🧠 Arquitetura de uma Rede Neural Artificial – Overview Completo

---

🔹 1. Camada de Entrada (Input Layer)

- É o **ponto inicial da rede**.
- Cada neurônio representa uma **variável de entrada** (ex: pixel, palavra, sensor, número).
- **Não realiza cálculos**, apenas repassa os dados para a primeira camada oculta.

📌 *Exemplo:*
Uma imagem de 28x28 pixels terá **784 neurônios** na entrada.

🔹 2. Camadas Ocultas (Hidden Layers)

- Onde ocorre o **processamento e aprendizado de padrões**.

Cada camada aplica:

ini
CopiarEditar
$z = \Sigma(w_i * x_i) + b$
$y = f(z)$

-
  - w_i: pesos
  - x_i: entradas
  - b: bias
  - f: função de ativação (ReLU, Sigmoid, Tanh etc.)

✳️ **Comportamento:**

- Cada camada transforma os dados em **representações mais abstratas**.

- Quanto mais camadas → mais **profunda** é a rede (deep learning).

🔁 **Conexões Possíveis:**

Tipo de Camada	Conexão
Densa (Fully Connected)	Cada neurônio conectado a todos os da próxima
Convolucional (CNN)	Apenas regiões locais (imagens)
Recorrente (RNN/LSTM)	Com memória de estado para sequências

🔹 3. Camada de Saída (Output Layer)

- Define o **tipo de problema resolvido**:
  - Classificação binária: `Sigmoid`
  - Classificação multi-classe: `Softmax`
  - Regressão: `Linear`

📌 *Exemplo:*
Classificação de dígitos de 0 a 9 → 10 neurônios na saída com Softmax.

🔧 Componentes Matemáticos do Neurônio

Componente	Função
w_i	Peso: controla a influência de cada entrada
b	Bias: desloca a curva de ativação, ajustando o ponto de ativação
$f(z)$	Função de ativação: dá "vida" ao neurônio, introduzindo não linearidade

---

🧮 Funções de Ativação Comuns

Nome	Fórmula	Faixa da saída	Uso típico

ReLU	`max(0, z)`	`[0, ∞)`	Visão computacional, deep networks
Sigmoid	`1 / (1 + e⁻ᶻ)`	`(0, 1)`	Saídas probabilísticas
Tanh	`(eᶻ - e⁻ᶻ)/(eᶻ + e⁻ᶻ)`	`(-1, 1)`	Redes recorrentes, NLP
Softmax	`eᶻⁱ / Σeᶻ`	`(0, 1)`	Classificação multi-classe
Linear	`f(z) = z`	`ℝ`	Regressão

🔁 Fluxo da Rede Neural

Forward Pass

1. Entrada → propagada camada a camada.
2. Cada neurônio calcula: `z = Σ(wᵢ * xᵢ) + b`, depois aplica ativação.
3. A saída final é gerada.

Backpropagation

1. O erro da saída é comparado com o valor esperado.
2. A rede calcula **gradientes (∂E/∂w)** para cada peso.
3. Os pesos e bias são **ajustados usando um otimizador** (ex: Adam, SGD).

python
CopiarEditar

```
w := w - η * ∂E/∂w
b := b - η * ∂E/∂b
```

- `η`: taxa de aprendizado
- `∂E/∂w`: gradiente do erro

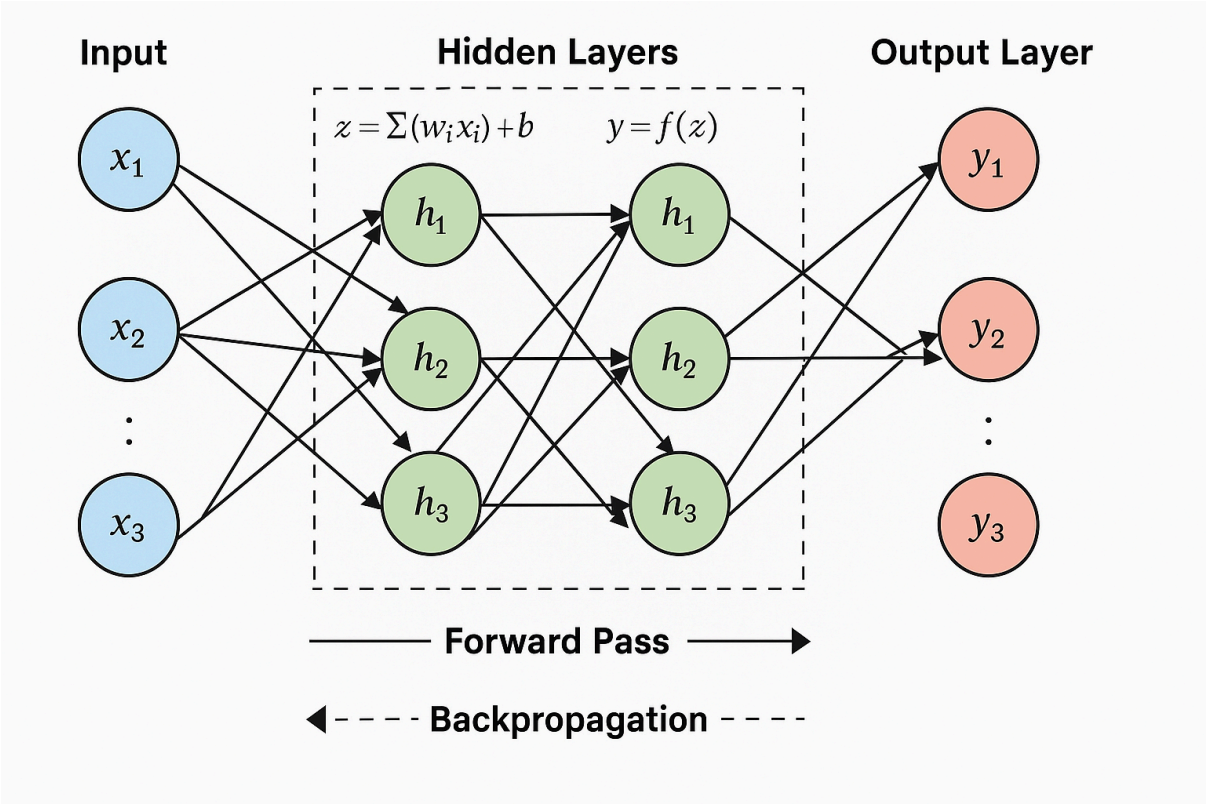
🎯 Resumo Visual da Arquitetura

scss
CopiarEditar

```
        Input Layer          Hidden Layer(s)          Output
Layer
       [x1]   [x2]   [x3]  →  [h1]   [h2]   [h3]   →   [y1]  [y2]
[y3]
             (features)       (transformações)
(resultado)
```

⚙️ Parâmetros de Arquitetura que você pode controlar

Parâmetro	Efeito no modelo
Nº de camadas ocultas	Maior profundidade = mais capacidade de aprender
Nº de neurônios/camada	Mais neurônios = mais representação
Funções de ativação	Mudam como os dados fluem e são aprendidos
Otimizador	Define a velocidade e estabilidade do aprendizado
Regularização	Evita overfitting (ex: Dropout, L2, BatchNorm)


Input
Hidden Layers
Output Layer
$z = \Sigma(w_i x_i) + b$
$y = f(z)$
x_1
x_2
x_3
h_1
h_2
h_3
h_1
h_2
h_3
y_1
y_2
y_3
Forward Pass
Backpropagation