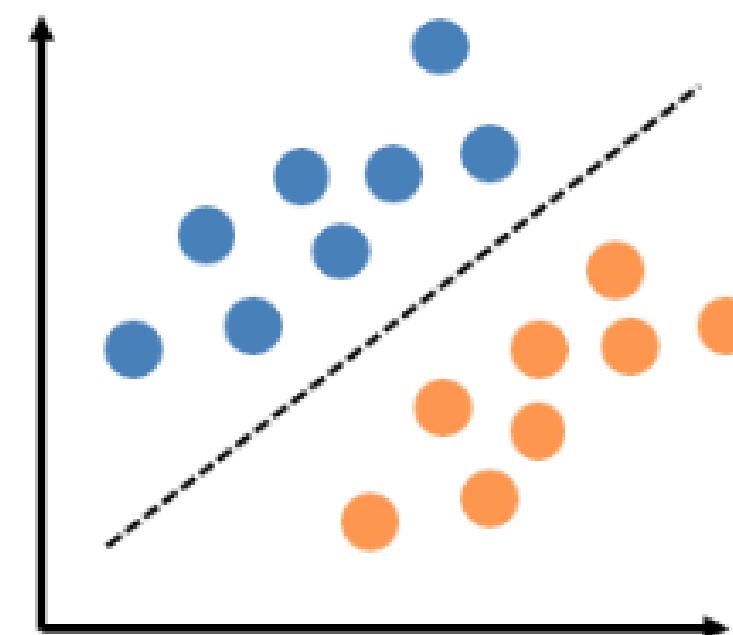
# MODELO FEED-FOWARD

O Perceptron segue o modelo feed-foward. Nesse tipo de modelo as entradas da rede são enviadas para o neurônio , são processadas e resultam em uma saída.

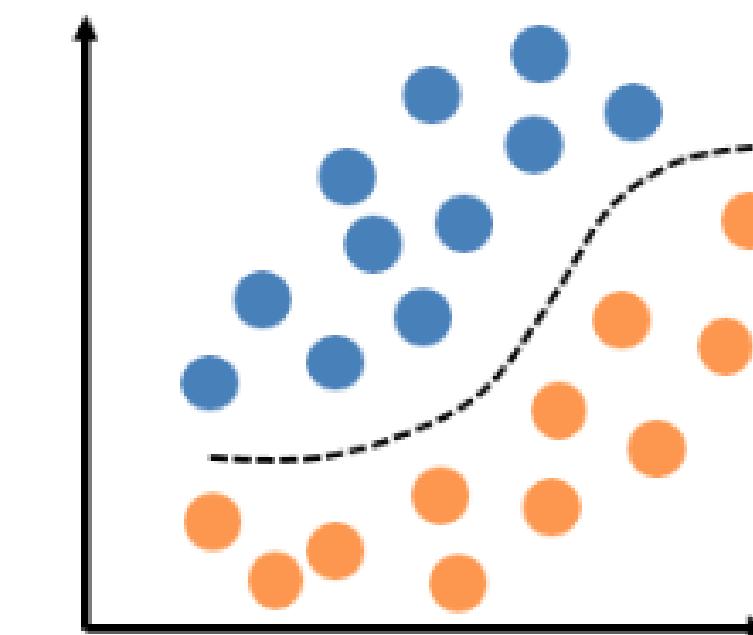


Um único perceptron pode resolver problemas lineares, a questão é que os dados do mundo real não são muito lineares, fazendo com que esse tipo de neurônio artificial não seja muito útil para solucionar problemas não lineares.

Linear

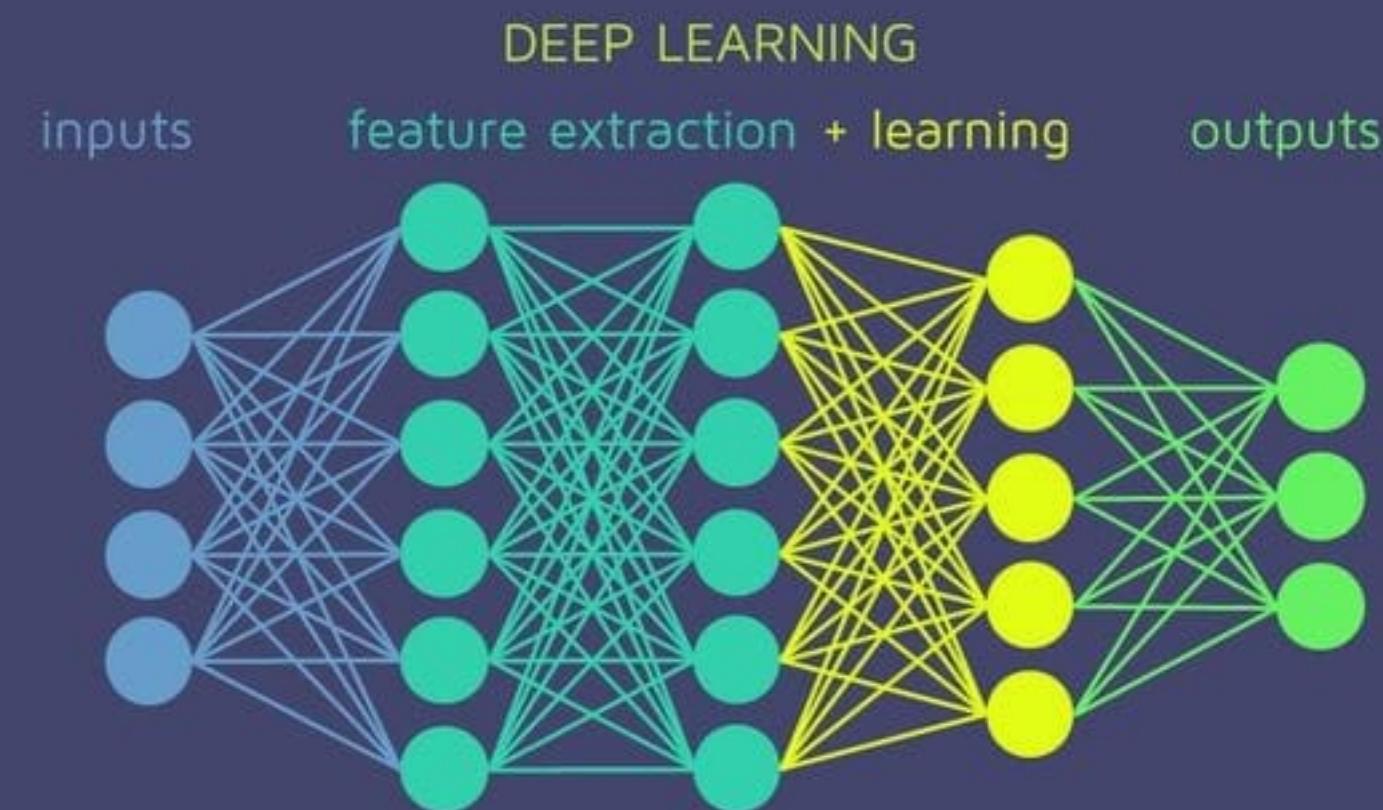


Nonlinear



E como podemos resolver problemas não lineares?

# REDES MULTILAYER PERCEPTRONS

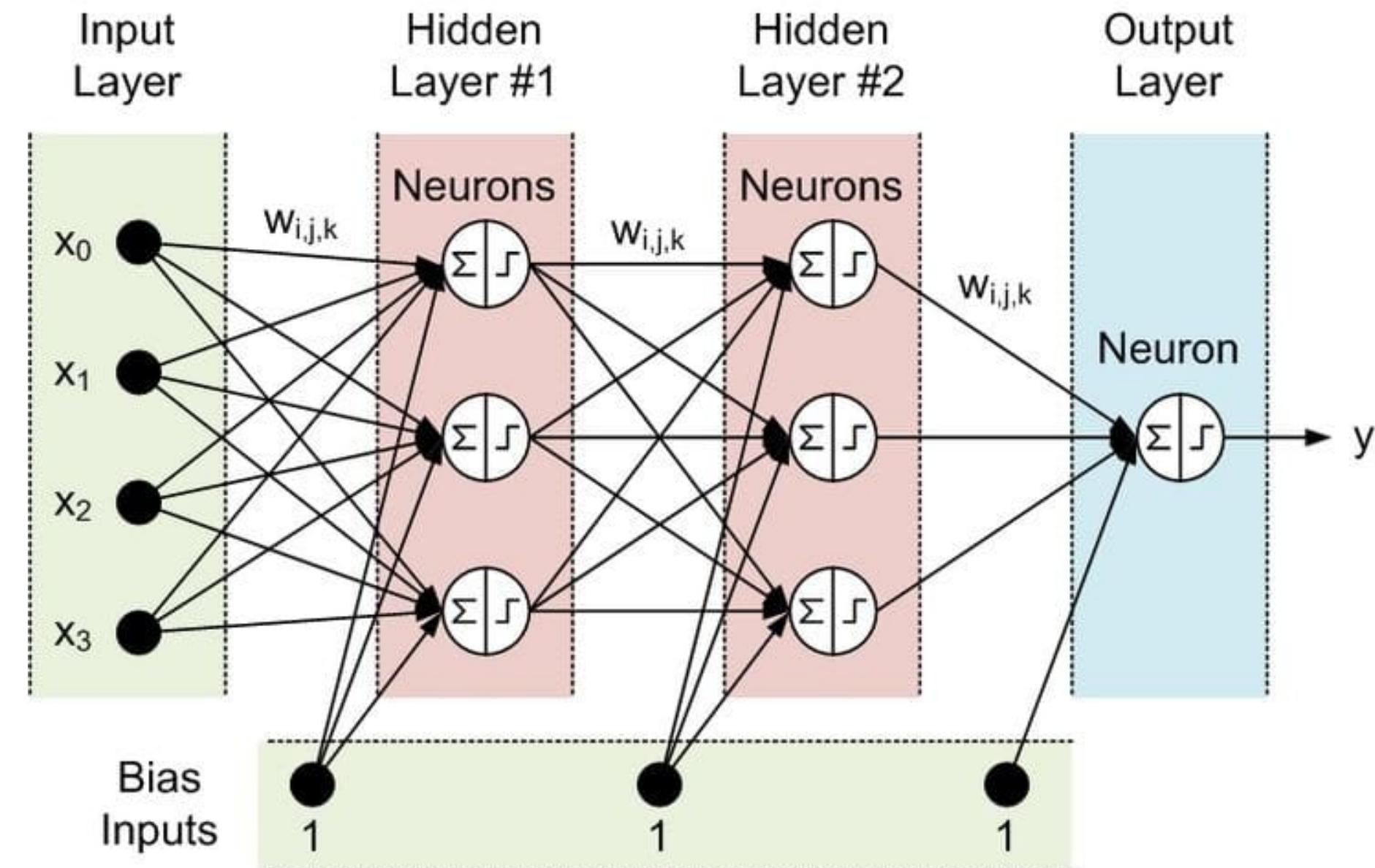


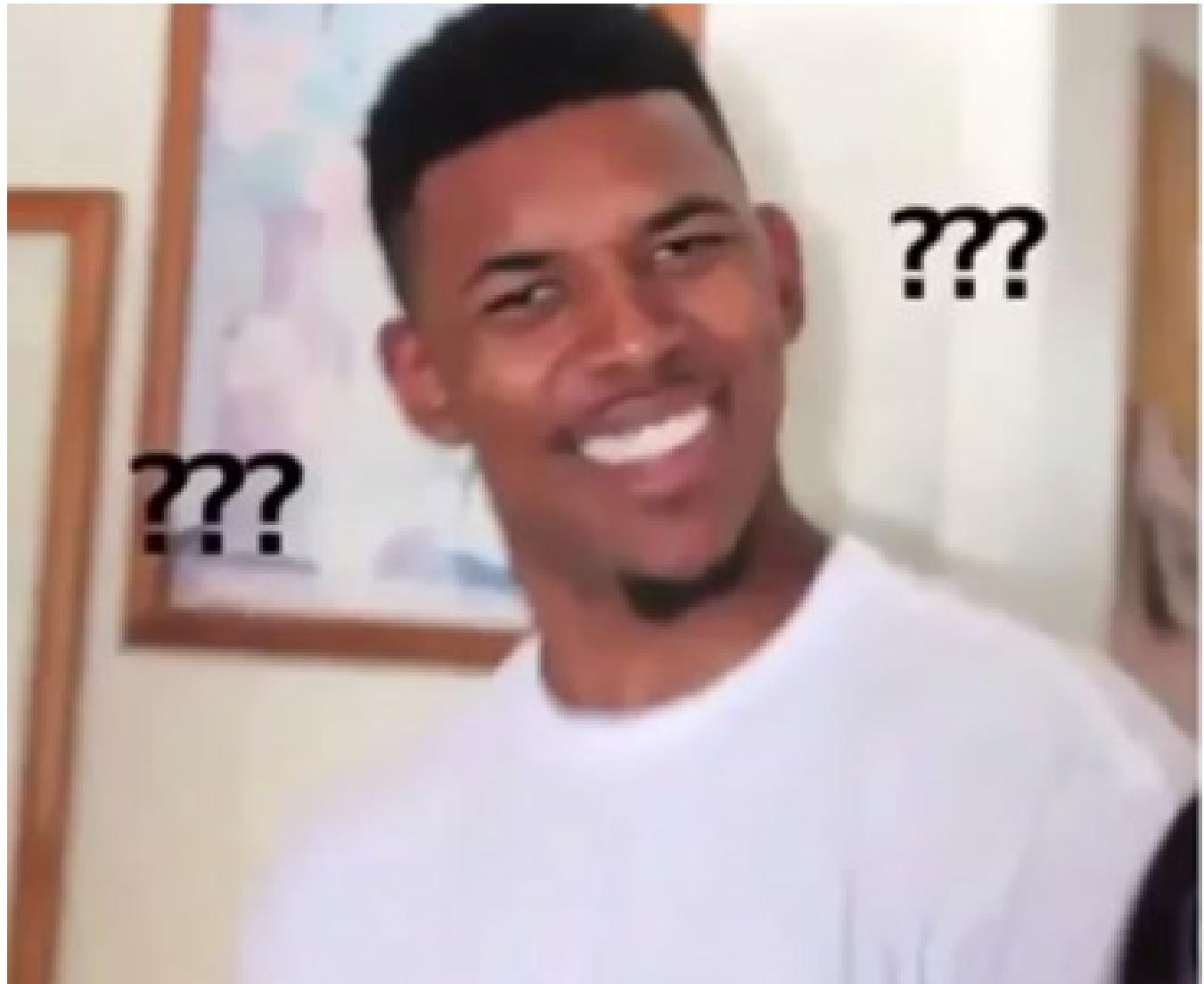
# REDES MULTILAYER PERCEPTRONS

Diferente do Perceptron, uma rede neural de multicamadas constrói uma rede com várias camadas ocultas, modelando assim a correlação (ou dependências) entre os dados de entrada com os de saída.

O treinamento envolve o ajuste dos parâmetros, ou os pesos e bias, do modelo para minimizar o erro.

Utilizado para aprendizagem supervisionada.





**Camadas ocultas?  
Minimizar erro?**

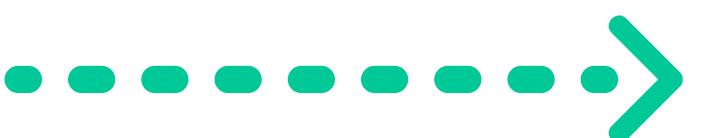
**Como essa rede neural funciona?**



# MODELO BACKPROPAGATION

O modelo de backpropagation é utilizado para fazer os ajustes dos pesos e de bias em relação ao erro. Esse erro pode ser calculado por várias maneiras, como por exemplo utilizando o erro quadrático médio (MSE – Mean Squared Error).

Esse modelo traz basicamente a propagação e a retropropagação.

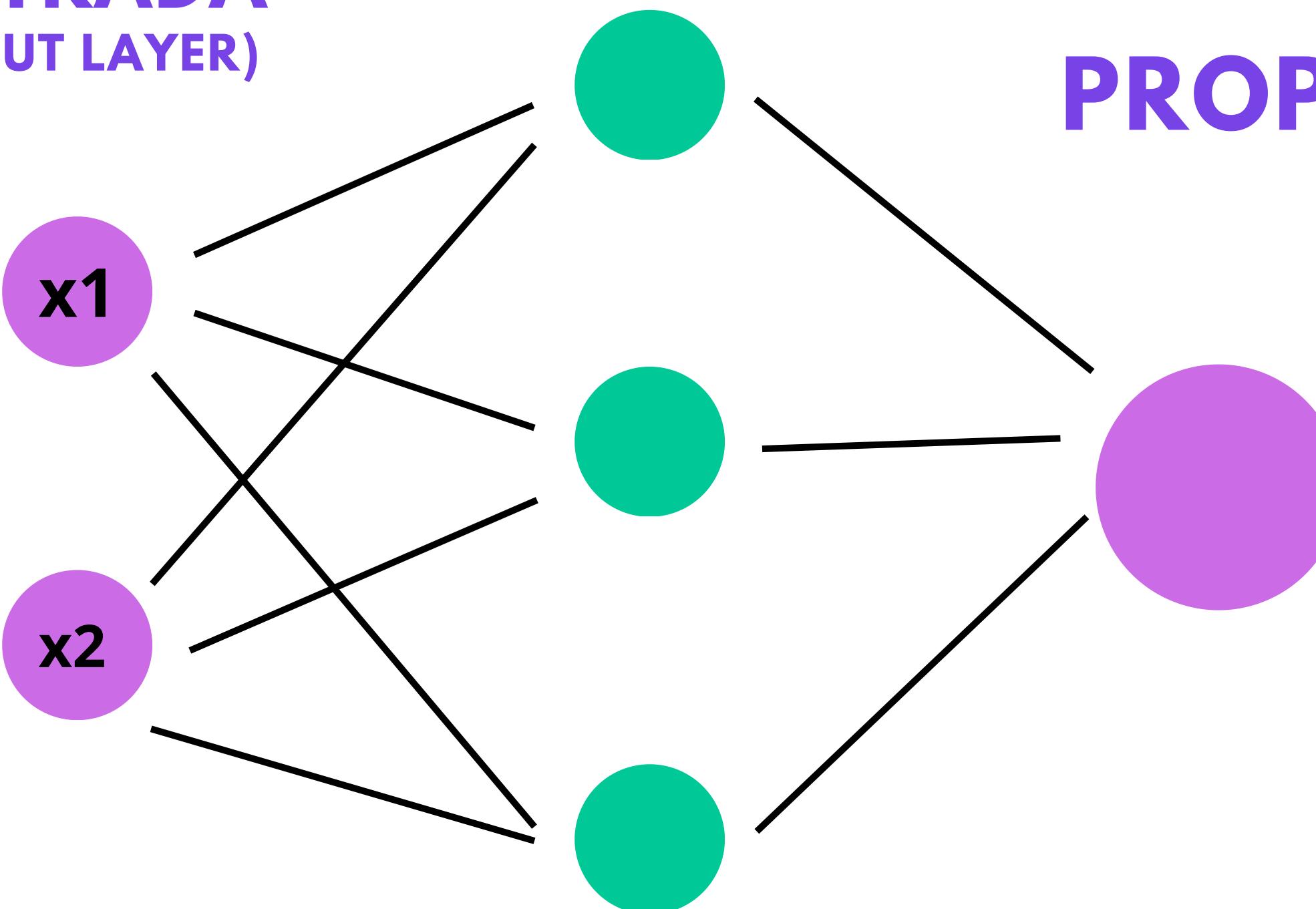


## PROPAGAÇÃO

## RETROPROPAGAÇÃO

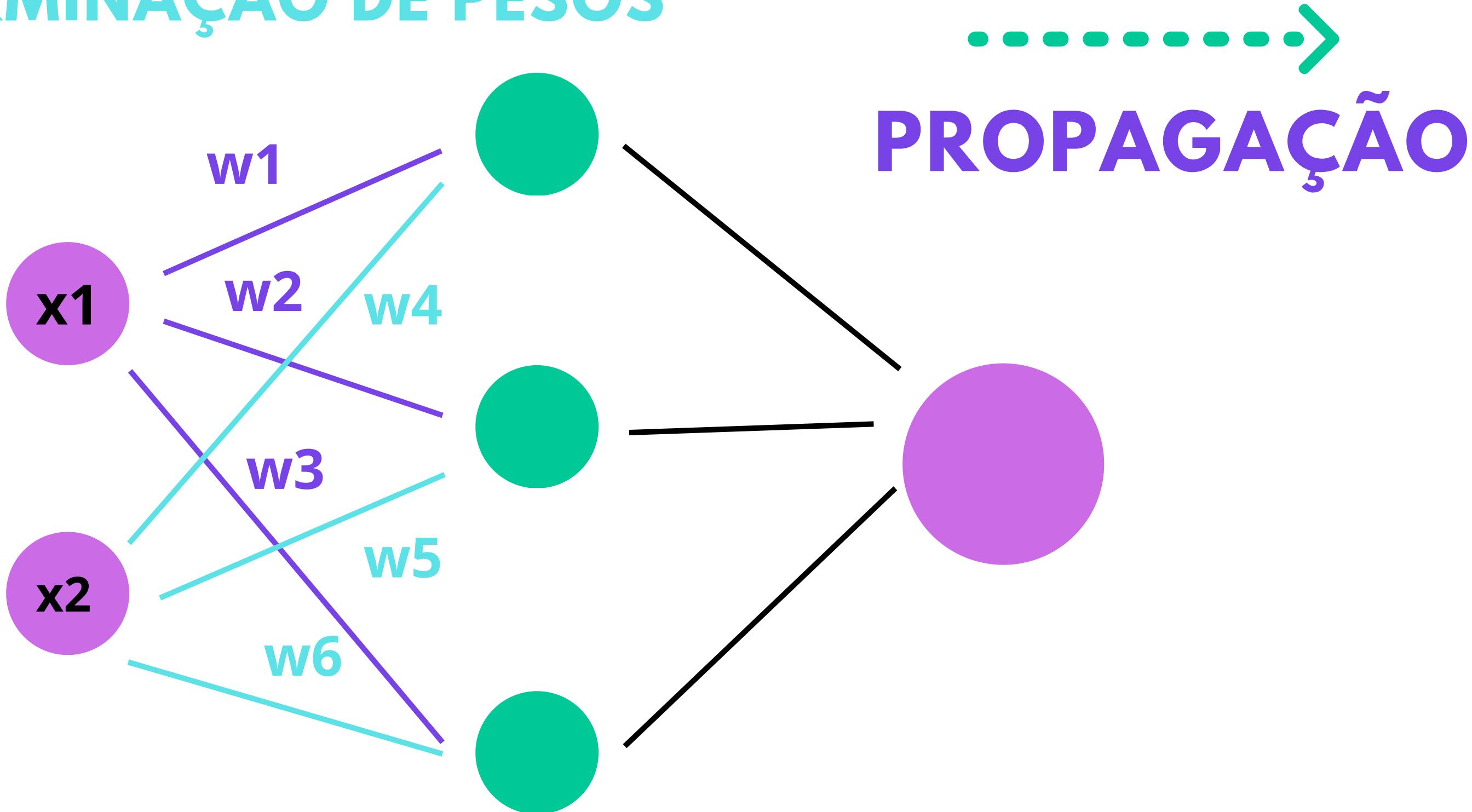


**ENTRADA**  
(INPUT LAYER)



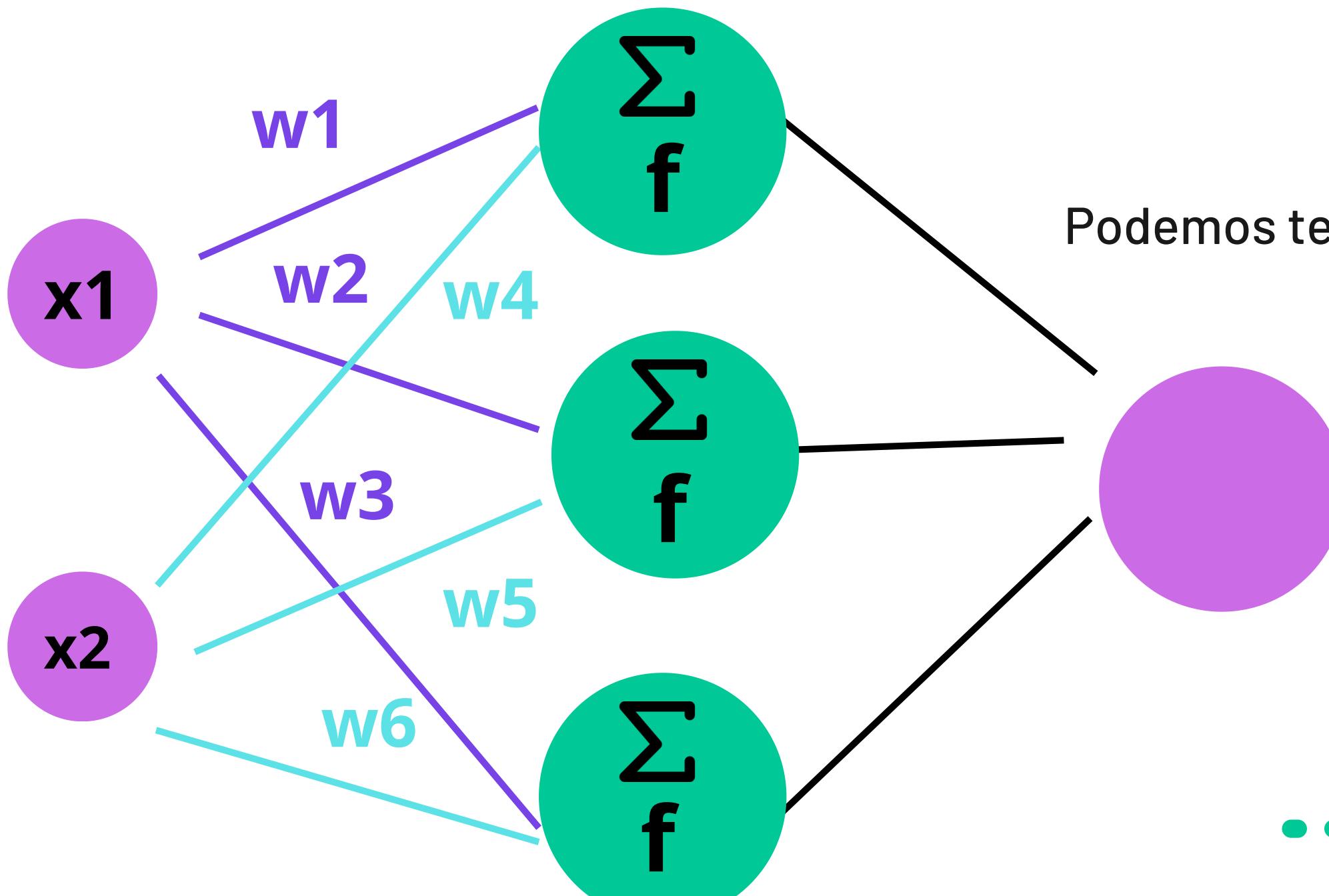
.....  
**PROPAGAÇÃO**

# DETERMINAÇÃO DE PESOS



**bias**

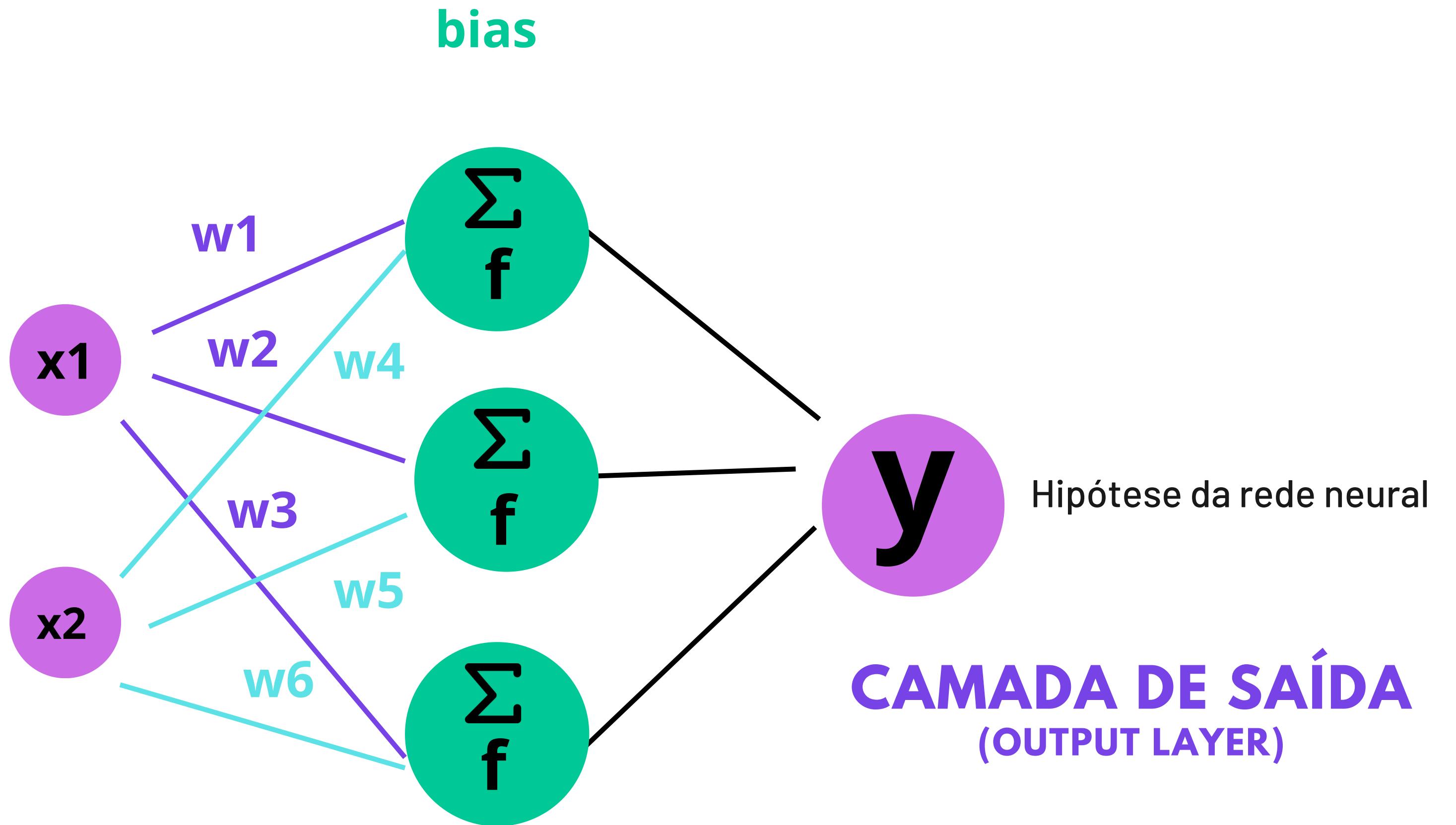
Bias é uma constante que ajuda o modelo se adaptar melhor aos dados fornecidos.

