人工智能原理与技术

习题课

TA：李睿哲

2025.03.19

主要内容

- 面向对象编程简介
- 实现一个神经网络
- 作业一发布

1 面向对象编程简介

- Python
- 从过程到对象
- 类(class)

数据类型：

- Python

  int，float，str

  list，tuple，set，dict

- 从过程到对象
- 类(class)

- Python
- 从过程到对象
- 类(class)

list:

```
x = [1, 2, 3, "python", [4, 5]]
print(x[0], x[-1])      # 访问元素
x.append(6)             # 追加元素
x.insert(2, "new")      # 插入元素
x.remove('python')      # 删除指定元素
print(x)
```

- Python
- 从过程到对象
- 类(class)

tuple：

```
x = (1, 2, 3, "python")
print(x[1])             # 访问元素
print(x[:3])            # 切片
x[0] = 100              # ❌ 会报错，元组不可修改
```

set:

- Python
- 从过程到对象
- 类(class)

```
s1 = {1, 2, 3, 3}
s2 = {3, 4, 5}

print(s1)              # 输出 {1, 2, 3}，去重
print(s1 | s2)         # 并集
print(s1 & s2)         # 交集
print(s1 - s2)         # 差集
```

- Python

dict:

```
x = {"name": "Alice", "age": 25}
print(x["name"])                # 访问值
x["age"] = 26                   # 修改值
x["city"] = "Beijing"           # 添加键值对
```

- 从过程到对象
- 类(class)

- Python
- 从过程到对象
- 类(class)

常见判断与迭代语句：

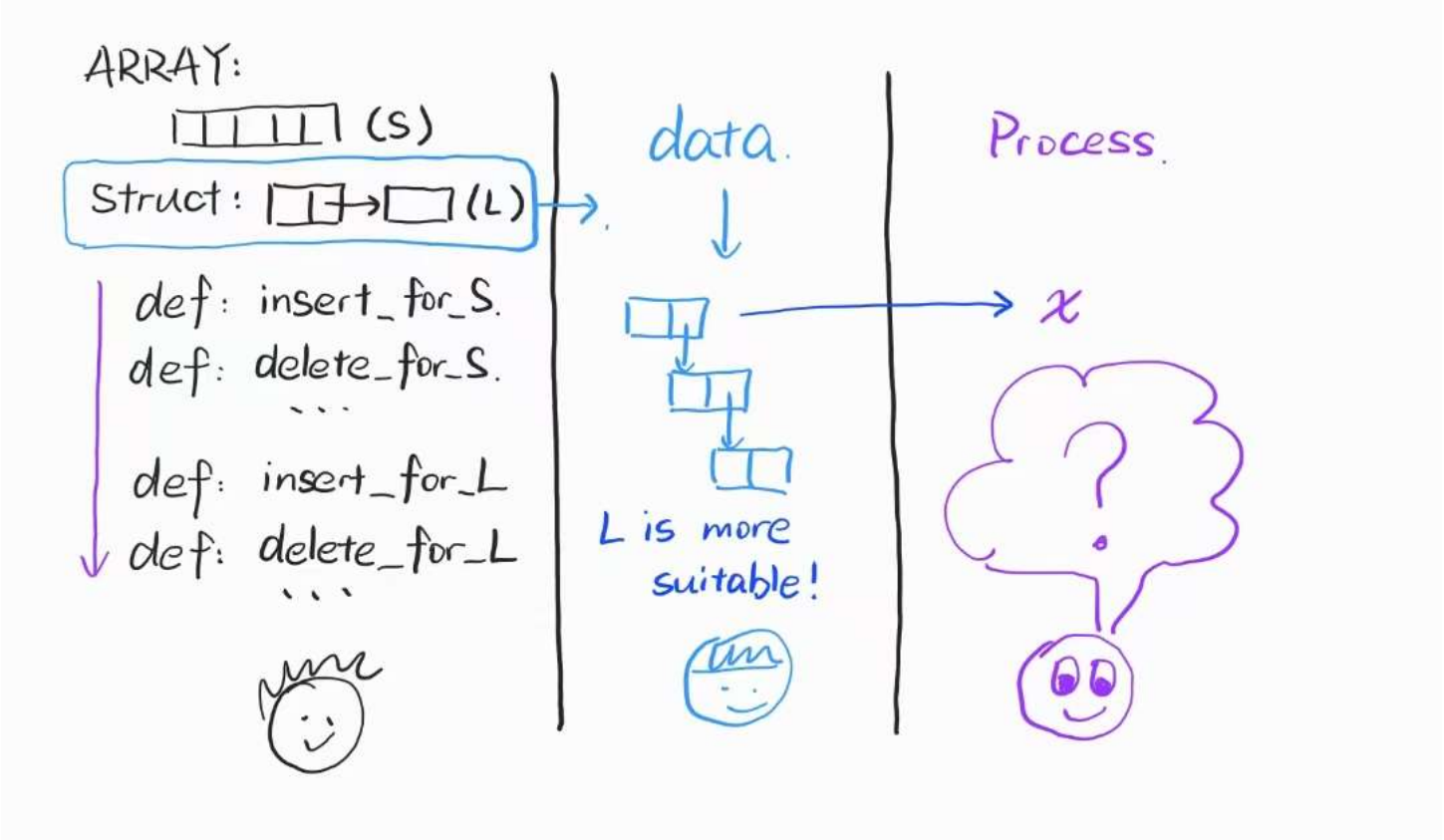
```
if x > 5 and y < 10:
if x > 5 or y > 10:
if not (x < 5):
if 3 in list_1:
if 5 not in list_1:

for i in range(5):
for c in str_1:
for x in list_1:
for key in dict_1:
for key, value in dict_1.items():

dict_1 = {x: x**2 for x in list_1 if x % 2 == 0}
...
```

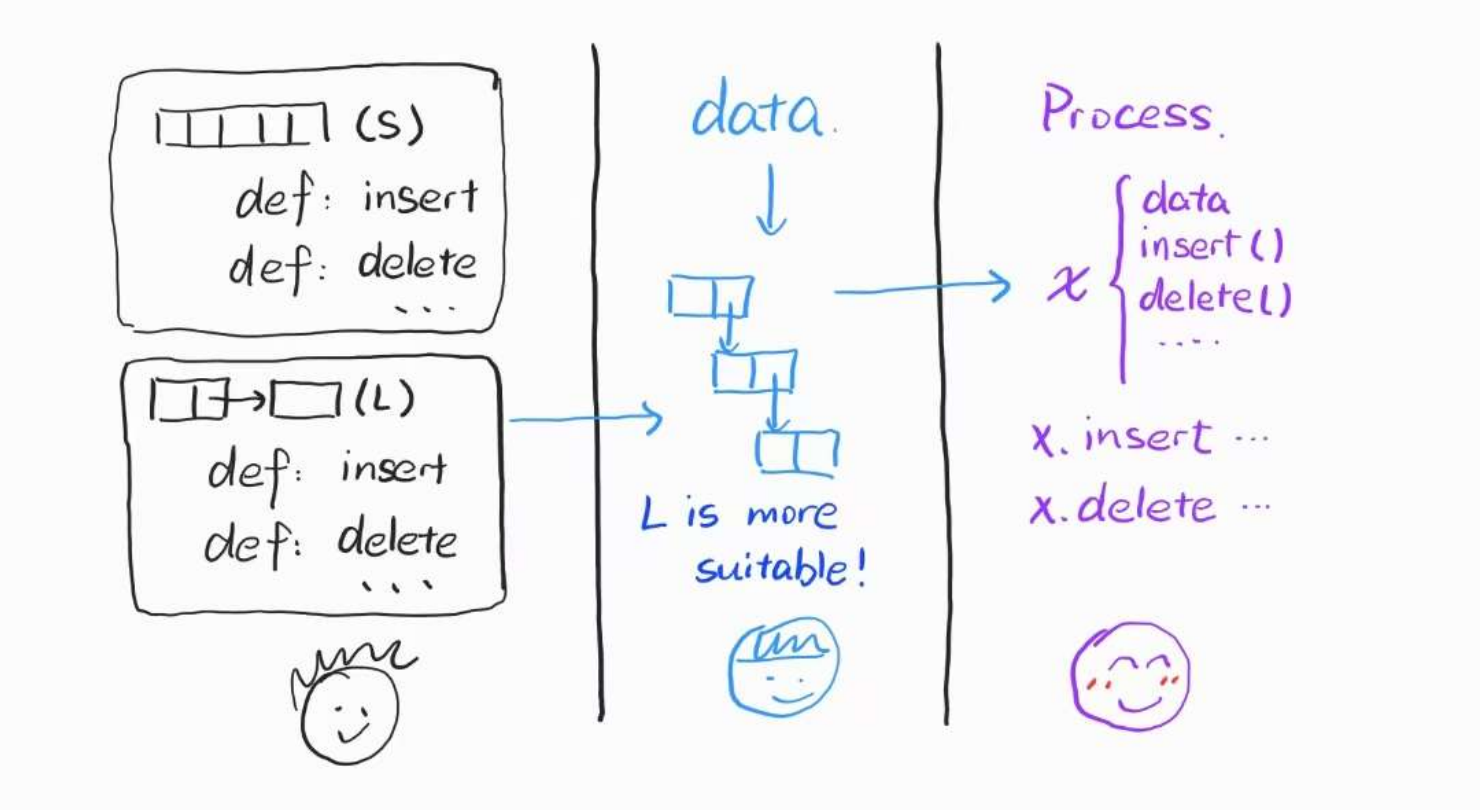
- Python
- 从过程到对象
- 类(class)

