北京大学

机器学习编程模型

Machine Learning Programming Model

主讲人：王乐业　讲义：董豪

内容提要

- 机器学习编程模型演进 Evolution
- 机器学习工作流 Workflow
- 定义深度神经网络 DNN Definition
- C/C++编程接口 C/C++ APIs

内容提要

- 机器学习编程模型演进 Evolution
- 机器学习工作流 Workflow
- 定义深度神经网络 DNN Definition
- C/C++编程接口 C/C++ APIs

机器学习编程模型演进

- API复杂
- 无GPU支持
- 手动实现网络

- 生态友好
- 分布式支持
- 效率优化

- 编译层优化
- 多场景多任务支持
- 丰富的套件支持

- AI编译器
- 大模型
- 全场景隐私与公平

2000年初期
萌芽阶段*

2012年~2014年
成长阶段*

2015年~2019年
稳定阶段*

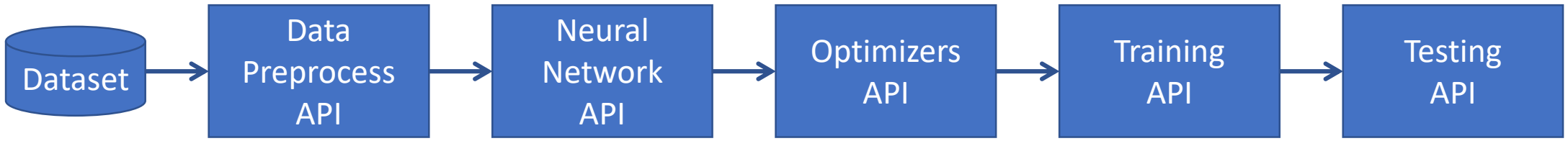
2020年~
深化阶段*

*《AI框架发展白皮书》

内容提要

- 机器学习编程模型演进 Evolution
- 机器学习工作流 Workflow
- 定义深度神经网络 DNN Definition
- C/C++编程接口 C/C++ APIs

机器学习工作流



Examples
- torch.data
- torch.DataLoader
- ms.MindData

Examples
- torch.Dense
- ms.nn.Dense
- tf.keras.Dense

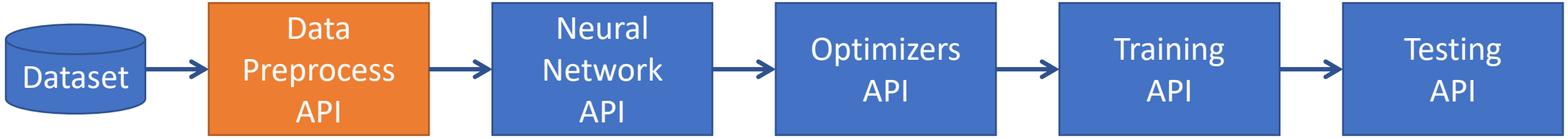
Examples
- torch.optimizer
- ms.nn.optim
- torch.Optimizer

Examples
- model.fit
- ms.Model.train

Examples
- model.eval
- ms.Model.eval

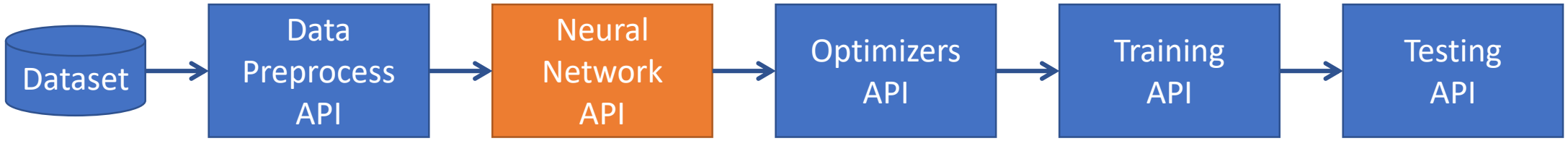
- 数据处理：数据处理 API将数据集从磁盘读入
- 模型结构：模型定义 API 来定义机器学习模型
- 损失函数和优化器：损失函数进行评估、优化器计算梯度更新训练参数
- 训练过程：将数据集中的数据按照小批量的方式读取，反复计算梯度更新模型
- 测试和调试：测试 API 对当前模型的精度进行评估

机器学习工作流--数据处理



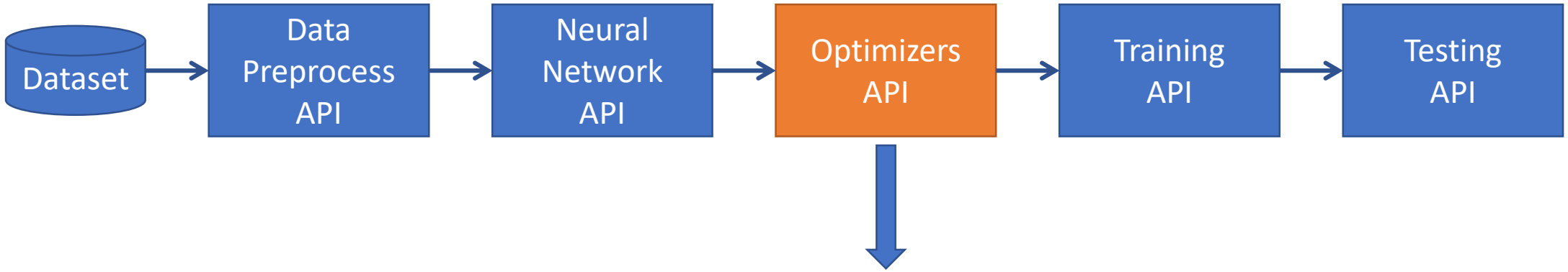

```python
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor

# Download training data from open datasets.
training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
)
# Download test data from open datasets.
test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor(),
)
batch_size = 64
# Create data loaders.
train_dataloader = DataLoader(training_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)

for X, y in test_dataloader:
    print(f"Shape of X [N, C, H, W]: {X.shape}")
    print(f"Shape of y: {y.shape} {y.dtype}")
    break
```

机器学习工作流--模型定义



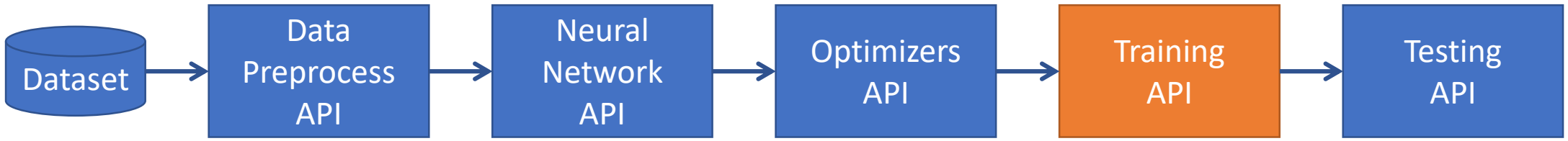

```python
# Define model
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
print(model)
```

机器学习工作流--损失函数和优化器



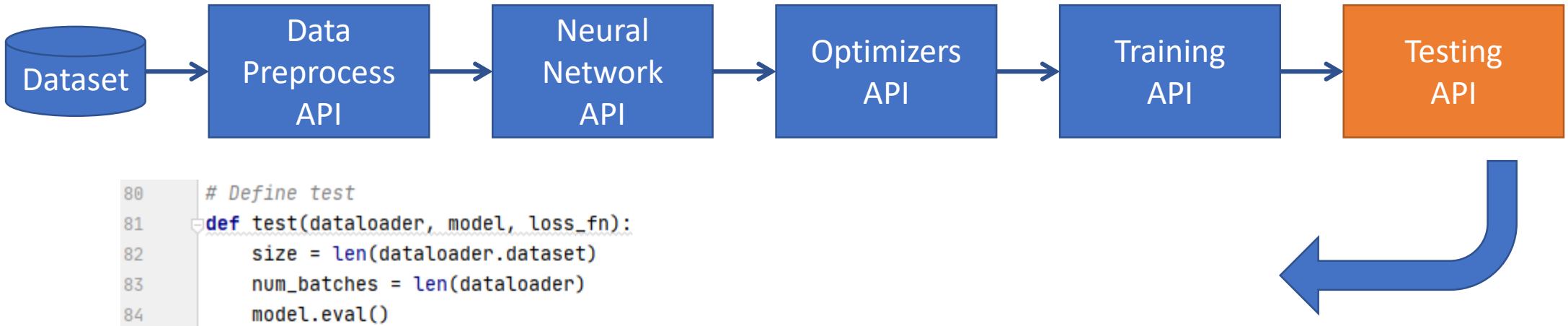

```
55  # Define loss function
56  loss_fn = nn.CrossEntropyLoss()
57  # Define optimizer
58  optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
```

机器学习工作流--训练过程




```python
# Define train
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), (batch + 1) * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

机器学习工作流--测试和验证



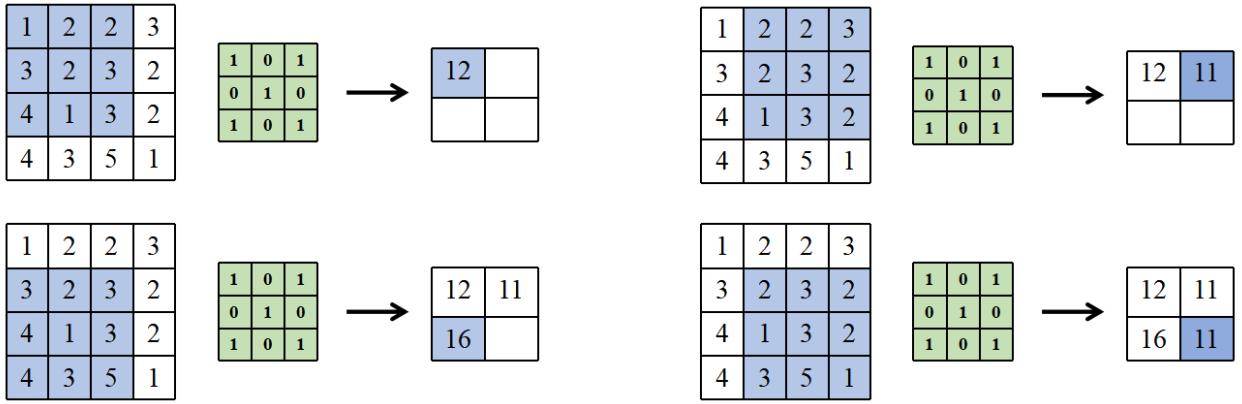