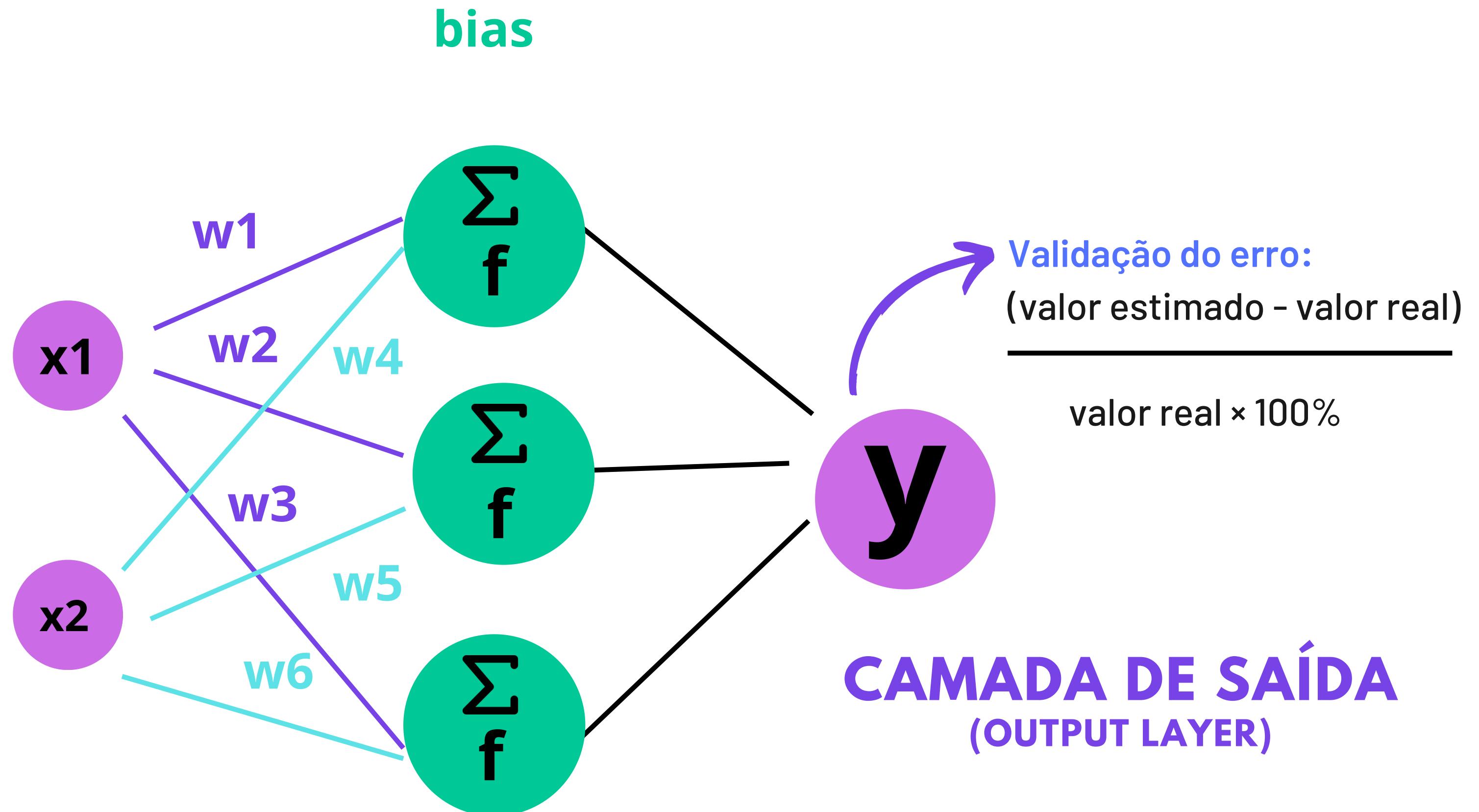


**CAMADA OCULTA  
(HIDDEN LAYER)**

**PROPAGAÇÃO**



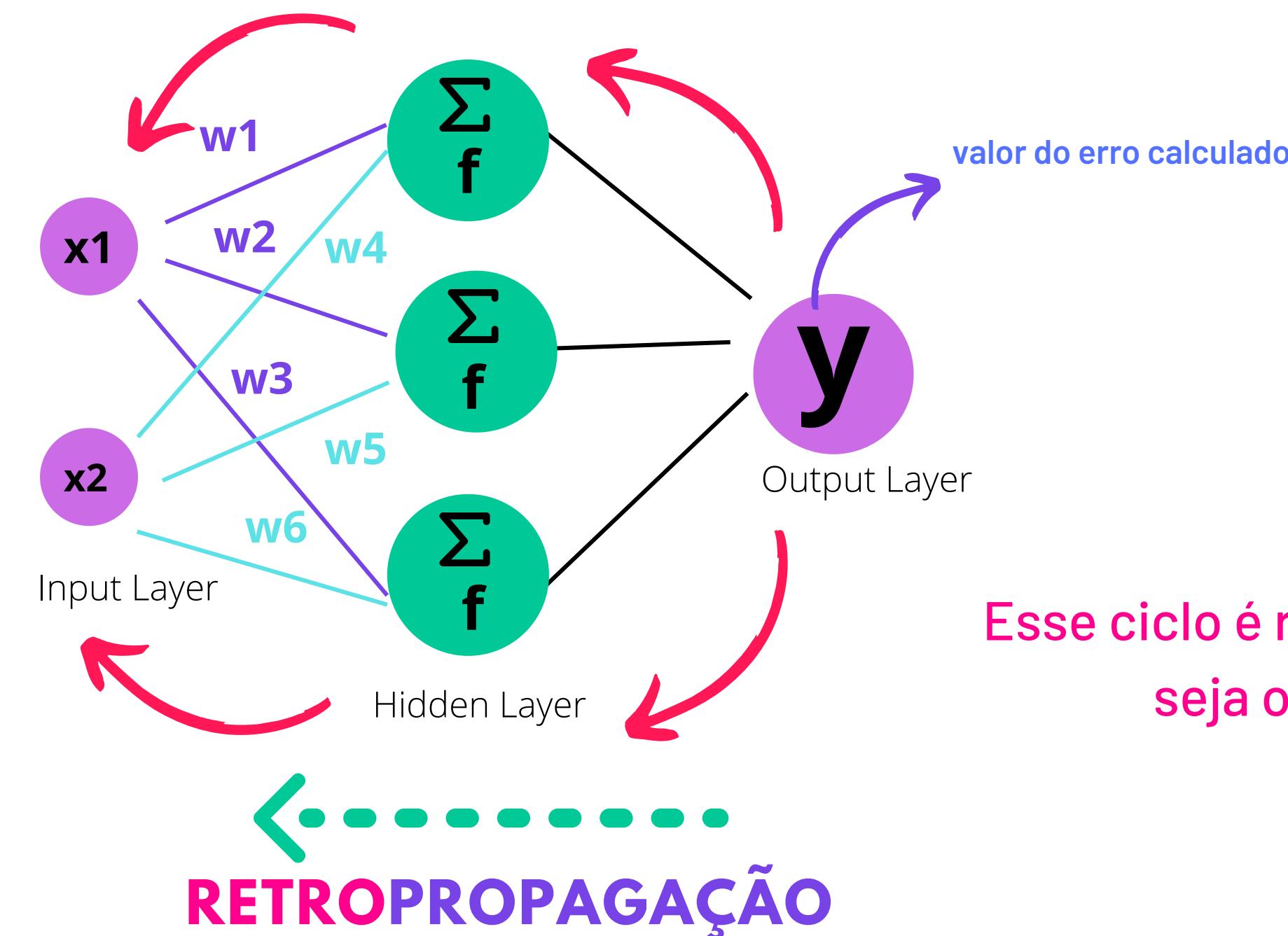




# AJUSTE DE PESOS DA REDE

Com base no valor de resultado do erro, vamos ter um parâmetro de quanto a nossa rede acertou o resultado.

Nem sempre esses erros vão ter valores bons. É nesse caso aonde entra a nossa retro propagação da rede para ajustar os pesos da rede neural e encontrar um valor ideal para nosso resultado de saída.



Esse ciclo é repetitivo até que o erro seja o menor possível.

# CRITÉRIOS DE PARADA DA REDE

Mas até quando devo ajustar meus pesos?



**Epochs:** São o total de vezes que são executadas as redes neurais. (Número máximo de épocas).

**Batch:** Conjunto de instâncias da rede neural ( número de exemplos de treinamento utilizados).

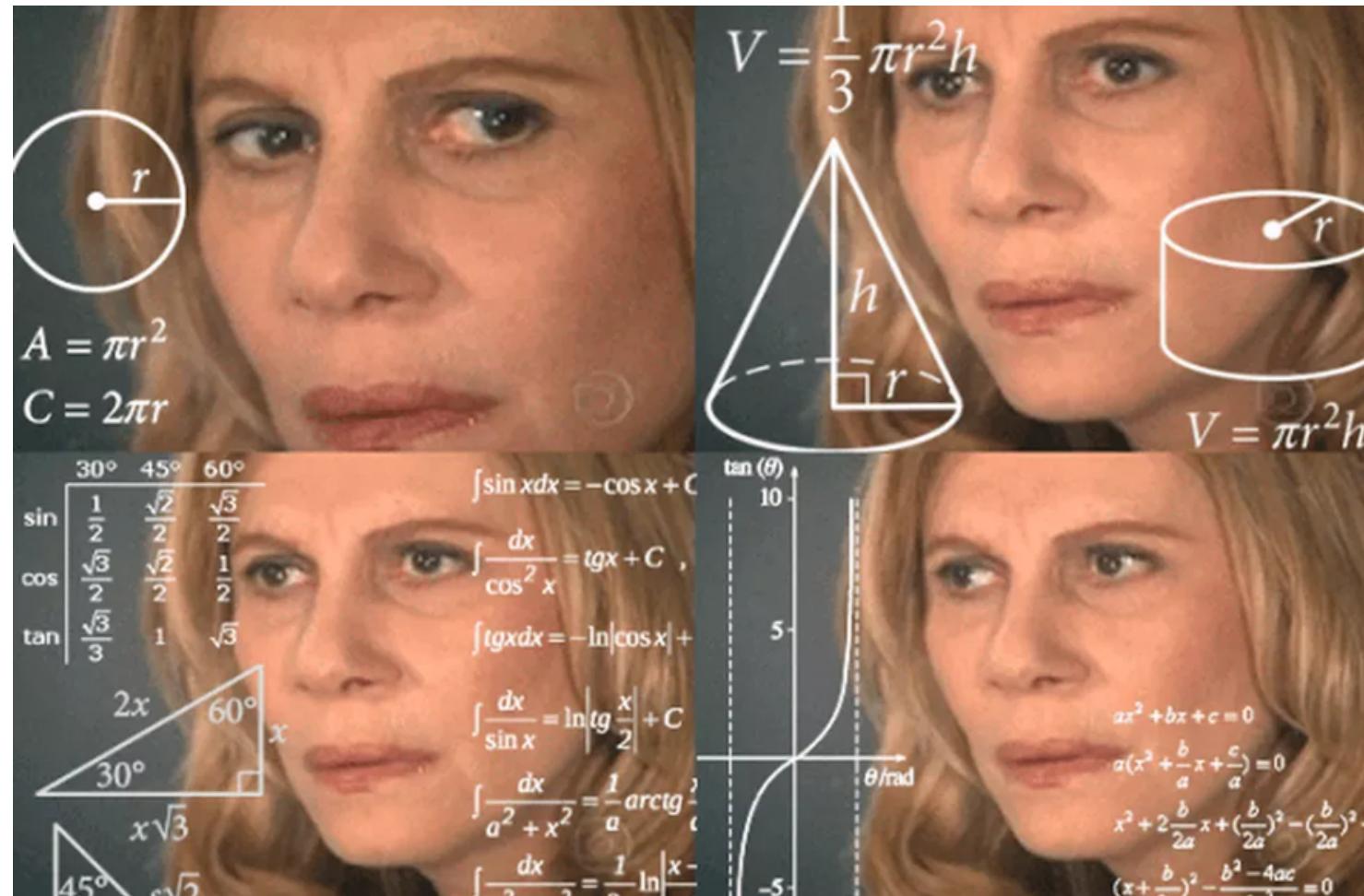
Valor limite à **taxa de erro mínima** previamente estabelecida.

# PODEMOS CONCLUIR QUE...

Uma rede neural pode ser considerada como **funções dentro de funções**.

As RNNs possuem uma “memória” que captura informações sobre o que foi calculado até agora.

A atualização dos pesos da rede é realizada com a regra da cadeia.



cálculo diferencial  
Ajuda a calcular a derivada de funções compostas.

Novo valor do neurônio

$$aw_{ji} = ng_{jy_i}$$

Novo valor do peso

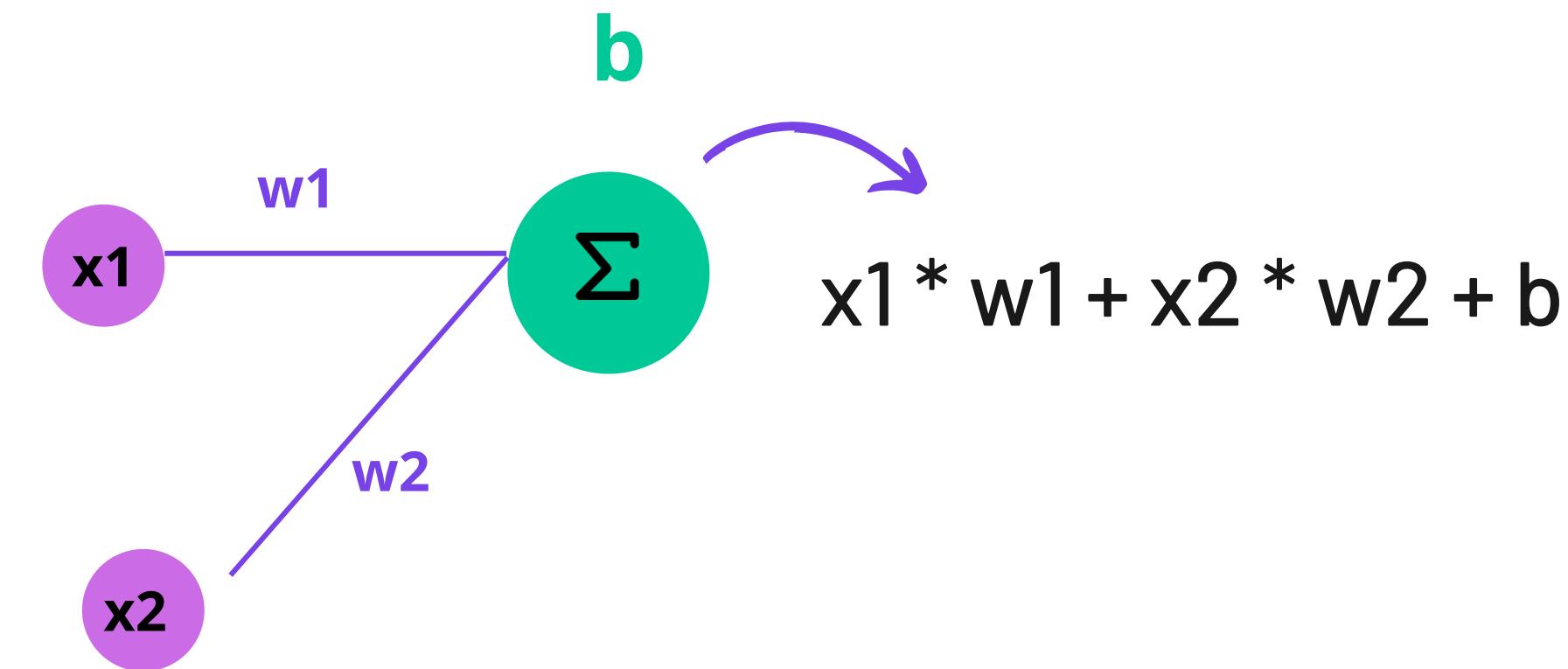
Razão do aprendizado  
Valor do peso

Sinal de entrada

Erro da equação anterior multiplicado pela derivada da função de ativação

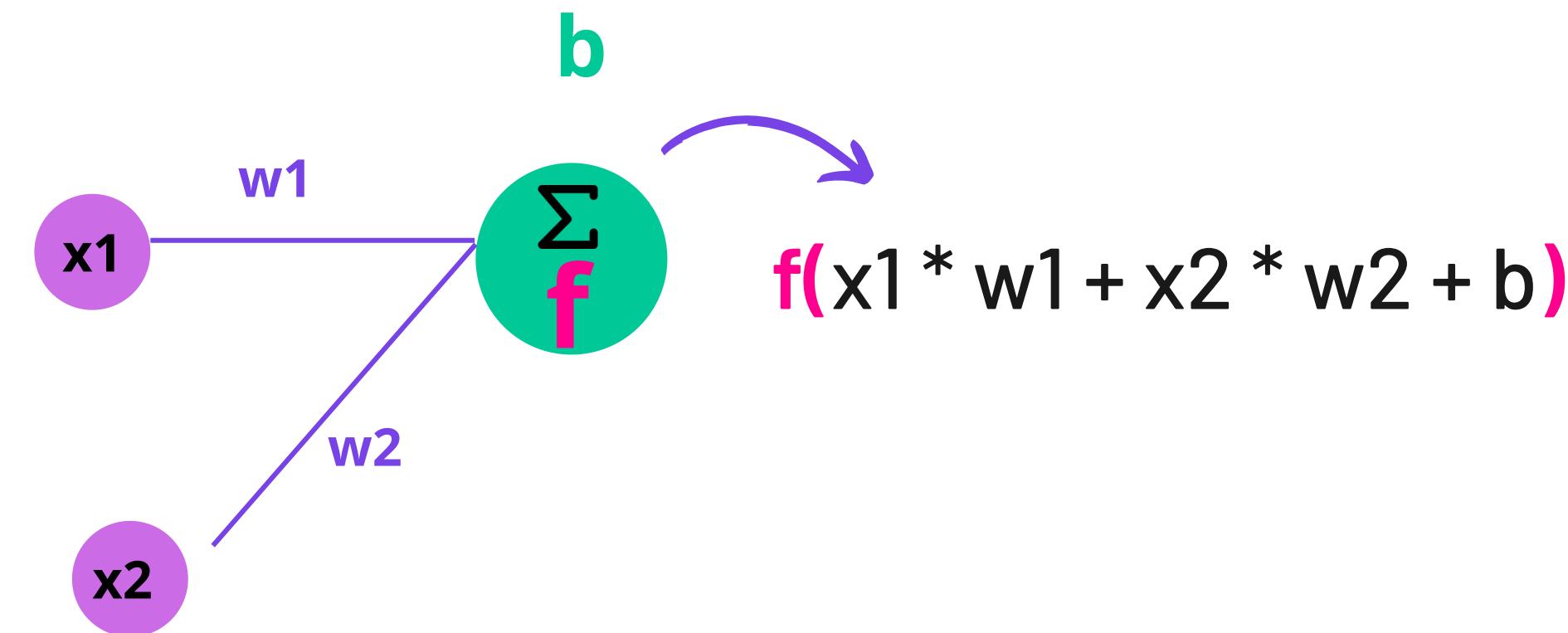
# A MATEMÁTICA DO BACKPROPAGATION

Soma ponderada do neurônio:



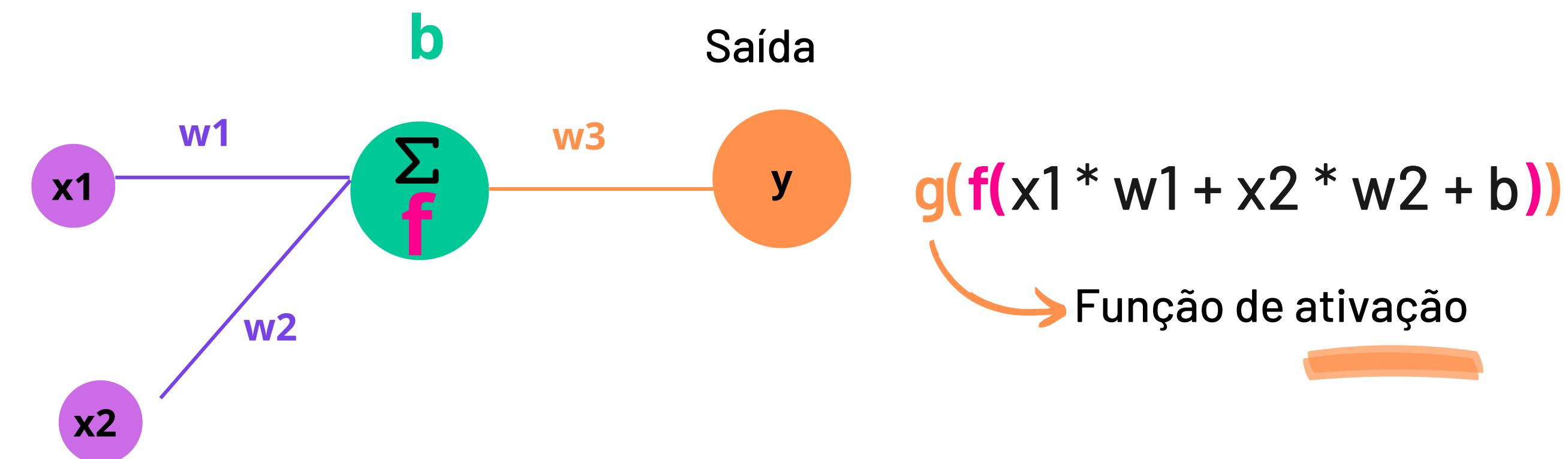
# A MATEMÁTICA DO BACKPROPAGATION

Os pesos e a somatório do neurônio estão envolvidos por uma função de ativação:



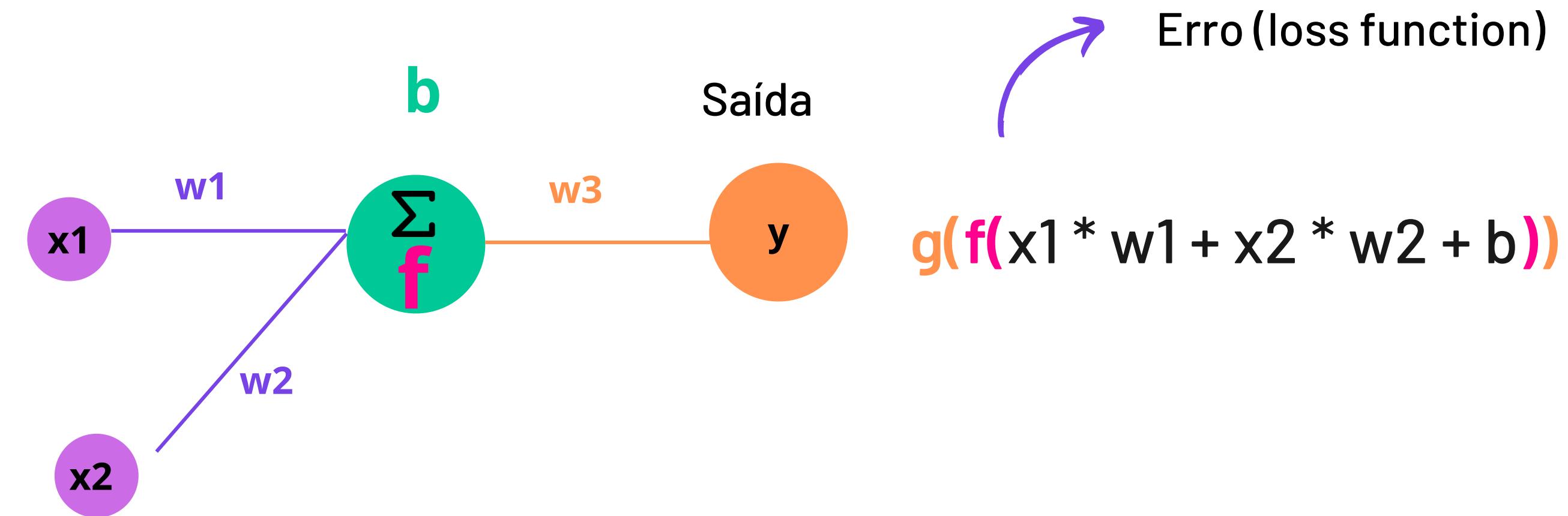
# A MATEMÁTICA DO BACKPROPAGATION

Entradas da camada de saída:



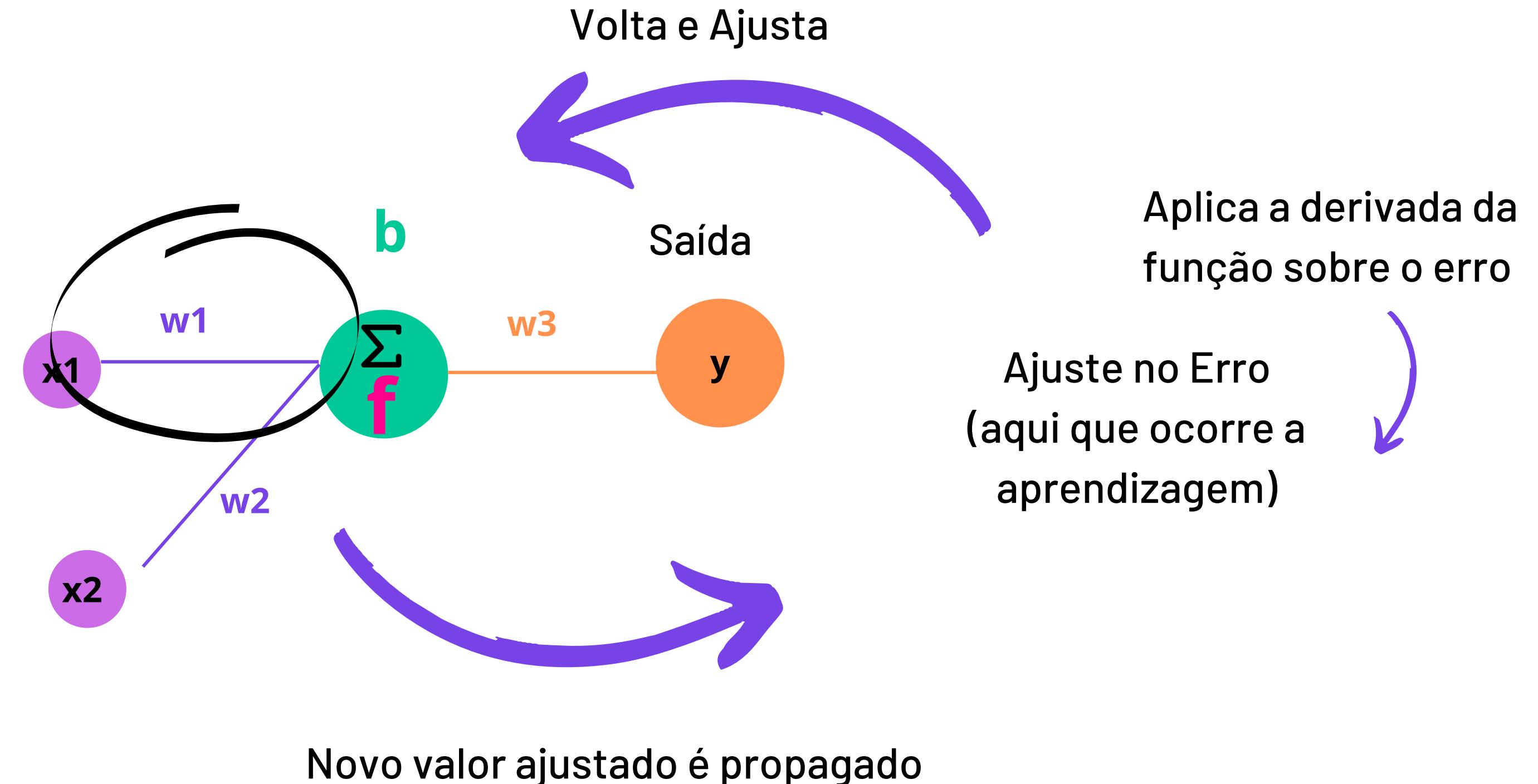
# A MATEMÁTICA DO BACKPROPAGATION

São gerados os erros para retro propagar:

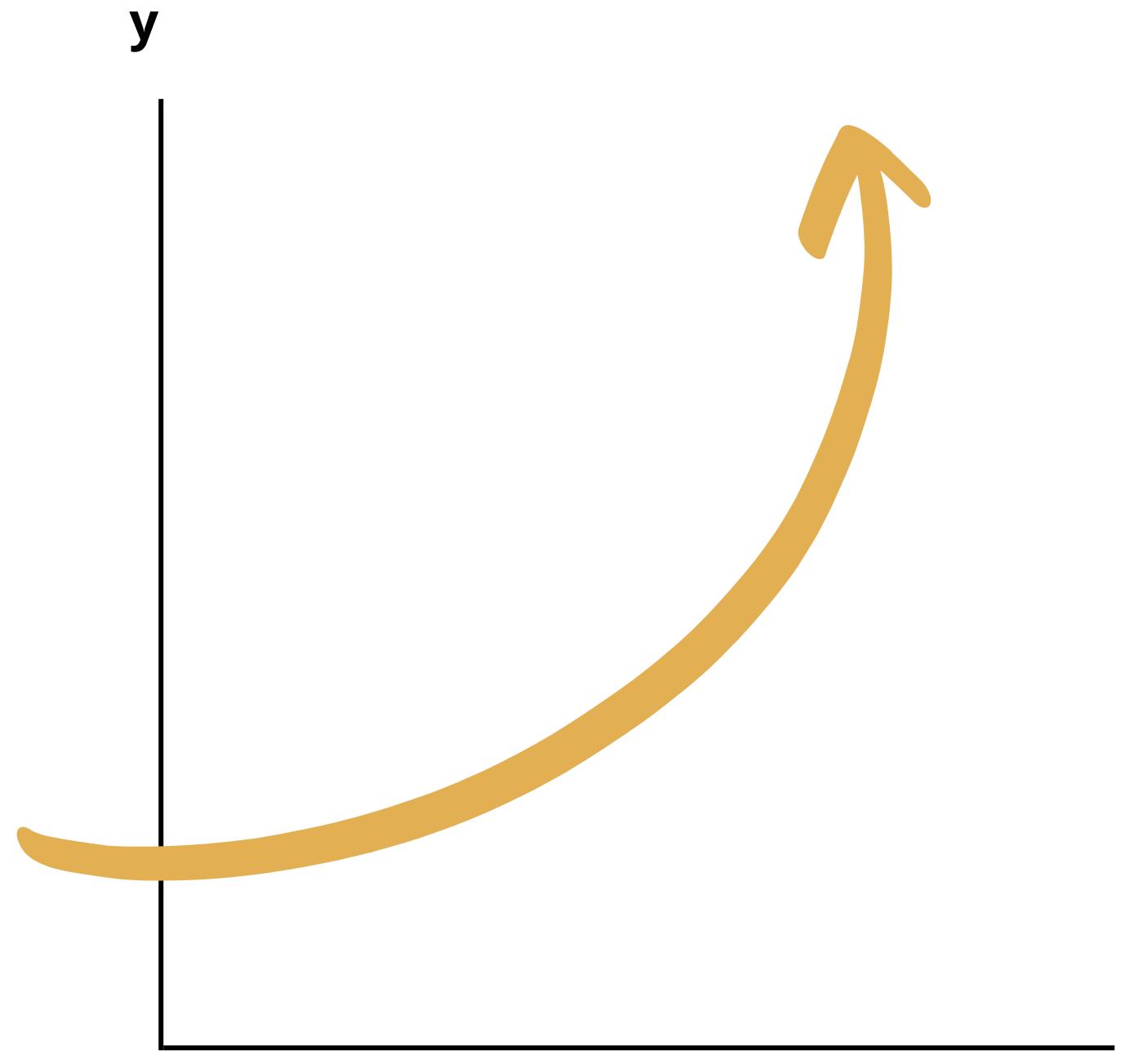


# REGRA DA CADEIA:

Derivação da função de erro em cada peso dentro da rede neural:

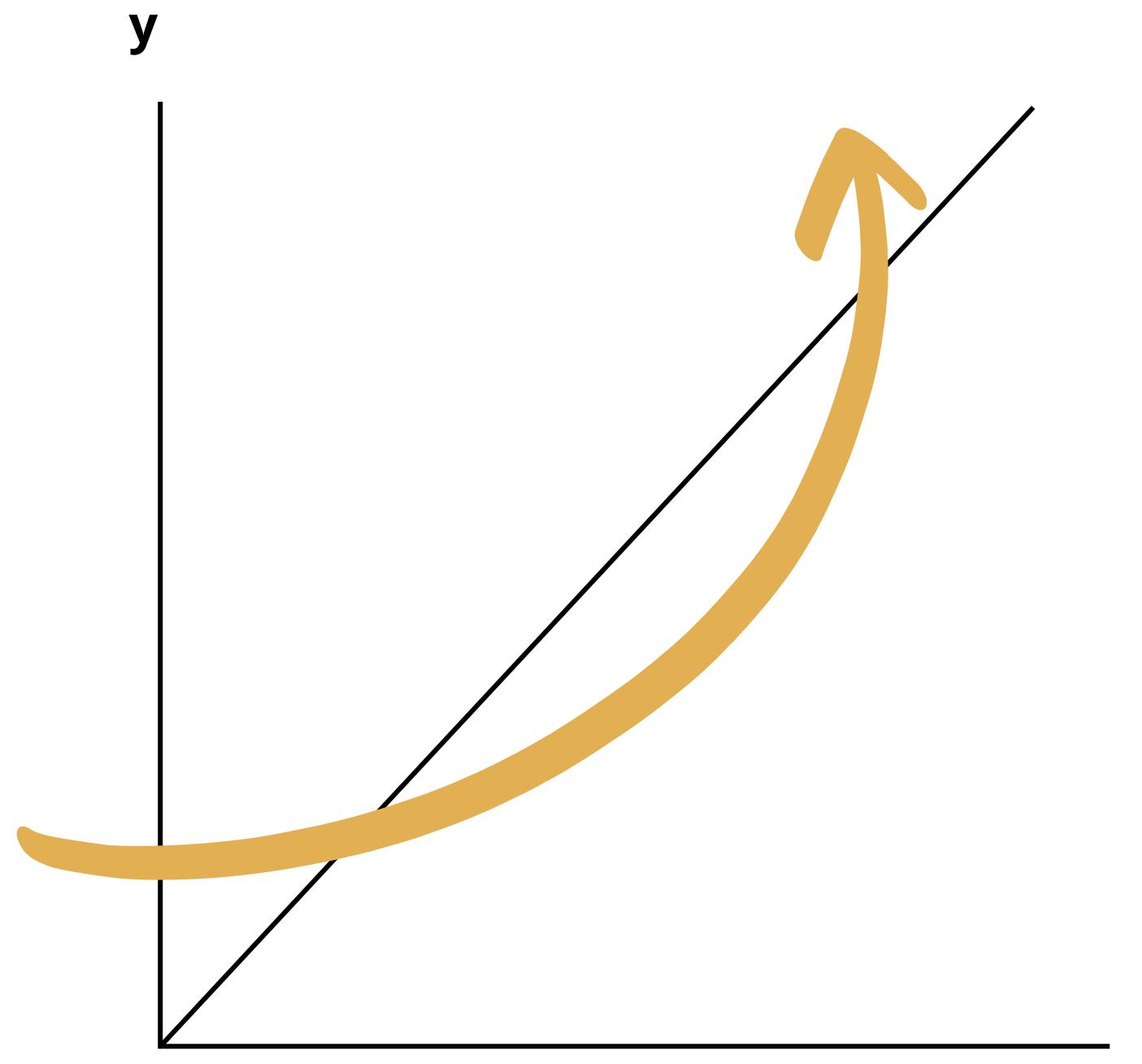


# COMO FUNCIONA A DERIVADA DE UMA FUNÇÃO?

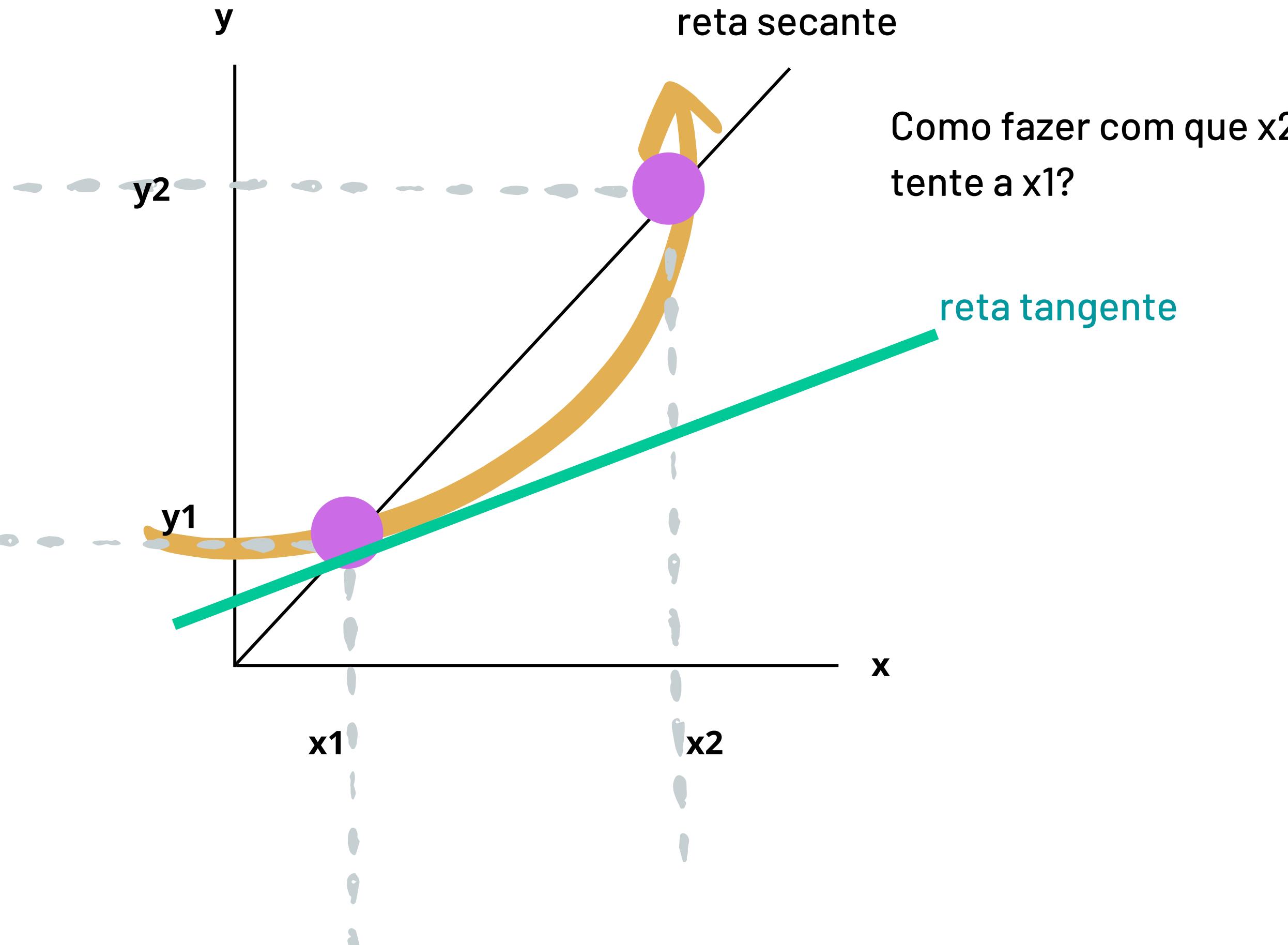


Para entender como funciona o ajuste dos pesos, vamos entender a derivada de uma função.

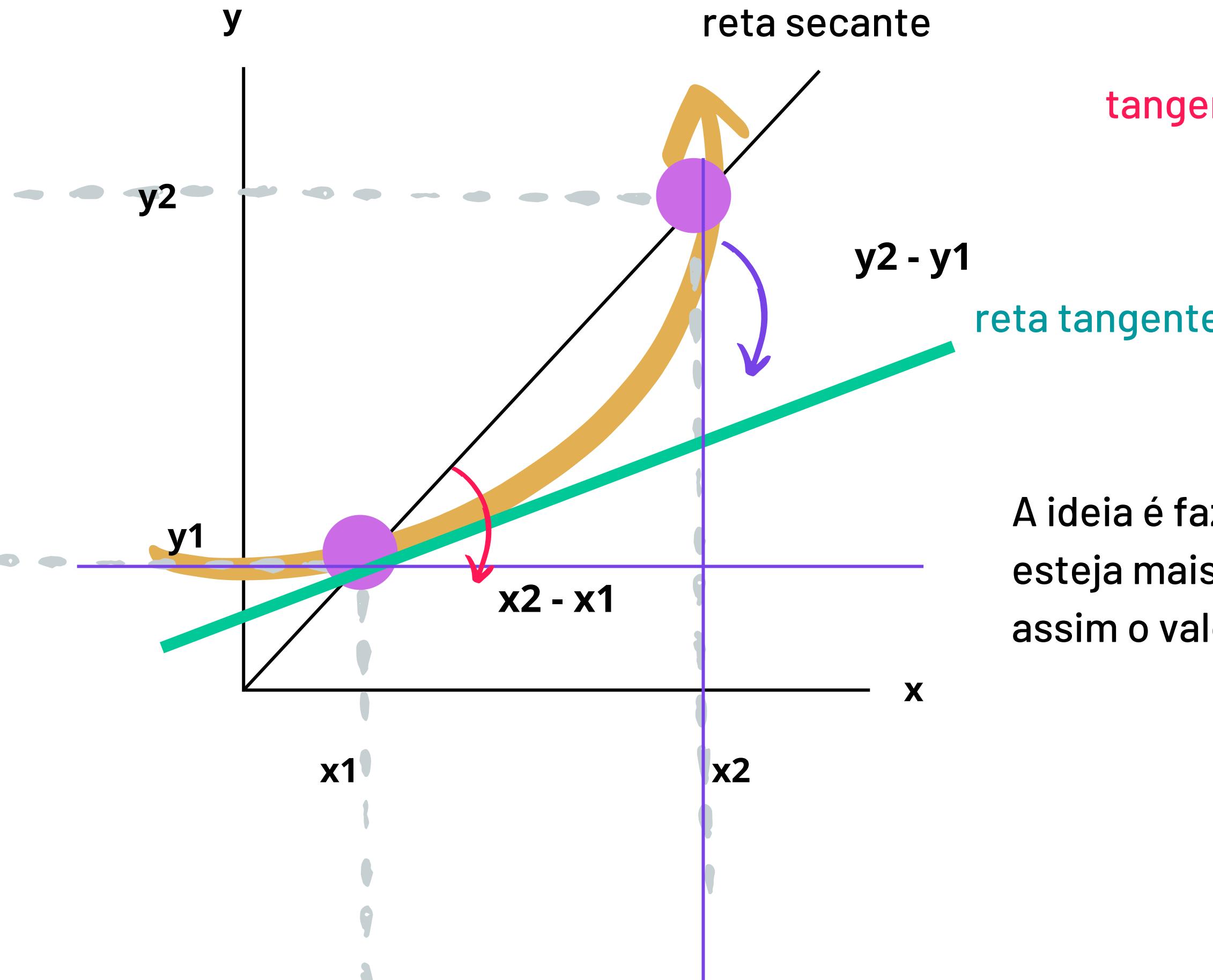
# COMO FUNCIONA A DERIVADA DE UMA FUNÇÃO?



# COMO FUNCIONA A DERIVADA DE UMA FUNÇÃO?



# COMO FUNCIONA A DERIVADA DE UMA FUNÇÃO?



$$\text{tangente} = \frac{\text{cateto oposto}}{\text{cateto adjacente}}$$

A ideia é fazer com que o ponto  $(x_2, y_2)$  esteja mais próximo de  $(x_1, y_1)$ , trazendo assim o valor do peso mais próximo do ideal.

# FUNÇÕES DE ATIVAÇÕES:

O objetivo de uma função de ativação é realizar a transformação não linear dos neurônios, aliás uma rede neural sem a função de ativação é essencialmente um modelo de regressão linear.

A função de ativação permite que as mudanças realizadas nos pesos e bias causem uma alteração na saída final do modelo (output).

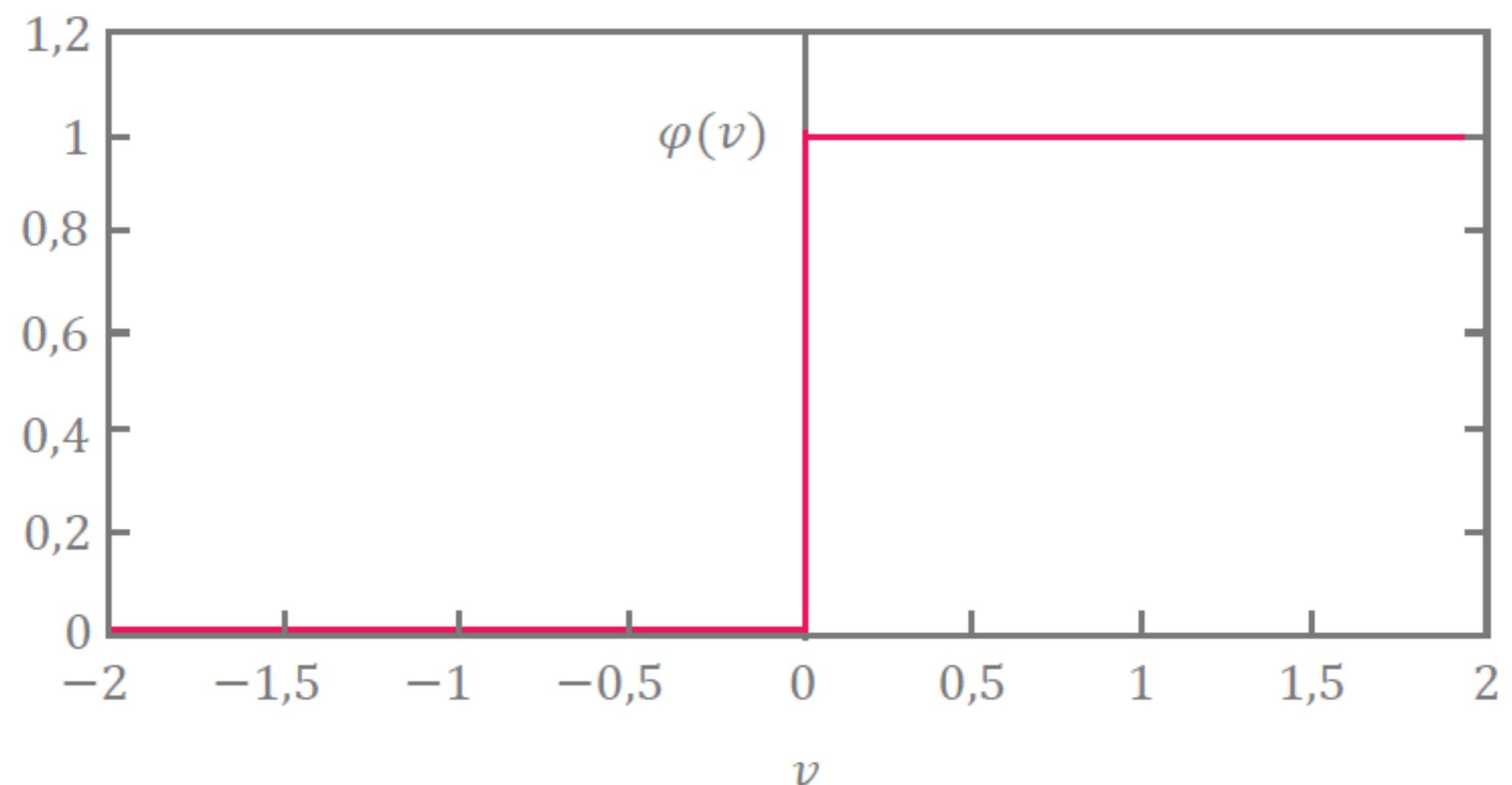
$$Y = \text{Activation}(\sum(\text{weight} * \text{input}) + \text{bias})$$

# TIPOS DE FUNÇÕES DE ATIVAÇÕES:

São as funções de ativação de uma rede neural que **propagam a ativação dos neurônios**. Vamos aprender as melhores funções para essa ativação.

## LIMIAR (THRESHOLD)

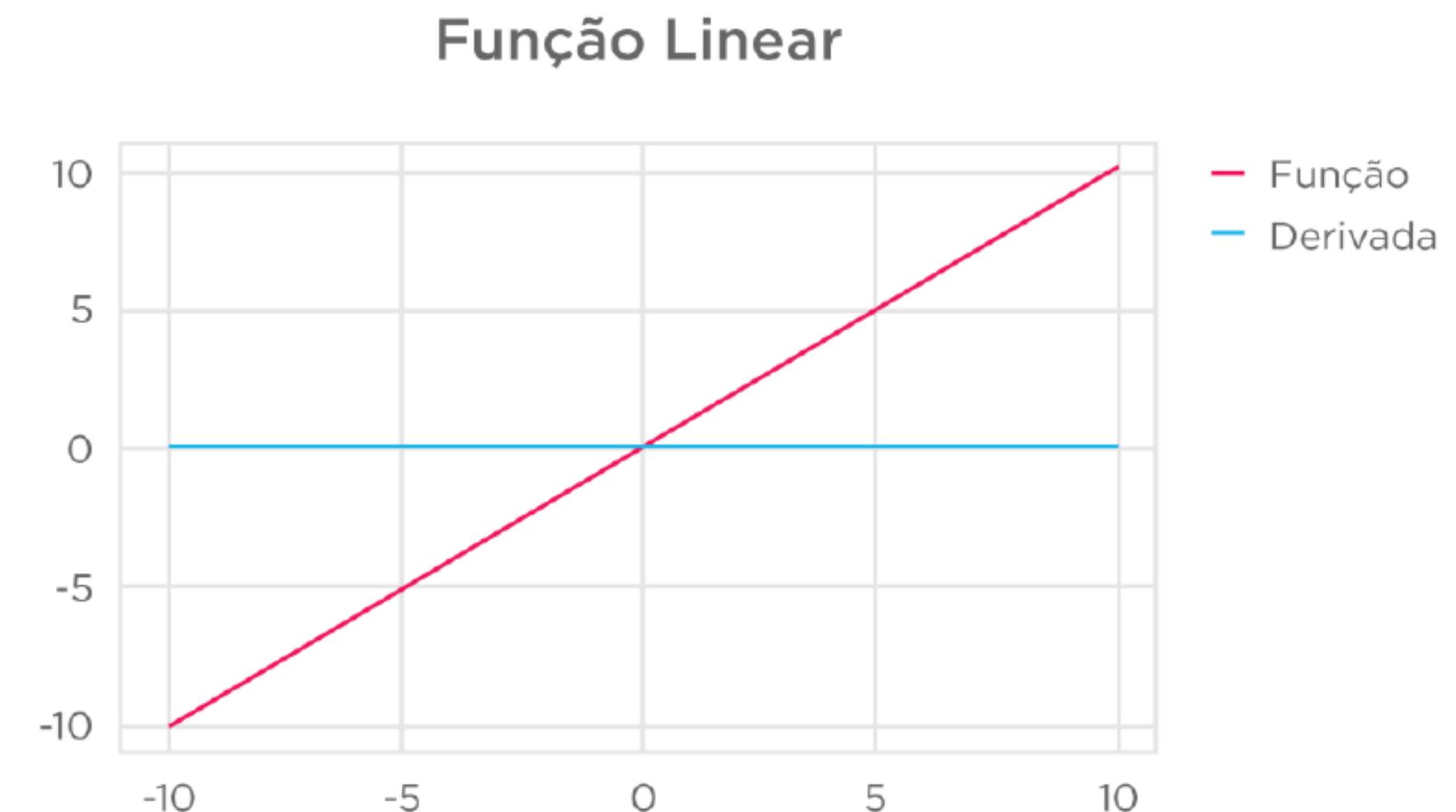
- Retorna valores binários.
- Se  $x < 0$  então 0. Se  $x > 0$  então 1.
- Pode ser utilizada para classificadores binários (Sim/Não).



# TIPOS DE FUNÇÕES DE ATIVAÇÕES:

## LINEAR

- A derivada de uma função linear é constante, isto é, não depende do valor de entrada  $x$ . Isso significa que toda vez que fazemos backpropagation, o gradiente seria o mesmo.(o erro não é melhorado).
- A função linear pode ser ideal para tarefas simples, onde a interpretabilidade é altamente desejada.

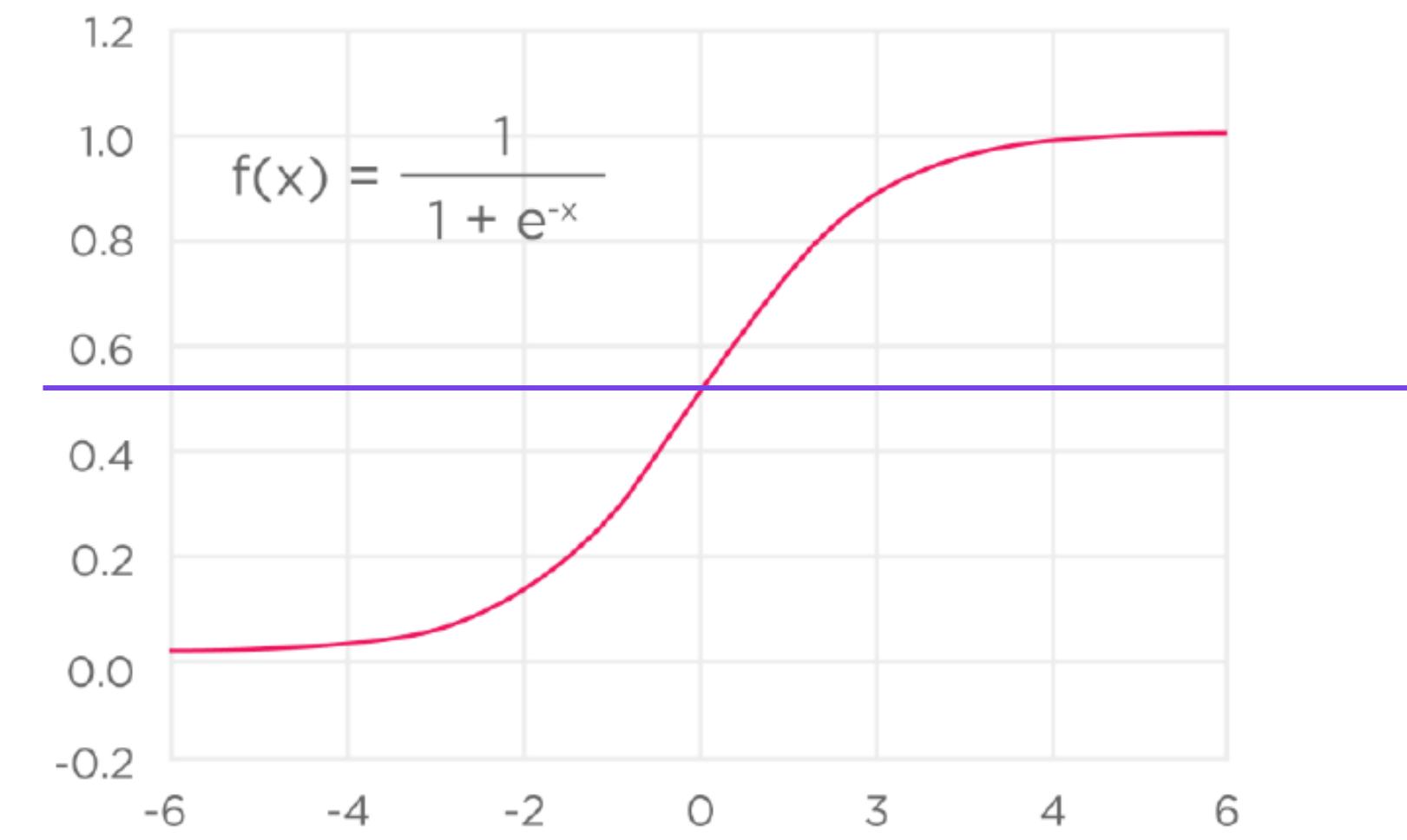


# TIPOS DE FUNÇÕES DE ATIVAÇÕES:

## SIGMÓIDE

- Saídas com 0 ou 1 (retorna dados binários).
- Possui uma suavização.
- Função não linear.
- A função essencialmente tenta empurrar os valores de Y para os extremos.
- Trabalha apenas com valores positivos.
- O resultado da função pode ser uma combinação de várias variáveis.
- Funciona muito bem para classificar os valores para até duas classes específicas.
- Pode apresentar problema quando gradientes se tornam muito pequenos. Isso significa que o gradiente está se aproximando de zero e a rede não está realmente aprendendo.

