Inteligência Artificial 2

Redes Profundas (MLP) e Forward Propagation

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

18 de agosto de 2026

Onde Paramos?

Na aula anterior, construímos o **Perceptron** — o átomo da IA:

- Entradas X, Pesos W, Viés b e Soma Ponderada $z = X \cdot W + b$.
- A vetorização NumPy (`np.dot`) que substitui os laços FOR.
- A Regra de Ouro dos Shapes: a dimensão interna **deve coincidir**.

A Pergunta de Hoje

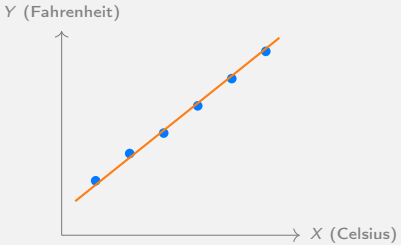
Temos o neurônio. Mas ele é **burro** — só traça retas.
Como fazê-lo **inteligente**?

Objetivos da Sessão

- **A Evolução Estrutural:** Escalar do perceptron simples para uma arquitetura profunda (MLP - Multilayer Perceptron).
- **A Quebra da Linearidade:** Compreender o papel insubstituível das Funções de Ativação (como a ReLU) no mapeamento de problemas reais.
- **A Viagem da Informação:** Explorar a matemática do *Forward Propagation* e o rastreamento das dimensões de matrizes pelo grafo computacional.

Revisitando a Aula 1: Regressão vs Classificação Linear

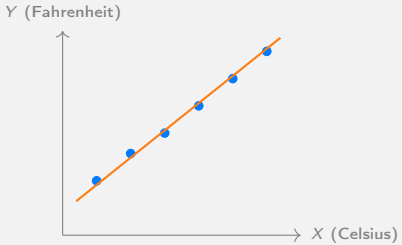
Regressão (Prevendo Valores Contínuos)



Na aula passada, o Perceptron atuou traçando uma reta que passa **sobre** os pontos, para prever valores em qualquer ponto do eixo X.

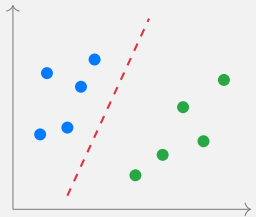
Revisitando a Aula 1: Regressão vs Classificação Linear

Regressão (Prevendo Valores Contínuos)



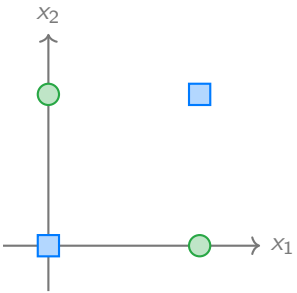
Na aula passada, o Perceptron atuou traçando uma reta que passa **sobre** os pontos, para prever valores em qualquer ponto do eixo X.

Classificação Linear (Separando Classes)



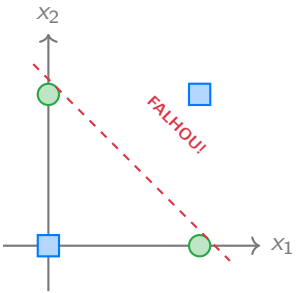
O mesmo Perceptron pode dividir o espaço. Tudo à esquerda da reta é da Classe Azul (Gato), à direita é Verde (Cachorro).

O Limite de um Neurônio Solitário: O Paradoxo do XOR



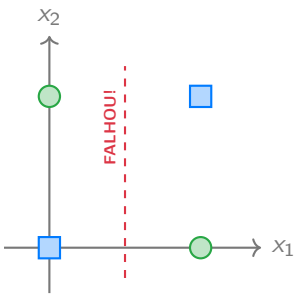
- Em problemas reais, os dados formam padrões complexos no espaço cartesiano (ex: o clássico **XOR**).

O Limite de um Neurônio Solitário: O Paradoxo do XOR



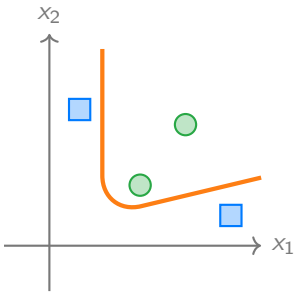
- Em problemas reais, os dados formam padrões complexos no espaço cartesiano (ex: o clássico **XOR**).
- Tente traçar **uma única reta** que separe os quadrados azuis das bolinhas verdes.

