

ALÉM DO CÓDIGO

TRILHANDO O CAMINHO DAS REDES NEURAIS



GABRIELA SILVA

01

FUNDAMENTOS DAS REDES NEURAIS

Neste capítulo, mergulharemos nos fundamentos das redes neurais, explorando desde a inspiração biológica até os princípios matemáticos por trás delas. Entenderemos como os neurônios artificiais se conectam e como as redes aprendem.

REDES NEURAIS ARTIFICIAIS

Para entendermos nosso presente, é essencial olhar para o passado e analisar nossa trajetória:

O cérebro humano é uma "máquina" complexa e habilidosa, capaz de processar inúmeras informações em frações de segundos e armazenar dados na casa dos Terabytes. Suas unidades principais são os neurônios, que transmitem informações através de sinapses elétricas.



O desenvolvimento cerebral é mais intenso nos dois primeiros anos de vida, mas continua ao longo da vida. Inspirados nesse modelo, pesquisadores tentaram simular o funcionamento cerebral, especialmente o aprendizado por experiência, para criar sistemas inteligentes capazes de executar tarefas como classificação, reconhecimento de padrões e processamento de imagens. Dessa pesquisa surgiu o neurônio artificial e, posteriormente, as Redes Neurais.

LINHA DO TEMPO - REDES NEURAIS

-Warren McCulloch e Walter Pitts publicam um artigo sobre neurônios, modelando uma rede neural simples com circuitos elétricos
-Modelo baseado em lógica de limiar, iniciaram pesquisas sobre processos biológicos cerebrais e IA

1943

Donald Hebb publica "The Organization of Behavior", propondo que caminhos neurais são fortalecidos com uso repetido

1949

Avanços nos computadores permitem a simulação de redes neurais. Nathaniel Rochester tenta a primeira simulação, sem sucesso

1950

Projeto de Pesquisa de Verão de Dartmouth sobre IA impulsiona pesquisas em redes neurais

1956

Anos seguintes ao Projeto Dartmouth

-John von Neumann sugere imitar funções de neurônios com relés ou tubos de vácuo.
-Frank Rosenblatt trabalha no Perceptron, a rede neural mais antiga ainda em uso

1959

Bernard Widrow e Marcian Hoff desenvolvem ADALINE e MADALINE para reconhecimento de padrões e eliminação de ecos em linhas telefônicas

O Perceptron, comprovadamente limitado por Marvin Minsky e Seymour Papert em sua obra "Perceptron"

1969

1980

Kunihiko Fukushima propõe a Neocognitron, uma rede neural multicamada para reconhecimento de padrões

Críticas e poucos progressos levam à redução de financiamento, resultando no "Inverno da IA"

Até 1981

1982

John Hopfield revitaliza o interesse em redes neurais com abordagem prática

O Instituto Americano de Física inicia reuniões anuais sobre Redes Neurais para Computação

1985

1986

Redes neurais de múltiplas camadas e técnica de Backpropagation ganham destaque, melhorando a precisão dos resultados

Meados Anos 2000

Primeira Conferência
Internacional sobre Redes
Neurais do IEEE atrai mais de
1.800 participantes

1987

Algoritmos de redes neurais
profundas são criados, mas tempos
de treinamento são impraticáveis

Juyang Weng publica o
Cresceptron para reconhecimento
automático de objetos 3D

1992

Termo “aprendizagem
profunda” populariza-se com
artigo de Geoffrey Hinton e
Ruslan Salakhutdinov

NIPS Workshop sobre Aprendizagem Profunda descobre que grandes conjuntos de dados eliminam necessidade de pré-treinamento

2009

2012

Algoritmos de reconhecimento de padrões alcançam desempenho em nível humano; Google identifica imagens de gatos, através do aprendizado profundo

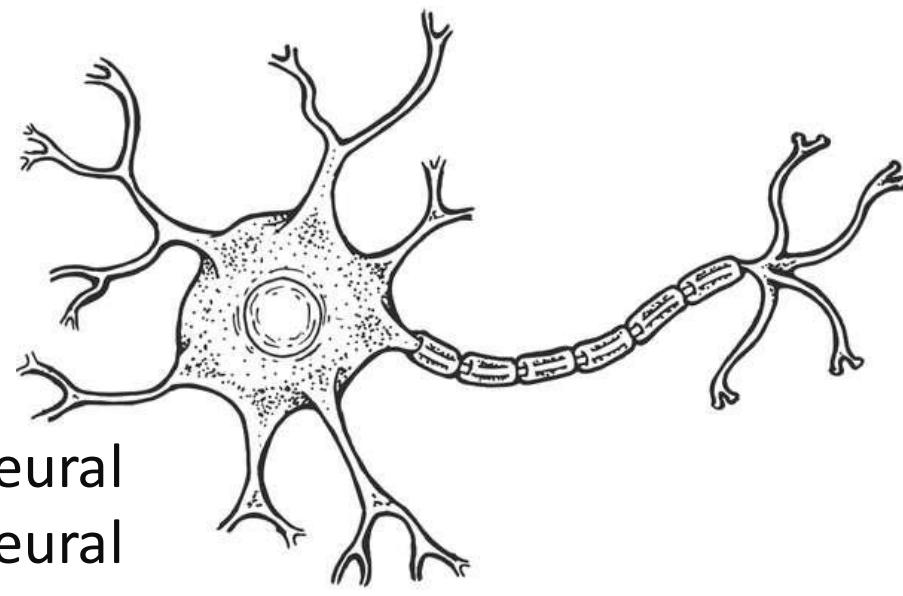
Facebook implementa DeepFace para reconhecimento facial automático, usando 120 milhões de parâmetros

2015

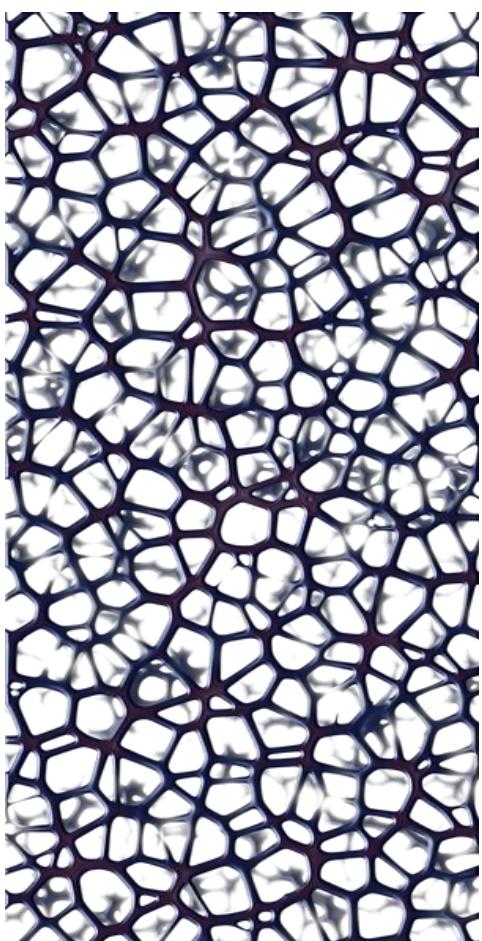
2017

Deep Learning adota-se amplamente em aplicações corporativas e mobile, impulsionando IA e Big Data

NEURONIOS BIOLOGICOS



A ideia de criar uma rede neural artificial veio a partir de uma rede neural biológica. Ou seja, essa tecnologia foi inspirada na rede neural já existente.



Ao analisarmos a anatomia de um neurônio, fica evidente que ela é muito complexa. Nela há vários elementos que fazem com que um neurônio seja, de fato, um neurônio. Entretanto, até hoje, ainda não se sabe completamente como cada uma dessas células funciona em conjunto.

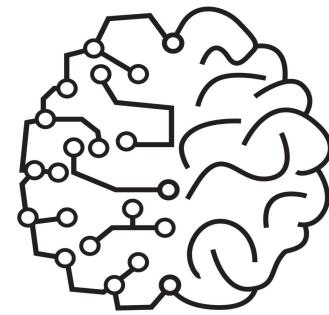
Sabe-se, porém, que existem vários tipos deles e que cada neurônio se conecta com muitos outros (em média sete mil conexões, cada), essas conexões são chamadas sinapses.

Existe, portanto, uma malha muito complexa de conexões entre os neurônios do cérebro humano.

CURIOSIDADE - O Cérebro Humano possui cerca de oitenta e seis bilhões de células nervosas!💡

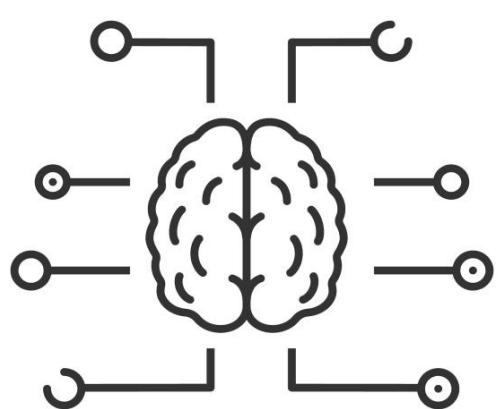
Certamente estamos longe de saber como o pensamento humano funciona exatamente, mas a ideia de interconexão entre neurônios e a transferência de informações entre eles pode ser aplicada no meio digital.

NEURONIOS ARTIFICIAIS



Temos um vasto campo de conhecimento sobre redes neurais. Antes de explorarmos mais profundamente, precisamos entender a arquitetura mais simples de uma rede neural artificial: o Perceptron.

O Perceptron foi o primeiro neurônio artificial modelado, também chamado de neurônio matemático. Foi desenvolvido nas décadas de 1950 e 1960 pelo cientista Frank Rosenblatt, inspirado nos trabalhos de Warren McCulloch e Walter Pitts.



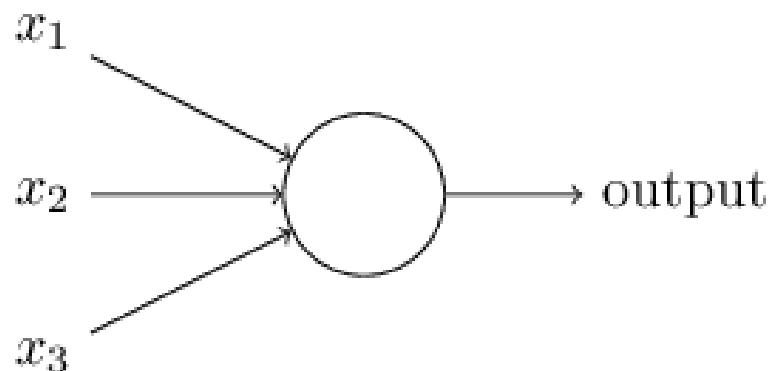
A ideia inicial do Perceptron era criar uma unidade lógica de processamento que recebesse diferentes sinais de entrada. Esse conceito tentava imitar o funcionamento de um neurônio biológico: o neurônio estaria ativado ou desativado, ou seja, ou estaria transmitindo informações para outros neurônios, ou não.

Naquela época, acreditava-se que o cérebro funcionava assim: determinados neurônios recebiam informações de outros neurônios e, com base nessas entradas, decidiam se iriam transmitir informações adiante.

Hoje em dia, usamos outros modelos de neurônios artificiais, mas o Perceptron ainda é fundamental para entender como uma rede neural funciona em termos matemáticos, sendo assim, excelente forma de introdução ao tema.

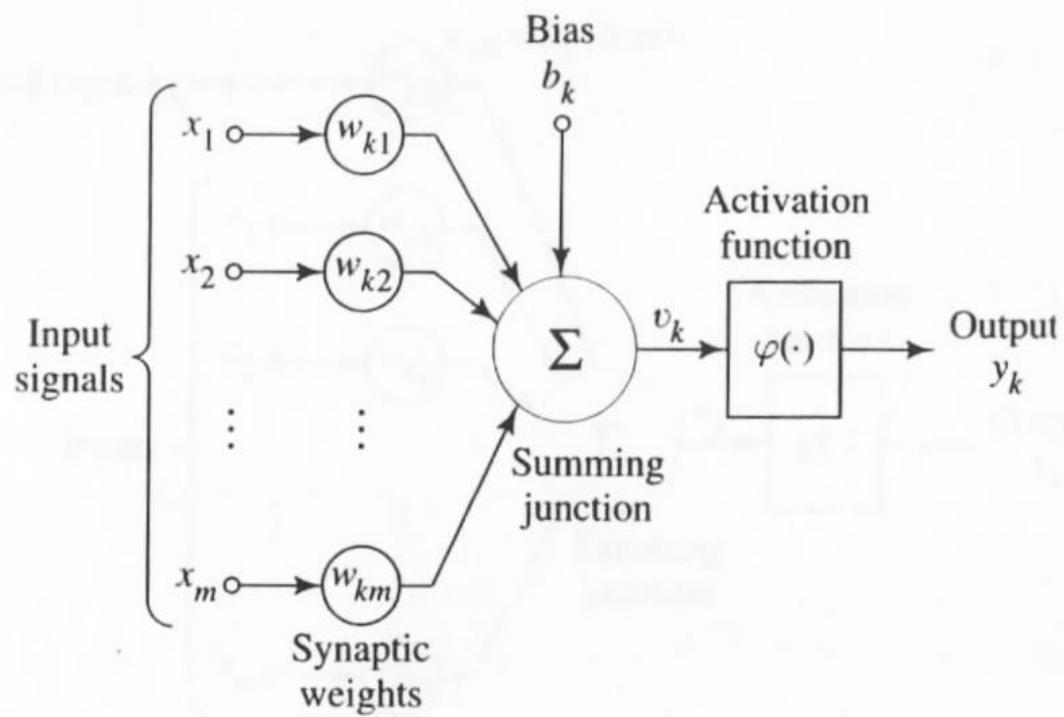
NEURONIOS ARTIFICIAIS

Na prática, um Perceptron é um modelo matemático que recebe várias entradas, x_1, x_2, \dots e produz uma única saída binária. Nesse exemplo, o Perceptron possui três entradas: x_1, x_2, x_3 .



Rosenblatt criou uma regra simples para calcular a saída usando pesos (w_1, w_2 , etc.), que indicam a importância de cada entrada.

A saída, 0 ou 1, é determinada pela soma ponderada das entradas, $\sum w_j x_j$, comparada a um valor limiar (Threshold). Assim como os pesos, o Threshold é um número real que é um parâmetro do neurônio.



Curiosidade -
Descobriu-se que o cérebro é muito mais complexo do que apenas células estarem ativas ou inativas; não é só uma questão de envio ou não de informações!💡



EXPLICANDO O PERCEPTRON USANDO RPG

IMAGINE QUE VOCÊ É UM HERÓI EM UM JOGO DE RPG E PRECISA DECIDIR SE DEVE ATACAR UM INIMIGO. VOCÊ CONSIDERA TRÊS FATORES:

- FORÇA DO INIMIGO (X1)
- NÍVEL DE SAÚDE DO SEU PERSONAGEM (X2)
- PRESENÇA DE ALIADOS PRÓXIMOS (X3)

PARA CADA FATOR, VOCÊ ATRIBUI UM PESO QUE REPRESENTA SUA IMPORTÂNCIA:

- FORÇA DO INIMIGO: $w_1 = 0.6$
- NÍVEL DE SAÚDE: $w_2 = 0.8$
- PRESENÇA DE ALIADOS: $w_3 = 1.0$

A LÓGICA É A SEGUINTE:

- SE O INIMIGO É FORTE, É MAIS PERIGOSO ATACÁ-LO, ENTÃO ISSO TEM UM PESO ALTO.
- SE SUA SAÚDE ESTÁ ALTA, VOCÊ ESTÁ MAIS CONFIANTE PARA ATACAR, ENTÃO ISSO TEM UM PESO ALTO.
- SE HÁ ALIADOS PRÓXIMOS, VOCÊ SE SENTE MAIS SEGURO PARA ATACAR, ENTÃO TAMBÉM TEM UM PESO ALTO.



AGORA, VAMOS CALCULAR A SOMA PONDERADA:

$$\text{SOMA PONDERADA} = \\ (X_1 \times W_1) + (X_2 \times W_2) + (X_3 \times W_3)$$

VAMOS SUPOR QUE:

- O INIMIGO É FORTE ($X_1 = 1$)
- SUA SAÚDE ESTÁ MÉDIA ($X_2 = 0.5$)
- HÁ ALIADOS PRÓXIMOS ($X_3 = 1$)



SUBSTITUINDO OS VALORES:

$$\text{SOMA PONDERADA} = \\ (1 \times 0.6) + (0.5 \times 0.8) + (1 \times 1.0) = 0.6 + 0.4 + 1.0 = 2.0$$

AGORA, VOCÊ COMPARA A SOMA PONDERADA COM UM VALOR LIMIAR (THRESHOLD).

VAMOS DIZER QUE O THRESHOLD É 1.5.

- SE A SOMA PONDERADA (2.0) FOR MAIOR QUE O THRESHOLD (1.5), VOCÊ DECIDE ATACAR O INIMIGO (SAÍDA 1).
- SE A SOMA PONDERADA FOR MENOR OU IGUAL AO THRESHOLD, VOCÊ DECIDE NÃO ATACAR (SAÍDA 0).

NESTE CASO, COMO 2.0 É MAIOR QUE 1.5, VOCÊ DECIDE ATACAR O INIMIGO.

ASSIM, O PERCEPTRON PODE AJUDAR A TOMAR DECISÕES ESTRATÉGICAS NO JOGO, COM BASE NA IMPORTÂNCIA DOS FATORES CONSIDERADOS RELEVANTES.

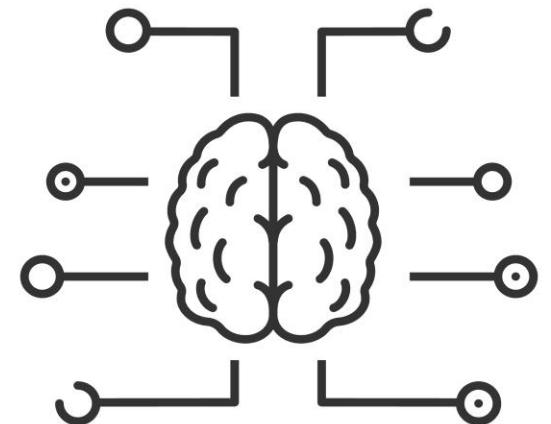
02

ARQUITETURA DE REDES NEURAIS

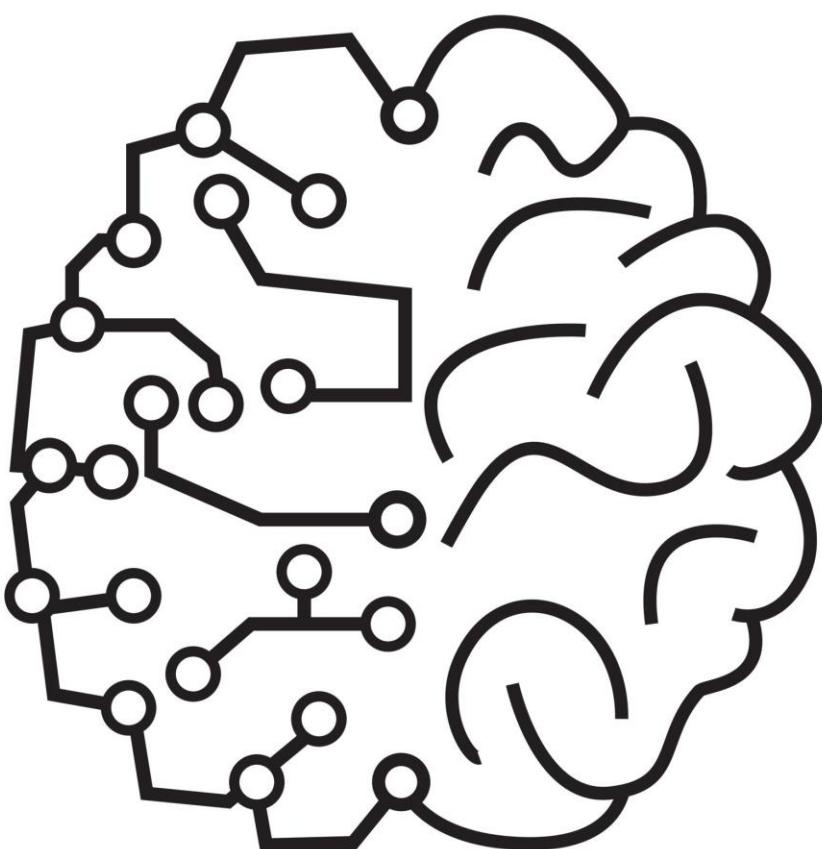
Aqui, examinaremos diferentes arquiteturas de Redes Neurais, desde as simples até as mais complexas. Discutiremos Redes Feedforward, Redes Recorrentes e Redes Convolucionais, destacando suas aplicações e diferenças.