$$R^n \rightarrow [0,1]$$

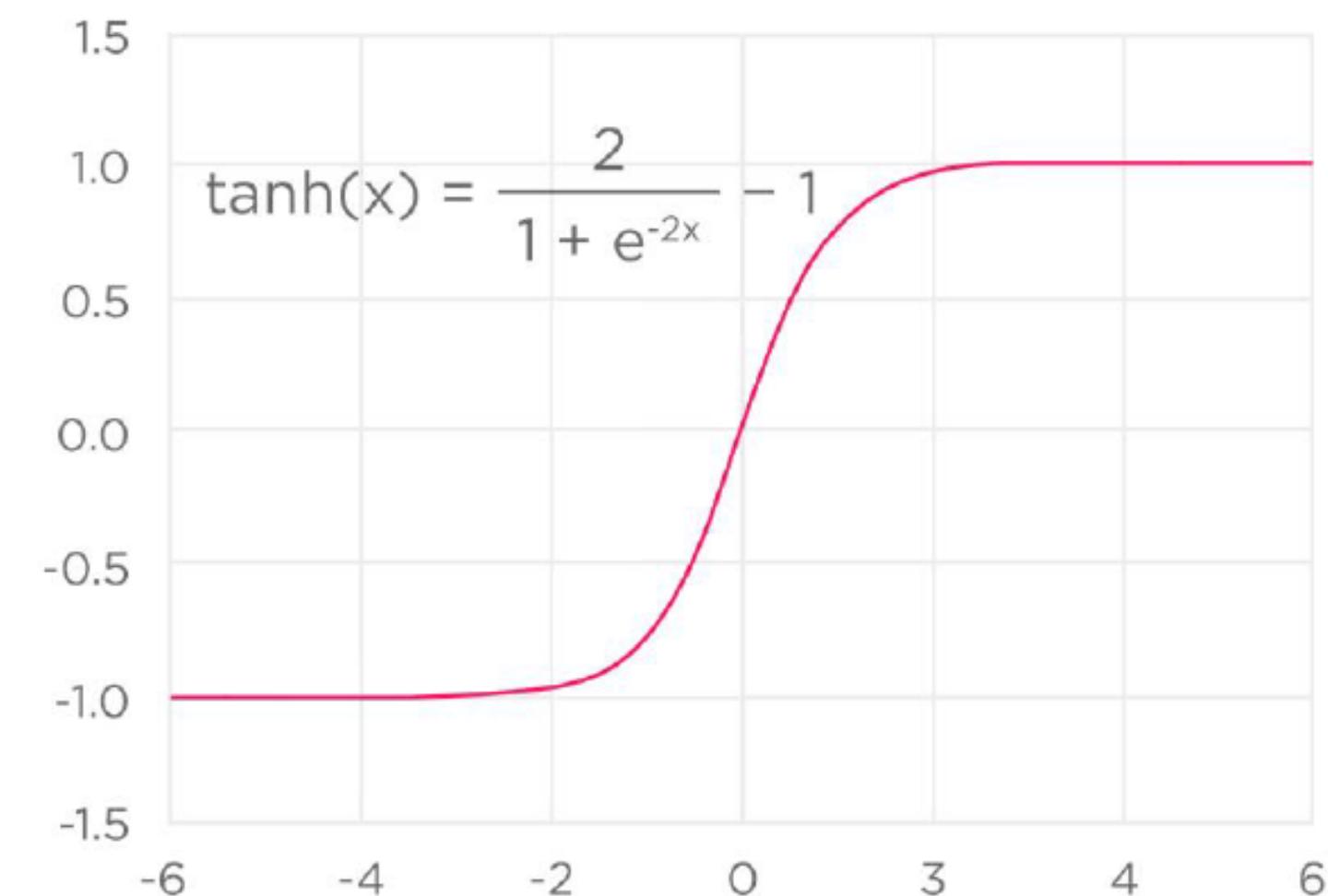


# TIPOS DE FUNÇÕES DE ATIVAÇÕES:

## TANGENTE HIPERBÓLICA (TANH)

- A função tanh é uma versão escalonada da função sigmóide.
- Varia de -1 a 1.
- Função não linear.
- A TanH se aproxima mais da identidade, sendo assim uma alternativa mais atraente do que a sigmoide para servir de ativação às camadas ocultas das RNAs.

$$R^n \rightarrow [-1,1]$$

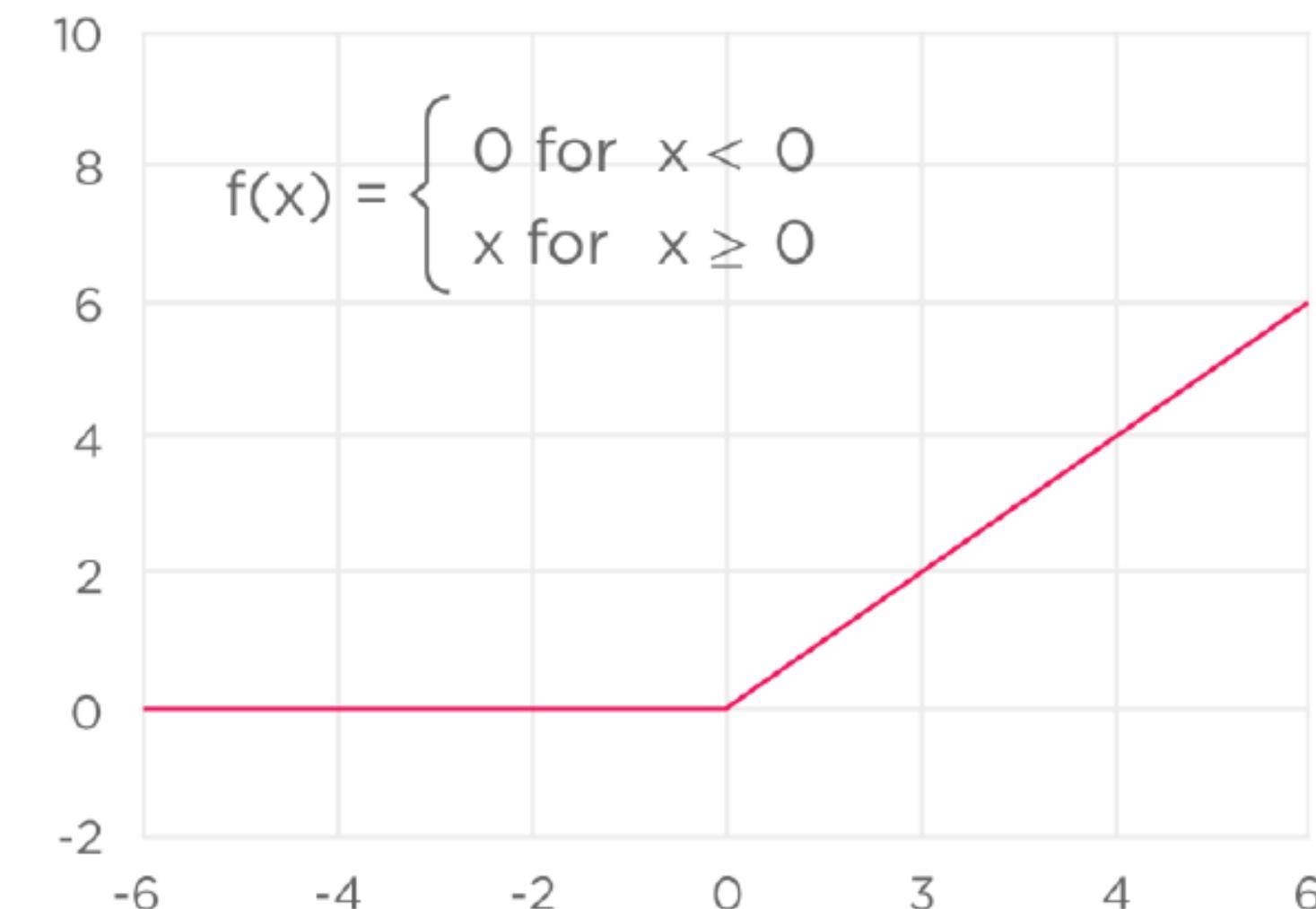


# TIPOS DE FUNÇÕES DE ATIVAÇÕES:

## RELU (UNIDADE LINEAR RETIFICADA)

- A função ReLU é não linear.
- Não ativa todos os neurônios ao mesmo tempo. Se você olhar para a função ReLU e a entrada for negativa, ela será convertida em zero e o neurônio não será ativado.
- Menor complexidade computacional.
- ReLU também pode ter problemas com os gradientes que se deslocam em direção a zero.
- A função ReLU é uma função de ativação geral e é usada na maioria dos casos atualmente.

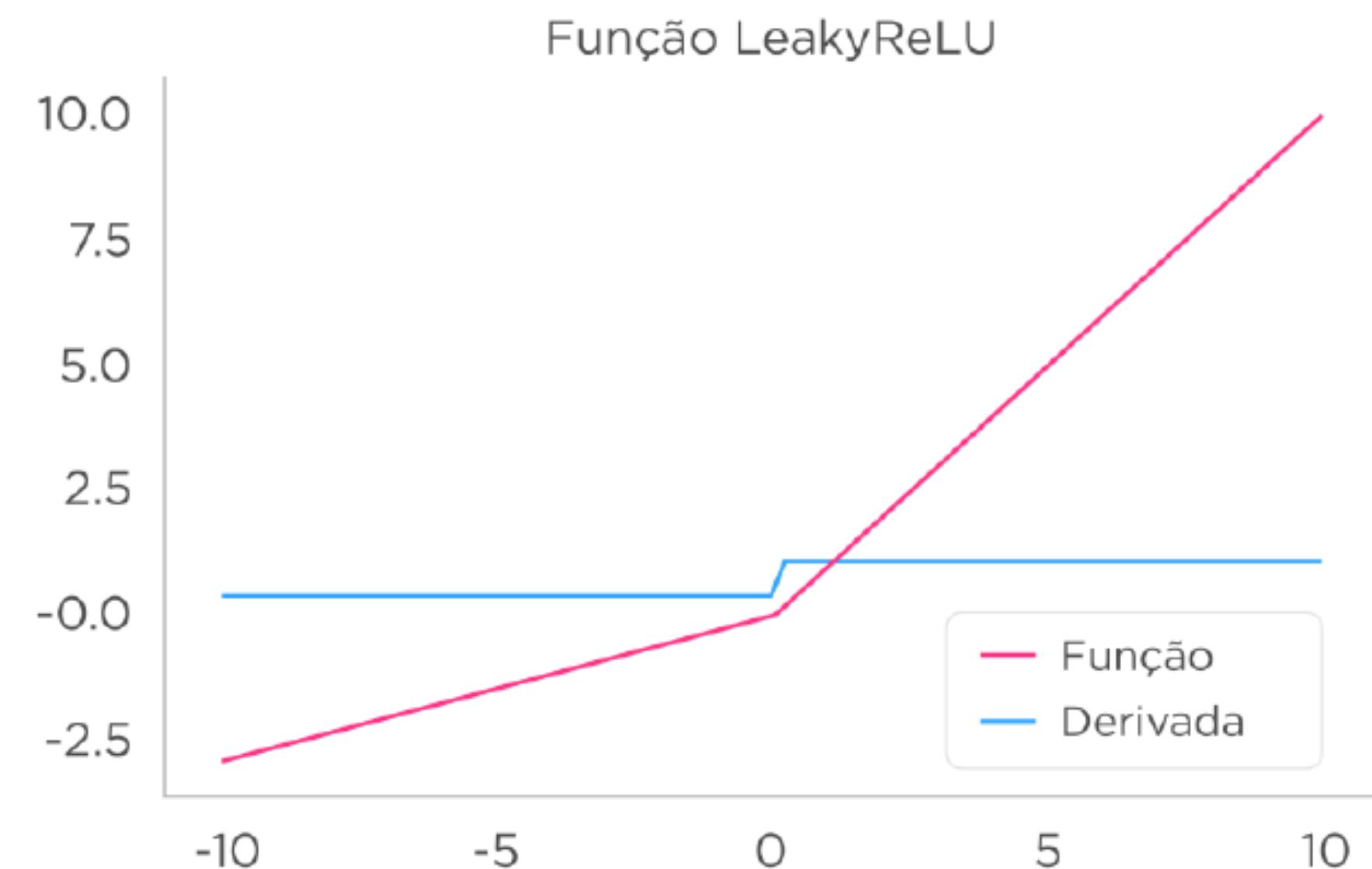
$$R^n \rightarrow R_+^n$$



# TIPOS DE FUNÇÕES DE ATIVAÇÕES:

## LEAKY RELU

- Versão melhorada da ReLU.
- Faz a ativação para valores  $x < 0$ .
- Para valores menores que 0, é definido um pequeno componente linear (0,01).
- A principal vantagem de substituir a linha horizontal é remover o gradiente zero.



# TIPOS DE FUNÇÕES DE ATIVAÇÕES:

## SOFTMAX

- Também é um tipo de função sigmóid.
- Além de transformar as saídas para 0 e 1 também divide pela soma das saídas.
- Atribui a probabilidade de resultado pertencer a uma determinada classe.
- A função softmax é idealmente usada na camada de saída do classificador, onde estamos tentando gerar as probabilidades para definir a classe de cada entrada.
- Ideal quando estamos tentando lidar com várias classes.



# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?

Monitoramento de carteira

Análise de crédito

Cliente é ativo?

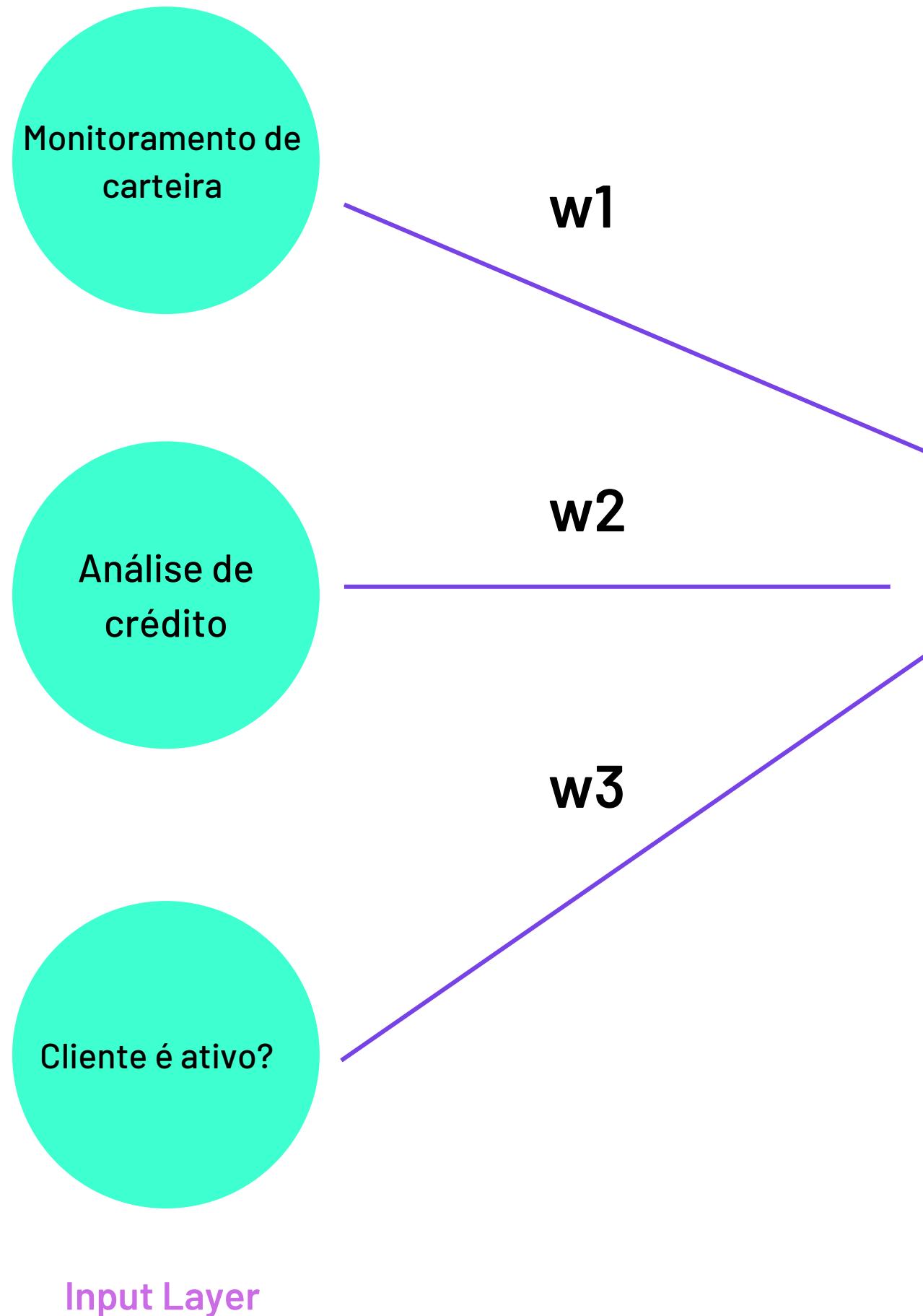
Input Layer

Quero classificar se o cliente vai pagar a fatura desse mês.

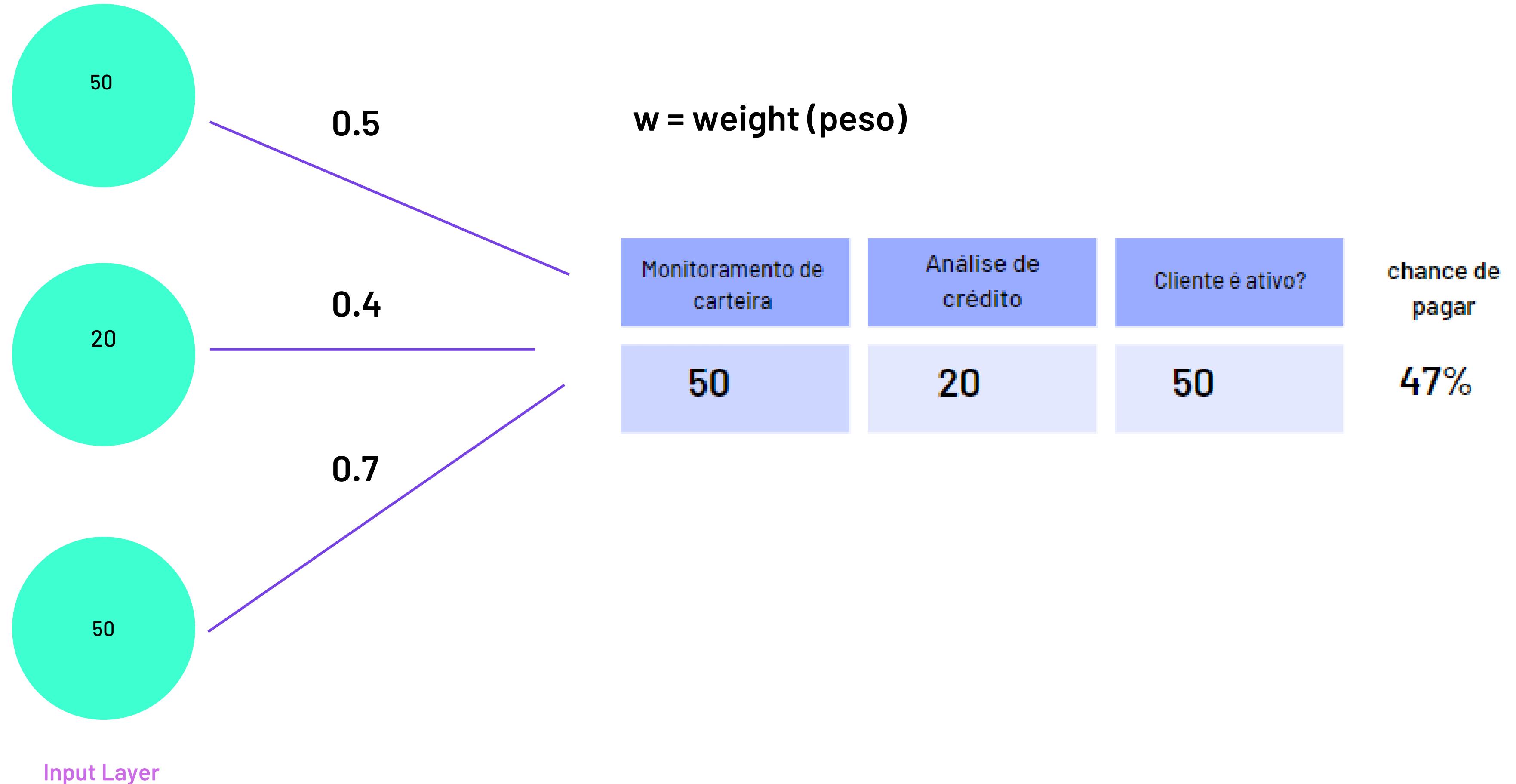
Tenho uma base histórica de pagamento de clientes:

Monitoramento de carteira	Análise de crédito	Cliente é ativo?	chance de pagar
50	20	50	47%
10	10	20	42%
80	90	100	89%

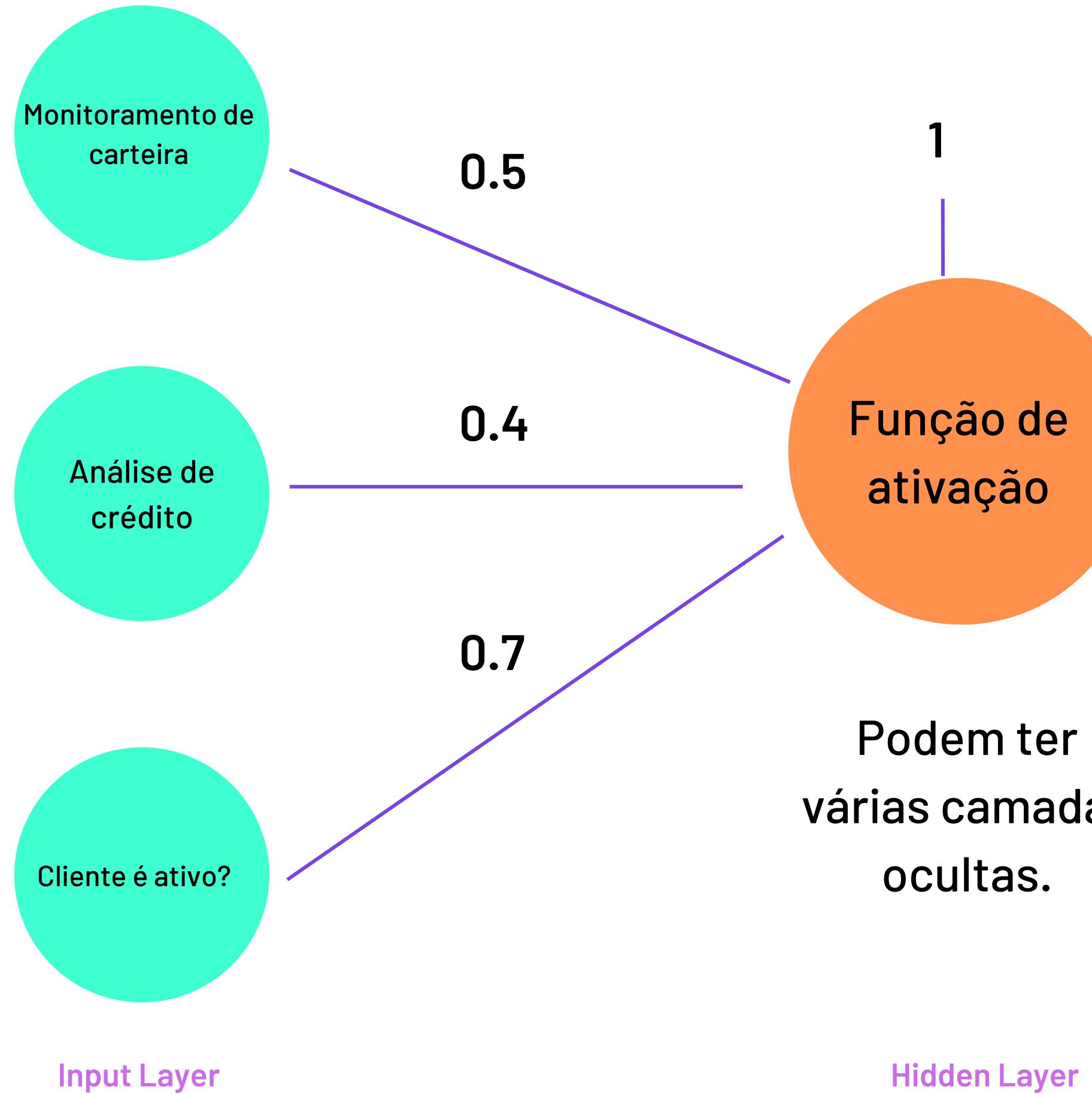
# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?



# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?

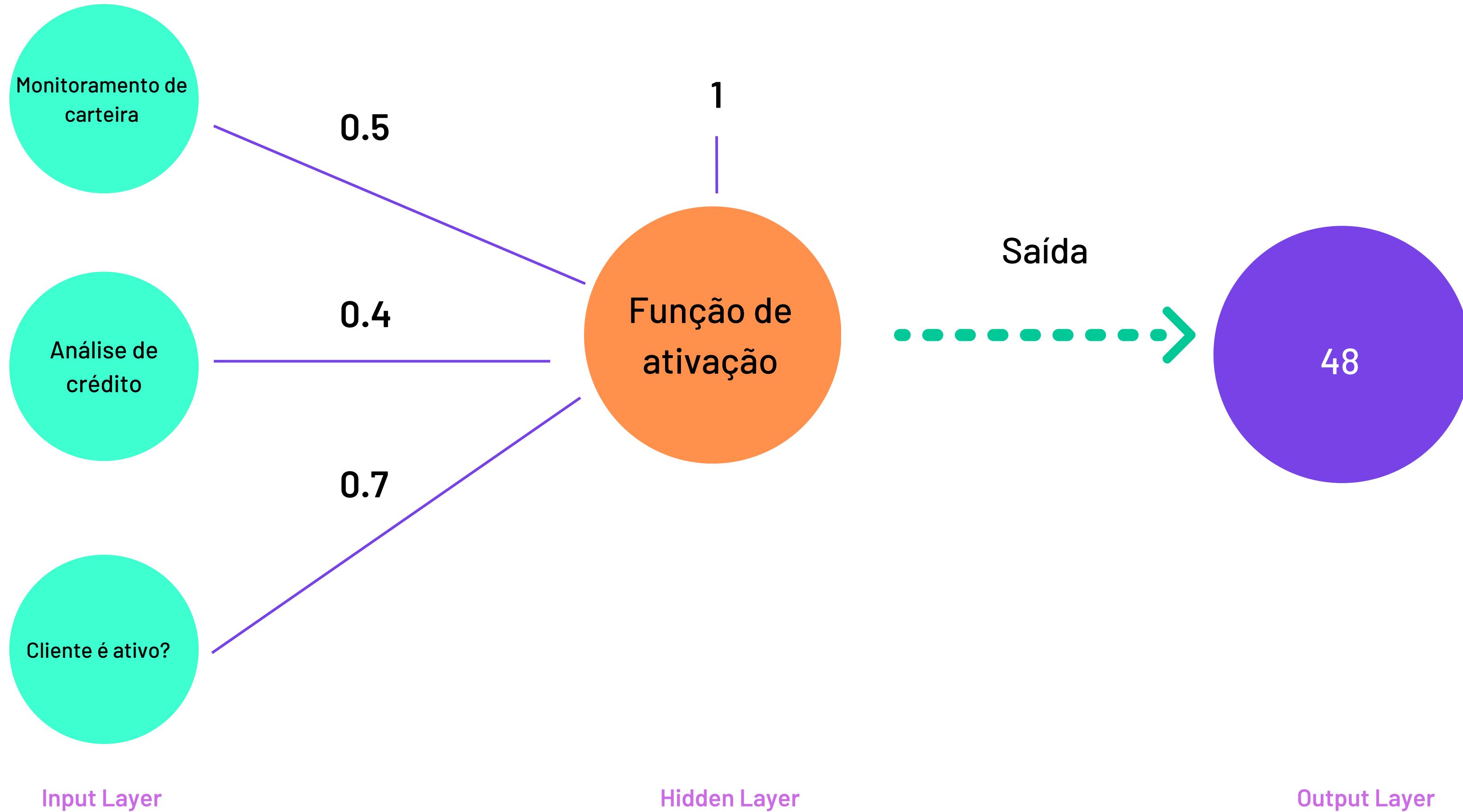


# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?