面向过程：数据结构+算法



- Python
- 从过程到对象
- 类(class)

面向对象：类+类+类+……



- Python

- 从过程到对象

- 类(class)

万物皆类：

- 某种数据结构
- 一种变换
- 一个算法
- ……

- Python
- 从过程到对象
- 类(class)

类(class)与实例(instance)：

```
class Animal:
    def __init__(self, name, species):
        self.name = name          # 变量
        self.species = species    # 变量

    def make_sound(self):         # 方法(method)
        print("Some generic animal sound")



mimi = Animal("Mimi", "Cat")
mimi.make_sound()      # 输出: "Some generic animal sound"
```

- Python
- 从过程到对象
- 类(class)

类(class)与子类(subclass):

```
class Animal:
    def __init__(self, name, species):
        self.name = name          # 变量
        self.species = species    # 变量

    def make_sound(self):         # 方法
        print("Some generic animal sound")

class Dog(Animal):                # Dog 继承 Animal
    def __init__(self, name, breed):
        super().__init__(name, "Dog")
        self.breed = breed

    def make_sound(self):         # 方法重写
        print("WerWer!")

XiaoBai = Dog("Xiaobai", "Beagles")
XiaoBai.make_sound()              # WerWer!
```

2 实现一个神经网络

- 一些有用的包
- 基本结构
- 详细实现

- 一些有用的包

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as data
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
```

- 基本结构
- 详细实现

- 一些有用的包
- 基本结构

```
class BostonHousingDataset(data.Dataset):
    def __init__(self, train=True)
    def __len__(self)
    def __getitem__(self, index)

class FNN(nn.Module):
    def __init__(self, input_dim)
    def forward(self, x):

class Trainer:
    def __init__(self, model, train_loader, test_loader, device=, lr, epochs)
    def train(self)
    def evaluate(self, calc_loss=False)
    def plot_losses(self)

if __name__ == "__main__":
```

- 详细实现

- 一些有用的包
- 基本结构
- 详细实现

```python
if __name__ == "__main__":
    # 设备选择：如果有 GPU 就用 GPU，否则用 CPU
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # 创建数据集
    train_dataset = BostonHousingDataset(train=True)
    test_dataset = BostonHousingDataset(train=False)
    # 创建数据加载器
    train_loader = data.DataLoader(train_dataset, batch_size=16, shuffle=True)
    test_loader = data.DataLoader(test_dataset, batch_size=16, shuffle=False)

    # 初始化模型
    model = FNN(input_dim=train_dataset.X.shape[1])

    # 训练器
    trainer = Trainer(model, train_loader, test_loader, device=device,
                      lr=0.001, epochs=20)
    trainer.train()
    trainer.evaluate()
    trainer.plot_losses()
```

- 一些有用的包
- 基本结构
- 详细实现

```python
class FNN(nn.Module):
    """
    经典的全连接神经网络（Feedforward Neural Network, FNN）
    结构：输入层 -> 隐藏层(ReLU) -> 输出层
    """
    def __init__(self, input_dim):
        super(FNN, self).__init__()
        self.fc1 = nn.Linear(input_dim, 64) # 输入层到隐藏层
        self.relu = nn.ReLU()               # ReLU 激活函数
        self.fc2 = nn.Linear(64, 1)         # 隐藏层到输出层

    def forward(self, x):
        """前向传播"""
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

- 一些有用的包
- 基本结构
- 详细实现

```
class Trainer:
    """
    训练和评估神经网络模型的类
    """
    def __init__(self, model, train_loader, test_loader,
                 device="cpu", lr=0.01, epochs=300):
        self.model = model.to(device)  # 将模型移动到指定设备
        self.train_loader = train_loader
        self.test_loader = test_loader
        self.device = device
        self.epochs = epochs
        # 使用均方误差（MSE）作为损失函数
        self.criterion = nn.MSELoss()
        # 使用 SGD 优化器
        self.optimizer = optim.SGD(self.model.parameters(), lr=lr)
        # 记录误差
        self.train_losses = []
        self.test_losses = []
```

- 一些有用的包
- 基本结构
- 详细实现

```
class Trainer:
    """
    训练和评估神经网络模型的类
    """
    def train(self):
        """训练模型"""
        self.model.train()  # 设置为训练模式
        for epoch in range(self.epochs):
            total_loss = 0
            for batch_X, batch_y in self.train_loader:
                batch_X, batch_y =
                        batch_X.to(self.device), batch_y.to(self.device)
                self.optimizer.zero_grad()                # 梯度清零
                outputs = self.model(batch_X)             # 前向传播
                loss = self.criterion(outputs, batch_y)   # 计算损失
                loss.backward()                           # 反向传播
                self.optimizer.step()                     # 更新参数
                total_loss += loss.item()
            # 计算训练误差……
            # 计算测试误差……
            # 每 1 轮打印一次损失……
```

- 一些有用的包
- 基本结构
- 详细实现

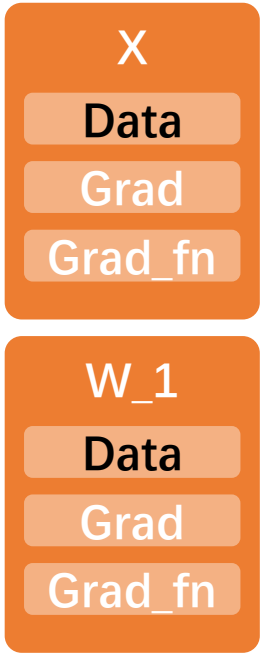
```
class Trainer:
    """
    训练和评估神经网络模型的类
    """
    def train(self):
        """训练模型"""
        self.model.train()  # 设置为训练模式
        for epoch in range(self.epochs):
            total_loss = 0
            for batch_X, batch_y in self.train_loader:
                batch_X, batch_y =
                        batch_X.to(self.device), batch_y.to(self.device)
                self.optimizer.zero_grad()                  # 梯度清零
                outputs = self.model(batch_X)               # 前向传播
                loss = self.criterion(outputs, batch_y)     # 计算损失
                loss.backward()                             # 反向传播
                self.optimizer.step()                       # 更新参数
                total_loss += loss.item()
            # 计算训练误差……
            # 计算测试误差……
            # 每 1 轮打印一次损失……
```

