Trabalho Final - Orientação a Objetos

Implementação de biblioteca com uso de aspectos da POO

Engenharia de Software - 2023.1

Programa de Pós-Graduação em Modelagem Computacional

Aluno: **Eduardo Santos de Oliveira Marques**

Professores: André Luiz de Oliveira
Bernardo Martins Rocha

Sumário

TensorFlow

OBJETIVO

Objetivo

Realizar uma implementação própria da biblioteca TensorFlow utilizando a linguagem C++ com Orientação a Objetos, sendo discutidos aspectos de orientação a objetos, tais como:

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

Ou seja, se deve realizar a implementação de ao menos 1 destes conceitos. Não é necessário implementar as diversas funcionalidades reais do sistema. Para facilitar o entendimento, sugere-se a impressão de mensagens.

Descrição (recordação)

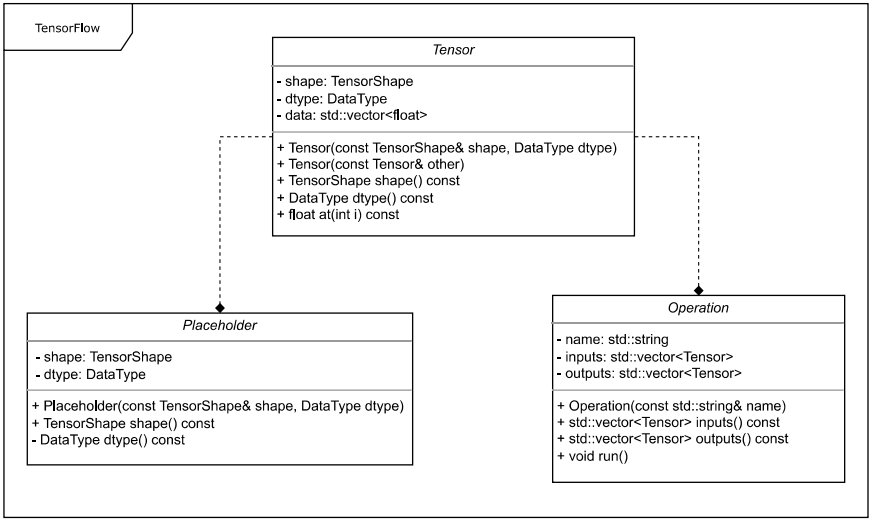
- **Classes:** As classes são a unidade básica de organização do código orientado a objetos. Elas permitem agrupar atributos e métodos relacionados em um único lugar;
- **Herança:** A herança permite que uma classe herde os atributos e métodos de outra classe. Isso pode ser usado para reutilizar código e criar classes mais genéricas;
- **Associação:** A associação permite que uma classe tenha uma referência a outra classe. Isso pode ser usado para representar relacionamentos entre objetos;
- **Encapsulamento:** O encapsulamento permite que os atributos de uma classe sejam privados, o que garante que eles só possam ser acessados por métodos da própria classe. Isso ajuda a proteger o estado da classe e torna o código mais robusto;

Descrição (recordação)

- **Polimorfismo:** O polimorfismo permite que objetos de diferentes classes possam ser tratados de forma semelhante. Isso pode ser usado para escrever código mais genérico e flexível;
- **Sobrecarga:** A sobrecarga permite que uma classe tenha métodos com o mesmo nome, mas com assinaturas diferentes. Isso pode ser usado para fornecer métodos com funcionalidades diferentes;
- **Abstração (classes abstratas/interface):** As classes abstratas e interfaces são usadas para definir a interface de uma classe ou objeto. Isso pode ser usado para restringir a implementação de classes ou objetos e tornar o código mais reutilizável;
- **Módulos:** Os módulos são usados para organizar o código em unidades menores e mais gerenciáveis. Isso pode ajudar a melhorar a legibilidade e a manutenção do código.

