



PUC Minas

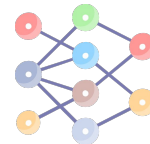
Frameworks para Deep Learning

Professor: Renan Santos Mendes

Introdução



Índice



Tópicos abordados nessa aula:

- Introdução
- Conceitos de POO
- Rede Neural Artificial
- Introdução ao Keras + Prática



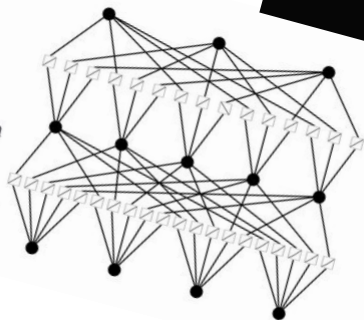
Introdução



Introducing Llama 3.1: Our most capable
to date

July 23, 2024 · 15 min

Decomposition



Mamba: A shallow dive into a new architecture for LLMs

GPT-4o mini: advancing cost-efficient intelligence

Introducing our most cost-efficient small model

Google DeepMind's new AI systems can now solve complex math problems

AlphaProof and AlphaGeometry 2 are steps toward building systems that can reason, which could unlock exciting new capabilities.

By Rhiannon Williams

July 25, 2024



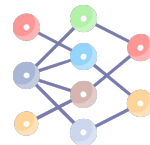
Bard é nova plataforma de IA do Google para rivalizar com ChatGPT

Corretores de imóveis nos EUA: Não imaginamos como trabalhar sem o ChatGPT agora

28/01/2023 às 20:51

Frameworks

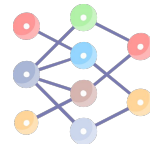




Frameworks

- Frameworks de deep learning **são bibliotecas de software para criação, treinamento e implantação de modelos** de redes neurais profundas em tarefas de aprendizado de máquina.
- Eles fornecem uma interface para definir, configurar e treinar modelos de deep learning, bem como realizar outras tarefas comuns.
- Os frameworks **oferecem implementações eficientes** de algoritmos de backpropagation e outras técnicas de otimização, o que ajuda a acelerar o processo de treinamento.
- Alguns exemplos de frameworks de deep learning populares incluem TensorFlow, PyTorch, Keras, Caffe, Theano, MXNet, entre outros.

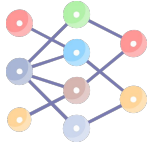




Frameworks

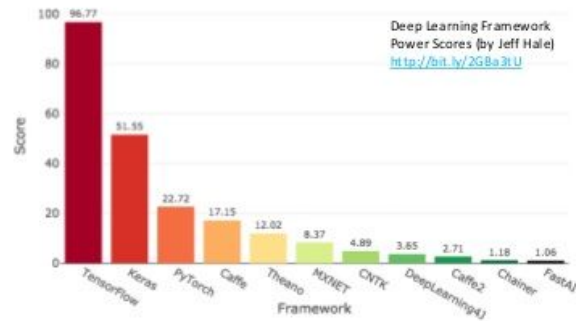
- Cada framework tem suas próprias características, pontos fortes e fracos, mas todos eles compartilham o objetivo comum de **tornar mais fácil e acessível a criação de modelos de deep learning** poderosos e eficientes.
- Os frameworks de deep learning são usados em diversas aplicações, como **visão computacional, NLP, speech**, entre outras áreas de **inteligência artificial**.
- Eles permitem que pesquisadores e engenheiros criem modelos de deep learning mais **rapidamente** e com **menos esforço**, permitindo que eles foquem em tarefas mais desafiadoras e criativas.





