
Implementação de biblioteca com uso de orientação a objetos

Eduardo Santos de Oliveira Marques

Engenharia de Software

Mestrado em Modelagem Computacional

Sumário



- Orientação a Objetos
- TensorFlow
 - Diagrama de classes em UML
 - Classes
 - Herança
 - Associação
 - Encapsulamento
 - Polimorfismo
 - Sobrecarga
 - Abstração
 - Módulos
- Implementação



TensorFlow

Orientação a Objetos



A programação orientada a objetos (POO) é um paradigma de programação que organiza o código em torno de “objetos”, podendo ser vistos como entidades que possuem dados (atributos) e comportamentos (métodos). Alguns aspectos importantes são:

- Diagrama de classes em UML;
- Classes;
- Herança;
- Associação;
- Encapsulamento;
- Polimorfismo;
- Sobrecarga;
- Abstração (classes abstratas/interface);
- Módulos.



TensorFlow



O TensorFlow é uma biblioteca de software de código aberto desenvolvida pelo Google para Machine Learning (ML) e computação numérica. Ele é amplamente utilizado para criar e treinar modelos de aprendizado de máquina, especialmente modelos de redes neurais artificiais.

Fornecer uma estrutura flexível para construir e treinar modelos de ML em uma ampla variedade de tarefas, como classificação de imagens, reconhecimento de fala, processamento de linguagem natural e muito mais. Ele oferece uma abstração de nível mais alto do que outras bibliotecas, permitindo aos desenvolvedores criar modelos complexos de forma mais fácil e eficiente.

É usado em diversos projetos e aplicações, desde a pesquisa acadêmica até o desenvolvimento de soluções de aprendizado de máquina em grande escala. A sua popularidade se deve (em partes) por ser uma biblioteca poderosa, flexível e bem documentada, oferecendo uma ampla gama de recursos para a implementação de modelos de ML com eficiência e facilidade.

Diagrama de classes em UML



Orientação a Objetos

O diagrama de classes em UML (Unified Modeling Language) é uma representação visual das classes, seus relacionamentos e da sua estrutura em um sistema orientado a objetos.

Ele mostra as classes, seus atributos e métodos, além dos relacionamentos entre as classes, como herança, associação, agregação, entre outros.

É uma ferramenta poderosa para projetar e documentar a estrutura de um sistema orientado a objetos.

TensorFlow

Embora não exista um diagrama de classes específico para a biblioteca TensorFlow em C++, é possível visualizar a estrutura de classes e seu relacionamento usando ferramentas de visualização de código.

A biblioteca TensorFlow possui uma hierarquia de classes bem definida para representar tensores, operações, modelos e muito mais.



Classes



Orientação a Objetos

Estrutura fundamental na POO. É uma representação abstrata de um objeto do mundo real, definindo os atributos (variáveis) e métodos (comportamentos) que esse objeto pode ter.

Elas são usadas como modelos para criar objetos específicos durante a execução de um programa.



TensorFlow

Utiliza classes para representar diversos conceitos, como tensores (*Tensor*), operações (*Operation*), grafos (*Graph*), sessões (*Session*), modelos (*Model*), entre outros.

Cada classe tem atributos e métodos específicos que permitem a criação, manipulação e execução de modelos de Machine Learning.

Herança



Orientação a Objetos

Mecanismo que permite que uma classe herde características (atributos e métodos) de outra classe.

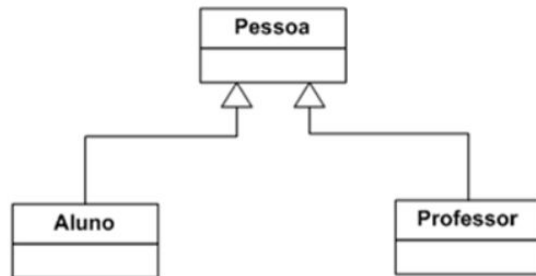
A classe herdada é chamada de classe derivada ou subclasse, e a classe da qual ela herda é chamada de classe base ou superclasse.

