# Relatório LAMIA 12
# Prática: Redes Neurais (II)

**Kaique Medeiros Lima**

## 1 Introdução

Redes neurais são modelos computacionais que tentam imitar no funcionamento do cérebro humano. Elas são feitas por neurônios artificiais que se conectam entre si para realizar tarefas de aprendizado e reconhecimento de padrões. Esse relatório resume os conceitos fundamentais das redes neurais ensinados no curso *Deep Learning com Python de A a Z — O Curso Completo*, como eles funcionam, e como são treinados para realizar tarefas específicas.

## 2 Descrição da atividade

### 2.1 Fundamentos biológicos

Neurônios, assim como no cérebro, estão conectados entre si e trocam informações, sendo responsáveis pela inteligência humana. Essa troca de informações permite que os neurônios aprendam coisas novas.

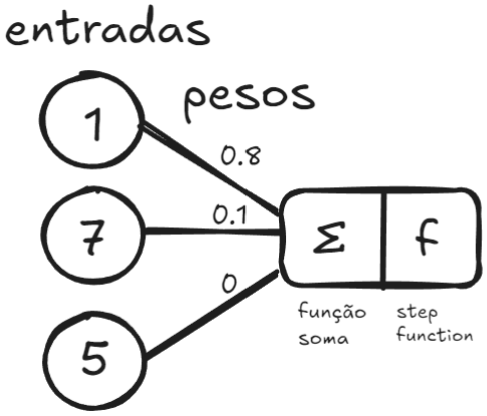
**Componentes do neurônio:**

- **Dendritos:** Recebem informações de outros neurônios.
- **Corpo celular:** Processa a informação recebida.
- **Axônio:** Transmite a informação processada até os terminais do axônio.
- **Terminais do axônio:** Enviam a informação processada para o próximo neurônio.

Essa troca de informação entre os neurônios é chamada de sinapse, e envolve processos elétricos que permitem a comunicação entre eles.

### 2.2 Perceptron de uma camada

Exemplo de Perceptron:




$$fsoma = \sum_{i=1}^{n} x_i \cdot w_i$$

Onde $x_i$ é a entrada, $w_i$ é o peso da entrada, e $n$ é o número de entradas.

### 2.2.1 Função Step

Se soma $\geq 1 \Rightarrow 1$, caso contrário 0. Os dendritos correspondem as entradas, o corpo celular realiza as operações matemáticas, e os terminais do axônio representam a saída do processamento feito no corpo celular.

### 2.2.2 Operador E (Lógica da computação)

Tabela examplo:

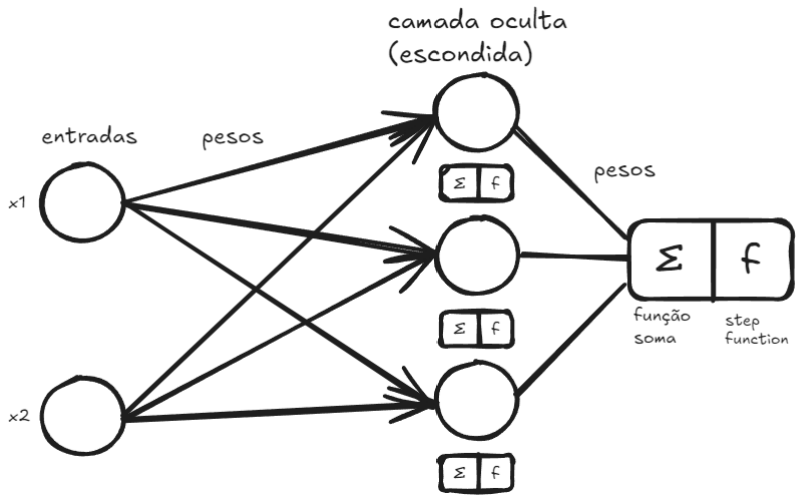
| x1 | x2 | classe |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

A saída é 1 somente se ambos os valores forem verdadeiros ($x_1 = 1$ e $x_2 = 1$).

```
if x1 == 1 and x2 == 1:
    classe = 1
```

## 2.3 Redes Multicamadas

Exemplo de rede multicamada:




### 2.3.1 Função Sigmóide

A função sigmóide é dada por:

$$y = \frac{1}{1 + e^{-x}}$$

Ela retorna valores entre 0 e 1. Se $x$ é alto, o valor será aproximadamente 1. Se $x$ é pequeno, o valor será aproximadamente 0. A função não retorna valores negativos.