O Limite de um Neurônio Solitário: O Paradoxo do XOR



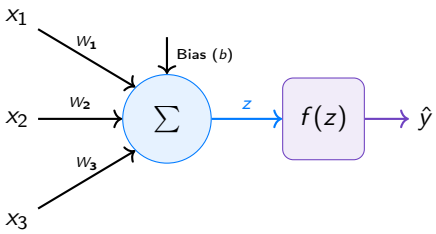
- Em problemas reais, os dados formam padrões complexos no espaço cartesiano (ex: o clássico **XOR**).
- Tente traçar **uma única reta** que separe os quadrados azuis das bolinhas verdes.
- **É impossível.** Se você girar ou deslocar a reta, sempre haverá invasores do lado errado. Um neurônio linear falha aqui!

A Solução: Fronteiras de Decisão Não-Lineares



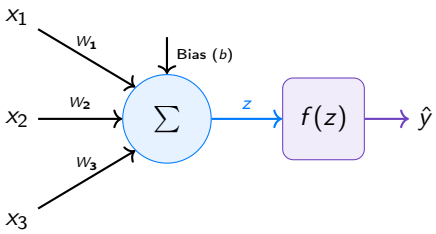
- Para separar os pontos verdes dos azuis, precisamos de uma curva!
- As **Redes Neurais Profundas** (com suas funções de ativação) conseguem dobrar o espaço e desenhar **fronteiras não-lineares**.
- É assim que a IA entende padrões complexos de voz, rostos e textos.

A Anatomia: Múltiplas Entradas e a "Maquininha"de Ativação



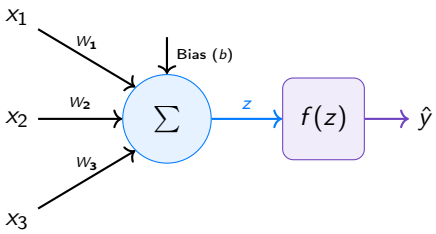
- **1. O Somatório (A Reta):** Multiplicamos os pesos W e somamos o Bias b num **Produto Escalar**. $z = X \cdot W + b$.

A Anatomia: Múltiplas Entradas e a "Maquininha"de Ativação



- **1. O Somatório (A Reta):** Multiplicamos os pesos W e somamos o Bias b num **Produto Escalar**. $z = X \cdot W + b$.
- **2. Função de Ativação:** É a "maquininha"que recebe a equação dessa reta infinita.

A Anatomia: Múltiplas Entradas e a "Maquininha"de Ativação



- **1. O Somatório (A Reta):** Multiplicamos os pesos W e somamos o Bias b num **Produto Escalar**. $z = X \cdot W + b$.
- **2. Função de Ativação:** É a "maquininha"que recebe a equação dessa reta infinita.
- **O Truque:** A maquininha transforma a reta e permite curvar o espaço, habilitando a classificação não-linear!

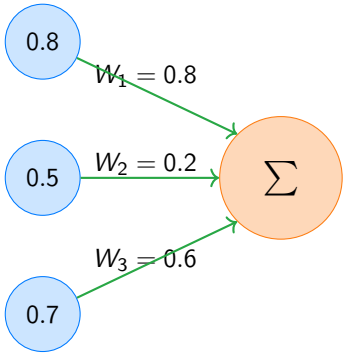
Exemplo Prático: Classificando Alunos

Objetivo: Prever se o aluno foi Aprovado (1) ou Reprovado (0) com base em 3 características.

Estudou (x_1)	**Conhecimento Prévio** (x_2)	**Facilidade** (x_3)	**Nome**	**Resultado** (y)
0.8	0.5	0.7	Miguel	**1 (Passou)**
0.3	0.2	0.8	Bruno	**0 (Reprovou)**

- Temos 3 entradas numéricas (x_1, x_2, x_3) e um problema de **Classificação Binária**.
- Vamos passar os dados do Miguel por um Perceptron já treinado!

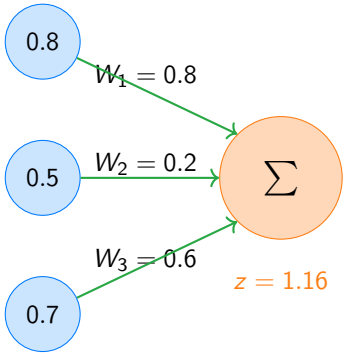
Exemplo Prático: O Somatório de Miguel



1. O Somatório ($X \cdot W$)

Multiplicamos as entradas pelos pesos (aprendidos):

Exemplo Prático: O Somatório de Miguel

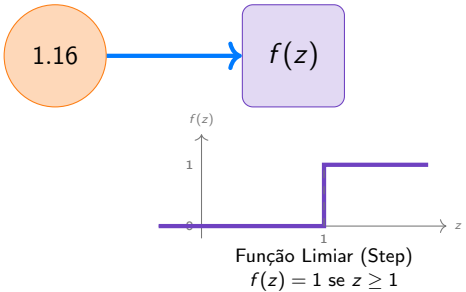


1. O Somatório ($X \cdot W$)

Multiplicamos as entradas pelos pesos (aprendidos):

$$
\begin{aligned}
z &= (0.8 \times 0.8) \\
&\quad + (0.5 \times 0.2) \\
&\quad + (0.7 \times 0.6) \\
z &= 0.64 + 0.10 + 0.42 \\
z &= 1.16
\end{aligned}
$$