A herança permite reutilizar código e estabelecer uma hierarquia de classes, onde as subclasses podem adicionar ou modificar comportamentos herdados da superclasse.

TensorFlow

A herança é utilizada no TensorFlow para estender as funcionalidades de classes existentes.

Por exemplo, a classe *TensorFlow* oferece recursos básicos para manipular tensores, e as classes derivadas, tais como *TensorFlow::FloatTensor* e *TensorFlow::IntTensor*, podem ser criadas para tratar de tipos específicos de tensores.



Herança: Aluno e Professor herdam de Pessoa

Associação



Orientação a Objetos

Relacionamento entre classes indicando a relação entre elas de alguma forma. Essa relação pode ser representada por meio de atributos ou métodos das classes envolvidas.

Por exemplo, uma classe *Cliente* pode estar associada a uma classe *Pedido* por meio de um atributo *pedidos*, indicando que um cliente pode ter vários.



TensorFlow

A associação é aplicada no TensorFlow por meio do uso de atributos que representam relacionamentos entre classes.

Por exemplo, a classe *Graph* possui atributos que armazenam as operações que compõem o grafo.

Essas associações permitem construir e organizar estruturas complexas de modelos de ML.

Encapsulamento



Orientação a Objetos

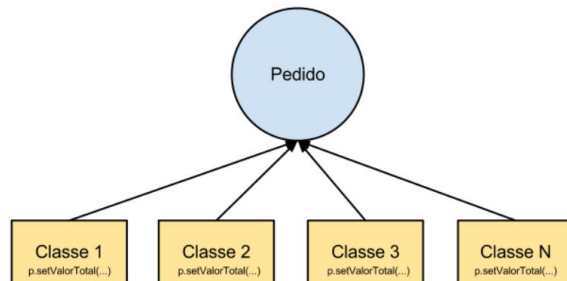
Um princípio da POO que envolve a combinação de dados e comportamentos relacionados em uma única unidade chamada classe.

Ele visa proteger os dados internos de uma classe, permitindo o acesso a eles somente por meio de métodos definidos por ela própria.

O encapsulamento fornece controle sobre o acesso aos dados e promove a segurança e a integridade dos objetos.

TensorFlow

O encapsulamento é uma base fundamental da orientação a objetos aplicado no TensorFlow. Os atributos e métodos das classes são definidos com modificadores de acesso, como público, privado e protegido, para controlar o acesso aos dados internos das classes. Isso garante que a manipulação dos objetos seja feita de maneira segura e controlada.



a mudança deverá ser feita em todas elas!

Polimorfismo



Orientação a Objetos

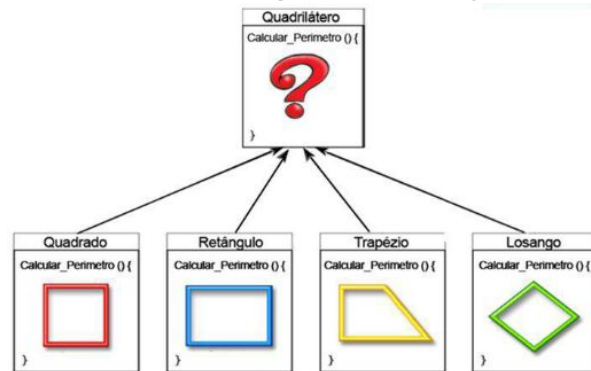
Capacidade de objetos de diferentes classes responderem de forma distinta a uma mesma mensagem.

Isso significa que um método pode ter várias implementações diferentes, dependendo do tipo do objeto que o está executando.

O polimorfismo permite tratar objetos de diferentes classes de forma uniforme, aumentando a flexibilidade e a reutilização de código.

TensorFlow

O polimorfismo é aplicado por meio do uso de classes base e classes derivadas. As classes derivadas podem substituir métodos herdados da classe base e fornecer implementações personalizadas. Isso permite tratar objetos com diferentes tipos de maneira uniforme, melhorando a flexibilidade e a reutilização de código.