Frameworks

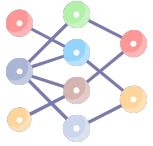
Deep Learning Frameworks



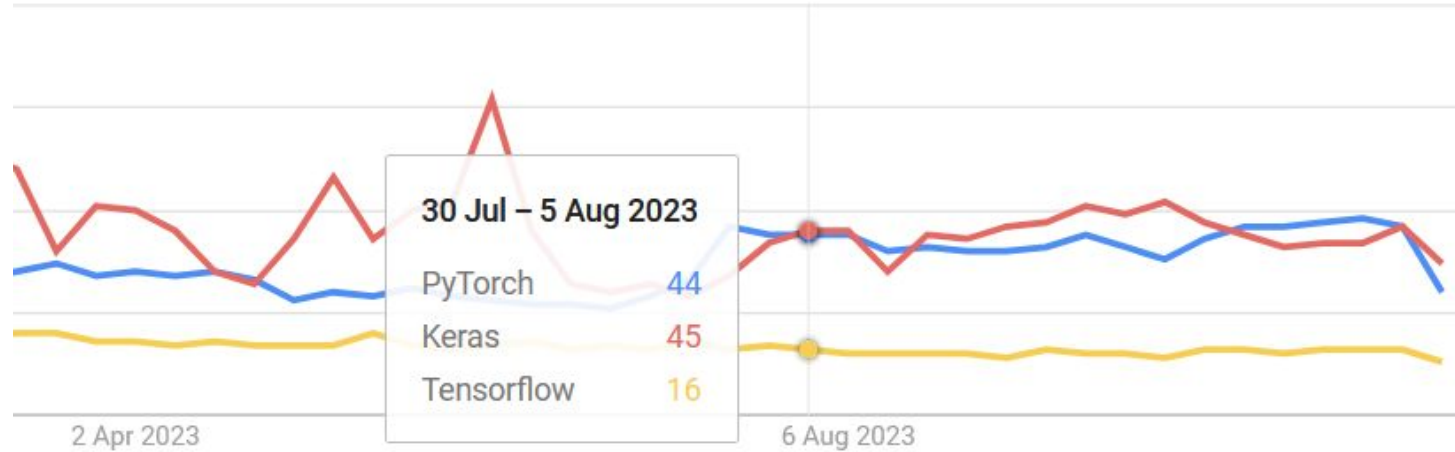
Factors to consider:

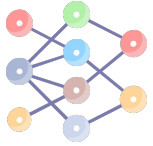
- Learning curve
- Speed of development
- Size and passion of community
- Number of papers implemented in framework
- Likelihood of long-term growth and stability
- Ecosystem of tooling

1.  TensorFlow
2.  Keras
3.  PyTorch
4.  Caffe
5.  theano
6.  mxnet
7.  CNTK
8.  DL4J
9.  Caffe2
10.  Chainer
11.  fast.ai