FEEDFORWARDS E BACKPROPAGATION

No capítulo anterior, aprendemos que o perceptron imita um neurônio humano e permite que o computador aprenda a distinguir entre duas classes separáveis linearmente. No entanto, o perceptron não consegue resolver problemas como o "OU" exclusivo (XOR), que requer duas retas para separar as classes.



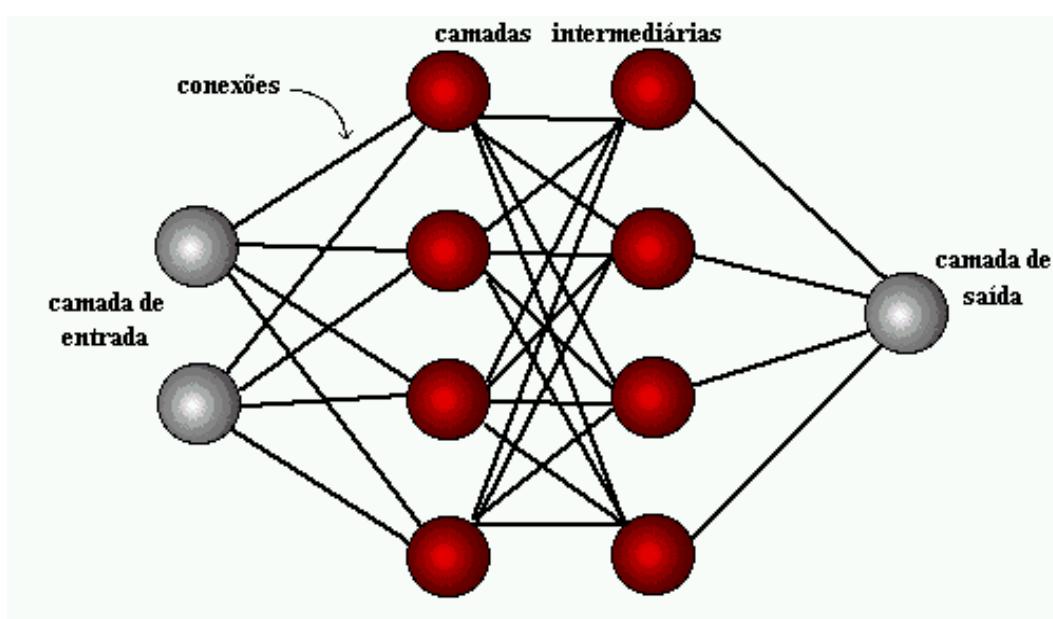
A solução é aumentar o número de neurônios, criando assim uma rede neural artificial (RNA) onde vários neurônios trabalham juntos para resolver o problema. Vamos usar redes neurais FeedForward para entender melhor essa dinâmica.



As redes feedforward são as mais comuns em aplicações práticas. Pois, elas são redes que possuem uma estrutura onde a primeira camada é a de entrada e a última é a de saída. Quando há mais de uma camada oculta, essas redes são chamadas de "redes neurais profundas" (Deep Learning).

REDES NEURAIS FEEDFORWARD

Nessa rede, cada camada está conectada à próxima, sem caminhos de volta. As atividades dos neurônios em cada camada são funções não-lineares das atividades na camada anterior. Todas as conexões seguem a mesma direção, da entrada para a saída.

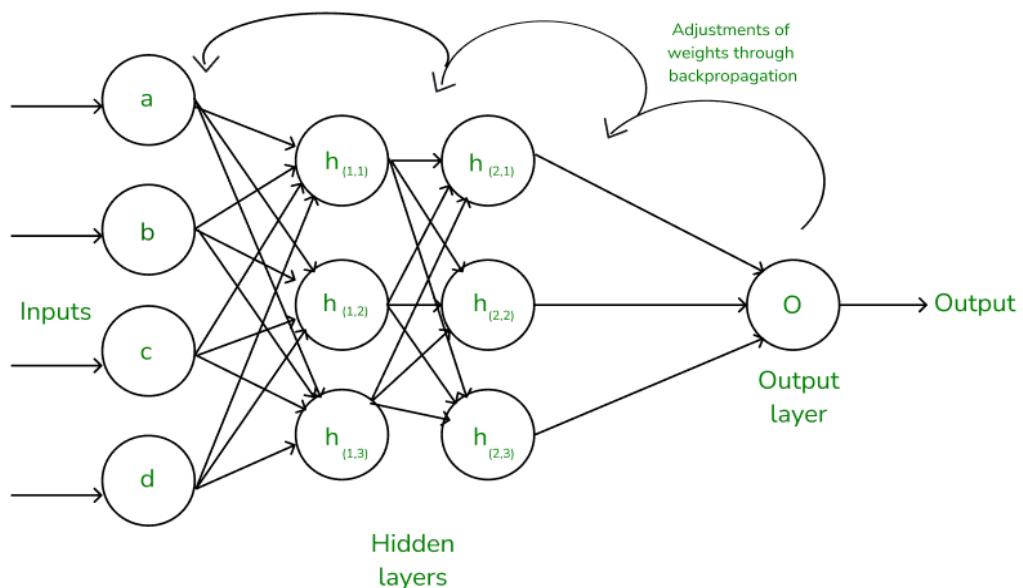


A **camada de entrada** de uma rede neural recebe informações iniciais. Cada neurônio dessa camada representa uma variável independente que afeta o resultado final da rede.

As **camadas Intermediárias** extraem características dos padrões de entrada. Seus pesos codificam essas características, permitindo que a rede crie uma representação mais rica e complexa do problema.

A **camada de Saída** envia um resultado para o ambiente externo. O número de neurônios nessa camada depende da tarefa da rede neural. Para classificadores, o número de neurônios geralmente é igual ao número de grupos distintos a serem identificados.

BACKPROPAGATION



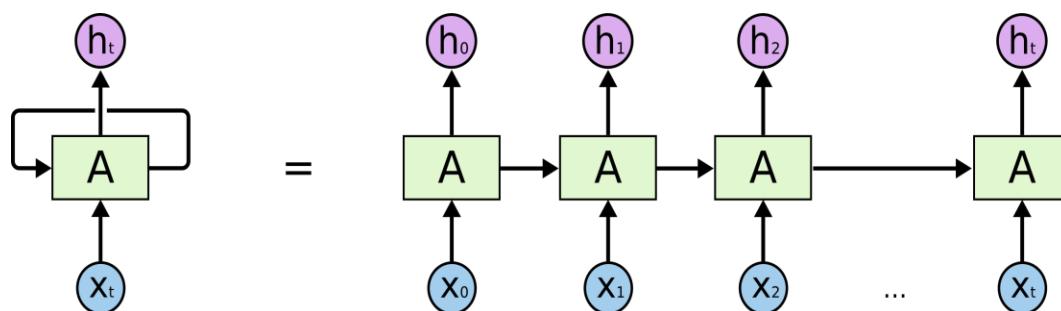
Backpropagation é um método de treinamento de redes neurais que ajusta os pesos usando gradiente descendente. Ele calcula o erro e corrige os pesos das camadas da saída para a entrada.

As redes neurais artificiais aprendem continuamente usando feedbacks corretivos para melhorar suas previsões. Os dados fluem do nó de entrada para o nó de saída por vários caminhos na rede. Apenas um caminho é correto e leva ao resultado certo. Para encontrar esse caminho, a rede usa um sistema de **Encaminhamento de feedbacks**:

- Cada nó na rede neural faz uma estimativa sobre qual será o próximo nó no caminho.
- A rede verifica se o palpite foi correto. Os nós atribuem valores mais altos aos caminhos que levaram a palpites precisos e valores mais baixos aos caminhos que levaram a palpites incorretos.
- Para o próximo conjunto de dados, os nós fazem novas previsões usando os caminhos com pesos mais altos. Esse processo é repetido para cada novo dado, ajustando continuamente os pesos dos caminhos com base na precisão das previsões anteriores.

REDES NEURAIS RECORRENTES

Como discutido anteriormente, há uma variedade de tipos de redes neurais, começando pelo Perceptron até as redes feedforward. Agora, exploraremos as redes recorrentes, que introduzem um aspecto dinâmico fascinante no processamento de dados sequenciais.



Características Básicas:

- Redes neurais recorrentes têm ciclos direcionados em suas conexões, permitindo retornar ao ponto de partida através das setas.
- São mais realistas biologicamente, mas sua dinâmica complexa torna o treinamento desafiador.

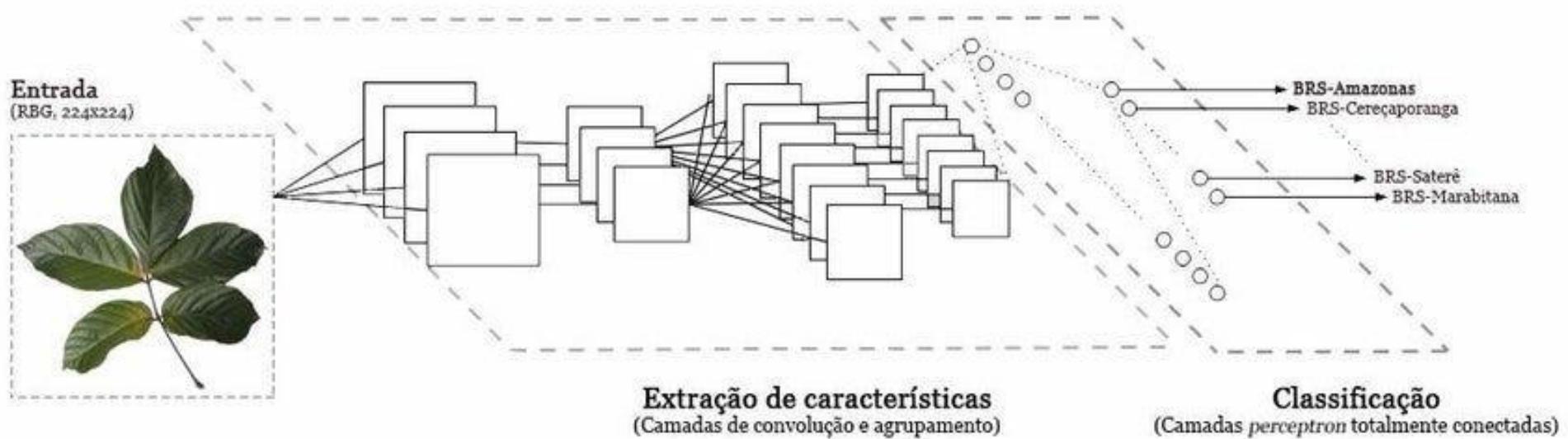
Aplicações e Desafios:

- São usadas para modelar dados sequenciais de forma natural.
- Equivalentes a redes profundas com camadas ocultas em cada passo temporal, compartilhando os mesmos pesos.
- Podem reter informações em seu estado oculto por longos períodos, mas é difícil treiná-las para usar esse potencial.

REDES NEURAIS CONVOLUCIONAIS

Redes Neurais Convolucionais são algoritmos de Aprendizado Profundo usados para analisar imagens de maneira eficiente. Elas extraem características importantes, como bordas e texturas, usando operações matemáticas chamadas convoluções.

CNNs são fundamentais em visão computacional para tarefas como classificação de imagens, detecção de objetos e reconhecimento facial, aprendendo padrões complexos diretamente das imagens.



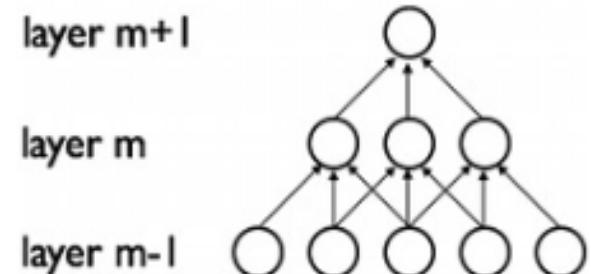
Benefícios e Aplicações

Essas redes são altamente eficazes para tarefas de classificação de imagens e reconhecimento de objetos.

Cada nó na rede tem pesos e um limite associados; se a saída de um nó exceder o limite, ele é ativado e envia dados para a próxima camada.

As redes neurais convolucionais se distinguem de outras redes neurais por seu desempenho superior com entradas de imagem, fala ou sinal de áudio. Eles têm três tipos principais de camadas, que são:

- 1- Camada Convolucional
- 2- Camada de Agrupamento
- 3- Camada Totalmente Conectada (FC)

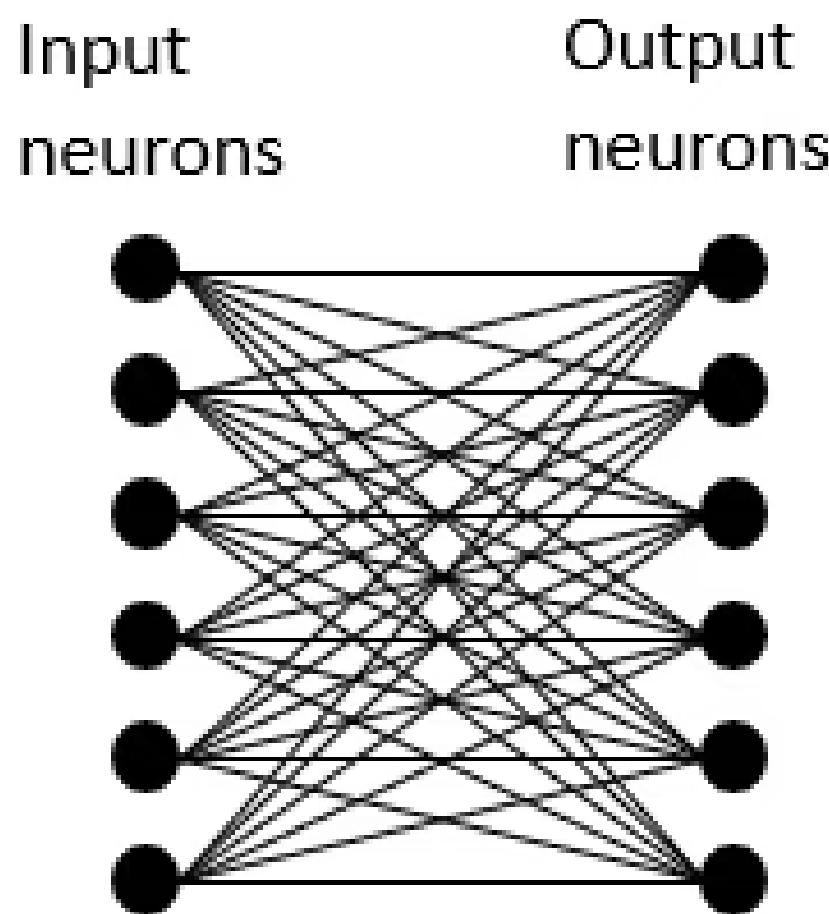


A **Camadas Convolucional** realiza uma operação de convolução na entrada, passando o resultado para a próxima camada. Essa operação converte todos os pixels em seu campo receptivo em um único valor.

Ao aplicar uma convolução a uma imagem, o tamanho da imagem é reduzido e todas as informações do campo são condensadas em um único pixel. A saída final da camada convolucional é um vetor.

A **Camadas de Agrupamento**, ou Downsampling, reduz a dimensionalidade e o número de parâmetros na entrada. Semelhante às Camadas Convolucionais, a operação de agrupamento varre um filtro por toda a entrada, mas sem pesos. O kernel aplica uma função de agregação aos valores dentro do campo receptivo, preenchendo a matriz de saída.

A **Camada Totalmente Conectada**, localizada no final da rede, permite a classificação da imagem com base nas características extraídas pelos blocos de processamento anteriores. Ela é chamada de totalmente conectada porque todas as entradas da camada estão conectadas aos neurônios de saída.



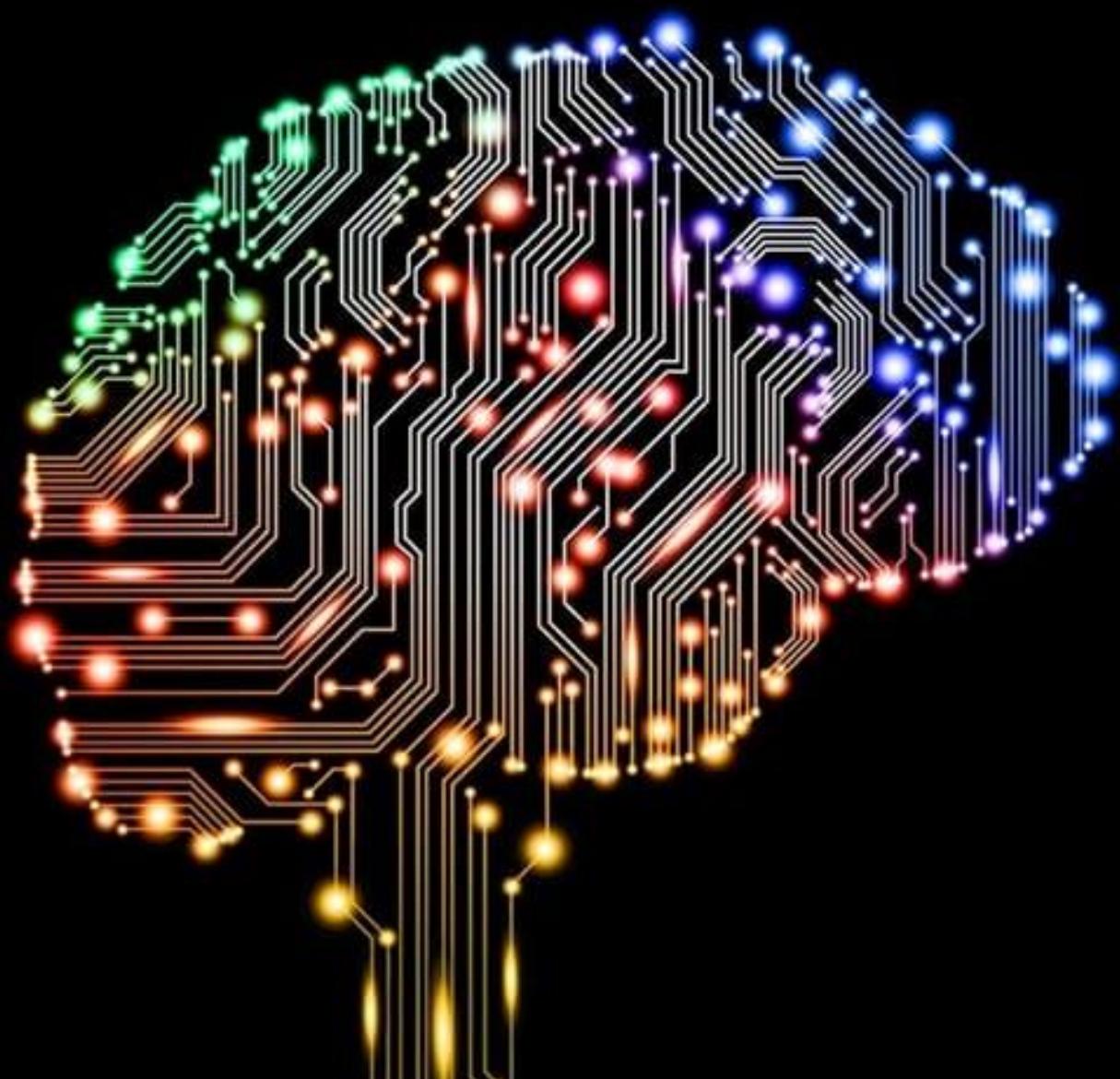
CURIOSIDADE – Dentro das Redes Neurais Convolucionais ainda existem outras camadas mais profundas com outras finalidades, como a **Camada de Correção (ReLU)**, **Camada Pooling (POOL)**, **Camada de Perda (LOSS)** e ainda possuem os **Hiperparâmetros**, que permitem monitorar a aprendizagem automática em grandes dimensões.💡

03

TREINAMENTO E OTIMIZAÇÃO

Dedicaremos este capítulo ao processo de treinamento das redes neurais. Abordaremos algoritmos de otimização, como o gradiente descendente, e técnicas para lidar com problemas de sobreajuste e underfitting.