```python
# Define test
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

内容提要

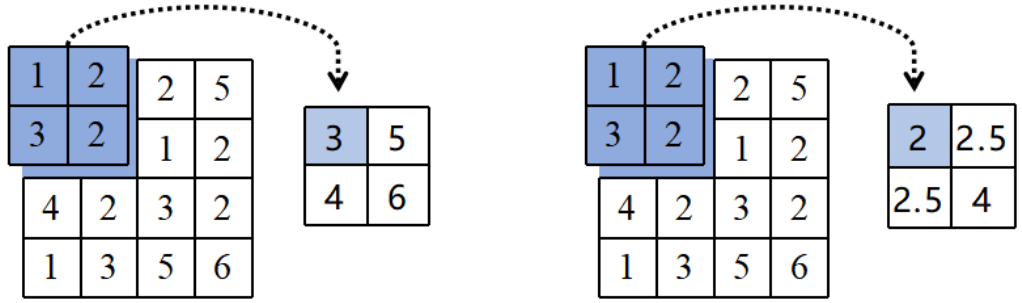
- 机器学习编程模型演进 Evolution
- 机器学习工作流 Workflow
- 定义深度神经网络 DNN Definition
- C/C++编程接口 C/C++ APIs

常用神经网络层

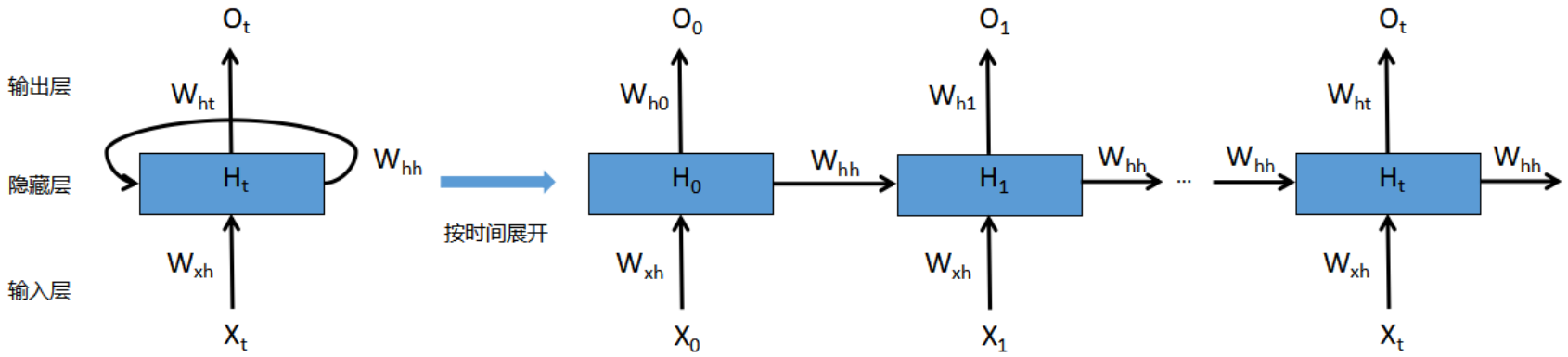
Convolution



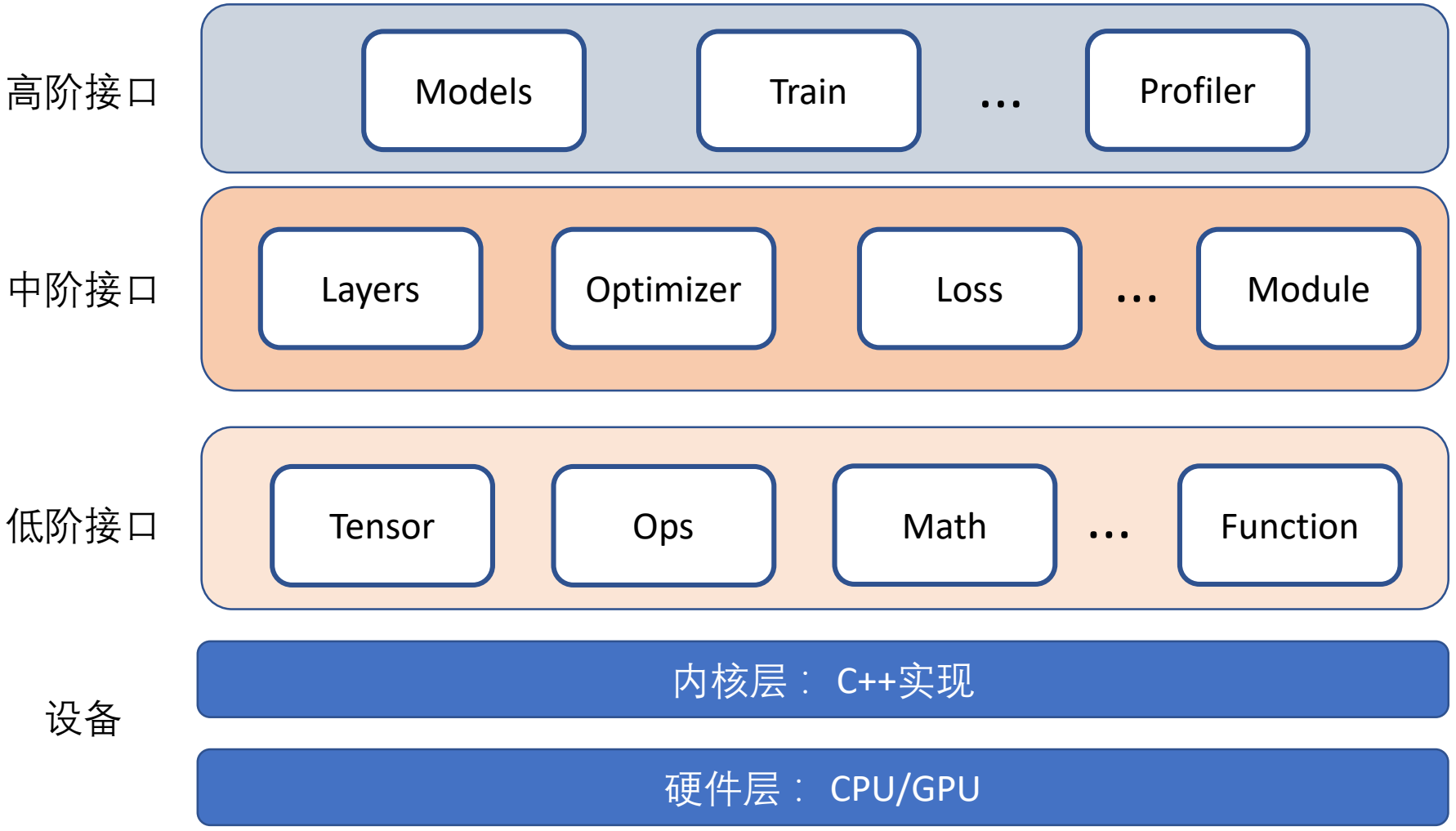
Pooling



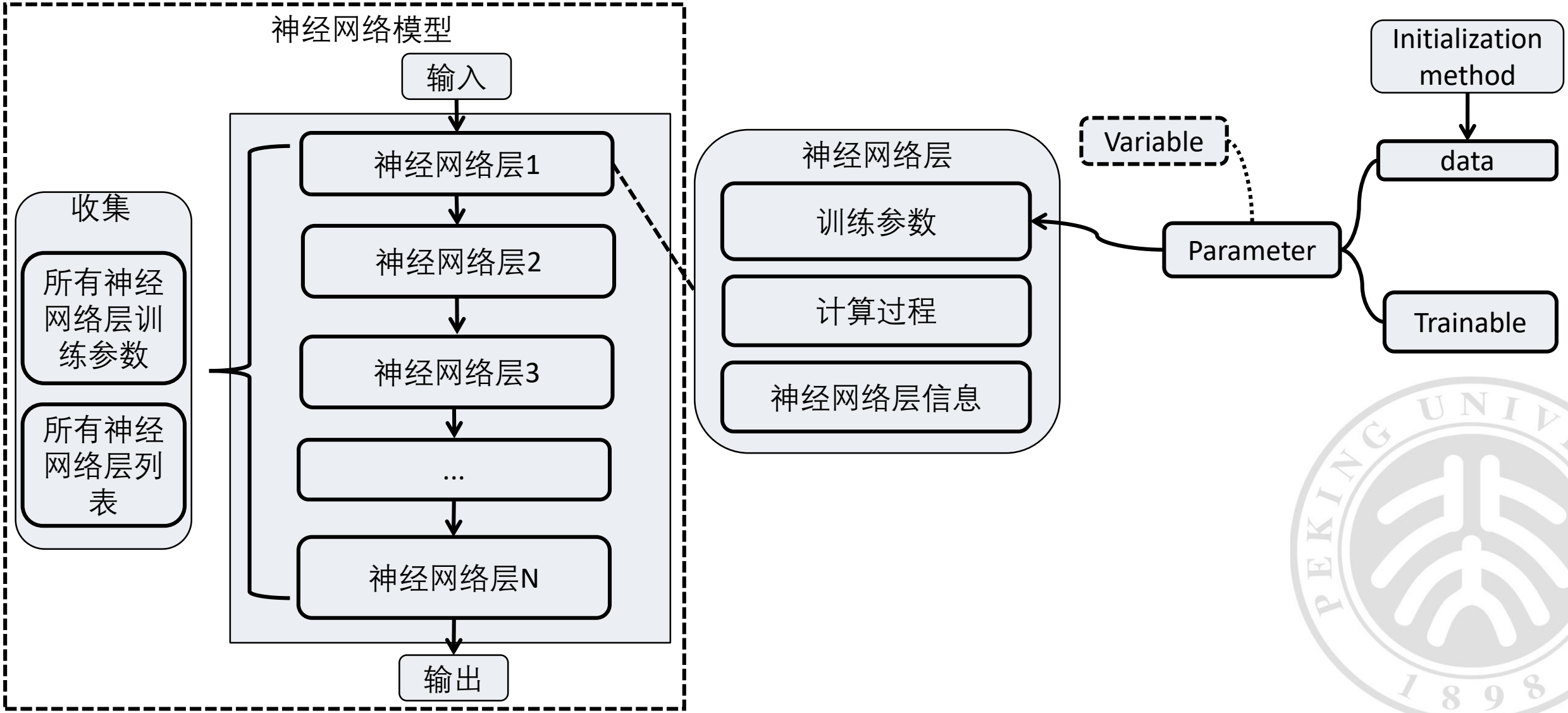
Recurrent Neural Network



神经网络层实现原理


高阶接口
Models
Train
...
Profiler
中阶接口
Layers
Optimizer
Loss
...
Module
低阶接口
Tensor
Ops
Math
...
Function
设备
内核层： C++实现
硬件层： CPU/GPU

神经网络层实现原理


神经网络模型
输入
神经网络层1
神经网络层2
神经网络层3
...
神经网络层N
输出
收集
所有神经网络层训练参数
所有神经网络层列表
神经网络层
训练参数
计算过程
神经网络层信息
Variable
Parameter
Initialization method
data
Trainable

神经网络层实现原理

深度学习模型构建组件基类（Module）

构造器（__init__）
self._params = OrderedDict()
self._modules = OrderedDict()
self.training = False

→ 使用OrderedDict初始化参数和神经网络层收集器

设置属性（__setattr__）
isinstance(value, parameter)
isinstance(value, Module)