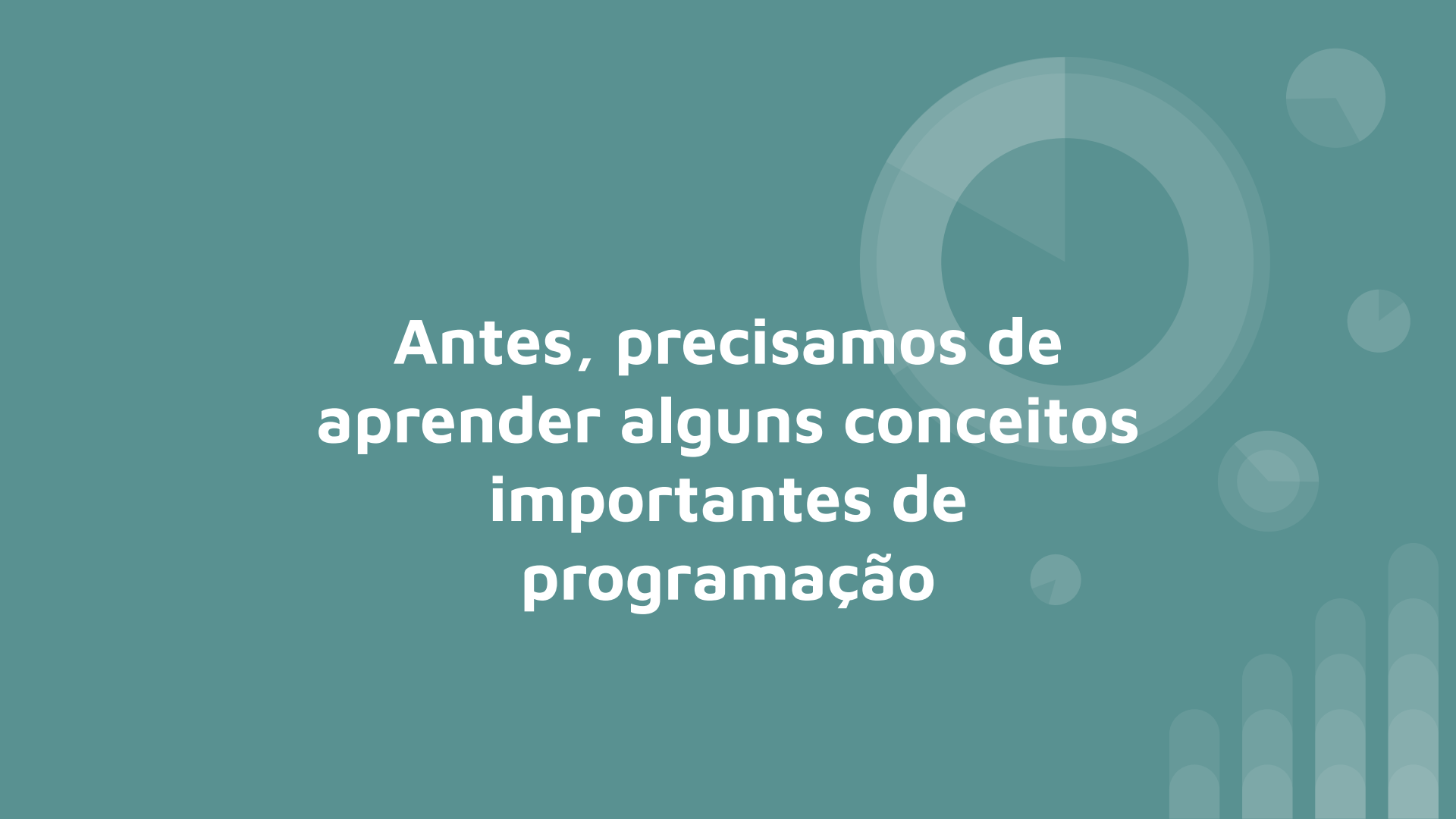
Frameworks





Frameworks





**Antes, precisamos de
aprender alguns conceitos
importantes de
programação**

Programação Orientada a Objetos (POO)





P OO - Introdução

- **Programação Orientada a Objetos** é um **paradigma de programação** baseado no conceito de **objetos**, que representam **entidades do mundo real**.
- Os objetos possuem **propriedades** (atributos) e **comportamentos** (métodos) específicos.
- O programa é organizado em **classes**, que são **modelos para a criação de objetos**, e **objetos**, que são **instâncias de uma classe**.
- Os objetos interagem entre si por meio de mensagens, que são enviadas de um objeto para outro.
- Cada objeto possui seu próprio estado e comportamento, e pode se comunicar com outros objetos para realizar ações mais complexas.





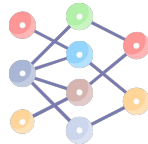
POO - Introdução

- A POO aumenta a **eficiência**, **modularidade**, **flexibilidade** e **reutilização de código** em programas mais complexos.
- Conceitos como **herança**, **encapsulamento**, **polimorfismo** e **abstração** permitem criar hierarquias de classes, esconder informações internas de uma classe, criar objetos com comportamentos diferentes a partir de uma mesma classe e simplificar o código ao definir interfaces mais abstratas.
- A POO é utilizada em diversas linguagens de programação, como Python, Java, C++, Ruby, entre outras.
- É especialmente útil em projetos de grande escala, onde a organização e reutilização do código são essenciais.



Conceitos Fundamentais





POO - Conceitos

- **Classes e objetos:** **classes** são estruturas que definem as características e comportamentos dos objetos. Os **objetos**, por sua vez, são instâncias de uma classe, ou seja, são criados a partir de uma classe.
- **Atributos e métodos:** **atributos** são as características dos objetos, como nome, idade, cor, entre outros. Já os **métodos** são as ações que os objetos podem realizar, como andar, falar, calcular, entre outros.





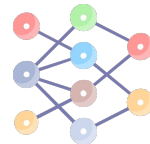
POO - Conceitos

- **Herança:** é o processo de **criar uma nova classe a partir de uma classe existente**, mantendo suas características e comportamentos. A nova classe (chamada de classe filha ou subclasse) pode adicionar novos atributos e métodos ou sobrescrever os existentes.
- **Encapsulamento:** é o princípio de **esconder a implementação** de um objeto de outros objetos. É possível tornar certos atributos e **métodos privados**, para que só possam ser acessados e modificados pela própria classe, garantindo maior segurança e controle de acesso às informações.