IMPLEMENTAÇÃO

Diagrama de Classes



Código desenvolvido I

```cpp
#include <iostream>
#include <vector>

class Tensor {
public:
    Tensor(int shape) : shape_(shape) {}

    virtual void print() const {
        std::cout << "Tensor de forma " << shape_ << std::endl;
    }

    virtual void operacao() {
        std::cout << "Operação em Tensor base" << std::endl;
    }

    int getShape() const {
        return shape_;
    }

```

Código desenvolvido II

```
20 private:
21     int shape_;
22 };
23
24 class TensorNumerico : public Tensor {
25 public:
26     TensorNumerico(int shape, std::vector<double> data) : Tensor(shape), data_(data) {}
27
28     void print() const override {
29         std::cout << "Tensor Numérico de forma " << getShape() << std::endl;
30     }
31
32     void operacao() override {
33         std::cout << "Operação em Tensor Numérico" << std::endl;
34     }
35
36 private:
37     std::vector<double> data_;
38 };
```

Código desenvolvido III

```
39 int main() {
40     Tensor tensor(3);
41     tensor.print();
42     tensor.operacao();
43
44     std::vector<double> data = {1.0, 2.0, 3.0};
45     TensorNumerico tensorNumerico(3, data);
46     tensorNumerico.print();
47     tensorNumerico.operacao();
48
49     return 0;
50 }
```

Outputs:

```
Tensor de forma 3
Operação em Tensor base
Tensor Numérico de forma 3
Operação em Tensor Numérico
```

Código desenvolvido

Aspectos

Neste código:

1. Têm-se uma classe base *'Tensor'* que representa um tensor genérico;
2. A classe derivada *'TensorNumerico'* representa um tensor numérico;
   - E estende a classe base *'Tensor'*;
3. A herança e o polimorfismo foram demonstrados;
   - Pois a função *'print'* é sobrescrita na classe derivada;
4. O encapsulamento foi mostrado ao manter os dados privados na classe *'TensorNumerico'*;
5. O conceito de abstração é ilustrado pelo método virtual *'operacao()'* na classe base;
   - Sendo implementado nas subclasses.

Código externo I

```
1  class Operações : public Figura {
2    public:
3      Operações(const std::string& nome, const std::function<void(Tensor* t)>& implementaç
       ão)
4          : Figura(nome) {
5          this->implementação = implementação;
6      }
7
8      void executar(Tensor* t) override {
9          implementação(t);
10     }
11
12   private:
13     std::function<void(Tensor* t)> implementação;
14 };
15
16
17
18
```

Código externo II

```
19 // EXEMPLO DE USO
20
21 #include "tensor.h"
22
23 int main() {
24   // Cria um tensor de 2 dimensões com tamanho 2 x 3.
25   Tensor t(std::vector<int>{2, 3});
26
27   // Adiciona uma operação de soma a cada dimensão do tensor.
28   t.adicionarOperação("Soma", [](Tensor* t) {
29     for (int i = 0; i < t->tamanho[0]; i++) {
30       for (int j = 0; j < t->tamanho[1]; j++) {
31         t->dados[i * t->tamanho[1] + j] += 1;
32       }
33     }
34   });
35
36
37
```

Código externo III

```
38    // Imprime o tensor.
39    for (int i = 0; i < t.tamanho[0]; i++) {
40      for (int j = 0; j < t.tamanho[1]; j++) {
41        std::cout << t.dados[i * t.tamanho[1] + j] << " ";
42      }
43      std::cout << std::endl;
44    }
45
46    return 0;
47  }
```

Outputs:

```
2 3
4 5
```

Código externo

Conceitos

- **Associação:** A classe *'Tensor'* associa uma operação a cada uma de suas dimensões. A associação é feita por meio de um mapeamento entre os nomes das dimensões e os ponteiros para as operações;
- **Encapsulamento:** Os atributos das classes *'Tensor'* e *'Operações'* são privados, o que garante que eles só possam ser acessados por métodos das próprias classes;
- **Polimorfismo:** A classe *'Operações'* é uma classe abstrata, o que significa que ela só pode ser usada como base para outras classes. Isso garante que apenas operações válidas possam ser adicionadas a um tensor;

Código externo

Conceitos

- **Sobrecarga:** O operador de atribuição é sobrecarregado para permitir que um tensor seja atribuído a outro tensor. Isso facilita a cópia de tensores;
- **Abstração:** A classe *'Operações'* é uma classe abstrata, o que significa que ela fornece uma interface para operações matemáticas. Essa interface pode ser usada para criar diferentes tipos de operações, sem que seja necessário modificar o código da classe *'Tensor'*;
- **Módulos:** O código do módulo está organizado em um único arquivo, chamado *'tensor.cpp'*. O arquivo *'tensor.h'* contém as declarações das classes e das funções;
- **Herança:** A classe *'Operações'* herda da classe *'Figura'*.