→ 通过Python内置方法setattr在属性设置时管理训练参数和神经网络层

执行计算（forward）
动态图模式：转发计算并且返回结果
静态图模式：编译并运行cell

→ 通过Python内置方法call来执行计算

训练参数（parameters）
遍历神经网络层返回所有神经网络层参数

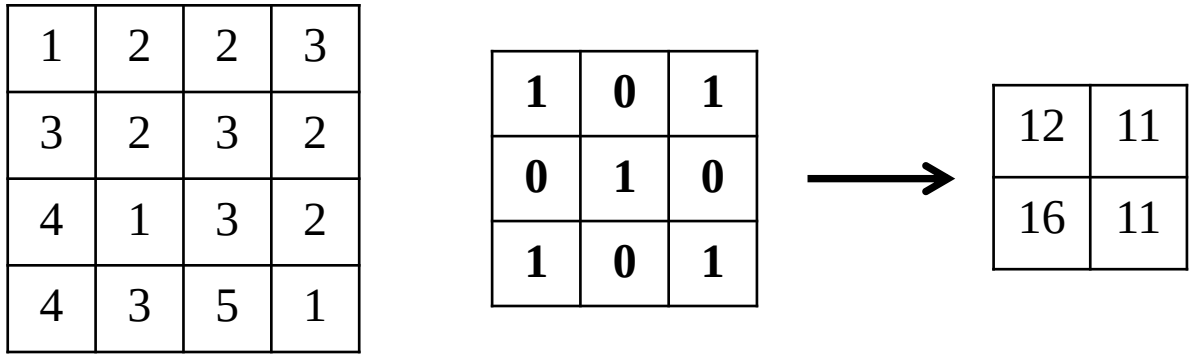
→ 利用收集到的神经网络层遍历返回所有层参数

神经网络层（modules）
返回所有神经网络层迭代器

→ 利用收集到的神经网络层遍历返回所有层

自定义方法

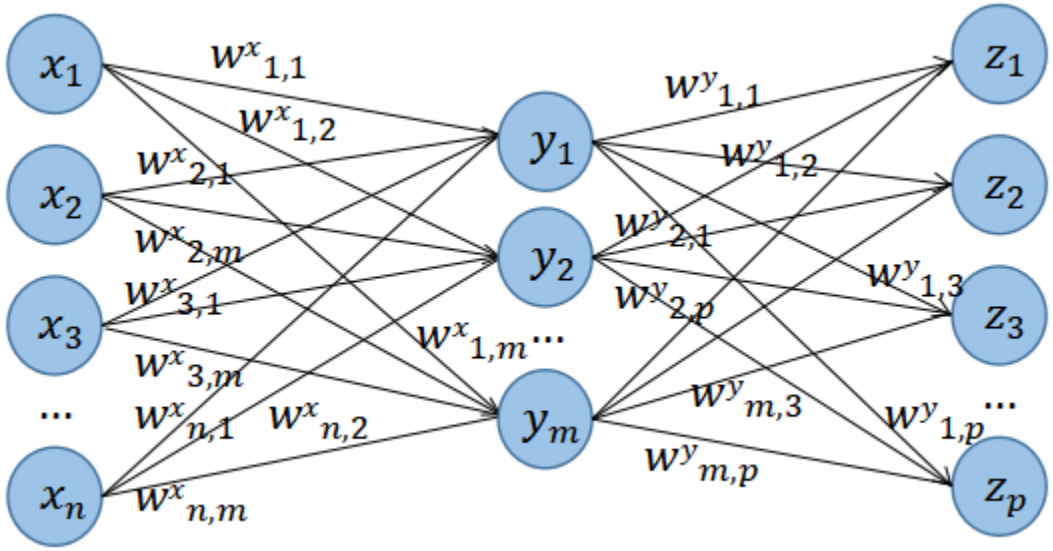
自定义神经网络层



输入　　卷积核　　输出

```
# 定义卷积层
class Conv2D(Cell):
    def __init__(self, in_channels, out_channels, ksize, stride, padding):
        # 卷积核大小为 ksize x ksize x inchannels x out_channels
        filters_shape = (out_channels, in_channels, ksize, ksize)
        self.stride = stride
        self.padding = padding
        self.filters = Variable(random_normal(filters_shape))

    def __call__(self, inputs):
        outputs = convolution(inputs, self.filters, self.stride, self.padding)
```

自定义神经网络模型




```python
# 导入需要用到的模块
import mindspore.nn as nn
# 定义线性模型
class MLPNet(nn.Cell):
    def __init__(self):
        super(MLPNet, self).__init__()
        self.flatten = nn.Flatten()
        self.dense1 = nn.Dense(32*32, 128)
        self.dense2 = nn.Dense(128, 64)
        self.dense3 = nn.Dense(64, 10)

    def construct(self, inputs):
        x = self.flatten(inputs)
        x = self.dense1(x)
        x = self.dense2(x)
        logits = self.dense3(x)
        return logits
# 实例化网络
net = MLPNet()
```

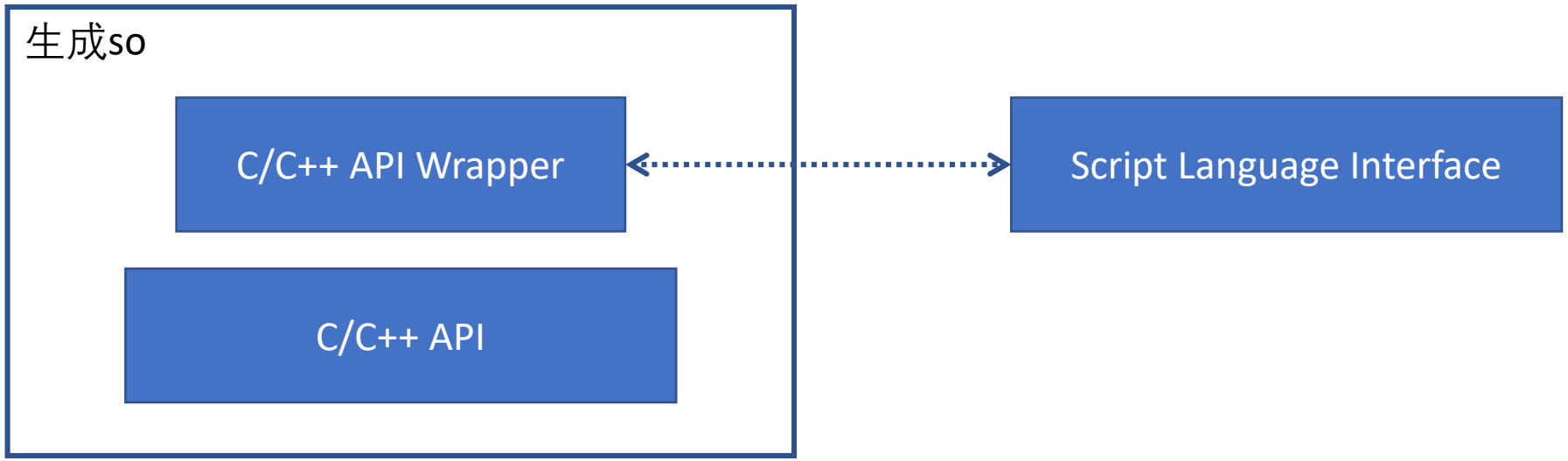
内容提要

- 机器学习编程模型演进 Evolution
- 机器学习工作流 Workflow
- 定义深度神经网络 DNN Definition
- C/C++编程接口 C/C++ APIs

C/C++编程接口

现代机器学习框架（包括TensorFlow、PyTorch和MindSpore）主要依赖Pybind11来将底层的大量C和C++函数自动生成对应的Python函数，这一过程一般被称为Python绑定（Binding）。在Pybind11出现以前，将C和C++函数进行Python绑定的手段主要包括：

- Python的C-API。
- 简单包装界面产生器（Simplified Wrapper and Interface Generator，SWIG)。
- Python的ctypes
- CPython
- Boost::Python



Pybind11

C++代码

```cpp
#include <pybind11/pybind11.h>

int add(int i, int j) {
    return i + j;
}

PYBIND11_MODULE(example, m) {
    m.doc() = "pybind11 example plugin"; // optional module docstring

    m.def("add", &add, "A function that adds two numbers");
}
```

```
(pybind11Env) yonglang@B-P7M7MD6R-0122 example % python
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:14:23)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import example
>>> example.add(10,12)
22
>>>
```

编译c++代码后可直接在python中调用

Q&A

北京大学

计算图

Computational Graph

主讲人：王乐业　讲义：董豪

内容提要

- 目的 Motivation
- 计算图的基本构成 Computational Graph Basics
- 计算图的生成 Generating a Computational Graph
- 计算图的调度 Scheduling a Computational Graph

内容提要

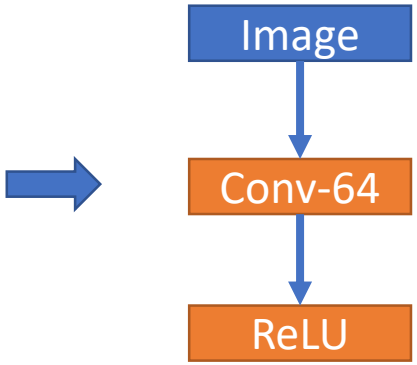
- 目的 Motivation
- 计算图的基本构成 Computational Graph Basics
- 计算图的生成 Generating a Computational Graph
- 计算图的调度 Scheduling a Computational Graph

目的

- 早期的AI计算框架

```
input: "data"
input_shape {
  dim: 1
  dim: 3
  dim: 224
  dim: 224
}

layer {
  name: "conv1_1"
  type: "Convolution"
  bottom: "data"
  top: "conv1_1"
  convolution_param {
    bias_term: true
    num_output: 64
    pad: 1
    kernel_size: 3
    stride: 1
  }
}

layer {
  name: "relu1_1"
  type: "ReLU"
  bottom: "conv1_1"
  top: "conv1_1"
}
```