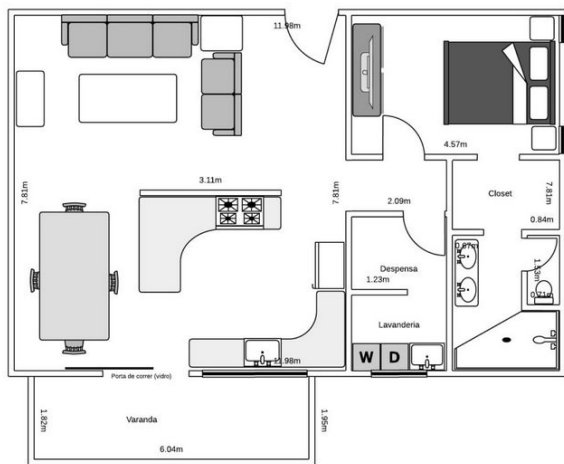
Diferenciando classes de objetos





POO - Conceitos

Classe



Planta → Projeto

Criação de
Uma Instância



Objeto



Casa → Instância





**Como fazemos isso
em Python?**



POO - Implementação

```
class Carro(object):  
    def __init__(self, marca, modelo, ano):  
        self.marca = marca  
        self.modelo = modelo  
        self.ano = ano  
  
    def acelerar(self):  
        print(f'{self.marca} {self.modelo} está acelerando...')
```



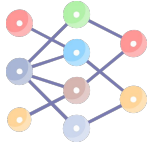


POO - Implementação

Criação da Classe

```
class Carro(object):  
    def __init__(self, marca, modelo, ano):  
        self.marca = marca  
        self.modelo = modelo  
        self.ano = ano  
  
    def acelerar(self):  
        print(f'{self.marca} {self.modelo} está acelerando...')
```



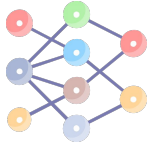


PОО - Implementação

```
class Carro(object):  
    def __init__(self, marca, modelo, ano):  
        self.marca = marca  
        self.modelo = modelo  
        self.ano = ano  
  
    def acelerar(self):  
        print(f'{self.marca} {self.modelo} está acelerando...')
```

Criação do
Construtor





PОО - Implementação

Uso da
Palavra "self"

```
class Carro(object):  
    def __init__(self, marca, modelo, ano):  
        self.marca = marca  
        self.modelo = modelo  
        self.ano = ano  
  
    def acelerar(self):  
        print(f'{self.marca} {self.modelo} está acelerando...')
```



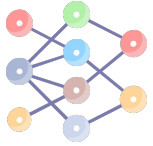


PОО - Implementação

Criação dos
Atributos

```
class Carro(object):  
    def __init__(self, marca, modelo, ano):  
        self.marca = marca  
        self.modelo = modelo  
        self.ano = ano  
  
    def acelerar(self):  
        print(f'{self.marca} {self.modelo} está acelerando...')
```

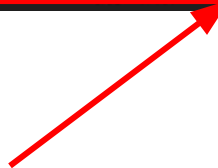




POO - Implementação

```
class Carro(object):  
    def __init__(self, marca, modelo, ano):  
        self.marca = marca  
        self.modelo = modelo  
        self.ano = ano  
  
    def acelerar(self):  
        print(f'{self.marca} {self.modelo} está acelerando...')
```

Criação dos
Métodos





P OO - Implementação

```
class Carro(object):  
    def __init__(self, marca, modelo, ano):  
        self.marca = marca  
        self.modelo = modelo  
        self.ano = ano  
  
    def acelerar(self):  
        print(f'[self.marca] {self.modelo} está acelerando...')
```

Uso dos Atributos
dentro dos métodos





POO - Implementação

- Uma classe é definida pela **palavra “class”** e em seguida seu nome;
- Usa-se os parênteses para indicar qual a classe mãe/pai;
- O método **__init__** é chamado de construtor;
- A palavra reservada **self** é usada para **acessar os atributos** e **métodos do objeto**;
- Para se criar um método, deve-se utilizar a palavra **def** (igual criamos uma função) e utilizar a palavra **“self”** como primeiro argumento;
- Para acessar um atributo dentro de um método, usa-se a palavra **self + nome do atributo**, por exemplo: `self.nome_do_atributo`