### 2.3.2 Operador XOR

Exemplo de código Python para o operador XOR

```
if x1 == 1 or x2 == 1:
    classe = 1
```

### 2.3.3 Cálculo do Erro

O algoritmo mais simples para o cálculo do erro é:

$$erro = respostaCorreta - respostaCalculada$$

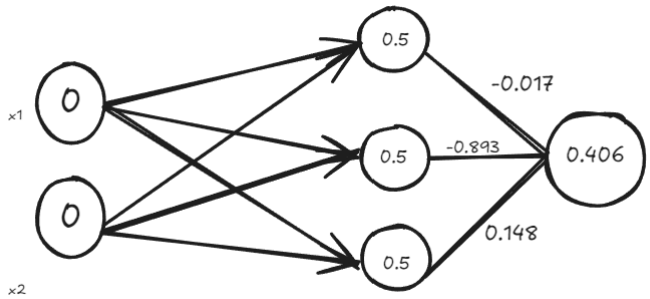
### 2.3.4 Descida do Gradiente por Derivada

A função sigmóide é dada por:

$$y = \frac{1}{1 + e^{-x}} \rightarrow \frac{d(y \cdot (1 - y))}{dx}$$

### 2.3.5 Cálculo do Parâmetro Delta

Exemplo de rede multicamada para cálculo do delta:




O processo envolve a função de ativação, a derivada da função, o cálculo do delta e do gradiente para chegar ao mínimo global onde o erro é menor. Para o cálculo do delta:

$$\begin{aligned}
\text{soma} &= -0.381 \\
\text{ativação} &= 0.406 \\
\text{erro} &= 0 - 0.406 = -0.406 \\
\text{derivada da ativação (sigmoide)} &= 0.241 \\
\text{deltaSaida} &= \text{erro} \times \text{derivadaSigmoide} = -0.406 \times 0.241 = -0.098 \\
\text{deltaEscondida} &= \text{derivadaSigmoide} \times \text{peso} \times \text{deltaSaida} \\
\text{deltaEscondida} &= 0.25 \times (-0.017) \times (-0.098) = 0.000 \\
\text{deltaEscondida} &= 0.25 \times (-0.893) \times (-0.098) = 0.022 \\
\text{deltaEscondida} &= 0.25 \times 0.148 \times (-0.098) = -0.004
\end{aligned}$$

## 2.4 Ajuste dos Pesos (Backpropagation)

$$peso_{n+1} = (peso_n * momento) + (entrada * delta * taxa\ de\ aprendizagem)$$

A fórmula deve ser aplicada em cada camada e passada ao próximo após o término da camada atual. O algoritmo minimiza o erro da rede neural. É dividida em duas fases: propagação e retropropagação. A propagação é a passagem da informação da entrada até a saída, e a retropropagação é a passagem do erro da saída até a entrada, assim atulizando os pesos. Essas atualizações são realizadas por métoidos de descida do gradiente.

### 2.4.1 Stochastic Gradient Descent

O Stochastic Gradient Descent (SGD) é um algoritmo para realizar o cálculo da função de perda. Diferente do Batch Gradient Descent (BGD), os pesos são atualizados para cada exemplo. Isso ajuda a prevenir mínimos locais e é mais rápido, pois não é necessário carregar todos os valores na memória. O SGD calcula o erro para cada registro e atualiza os pesos individualmente.