Caffe网络结构定义



特点

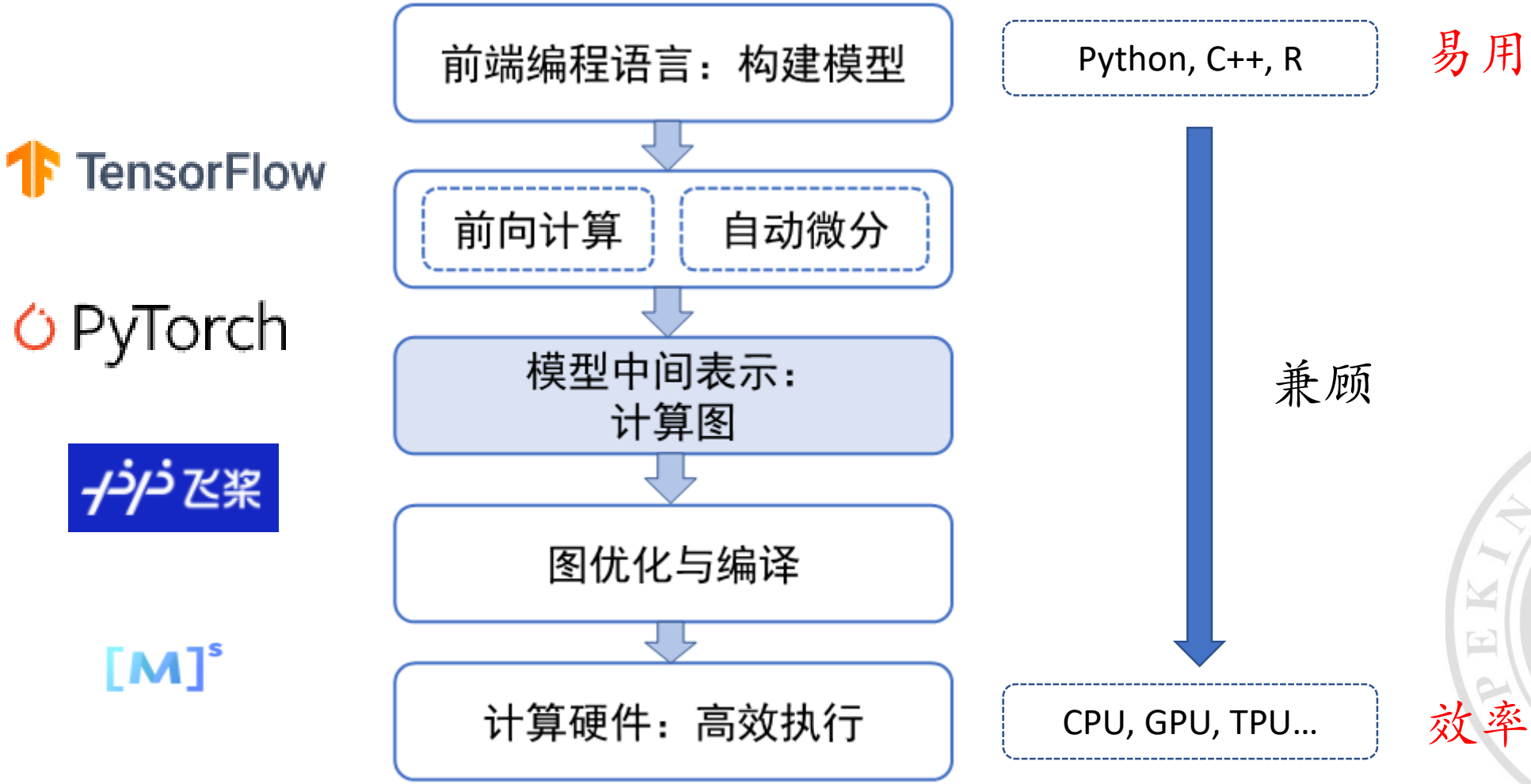
- 通过简单配置文件形式定义神经网络。
- 主要支持以CNN网络为主的CV类模型。

局限

- 对序贯网络支持较好，但是很难支持更加复杂的网络结构，比如RNN，GAN。
- 对梯度和参数更新策略更复杂的优化器支持较差。

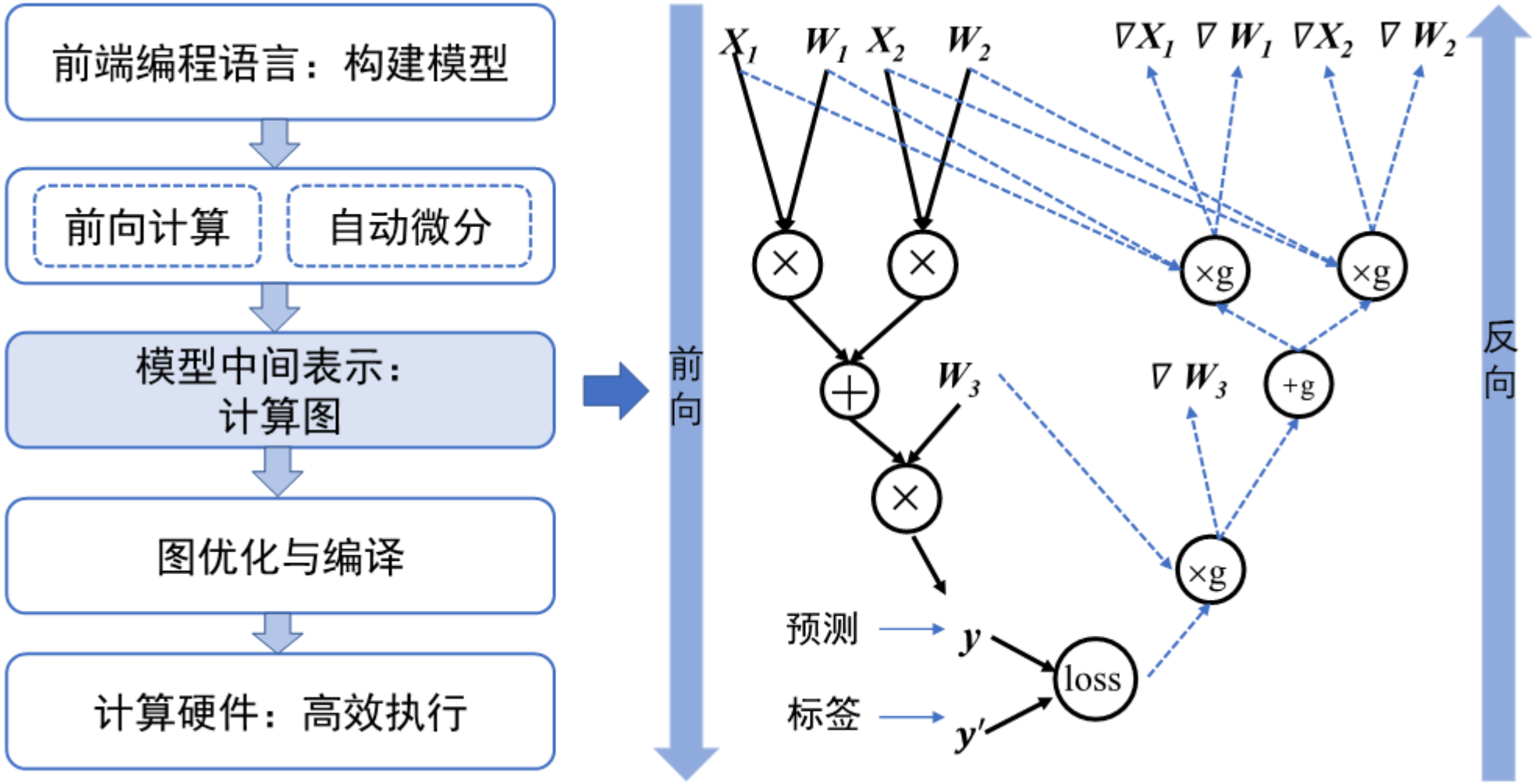
目的

- 基于计算图的AI计算框架



目的

- 基于计算图的AI计算框架



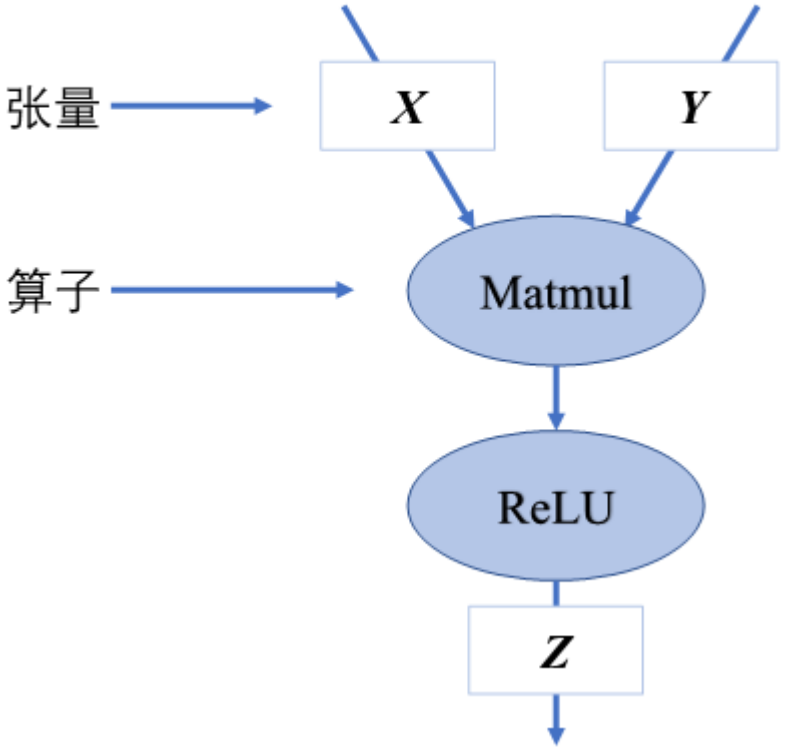
计算图用通用的数据结构来表达、理解神经网络模型。

Computational Graph Basics

计算图的基本构成

- 一个简单的计算图

$$Z = \mathrm{ReLU}(X \times Y)$$



- 计算图由张量（Tensor）和运算单元算子构成；
- 用节点来表示算子；
- 有向边来表示张量状态，同时也描述了计算间的依赖关系。

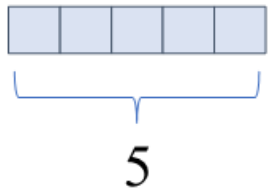
计算图的基本构成

- 张量

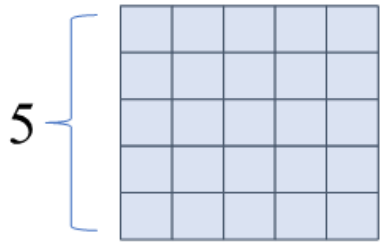
张量不仅存储数据，还需要存储张量的属性。

- 形状(shape) : 存储张量的每个维度的长度
- 秩或维数(dim) : 表示张量的轴数或者维数
- 数据类型(dtype) :表示存储的数据类型
- 存储位置(device) :创建张量时可以指定存储的设 备位置