Exemplo Prático: A "Maquininha"de Ativação

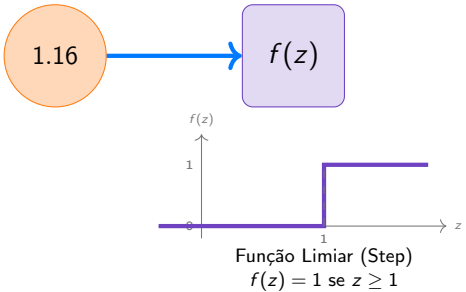


Função Limiar (Step)
$f(z) = 1$ se $z \geq 1$

2. A Ativação (O Limiar)

- O valor final da reta foi $z = 1.16$.

Exemplo Prático: A "Maquininha"de Ativação

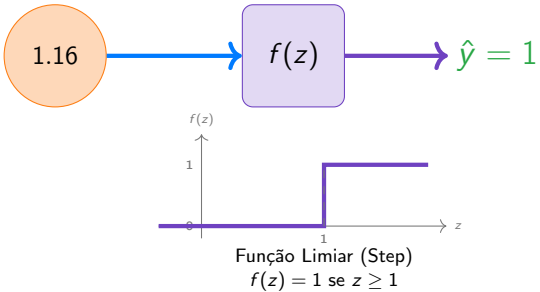


Função Limiar (Step)
$f(z) = 1$ se $z \geq 1$

2. A Ativação (O Limiar)

- O valor final da reta foi $z = 1.16$.
- A função **Limiar (Threshold)** que usamos corta em 1.

Exemplo Prático: A "Maquininha"de Ativação

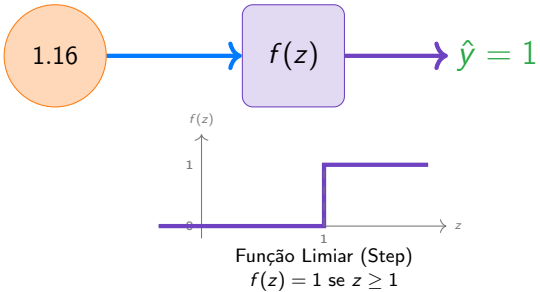


Função Limiar (Step)
$f(z) = 1$ se $z \geq 1$

2. A Ativação (O Limiar)

- O valor final da reta foi $z = 1.16$.
- A função **Limiar (Threshold)** que usamos corta em 1.
- Como $1.16 \geq 1$, a função dispara e retorna **1**.

Exemplo Prático: A "Maquininha"de Ativação

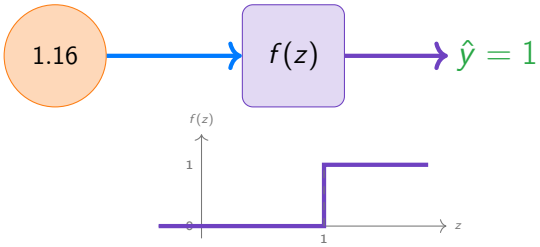


Função Limiar (Step)
$f(z) = 1$ se $z \geq 1$

2. A Ativação (O Limiar)

- O valor final da reta foi $z = 1.16$.
- A função **Limiar (Threshold)** que usamos corta em 1.
- Como $1.16 \geq 1$, a função dispara e retorna 1.
- A rede previu: **Miguel passou!**

Exemplo Prático: A "Maquininha"de Ativação

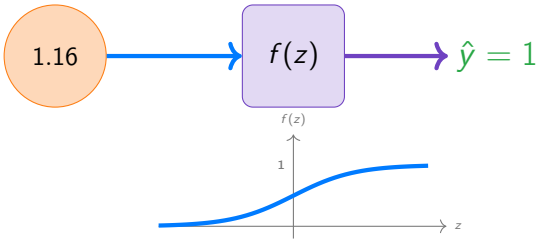


Função Limiar (Step)
$f(z) = 1$ se $z \geq 1$

2. A Ativação (O Limiar)

- O valor final da reta foi $z = 1.16$.
- A função **Limiar (Threshold)** que usamos corta em 1.
- Como $1.16 \geq 1$, a função dispara e retorna 1.
- A rede previu: **Miguel passou!**
- *Nota: Pro Bruno,* $z < 1 \implies f(z) = 0$.

