Aula 1: O Neurônio Artificial

Aléssio Miranda Júnior

Agosto - 2026/2

O que você verá neste capítulo

Este capítulo inaugura a disciplina de Inteligência Artificial II. Partindo de uma reflexão sobre o que foi visto em IA I (Lógica Fuzzy, Algoritmos Genéticos), construímos a intuição do neurônio artificial através de metáforas humanas e analogias cotidianas, formalizamos a álgebra do Perceptron com produto escalar vetorizado, e culminamos na expansão para Redes Neurais Multicamadas (MLP) — a arquitetura que fundamenta toda a revolução do *Deep Learning*.

1 Introdução: Do Espelho Humano ao Átomo da IA

1.1 De IA I para IA II: Onde Está a Aprendizagem?

Antes de mergulharmos na matemática das redes neurais, é fundamental olharmos pelo retrovisor e refletirmos sobre o que foi construído em Inteligência Artificial I. Lá, o percurso passou por duas vertentes poderosas:

- **Lógica Fuzzy:** Vocês aprenderam a traduzir conceitos subjetivos da linguagem humana (“o ambiente está muito quente”, “o carro está rápido”) para curvas de pertinência matemáticas. Regras SE-ENTÃO nebulosas permitiam tomar decisões sem fronteiras rígidas, com graus de pertinência μ variando continuamente entre 0 e 1. Mas faça a si mesmo a pergunta: quem teve que sentar e desenhar as curvas de temperatura? Quem definiu se a regra disparava o ventilador? **Foi você, o engenheiro**.
- **Algoritmos Genéticos:** Uma abordagem belíssima inspirada na biologia evolutiva. Vocês criaram populações de soluções candidatas, aplicaram operadores de seleção, *crossover* e mutação. As soluções “evoluíam” ao longo de gerações, otimizando sem gradiente. Mas quem construiu a Função de Aptidão (*Fitness*) que dizia quem sobrevivia e quem morria? **Novamente, você**.

- **Perceptron (Introdução):** No final da disciplina, vocês viram brevemente o conceito de um neurônio como tomador de decisão — com pesos, entradas e um limiar de ativação. A ideia de separação linear simples foi plantada como semente.

Fica então a provocação central: em que momento, de fato, ocorreu *aprendizagem* por parte da máquina? Na Lógica Fuzzy, a máquina executou regras pré-programadas por um especialista humano; nos Algoritmos Genéticos, ela realizou uma busca estocástica otimizada no espaço de soluções, guiada por uma bússola (a função fitness) inteiramente projetada pelo programador. Em IA 1, **o programador dita a lógica** — direta ou indiretamente.

△ Importante: A Ponte para IA II

A transição para **IA 2** (centrada em *Machine Learning* e *Deep Learning*) marca uma mudança de paradigma. A partir de agora, **a máquina irá extrair a própria lógica** a partir dos dados brutos. O seu papel como programador não será mais ditar regras elaboradas, mas sim **preparar uma arquitetura matemática e um ambiente** para que o modelo construa sua própria representação interna do mundo. O Perceptron que vocês viram brevemente será o nosso ponto de partida.

O primeiro choque de realidade é que a "mágica" da Inteligência Artificial resume-se pura e simplesmente à **matemática vetorial**, **multiplicação de matrizes** e **busca sistemática e iterativa por padrões geométricos**. Não há consciência, não há compreensão mística. Para criarmos arquiteturas gigantescas que conversam fluentemente (como os Transformers), precisamos antes dominar o alicerce absoluto: **o Neurônio Artificial**.

1.2 IA no Espelho: Metáforas Humanas

◇ Informação

IA no Espelho
No dia a dia, usamos termos como "a IA alucina", "a IA aprende", "a IA decide". Essas expressões são úteis para comunicar, mas são **metáforas**. Curiosamente, humanos também alucinam (memórias falsas, pareidolia, sonhos), também aprendem por erro e repetição (queimar a mão no fogão), também são enviesados (viés de confirmação) e também decidem por critérios ponderados (ir ou não ao festival). Ao longo desta disciplina, vamos desconstruir cada uma dessas metáforas e substituí-las pela mecânica matemática real que as produz. Quando você entender que "alucinar" é interpolar fora da distribuição de treino, a metáfora se torna uma ferramenta de diagnóstico — não de antropomorfização.

1.3 Criatividade: Juntar o Improvável

O que é criatividade? É a arte de combinar **conceitos distantes e improváveis** numa configuração nova. Se todos pensassem de maneira idêntica, não haveria criatividade — apenas repetição. Um músico que mistura samba com eletrônica, um chef que combina chocolate com pimenta, um cientista que aplica teoria dos grafos à biologia: todos estão fazendo a mesma operação mental de *aproximar o que parecia distante*.