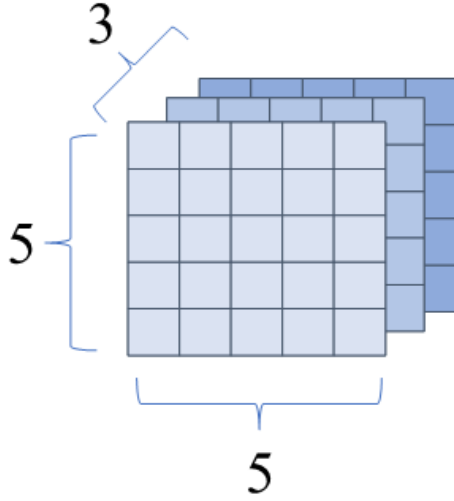
数据为5的标量



长度为5的
一维张量



形状为5×5的
二维张量



形状为5×5×3的
三维张量

计算图的基本构成

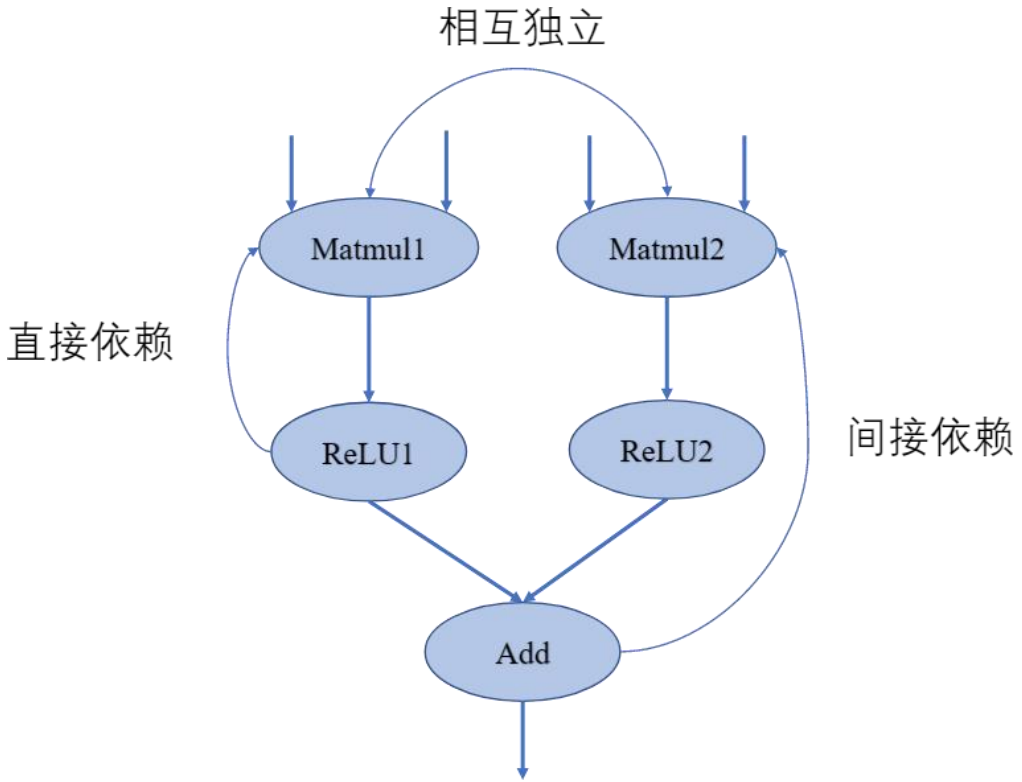
- 算子

算子是构成神经网络的基本计算单元。

- 数值计算: Add，Mul，Tanh，Div …
- 形态变换: Resize，Stack，Concat，…
- 神经网络算子: 损失函数Loss，激活函数Activation，优化器Optimizer, …
- 控制流算子：条件分支switch/if，循环 for/while，…

计算图的基本构成

- 计算图中的算子依赖



依赖关系：

- 相互独立：节点间无数据流动。
- 直接依赖：节点间数据的输出是下一个节点的输入。
- 间接依赖：节点间数据经过了一个或多个中间节点进行处理。

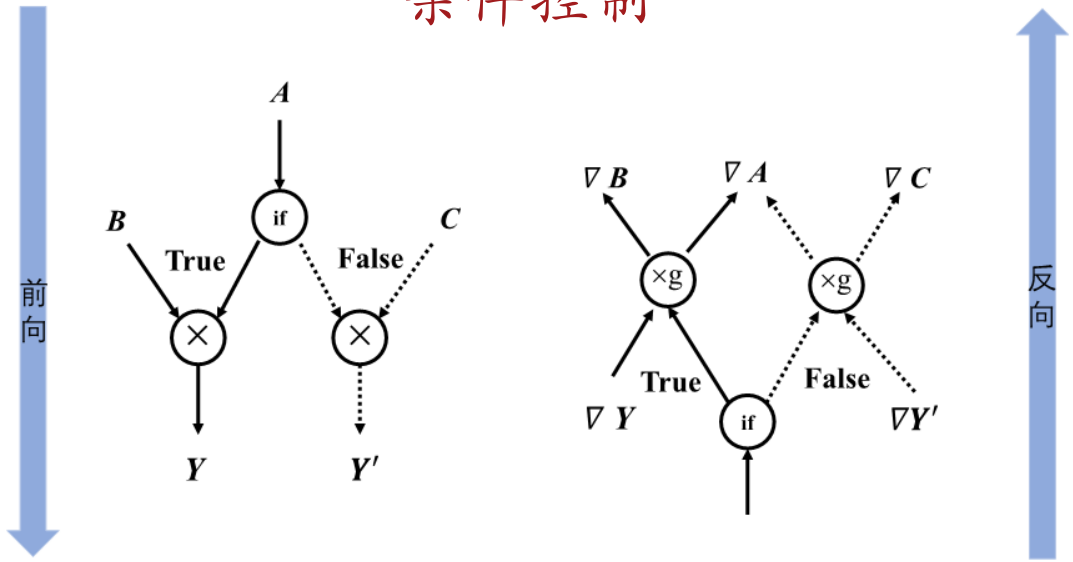
依赖关系影响了算子的执行顺序与并行情况。

计算图的基本构成

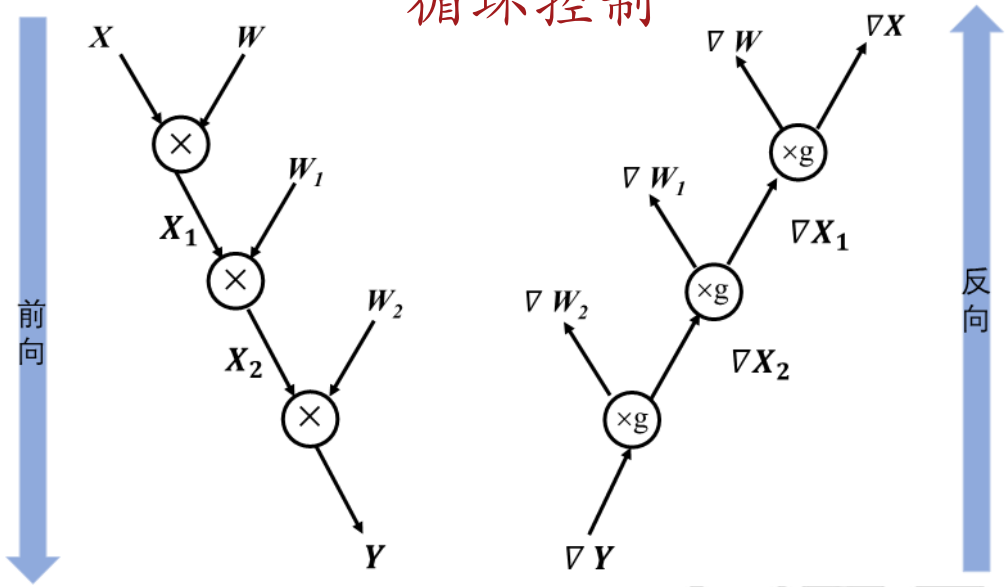
- 计算图中的控制流

控制流同时对计算图的前向计算与反向梯度计算产生影响。

条件控制

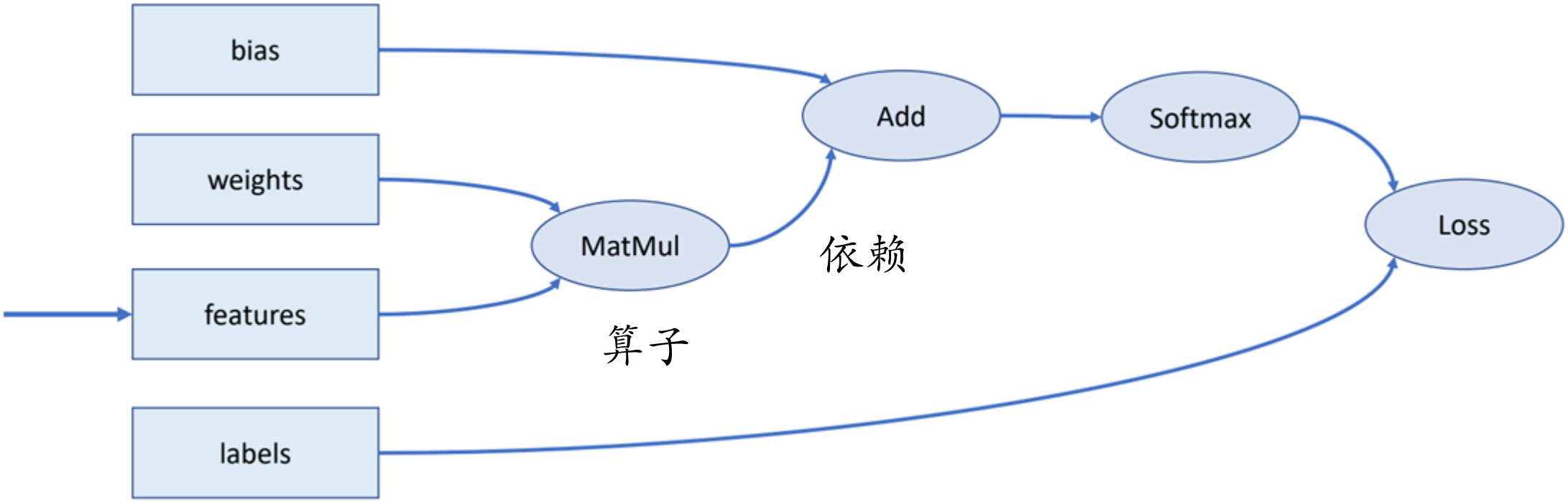


循环控制



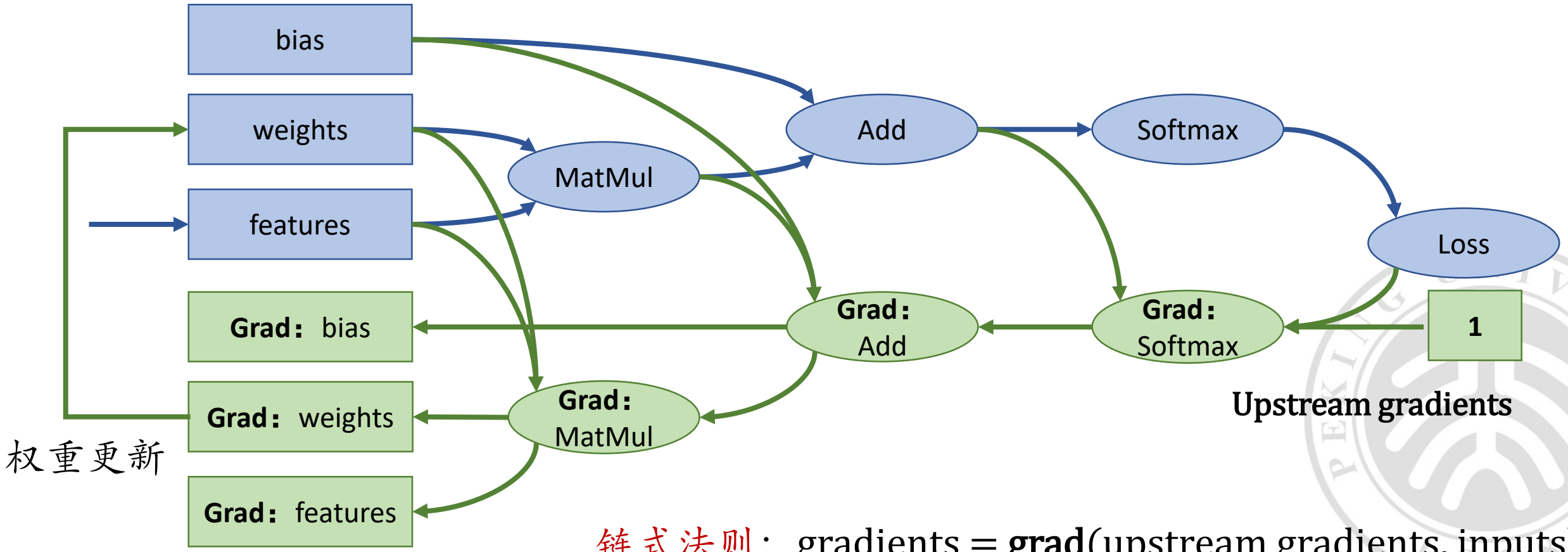
计算图的基本构成

- 计算图中的链式法则



计算图的基本构成

- 计算图中的链式法则



链式法则： gradients = **grad**(upstream gradients, inputs)