TREINAMENTO DE REDES NEURAIS

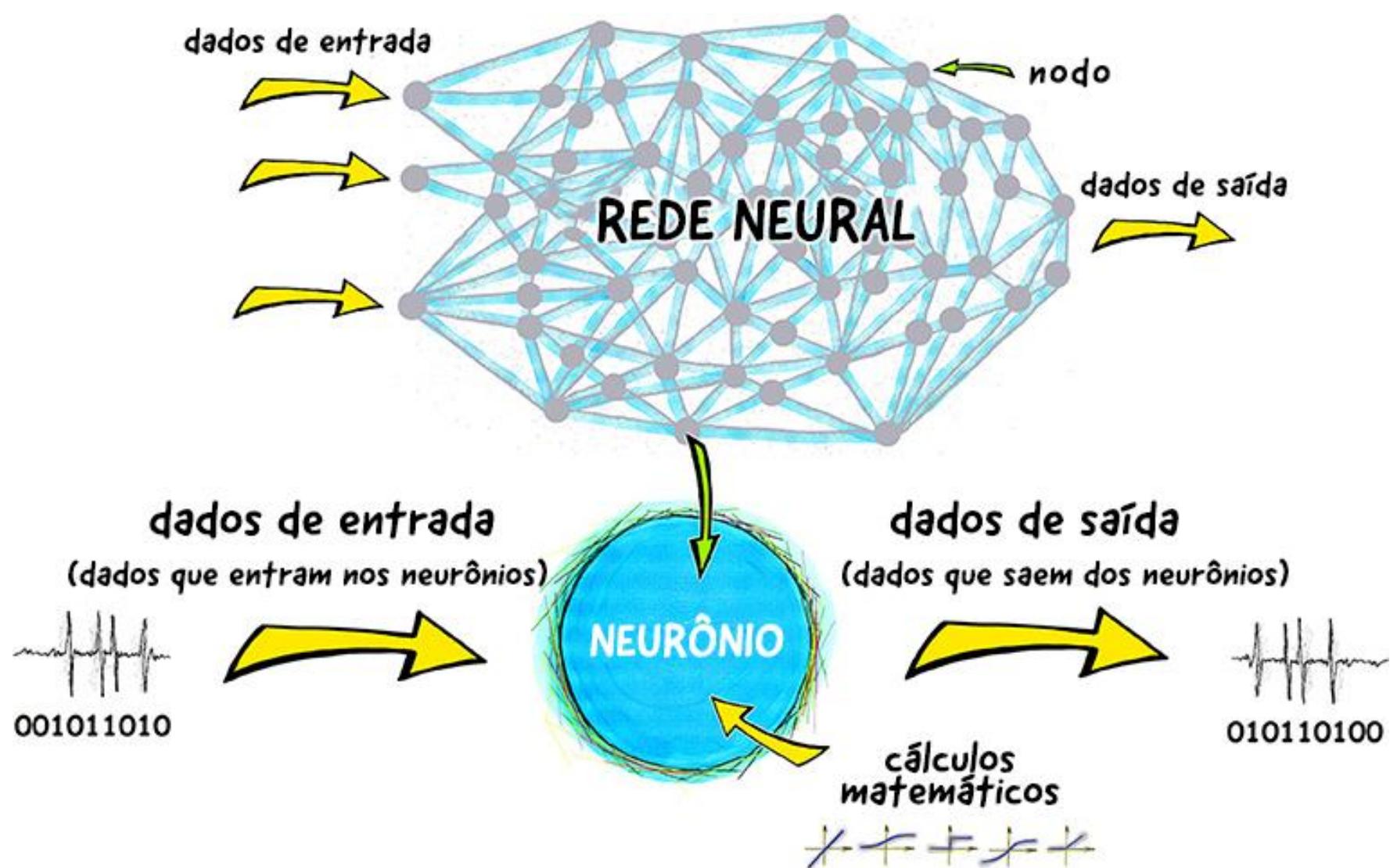


Como vimos anteriormente, o processo de treinamento de redes neurais envolve o aprendizado através da conexão de milhares de neurônios, também chamados de unidades de processamento. Esses neurônios estão interligados por canais de comunicação, cada um com um peso específico.

Em uma rede neural artificial, pode haver centenas ou até milhares desses "neurônios" conectados, permitindo que a rede aprenda e processe informações complexas.

Resumidamente, o treinamento de uma rede neural é o processo de ensinar a rede a executar uma tarefa. As redes aprendem processando grandes conjuntos de dados rotulados ou não rotulados. Esse aprendizado envolve ajustar os pesos das conexões entre neurônios.

O algoritmo de backpropagation faz ajustes contínuos nesses pesos durante o treinamento, minimizando a diferença entre a saída prevista e a desejada.

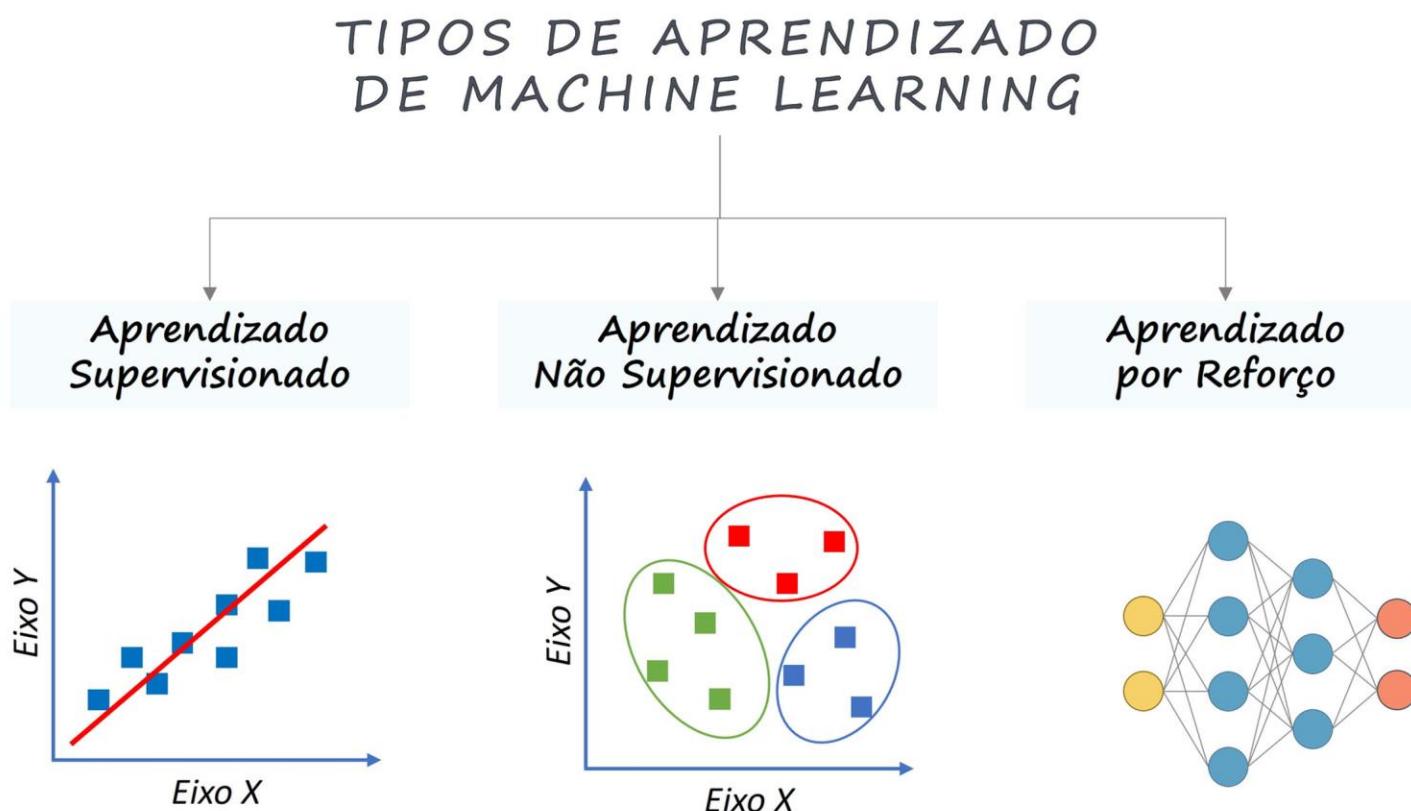


TREINAMENTO DE REDES NEURAIS

Um algoritmo de aprendizado é um conjunto de regras para resolver problemas de aprendizado. Existem muitos tipos específicos para diferentes modelos de redes neurais, e eles diferem principalmente na forma como os pesos são ajustados.

A relação da rede neural com o ambiente define três paradigmas de aprendizado:

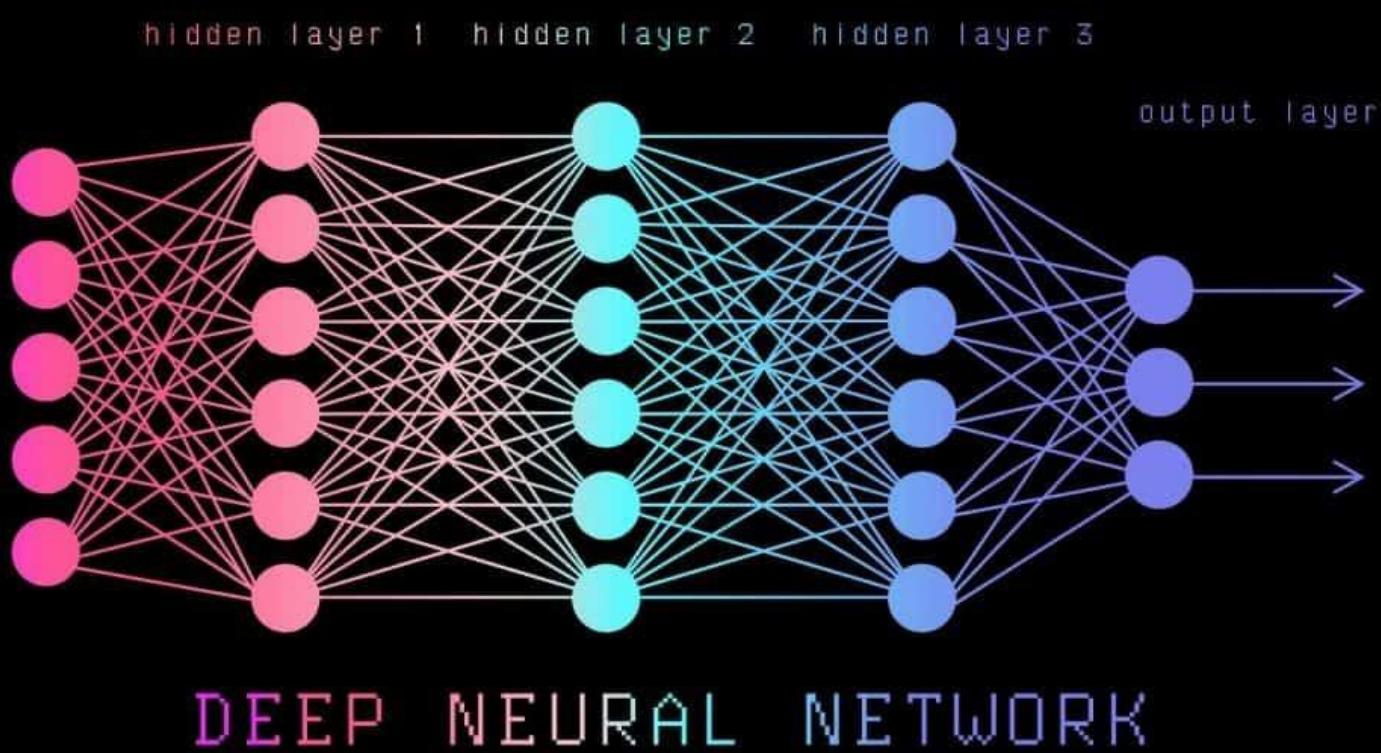
- **Aprendizado Supervisionado:** um agente externo indica a resposta desejada para a entrada.
- **Aprendizado Não Supervisionado:** não há agente externo, e a rede organiza os padrões de entrada por conta própria.
- **Aprendizado por Reforço:** um crítico externo avalia a resposta fornecida pela rede.



No processo de aprendizado de uma rede neural, um ciclo é a apresentação de todos os N pares (entrada e saída) do conjunto de treinamento. A correção dos pesos pode ser feita de duas maneiras:

Modo Padrão: A correção dos pesos ocorre a cada apresentação de um exemplo do conjunto de treinamento, baseada no erro desse exemplo específico. Em cada ciclo, ocorrem N correções.

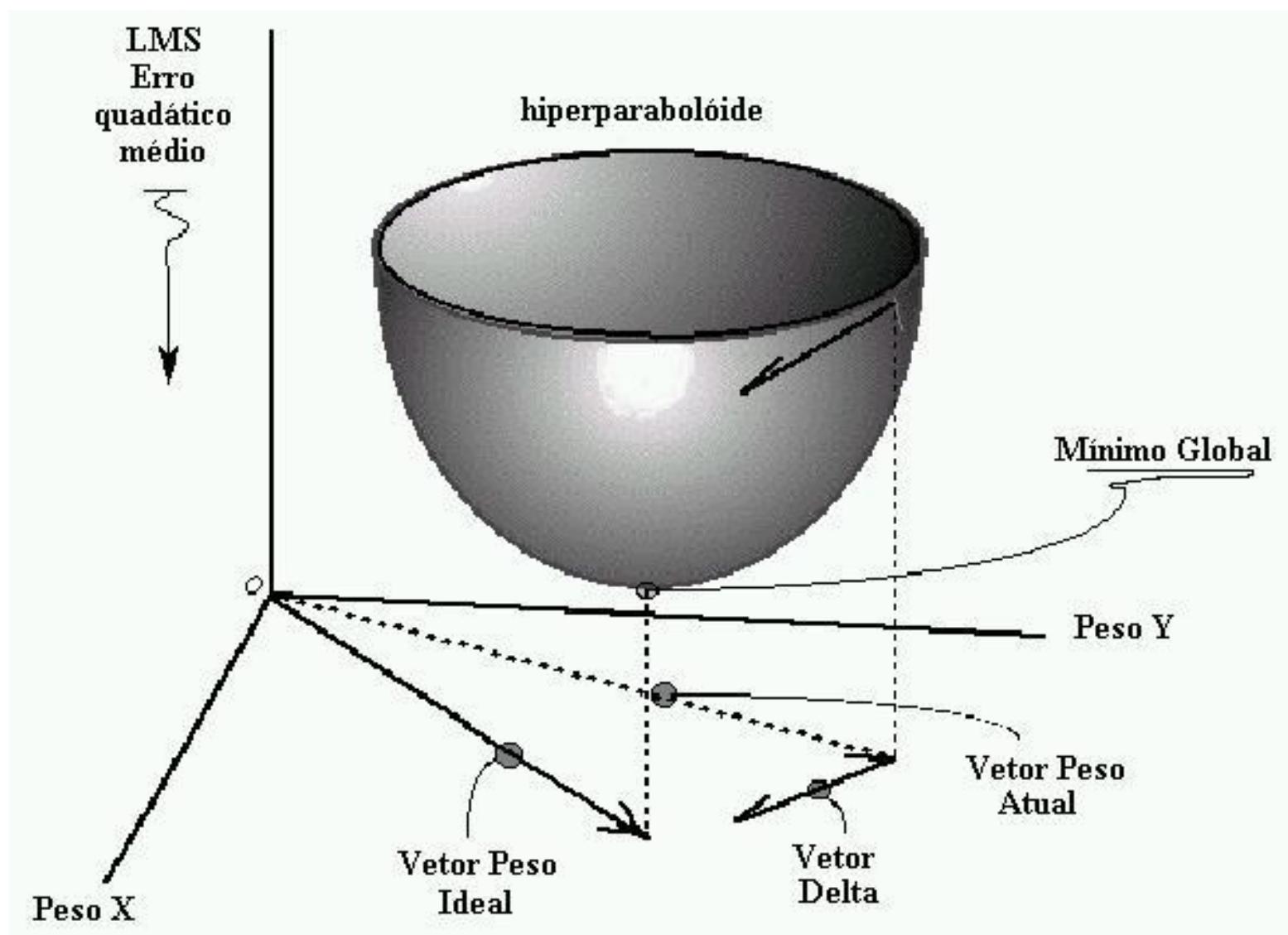
Modo Batch: Uma única correção é feita por ciclo, após apresentar todos os exemplos do conjunto de treinamento. O erro médio é calculado e as correções dos pesos são feitas com base nesse erro.



APRENDIZADO SUPERVISIONADO

O treinamento supervisionado do modelo de rede Perceptron ajusta os pesos e thresholds das unidades para obter a classificação desejada. O threshold é tratado como um peso com entrada sempre igual a -1, ajustando o peso relativo a essa entrada.

Quando um padrão é apresentado à rede, ela gera uma saída. A diferença entre a resposta atual e a desejada é medida, e os pesos são ajustados para reduzir essa diferença. Esse método é chamado de **Regra Delta**, como representado na figura a seguir:



ESQUEMA DE TREINAMENTO

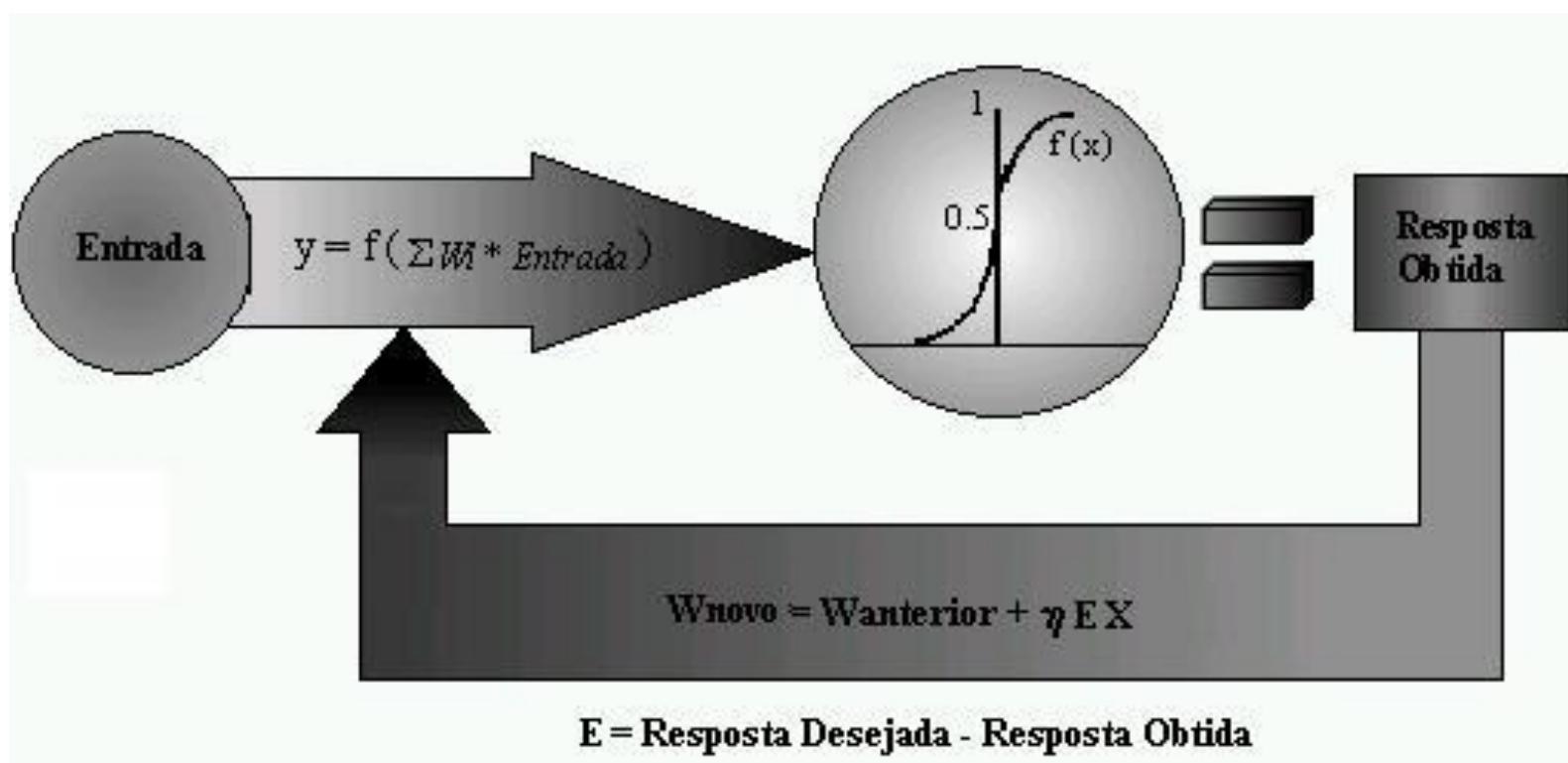
- 1- Iniciar todas as conexões com pesos aleatórios;
- 2- Repita até que o erro E seja satisfatoriamente pequeno ($E = e$)
- 3- Para cada par de treinamento (X, d), faça:
- 4- Calcular a resposta obtida (O);

Se o erro não for satisfatoriamente pequeno ($E > e$), então:

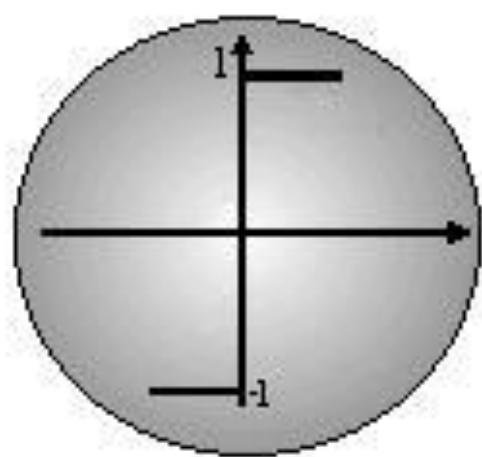
- 5- Atualizar pesos: $W_{\text{novo}} := W_{\text{anterior}} + \eta E X$

Onde:

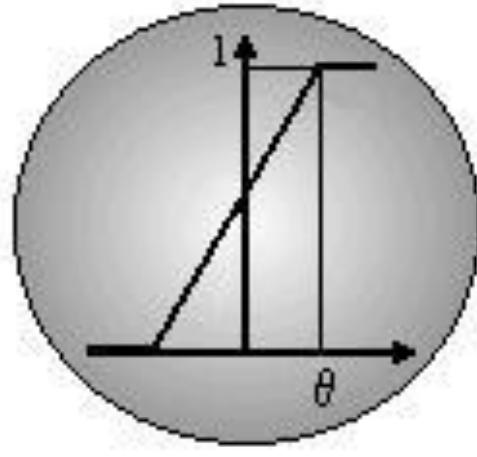
- O par de treinamento (X, d) corresponde ao padrão de entrada e a sua respectiva resposta desejada;
- O erro E é definido como: Resposta Desejada - Resposta Obtida ($d - O$);
- A taxa de aprendizado neta é uma constante positiva, que corresponde à velocidade do aprendizado.