Uma IA generativa faz algo mecanicamente similar. Ela aprendeu milhões de exemplos e os codificou como pontos num espaço vetorial de alta dimensão (as chamadas **representações latentes**). Ao gerar algo "novo", ela está combinando regiões distantes desse espaço — produzindo uma interpolação que *parece* original. A diferença fundamental é que a IA não *sabe* que está sendo criativa: ela apenas calcula a próxima combinação linear mais provável. Nós, humanos, atribuímos significado ao resultado. Ao longo do Arco 3 (IA Generativa), veremos exatamente como essa mecânica vetorial funciona.

1.4 Alucinar na Prova: Uma Metáfora Reveladora

Pense na seguinte situação universal: você está fazendo uma prova. Tem uma **base de dados** (o que você estudou), uma **obrigatoriedade de resposta** (a prova exige que você preencha algo) e um **tempo limitado** (o relógio está correndo). Quando o tempo se esgota, o que acontece? Você *chuta*. Dá uma resposta porque **precisa** responder, mesmo sem certeza absoluta. E muitas vezes, essa resposta sai com uma convicção surpreendente — mesmo estando completamente errada.

Essa é **exatamente** a mecânica de um Large Language Model (LLM). A cada instante, o modelo é **forçado** a gerar o próximo token (a próxima palavra). Ele não possui a opção mecânica de dizer "não sei" — o algoritmo *obriga* uma saída. Quando os dados de treino não cobrem adequadamente o caso atual, o modelo "chuta" com alta confiança estatística. Isso é o que chamamos de **alucinação**: não é um defeito de caráter da máquina, é uma consequência direta da arquitetura de geração autoregressiva. Assim como o aluno na prova, a IA está num cenário de *resposta obrigatória com informação incompleta*.

2 O Perceptron: Da Intuição Cotidiana ao Modelo Matemático

Antes de formalizarmos a matemática matricial, é fundamental entender o Perceptron de maneira intuitiva. Por trás das equações, o Perceptron nada mais é do que um **mecanismo automático de tomada de decisão baseada em critérios ponderados**.

2.1 Uma Analogia Prática: A Decisão do Fim de Semana

Imagine que você precisa decidir se vai a um **festival de música no fim de semana**. Para tomar essa decisão, seu cérebro avalia intuitivamente diversos fatores (as **entradas** ou atributos x_i):

- x_1: O ingresso está barato? ($1 = \text{Sim}, 0 = \text{Não}$)
- x_2: Seus melhores amigos vão? ($1 = \text{Sim}, 0 = \text{Não}$)
- x_3: A previsão do tempo indica sol? ($1 = \text{Sim}, 0 = \text{Não}$)

Porém, nem todos os fatores possuem a mesma importância para você. Se a companhia dos amigos for indispensável, mas a previsão do tempo pouco importar, você atribuirá **pesos** (w_i) diferentes para cada entrada:

- $w_1 = 2$ (Peso moderado para o preço)
- $w_2 = 6$ (Peso alto para a presença dos amigos)
- $w_3 = 1$ (Peso baixo para o tempo)

Além disso, cada pessoa possui uma **inclinação natural** (um perfil prévio): se você for uma pessoa extremamente festeira, terá uma tendência inicial positiva para ir ($b > 0$), mesmo se as condições não forem ideais. Se for caseira, precisará de fortes incentivos para sair de casa ($b < 0$). Essa tendência inicial é o **Viés (Bias, b)**.

Perceba: esse cálculo é exatamente o que você faz intuitivamente todos os dias. A diferença é que o seu cérebro biológico opera com milhares de fatores simultâneos, ajustados por décadas de experiência. O Perceptron faz a mesma conta — só que com números explícitos, em escala industrial, e a uma velocidade sobre-humana.

O seu cérebro calcula o valor combinado:

$$z = (x_1 \cdot w_1) + (x_2 \cdot w_2) + (x_3 \cdot w_3) + b$$

Se essa soma z ultrapassar o seu limiar interno (for maior que 0), a sua decisão é **Ir ao evento ($y = 1$)**; caso contrário, a decisão é **Ficar em casa ($y = 0$)**. O Perceptron artificial executa exatamente esse cálculo!

2.2 Do Organismo Biológico à Abstração Computacional

Historicamente, essa estrutura também foi inspirada pela morfologia do cérebro humano. No córtex cerebral, um **neurônio biológico** possui ramificações chamadas de dendritos que recebem milhares de impulsos elétricos de neurônios vizinhos. O corpo celular (núcleo) acumula, filtra e processa essa carga elétrica e, ao atingir um limiar de disparo, transmite um sinal via axônio para a rede sináptica adjacente.