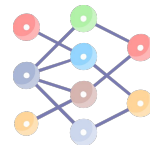
Neurônio Artificial



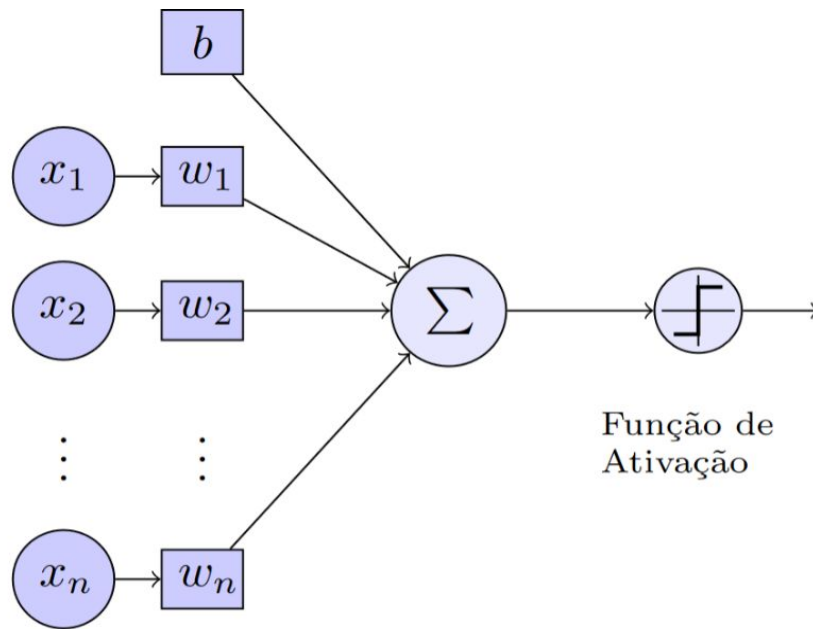
Recapitulando

Neurônio Artificial





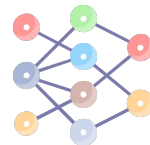
Neurônio Artificial



Entradas Pesos



PUC Minas



Neurônio Artificial

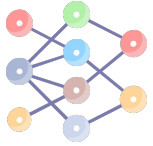
Elementos básicos do neurônio artificial:

1. **Conjunto de sinapses** em que cada uma é caracterizada por um peso (ou força) entre os neurônios com valores podendo ser positivos ou negativos;
2. **Somador** para agregar os sinais que chegam ao neurônio, ponderados pelos respectivos pesos (combinação linear);
3. **Função de ativação** que limita a amplitude da saída de um neurônio.