Código externo

Classes - Características

- *'Tensor'*: Representa um tensor (estrutura de dados multidimens. para armazenar número);
  - Tamanho: Um vetor que armazena o tamanho de cada dimensão do tensor;
  - Dados: Um ponteiro para os dados do tensor.
  - O construtor da classe *'Tensor'* recebe o tamanho e os dados do tensor como argumentos.
- *'Operações'*: Representa uma operação matemática que pode ser aplicada a um tensor:
  - Nome: O nome da operação;
  - Implementação: Um ponteiro para a função que implementa a operação.
  - O construtor da classe *'Operações'* recebe o nome e a implem. da operação como argumentos.

Aplicação em Python (simples) I

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Aplicação em Python (simples)

Observações

- ***'mnist'*:** Imagens de dígitos manuscritos são carregados usando a biblioteca *TensorFlow/Keras*. A classe *'mnist'* é um objeto que permite carregar e acessar os dados;
- **Dados:** Os dados são divididos em conjuntos de treinamento e teste, onde *'x_train'* e *'x_test'* são as imagens e *'y_train'* e *'y_test'* são os rótulos correspondentes. Operações de normalização também são aplicadas aos dados;
- **Modelo:** Um modelo sequencial é criado usando *'tf.keras.models.Sequential'*. O modelo é construído como uma sequência de camadas. Cada camada é um objeto que representa uma operação específica na rede neural. As camadas são definidas de forma sequencial;

Aplicação em Python (simples)

Observações

- ***'compile'*:** O modelo é compilado, envolvendo a configuração de parâmetros de treinamento, como o otimizador (Adam), a função de perda (entropia cruzada esparsa categórica) e as métricas (precisão) que serão usadas para avaliar o desempenho do modelo;

- ***'fit'*:** O modelo é treinado usando o método *'fit'*. Durante o treinamento, os dados de treinamento (*'x_train'* e *'y_train'*) são usados para ajustar os pesos da rede neural com base nas configurações fornecidas. O treinamento ocorre ao longo de várias épocas;

- ***'evaluate'*:** O modelo é avaliado usando o método *'evaluate'* com os dados de teste (*'x_test'* e *'y_test'*). Isso calcula a perda e a precisão do modelo nos dados de teste.