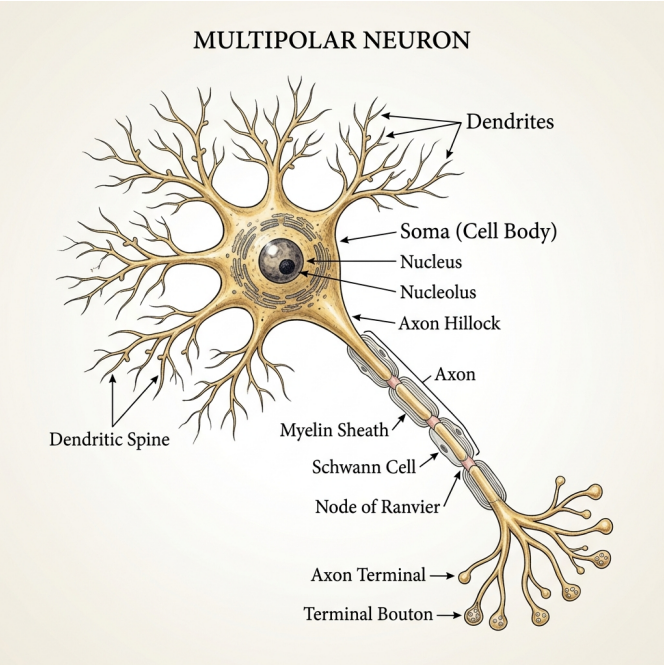


Figura 1: Diagrama esquemático de um neurônio biológico, mostrando dendritos (receptores), corpo celular (processamento) e axônio (transmissão). A célula nervosa serviu como a primeira grande inspiração morfológica para o modelo computacional.

Em 1957, Frank Rosenblatt formalizou o **Perceptron**, traduzindo a biologia para quatro partes mecânicas primárias. A tabela a seguir mostra o paralelo direto entre a célula nervosa humana e a máquina de Rosenblatt:

Célula Nervosa Humana	**Máquina de Rosenblatt**
Dendritos: Receptores de sinais externos	**Entradas (X):** Valores x_i do mundo real
Sinapse: Força bioquímica de ligação	**Pesos (W):** Conexões ajustáveis (relevância)
Corpo Celular: Acumula a carga iônica	**Soma Ponderada (Σ):** Acumulador algébrico (z)
Axônio: Efetua o disparo elétrico	**Ativação:** Disparo analítico via $f(z)$

Tabela 1: Paralelo morfológico entre o neurônio biológico e o Perceptron computacional.

2.3 Modelagem Algébrica e Diagrama Vetorial

A equação mecânica basilar de qualquer sistema de Machine Learning é dada pela acumulação linear do Produto Escalar (*Dot Product*):

$$z = (X \cdot W) + b = \left(\sum_{i=1}^{n} x_i \cdot w_i \right) + b \tag{1}$$

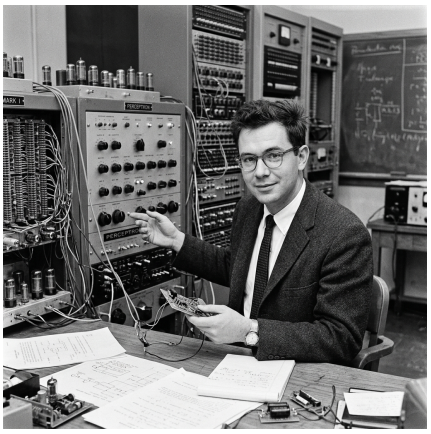


Figura 2: Frank Rosenblatt, o inventor do Perceptron, no laboratório de IA da Universidade Cornell (1957).