- 一些有用的包
- 基本结构
- 详细实现

loss ∈

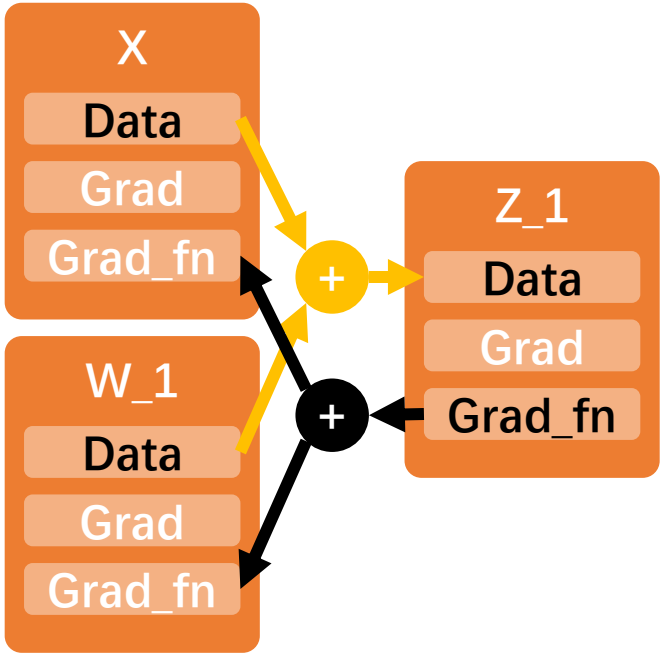
- 一些有用的包
- 基本结构
- 详细实现

Forward(由model实现)



- 一些有用的包
- 基本结构

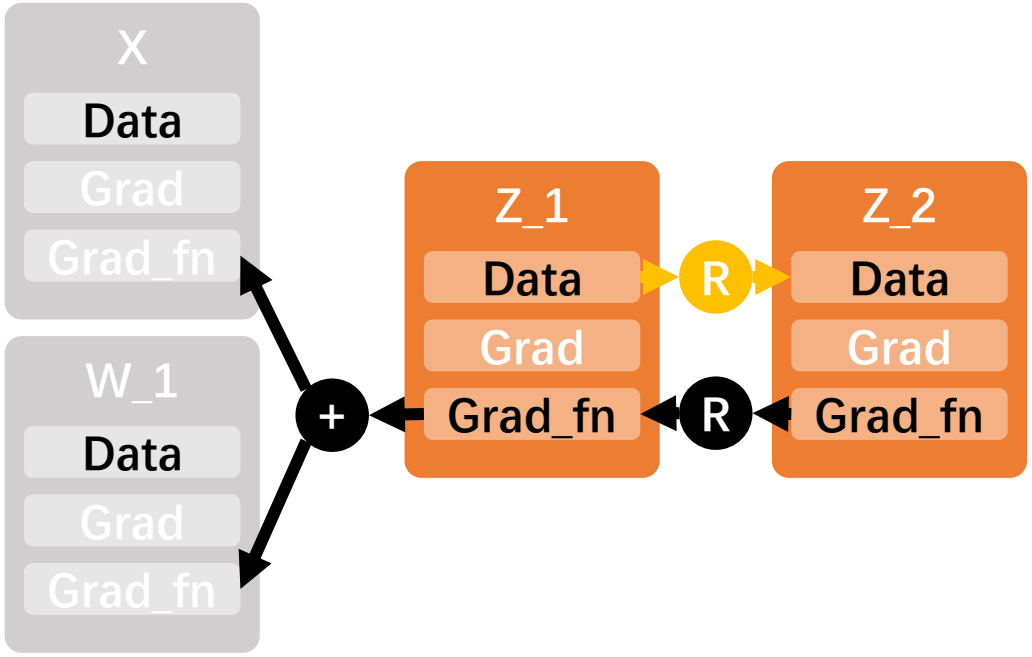
Forward(由model实现)