Exemplo Prático: A "Maquininha"de Ativação

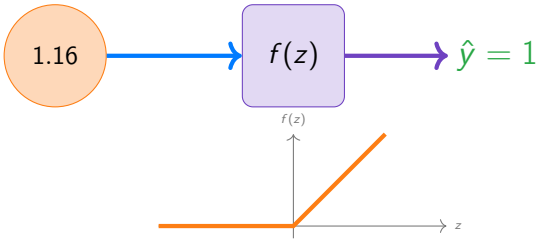


Spoiler: Sigmoid (Curva S)
Probabilidade suave de 0 a 1

2. A Ativação (O Limiar)

- O valor final da reta foi $z = 1.16$.
- A função **Limiar (Threshold)** que usamos corta em 1.
- Como $1.16 \geq 1$, a função dispara e retorna **1**.
- A rede previu: **Miguel passou!**
- *Nota: Pro Bruno,* $z < 1 \implies f(z) = 0$.
- **Spoiler:** Na prática, trocamos o Degrau rígido por curvas flexíveis (**Sigmoid**) ou rampas (**ReLU**)!

Exemplo Prático: A "Maquininha"de Ativação

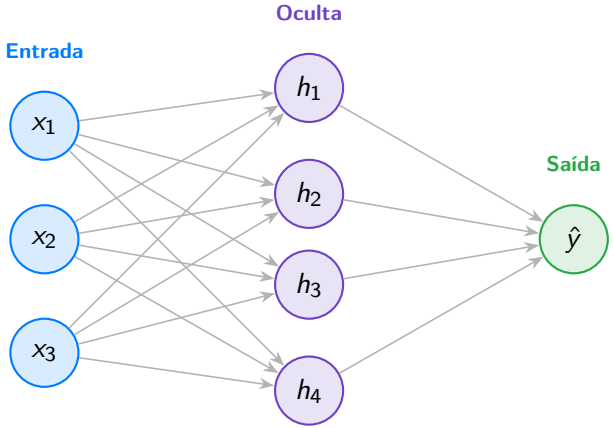


Spoiler: ReLU (Rampa)
Mantém o formato se for > 0

2. A Ativação (O Limiar)

- O valor final da reta foi $z = 1.16$.
- A função **Limiar (Threshold)** que usamos corta em 1.
- Como $1.16 \geq 1$, a função dispara e retorna 1.
- A rede previu: **Miguel passou!**
- *Nota: Pro Bruno,* $z < 1 \implies f(z) = 0$.
- **Spoiler:** Na prática, trocamos o Degrau rígido por curvas flexíveis (**Sigmoid**) ou rampas (**ReLU**)!

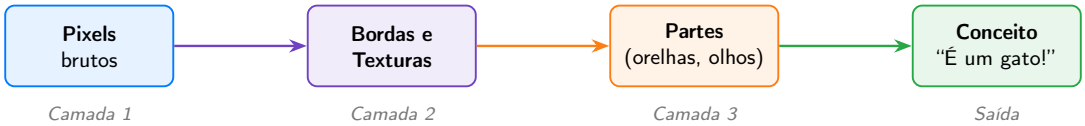
Multilayer Perceptron (MLP)



- Introduzimos as **Camadas Ocultas** (Hidden Layers).
- Os tensores atravessam o grafo camada por camada (*Forward Pass*).

As Camadas Ocultas: Extração de Padrões (Feature Extraction)

- O que acontece matematicamente nas **Hidden Layers**?
- A primeira camada oculta não tenta prever o resultado final, ela extrai **padrões intermediários** (bordas, texturas, formas).
- Camadas mais profundas combinam esses padrões em **conceitos de alto nível**.



A Ameaça do Colapso Linear

O que acontece se empilharmos apenas multiplicações de matrizes?

Camada 1: $Z_1 = X \cdot W_1$ Camada 2: $Z_2 = Z_1 \cdot W_2$

Substituindo Z_1 na Camada 2:

$$Z_2 = (X \cdot W_1) \cdot W_2 = X \cdot \underbrace{(W_1 \cdot W_2)}_{W_{eq}} \tag{1}$$

Conclusão: Uma rede de 100 camadas puramente lineares é matematicamente equivalente a uma rede de 1 única camada W_{eq}. O poder preditivo **não aumenta**!

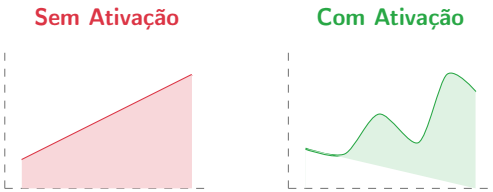
Analogia

É como empilhar filtros de óculos coloridos: azul + amarelo = verde. Não importa quantos filtros lineares você empilhe — no final, sempre existe **um único filtro equivalente**. Você precisa de algo que *quebre* essa fusão.