Sobrecarga



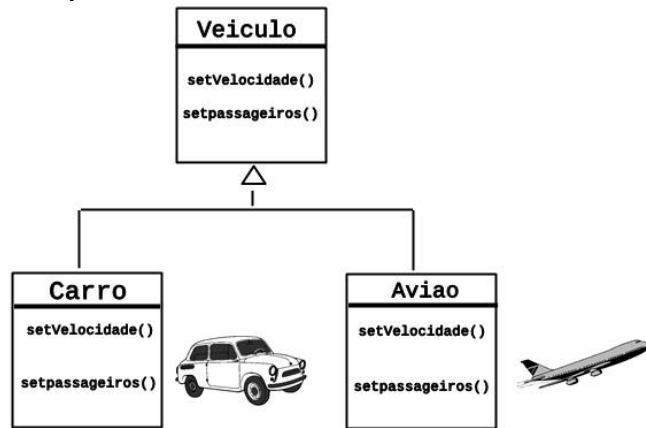
Orientação a Objetos

Ocorre quando uma classe tem vários métodos com o mesmo nome, mas com parâmetros diferentes. Isso permite que um método seja executado de maneira diferente, dependendo dos argumentos passados para ele.

A sobrecarga de métodos permite criar interfaces mais intuitivas e flexíveis, adaptando-se automaticamente aos diferentes tipos de dados ou situações.

TensorFlow

É aplicada para fornecer interfaces flexíveis. Por exemplo, a classe *TensorFlow::Session* oferece vários métodos com o mesmo nome, mas com assinaturas diferentes, permitindo executar operações de diferentes maneiras, dependendo dos argumentos passados.



Abstração



Orientação a Objetos

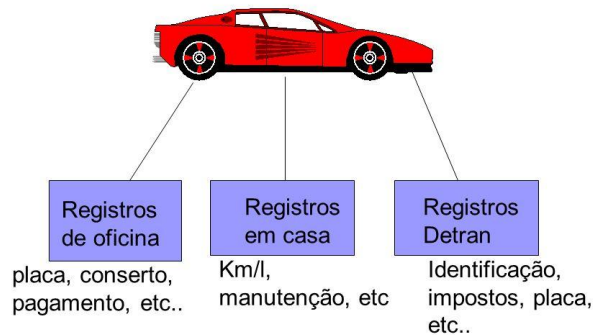
Conceito que permite modelar objetos complexos do mundo real de forma simplificada. Classes abstratas e interfaces são elementos usados para criar abstrações na programação orientada a objetos.

Uma classe abstrata não pode ser instanciada, servindo como modelo para outras classes. Ele define métodos que devem ser implementados pelas subclasses. Uma interface é uma coleção de métodos abstratos que definem um contrato para uma classe implementar. Ambos os conceitos permitem criar estruturas genéricas que podem ser especializadas por subclasses.

TensorFlow

A biblioteca utiliza classes abstratas e interfaces para fornecer abstrações e contratos para implementações específicas.

Por exemplo, a classe *TensorFlow::Operation* é uma classe abstrata que define métodos que devem ser implementados por classes derivadas para representar diferentes tipos de operações.



Módulos



Orientação a Objetos

Em geral, referem-se a unidades de código independentes que podem ser usadas para organizar e modularizar um sistema.

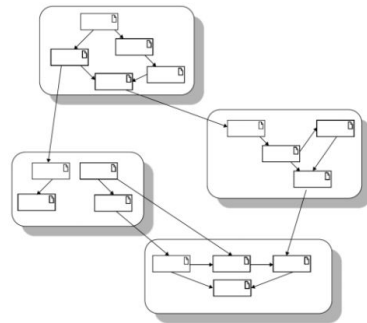
Na orientação a objetos, os módulos podem ser entendidos como pacotes, bibliotecas ou até mesmo classes que agrupam funcionalidades relacionadas.

Eles ajudam a dividir o código em partes menores e mais gerenciáveis, permitindo a reutilização e a manutenção mais fácil do sistema como um todo.