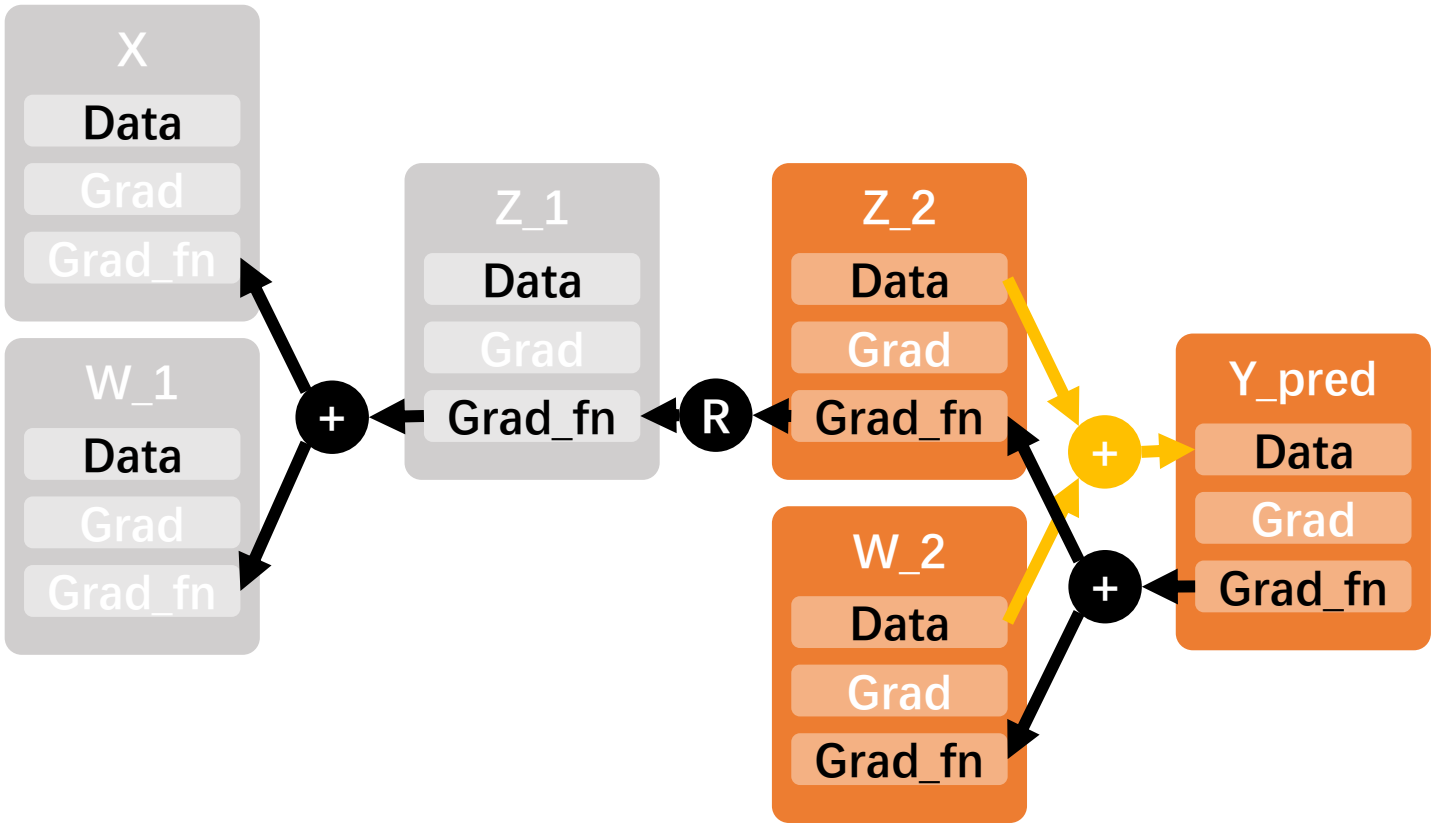
- 详细实现

- 一些有用的包
- 基本结构
- 详细实现

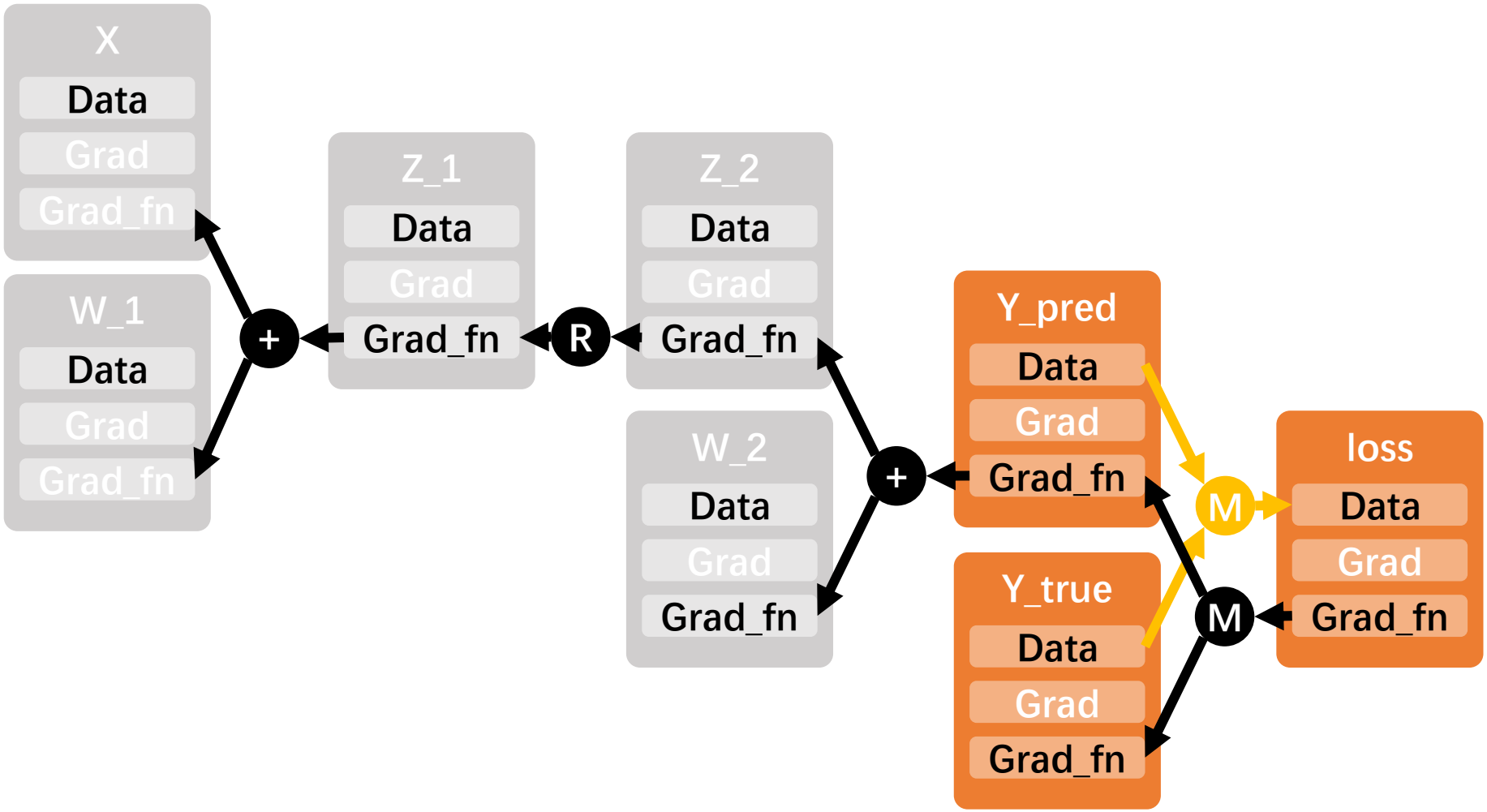
Forward(由model实现)