Generating a Computational Graph

计算图的生成

- 计算图的生成方式

静态生成 Graph

Define and run

动态生成 Eager

Define by run

计算图的生成

```
# matmul表示矩阵乘法算子
# relu表示ReLU算子
def model(X, flag): #X Input
    if flag>0:
        Y = matmul(W1, X) #W1 Parameter
    else:
        Y = matmul(W2, X) #W2 Parameter
    Y = Y + b
    Y = relu(Y)
    return Y
```

计算图的生成

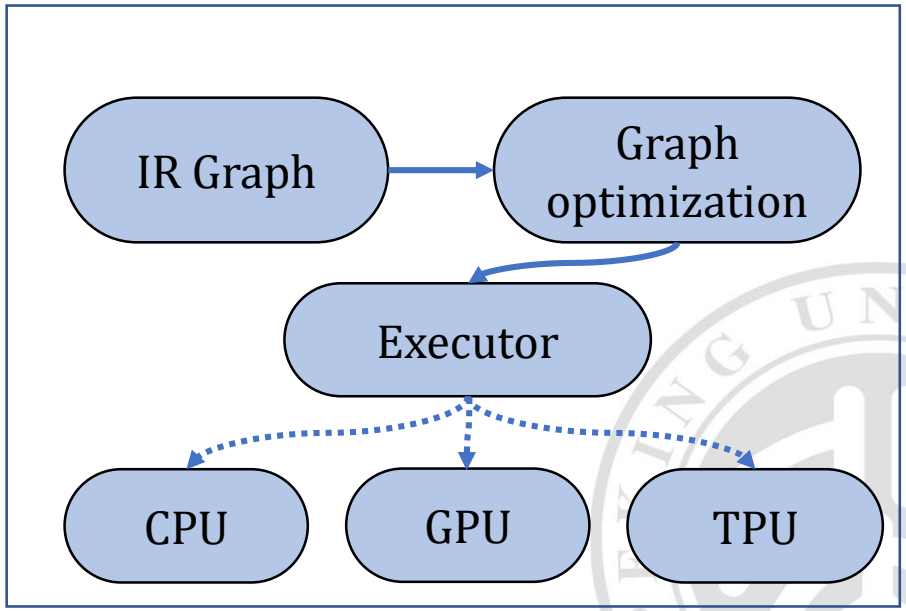
- 静态生成方式

Python front-end

```
# matmul表示矩阵乘法算子
# relu表示ReLU算子
def model(X, flag): #X Input
    if flag>0:
        Y = matmul(W1, X) #W1 Parameter
    else:
        Y = matmul(W2, X) #W2 Parameter
    Y = Y + b
    Y = relu(Y)
    return Y
```

compile

C++ back-end



IR： Intermediate Representation

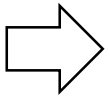
计算图的生成

- 静态生成方式

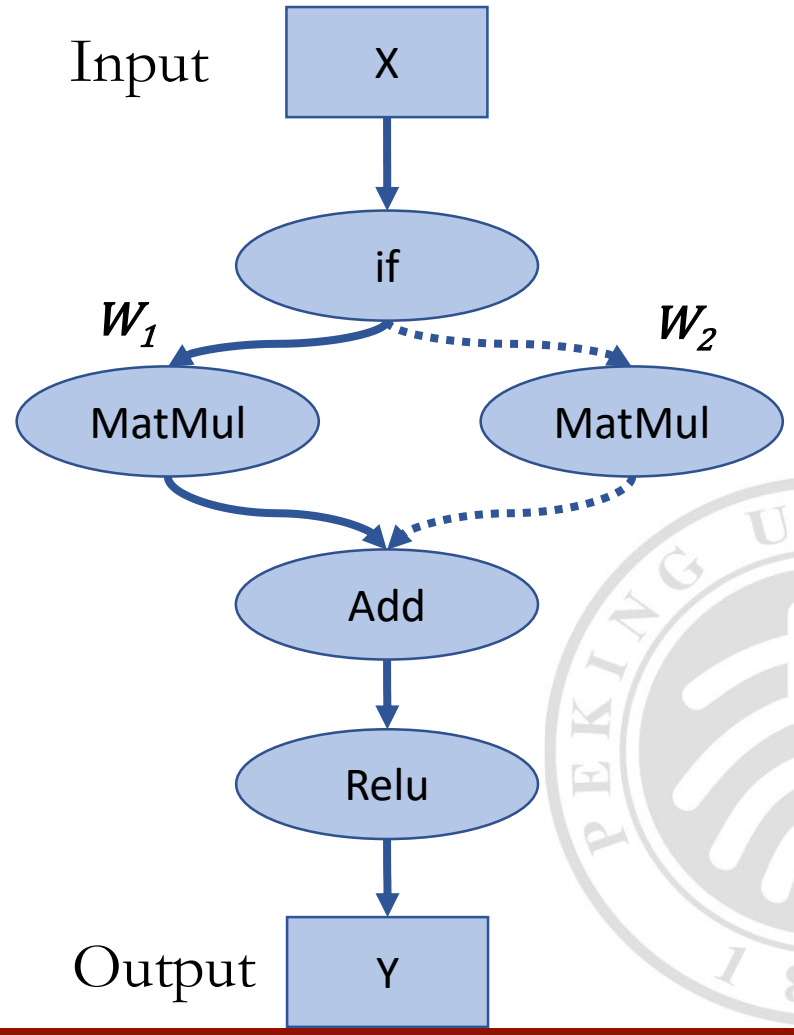
Python front-end

```
# matmul表示矩阵乘法算子
# relu表示ReLU算子
def model(X, flag): #X Input
    if flag>0:
        Y = matmul(W1, X) #W1 Parameter
    else:
        Y = matmul(W2, X) #W2 Parameter
    Y = Y + b
    Y = relu(Y)
    return Y
```