TensorFlow

Na biblioteca eles são utilizados para agrupar funcionalidades relacionadas. Por exemplo, existem módulos específicos para manipulação de tensores, construção de grafos, treinamento de modelos, entre outros.

Isso ajuda a organizar e modularizar o código, permitindo uma melhor reutilização e manutenção.



Implementação



```
1 #include <iostream>
2 #include <vector>
3
4 // Classe base para representar um Tensor
5 class Tensor {
6 public:
7     virtual void print() const = 0;
8 };
9
10 // Classe para representar um Tensor Float
11 class FloatTensor : public Tensor {
12 private:
13     std::vector<float> data;
14
15 public:
16     FloatTensor(const std::vector<float>& input) : data(input) {}
17
18     void print() const override {
19         for (const auto& value : data) {
20             std::cout << value << " ";
21         }
22         std::cout << std::endl;
23     }
24 };
25
26 // Classe para representar um Tensor Inteiro
27 class IntTensor : public Tensor {
28 private:
29     std::vector<int> data;
30
31 public:
32     IntTensor(const std::vector<int>& input) : data(input) {}
33
34     void print() const override {
35         for (const auto& value : data) {
36             std::cout << value << " ";
37         }
38         std::cout << std::endl;
39     }
40 };
```

```
41
42 // Classe para representar um Modelo
43 class Model {
44 public:
45     virtual void train() = 0;
46     virtual void predict() = 0;
47 };
48
49 // Classe para representar um Modelo Linear
50 class LinearModel : public Model {
51 public:
52     void train() override {
53         std::cout << "Treinando o modelo linear" << std::endl;
54     }
55
56     void predict() override {
57         std::cout << "Realizando previsões com o modelo linear" << std::endl;
58     }
59 };
60
61 int main() {
62     std::vector<float> floatData = {1.0, 2.0, 3.0, 4.0, 5.0};
63     std::vector<int> intData = {1, 2, 3, 4, 5};
64
65     FloatTensor floatTensor(floatData);
66     IntTensor intTensor(intData);
67
68     floatTensor.print();
69     intTensor.print();
70
71     LinearModel linearModel;
72     linearModel.train();
73     linearModel.predict();
74
75     return 0;
76 }
```

Implementação



```
1 #include <iostream>
2 #include <vector>
3
4 // Classe base para representar um Tensor
5 class Tensor {
6 public:
7     virtual void print() const = 0;
8 };
9
10 // Classe para representar um Tensor Float
11 class FloatTensor : public Tensor {
12 private:
13     std::vector<float> data;
14
15 public:
16     FloatTensor(const std::vector<float>& input) : data(input) {}
17
18     void print() const override {
19         for (const auto& value : data) {
20             std::cout << value << " ";
21         }
22         std::cout << std::endl;
23     }
24 };
25
26 // Classe para representar um Tensor Inteiro
27 class IntTensor : public Tensor {
28 private:
29     std::vector<int> data;
30
31 public:
32     IntTensor(const std::vector<int>& input) : data(input) {}
33
34     void print() const override {
35         for (const auto& value : data) {
36             std::cout << value << " ";
37         }
38         std::cout << std::endl;
39     }
40 };
```

Nesta implementação, as classes foram utilizadas para representar diferentes conceitos da biblioteca TensorFlow.

A classe *Tensor* é uma classe abstrata que serve como base para representar diferentes tipos de tensores, como *FloatTensor* e *IntTensor*.

Cada uma dessas classes concretas herda de *Tensor* e implementa o método *print()* para exibir os valores dos tensores.



Implementação



Além disso, têm-se a classe *Model*, também abstrata, que representa um modelo na biblioteca TensorFlow. A classe *LinearModel* é uma classe concreta que herda de *Model* e implementa os métodos *train()* e *predict()* para treinar e realizar previsões com um modelo linear.