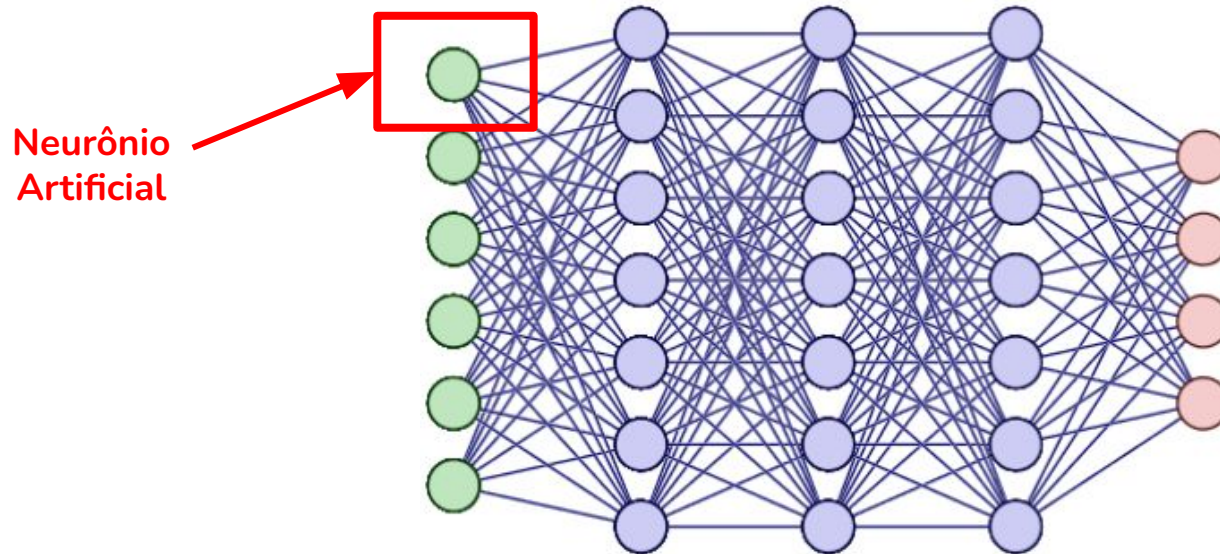


Rede Neural Artificial



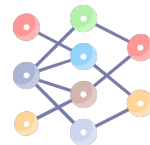


Rede Neural Artificial



Nomenclatura

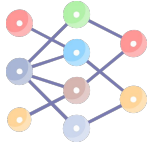




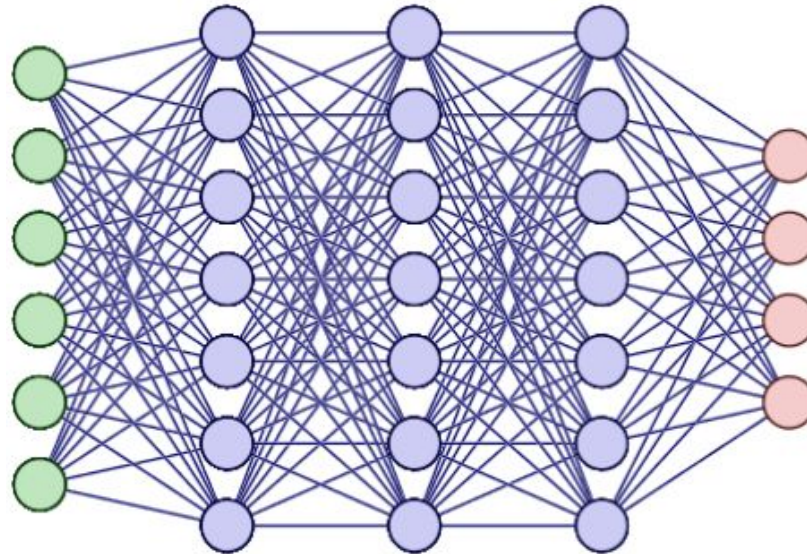
Nomenclatura

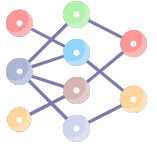
- **Camada de entrada:** camada pela qual os dados entram na rede;
- **Camada oculta:** uma ou mais camadas intermediárias;
- **Camada de saída:** camada por onde o processamento feito pela rede é obtido;
- **Conexões:** ligações entre os neurônios (pesos da rede).