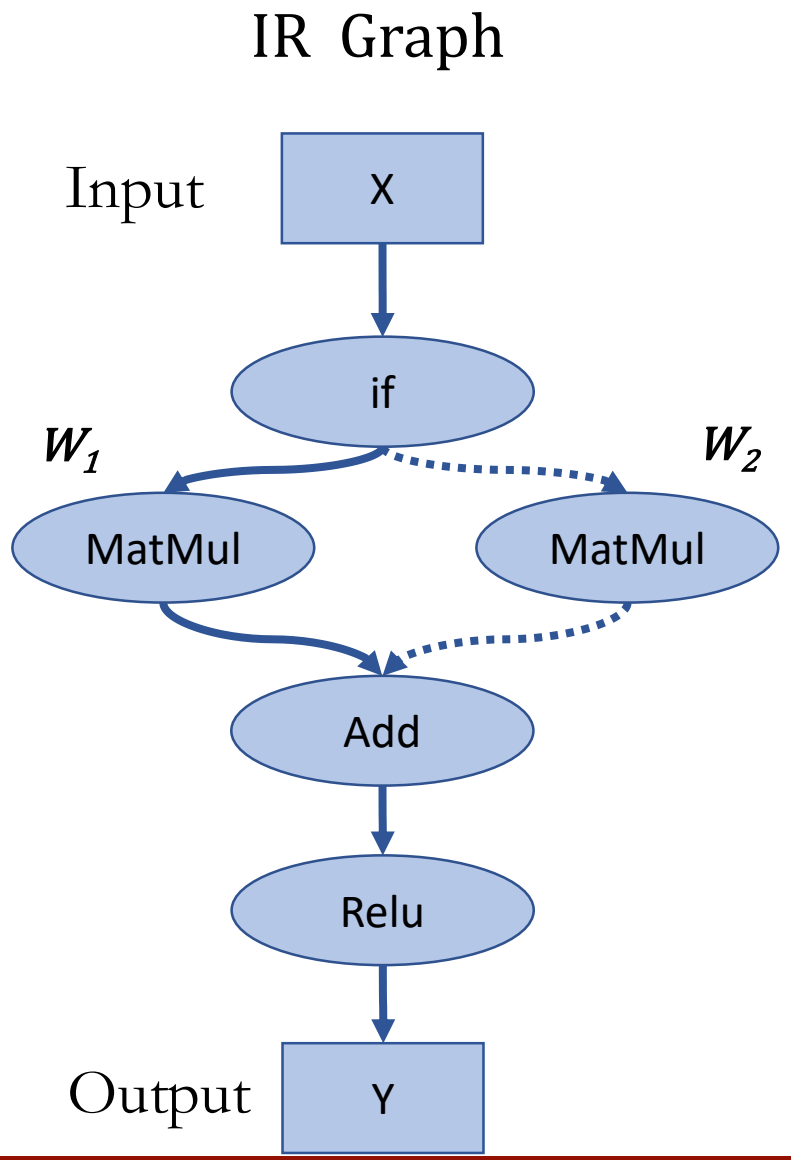
compile



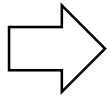
IR Graph



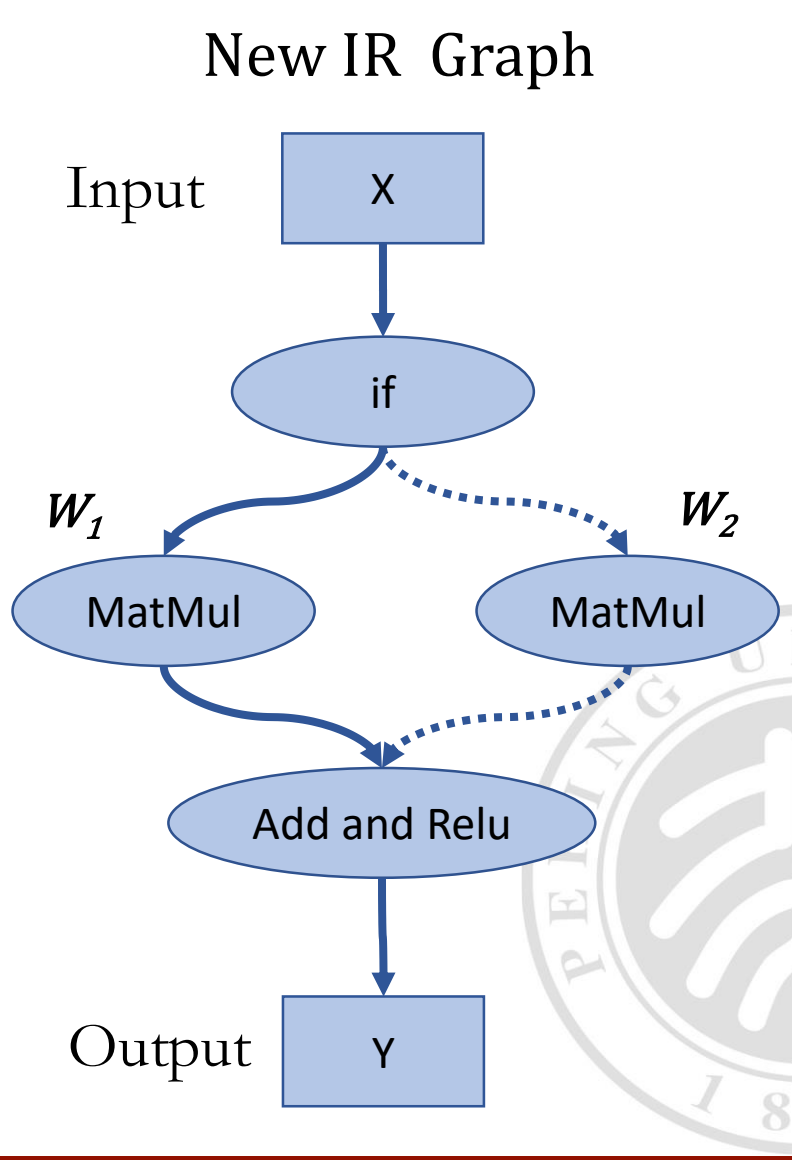
计算图的生成



Optimization



提高计算效率
降低内存消耗

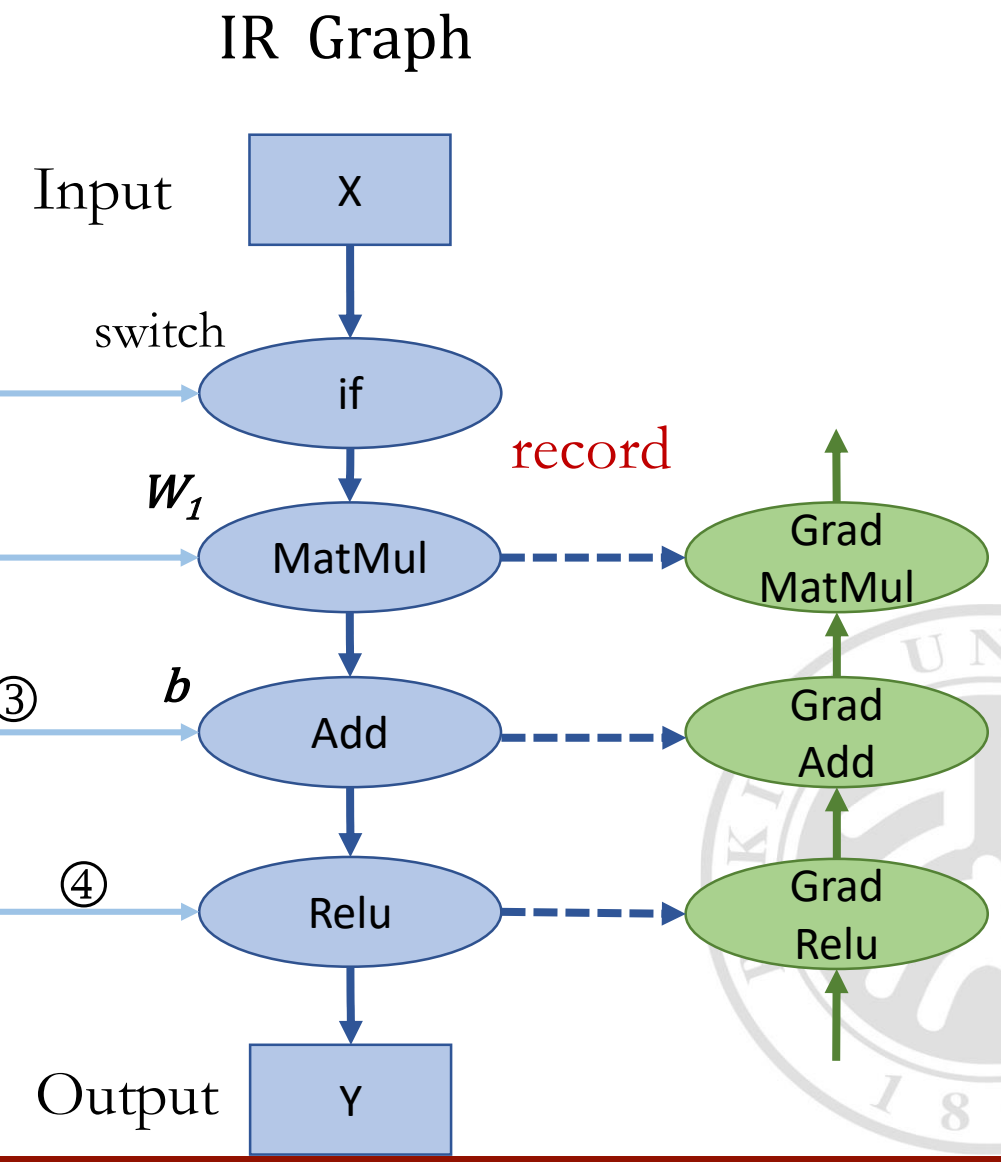


计算图的生成

- 动态生成方式

Python front-end

```
# matmul表示矩阵乘法算子
# relu表示ReLU算子
def model(X, flag): #X Input
    if flag>0:
        Y = matmul(W1, X) #W1 Parameter
    else:
        Y = matmul(W2, X) #W2 Parameter
    Y = Y + b
    Y = relu(Y)
    return Y
```



计算图的生成

特性	静态图	动态图
即时获取中间结果	否	是
代码调试难易	难	易
控制流实现方式	特定的语法	前端语言语法
性能	优化策略多，性能更佳	图优化受限，性能较差
内存占用	内存占用少	内存占用相对较多

计算图的生成

- 动静结合

TensorFlow　PyTorch　飞桨　[M]^{s}

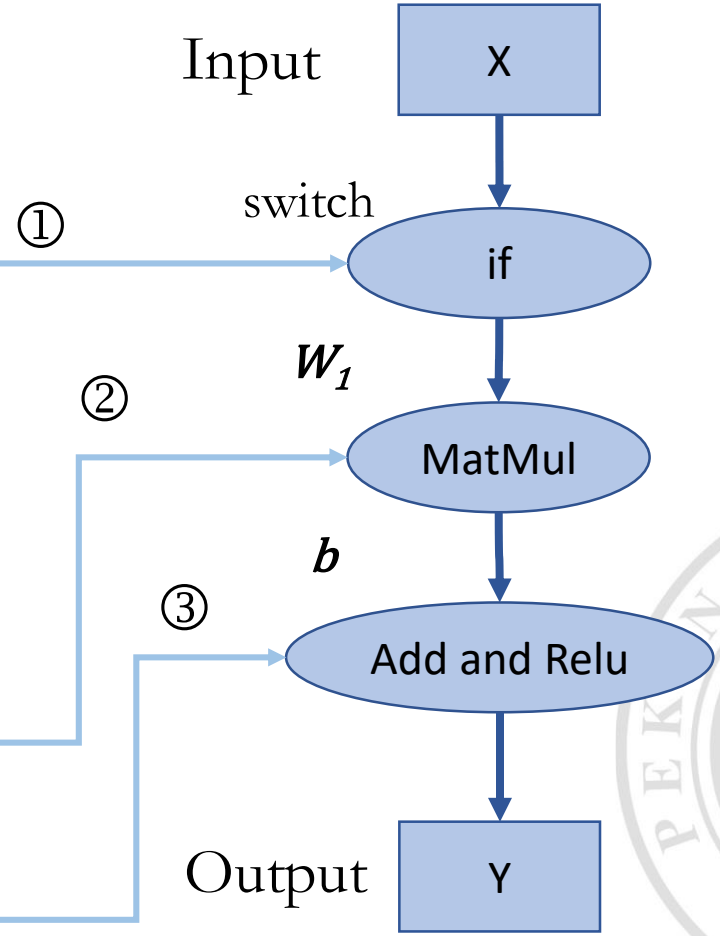
- **基于追踪转换（Trace）**：以动态图模式执行并记录调度的算子，构建和保存为静态图模型。
- **基于源码转换（Compile）**：分析前端代码来将动态图代码自动转写为静态图代码，并在底层自动帮用户使用静态图执行器运行。

计算图的生成

- 动静结合

Python front-end

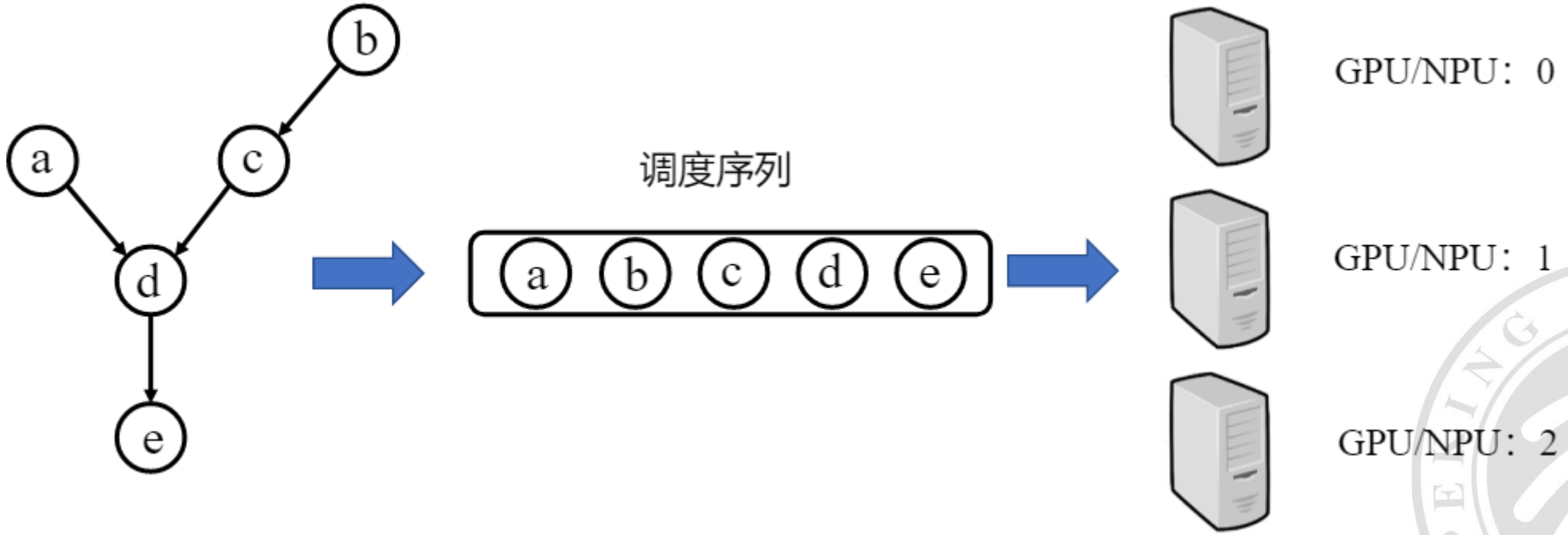
```
# matmul表示矩阵乘法算子
# relu表示ReLU算子
@tf_function #tensorflow源码转换装饰器
def add_and_relu(Y, b):
    Y = Y + b
    Y = relu(Y)
    return Y

def model(X, flag): #X Input
   if flag>0:
       Y = matmul(W1, X) #W1 Parameter
   else:
       Y = matmul(W2, X) #W2 Parameter
Y = add_and_relu(Y, b)
   return Y
```