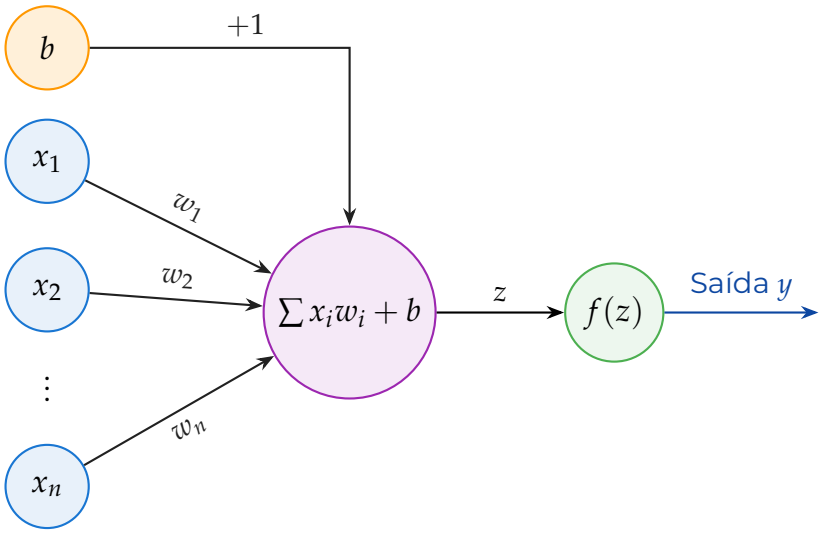


Figura 3: Diagrama esquemático estrutural de um Perceptron moderno com entradas (x_i), pesos (w_i), viés (b), soma ponderada ($\sum$) e ativação ($f(z)$).

Após obter z, o resultado é lançado através de uma **Função de Ativação** não linear, $f(z)$, como a Sigmoid ou ReLU, que ditará a amplitude do "disparo" do axônio y.

2.4 O Poder da Vetorização: NumPy vs. Laços Tradicionais

Se fôssemos traduzir a fórmula $z = \sum(x_i \cdot w_i) + b$ para código usando a mentalidade tradicional de um desenvolvedor júnior (por exemplo, em Java clássico, C# ou C puro sem otimização), escreveríamos um laço iterativo `for` encadeado:

```
double z = bias;
for (int i = 0; i < entradas.length; i++) {
    z += entradas[i] * pesos[i];
}
```

Listing 1: Implementação escalar clássica (Lenta para IA).

Na Inteligência Artificial moderna, operar **escalarmente** (um número por vez) no processador é catastrófico. Uma rede neural real processa milhões de parâmetros por segundo; laços sequenciais inviabilizariam o tempo de treinamento.

A modernidade exige o uso da **Álgebra Linear e Vetorização**. Em Python, utilizamos a biblioteca `NumPy`. Embora você escreva em Python, por baixo dos panos o `NumPy` delega a multiplicação de matrizes para bibliotecas de baixo nível hiper-otimizadas em C, C++ e Fortran (como a BLAS - *Basic Linear Algebra Subprograms*).

Mas o verdadeiro ganho físico de velocidade está nas instruções **SIMD (*Single Instruction, Multiple Data*)** do hardware. Quando usamos a operação vetorial matricial `np.dot(X, W)`, o processador não multiplica um par por vez; ele carrega um lote inteiro de números na memória do chip e executa dezenas de multiplicações simultaneamente num único ciclo de clock!

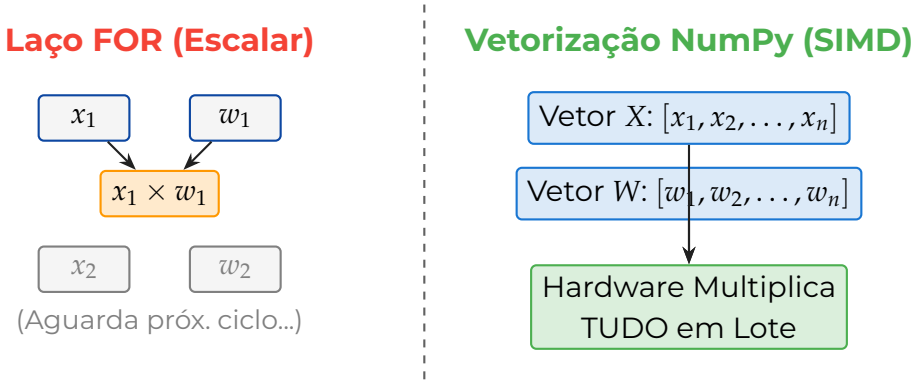


Figura 4: Comparação arquitetural: Laços iterativos sequenciais vs. Multiplicação matricial vetorizada na CPU via instruções SIMD.

2.5 A Regra de Ouro Geométrica: *Shapes*

Há, entretanto, uma regra de ouro que permeia toda a disciplina: **as dimensões (*Shapes*) dos tensores precisam convergir perfeitamente**. O maior causador de *crashes* em IA não é a lógica algorítmica, mas sim o colapso dimensional (*Broadcasting Errors*).

Ao executarmos o produto matricial de uma Matriz A de dimensão $(m \times n)$ por uma Matriz B de dimensão $(n \times p)$, obtemos uma Matriz C de dimensão $(m \times p)$. A dimensão interna (n) deve **obrigatoriamente coincidir**.