A Mágica da Não-Linearidade: Função de Ativação

Para impedir o colapso, aplicamos uma **Função de Ativação Não-Linear** imediatamente após cada multiplicação matricial.

- Ela **quebra** a propriedade associativa das matrizes.
- Permite que a rede "dobre" e "curve" o espaço vetorial.
- **Sem ela:** N camadas $= 1$ camada.
- **Com ela:** cada camada ganha poder expressivo real.



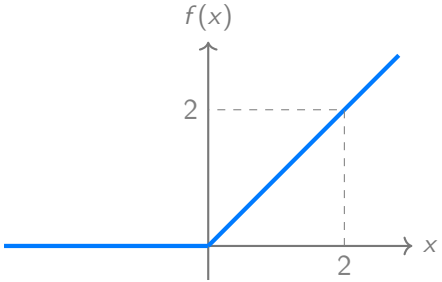
A Função ReLU

A **Rectified Linear Unit (ReLU)** é o padrão da indústria.

$$f(x) = \max(0, x) \qquad (2)$$

- Tudo que for negativo vira 0.
- Tudo que for positivo passa reto.

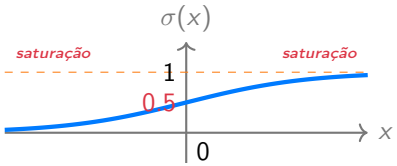
```
import numpy as np

z = np.array([-2.5, 0.0, 3.14])
ativacao = np.maximum(0, z)
print(ativacao) # [0.   0.   3.14]
```



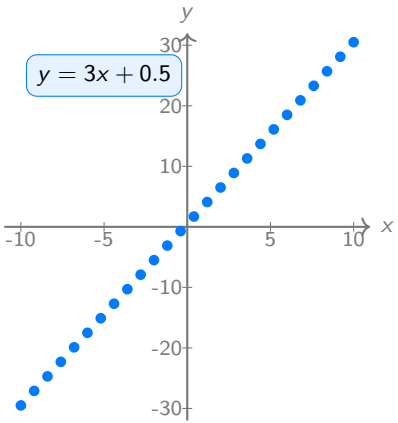
Outra Ativação Clássica: Sigmoid (σ)

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

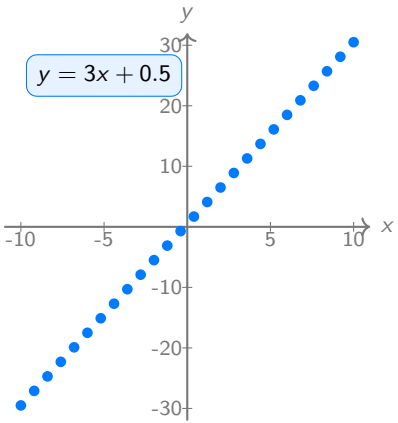
- Ela "esmaga" qualquer número num intervalo estrito de $(0, 1)$.
- Antigamente era muito usada nas camadas ocultas, mas perdeu força para a ReLU.
- Hoje, seu uso é **obrigatório** na Camada de Saída para prever **probabilidades** em classificações binárias.

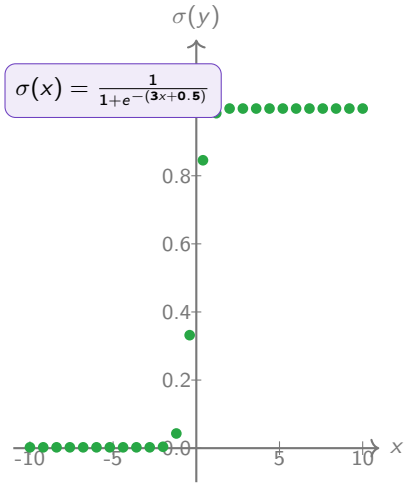


Na Prática: A Sigmoid "Esmagando"uma Reta

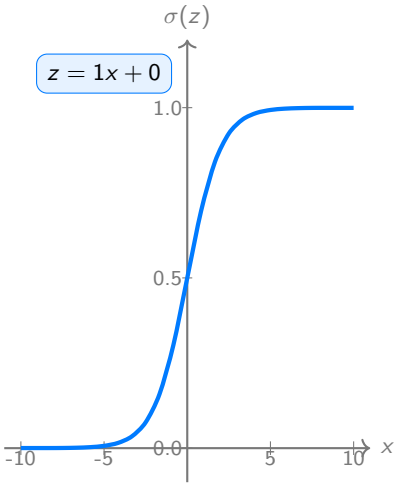


Na Prática: A Sigmoid "Esmagando"uma Reta





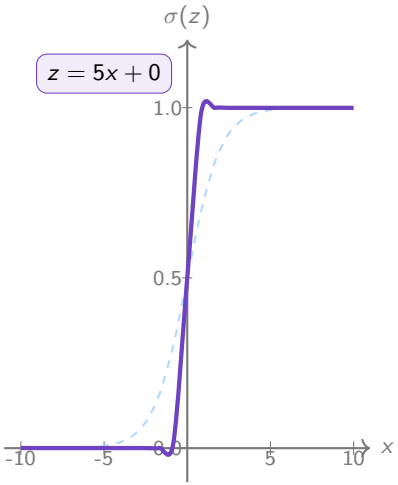
A Dança dos Pesos: Como W e b moldam a Sigmoid