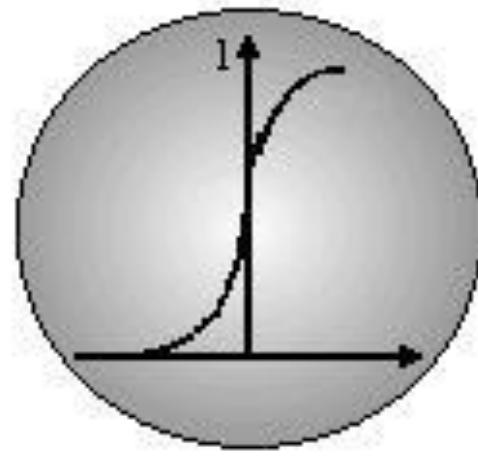
As respostas geradas pelas unidades são calculadas através de uma função de ativação. Existem vários tipos de funções de ativação, as mais comuns são: **Hard Limiter**, **Threshold Logic** e **Sigmoid**.



Hard Limiter



Threshold Logic



Sigmoid

Hard Limiter: Esta função transforma a entrada em 0 ou 1. Se a entrada for maior que um certo valor (threshold), a saída é 1; caso contrário, é 0.

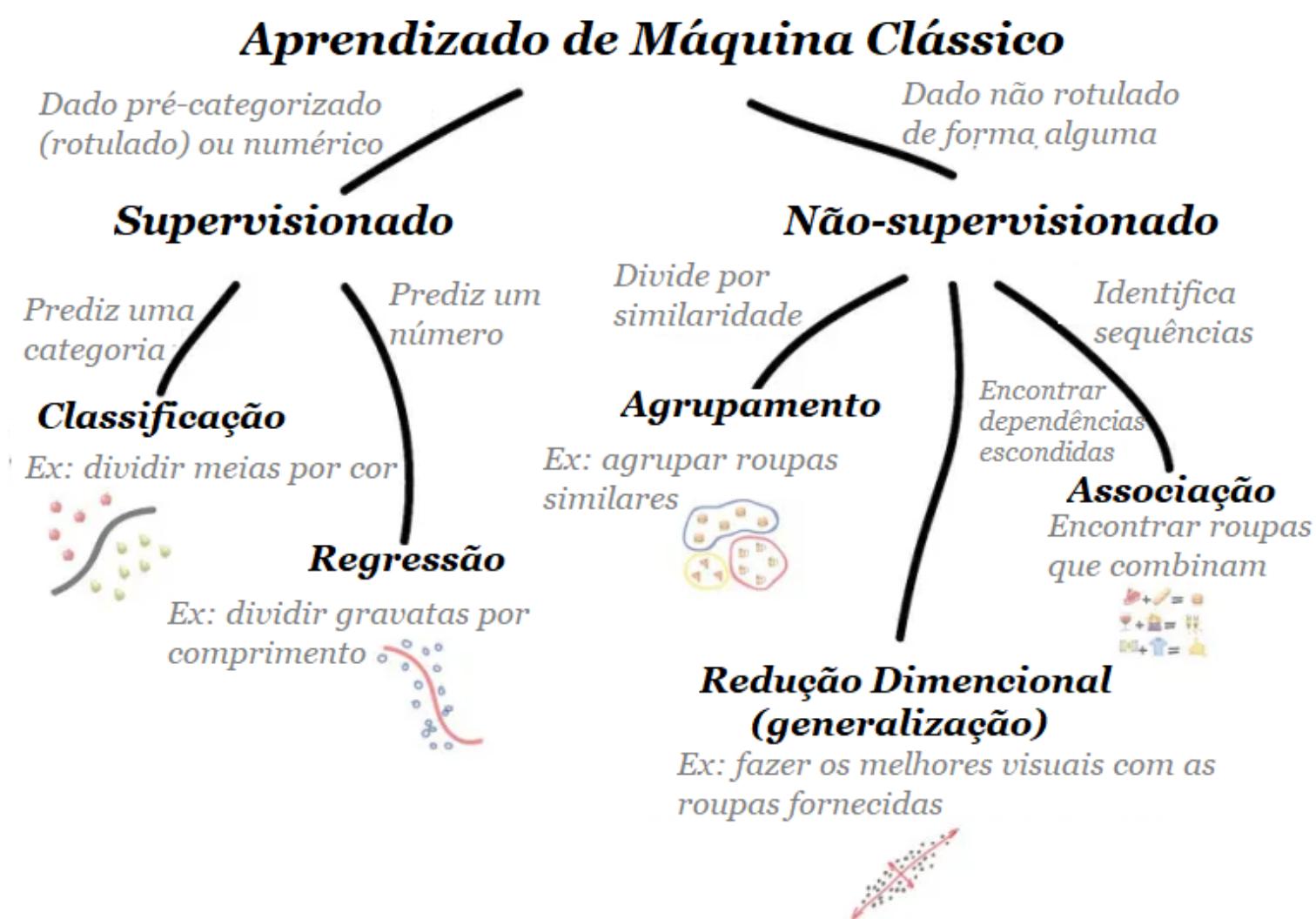
Threshold Logic: Similar ao Hard Limiter, essa função compara a entrada com um valor de threshold. Se a entrada excede o threshold, a saída é 1; se não, a saída é 0. É usada para tomar decisões binárias.

Sigmoid: Esta função suaviza a transformação da entrada, produzindo um valor entre 0 e 1. Quanto maior a entrada, mais próximo de 1 é o resultado; quanto menor a entrada, mais próximo de 0. É usada em redes neurais para introduzir não-linearidade e ajudar na convergência do modelo.

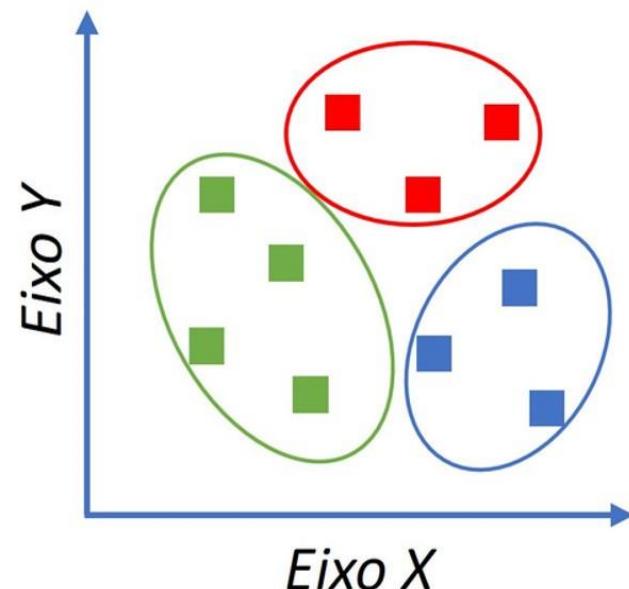
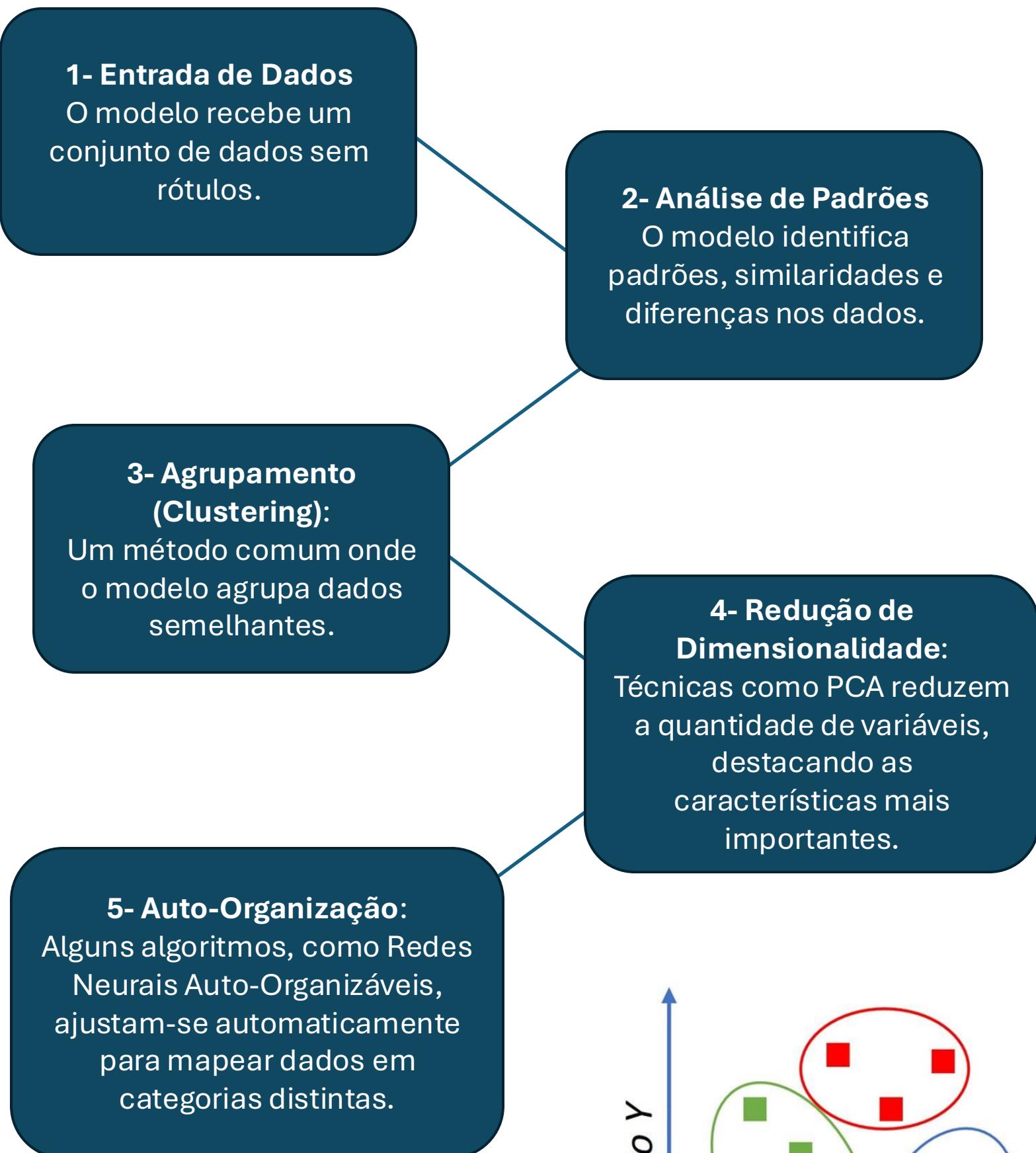
APRENDIZADO NÃO SUPERVISIONADO

O aprendizado não supervisionado é uma técnica onde o modelo aprende padrões e estruturas dos dados sem rótulos ou respostas pré-definidas.

Além disso, é útil para explorar dados desconhecidos, encontrar agrupamentos naturais e reduzir a dimensionalidade, o que facilita a visualização e a compreensão dos dados complexos.



PROCESSO DE TREINAMENTO



APRENDIZADO POR REFORÇO

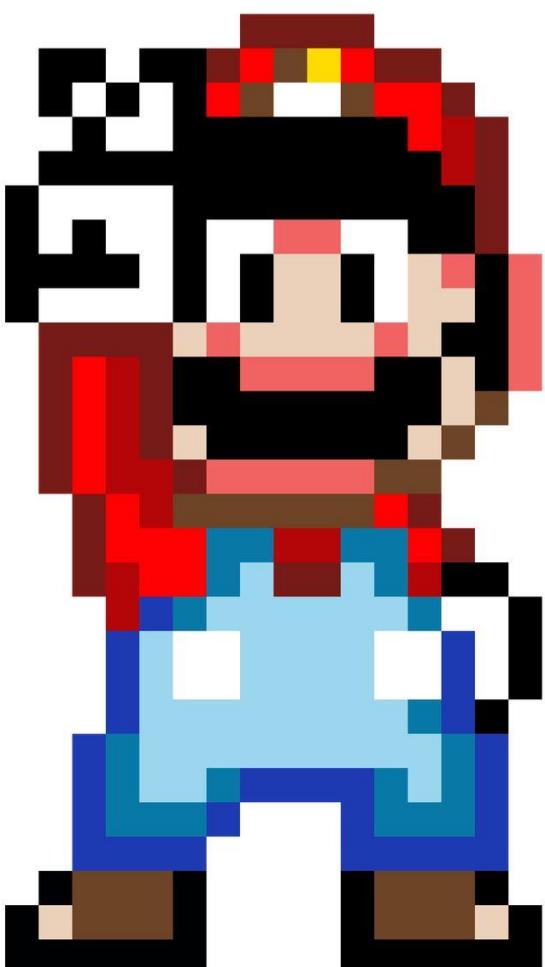
O aprendizado por reforço é uma técnica onde um agente aprende a tomar decisões ao interagir com um ambiente, visando maximizar uma recompensa acumulada.

Além disso, é utilizado em situações onde as decisões precisam ser feitas em sequência, como jogos, controle de robôs e sistemas de recomendação.

Ele permite que o agente aprenda a melhorar seu desempenho através da experiência direta no ambiente.



PROCESSO DE TREINAMENTO

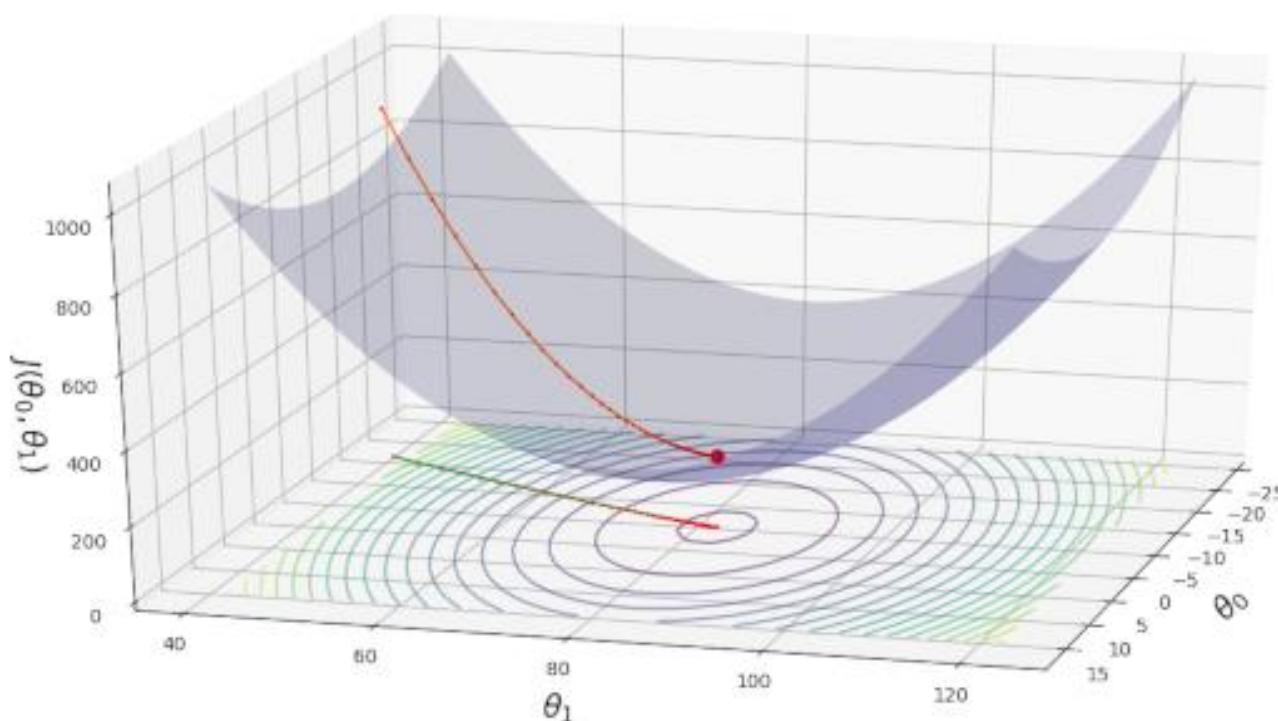


ALGORITMOS DE OTIMIZAÇÃO

Os algoritmos de otimização são usados em redes neurais para ajustar os pesos dos modelos e minimizar a função de perda, que mede a diferença entre as previsões do modelo e os valores reais. Vejamos os principais algoritmos de otimização utilizados em redes neurais:

Gradient Descent (GD): É o algoritmo de otimização mais simples. Ele ajusta os pesos da rede neural na direção oposta ao gradiente da função de perda. O método é baseado em uma função convexa e ajusta os parâmetros de forma iterativa para minimizar o erro.

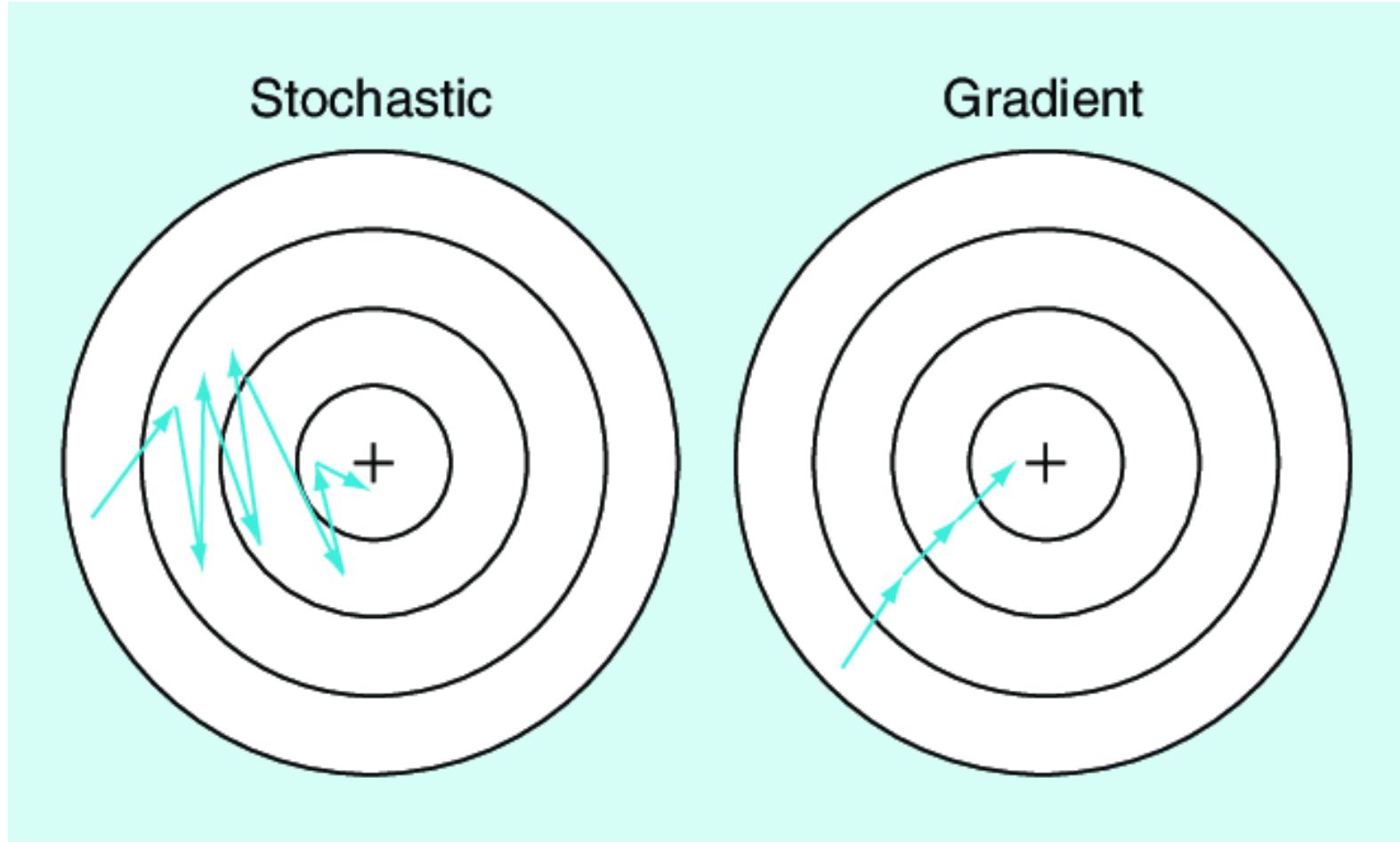
Seu objetivo é encontrar os valores dos coeficientes que minimizam a função de erro, $E(x)$, o máximo possível. O Gradient Descent ajusta os valores dos pesos, (θ), ou parâmetros, gradualmente, até encontrar um valor mínimo.