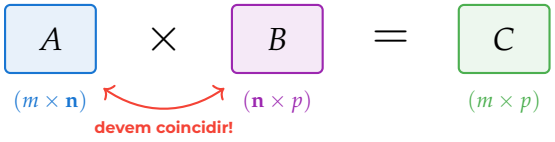


Figura 5: Regra de ouro da multiplicação matricial: a dimensão interna (n) de A e B deve ser idêntica. Controlar mentalmente os *shapes* dos tensores é a habilidade mais crítica para depurar erros em IA.

Controlar mentalmente as dimensões dos tensores é a habilidade mais crítica do Engenheiro de IA. Na prática, ao executarmos o produto de $X_{(1\times3)}$ por $W_{(3\times1)}$, obtemos por consequência um tensor escalar de formato (1×1) — exatamente o nosso z.

3 Redes Multicamadas: O Multilayer Perceptron (MLP)

Embora um único Perceptron seja capaz de tomar decisões ponderadas simples, ele possui uma limitação geométrica severa: **ele só consegue traçar uma linha reta (ou hiperplano) para separar dados**.

3.1 A Limitação da Separabilidade Linear (O Problema do XOR)

Em 1969, Marvin Minsky e Seymour Papert publicaram o livro *Perceptrons*, demonstrando que um único Perceptron é incapaz de aprender a simples função lógica **XOR (OU Exclusivo)**.

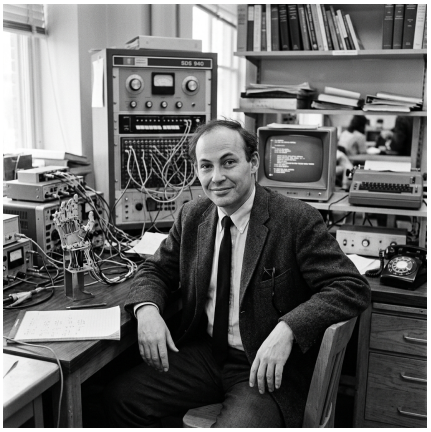


Figura 6: Marvin Minsky, coautor do livro *Perceptrons* (1969), cujo trabalho expôs as limitações de separabilidade linear de um neurônio isolado.

Em portas lógicas como **AND** ou **OR**, as saídas 0 e 1 podem ser separadas por uma única linha reta no plano cartesiano. Na porta **XOR**, entretanto, os pontos da mesma classe ficam em diagonais opostas, tornando impossível separá-los com uma reta:

Essa constatação histórica provou que um neurônio isolado não é suficiente para encarar problemas reais e não lineares (como visão computacional ou processamento de linguagem). O impacto foi tão profundo que levou ao primeiro "Inverno da IA" — uma década de descrença e desinvestimento na área.

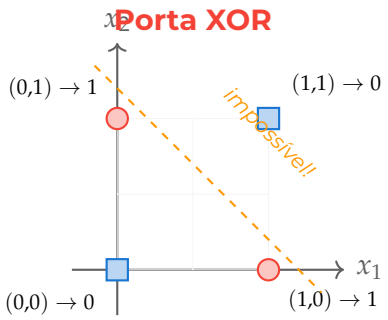


Figura 7: O problema do XOR: os pontos das classes 0 (■) e 1 (●) ficam em diagonais cruzadas. Nenhuma reta única é capaz de separá-los, demonstrando a limitação fatal do Perceptron isolado.

3.2 A Arquitetura Multicamadas

Para superar esse limite, conectamos múltiplos neurônios em uma rede hierárquica dividida em camadas, formando o **Multilayer Perceptron (MLP)**:

1. **Camada de Entrada (*Input Layer*)**: Recebe os dados brutos X do problema (características, pixels, atributos).
2. **Camada(s) Oculta(s) (*Hidden Layer*)**: Camadas intermediárias onde cada neurônio tira uma "perspectiva" diferente sobre os dados. Em conjunto, essas camadas "dobram" e deformam o espaço cartesiano, permitindo separar classes complexas — inclusive o XOR.
3. **Camada de Saída (*Output Layer*)**: Toma a **decisão final**, combinando as conclusões parciais das camadas ocultas para produzir a resposta da rede (classificação ou regressão).

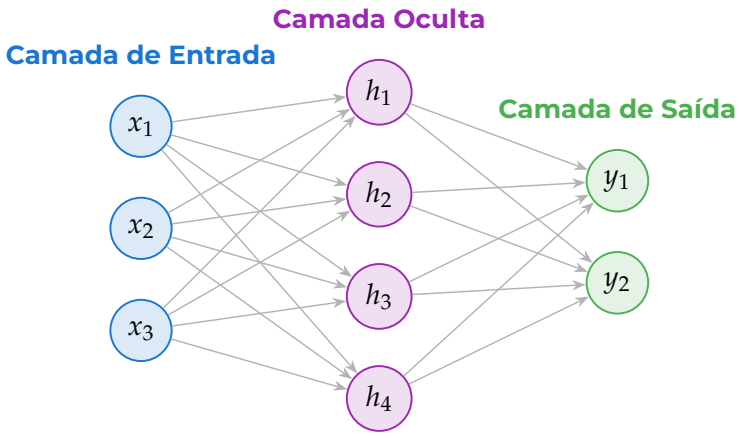


Figura 8: Estrutura esquemática de um Multilayer Perceptron (MLP) de 3 camadas. A informação se propaga sequencialmente da entrada para a saída (*Forward Pass*).