- 一些有用的包
- 基本结构
- 详细实现

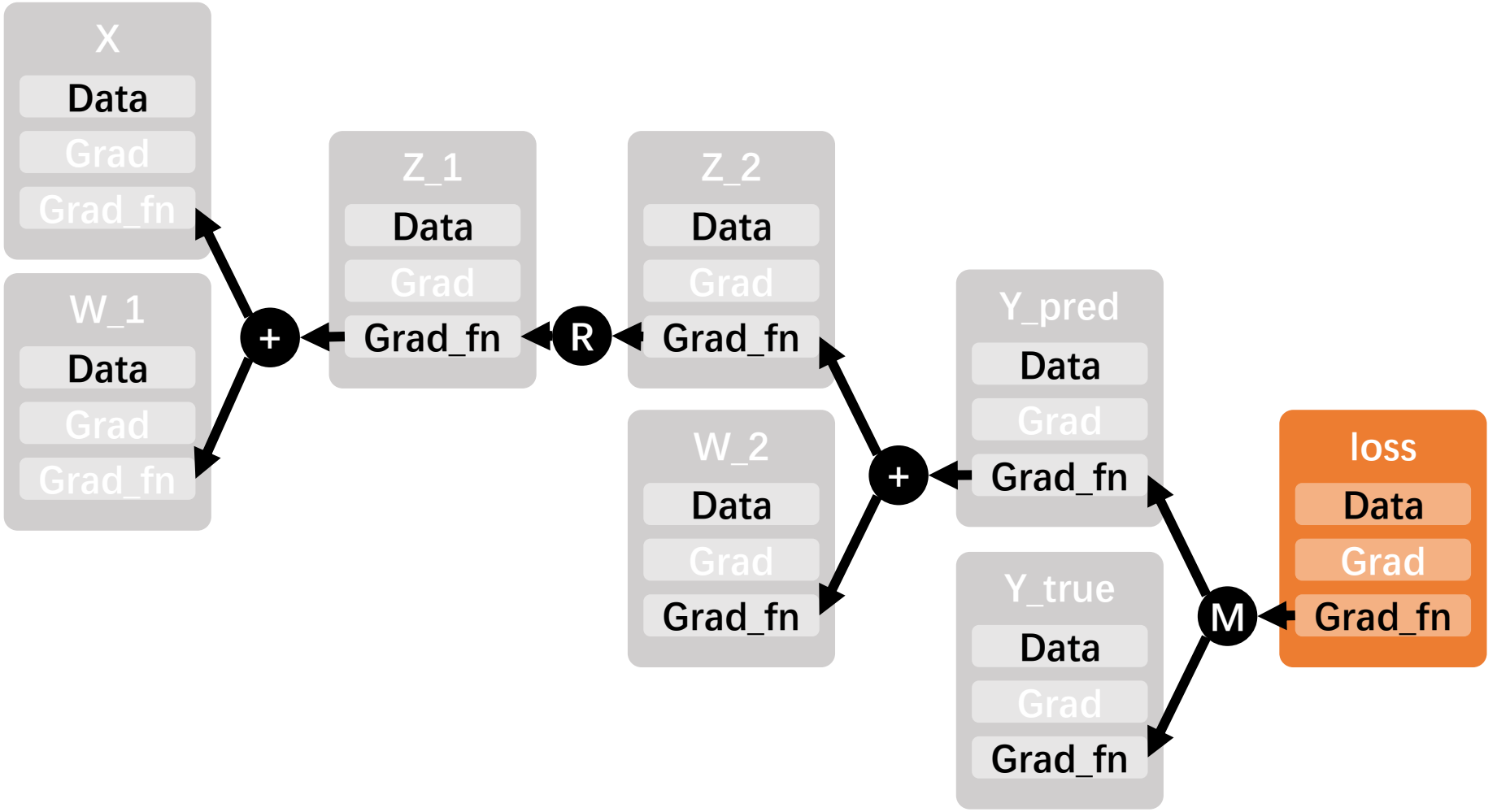


- 一些有用的包
- 基本结构
- 详细实现



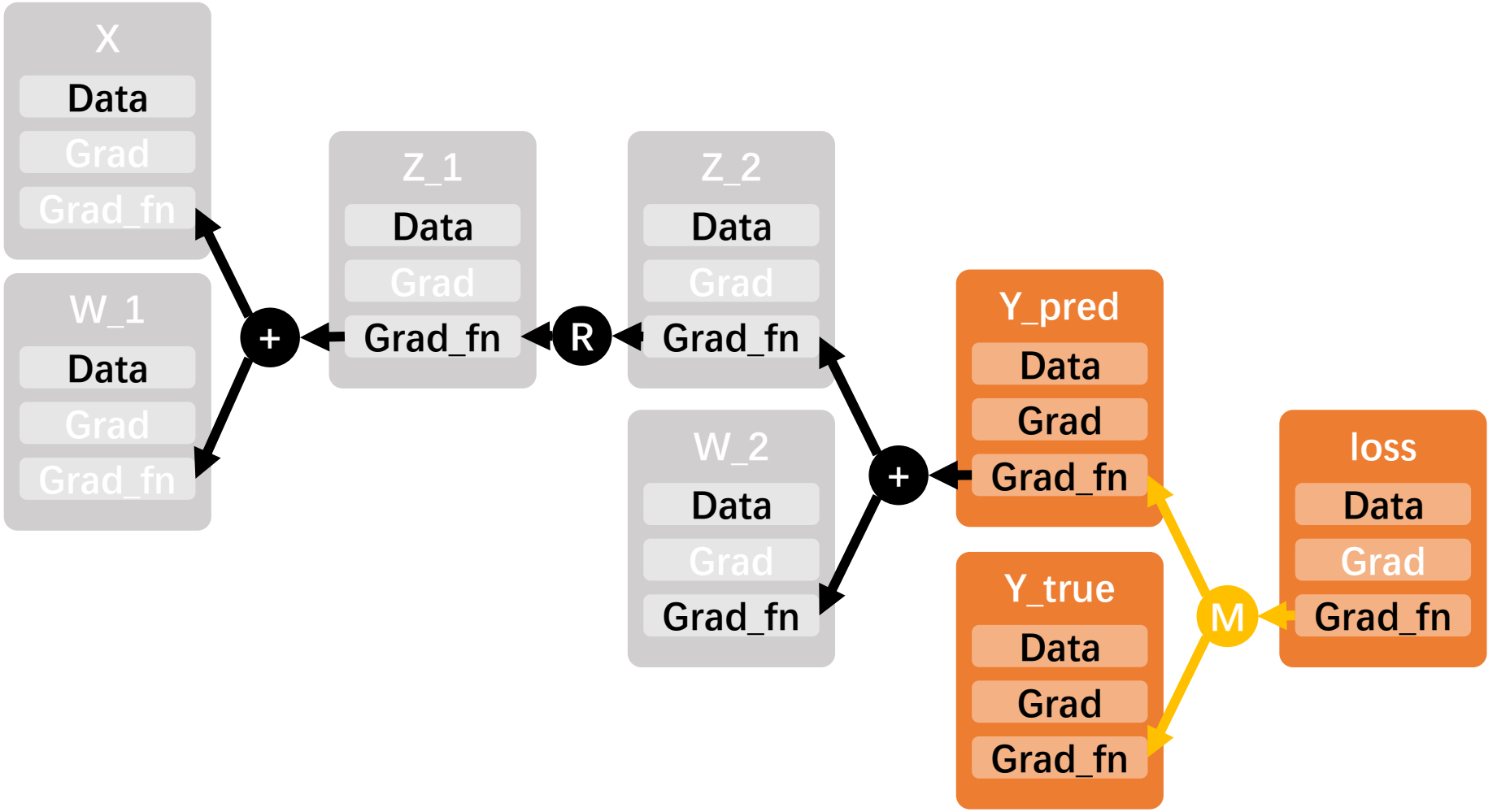
- 一些有用的包
- 基本结构
- 详细实现

Backward(由tensor递归实现)

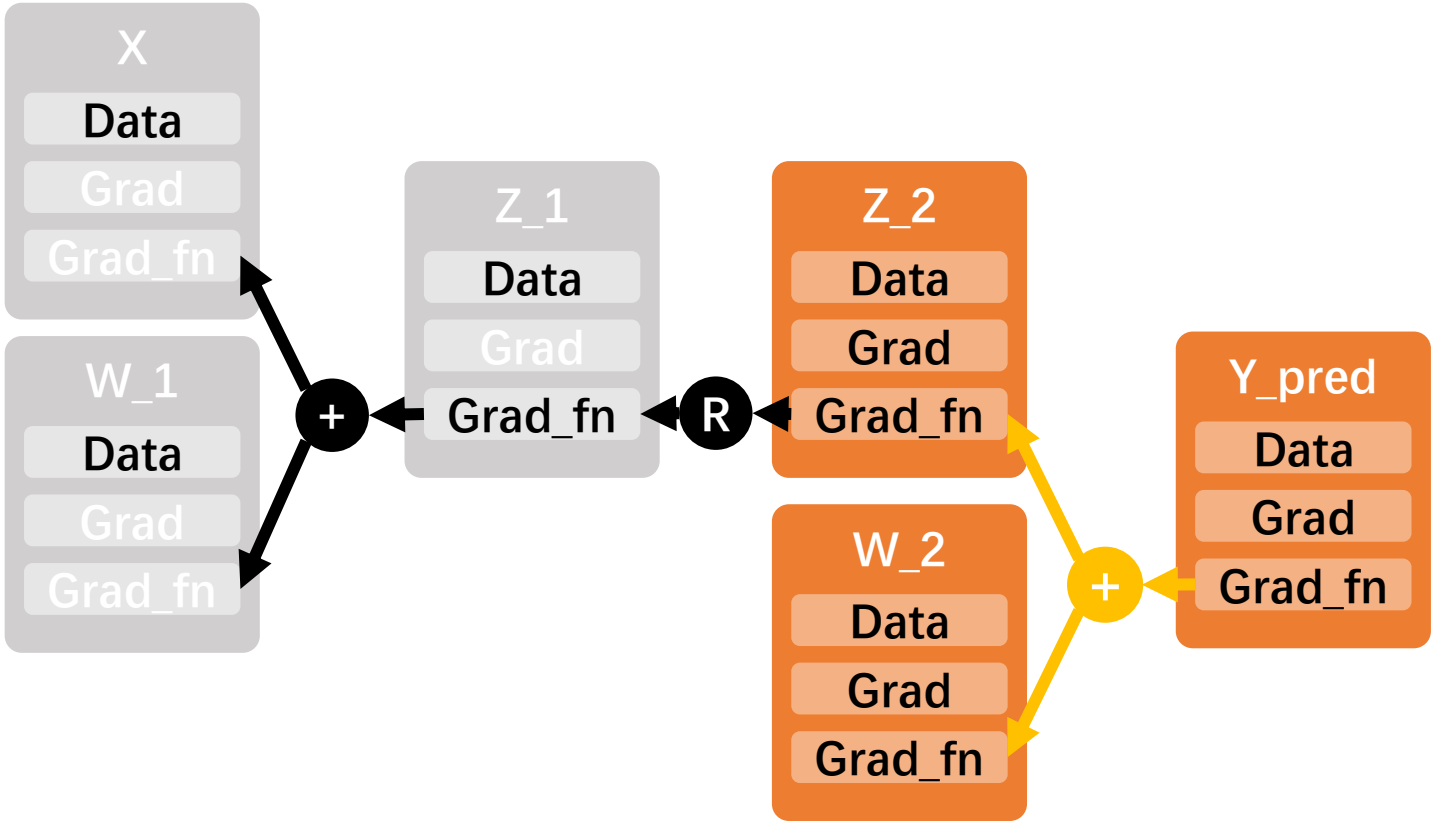


- 一些有用的包
- 基本结构
- 详细实现

Backward(由tensor递归实现)