O Padrão ($W = 1, b = 0$)

- A curva cruza o eixo central exatamente em 0.5.
- A transição entre 0 e 1 é gradual e bem distribuída.

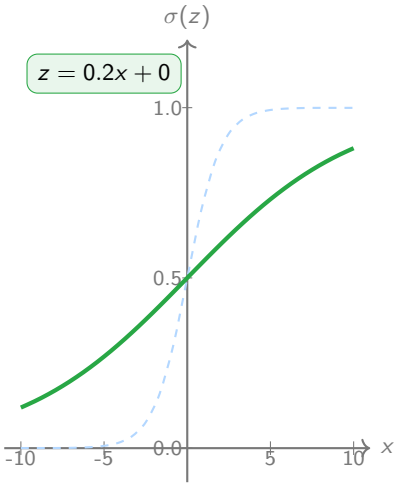
A Dança dos Pesos: Como W e b moldam a Sigmoid



Aumentando o Peso ($W = 5$)

- O Peso **W** controla a **inclinação**.
- **Efeito Prático:** O neurônio fica "confiante". A ativação sai de 0 para 1 quase instantaneamente, virando um *degrau* (*step*).

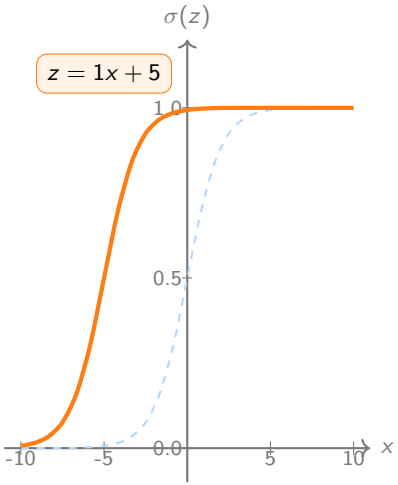
A Dança dos Pesos: Como W e b moldam a Sigmoid



Diminuindo o Peso ($W = 0.2$)

- O Peso **W** controla a **inclinação**.
- **Efeito Prático:** A transição fica super "preguiçosa". O neurônio ganha uma longa zona de incerteza antes de decidir.

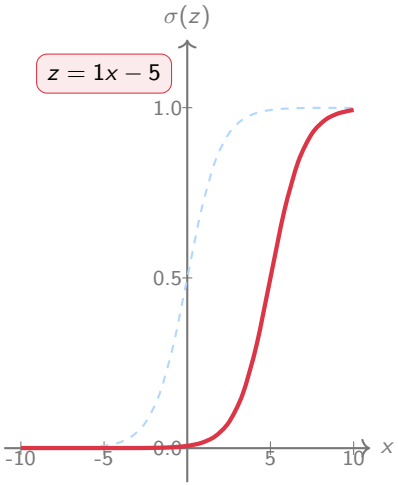
A Dança dos Pesos: Como W e b moldam a Sigmoid



Aumentando o Viés ($b = +5$)

- O Viés **b** (*Bias*) move a curva **horizontalmente**.
- **Efeito Prático:** Ao somar $+5$, a curva vai para a esquerda. O neurônio **dispara mais cedo**. Exige um estímulo x muito menor.

A Dança dos Pesos: Como W e b moldam a Sigmoid



Diminuindo o Viés ($b = -5$)

- O Viés **b** (*Bias*) move a curva **horizontalmente**.
- **Efeito Prático:** Com -5, a curva vai para a direita. O neurônio fica **resistente/teimoso**. Só dispara com um estímulo bem forte.

ReLU vs Sigmoid: Quando Usar Cada Uma?

	ReLU $f(x) = \max(0, x)$	**Sigmoid** $\sigma(x) = \frac{1}{1+e^{-x}}$
Intervalo de saída	$[0, +\infty)$	$(0, 1)$
Onde usar?	**Camadas Ocultas** (padrão!)	**Camada de Saída** (probabilidades)
Vantagem	Gradiente constante ($= 1$) nos positivos	Interpretação probabilística direta
Problema	"Neurônios mortos" (saída travada em 0)	**Vanishing Gradient** (saturação)
Velocidade	Ultra-rápida (`max`)	Lenta (e^{-x})

Regra Prática

Camadas ocultas $\rightarrow$ use ReLU. **Saída binária** $\rightarrow$ use Sigmoid.