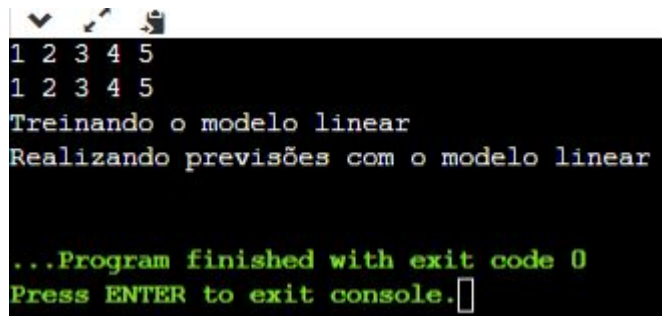
Ao executar o programa, são criados tensores de float e inteiros e são exibidos seus valores através do método *print()*. Em seguida, um modelo linear é instanciado e são chamados os métodos *train()* e *predict()* para demonstrar o funcionamento da classe *LinearModel*.

```
41 // Classe para representar um Modelo
42 class Model {
43 public:
44     virtual void train() = 0;
45     virtual void predict() = 0;
46 };
47
48 // Classe para representar um Modelo Linear
49 class LinearModel : public Model {
50 public:
51     void train() override {
52         std::cout << "Treinando o modelo linear" << std::endl;
53     }
54
55     void predict() override {
56         std::cout << "Realizando previsões com o modelo linear" << std::endl;
57     }
58 };
59
60
61 int main() {
62     std::vector<float> floatData = {1.0, 2.0, 3.0, 4.0, 5.0};
63     std::vector<int> intData = {1, 2, 3, 4, 5};
64
65     FloatTensor floatTensor(floatData);
66     IntTensor intTensor(intData);
67
68     floatTensor.print();
69     intTensor.print();
70
71     LinearModel linearModel;
72     linearModel.train();
73     linearModel.predict();
74
75     return 0;
76 }
```


Implementação



Resultados



```
1 2 3 4 5
1 2 3 4 5
Treinando o modelo linear
Realizando previsões com o modelo linear

...Program finished with exit code 0
Press ENTER to exit console.
```

O código foi executado na plataforma *GDB Online*, para acessar, basta acessar o link abaixo.

Link: <https://onlinegdb.com/ldpQMz2bk>

Neste exemplo, foram utilizados os conceitos de classes, herança, polimorfismo (através dos métodos virtuais), abstração (classes abstratas), encapsulamento (através de modificadores de acesso) e módulos (divisão do código em diferentes classes).

OBS: Não foram apresentados os resultados do modelo de Machine Learning, apenas o seu funcionamento.

É importante ressaltar que é uma implementação simplificada, não abrangendo todos os recursos e funcionalidades da biblioteca TensorFlow. Ela serve apenas para ilustrar alguns conceitos de orientação a objetos em C++.

Referências



1. <https://www.tensorflow.org/?hl=pt-br>
2. https://supygirls.readthedocs.io/en/latest/intro_comp/PythonOO.html
3. <https://www.devmedia.com.br/orientacoes-basicas-na-elaboracao-de-um-diagrama-de-classes/37224>
4. <https://programadoresdepre.com.br/o-que-e-programacao-orientada-a-objetos-poo/>
5. <https://www.usandoaccess.com.br/tutoriais/classe-no-access-orientacao-a-objetos.asp>
6. https://www.macoratti.net/18/02/c_objrela1.htm
7. <https://www.alura.com.br/artigos/revisitando-a-orientacao-a-objetos-encapsulamento-no-java>
8. <https://slideplayer.com.br/slide/1705805/>
9. <https://www.devmedia.com.br/sobrecarga-e-sobreposicao-de-metodos-em-orientacao-a-objetos/33066>
10. <https://slideplayer.com.br/slide/46561/>
11. <http://tieconcursos.blogspot.com/2015/03/programacao-modular-programacao.html>
12. https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_orientada_a_objetos
13. <https://en.wikipedia.org/wiki/C%2B%2B>



PPG em
Modelagem Computacional

| ufjf

Obrigado pela atenção!