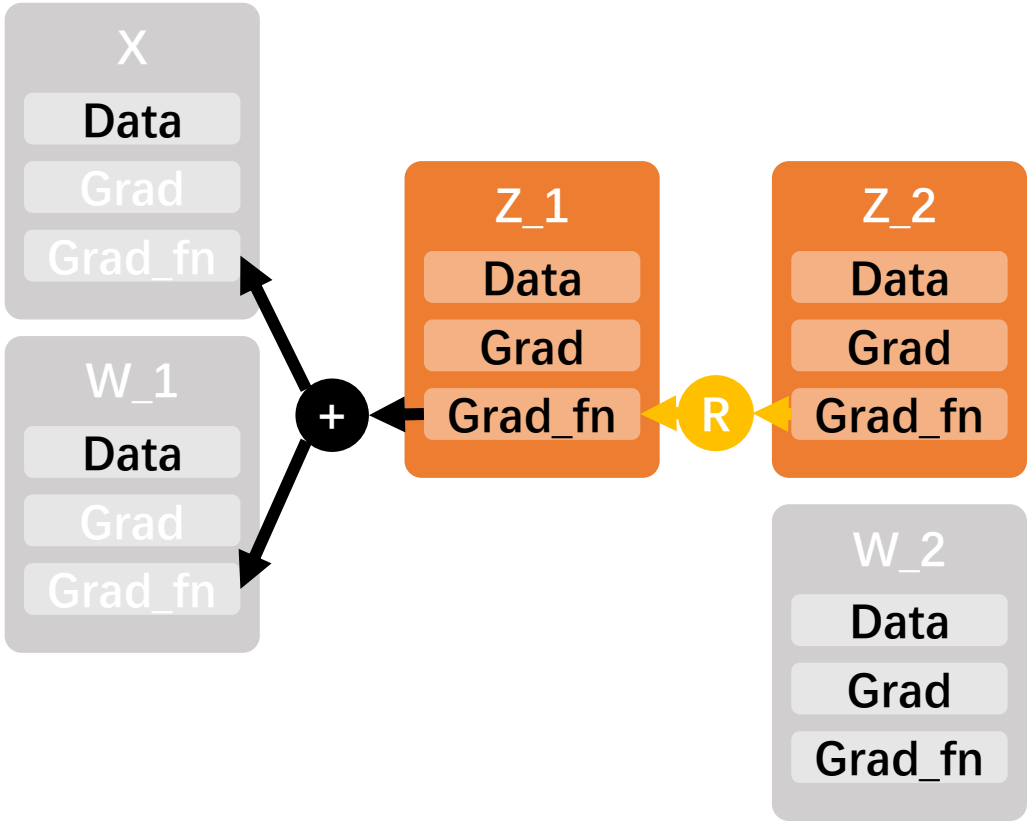


- 一些有用的包
- 基本结构
- 详细实现



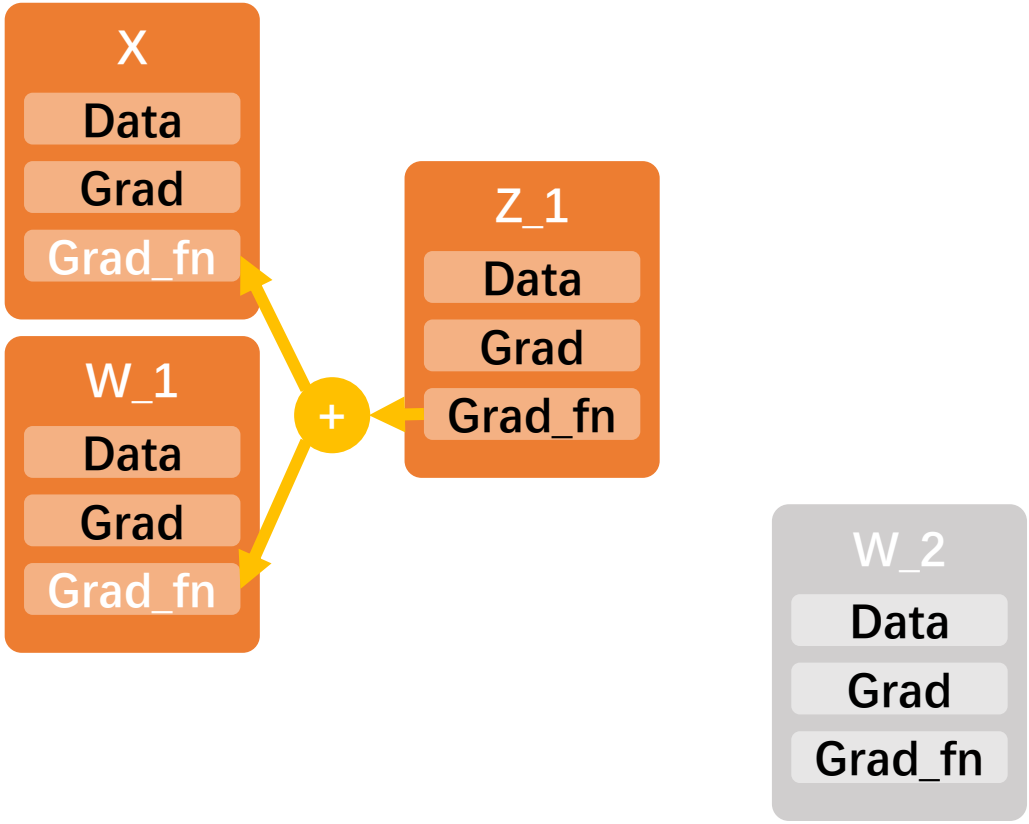
- 一些有用的包
- 基本结构
- 详细实现

Backward(由tensor递归实现)



- 一些有用的包
- 基本结构

Backward(由tensor递归实现)