ALGORITMOS DE OTIMIZAÇÃO

Stochastic Gradient Descent (SGD): Em vez de calcular o gradiente em todo o conjunto de dados (como no GD), o SGD calcula o gradiente usando apenas um único exemplo de treinamento ou um pequeno lote de exemplos.

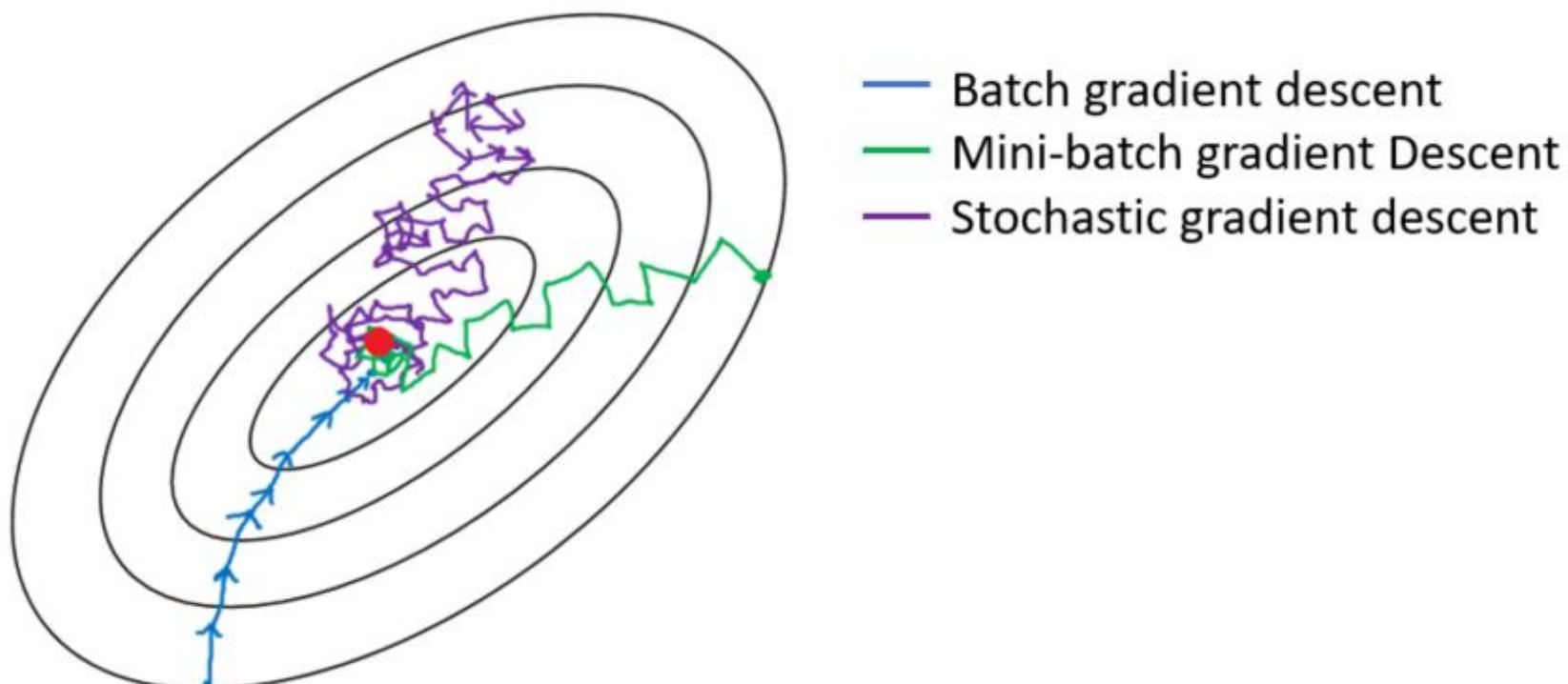
Esse método pode escapar de mínimos locais e é mais rápido em termos de tempo de processamento por iteração. Porém, tem mais variação no gradiente, o que pode levar a uma convergência menos estável.



ALGORITMOS DE OTIMIZAÇÃO

Mini-Batch Gradient Descent: Combina os benefícios do GD e do SGD ao calcular o gradiente em pequenos lotes de exemplos de treinamento. Em cada iteração, o algoritmo calcula o gradiente da função de custo em relação aos parâmetros do modelo usando um mini-batch de dados.

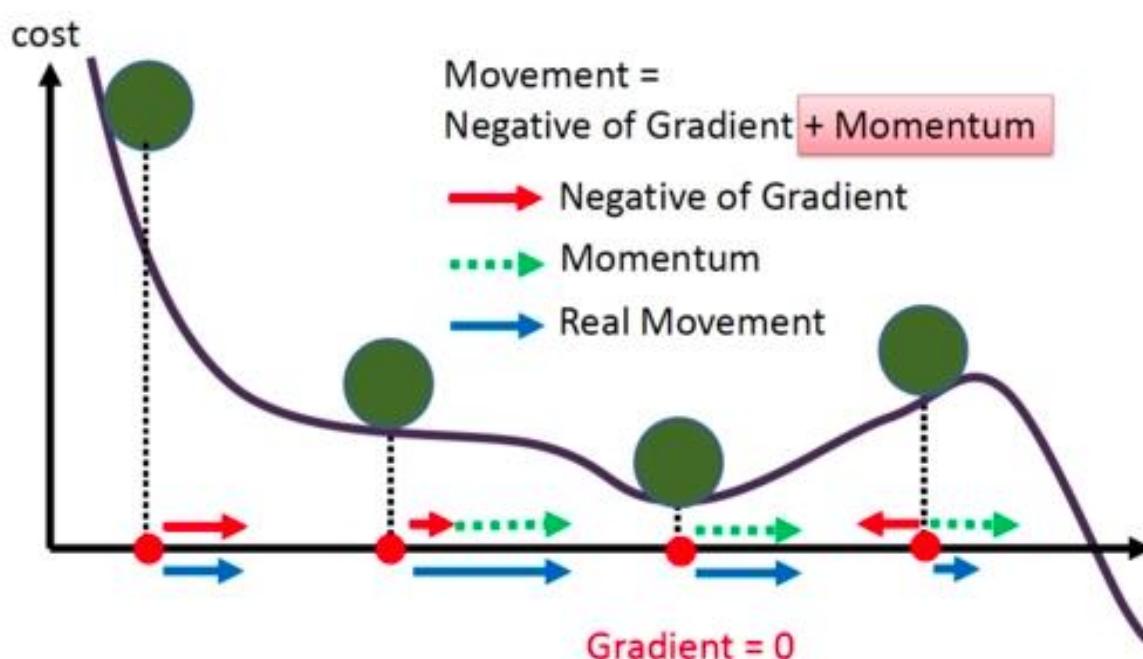
Em seguida, ele atualiza os parâmetros do modelo dando um passo na direção oposta ao gradiente calculado. Por conta disso, este método acaba sendo mais eficiente computacionalmente do que o GD e menos ruidoso do que o SGD.



ALGORITMOS DE OTIMIZAÇÃO

Momentum: É uma técnica usada em algoritmos de otimização para acelerar a convergência e evitar ficar preso em mínimos locais durante o treinamento de redes neurais. Ele adiciona um termo que lembra as atualizações anteriores dos gradientes, ajudando a suavizar e acelerar o caminho de descida. Suas vantagens são:

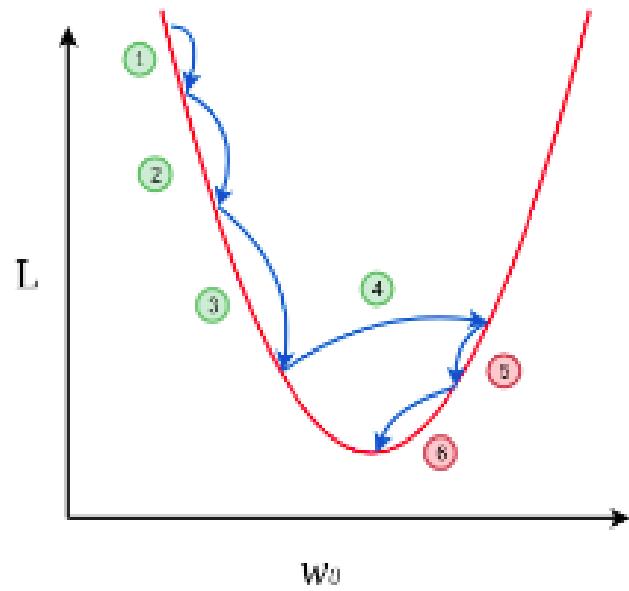
- **Convergência Mais Rápida:** Acelera o processo de otimização em direções onde os gradientes são consistentes.
- **Suavização do Caminho de Descida:** Reduz oscilações e variações bruscas nas atualizações dos pesos.
- **Escape de Mínimos Locais:** Ajuda a evitar ficar preso em mínimos locais, permitindo encontrar melhores soluções.]
- **Estabilidade:** Melhora a estabilidade do treinamento, resultando em uma trajetória de descida mais suave e eficiente.



ALGORITMOS DE OTIMIZAÇÃO

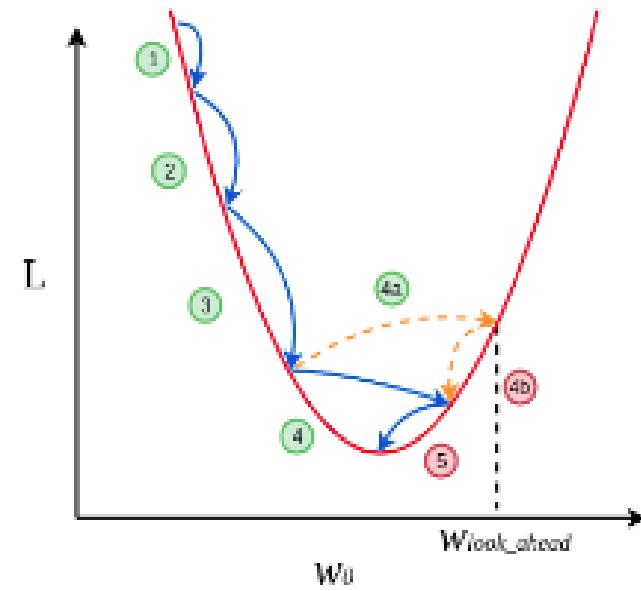
Nesterov Accelerated Gradient (NAG): Uma versão melhorada do Momentum que calcula o gradiente não na posição atual, mas em uma posição extrapolada, ajudando a prevenir saltos grandes em direção a mínimos locais.

NAG calcula o gradiente em uma posição antecipada, prevendo a próxima atualização dos pesos. Isso resulta em ajustes mais precisos e eficientes, acelerando o treinamento. Além disso, NAG suaviza o caminho de descida de forma mais eficaz que o Momentum, evitando grandes variações e promovendo uma trajetória mais estável.



(a) Momentum-Based Gradient Descent

$$\text{Green Circle} \Rightarrow \frac{\partial L}{\partial w_0} = \frac{\text{Negative}(-)}{\text{Positive}(+)}$$



(b) Nesterov Accelerated Gradient Descent

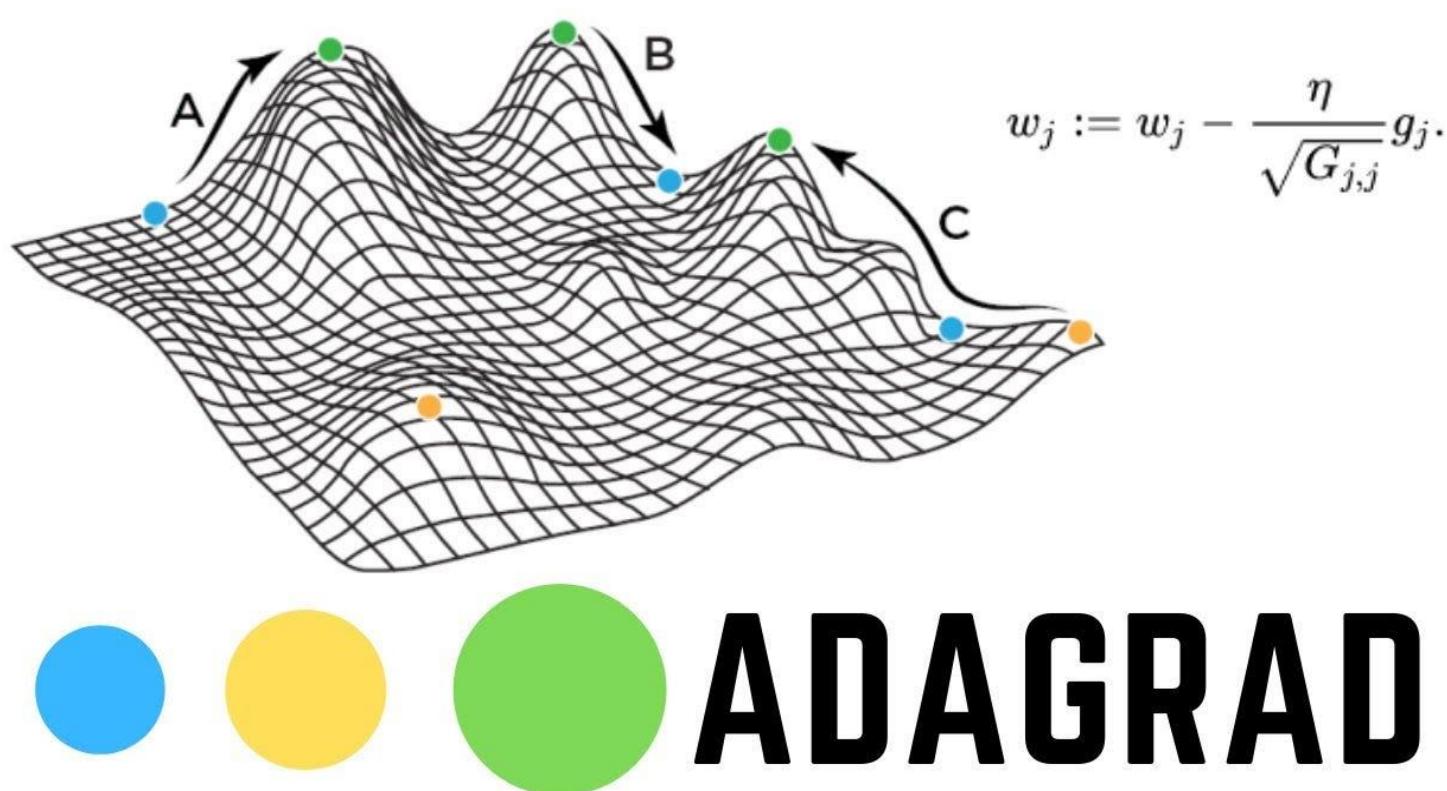
$$\text{Red Circle} \Rightarrow \frac{\partial L}{\partial w_0} = \frac{\text{Negative}(-)}{\text{Negative}(-)}$$

ALGORITMOS DE OTIMIZAÇÃO

Adagrad (Adaptive Gradient Algorithm): É um algoritmo de otimização que ajusta a taxa de aprendizado para cada parâmetro individualmente, permitindo uma adaptação mais fina e eficiente durante o treinamento de modelos, especialmente em problemas com gradientes esparsos.

Imagine que estamos treinando um modelo de machine learning para classificar imagens de dígitos escritos à mão (como no dataset MNIST).

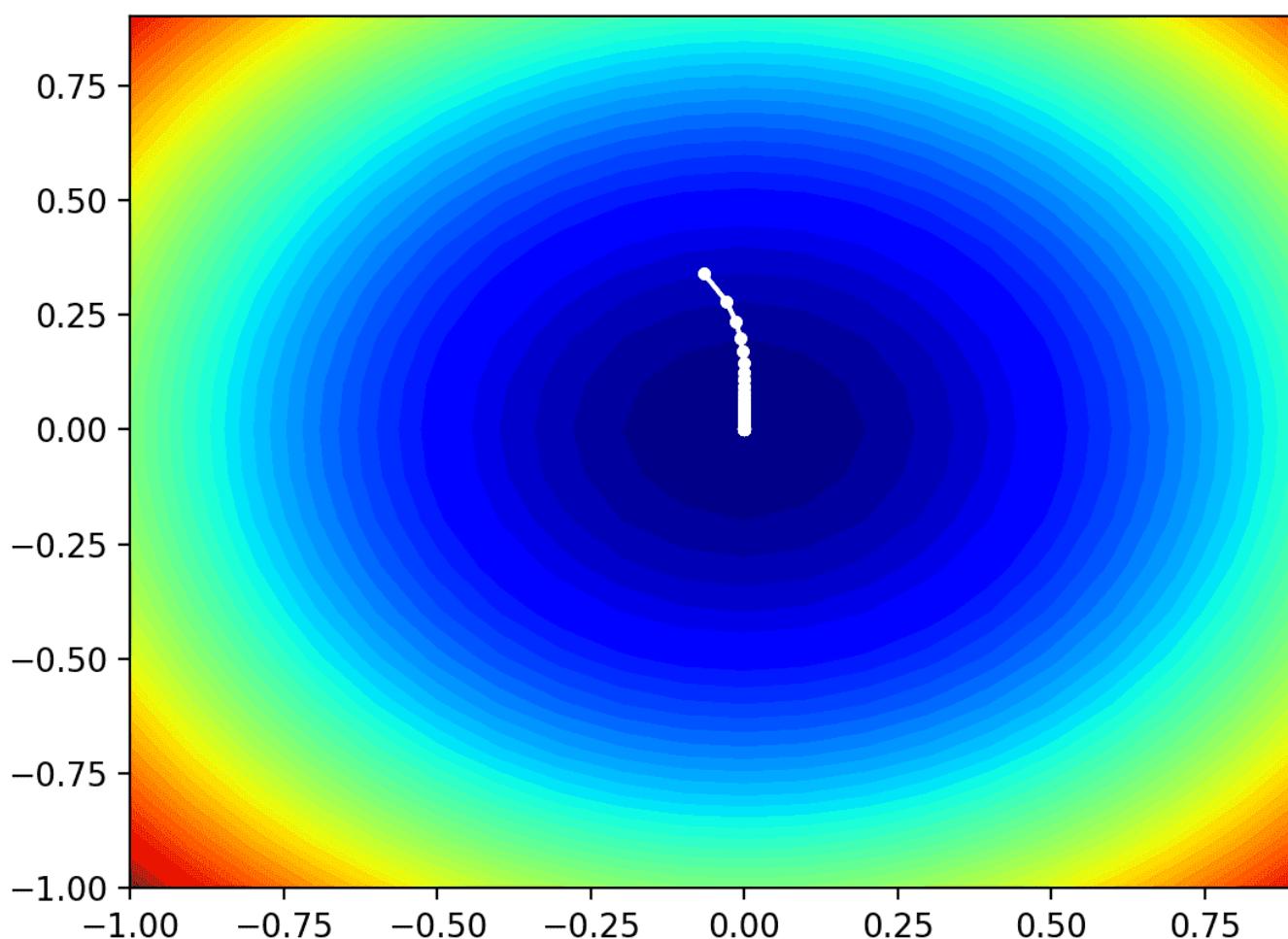
Com Adagrad, os parâmetros associados a pixels menos informativos (que raramente são ativados) terão suas taxas de aprendizado aumentadas, enquanto os parâmetros associados a pixels mais informativos (frequentemente ativados) terão suas taxas de aprendizado diminuídas.