Ao encadearmos camadas com funções de ativação não lineares, o MLP adquire a incrível capacidade de aproximar qualquer função matemática contínua (conhecido como o **Teorema da Aproximação Universal**), fundamentando toda a revolução moderna de *Deep Learning*.

É precisamente essa capacidade de aproximar qualquer função que explica por que redes profundas exibem comportamentos que **parecem** inteligentes: fluência verbal, criatividade visual, raciocínio lógico aparente. Não há "compreensão" mística — há milhões de neurônios artificiais empilhados, cada um ajustando pesos via álgebra linear, cuja **composição coletiva** produz comportamento emergente sofisticado. É a mesma lógica de como bilhões de neurônios biológicos simples, conectados em rede, produzem a consciência humana — embora a analogia tenha limites que exploraremos criticamente ao longo do curso.

✔ Takeaways

- Em IA I, o programador dita a lógica (Fuzzy) ou a busca (AG). Em IA II, a máquina **aprende a lógica** via dados.
- O Perceptron toma decisões algorítmicas usando a fórmula: $z = (X \cdot W) + b$ (soma ponderada + viés).
- A IA moderna exige processamento vetorizado (NumPy + SIMD) para velocidade, tornando laços `for` obsoletos.
- Um único Perceptron só traça linhas retas e falha em problemas não lineares, como o XOR.
- O *Multilayer Perceptron* (MLP) com camadas não lineares resolve o XOR e aproxima qualquer função contínua (Teorema da Aproximação Universal).

4 Conclusão

Nesta aula, percorremos a jornada teórica desde a reflexão sobre IA I (onde o programador dita a lógica) até o paradigma de IA II (onde a máquina extrai padrões dos dados). Passamos pelas metáforas humanas que nos ajudam a comunicar — criatividade como combinação do improvável, alucinação como resposta forçada sob incerteza — e as conectamos à mecânica real.

Construímos a intuição do Perceptron desde a decisão cotidiana de ir ao festival, formalizamos o produto escalar vetorizado, entendemos por que o `NumPy` é essencial, e descobrimos o limite fatal do neurônio isolado (XOR). A solução — empilhar camadas no MLP — abriu a porta para o Teorema da Aproximação Universal e para tudo que está por vir.

✎ Questões: Autoavaliação

1 **[Direta]**Qual é o papel dos pesos (W) e do viés (b) na equação matemática de um Perceptron?

2 **[Direta]**Explique por que o uso da biblioteca NumPy (Álgebra Linear e SIMD) é obrigatório em implementações modernas, em contraposição ao uso de laços clássicos.

3 **[Direta]**Descreva como o problema da porta lógica XOR evidenciou a maior fraqueza teórica do Perceptron isolado.

4 **[Reflexiva]**Se fôssemos modelar a decisão de "contratar ou não um candidato" como um Perceptron simples, cite três atributos de entrada que você usaria. Que riscos éticos a definição manual dos pesos por um programador poderia introduzir no processo seletivo?

5 **[Reflexiva]**À luz da metáfora da prova e da alucinação abordada neste capítulo, justifique a afirmação: "Quando um modelo generativo inventa uma resposta incorreta (alucina), ele não está defeituoso; ele está executando exatamente a sua função estatística e arquitetural".

Lab. 01: O Primeiro Neurônio e o Versionamento

⋆ Objetivo do Laboratório

Bem-vindo ao seu primeiro dia prático. A missão de hoje é estabelecer a base da sua carreira como Engenheiro de Sistemas Inteligentes. Faremos duas coisas que são vitais para o mercado de trabalho:

1. Montar um fluxo corporativo de versionamento de código no GitLab.
2. Programar a matemática "crua" de um neurônio artificial utilizando apenas Python puro e Álgebra Linear, entendendo que a Inteligência Artificial não possui mágica, apenas matrizes.

—

5 Parte 1: Ambiente e Cultura Git

Na indústria tecnológica de alto impacto, nenhum código sobrevive isolado na máquina de um único engenheiro.