- 详细实现

- 一些有用的包
- 基本结构

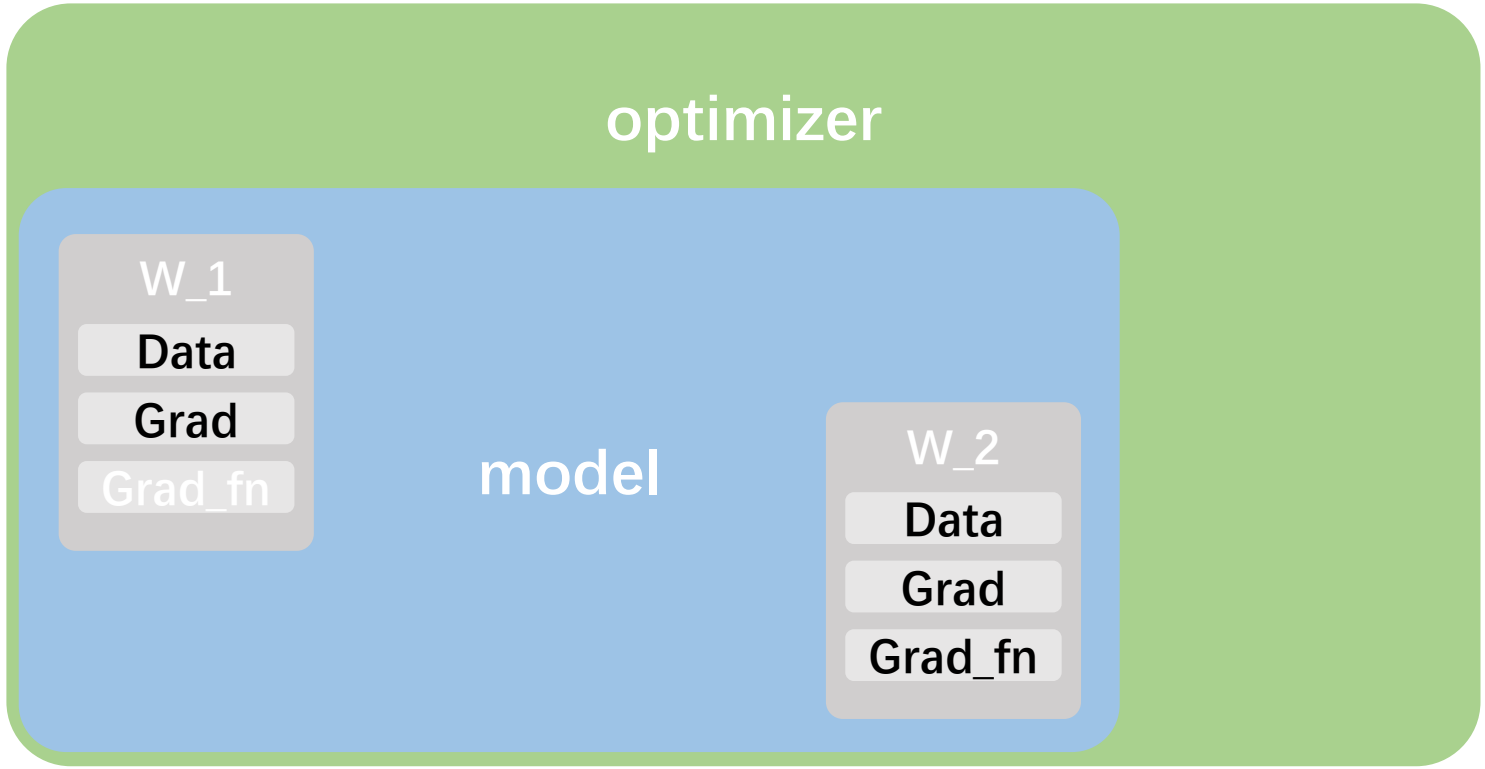
Backward(由tensor递归实现)

- 详细实现

- 一些有用的包
- 基本结构

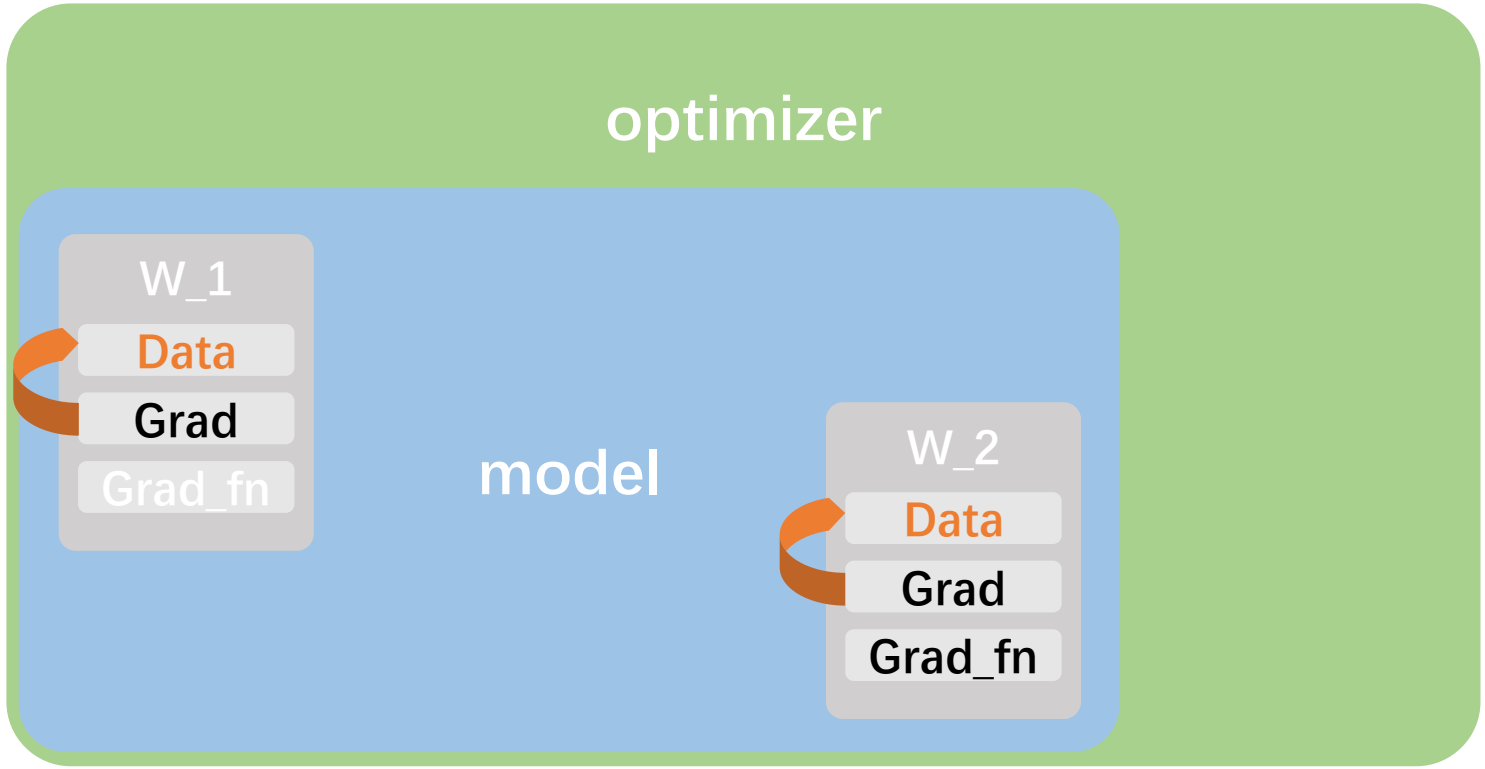
optimizer.step()