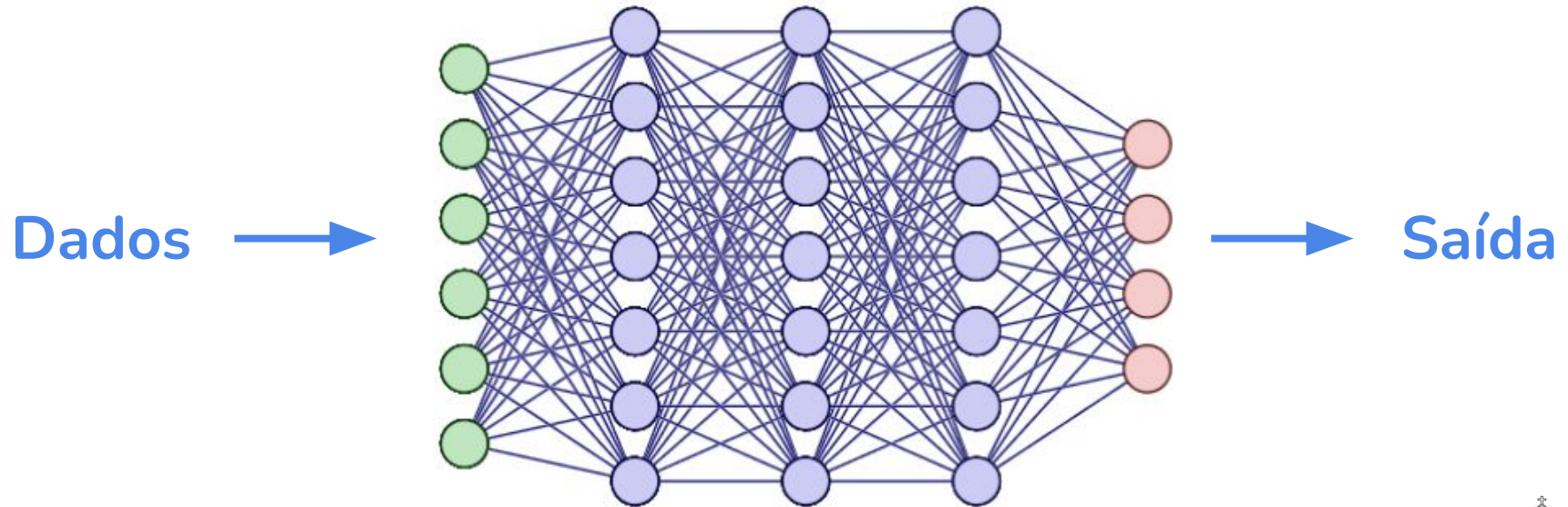


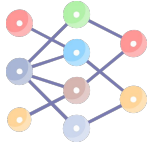
Nomenclatura em uma Rede Neural



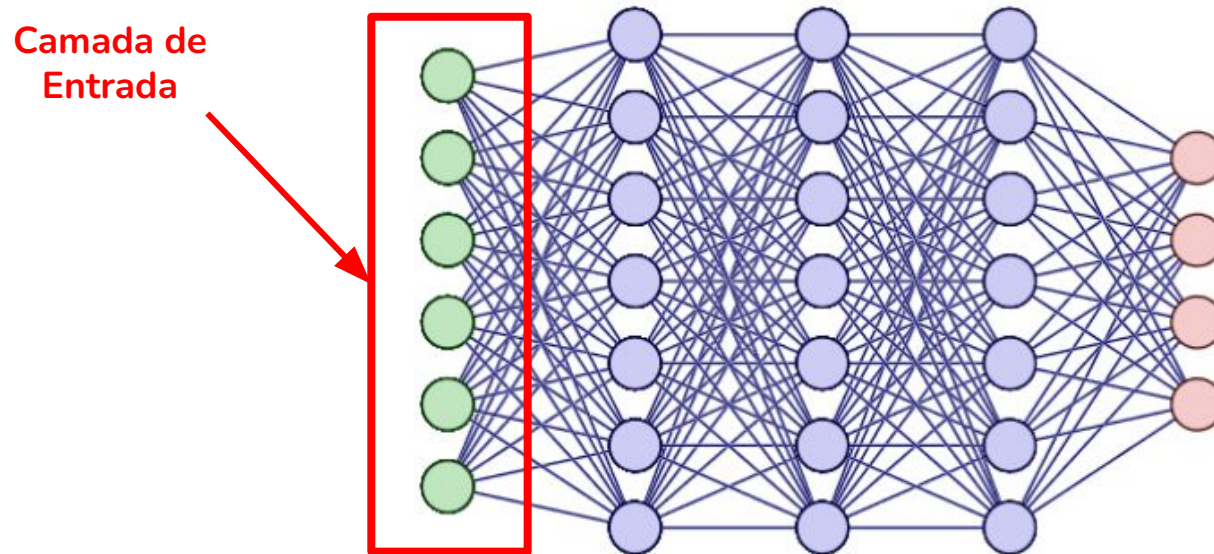


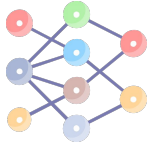
Nomenclatura em uma Rede Neural



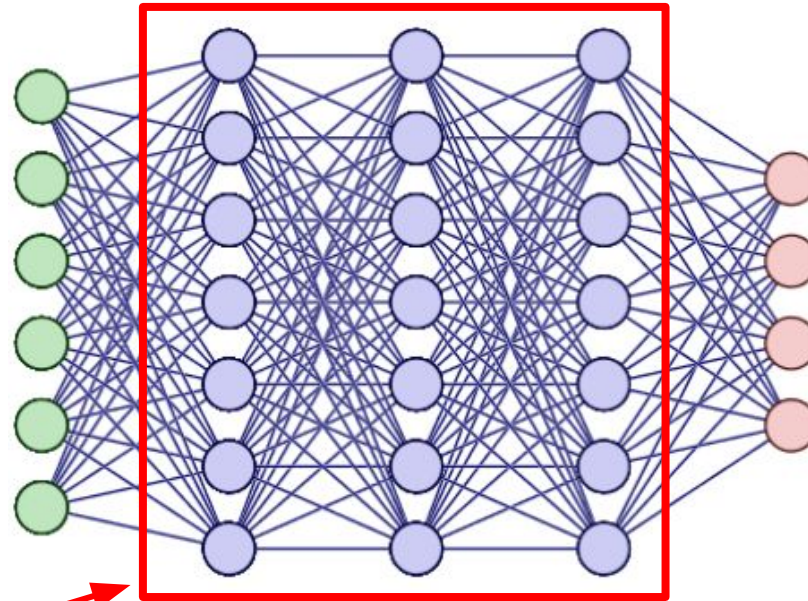


Nomenclatura em uma Rede Neural

