⚖️ 1. Papel dos Pesos (`w`)

Os **pesos** controlam a **importância de cada entrada** (`x`) no resultado da soma.

- Um peso alto **valoriza mais** a entrada.
- Um peso baixo ou negativo **desvaloriza ou inverte** a influência da entrada.

Exemplo:

txt
CopiarEditar

```
Entrada x₁ = 0.8
Peso w₁ = 2.0
→ Contribuição: 0.8 * 2.0 = 1.6
```

Se outro input tiver peso -1.0:

txt
CopiarEditar

```
x₂ = 0.5, w₂ = -1.0 → -0.5
```

🎯 2. Papel do Bias (`b`)

O **bias** funciona como um "ajuste fino" na equação.
Ele **desloca o valor da soma ponderada**, ajudando o modelo a se ajustar melhor.

Sem bias, a rede pode ficar muito limitada — imagine que todos os neurônios só disparam se a soma exata for 0.

Com bias, a rede aprende a **deslocar o ponto de ativação**:

ini
CopiarEditar

```
z = Σ(wᵢ * xᵢ) + b
```

> Bias ≈ intercepto de uma reta:
> y = mx + b → b define onde a reta cruza o eixo y.

🧠 3. Como São Definidos Pesos e Bias?

No início, eles são **valores aleatórios pequenos**.

Depois, o modelo aprende os melhores valores durante o **treinamento** com base em erros cometidos.

Etapas:

1. **Inicialização:**
   Pesos e bias começam com números aleatórios (normalmente próximos de zero).

2. **Forward pass:**
   A entrada percorre a rede e produz uma saída.

3. **Erro:**
   Compara-se a saída real com a saída esperada (target).

4. **Backpropagation:**
   Calcula-se o gradiente do erro em relação a cada peso/bias.

5. **Atualização:**
   Pesos e bias são ajustados via **algoritmos de otimização**, como:

   - Gradient Descent
   - Adam
   - RMSprop

Fórmula de atualização:

go
CopiarEditar

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

- `η` é a **taxa de aprendizado**
- `∂E/∂w` é o gradiente do erro em relação ao peso

📈 Como Eles Afetam o Resultado?

Vamos ver na prática:

txt
CopiarEditar
```
x₁ = 1.0, x₂ = 2.0
w₁ = 0.5, w₂ = -1.0
b = 0.2

Soma: z = (1.0 * 0.5) + (2.0 * -1.0) + 0.2 = 0.5 - 2.0 + 0.2 = -1.3
```

Se usarmos função ReLU:

ini
CopiarEditar
```
y = max(0, -1.3) = 0 → Neurônio desativado
```

Mas se ajustarmos o **bias para 2.0**:

lua
CopiarEditar
```
z = 0.5 - 2.0 + 2.0 = 0.5
→ y = max(0, 0.5) = 0.5 → Neurônio ativado
```

Se quiser, posso simular esse comportamento com gráficos interativos ou códigos em Python. Quer ver na prática?