- 详细实现



- 一些有用的包
- 基本结构

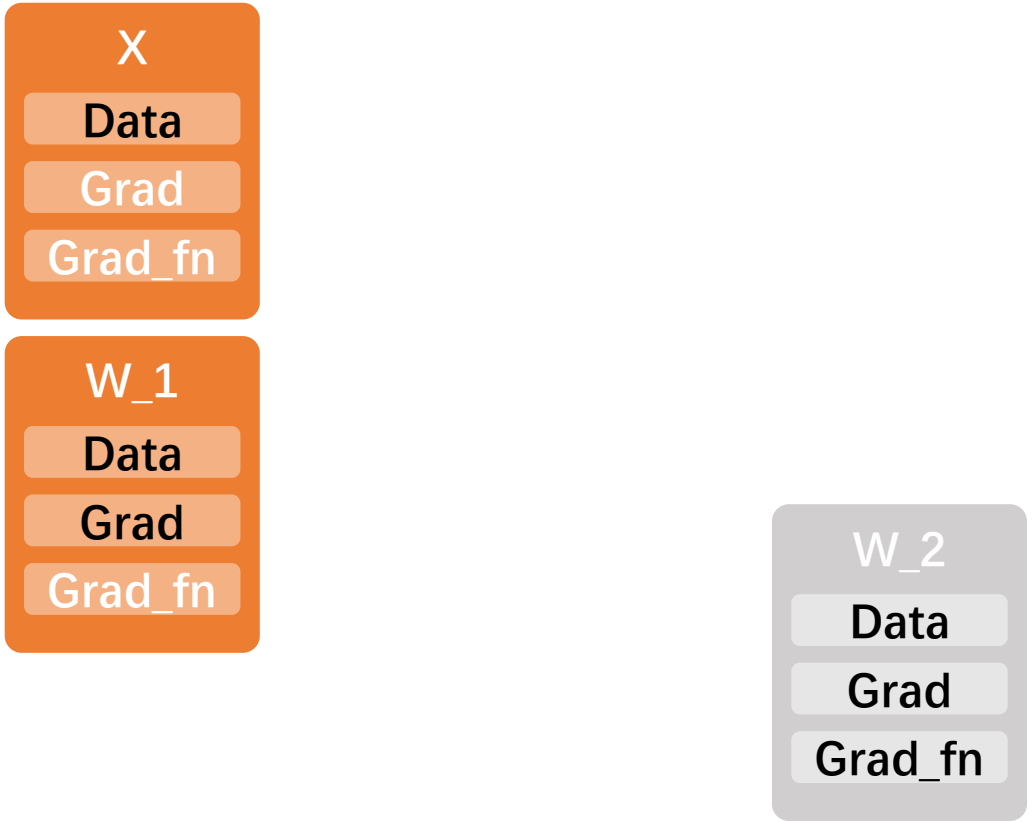
optimizer.step()



- 详细实现

- 一些有用的包
- 基本结构
- 详细实现

Forward(由model实现)



- 一些有用的包
- 基本结构
- 详细实现

```
class Trainer:
    """
    训练和评估神经网络模型的类
    """
    def train(self):
        """训练模型"""
        self.model.train()  # 设置为训练模式
        for epoch in range(self.epochs):
            total_loss = 0
            for batch_X, batch_y in self.train_loader:
                batch_X, batch_y =
                        batch_X.to(self.device), batch_y.to(self.device)
                self.optimizer.zero_grad()                  # 梯度清零
                outputs = self.model(batch_X)               # 前向传播
                loss = self.criterion(outputs, batch_y)     # 计算损失
                loss.backward()                             # 反向传播
                self.optimizer.step()                       # 更新参数
                total_loss += loss.item()
            # 计算训练误差……
            # 计算测试误差……
            # 每 1 轮打印一次损失……
```

- 一些有用的包
- 基本结构
- 详细实现

```
class Trainer:
    """
    训练和评估神经网络模型的类
    """
    def evaluate(self, calc_loss=False):
        """在测试集上评估模型"""
        self.model.eval()  # 设置为评估模式
        total_loss = 0
        with torch.no_grad():  # 不计算梯度
            for batch_X, batch_y in self.test_loader:
                batch_X, batch_y = batch_X.to(self.device),
                                   batch_y.to(self.device)
                outputs = self.model(batch_X)
                loss = self.criterion(outputs, batch_y)
                total_loss += loss.item()

        avg_loss = total_loss / len(self.test_loader)

        if not calc_loss:
            print(f"Test MSE Loss: {avg_loss:.4f}")

        return avg_loss
```

3 作业—发布

图 1: 你可以说出图中各个点对应哪些地方吗？（不要求回答）

现有地图软件均有路线规划的功能，可以找到从 A 点到 B 点的最佳路径，若仅用距离来衡量路径的优劣，则该问题则退化为了求最短路径。

在本次作业中，我们将制作一个“校内路线规划”程序，求解从校内一点到另一点的最短路径。此外，我们还将为该程序增加更强大的功能：你不仅可以直接查询从宿舍到三教的最短路径，还可以进行更复杂的查询，比如从宿舍出发，途经图书馆（不限东区图书馆或西区图书馆）和咖啡店，最后到达某个自习室的最短路径。

本次作业包括书面部分（问题回答）和编程部分（代码填空）。

书面部分需使用 LaTeX 完成，模板文件为 report_template.tex。注意：为方便助教批改，你还需要将编程部分完成后的代码复制到模板中的对应位置。书面作业完成后，请导出为 report.pdf 文件。

编程部分的完整代码位于 Project 文件夹中。你仅需在 submission.py 中指定位置完成代码，具体如下：

```
# BEGIN_YOUR_CODE

# END_YOUR_CODE
```

除 submission.py 之外，请勿修改其他任何文件。建议每完成一部分代码后，及时运行单个样例进行测试，例如执行 python grader.py 1a-1-basic，以便快速排查代码问题。每个测试样例的编号已经在 README 中给出。

完成所有代码后，运行 python grader.py 可查看当前得分。在本次作业中，每一道代码题的得分将有一半来自 grader.py 给出的分数，另一半来自助教对代码的打分。grader.py 包含可见样例和隐藏样例，你在本地运行时看到的得分仅为可见样例的得分，最终成绩将同时包含可见和隐藏样例。

完成所有内容后，请将 submission.py 与 report.pdf 打包成名为“学号 _ 姓名 _HW1.zip”的压缩文件，并上传至 BB 系统。助教将根据你提交的文件进行评分。

注意事项:

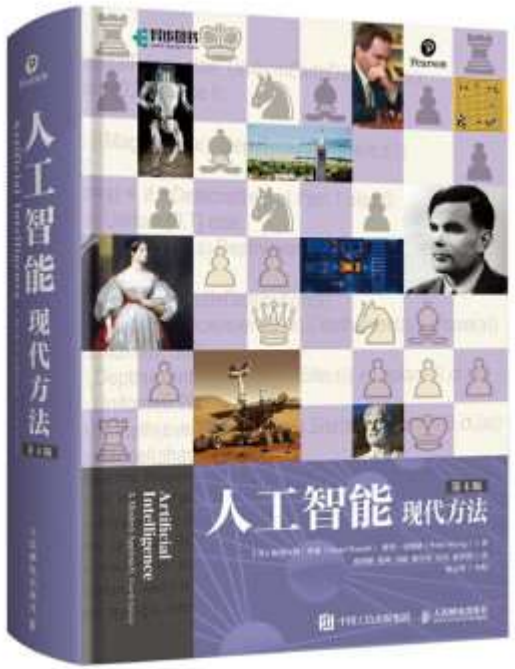
- 本次作业必须独立完成，任何形式的抄袭都不允许。如发现有互相抄袭的情况，抄袭者将平分获得的分数。
- 作业的截止时间为 2024 年 4 月 10 日 23:59:59。迟交 1/2/4/7 天分别会扣除 5/15/40/100 分。
- 如果在做作业过程中遇到问题，可以在 Github Issue 中提问或者线下在答疑课提问，也欢迎同学们相互解答问题。

体验反馈 [2%]

你可以写下任何反馈，包括但不限于以下几个方面：课堂、作业、助教工作等等。

(a) [**必做**] 你在本次作业花费的时间大概是?

(b) [**选做**] 你可以随心吐槽课程不足的方面，或者给出合理的建议。若没有想法，直接忽略本小题即可。你的回答不会影响给分，若仍然担心，你可以通过该匿名问卷回答



一些参考资料：

课程内容：

Artificial Intelligence: A Modern Approach, 4th

https://aima.cs.berkeley.edu/

Python：

CS221 Tutorial: Python Review – Colab

https://docs.python.org/3/tutorial/classes.html#random-remarks

谢谢大家