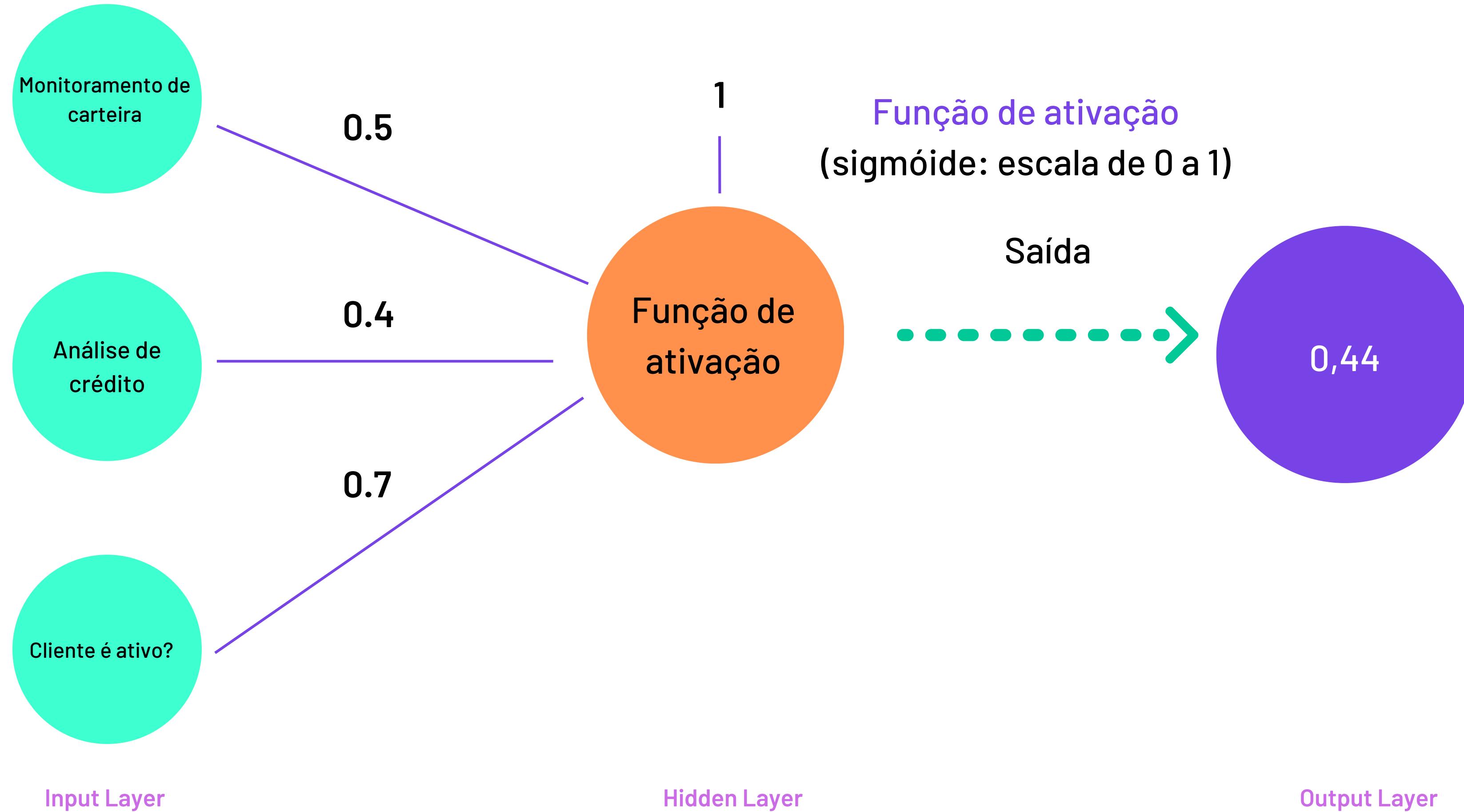


$$50 \cdot 0.5 + 20 \cdot 0.4 + 20 \cdot 0.7 + 1 = \\ 25 + 8 + 14 + 1 = 48$$

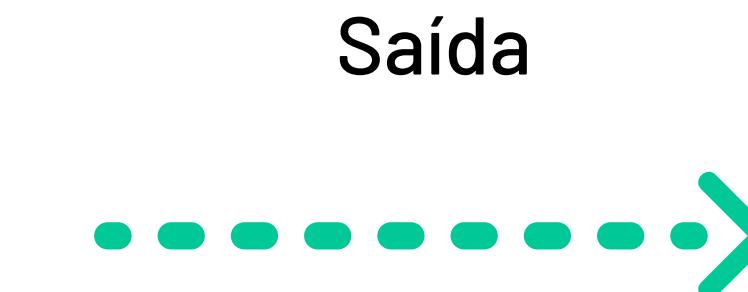
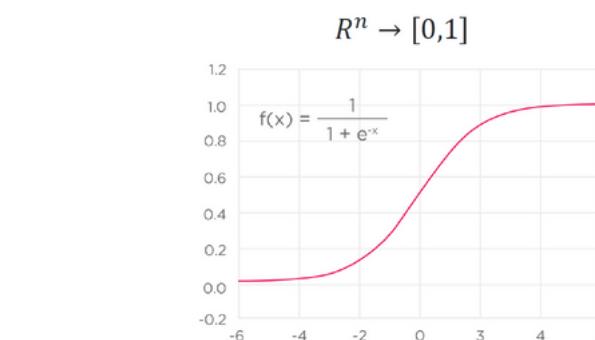
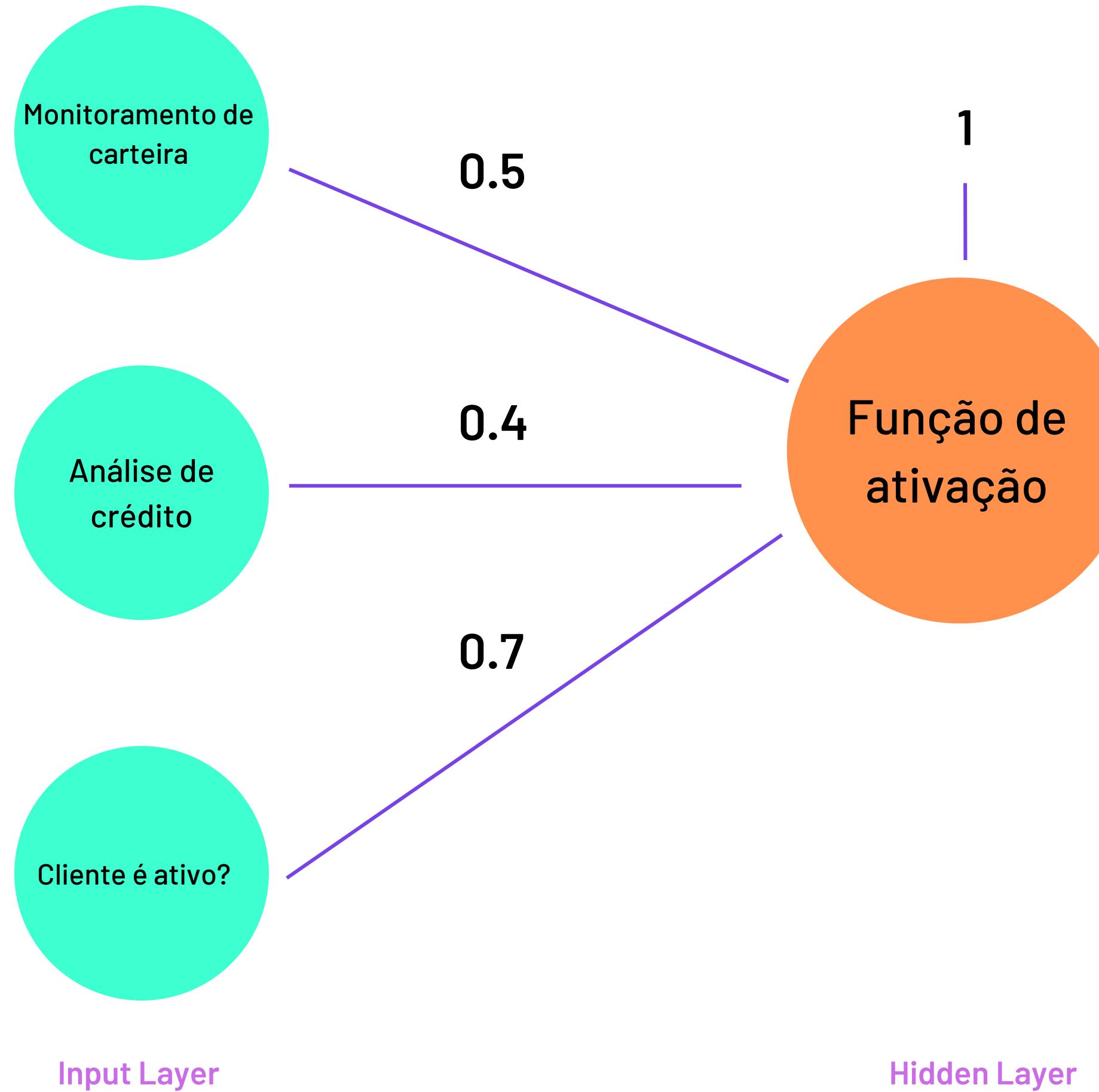
# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?



# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?



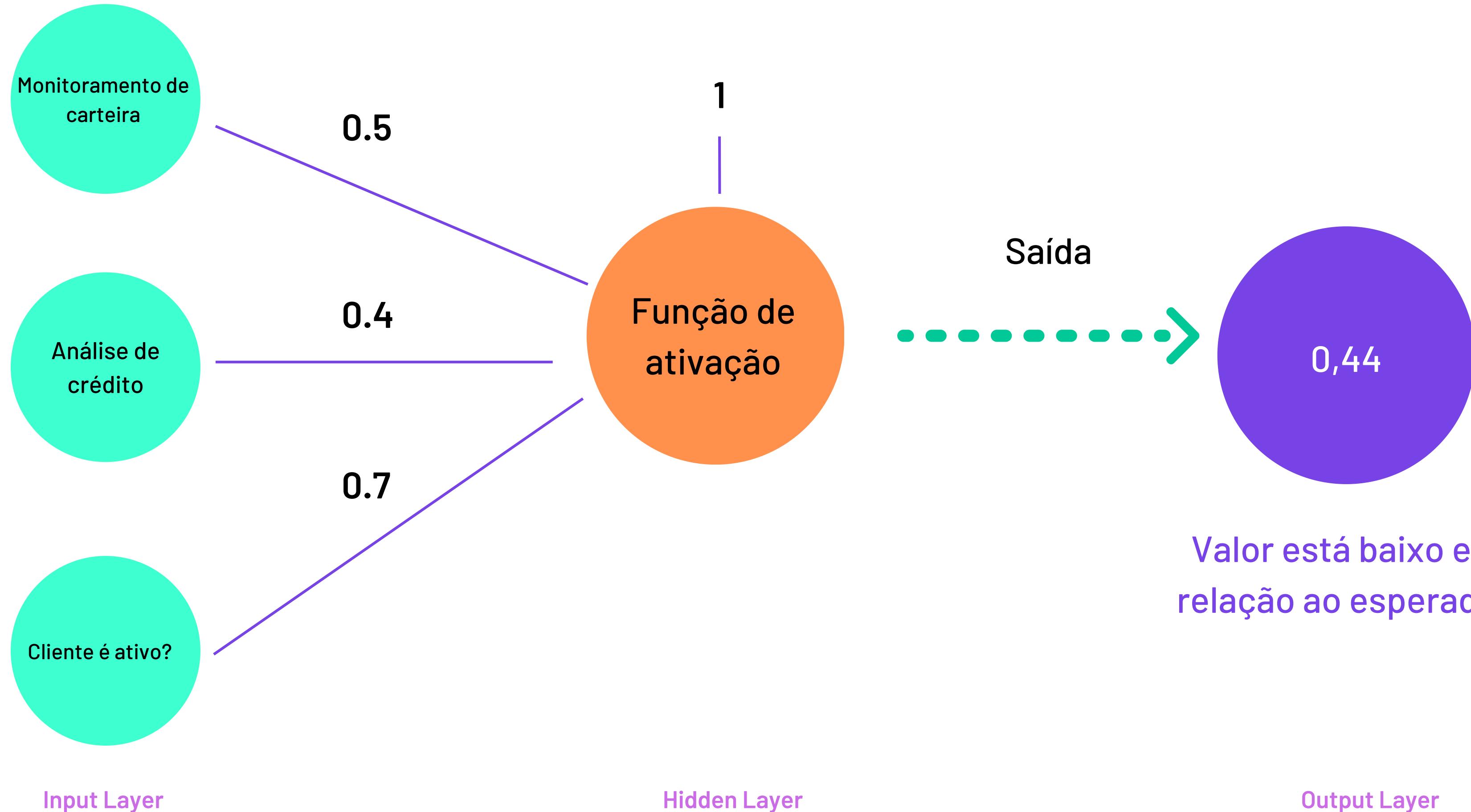
# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?



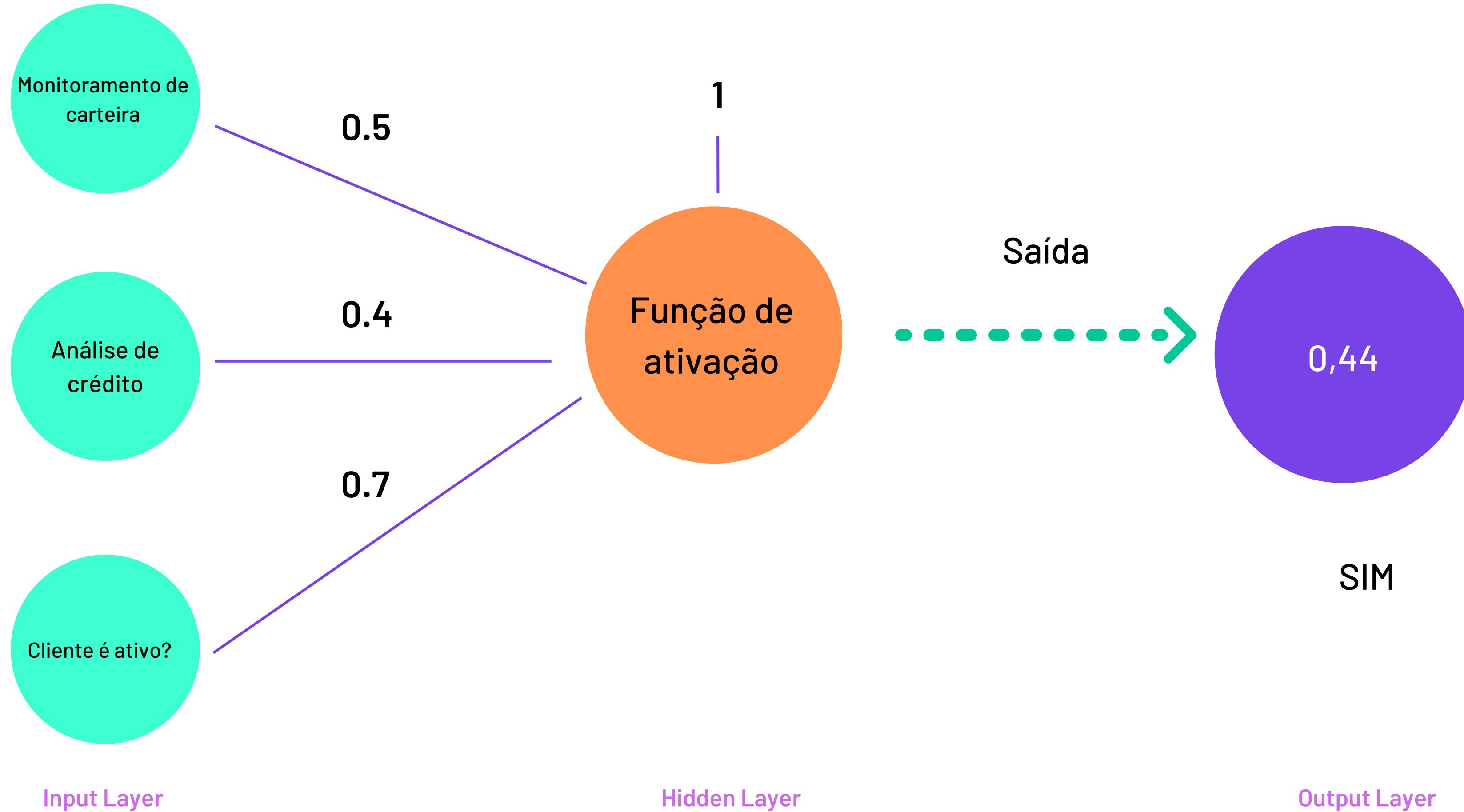
Submeter o valor sobre a loss function (cálculo de erro)

Output Layer

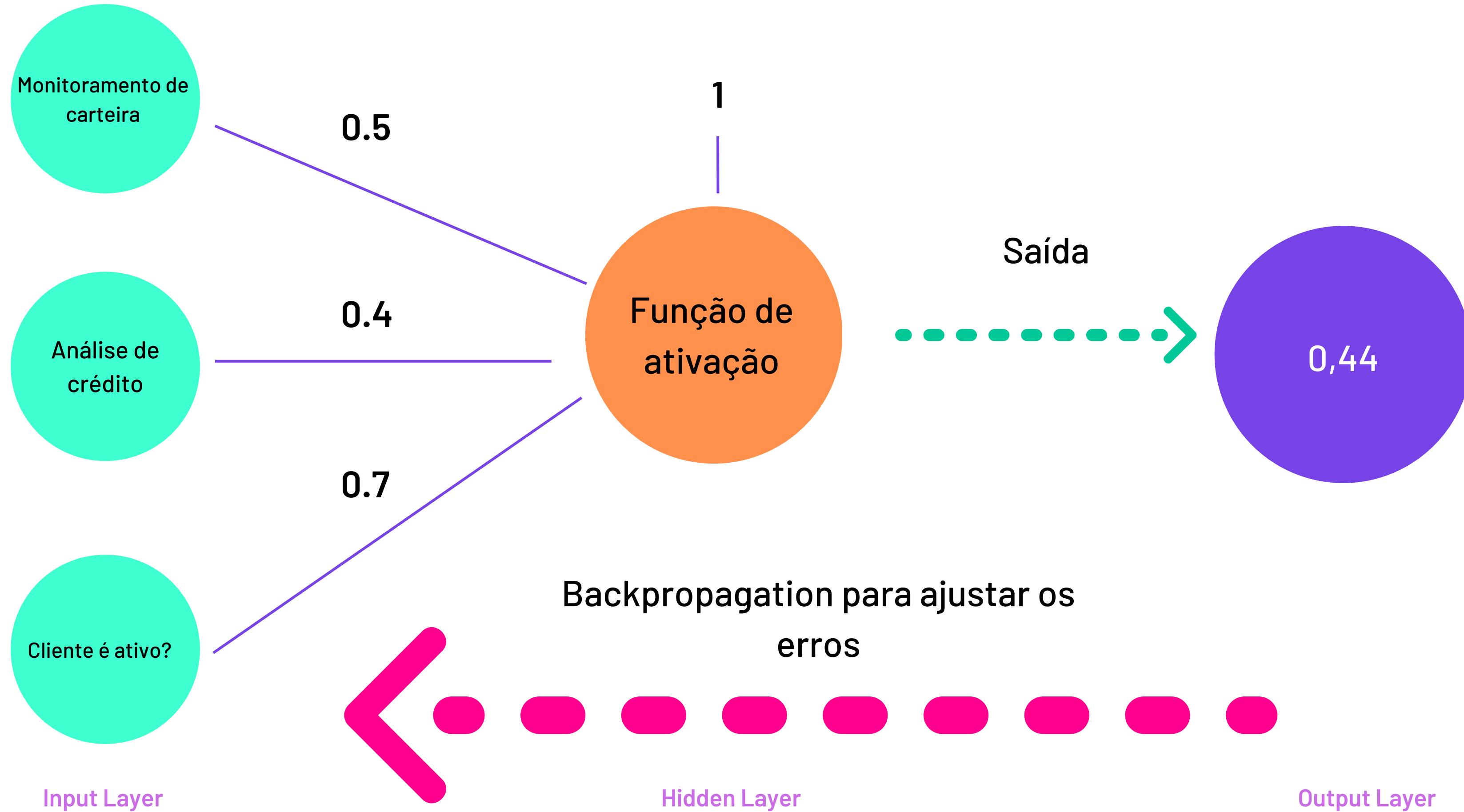
# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?



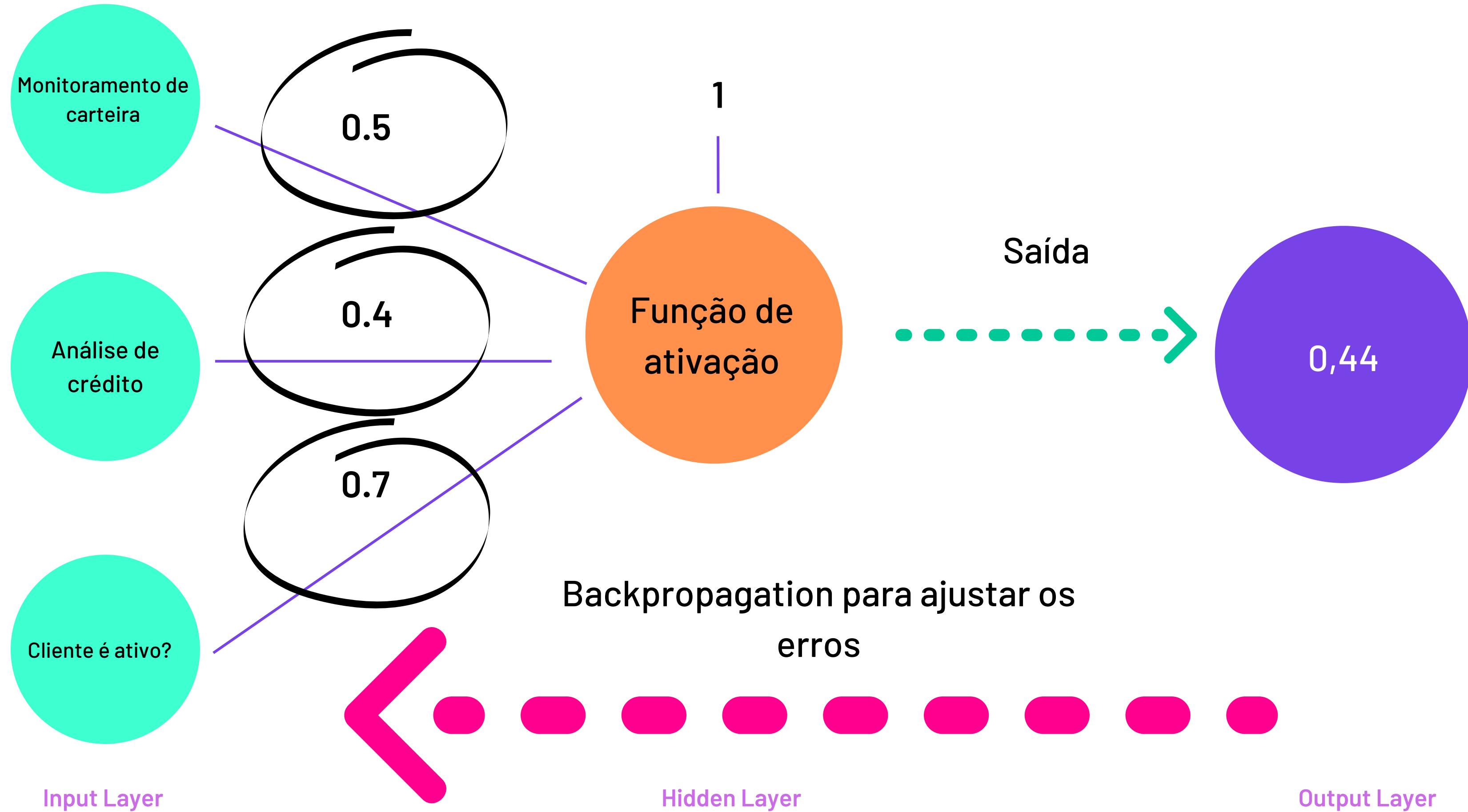
# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?



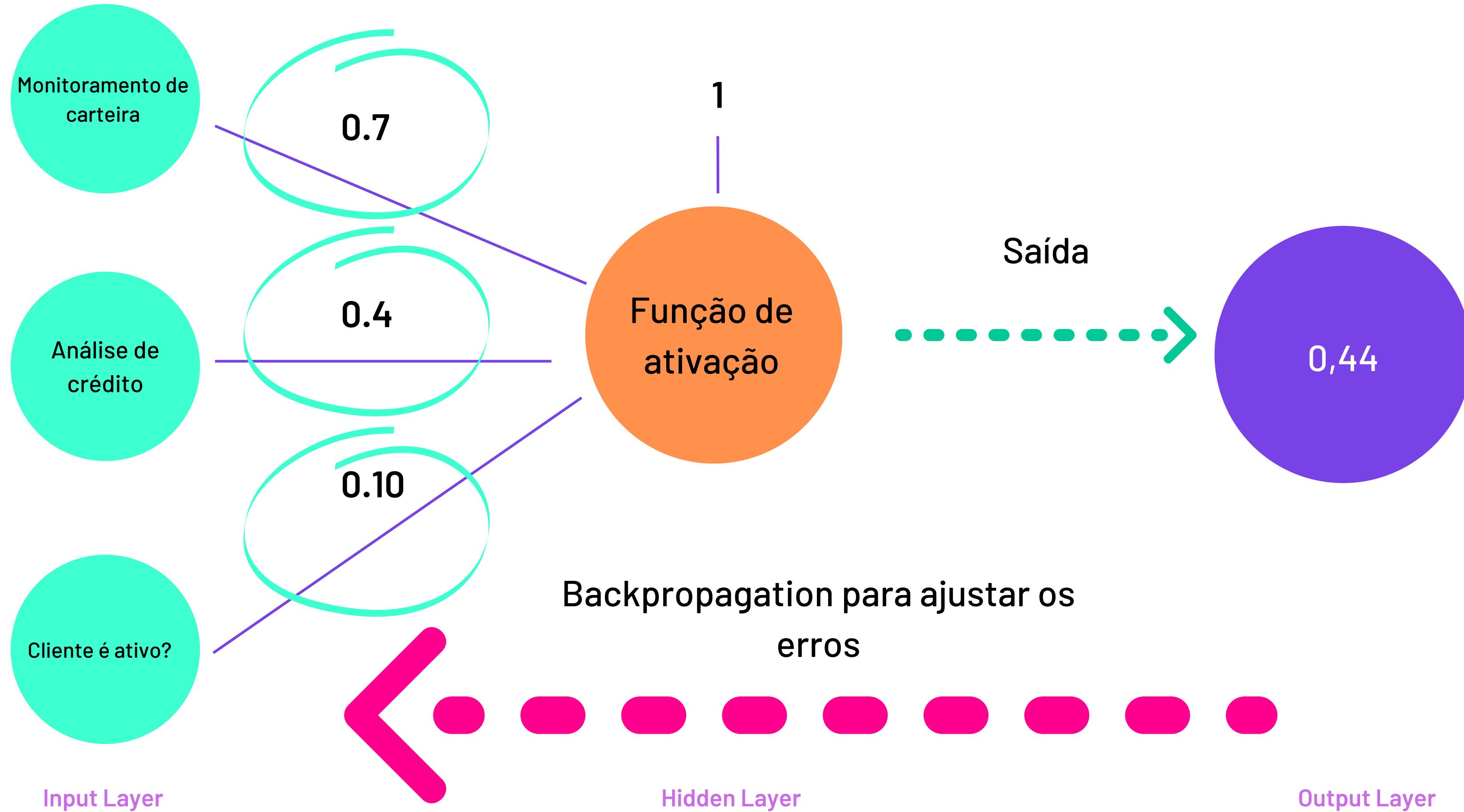
# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?



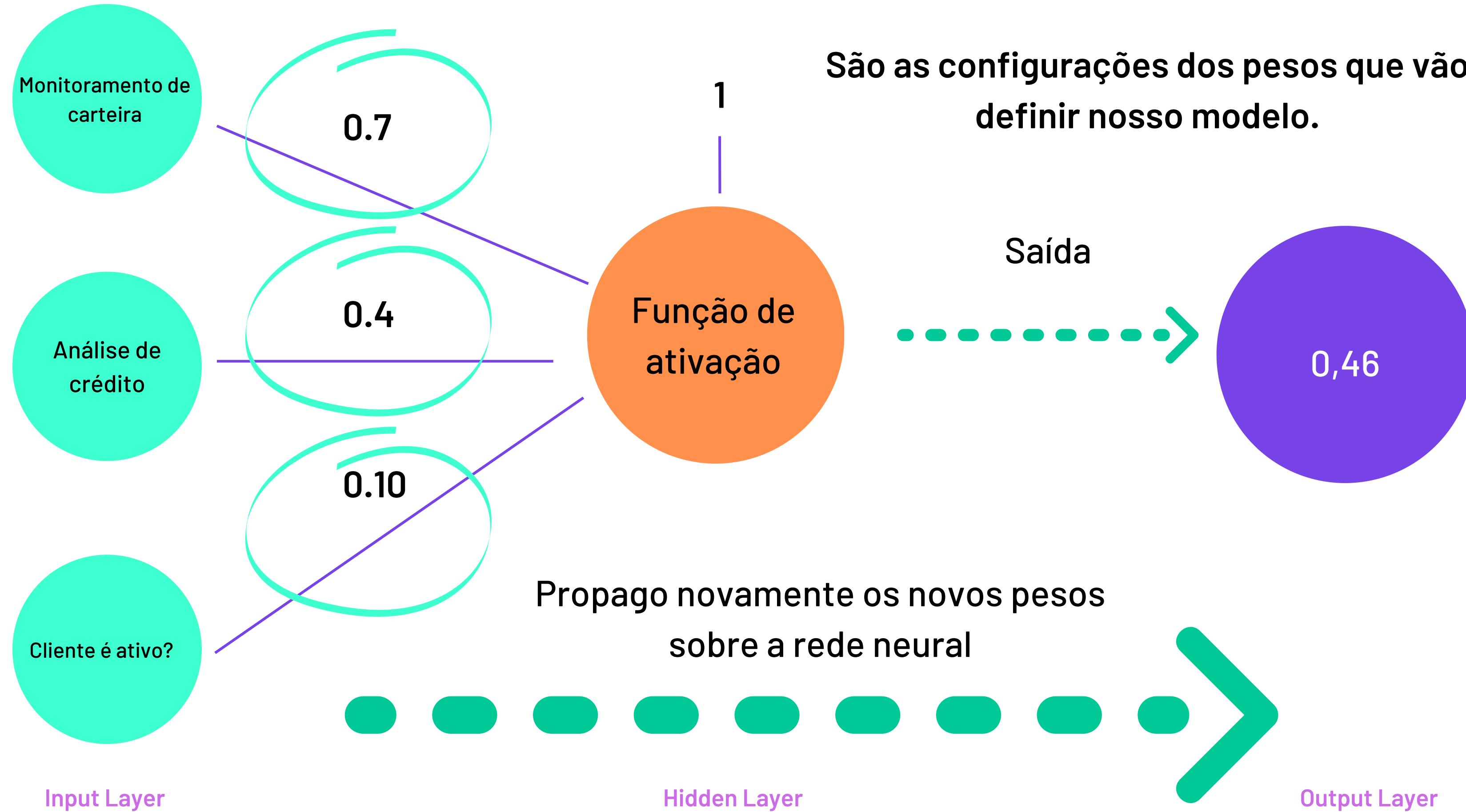
# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?



# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?