ALGORITMOS DE OTIMIZAÇÃO

RMSprop (Root Mean Square Propagation): É um algoritmo de otimização que modifica o Adagrad para melhorar a eficiência do treinamento de modelos de aprendizado de máquina, especialmente em cenários onde a taxa de aprendizado do Adagrad se torna muito pequena com o tempo.

Imagine que você está tentando descer uma colina com muitas ondulações. Adagrad tende a ser muito cauteloso e pode diminuir sua velocidade excessivamente. RMSprop, por outro lado, ajusta sua velocidade de acordo com a direção em que está se movendo, permitindo um movimento mais suave e eficiente.



ALGORITMOS DE OTIMIZAÇÃO

Adam (Adaptive Moment Estimation): É um algoritmo de otimização amplamente utilizado no treinamento de redes neurais devido à sua eficiência e robustez. Ele combina as vantagens do RMSprop e do Momentum, ajustando dinamicamente a taxa de aprendizado para cada parâmetro e acumulando gradientes passados para acelerar a convergência.

Aqui temos um exemplo de como usar Adam com a biblioteca TensorFlow:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# Criar um modelo de rede neural simples
model = Sequential([
    Dense(64, activation='relu', input_shape=(784,)),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

# Compilar o modelo com o otimizador Adam
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Treinar o modelo em dados de exemplo
model.fit(train_images, train_labels, epochs=10, batch_size=32)
```

snappyf.com

PREVENÇÃO DE SOBREAJUSTES

Embora os algoritmos de otimização como Gradient Descent, Momentum, RMSprop, e Adam sejam eficazes para melhorar a convergência e a eficiência do treinamento de redes neurais, lidar diretamente com problemas de sobreajuste (overfitting) geralmente envolve outras técnicas específicas.

Aqui veremos algumas dessas técnicas que ajudam a prevenir o sobreajuste:

Regularização (L1 e L2)

L1 Regularization (Lasso): Adiciona uma penalidade igual à soma absoluta dos valores dos coeficientes. Incentiva a criação de pesos esparsos, ajudando a reduzir a complexidade do modelo.

L2 Regularization (Ridge): Adiciona uma penalidade igual à soma dos quadrados dos valores dos coeficientes. Isso incentiva pesos menores, evitando que o modelo se ajuste demais aos dados de treinamento.

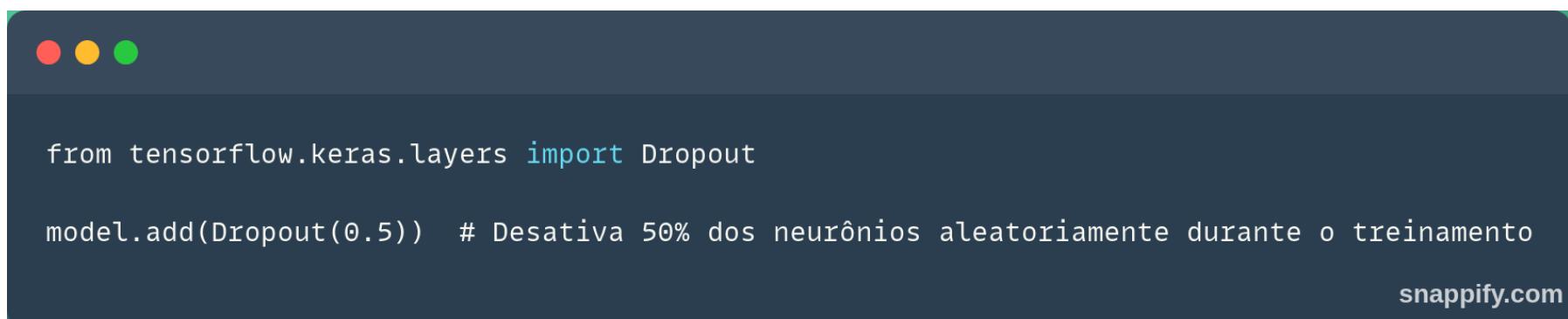


```
from tensorflow.keras.regularizers import l2  
  
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.01)))
```

snappify.com

PREVENÇÃO DE SOBREAJUSTES

Dropout: É uma técnica onde, durante o treinamento, unidades (neurônios) da rede neural são desativadas (definidas como zero) aleatoriamente em cada iteração. Isso impede que a rede dependa demais de unidades específicas, promovendo a generalização.



```
from tensorflow.keras.layers import Dropout  
  
model.add(Dropout(0.5)) # Desativa 50% dos neurônios aleatoriamente durante o treinamento
```

snappify.com

Early Stopping: Monitora o desempenho do modelo em dados de validação e interrompe o treinamento quando o desempenho não melhora por um número especificado de épocas, prevenindo o sobreajuste.



```
from tensorflow.keras.callbacks import EarlyStopping  
  
early_stopping = EarlyStopping(monitor='val_loss', patience=5)  
model.fit(train_images, train_labels, epochs=100, validation_split=0.2, callbacks=[early_stopping])
```

snappify.com

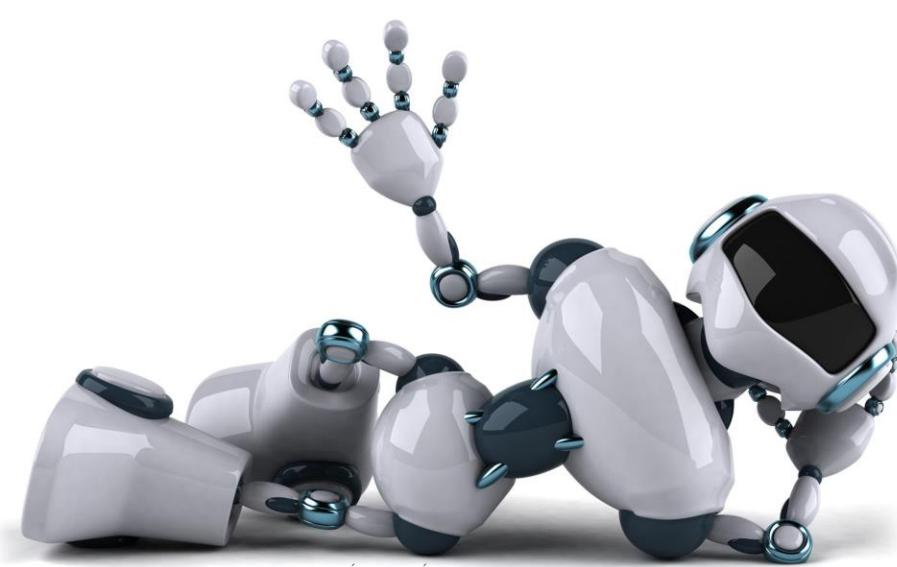
PREVENÇÃO DE SOBREAJUSTES

Data Augmentation: Envolve criar novas amostras de dados a partir dos dados de treinamento existentes, aplicando transformações como rotação, translação, zoom e espelhamento. Isso aumenta a diversidade dos dados de treinamento e melhora a capacidade de generalização do modelo.



```
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
  
datagen = ImageDataGenerator(  
    rotation_range=20,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True  
)  
  
datagen.fit(train_images)
```

snappyf.com



ALÉM DO CÓDIGO - GABRIELA SILVA

PREVENÇÃO DE SOBREAJUSTES

Batch Normalization: normaliza as ativações de uma camada para uma média e variância fixas, o que pode ajudar a estabilizar e acelerar o treinamento, além de ter um efeito de regularização leve.