Aplicação em Python (avançado) I

```
class MyModel(tf.keras.Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = Conv2D(32, 3, activation='relu')
    self.flatten = Flatten()
    self.d1 = Dense(128, activation='relu')
    self.d2 = Dense(10, activation='softmax')

  def call(self, x):
    x = self.conv1(x)
    x = self.flatten(x)
    x = self.d1(x)
    return self.d2(x)
model = MyModel()

with tf.GradientTape() as tape:
  logits = model(images)
  loss_value = loss(logits, labels)
grads = tape.gradient(loss_value, model.trainable_variables)
optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

Aplicação em Python (avançado)

Observações

- ***'MyModel'*:** Uma nova classe chamada *'MyModel'* é definida, que é uma subclasse da classe *'tf.keras.Model'*. Isso significa que *'MyModel'* herda funcionalidades e comportamentos de *'tf.keras.Model'*;

- ***'__init__'*:** No método *'__init__'*, o construtor da classe é definido. Nele, são criadas as camadas da rede neural como atributos da instância *'MyModel'*. As camadas incluem uma camada de convolução (*'self.conv1'*), uma camada de achatamento (*'self.flatten'*), e duas camadas densas (*'self.d1'* e *'self.d2'*). Essas camadas definem a arquitetura da rede;

- ***'call'*:** O método *'call'* é responsável por definir como os dados fluem através da rede. Ele recebe um tensor *'x'* como entrada e aplica as camadas sequencialmente, conectando-as para produzir a saída do modelo. No final, a saída é retornada;

Aplicação em Python (avançado)

Observações

- **Instanciação:** Uma instância do modelo personalizado *'MyModel'* é criada. Isso instancia o modelo com todas as camadas definidas no método *'__init__'*;

- **Treinamento:** Nesta parte do código, o modelo é treinado. O *'TensorFlow GradientTape'* é usado para calcular gradientes. O modelo é chamado com *'images'* para obter as previsões (*'logits'*). A perda é calculada comparando as previsões com os rótulos (*'labels'*). Em seguida, os gradientes em relação aos parâmetros treináveis são calculados usando *'tape.gradient'*. Finalmente, o otimizador é usado para aplicar as atualizações de gradiente nos parâmetros do modelo.

CONCLUSÃO

Conclusões

Esta implementação demonstra como os conceitos de orientação a objetos podem ser usados para criar uma biblioteca de operações básicas de tensores em C++. A biblioteca é simples e fácil de usar, e pode ser usada como base para a criação de aplicações mais complexas de aprendizado de máquina.

É importante notar que são exemplos simples, não se tratando de uma implementação completa do TensorFlow. É uma demonstração básica de como os conceitos de orientação a objetos podem ser aplicados em uma implementação de tensor em C++.

Os exemplos em Python mostram a flexibilidade da biblioteca, permitindo a definição de modelos personalizados e no controle sobre o processo de treinamento.

Referências I

[1] TENSORFLOW. **Criar modelos de machine learning no nível de produção com o TensorFlow**. 2023. Último acesso: 09/08/2023. Disponível em: <https://www.tensorflow.org/?hl=pt-br>.

[2] DEPRê, P. **O que é programação orientada a objetos? (POO)**. 2021. Último acesso: 09/08/2023. Disponível em: <https://programadoresdepre.com.br/o-que-e-programacao-orientada-a-objetos-poo/>.

[3] DEVMEDIA. **Orientações básicas na elaboração de um diagrama de classes**. 2023. Último acesso: 09/08/2023. Disponível em: <https://www.devmedia.com.br/orientacoes-basicas-na-elaboracao-de-um-diagrama-de-classes/37224>.

[4] ACCESS, U. **Utilizando Classe no Access – Orientação a Objetos**. 2010. Último acesso: 09/08/2023. Disponível em: <https://www.usandoaccess.com.br/tutoriais/classe-no-access-orientacao-a-objetos.asp>.

Referências II

[5] MACORATTI.NET. **C# - Entendendo o relacionamento entre objetos (Herança/Composição)**. 2010. Último acesso: 09/08/2023. Disponível em: <https://www.macoratti.net/18/02/c_objrela1.htm>.

[6] ALURA. **Revisitando a Orientação a Objetos: encapsulamento no Java**. 2012. Último acesso: 09/08/2023. Disponível em: <https://www.alura.com.br/artigos/revisitando-a-orientacao-a-objetos-encapsulamento-no-java>.

[7] DEVMEDIA. **Sobrecarga e sobreposição de métodos em orientação a objetos**. 2023. Último acesso: 09/08/2023. Disponível em: <https://www.devmedia.com.br/sobrecarga-e-sobreposicao-de-metodos-em-orientacao-a-objetos/33066>.