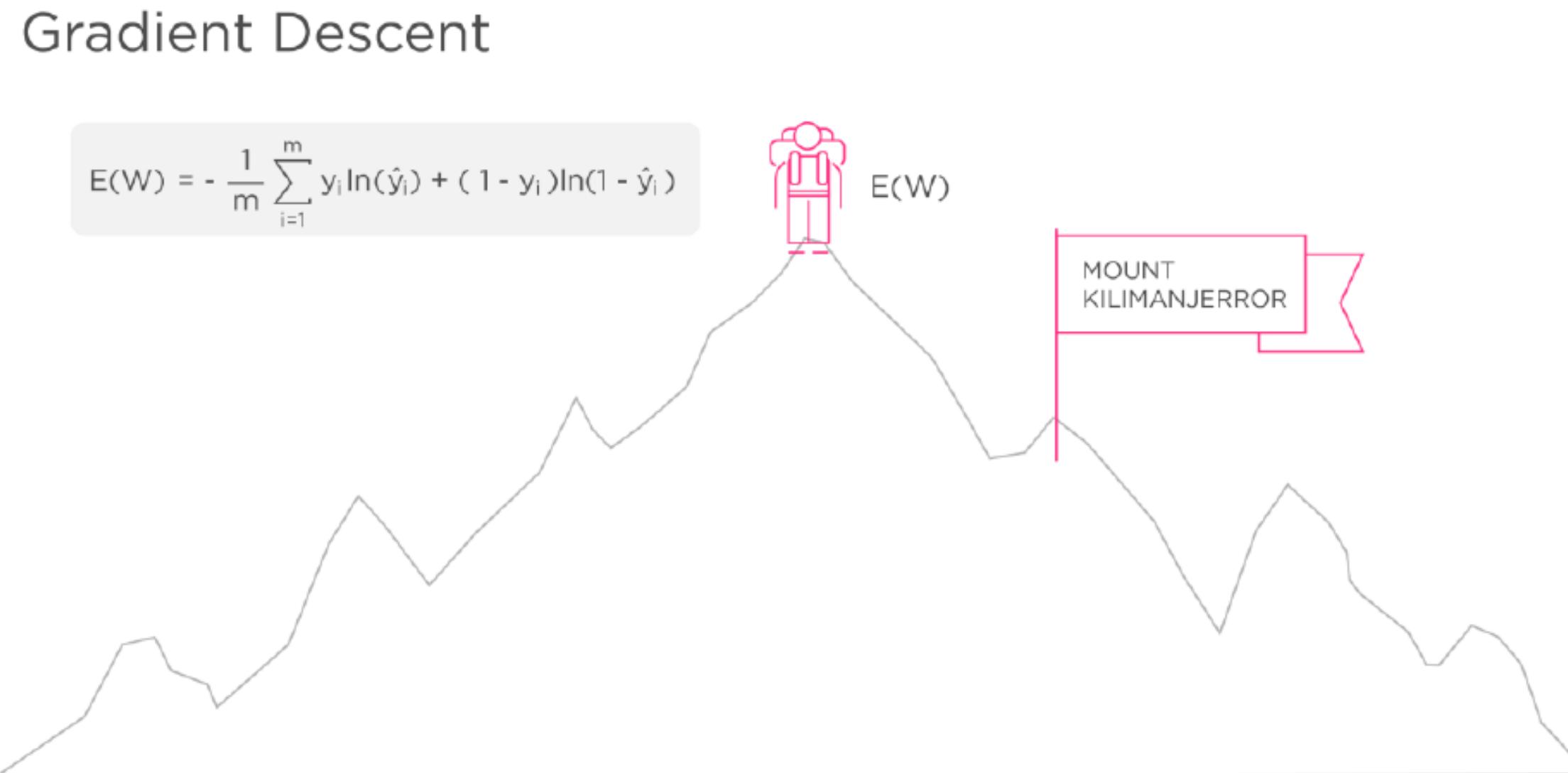
# EXEMPLO DE REDE NEURAL MULTICAMADA: CLIENTE VAI PAGAR A CONTA?



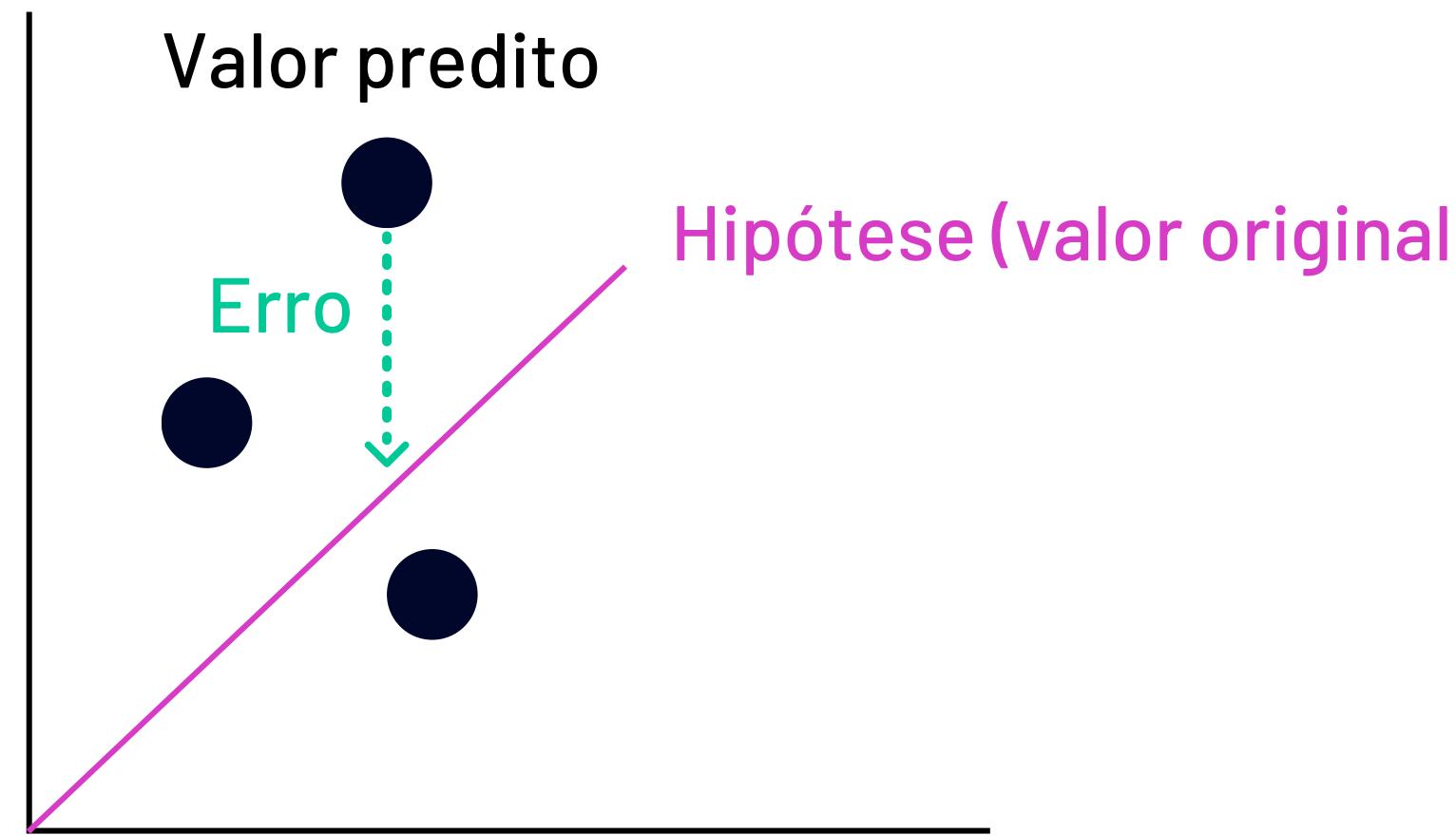
# GRADIENT DESCENT

A descida do gradiente é um algoritmo de otimização para encontrar os valores de parâmetros (pesos) de uma função que minimizam uma função de custo.

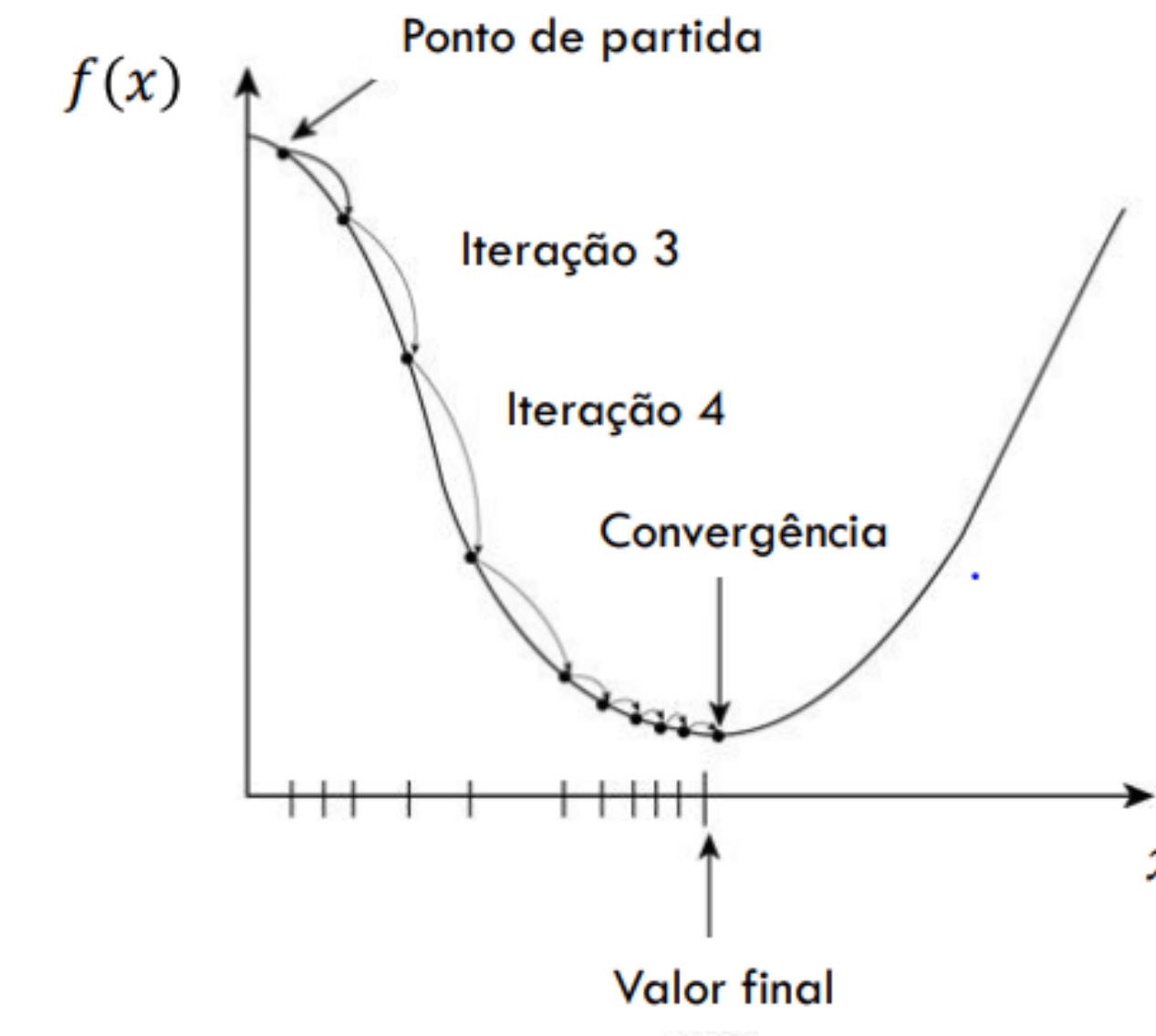
- Otmizaçāo na busca dos melhores pesos.
- Um gradiente simplesmente mede a mudança em todos os pesos em relação à mudança no erro.



Você também pode pensar em um gradiente como a inclinação de uma função. Quanto maior o gradiente, mais inclinada é a subida e mais rápido o modelo pode aprender. Mas se a inclinação for zero, o modelo para de aprender. O Gradiente Descendente (GD) é um algoritmo utilizado para encontrar o mínimo de uma função de forma iterativa.



Taxa de aprendizagem (aceleração)  
Gradiente (direção) Como procurar a  
parte mais baixa da função de custo?

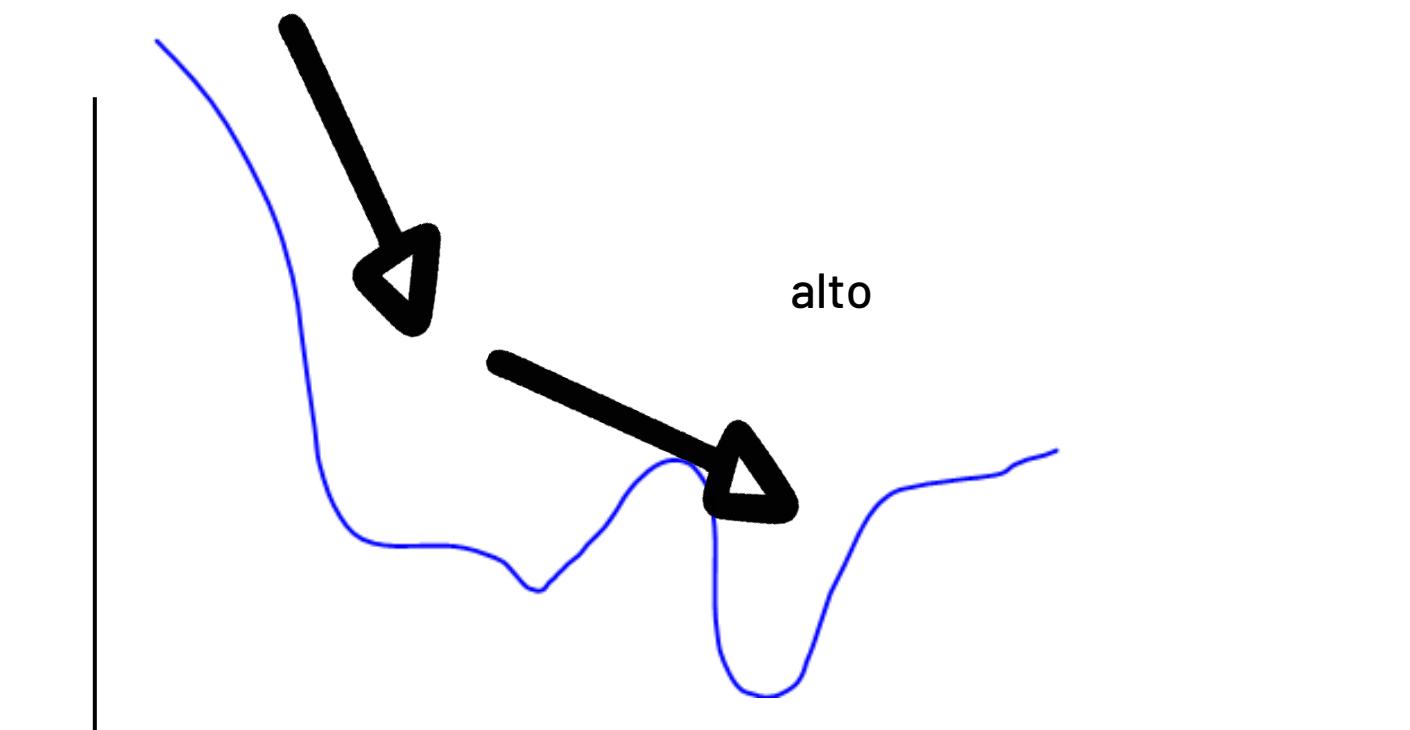


# TAXA DE APRENDIZAGEM

Taxa de aprendizagem controla o tamanho do passo em cada iteração.



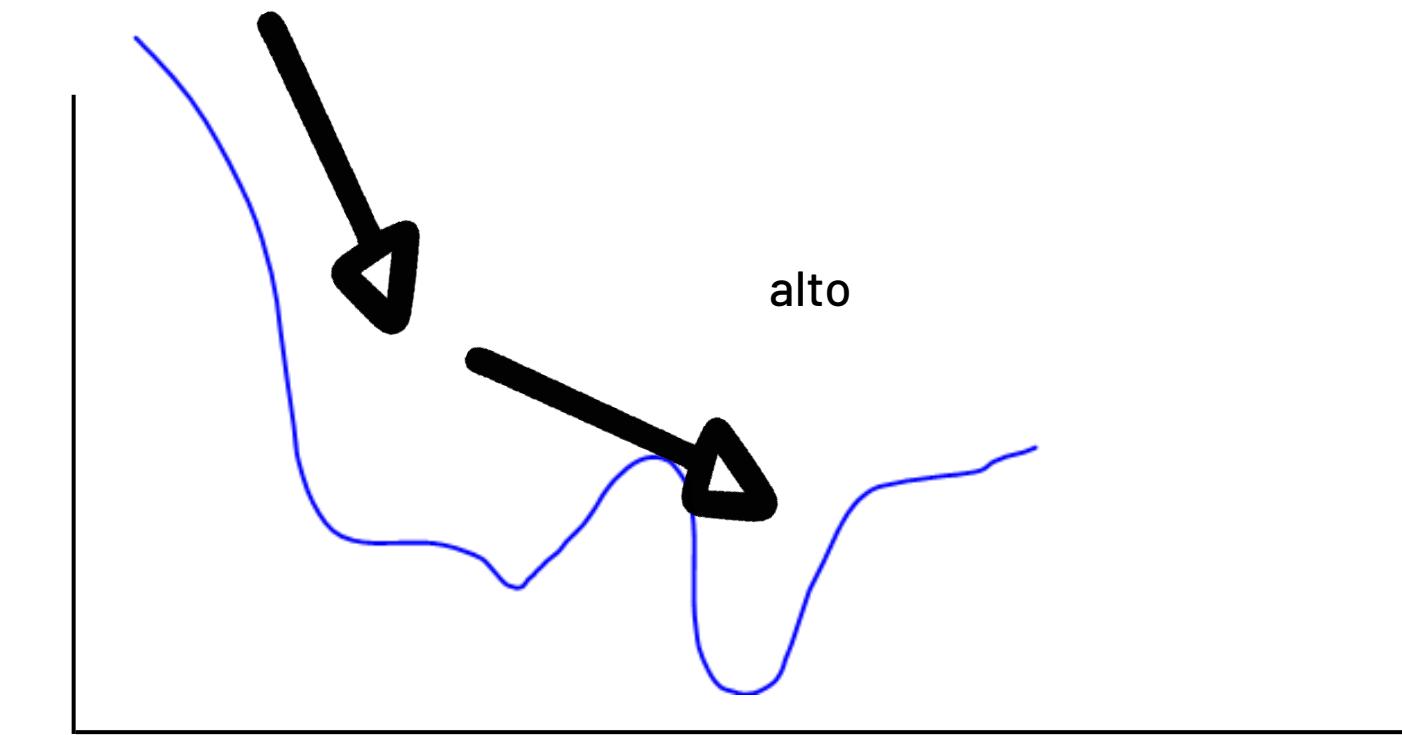
Uma taxa de aprendizagem baixa pode ser mais precisa, porém pode demorar para encontrar o mínimo local. Uma taxa de aprendizado alta pode deixar passar o melhor mínimo local.



Tente ... 0,001 – 0,003 – 0,01 – 0,03 – 0,1 – 0,3 –  
1 ... Suba/Desça ~ 3x em 3x

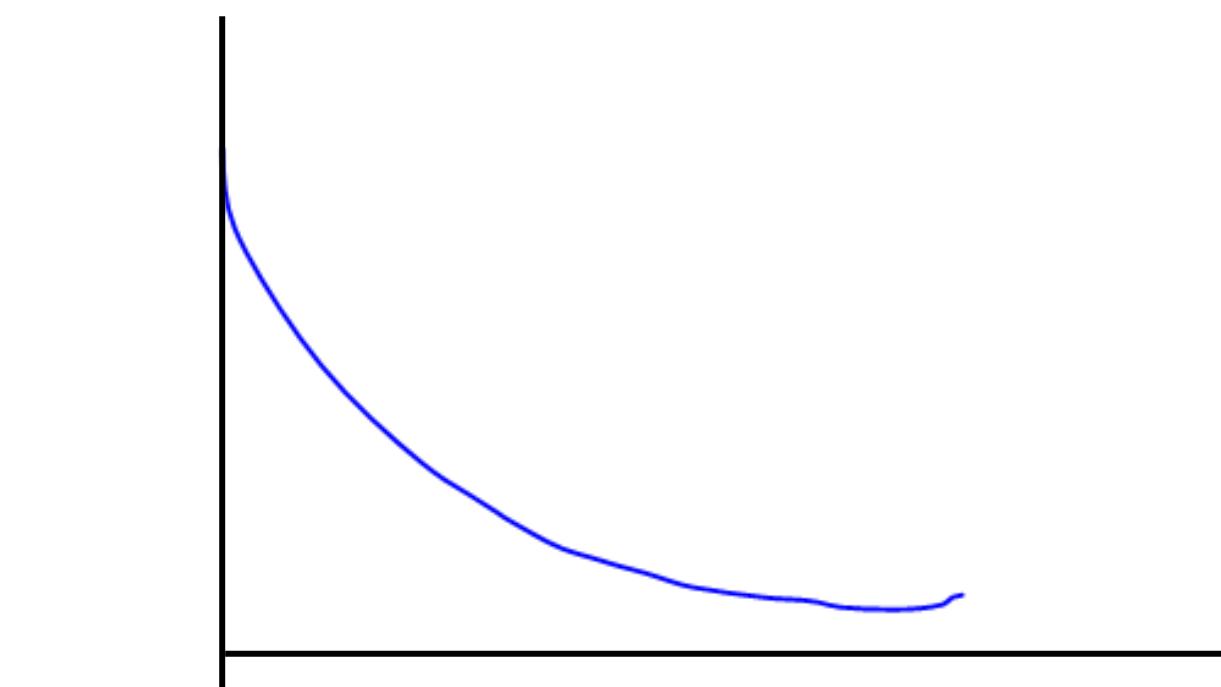
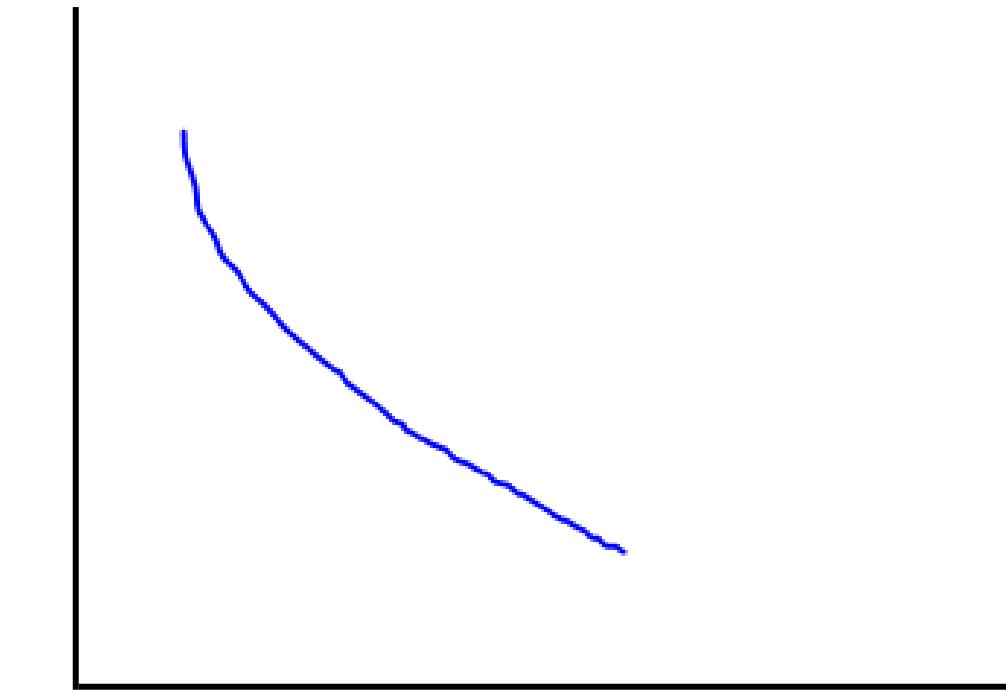
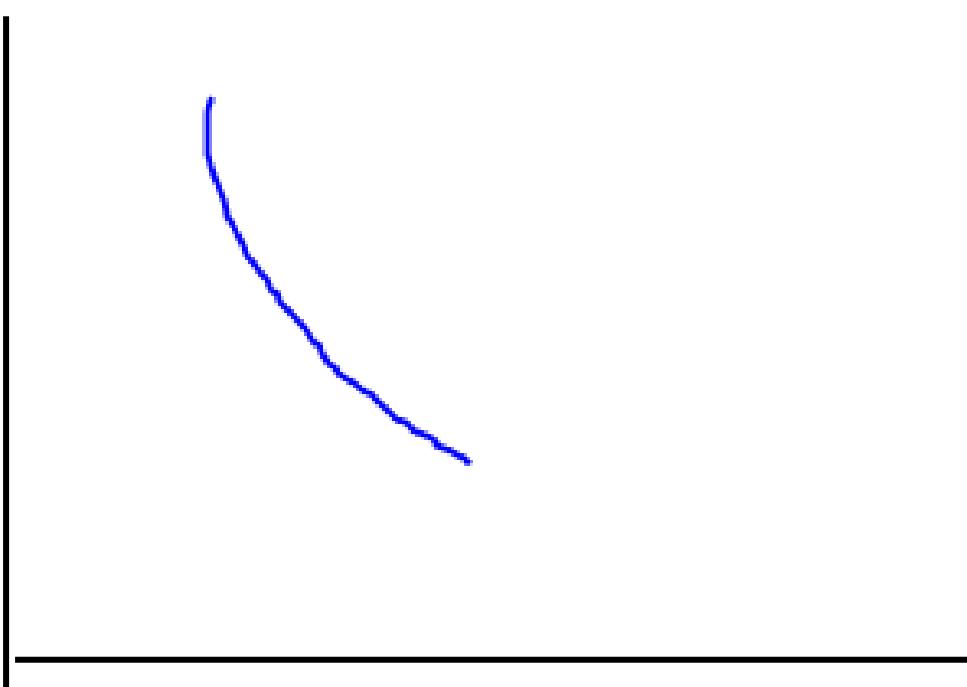
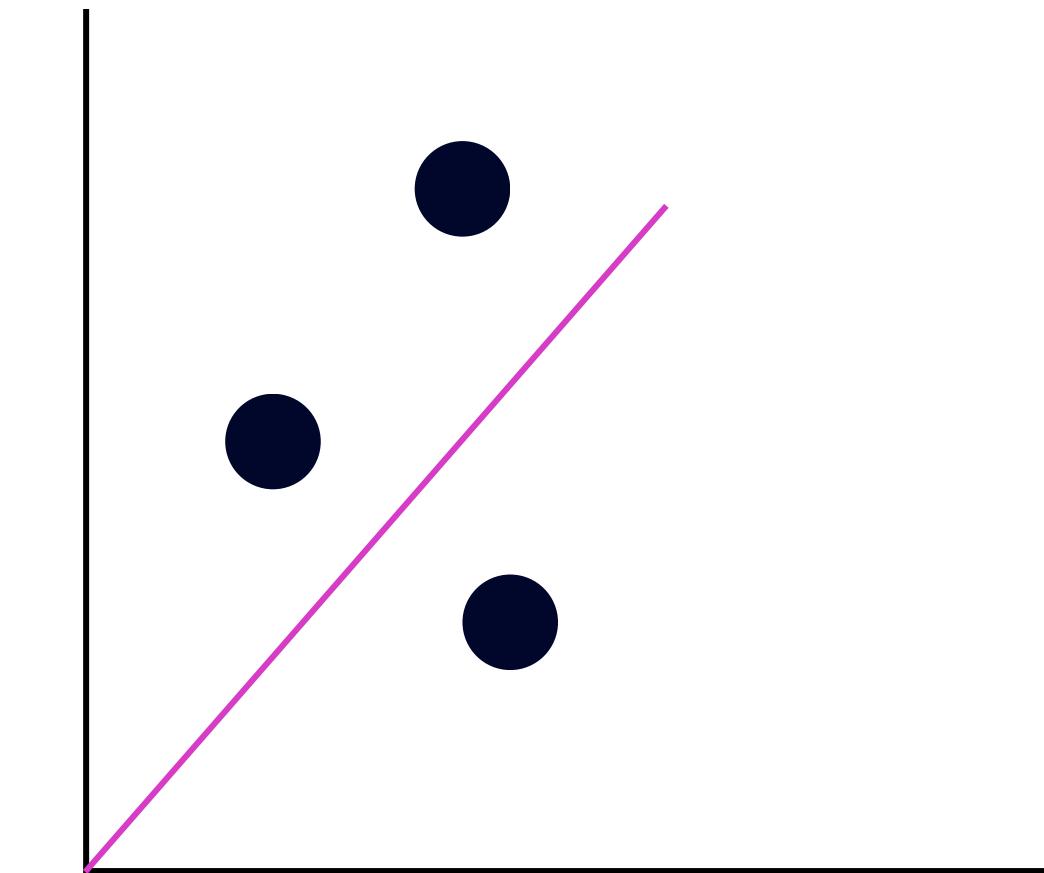
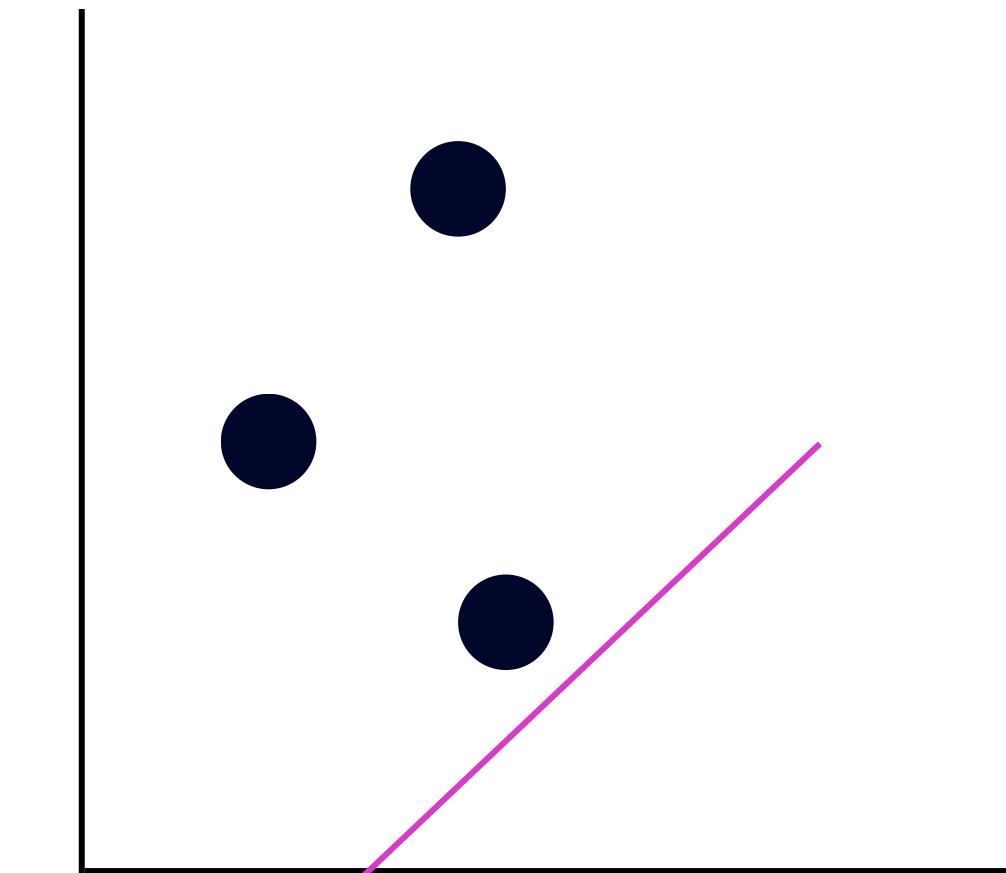
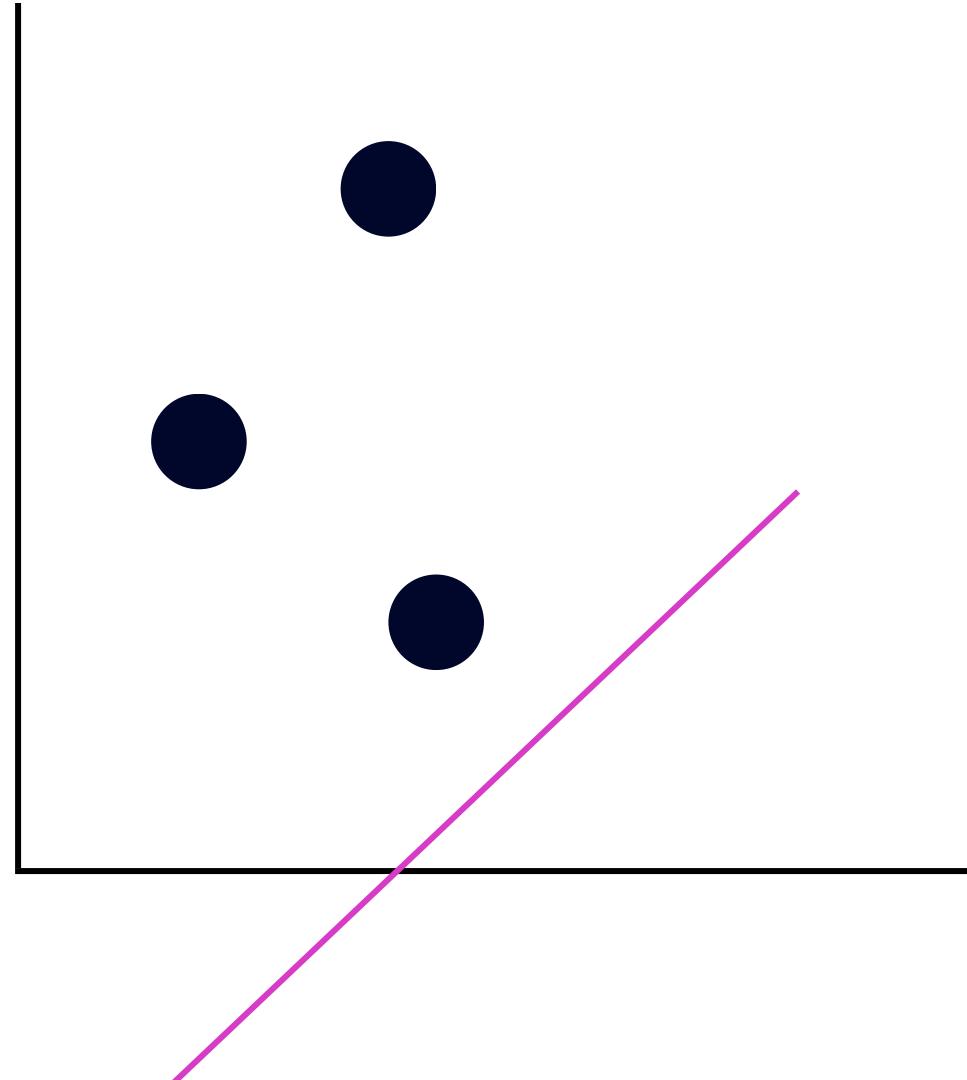
# TAXA DE APRENDIZAGEM