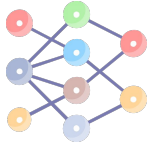




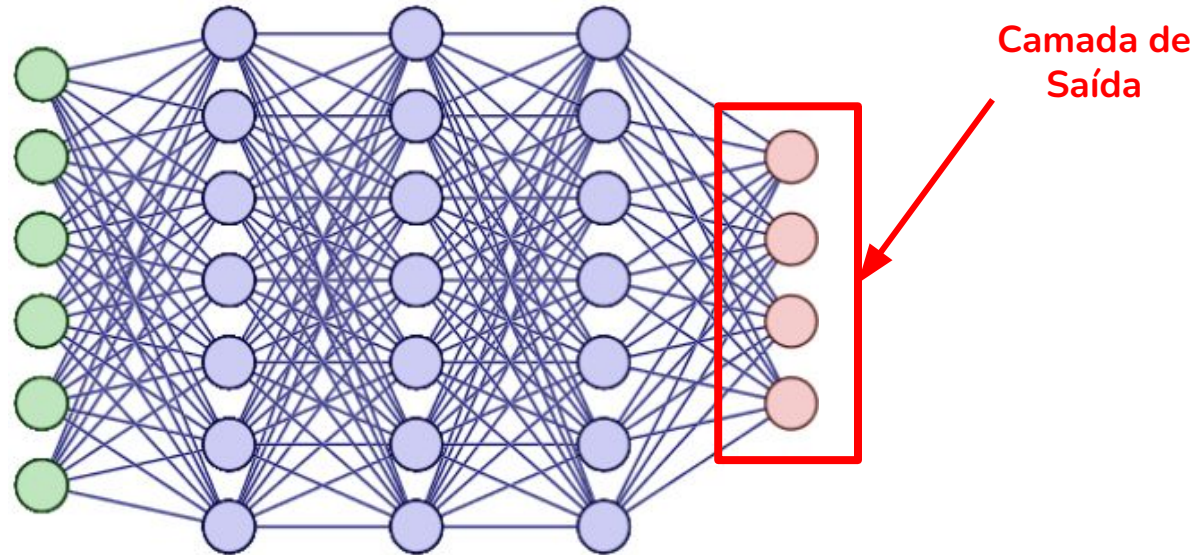
Nomenclatura em uma Rede Neural

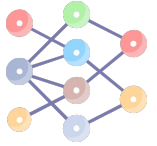


Camadas
Ocultas

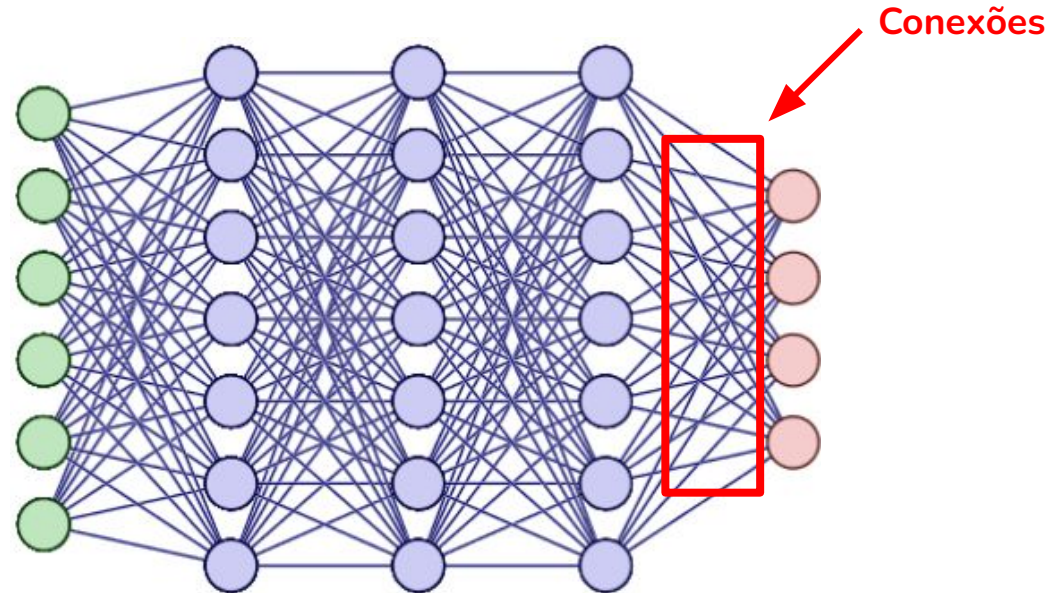


Nomenclatura em uma Rede Neural





Nomenclatura em uma Rede Neural



Introdução ao Keras