### 2.4.2 Batch Gradient Descent

O Batch Gradient Descent (BGD) é um algoritmo para realizar os cálculos da função de perda. Neste método, todos os dados são processados antes da atualização dos pesos. O erro é calculado para todos os registros e os pesos são atualizados somente após o cálculo completo.

### 2.4.3 Mini Batch Gradient Descent

O Mini Batch Gradient Descent é uma abordagem que escolhe um número de registros para processar e atualizar os pesos. Em vez de atualizar os pesos um por um (como no SGD) ou após todos os registros serem processados (como no BGD), você seleciona um lote de registros para rodar o algoritmo. Esse método oferece um equilíbrio entre a velocidade e a qualidade da atualização dos pesos. Por isso é o mais utilizado.

## 2.5 Bias, Erro e mais parâmetros

### 2.5.1 Bias

O bias é um valor que pode alterar a saída da rede, mesmo quando todas as entradas são zero. Ele é utilizado para ajustar a saída com uma unidade de bias, permitindo que a rede tenha um maior poder de representação.

### 2.5.2 Erro

Um algoritmo simples para o erro é calculado como:

$$\text{erro} = \text{respostaCorreta} - \text{respostaCalculada}$$

### 2.5.3 Mean Squared Error (MSE)

O erro quadrático médio é dado por:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

Erros maiores são penalizados mais com essa fórmula.

### 2.5.4 Root Mean Squared Error (RMSE)

O erro quadrático médio da raiz é calculado como:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2}$$

Essa fórmula penaliza mais os erros, dando maior peso a grandes diferenças entre a previsão e o valor real.

## 2.6 Tipos de camadas

### 2.6.1 Camada de Entrada

Responsável por receber os dados brutos e passá-los para as camadas seguintes.

### 2.6.2 Camada Oculta

Processa as informações recebidas, aplicando funções de soma e ativação para transformar os dados.

### 2.6.3 Camada de Saída

Gera o resultado final da rede neural, produzindo a saída da rede após o processamento das camadas anteriores.

## 2.7 Parâmetros

### 2.7.1 Learning Rate (Taxa de aprendizado)

O *learning rate* (taxa de aprendizado) determina a velocidade com que o algoritmo atualiza os pesos. Um valor maior pode acelerar o treinamento, mas pode causar instabilidade, enquanto um valor menor pode resultar em um treinamento mais lento, mas mais estável.

### 2.7.2 Batch Size (Tamanho do Lote)

O *batch size* define o número de registros que serão processados antes de atualizar os pesos. Um tamanho de lote maior pode melhorar a eficiência do treinamento, enquanto um tamanho menor pode fornecer uma atualização mais frequente dos pesos, mas com maior variabilidade.

### 2.7.3 Epochs (Épocas)

As *epochs* indicam quantas vezes o algoritmo deve atualizar os pesos passando por todos os registros da base de dados. Cada vez que todos os registros são processados e os pesos são atualizados, considera-se que uma época foi concluída. Um maior número de épocas pode levar a um melhor ajuste do modelo, mas também pode aumentar o risco de overfitting.

## 2.8 Underfitting e Overfitting

### 2.8.1 Underfitting

Os resultados do modelo estão ruins na base de treinamento. O Underfitting é quando o modelo não é complexo o suficiente para o problema. Resultando no mau desempenho.

### 2.8.2 Overfitting

Os resultados são bons na base de treinamento, mas são ruins na base de teste. Complexo demais para um problema mais simples. O modelo é também muito específico, pois se adaptou demais apenas para a base de treinamento, como se tivesse memorizado os dados. Irá apresentar muitos erros quando apresentado a novos dados.

# 3 Conclusão

Esse card foi de muita importância para o aprofundamento do conhecimento sobre o funcionamento das redes neurais, é interessante como a busca da inteligência artificial se dá através da imitação da natureza. As aulas de implementação foram bem reforçadas, implicando muito na prática de desenvolvimento de redes neurais e apresentaram conceitos fundamentais para a manipulação de dados para a interpretação do algoritmo de rede neural.