Taxa de aprendizagem controla o tamanho do passo em cada iteração.



Tente ... 0,001 – 0,003 – 0,01 – 0,03 – 0,1 – 0,3 –  
1 ... Suba/Desça ~ 3x em 3x

O custo **baixa** cada vez que a reta se aproximar ao valor ideal para meu problema.



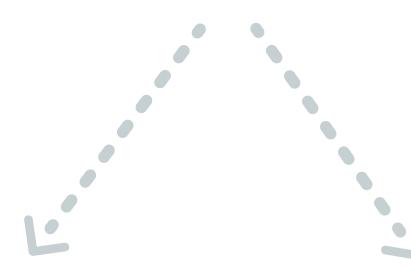
Pensando na questão da matemática por trás dos gradientes, podemos dizer que o gradiente descendente é a derivada da função do erro em relação a nosso peso.

Repita "N" vezes: Aceleração (taxa de aprendizagem).

Pesos

Definido pelo usuário.

Gradiente

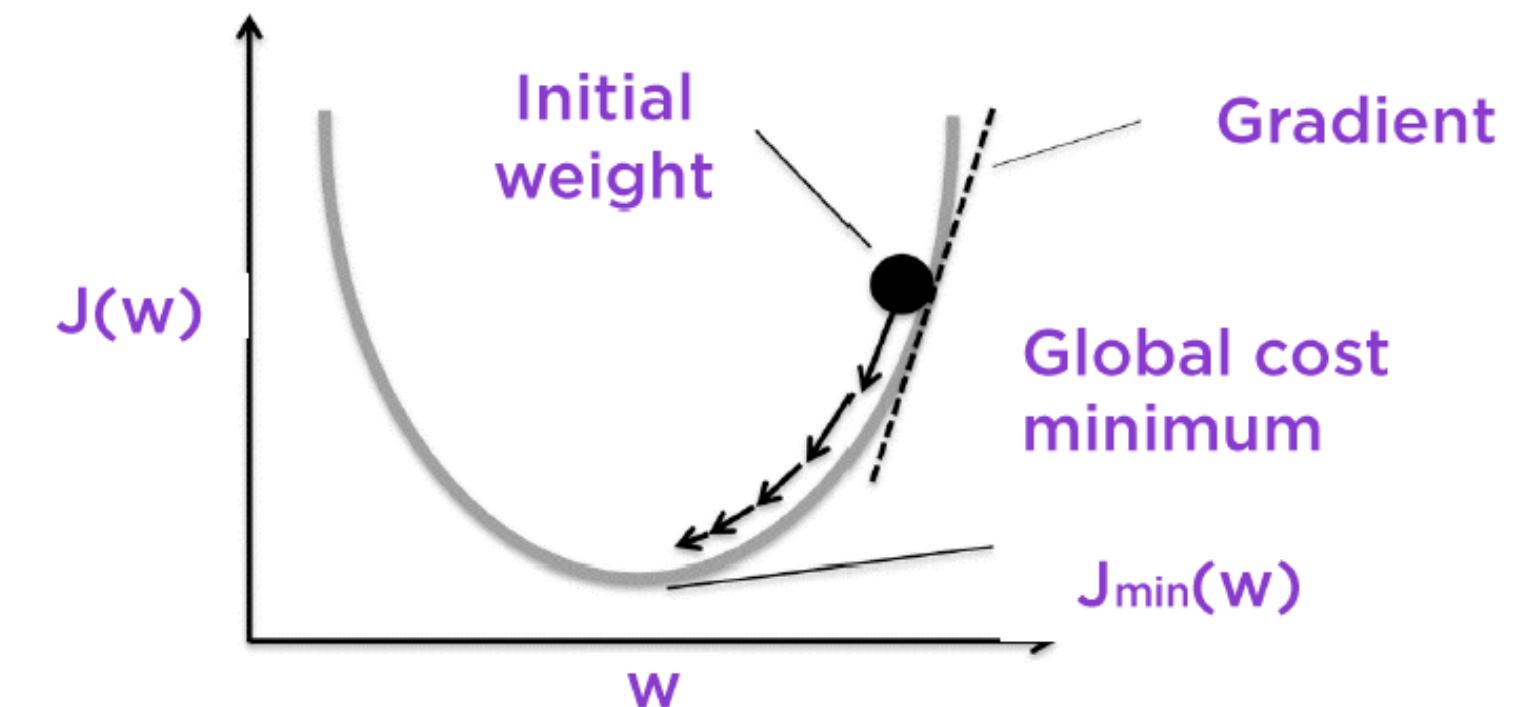


O gradiente é a derivada da função do erro em relação a nosso peso.

$$w_0 := w_0 - \alpha \nabla_{w_0}$$

$$w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^m \text{erro}$$

Quanto mais o erro se aproxime de zero, melhor é a taxa de erro.

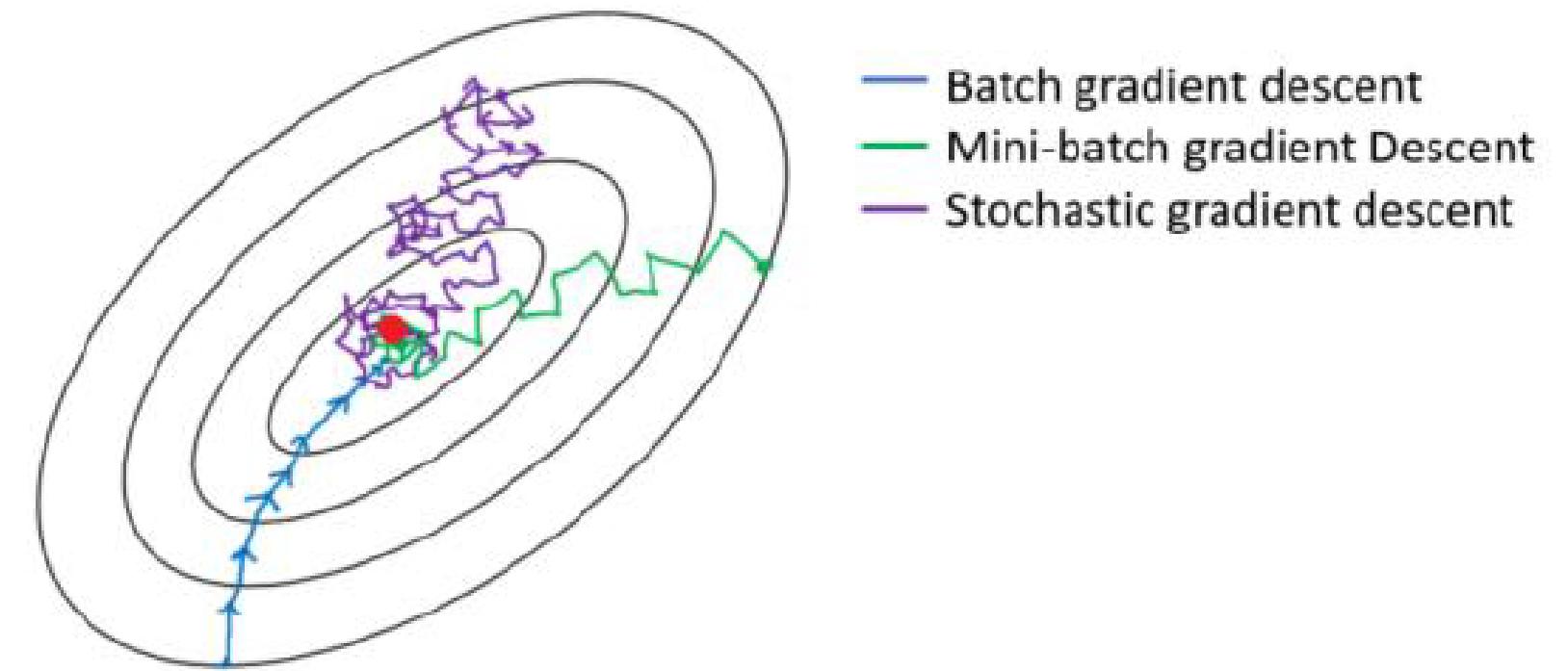
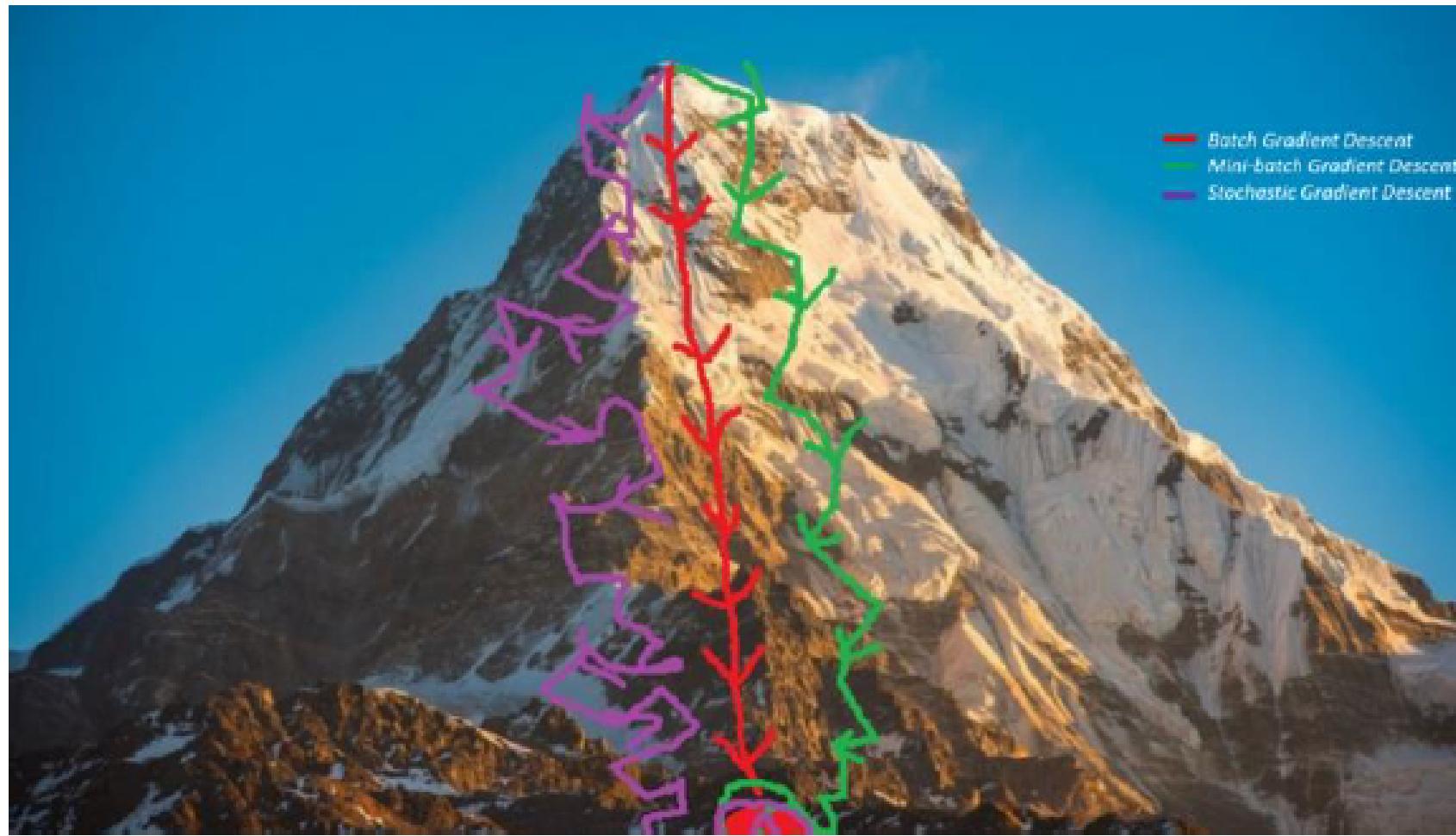


# GRADIENT DESCENT



Mas nem sempre é o valor ideal (melhor).

# DIFERENTES ALGORITMOS DE GRADIENT DESCENT



## BATCH GRADIENT DESCENT (EM LOTE) - BATCH GRADIENT DESCENT, BGD

Calcula o erro para cada exemplo dentro do conjunto de dados de treinamento, mas somente após a avaliação de todos os exemplos de treinamento o modelo é atualizado. Todo esse processo é como um ciclo e é chamado de época de treinamento.

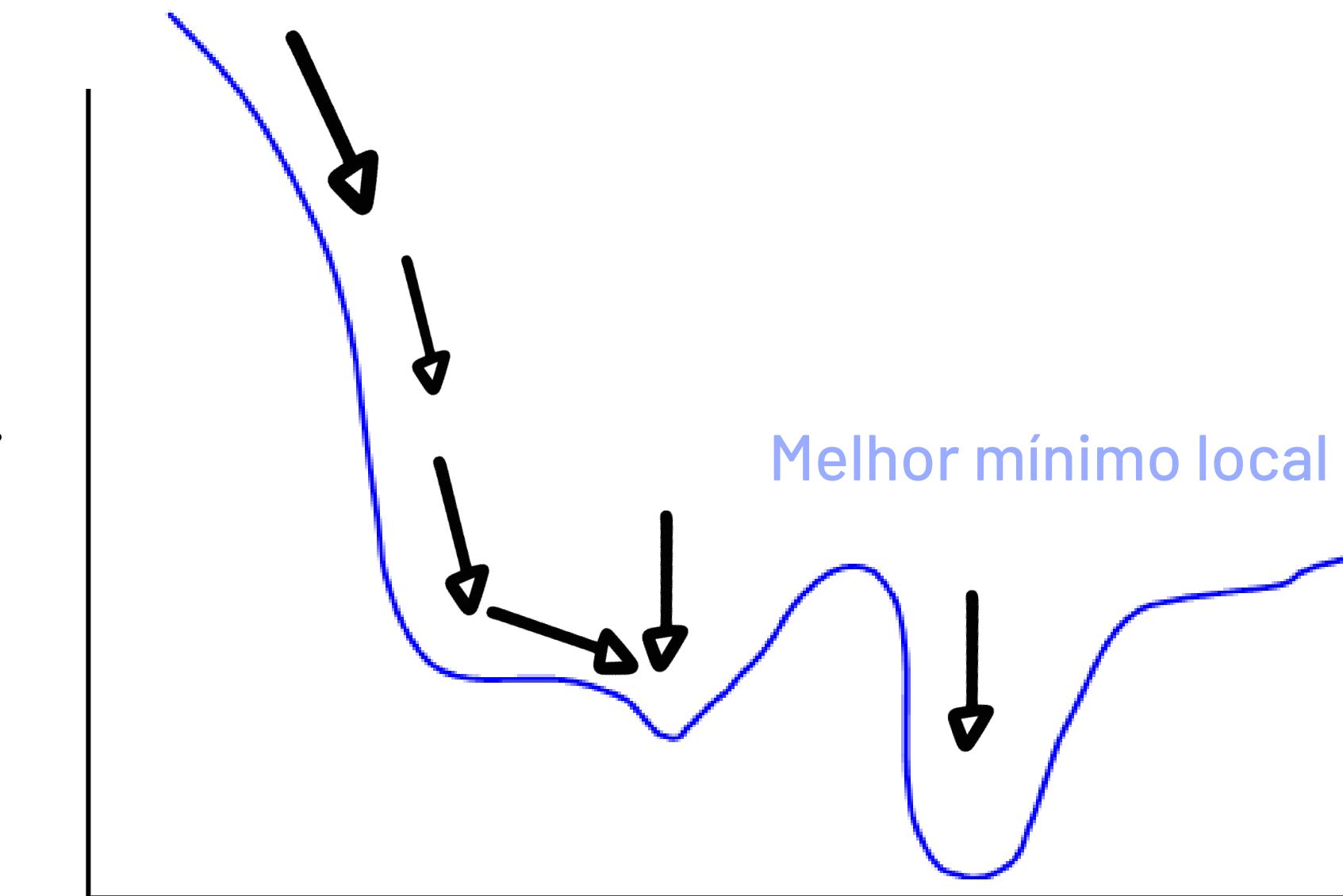
- Exige Alta Eficiência computacional.
- Utiliza todo o dataset.
- Imagine se você tem 10000 dados, cada dado com 10 features, são 100mil valores para computar a cada iteração, em cada época...



## STOCHASTIC GRADIENT DESCENT - SGD

Os pesos são atualizados após cada linha de processamento da rede. Uma vantagem é que as atualizações frequentes nos permitem ter uma taxa de melhoria bastante detalhada.

- Eficiência computacional não tão rápida.
- Dados aleatórios de treinamento por iteração, para atualizar os parâmetros.

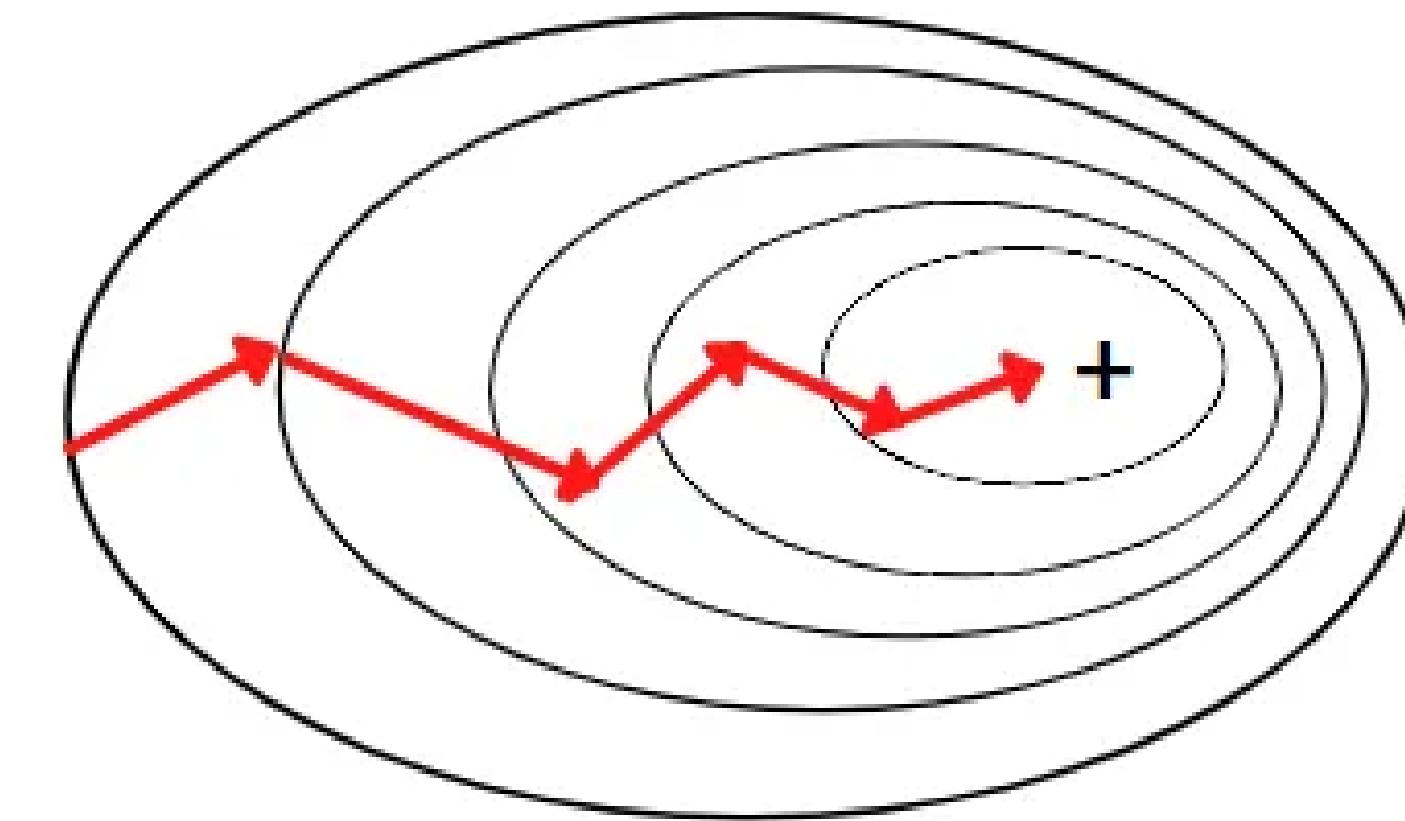


## MINI BATCH GRADIENT DESCENT (EM PEQUENOS LOTE) - MBGD

Combinação da batch com a stochastic. Divide o conjunto de dados de treinamento em pequenos lotes e executa uma atualização para cada um desses lotes. Isso cria um equilíbrio entre a robustez da descida de gradiente estocástica e a eficiência da descida de gradiente em lote.

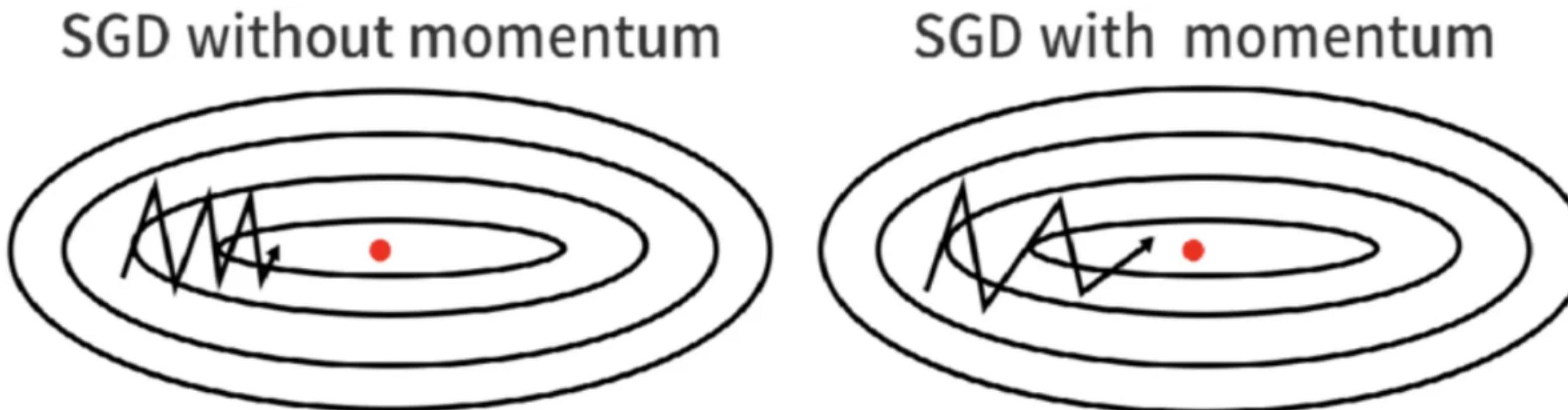
### Mini-Batch Gradient Descent

- Média Eficiência computacional.