```
from tensorflow.keras.layers import BatchNormalization  
model.add(BatchNormalization())
```

snappyf.com

K-Fold Cross-Validation: Divide os dados de treinamento em K subconjuntos e treina o modelo K vezes, cada vez usando um subconjunto diferente como dados de validação e os restantes como dados de treinamento. Isso fornece uma estimativa mais robusta do desempenho do modelo.



PREVENÇÃO DE SOBREAJUSTES

Resumidamente, Os algoritmos de otimização, como Adam, RMSprop e Momentum, são fundamentais para treinar redes neurais de forma eficiente e eficaz, permitindo que modelos complexos aprendam a partir de grandes conjuntos de dados.

Eles podem ser combinados com técnicas para prevenir sobreajuste, melhorando a eficiência do treinamento e garantindo que o modelo generalize bem aos novos dados.

Pontos principais:

- Importância dos algoritmos de otimização.
- Treinamento eficiente e eficaz de redes neurais.
- Prevenção de sobreajuste.
- Melhoria na generalização dos modelos aos novos dados.



EFFICIENCY

UNDERFITTING

O underfitting ocorre quando um modelo de rede neural é incapaz de capturar a estrutura subjacente dos dados de treinamento, resultando em um desempenho insuficiente tanto nos dados de treinamento quanto nos dados de validação ou teste. Em outras palavras, o modelo é muito simples para representar os padrões presentes nos dados.

CAUSAS DE UNDERFITTING

Modelo Muito Simples: Utilizar uma arquitetura de rede neural com poucas camadas ou neurônios, que não possui capacidade suficiente para aprender os padrões complexos dos dados.

Poucos Dados de Treinamento: Insuficiência de dados pode levar o modelo a não aprender as características necessárias para fazer boas previsões.

Alta Regularização: Aplicar regularização excessiva (como dropout, L1, L2) pode forçar o modelo a ser muito simples.

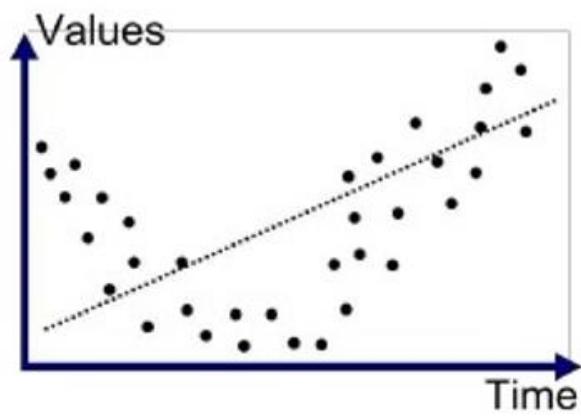
Parâmetros de Treinamento Mal Escolhidos: Taxa de aprendizado inadequada, número insuficiente de épocas de treinamento, ou má escolha de funções de ativação.

UNDERFITTING

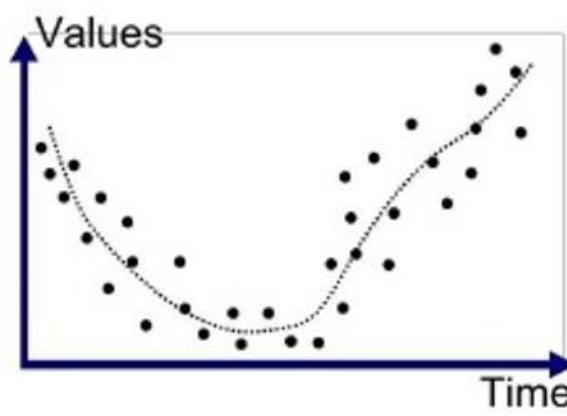
SINAIS DE UNDERFITTING

Baixa Precisão nos Dados de Treinamento: O modelo apresenta um erro significativo nos dados de treinamento.

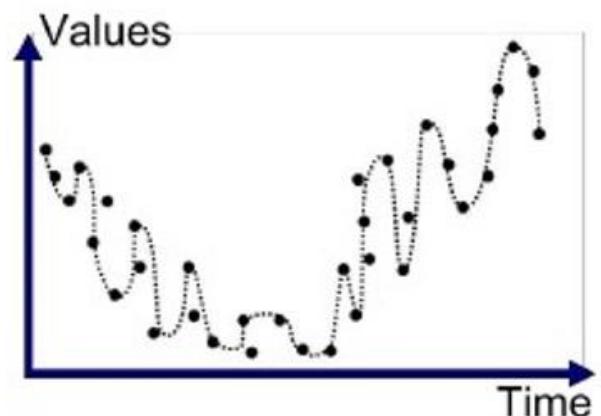
Desempenho Similar nos Dados de Treinamento e Teste: Ambos apresentam alta taxa de erro, indicando que o modelo não está aprendendo adequadamente.



Underfitted



Good Fit/R robust



Overfitted

UNDERFITTING

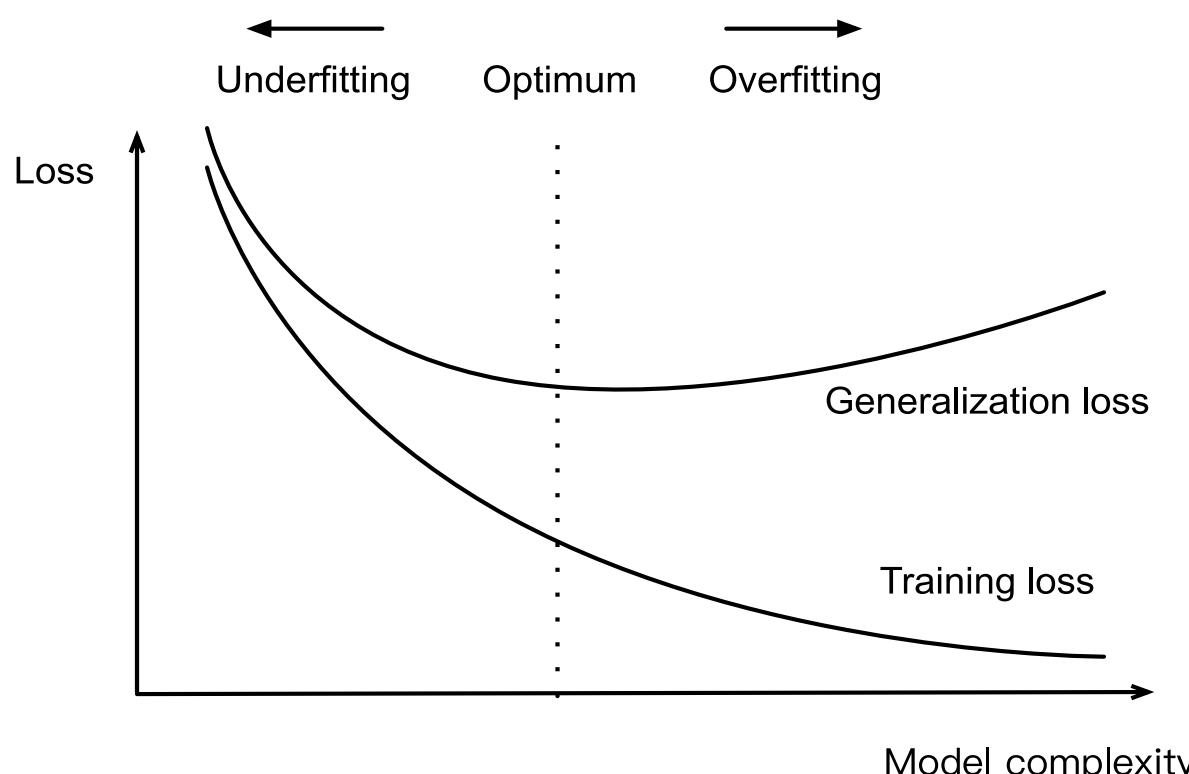
MITIGANDO UNDERFITTING

Aumento a Complexidade do Modelo: Adicionar mais camadas e neurônios à rede neural pode aumentar a capacidade do modelo de capturar padrões complexos.

Coleta de Mais Dados de Treinamento: Ter mais exemplos pode ajudar o modelo a aprender melhor.

Redução a Regularização: Diminuir a intensidade das técnicas de regularização pode permitir que o modelo se ajuste melhor aos dados de treinamento.

Ajuste dos Parâmetros de Treinamento: Utilizar uma taxa de aprendizado adequada e aumentar o número de épocas de treinamento para garantir que o modelo tenha tempo suficiente para aprender.

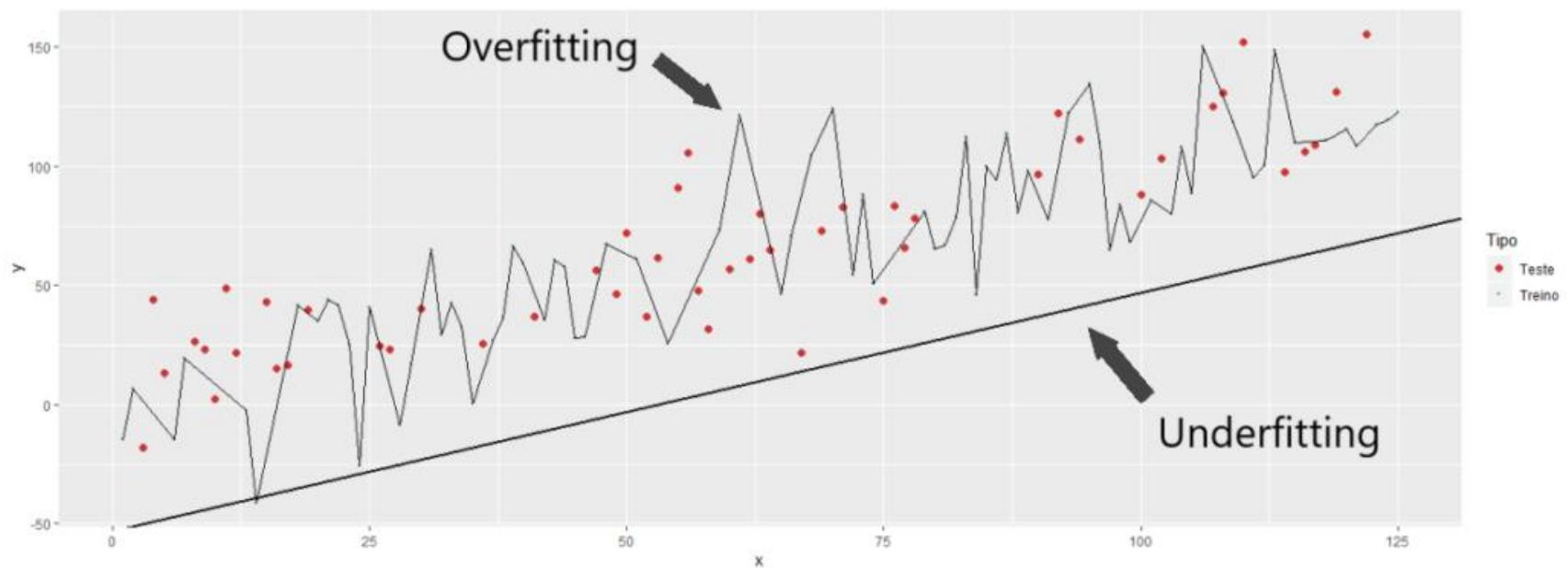


UNDERFITTING

EXEMPLOS

Regressão Linear Simples em Dados Não Lineares: Quando utilizamos uma regressão linear para um problema onde a relação entre as variáveis é não linear, o modelo pode apresentar underfitting.

Redes Neurais com Poucas Camadas: Uma rede neural com apenas uma camada oculta e poucos neurônios pode não ser capaz de aprender os padrões complexos presentes em um conjunto de dados rico.



Em resumo, underfitting é um problema onde o modelo de rede neural é incapaz de aprender adequadamente a partir dos dados, geralmente devido à sua simplicidade excessiva ou a parâmetros de treinamento inadequados. A mitigação do underfitting envolve aumentar a capacidade do modelo e ajustar os parâmetros de treinamento para melhor capturar os padrões dos dados.

04

APLICAÇÕES DAS REDES NEURAIS

Neste capítulo, exploraremos algumas aplicações das redes neurais em diferentes domínios, como visão computacional, processamento de linguagem natural e jogos. Analisaremos exemplos de modelos amplamente utilizados e seus impactos.

APLICAÇÃO DAS REDES NEURAIS EM DIFERENTES DOMÍNIOS

VISÃO COMPUTACIONAL

É uma área onde as redes neurais têm mostrado resultados impressionantes, impulsionando o avanço de várias tecnologias. Aqui estão algumas das principais aplicações:

Reconhecimento de Imagens: Redes neurais convolucionais (CNNs) são amplamente utilizadas para o reconhecimento de imagens. Elas são capazes de identificar e classificar objetos em imagens com alta precisão. Isso é aplicado em:

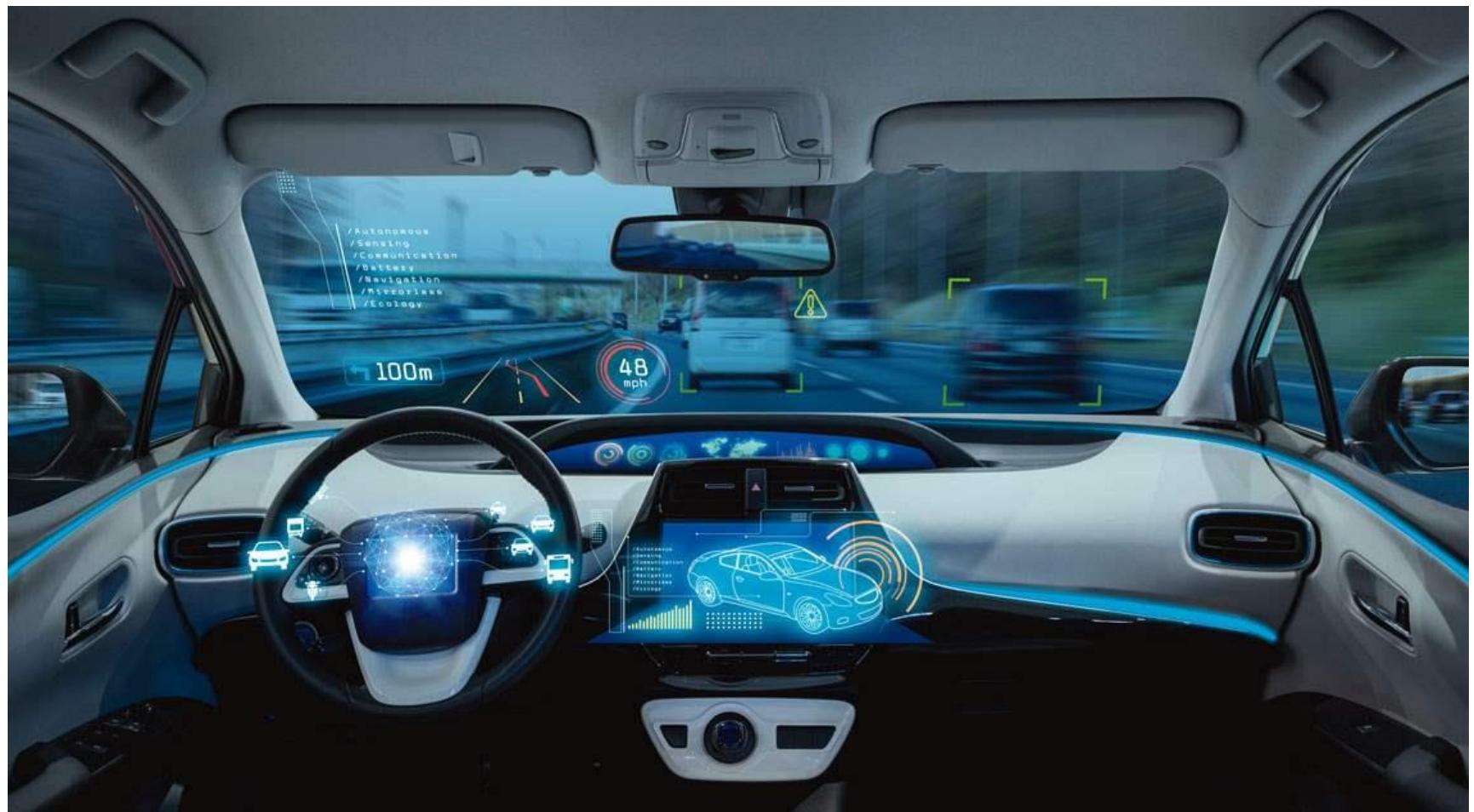
- **Sistemas de segurança:** reconhecimento facial para controle de acesso.
- **Redes sociais:** identificação de pessoas em fotos.
- **Saúde:** diagnóstico de doenças a partir de imagens médicas, como radiografias e ressonâncias magnéticas.



DETECÇÃO DE OBJETOS

A detecção de objetos vai além do reconhecimento de imagens, identificando e localizando objetos específicos dentro de uma imagem. Aplicações incluem:

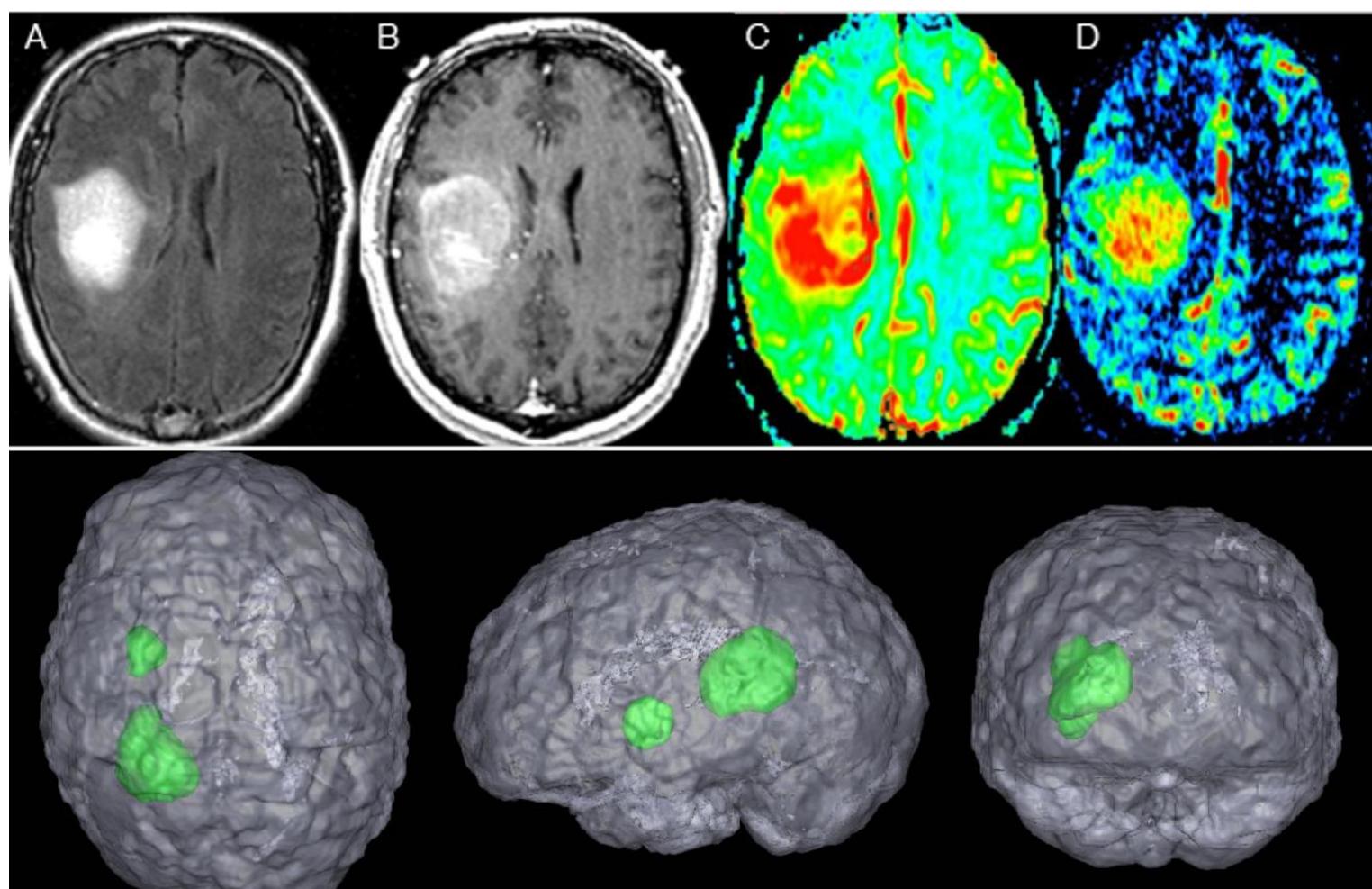
- **Carros autônomos:** detecção de pedestres, sinais de trânsito e outros veículos.
- **Varejo:** monitoramento de prateleiras para controle de estoque.
- **Agricultura:** monitoramento de culturas e detecção de pragas.



SEGMENTAÇÃO DE IMAGENS

A segmentação de imagens envolve a divisão de uma imagem em várias partes ou segmentos, geralmente para facilitar a análise. Usos incluem:

- **Medicina:** segmentação de órgãos ou tumores em exames de imagem.
- **Indústria:** inspeção de qualidade em linhas de produção.
- **Pesquisa ambiental:** monitoramento de mudanças em paisagens naturais através de imagens de satélite.

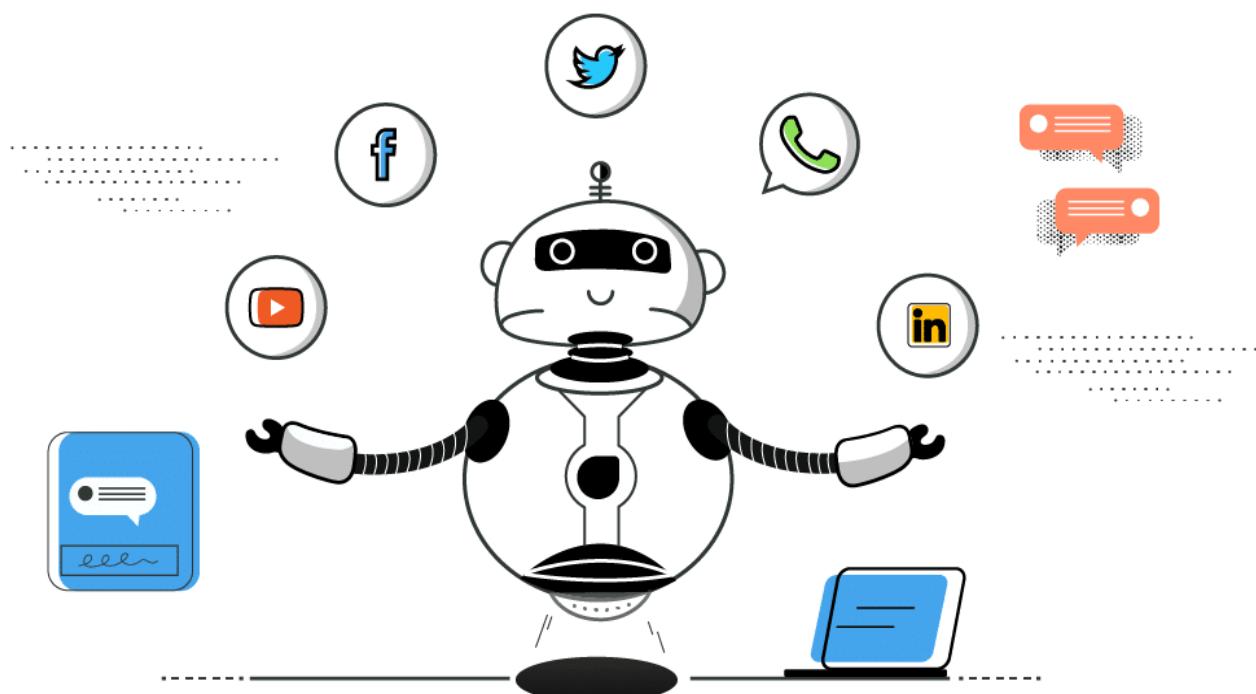


PROCESSAMENTO DE LINGUAGEM NATURAL (PLN)

No campo do processamento de linguagem natural, as redes neurais transformaram a forma como as máquinas entendem e interagem com a linguagem humana. Algumas aplicações notáveis são:

Tradução Automática: Modelos de redes neurais, como os Transformadores, são usados em serviços de tradução automática, permitindo a tradução precisa e fluida de texto entre diferentes idiomas. Exemplos incluem Google Translate e DeepL.

Chatbots e Assistentes Virtuais: Redes neurais alimentam chatbots e assistentes virtuais como Siri, Alexa e Google Assistant. Eles conseguem entender comandos de voz, responder perguntas e realizar tarefas simples, proporcionando uma interface natural entre humanos e máquinas.



Análise de Sentimento: Empresas usam redes neurais para analisar opiniões em redes sociais, avaliações de produtos e feedback de clientes, ajudando a compreender melhor a percepção pública e melhorar produtos e serviços.



Resumo Automático de Textos: Redes neurais são utilizadas para gerar resumos automáticos de longos documentos, artigos e notícias, facilitando a extração rápida de informações essenciais.



JOGOS



A inteligência artificial e as redes neurais têm desempenhado um papel crucial na evolução dos jogos, oferecendo experiências mais imersivas e desafiadoras. Algumas aplicações incluem:

Agentes Autônomos: Redes neurais são usadas para criar agentes autônomos que podem aprender e adaptar-se a diferentes cenários de jogo. Exemplos incluem:

- **Jogos de tabuleiro:** AlphaGo, da DeepMind, que venceu campeões humanos no jogo de Go.
- **Jogos de estratégia:** IA em jogos como StarCraft II, onde a IA pode aprender táticas complexas para vencer oponentes humanos.



JOGOS



Personalização da Experiência do Jogador

Alguns jogos usam redes neurais para adaptar a dificuldade e os desafios com base no desempenho e nas preferências do jogador, proporcionando uma experiência personalizada e mais envolvente.

Criação Procedural de Conteúdo

As redes neurais ajudam na criação procedural de conteúdo, gerando novos níveis, mapas e personagens de forma autônoma, aumentando a rejogabilidade e a diversidade dos jogos.



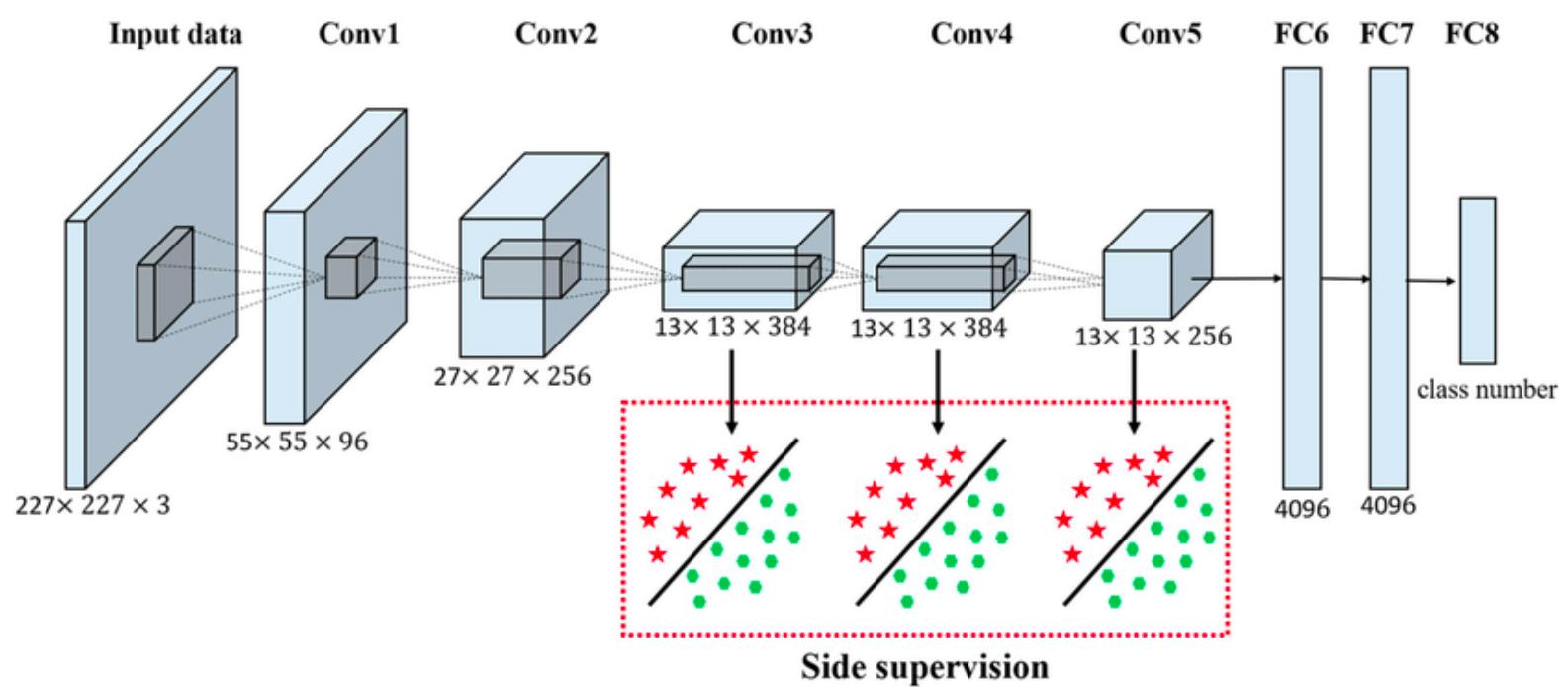
EXEMPLOS DE MODELOS DE REDES NEURAIS E SEUS IMPACTOS

Redes Neurais Convolucionais (CNNs)

Modelo: **AlexNet**

Impacto:

- **Vencedor da competição ImageNet 2012:** AlexNet revolucionou o campo da visão computacional ao vencer a competição com uma margem significativa, demonstrando o poder das redes neurais profundas.
- **Aplicações amplas:** Usado em sistemas de segurança para reconhecimento facial, diagnósticos médicos em imagens radiológicas, e sistemas de condução autônoma para detecção de objetos.



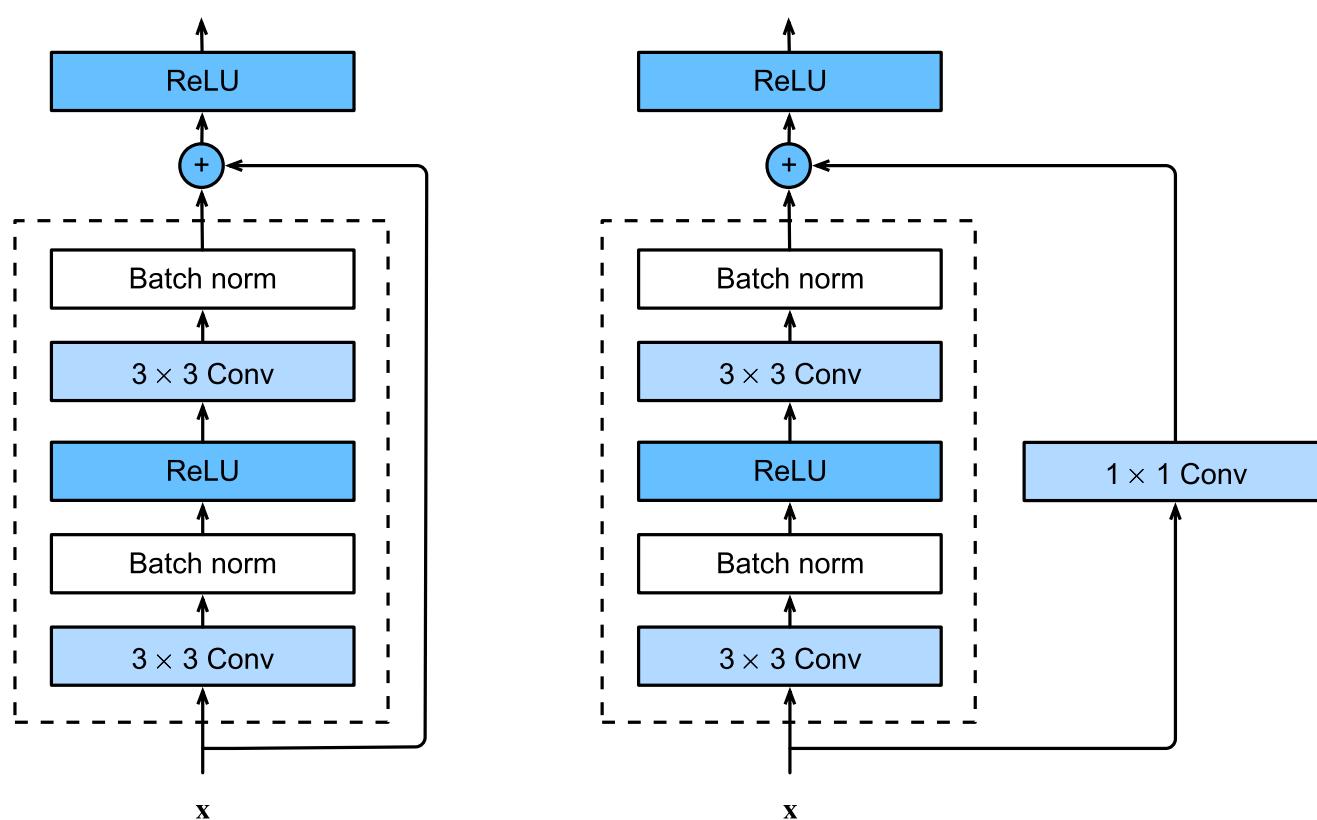
EXEMPLOS DE MODELOS DE REDES NEURAIS E SEUS IMPACTOS

Redes Neurais Convolucionais (CNNs)

Modelo: [ResNet \(Residual Networks\)](#)

Impacto:

- **Resolução de problemas de treinamento profundo:** Com sua estrutura inovadora de blocos residuais, ResNet permitiu a construção de redes muito profundas (até 152 camadas) sem sofrer de problemas de degradação.
- **Aplicações em múltiplos domínios:** Utilizado em sistemas de detecção de anomalias industriais, análise de imagem médica, e reconhecimento de voz.



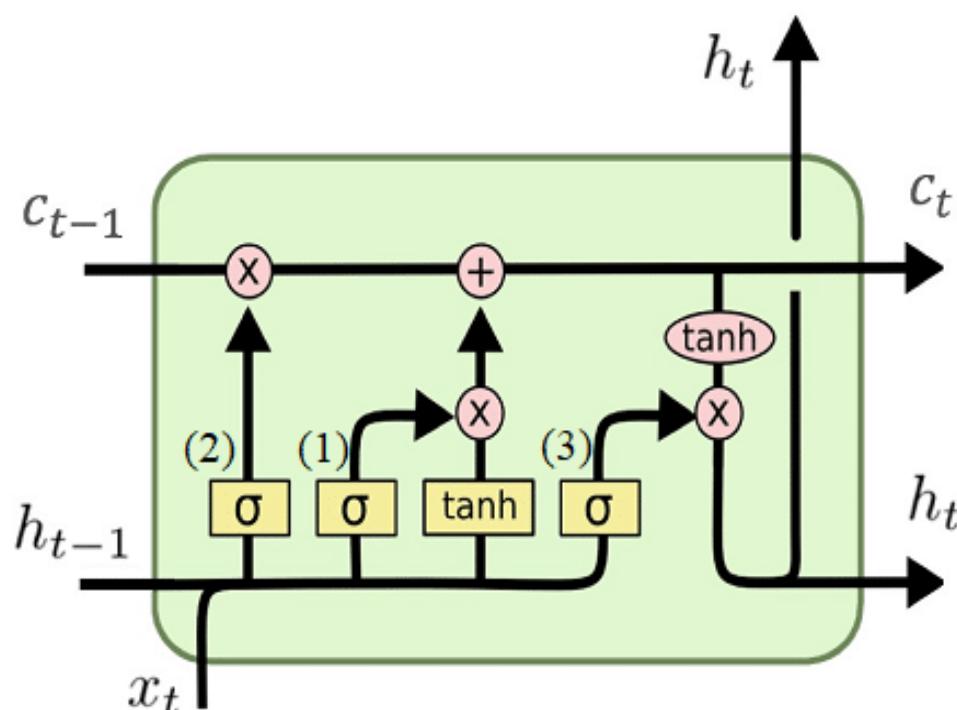
EXEMPLOS DE MODELOS DE REDES NEURAIS E SEUS IMPACTOS

Redes Neurais Recorrentes (RNNs) e Long Short-Term Memory (LSTM)

Modelo: **LSTM**

Impacto:

- **Melhoria no processamento de sequências:** LSTMs resolveram o problema de longo prazo das dependências em RNNs tradicionais, tornando-as eficazes para tarefas de sequência longa.
- **Aplicações em PLN:** Utilizadas em modelos de tradução automática, como o Google Translate, reconhecimento de fala e sistemas de previsão de séries temporais financeiras.



LSTM
(Long-Short Term Memory)

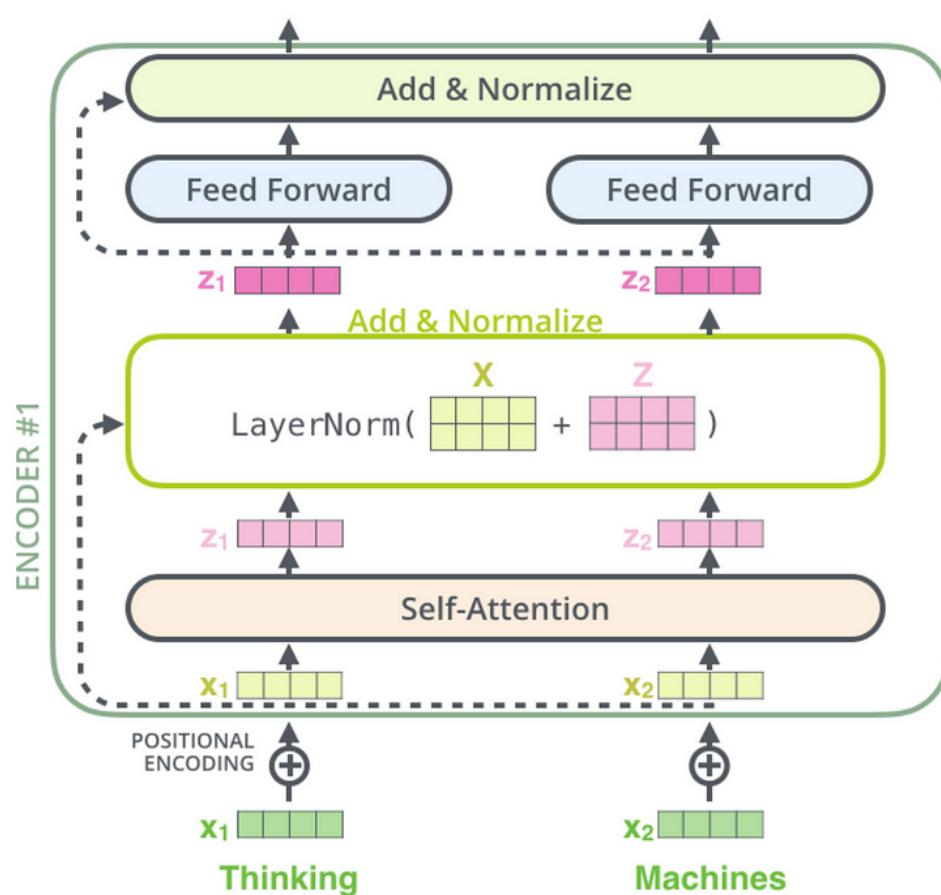
EXEMPLOS DE MODELOS DE REDES NEURAIS E SEUS IMPACTOS

Transformadores

Modelo: **BERT (Bidirectional Encoder Representations from Transformers)**

Impacto:

- **Mudança de paradigma no PLN:** BERT introduziu uma abordagem bidirecional para modelar contextos de palavras, melhorando significativamente o desempenho em várias tarefas de PLN.
- **Amplamente adotado:** Utilizado em motores de busca para compreensão de consultas, chatbots avançados e assistentes virtuais, e análise de sentimentos.



EXEMPLOS DE MODELOS DE REDES NEURAIS E SEUS IMPACTOS

Transformadores

Modelo: GPT-3 (Generative Pre-trained Transformer 3)

Impacto:

- **Capacidades de geração de texto:** GPT-3, com 175 bilhões de parâmetros, é um dos maiores e mais poderosos modelos de linguagem, capaz de gerar texto altamente coerente e humano.
- **Aplicações diversificadas:** Utilizado em escrita criativa, geração de código, respostas automáticas em serviços ao cliente e até na criação de conteúdos personalizados para marketing.

ChatGPT		
Examples	Capabilities	Limitations
"Explain quantum computing in simple terms" →	Remembers what user said earlier in the conversation	May occasionally generate incorrect information
"Got any creative ideas for a 10 year old's birthday?" →	Allows user to provide follow-up corrections	May occasionally produce harmful instructions or biased content
"How do I make an HTTP request in Javascript?" →	Trained to decline inappropriate requests	Limited knowledge of world and events after 2021

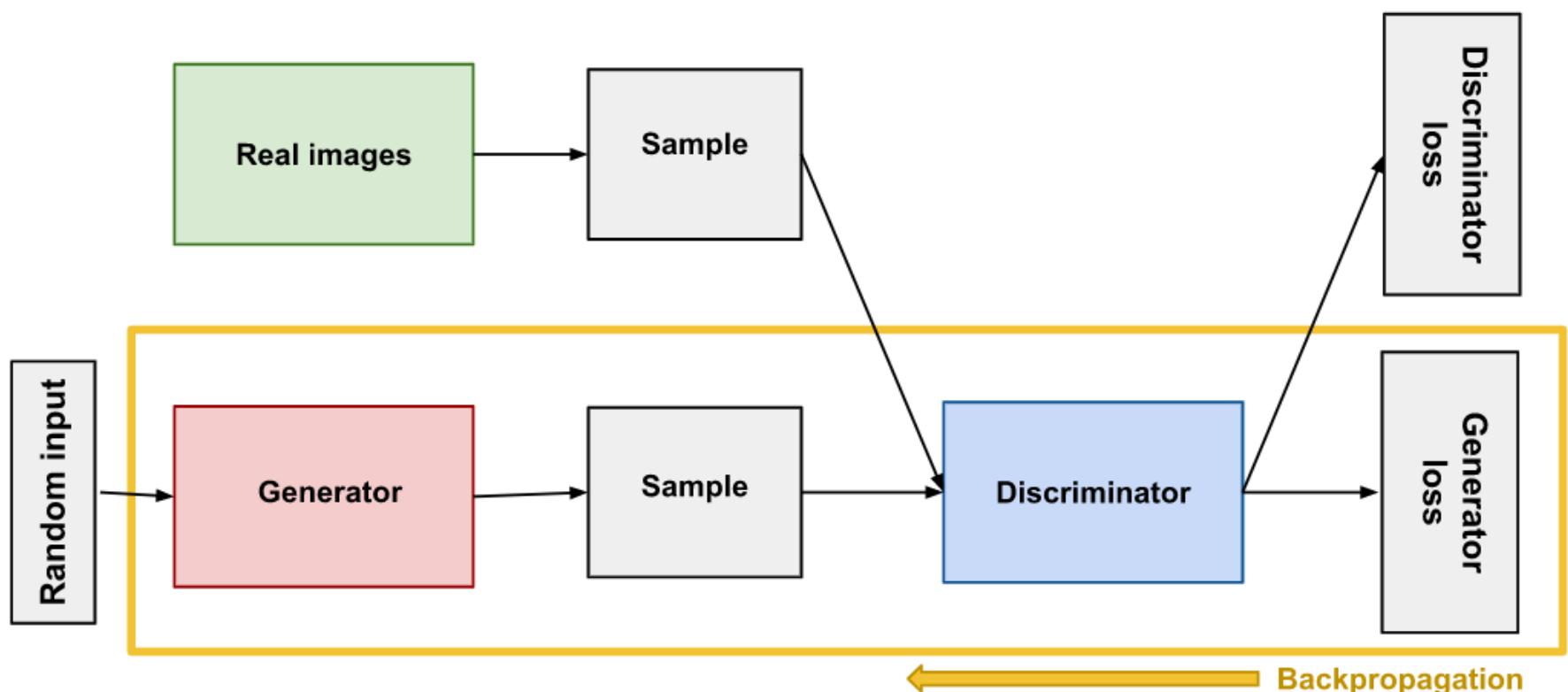
EXEMPLOS DE MODELOS DE REDES NEURAIS E SEUS IMPACTOS

Redes Adversariais Generativas (GANs)

Modelo: **DCGAN (Deep Convolutional GAN)**

Impacto:

- **Geração de imagens realistas:** DCGANs são conhecidas por gerar imagens realistas a partir de ruído, impactando áreas como criação de arte digital, design de moda e geração de conteúdo para jogos.
- **Aplicações em aprimoramento de imagem:** Utilizadas para aumento de resolução de imagens (super-resolução), restauração de imagens danificadas e geração de novos dados sintéticos para treinamento de modelos.



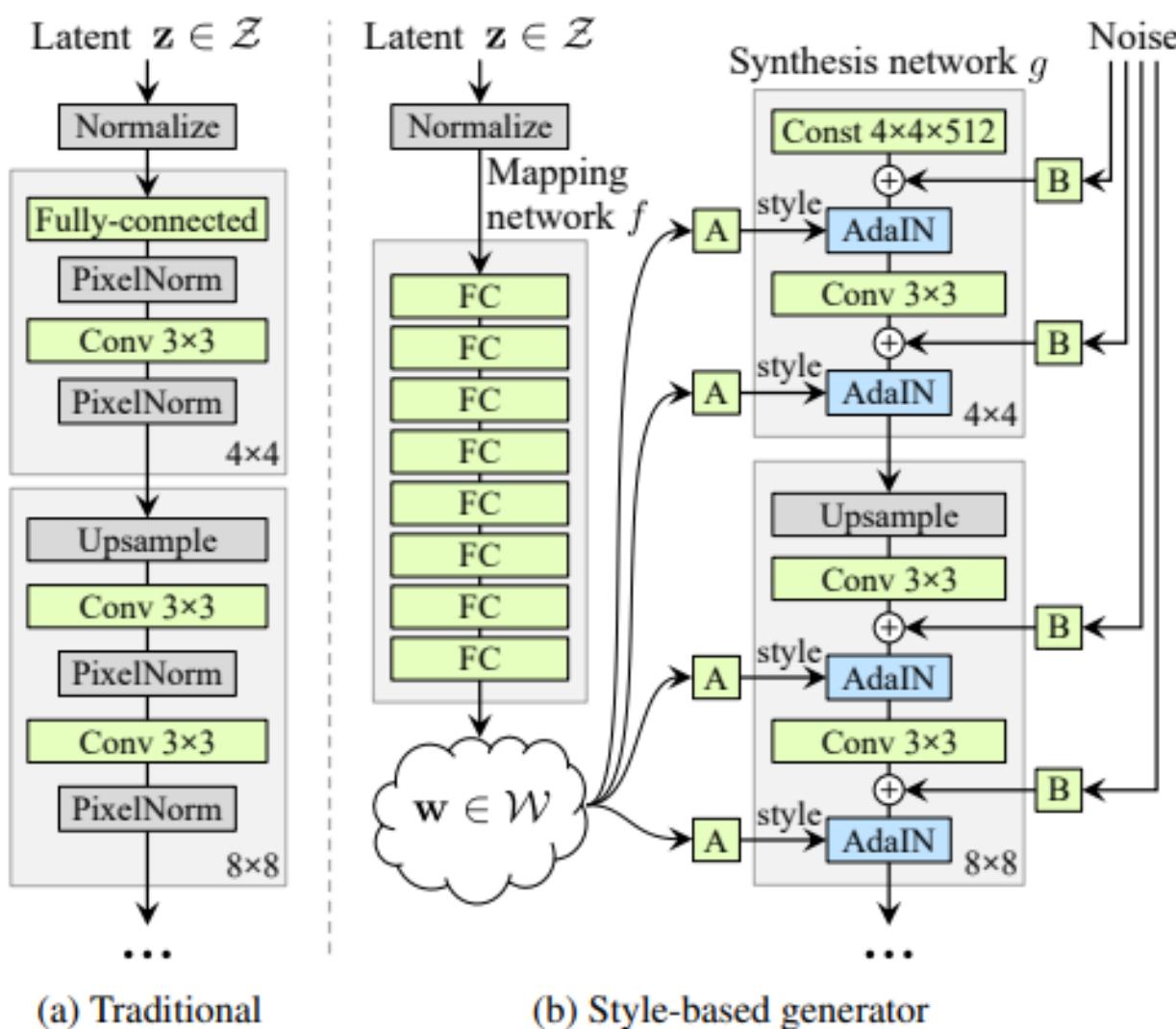
EXEMPLOS DE MODELOS DE REDES NEURAIS E SEUS IMPACTOS

Redes Adversariais Generativas (GANs)

Modelo: **StyleGAN**

Impacto:

- **Geração de rostos humanos realistas:** StyleGAN é famoso por criar imagens de rostos humanos que são indistinguíveis de fotos reais, revolucionando a criação de avatares virtuais e personagens de jogos.
- **Aplicações em design e entretenimento:** Utilizado em produção cinematográfica para criar personagens virtuais e em aplicativos de personalização de avatares.



05

DA TEORIA À PRÁTICA: CONSTRUINDO SUA PRIMEIRA REDE NEURAL

Por fim, vamos passar da teoria à prática, guiando os desenvolvedores juniores na construção de sua primeira rede neural.

Mostraremos como começar com modelos simples, proporcionando uma base sólida para que você possa, gradualmente, enfrentar desafios mais complexos.

CONSTRUINDO SUA PRIMEIRA REDE NEURAL

Agora vamos construir uma rede neural simples usando **Python** e o popular framework **TensorFlow** com sua API de alto nível **Keras**. Vamos criar um modelo para classificar dígitos escritos à mão usando o dataset **MNIST**.

Passo a Passo para Construir uma Rede Neural Simples

1. Instalação dos Pacotes Necessários: Primeiro, você precisa instalar o TensorFlow. Você pode fazer isso via pip:

```
– □ ×  
pip install tensorflow  
snappify.com
```

2. Importar as Bibliotecas: Vamos começar importando as bibliotecas necessárias:

```
– □ ×  
  
import tensorflow as tf  
from tensorflow.keras import layers, models  
import numpy as np  
import matplotlib.pyplot as plt
```

CONSTRUINDO SUA PRIMEIRA REDE NEURAL

3. Carregar e Preparar o Dataset: O dataset MNIST está disponível diretamente no TensorFlow, então podemos carregá-lo facilmente:

```
- □ ×  
  
# Carregar o dataset MNIST  
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()  
  
# Normalizar os dados para que os valores estejam entre 0 e 1  
x_train, x_test = x_train / 255.0, x_test / 255.0
```

4. Construir o Modelo da Rede Neural: Vamos construir uma rede neural com uma camada de entrada (flatten), uma camada oculta densa com 128 neurônios e uma camada de saída com 10 neurônios (uma para cada dígito de 0 a 9):

```
- □ ×  
  
model = models.Sequential([  
    layers.Flatten(input_shape=(28, 28)), # Camada de entrada  
    layers.Dense(128, activation='relu'), # Camada oculta com 128 neurônios e função de ativação ReLU  
    layers.Dense(10, activation='softmax') # Camada de saída com 10 neurônios e função de ativação Softmax  
])
```

CONSTRUINDO SUA PRIMEIRA REDE NEURAL

5. Compilar o Modelo: Compilamos o modelo especificando o otimizador, a função de perda e as métricas de avaliação:

```
- □ ×  
  
model.compile(optimizer='adam',  
               loss='sparse_categorical_crossentropy',  
               metrics=['accuracy'])
```

6. Treinar o Modelo: Agora, treinamos o modelo com os dados de treinamento:

```
- □ ×  
  
model.fit(x_train, y_train, epochs=5)
```

7. Avaliar o Modelo: Finalmente, avaliamos o modelo usando os dados de teste para ver a precisão:

```
- □ ×  
  
test_loss, test_acc = model.evaluate(x_test, y_test)  
print(f'\nTest accuracy: {test_acc}')
```

CONSTRUINDO SUA PRIMEIRA REDE NEURAL

Código Completo

Aqui está o código completo, pronto para ser executado:

```
- □ ×

import tensorflow as tf
from tensorflow.keras import layers, models

# Carregar o dataset MNIST
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Normalizar os dados para que os valores estejam entre 0 e 1
x_train, x_test = x_train / 255.0, x_test / 255.0

# Construir o modelo da rede neural
model = models.Sequential([
    layers.Flatten(input_shape=(28, 28)), # Camada de entrada
    layers.Dense(128, activation='relu'), # Camada oculta com 128 neurônios e função de ativação ReLU
    layers.Dense(10, activation='softmax') # Camada de saída com 10 neurônios e função de ativação Softmax
])

# Compilar o modelo
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Treinar o modelo
model.fit(x_train, y_train, epochs=5)

# Avaliar o modelo
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'\nTest accuracy: {test_acc}')
```

Este é um exemplo básico, mas funcional, de como construir e treinar uma rede neural simples com Python e TensorFlow.

CONSTRUINDO SUA PRIMEIRA REDE NEURAL

Explicação Resumida

- **Importação de bibliotecas:** Importamos TensorFlow e outras bibliotecas necessárias.
- **Carregamento e preparação dos dados:** Carregamos o dataset MNIST e normalizamos os valores das imagens para o intervalo de 0 a 1.
- **Construção do modelo:** Usamos a API Keras para definir um modelo sequencial com uma camada de entrada, uma camada oculta e uma camada de saída.
- **Compilação do modelo:** Especificamos o otimizador, a função de perda e as métricas.
- **Treinamento do modelo:** Treinamos o modelo nos dados de treinamento.
- **Avaliação do modelo:** Avaliamos a precisão do modelo nos dados de teste.

AGRADECIMENTOS

OBRIGADA POR LER ATÉ AQUI!

Este Ebook foi criado com conteúdo gerado por IA, complementado com pesquisas acadêmicas e Diagramado por Humano.

O Passo a Passo se encontra no meu GitHub.

Este Ebook foi criado com fins didáticos de construção, foram realizadas revisões básicas por humano (Dev Jr), mas pode conter erros de validação gerados por IA.



<https://github.com/SabrinaAll>



Autora



Gabriela Silva
[GitHub](#) | [LinkedIn](#)