Scheduling a Computational Graph

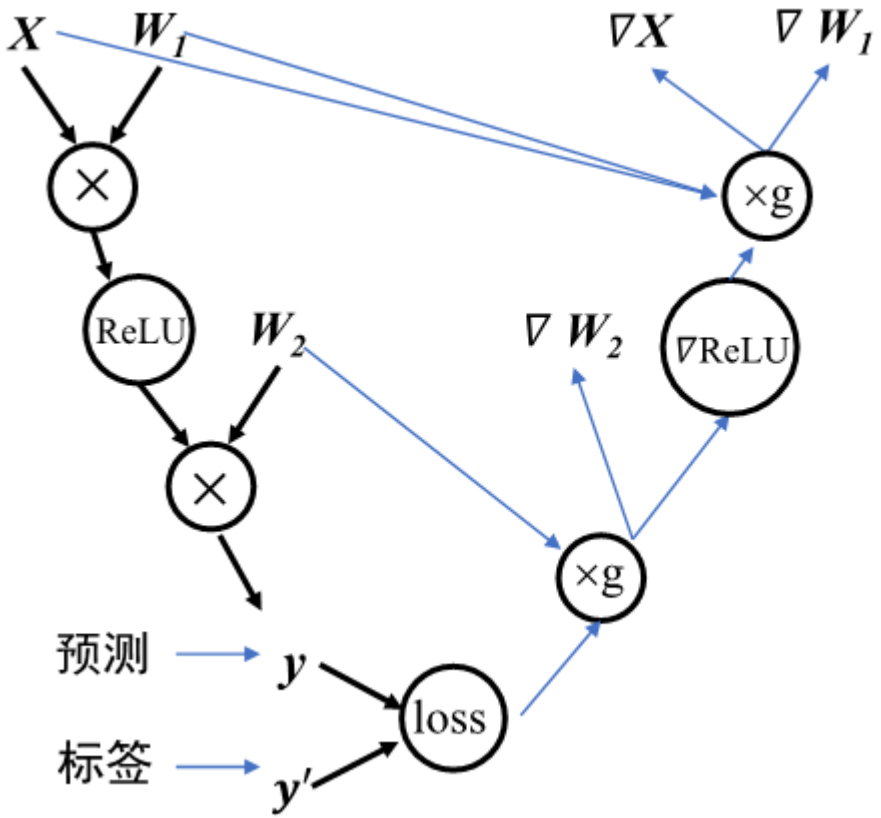
计算图的调度

- 算子调度执行

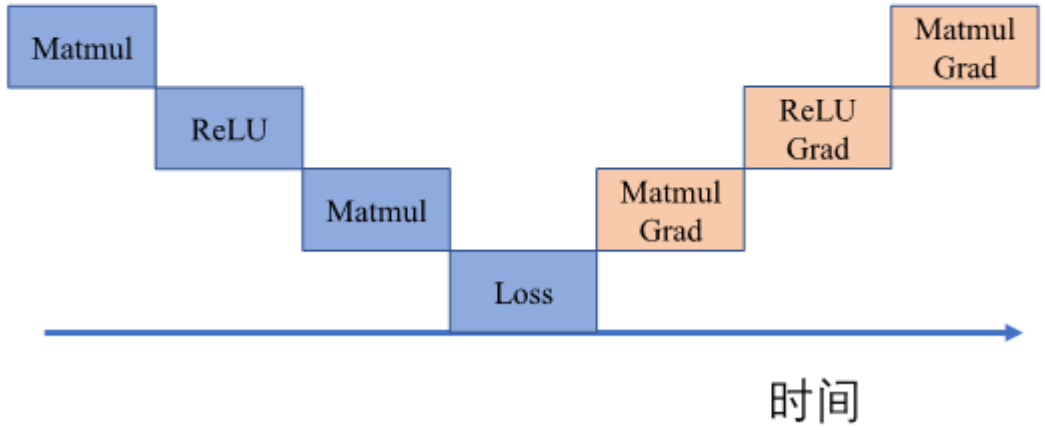


计算图的调度

- 算子调度执行



串行

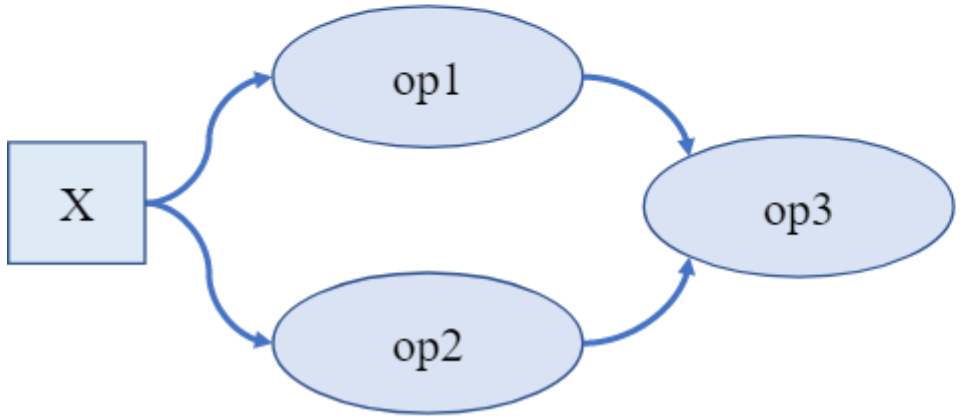


计算图的调度

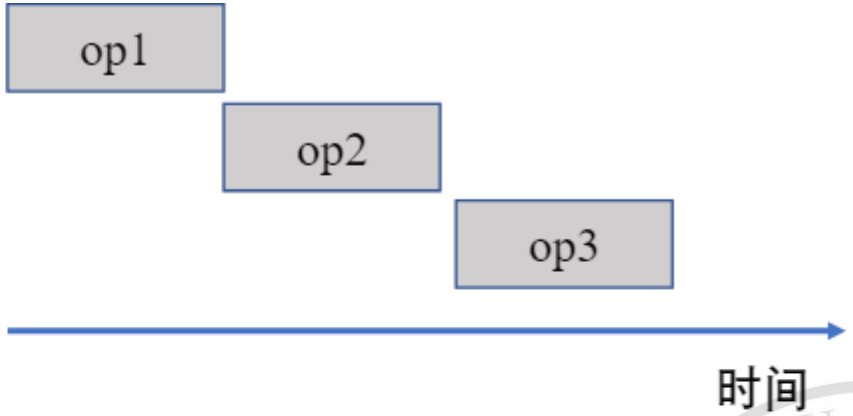
- 算子调度执行

```
def model(x):
    y1 = op1(x)
    y2 = op2(x)
    return op3(y1, y2)
```

Equivalent Graph



串行



并行

Q&A

选择题

1、基于计算图的机器学习系统优点包含（多选）：

A. 便于分析算子依赖　　B. 统一模型中间表达

C. 分析变量生命周期　　D. 便于优化模型执行

选择题

2、一张高为96像素、宽为96像素的RGB彩色三通道图像保存为张量时，属性不正确的一项是：

A. shape=[96,96,3]

B. dim=3

C. dtype=bool

D. device=cpu

选择题

3、算子间的正确依赖关系不应包含下列哪一个：

A. 直接依赖　　B. 间接依赖

C. 相互独立　　D. 循环依赖

选择题

4、下列不属于静态图优势的选项是:

A. 部署方便

B. 优化性能

C. 调试简单

D. 内存占用少

选择题

5、下列不属于动态图特点的选项是：

A. 内存优化

B. 生成临时拓扑结构

C. 调试简单

D. 无法获取模型全局信息

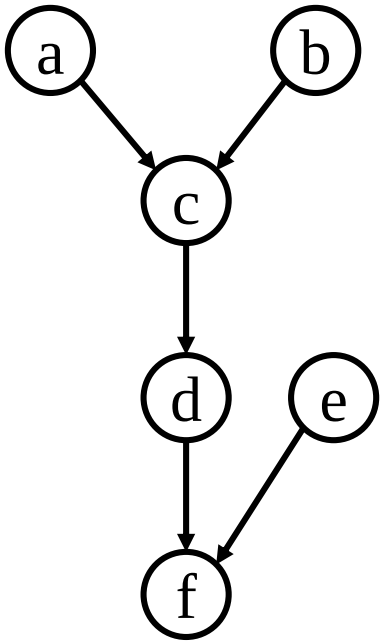
选择题

6、使用基于追踪转换的动态图转静态图过程中，不可能出现的步骤是：

A. 动态图方式执行代码　　B. 记录未使用的分支结构

C. 记录网络前向拓扑结构　　D. 记录反向梯度算子

选择题



7、右图算子调度执行顺序错误的一项是：

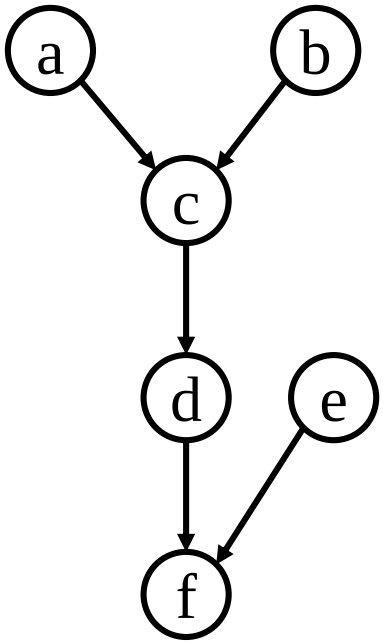
A. abcdef

B. beacdf

C. ebcadf

D. aebcdf

选择题



8、右图算子调度若需要并行加快计算，最多可同时执行几个算子：

A. 2

B. 3

C. 4

D. 5