[8] CHAPA, M. **Programação Orientada à Objetos**. 2014. Último acesso: 09/08/2023. Disponível em: <https://slideplayer.com.br/slide/1705805/>.

[9] BRAGA, M. **Introdução a Programação Orientada a Objetos**. 2014. Último acesso: 09/08/2023. Disponível em: <https://slideplayer.com.br/slide/46561/>.

Referências III

[10] CONCURSOS, T. e. **[Programação Modular] Programação Orientada a Objetos**. 2015. Último acesso: 09/08/2023. Disponível em: <https://tieconcursos.blogspot.com/2015/03/programacao-modular-programacao.html>.

[11] WIKIPEDIA. **Programação orientada a objetos**. 2023. Último acesso: 09/08/2023. Disponível em: <https://pt.wikipedia.org/wiki/Programa

[12] DIAGRAMS. **Flowchart Maker and Online Diagram Software**. 2023. Último acesso: 09/08/2023. Disponível em: <https://app.diagrams.net/>.

[13] DEVMEDIA. **Técnicas e fundamentos de Testes de Software**. 2023. Último acesso: 15/08/2023. Disponível em: <https://www.devmedia.com.br/guia/tecnicas-e-fundamentos-de-testes-de-software/34403>.

[14] CONCEITO.DE. **Conceito de Diagrama de Blocos**. 2023. Último acesso: 15/08/2022. Disponível em: <https://conceito.de/diagrama-de-blocos>.

Referências IV

[15] ONLINEGDB. **Online compiler and debugger for c/c++**. 2019. Último acesso: 15/08/2022. Disponível em: <https://repositorio.ufersa.edu.br/bitstream/prefix/5981/1/T%C3%A1ssioFC_MONO.pdf>.

[16] TENSORFLOW. **Compreender a biblioteca C++**. 2023. Último acesso: 10/09/2023. Disponível em: <https://www.tensorflow.org/lite/microcontrollers/library?hl=pt-br>.

[17] TENSORFLOW. **TensorFlow C++ API Reference**. 2023. Último acesso: 10/09/2023. Disponível em: <https://www.tensorflow.org/api_docs/cc>.

[18] KSACHDEVA. **tensorflow-cc-examples**. 2018. Último acesso: 10/09/2023. Disponível em: <https://github.com/ksachdeva/tensorflow-cc-examples/tree/master>.

[19] ITNEXT. **Creating a TensorFlow DNN in C++ Part 1**. 2019. Último acesso: 10/09/2023. Disponível em: <https://itnext.io/creating-a-tensorflow-dnn-in-c-part-1-54ce69bbd586>.

Referências V

[20] RJPOWER. **tensorflow**. 2023. Último acesso: 10/09/2023. Disponível em: <https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/label_image/main.cc>.

[21] D0K. **tensorflow**. 2023. Último acesso: 10/09/2023. Disponível em: <https://github.com/tensorflow/tensorflow/tree/master/tensorflow>.

[22] SCIENCE, T. D. **Creating a TensorFlow CNN in C++ (Part 2)**. 2019. Último acesso: 10/09/2023. Disponível em: <https://towardsdatascience.com/creating-a-tensorflow-cnn-in-c-part-2-eea0de9dcada>.

[23] PROFHARISELDON. **C++ TensorFlow Build doesn't find TensorFlow .h files that exist. Also "Please update your includePath"**. 2020. Último acesso: 10/09/2023. Disponível em: <https://github.com/tensorflow/tensorflow/issues/41506>.

[24] RANGSIMANKETKAEW. **tensorflow-cpp-api**. 2022. Último acesso: 10/09/2023. Disponível em: <https://github.com/rangsimanketkaew/tensorflow-cpp-api>.

Referências VI

[25] CPPFLOW. **CppFlow - Easily run TensorFlow models from C++**. 2020. Último acesso: 10/09/2023. Disponível em: <https://serizba.github.io/cppflow/>.

[26] FIRESHIP. **TensorFlow in 100 Seconds**. 2022. Último acesso: 29/09/2023. Disponível em: <https://www.youtube.com/watch?v=i8NETqtGHms&ab_channel=Fireship>.

[27] TENSORFLOW. **TensorFlow in 100 Seconds**. 2023. Último acesso: 05/10/2023. Disponível em: <https://github.com/tensorflow>.

[28] AIM. **Should Developers Choose C++ Over Python for Machine Learning?** 2023. Último acesso: 05/10/2023. Disponível em: <https://analyticsindiamag.com/should-developers-choose-c-over-python-for-machine-learning/>.

[29] TECH, D. **O que é TensorFlow? Para que serve?** 2023. Último acesso: 05/10/2023. Disponível em: <https://didatica.tech/o-que-e-tensorflow-para-que-serve/>.

Referências VII

[30] TENSORFLOW. **TensorFLow - Exemplos**. 2023. Último acesso: 05/09/2023. Disponível em: <https://www.tensorflow.org/overview?hl=pt-br>.

[31] MEDIUM. **Conhecendo a visão do computador: Redes Neurais Convolucionais**. 2019. Último acesso: 05/10/2023. Disponível em: <https://vitorborbarodrigues.medium.com/conhecendo-a-vis%C3%A3o-do-computador-redes-neurais-convolucionais-e1c2b14bf426>.

[32] MORAES, R. G. R. Leonardo G. de. **Estudo de Técnicas de Aprendizado Baseado em Uma Única Classe para o Reconhecimento Facial**. 2018. Último acesso: 05/10/2023. Disponível em: <https://cptl.ufms.br/files/2020/12/TCC-LeonardoGM.pdf>.