A. Criando o Repositório Institucional a partir do Template

1. Acesse o **GitLab Institucional** em `https://git.juninho.com.br`.
2. Navegue até o repositório modelo público: `https://git.juninho.com.br/cefet-ia2-2026/modulo-1/ia2-modulo1-template`.
3. Clique no botão **Fork** no canto superior direito para criar a sua cópia pessoal do projeto no GitLab.
4. Clone o seu Fork no seu computador de trabalho:

```
git clone https://git.juninho.com.br/<seu-usuario>/ia2-modulo1-template.git
cd ia2-modulo1-template
```

B. Clonando e Configurando na Máquina Local

1. Abra o terminal do computador do laboratório.
2. Clone o seu novo repositório: `git clone URL_DO_SEU_REPOSITORIO`.
3. Entre na pasta gerada: `cd ia2-2025-SeuNome`.
4. Abra a pasta no seu editor de código (como o VS Code): `code .`.

—

6 Parte 2: O Desafio do Perceptron

Nós não vamos usar *TensorFlow* ou bibliotecas avançadas prontas. Você vai construir o "cérebro" de um neurônio na unha utilizando a biblioteca `NumPy`.

Problema A: O Robô Jardineiro

Contexto: A prefeitura de *TechVille* instalou um robô autônomo na praça central para regar o jardim. A bateria é cara, então ele só deve ligar a mangueira quando realmente for necessário. Os engenheiros da prefeitura já calibraram os sensores e determinaram os pesos ideais — a sua missão é **traduzir esse projeto em código**.

Modelo de Entrada (X): Vetor com 3 leituras dos sensores do robô (0 ou 1):

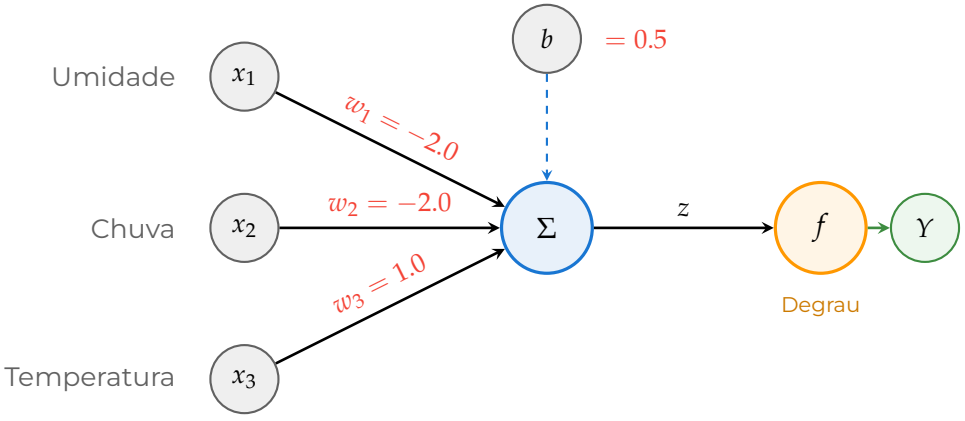
- x_1: Umidade do Solo (0 = Seco, 1 = Molhado)
- x_2: Previsão de Chuva (0 = Céu Limpo, 1 = Tempestade)
- x_3: Temperatura (0 = Frio, 1 = Quente)

Modelo de Saída (Y): Decisão binária via Função Degrau (limiar $\theta = 1.0$):

- $Y = 1$ → Ligar a mangueira
- $Y = 0$ → Ficar em repouso

A. O Projeto do Neurônio

Os engenheiros entregaram a você o seguinte diagrama com os pesos já calibrados. O neurônio calcula a soma ponderada $z = \mathbf{x} \cdot \mathbf{w} + b$ e aplica a função degrau para decidir.



$$z = x_1 \cdot (-2.0) + x_2 \cdot (-2.0) + x_3 \cdot (1.0) + 0.5$$

$$Y = \begin{cases} 1, & \text{se } z \geq 1.0 \\ 0, & \text{caso contrário} \end{cases}$$

B. Tabela-Verdade Esperada

Com os pesos do diagrama acima, o comportamento esperado do robô para **todas** as combinações possíveis dos sensores é:

x_1 **(Umidade)**	x_2 **(Chuva)**	x_3 **(Temp.)**	$z = \mathbf{x} \cdot \mathbf{w} + b$	Y **(Decisão)**
0 (Seco)	0 (Limpo)	0 (Frio)	0.5	0 — Repouso
0 (Seco)	0 (Limpo)	1 (Quente)	1.5	**1 — Regar!**
0 (Seco)	1 (Chuva)	0 (Frio)	−1.5	0 — Repouso
0 (Seco)	1 (Chuva)	1 (Quente)	−0.5	0 — Repouso
1 (Molhado)	0 (Limpo)	0 (Frio)	−1.5	0 — Repouso
1 (Molhado)	0 (Limpo)	1 (Quente)	−0.5	0 — Repouso
1 (Molhado)	1 (Chuva)	0 (Frio)	−3.5	0 — Repouso
1 (Molhado)	1 (Chuva)	1 (Quente)	−2.5	0 — Repouso