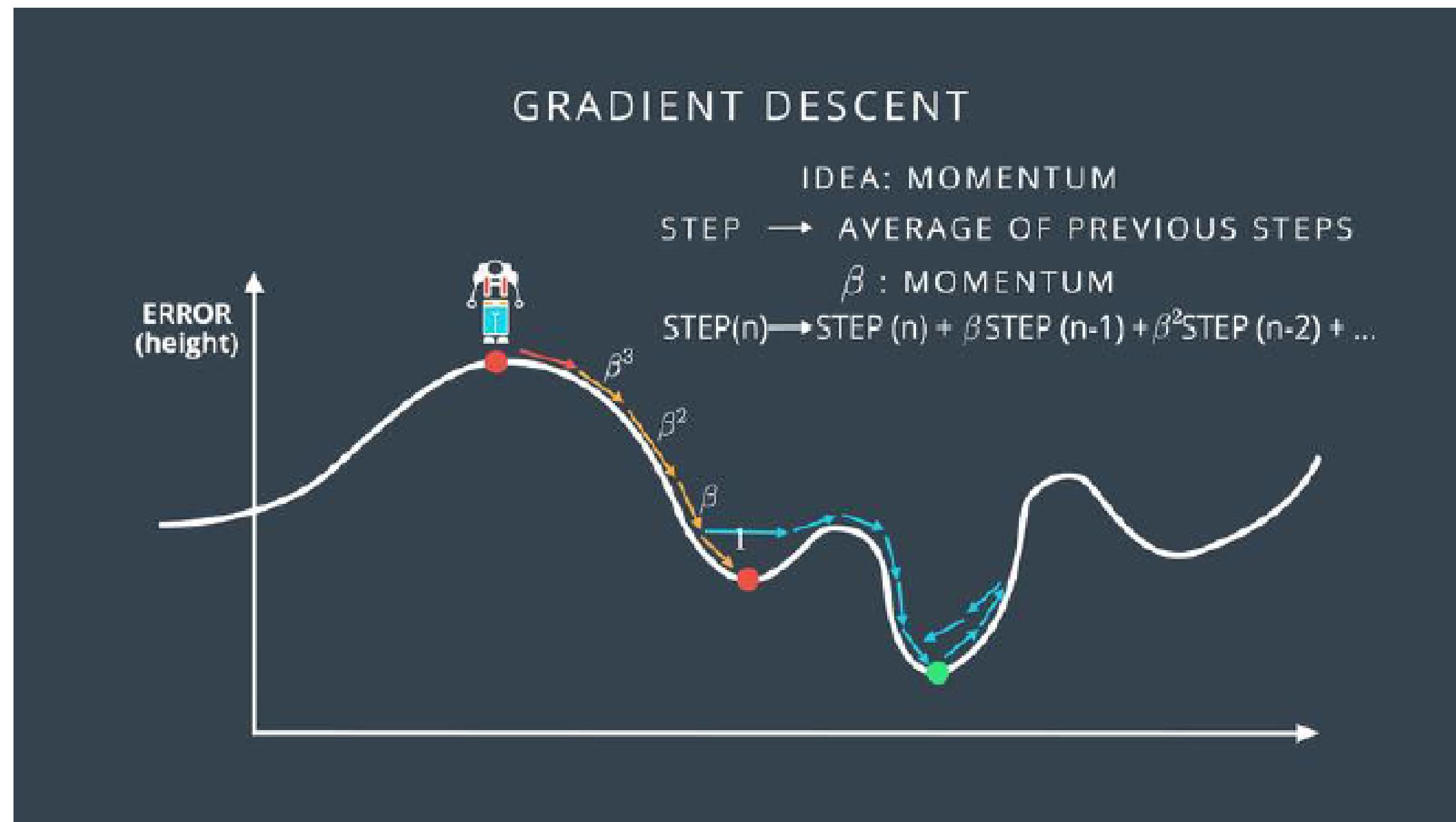
## GRADIENT DESCENT COM MOMENTUM (IMPULSO)

Exemplificando de forma bem simples, podemos comparar esses tipos de otimizador como uma bola de boliche rolando em um declive suave em uma superfície lisa (sua rolagem irá iniciar de forma lenta, porém logo pegará impulso até que chegue à velocidade final). A otimização Momentum se preocupa muito com os gradientes anteriores: a cada iteração, ela subtrai o gradiente local do vetor momentum  $m$  (multiplicado pela taxa de aprendizado  $n$ ) e atualiza os pesos adicionando este vetor momentum. O algoritmo adiciona um novo hiperparâmetro  $\beta$  (momentum) que deve ser ajustado entre 0 até 1. Um bom e típico valor para esse hiperparâmetro costuma ser 0,9.



# GRADIENT DESCENT COM MOMENTUM (IMPULSO)

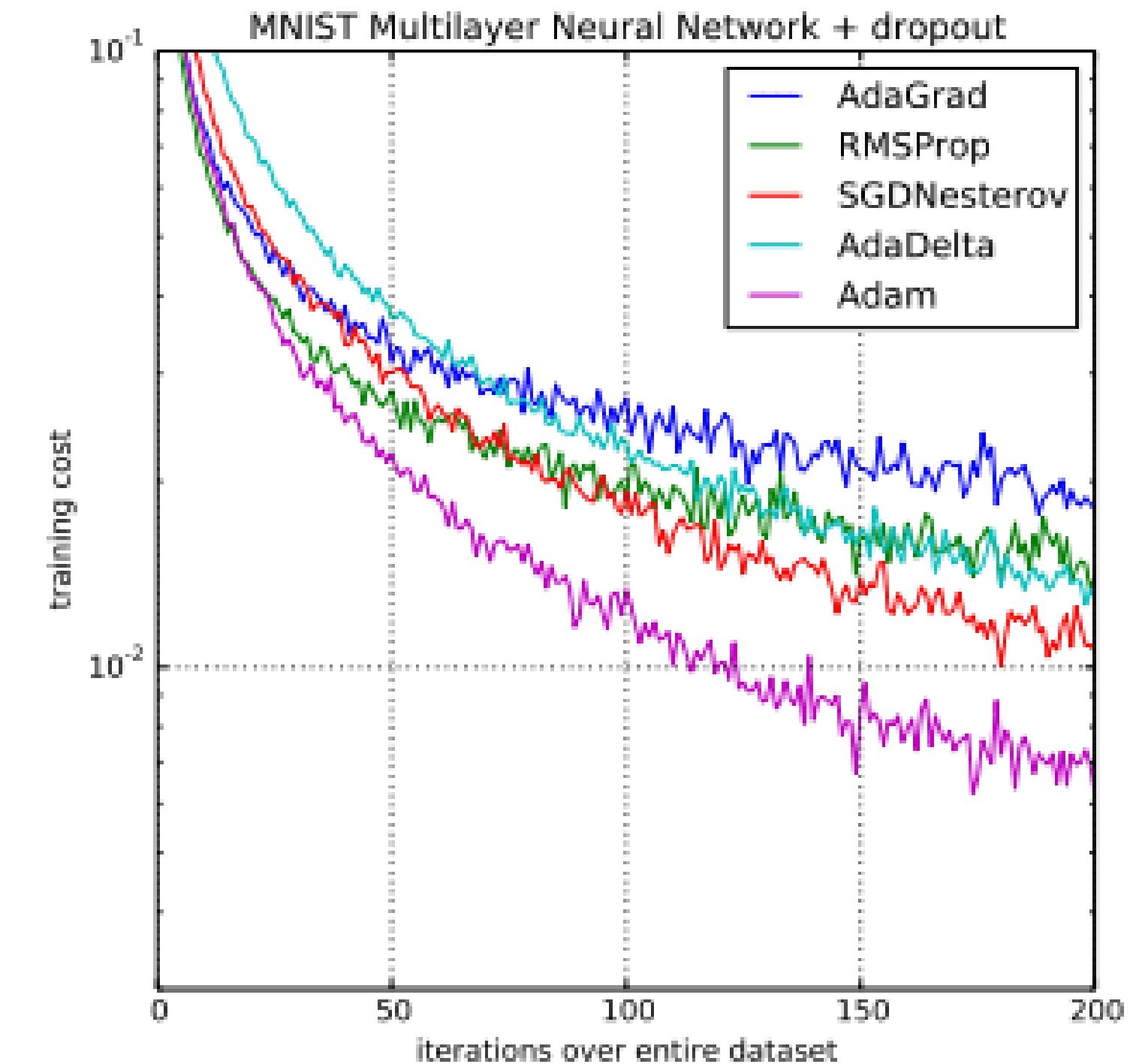
Este gradiente retido é multiplicado por um valor denominado "Coeficiente de Momentum", que é a porcentagem do gradiente retido a cada iteração.



# GRADIENT DESCENT ADAM - ADAPTATIVE MOMENT ESTIMATION

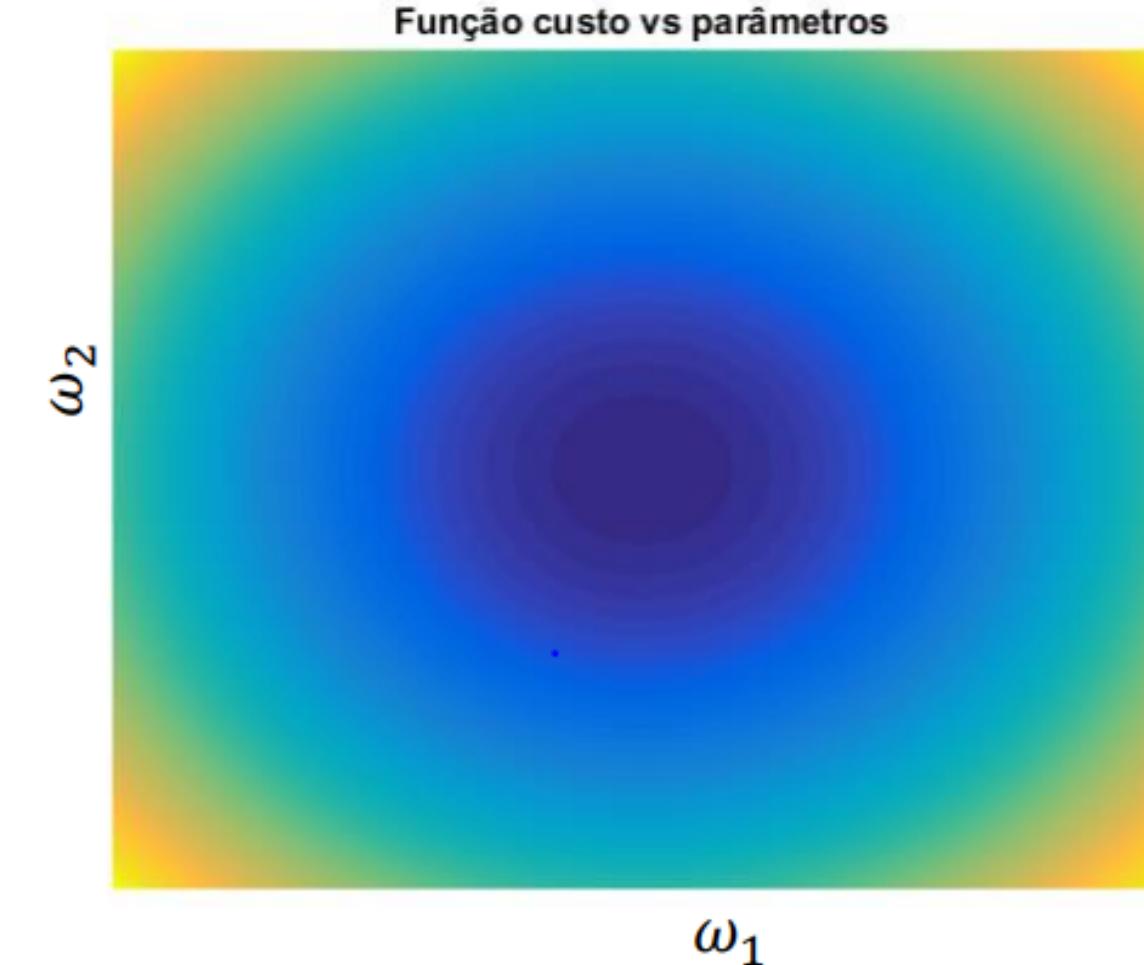
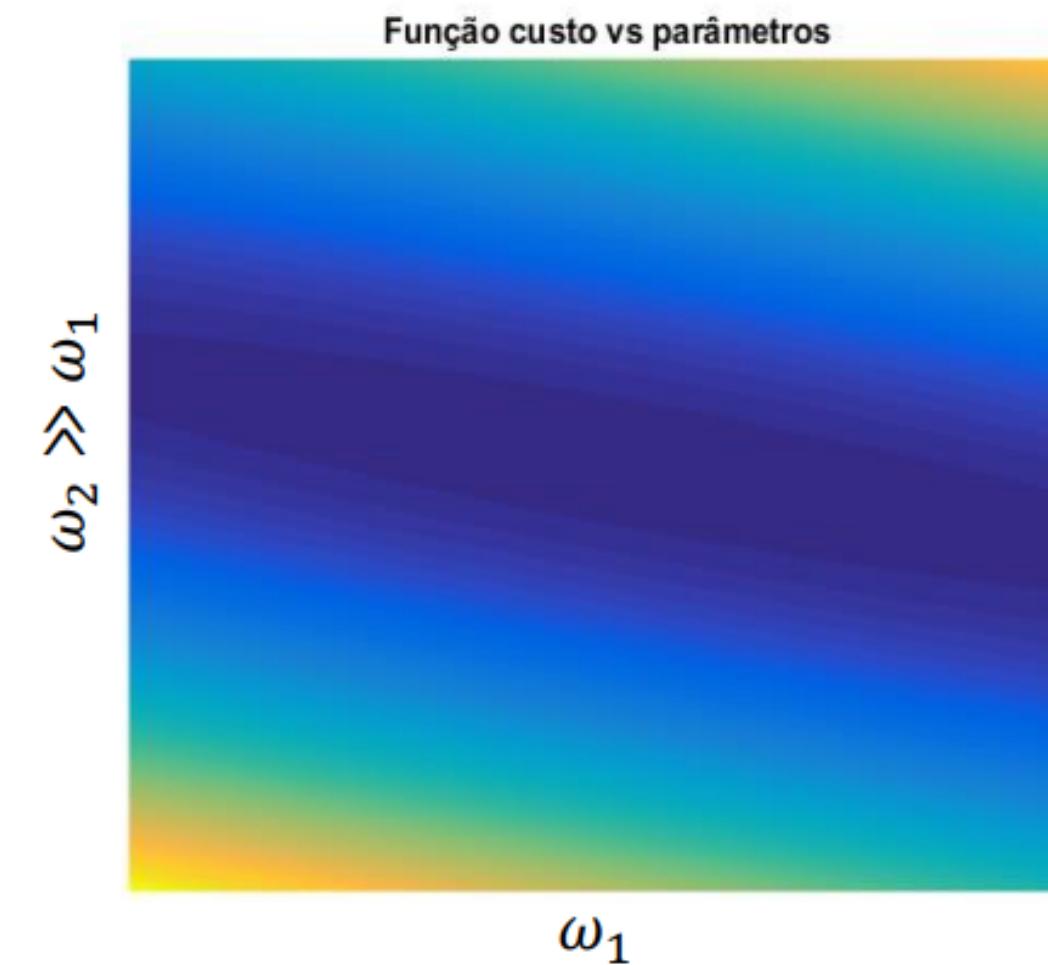
Adaptative moment estimation (Adam) combina as ideia do Momentum e do RMSProp: assim como a otimização de momentum, o Adam acompanha uma média exponencialmente decadente de gradientes passados, e assim como o RMSProp, controla uma média exponencialmente decadente de gradientes quadrados passados.

O otimizador Adam requer menos configuração de hiperparâmetro da taxa de aprendizagem, então um valor padrão  $n = 0,001$  pode ser uma boa opção!



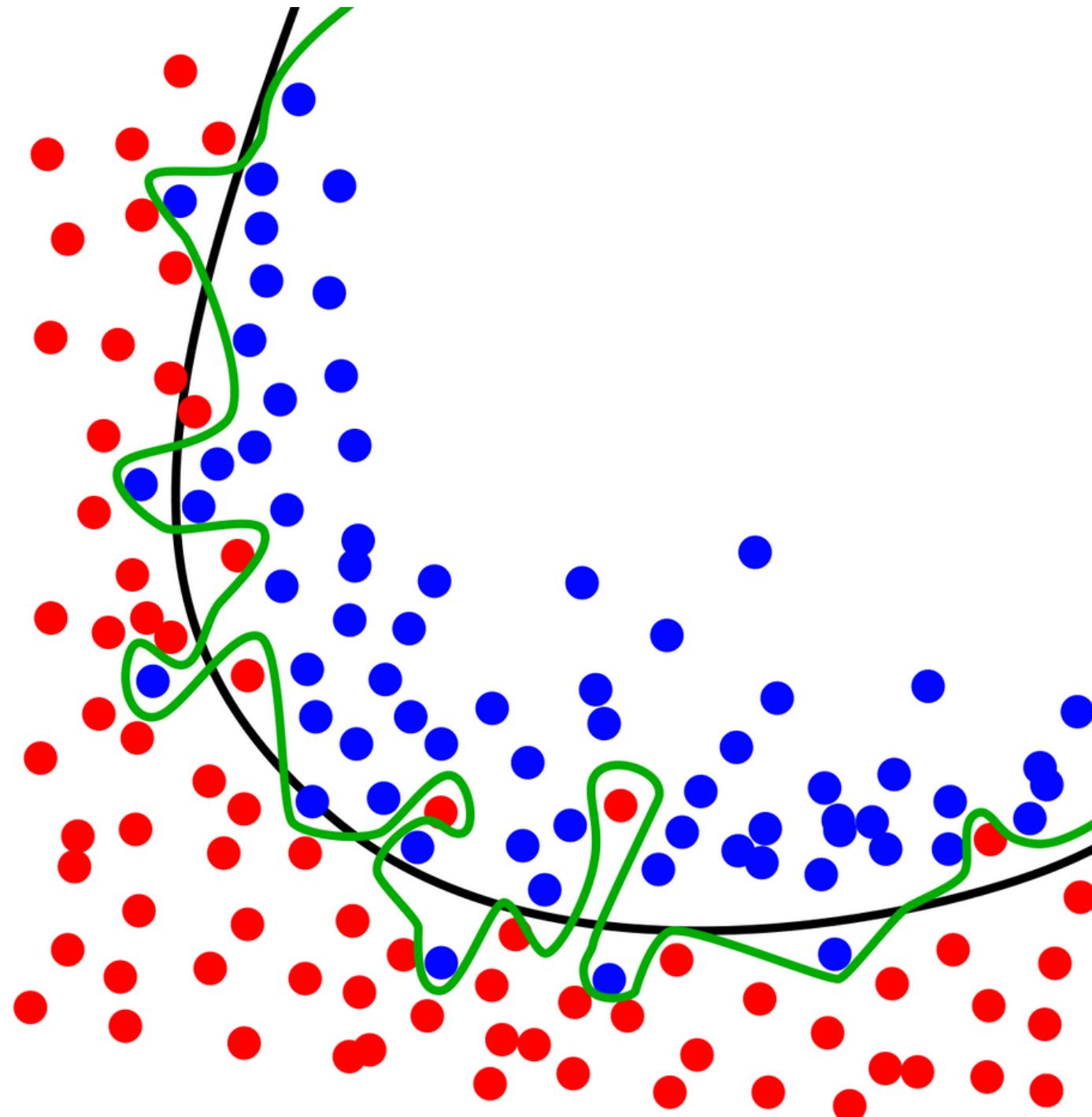
## ESCALA DOS DADOS DE ENTRADA

Importante termos em mente que uma rede neural aprende melhor quando os dados de entrada estão em uma escala normalizada. A escala dos dados padronizada encontra uma convergência mais rápida.



# REGULARIZAÇÃO

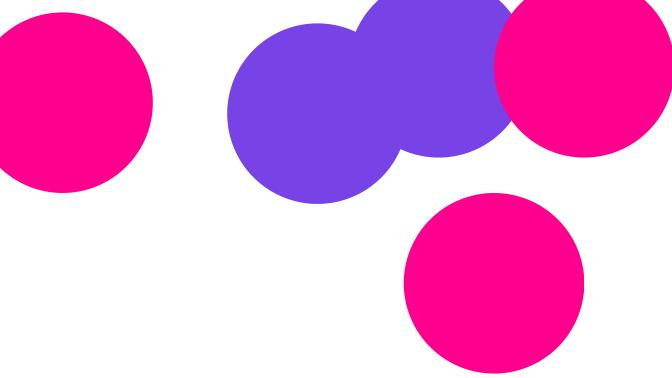
Técnicas para minimizar super ajustes em ML. A regularização remove os ruídos e deixa o modelo com melhor performance.



O que é um modelo super ajustado? (Overfitting)

Quando o algoritmo colocado em produção, não consegue validar dados do mundo real. Os algoritmo praticamente "decorou" os dados do modelo de treinamento.

Como determinar os melhores parâmetros num aprendizado de máquina?



## L1 LASSO

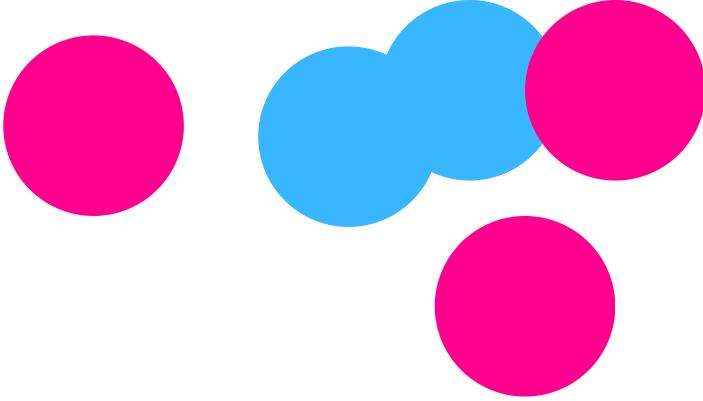
- L1 vai aplicar uma restrição aos coeficientes menos importantes, levando-os a zero.
- Ao zerar um coeficiente, elimina-se o atributo.
- Melhor performance.

## L2 RIDGE

- L2 penaliza os coeficientes que assumem valores muito grandes, levando-os a tender a zero.
- Ajuda a resolver o problema de atributos muito correlacionados.
- Por não reduzir atributos, é muito bem utilizado em datasets menores.
- A grande diferença aqui é que este tipo não chega a zerar o coeficiente, mas reduz os menos importantes a valores muito baixos e mantendo todos eles no modelo.

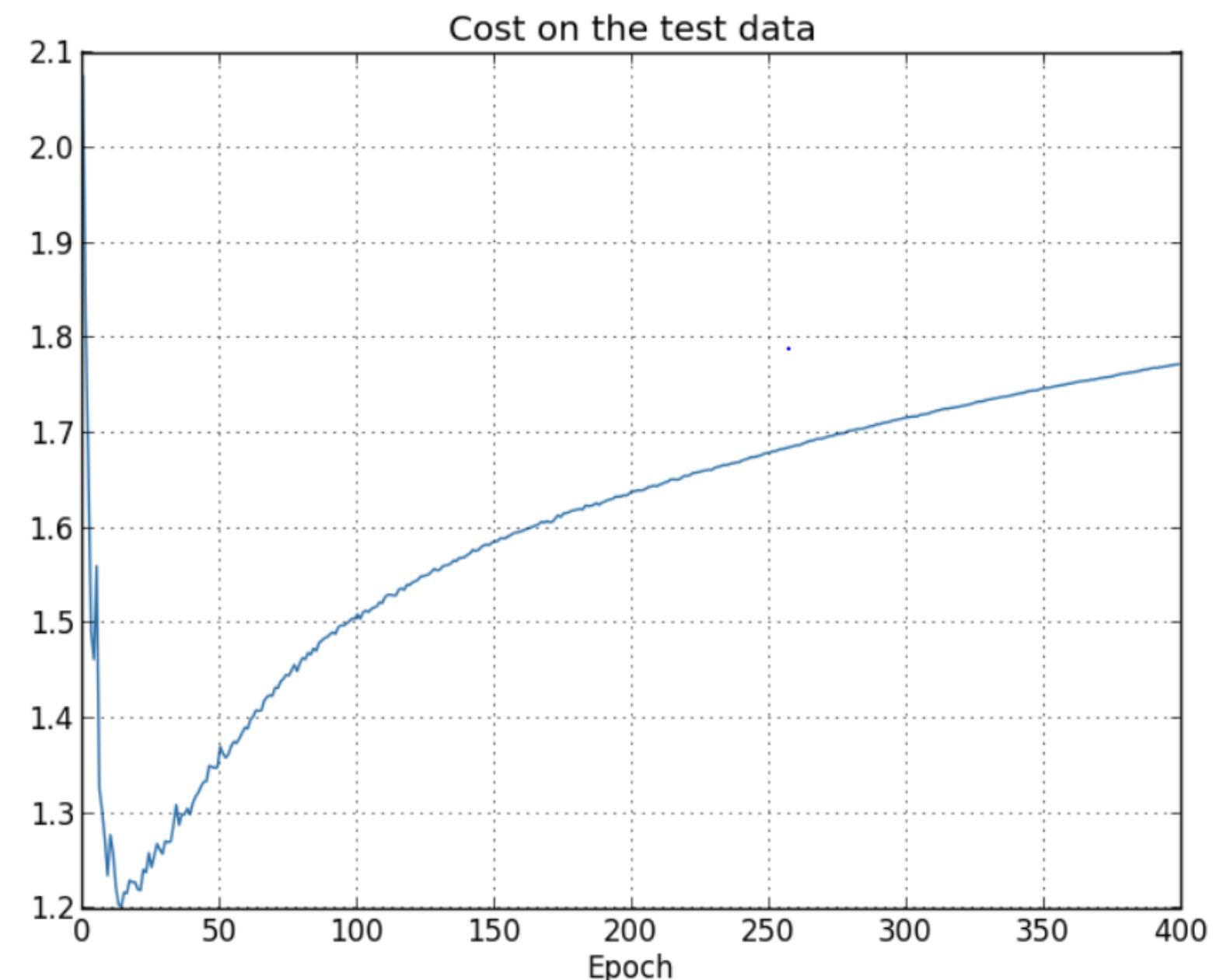
# DROPOUT

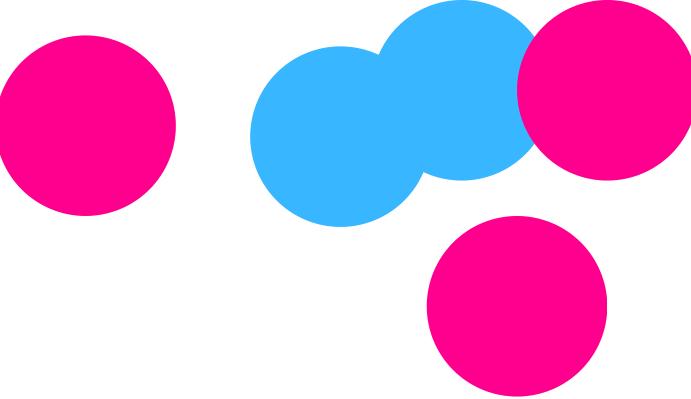
- Remove aleatoriamente neurônios e suas conexões.
- Melhor performance.



# EARLY STOPPING

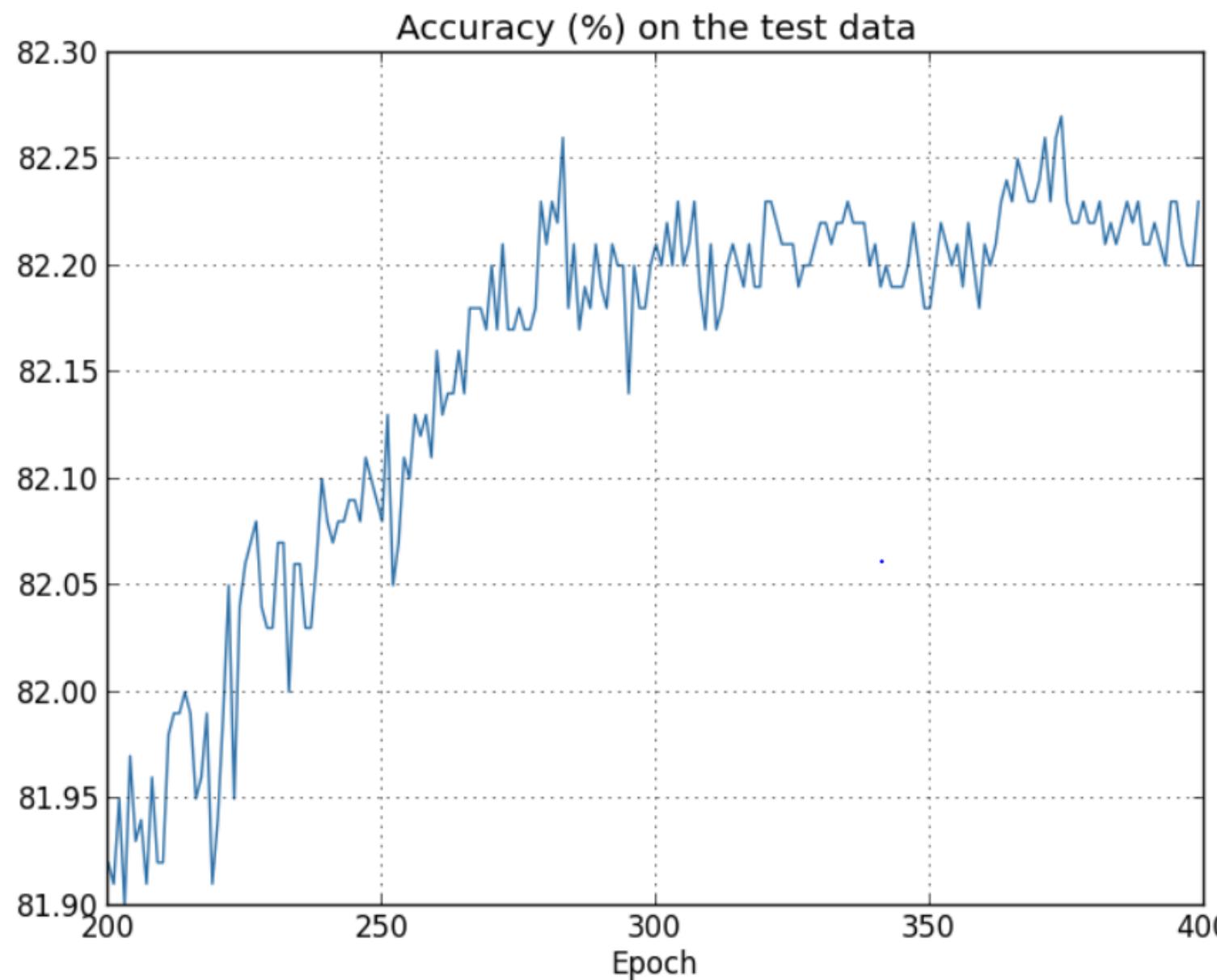
A rede neural é interrompida quando começa a ter uma degradação da rede. Quando é gerado muitas epochs pode-se levar a queda da performance da rede, para evitar esse tipo de situação, a early stopping identifica essa degradação e para as epochs antes do modelo não ficar performático.





## ALTA ACURÁCIA NO MODELO

Um modelo que obtêm grande taxa de acertos (~100%) pode conter dados muito ajustados, podendo ter decorado os dados e não performar corretamente a finalidade do algoritmo.



Dizemos que a rede está super adaptando ou com sobreajuste ou ainda com overfitting, a partir da época 280.

# REFERÊNCIAS:

<https://www.deeplearningbook.com.br/>