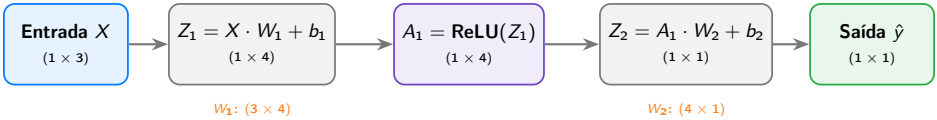
Escolhendo a Função de Ativação: O Guia Prático

Na prática, o tipo de problema dita qual função usar.

- **Nas Camadas Ocultas (Hidden Layers):**
  - **Use sempre ReLU** (ou variações: Leaky ReLU, GELU).
  - **Por quê?** Ela é matematicamente simples e evita que a rede trave (Gradient Vanishing) quando formos treinar.

- **Na Camada de Saída (Output Layer):**
  - **Regressão** (ex: prever um preço em R$): **Nenhuma (Linear)**. O valor deve sair puro.
  - **Classificação Binária** (ex: Spam ou Não Spam): **Sigmoid**. Mapeia a saída para uma probabilidade de 0 a 1.
  - **Classificação Multiclasse** (ex: Gato, Cachorro ou Urso): **Softmax** (veremos na próxima aula).

O Pipeline do Forward Propagation



Regra de Ouro

A dimensão interna de cada multiplicação $A_{(m \times \mathrm{n})} \cdot W_{(\mathrm{n} \times p)}$ **deve coincidir**. Se não coincidir: `ValueError: shapes not aligned`.

O Passo-a-Passo Numérico (Camada Oculta)

- Entrada $X = [1.0, 2.0]$
- Pesos $W_1 = \begin{bmatrix} 0.5 & -1.0 \\ 0.2 & 0.8 \end{bmatrix}$ e Viés $b_1 = [0.1, 0.1]$

1. Multiplicamos a matriz ($Z_1 = X \cdot W_1 + b_1$):

$$Z_1 = [(1)(0.5) + (2)(0.2), \quad (1)(-1.0) + (2)(0.8)] + [0.1, 0.1]$$
$$Z_1 = [0.9, \quad 0.6] + [0.1, 0.1] = [1.0, 0.7]$$

2. Aplicamos a ReLU:

$$A_1 = \text{ReLU}([1.0, \quad 0.7]) = [1.0, 0.7] \quad \text{(Ambos positivos — passam retos!)}$$

O Passo-a-Passo Numérico (Camada de Saída)

Agora levamos A_1 para a camada final.

- $A_1 = [1.0, 0.7]$ (resultado da Hidden Layer)
- Pesos $W_2 = \begin{bmatrix} 0.3 \\ -0.5 \end{bmatrix}$ e Viés $b_2 = [0.2]$

3. Multiplicamos ($Z_2 = A_1 \cdot W_2 + b_2$):

$$Z_2 = [(1.0)(0.3) + (0.7)(-0.5)] + [0.2]$$
$$Z_2 = [0.3 - 0.35] + [0.2] = [-0.05] + [0.2] = [0.15]$$

4. Resultado final (sem ativação para regressão):

$$\boxed{\hat{y} = 0.15}$$

A Viagem Completa

$$X_{(1\times2)} \xrightarrow{W_1} Z_{1(1\times2)} \xrightarrow{\text{ReLU}} A_{1(1\times2)} \xrightarrow{W_2} Z_{2(1\times1)} = \hat{y}$$

Matemática da Propagação (*Forward Pass*)

```
# Camada Oculta H1
Z1 = np.dot(X, W1) + b1
A1 = np.maximum(0, Z1) # ReLU Ativada!

# Camada de Saída (Output)
Z2 = np.dot(A1, W2) + b2
# Se for regressão, não precisa ativar.
# Se for classificação, aplicaremos Sigmoid (Aula 3).
```

Intuição Visual (TensorFlow Playground)

A matemática pura parece abstrata. Mas o que as ativações não-lineares estão fazendo com o espaço geométrico?

Atividade Dirigida: Vamos acessar o Playground agora e observar a mágica acontecendo.

Roteiro da Demonstração

1. Acesse `playground.tensorflow.org`
2. Dataset: selecione o padrão **XOR** (ícone com 4 quadrantes coloridos).
3. Arquitetura: coloque **2 hidden layers** com **4 neurônios cada**.
4. Ativação: selecione **ReLU**.
5. Clique **Play** e observe as fronteiras de decisão se curvando.