Observe: o robô só liga a mangueira no **único** cenário em que faz sentido — solo seco, sem previsão de chuva, e dia quente. Os pesos negativos em w_1 e w_2 funcionam como "penalidades" que inibem o disparo quando já está molhado ou quando vai chover.

C. Implementando o Código

Agora traduza o diagrama acima para código. Crie o arquivo `src/neuronio.py` e implemente as duas funções. O `numpy` aplica o produto escalar sem precisarmos de laços `for`:

```python
import numpy as np

def funcao_degrau(soma):
    """Função de Ativação Degrau (Step Function).
    Retorna 1 se soma >= limiar, 0 caso contrário."""
    return 1 if soma >= 1.0 else 0

def neuronio(entradas, pesos, vies):
    """Equação do Perceptron: Y = f(X . W + b)"""
    soma_ponderada = np.dot(entradas, pesos) + vies
    return funcao_degrau(soma_ponderada)

if __name__ == "__main__":
    # Pesos calibrados pelos engenheiros (do diagrama)
    W = np.array([-2.0, -2.0, 1.0])
    b = 0.5

    # Teste: Solo Seco(0), Sem Chuva(0), Quente(1) -> deve regar (1)
    X = np.array([0, 0, 1])
    decisao = neuronio(X, W, b)
    print(f"Cenário {X} -> Decisão: {decisao}")

```

```
    # Teste: Solo Molhado(1), Sem Chuva(0), Quente(1) -> repouso (0)
    X2 = np.array([1, 0, 1])
    decisao2 = neuronio(X2, W, b)
    print(f"Cenário {X2} -> Decisão: {decisao2}")
```

—

7 Parte 3: Garantia de Qualidade (Testes Unitários)

Em sistemas reais de Inteligência Artificial, a predição deve ser rigorosamente testada antes da entrega. Os testes já estão prontos no arquivo `tests/test_perceptron.py` do repositório. Abra-o e observe a estrutura:

```
import numpy as np
from src.neuronio import neuronio

# Pesos do diagrama do Robô Jardineiro
W = np.array([-2.0, -2.0, 1.0])
b = 0.5

def test_regar_cenario_ideal():
    """Solo Seco, Sem Chuva, Quente -> deve regar (1)"""
    assert neuronio(np.array([0, 0, 1]), W, b) == 1

def test_repouso_solo_molhado():
    """Solo Molhado, Sem Chuva, Quente -> repouso (0)"""
    assert neuronio(np.array([1, 0, 1]), W, b) == 0

def test_repouso_chuva():
    """Solo Seco, Chuva, Quente -> repouso (0)"""
    assert neuronio(np.array([0, 1, 1]), W, b) == 0

def test_repouso_frio():
    """Solo Seco, Sem Chuva, Frio -> repouso (0)"""
    assert neuronio(np.array([0, 0, 0]), W, b) == 0
```

1. Rode os testes no terminal com o comando: `pytest tests/ -v`.
2. Se a tela exibir **4 passed** em verde, significa que o seu neurônio reproduz corretamente a tabela-verdade do diagrama.

—

8 Parte 4: Integração Contínua (A Entrega)

Sua implementação está pronta e validada localmente. Agora vamos enviar o código para o servidor e deixar o pipeline de CI/CD confirmar que tudo está correto.

1. No terminal, adicione os arquivos à zona de preparação: `git add src/neuronio.py`.
2. Empacote a versão com um comentário claro: `git commit -m "Lab 01: Implementação do Perceptron Jardineiro"`.
3. Envie a atualização para o servidor remoto: `git push origin main`.

Critério de Avaliação: O pipeline de CI/CD do GitLab irá interceptar o seu *push*, criará um ambiente em nuvem invisível, e executará o `pytest` automaticamente. Você terá nota máxima ao obter o "Selo Verde" de aprovação na plataforma.

✔ Takeaways

Parabéns! Você acabou de implementar a matemática fundamental de um neurônio artificial — o mesmo bloco que, empilhado milhões de vezes, forma o ChatGPT e os sistemas de visão computacional modernos. Na próxima aula, vamos empilhar vários desses neurônios para criar uma **Rede Neural Multicamadas (MLP)** e resolver problemas que um único Perceptron não consegue.