O que Observar

Repare como cada neurônio oculto desenha uma "fatia" do espaço. Juntas, as fatias formam a curva que separa o XOR — algo que um Perceptron **sozinho** jamais conseguiria.

Recapitulação — O que Levar Desta Aula

1. Um Perceptron sozinho = uma reta. Não resolve o XOR nem o mundo real.
2. O MLP empilha camadas ocultas que extraem padrões progressivamente complexos.
3. **Sem Ativação Não-Linear**, N camadas colapsam em 1 (o colapso linear).
4. **ReLU** = padrão da indústria. **Sigmoid** = obrigatória na saída para probabilidades.
5. O *Forward Pass* é uma cascata de $Z = X \cdot W + b$ seguido de ativação.

Próxima Aula

Já sabemos como a informação *vai* da entrada à saída. Mas como os pesos W são **ajustados** para que a rede aprenda? A resposta: **Descida do Gradiente** e **Backpropagation**.

De Volta ao Laboratório: Lab 01 vs Lab 02

Lab 01: O Perceptron

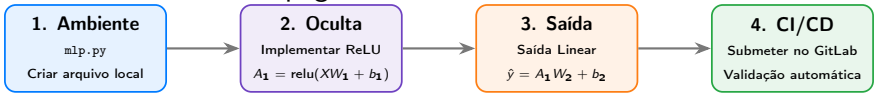
- Você construiu um classificador linear simples.
- Equação: $\hat{y} = \sigma(X \cdot W + b)$.
- **Limitação:** Seu código só conseguia traçar *retas*. Problemas como o XOR eram impossíveis de resolver com uma única camada.

Lab 02: A Rede Neural Profunda

- Vamos evoluir seu código empilhando matrizes e não-linearidades!
- Equações: $A_1 = \text{ReLU}(X \cdot W_1 + b_1)$ e $\hat{y} = A_1 \cdot W_2 + b_2$.
- **O Poder:** Você programará a base que permite curvar o espaço de decisão.

Roteiro do Laboratório 02

Missão: Codificar o Forward Propagation de uma MLP de 2 camadas do zero com NumPy.



Recap — Os Conceitos-Chave da Aula

1. O **Colapso Linear**: empilhar camadas lineares é equivalente a uma só camada ($W_1 \cdot W_2 = W_{eq}$).
2. A **ReLU** ($\max(0, z)$) quebra a linearidade — sem ela, redes profundas são inúteis.
3. **Forward Propagation**: $Z_1 = X \cdot W_1 + b_1 \rightarrow A_1 = \text{ReLU}(Z_1) \rightarrow \hat{y} = A_1 \cdot W_2 + b_2$.
4. **Batch Processing**: processar m amostras de uma vez via multiplicação matricial.
5. **Shapes**: a dimensão interna deve coincidir $(n \times \text{k}) \cdot (\text{k} \times p) = (n \times p)$.

Conexão com Álgebra Linear

A propriedade associativa das matrizes que vocês viram em Álgebra Linear é exatamente o que causa o Colapso Linear — e a ReLU é o que o **impede**.

Exercício Mental: Forward Pass na Mão

Dado: $X = [1, 0]$, $W_1 = \begin{bmatrix} 0.5 & -0.3 \\ 0.2 & 0.8 \end{bmatrix}$, $b_1 = [0, 0]$.

1. $Z_1 = X \cdot W_1 + b_1 = [1 \times 0.5 + 0 \times 0.2,\ 1 \times (-0.3) + 0 \times 0.8] = [0.5,\ -0.3]$
2. $A_1 = \text{ReLU}(Z_1) = [\max(0, 0.5),\ \max(0, -0.3)] = [0.5,\ 0]$

Observe a ReLU em Ação!

O valor -0.3 foi "cortado" para zero pela ReLU. Sem ela, ambos os valores passariam e a rede perderia a capacidade de "escolher" quais sinais são relevantes.

No Lab

Vocês farão exatamente esse cálculo com `np.dot(X, W1) + b1` e `np.maximum(0, Z1)`.

Mão na massa!

- Clone/Puxe o repositório da disciplina para ter acesso à estrutura do `lab02`.
- Rode os testes locais com `pytest test_mlp.py` antes de submeter.

Spoiler da Aula 3

Sua rede já propaga os tensores, mas as matrizes W ainda são inicializadas aleatoriamente! Na próxima aula, a rede aprenderá, *sozinha*, a encontrar os melhores pesos através do **Gradiente Descendente**.

Dúvidas?

Email: alessio@cefetmg.br
Aula: Segunda e Quarta-Feira, 7h as 8:40h (Sala 39)
WhatsApp: Dúvidas rápidas

Lembrete

Confira sua Issue no GitLab!

Próxima aula: Descida do Gradiente

100% da Aula 2 Concluída!