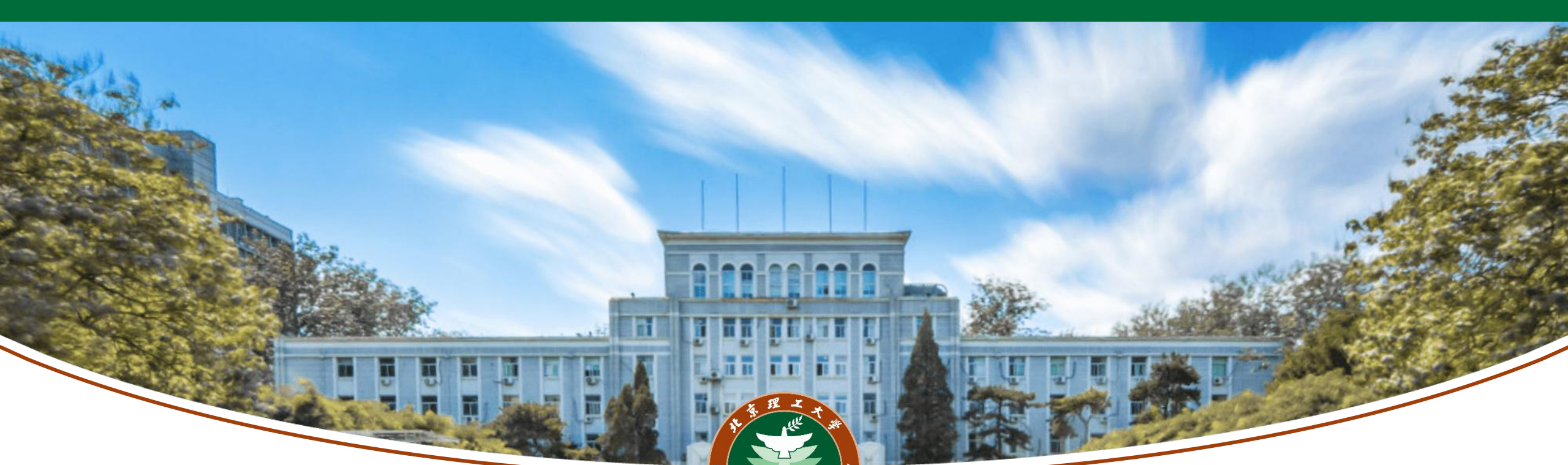




深度学习平台

小组成员：张俊辉、刘陆、刘欣远、穆育松、刘思辰、依西降参

时 间：2021/9/18



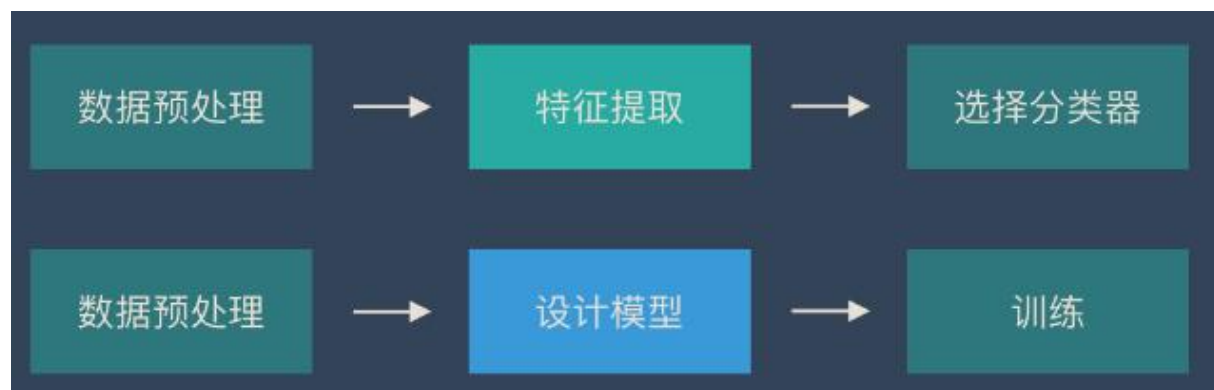
目录

CONTENTS

- 1 概述
- 2 深度学习平台简介
- 3 深度学习平台实践
- 4 总结

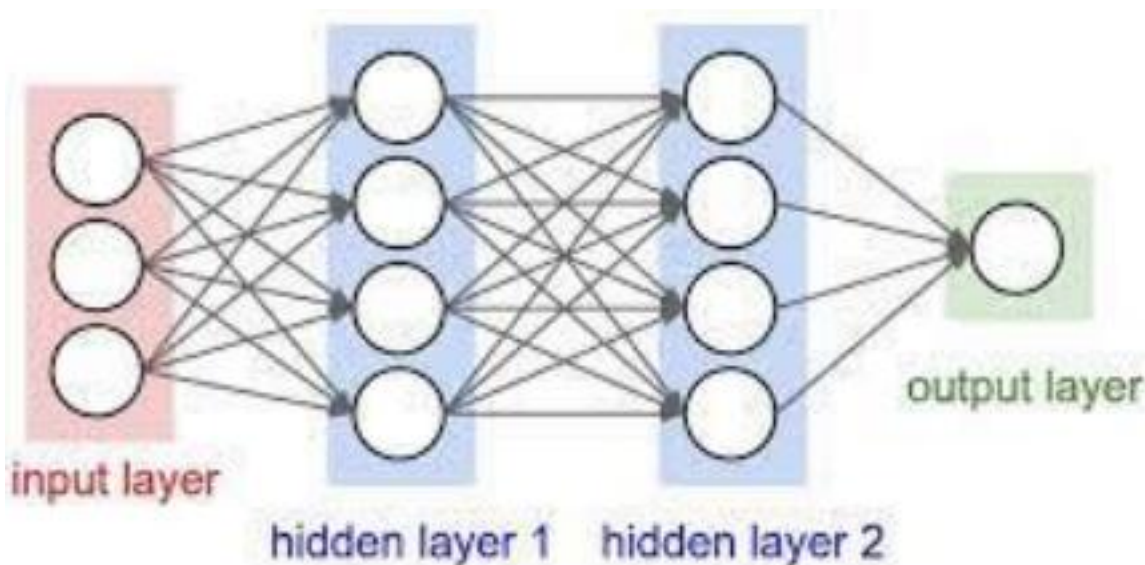


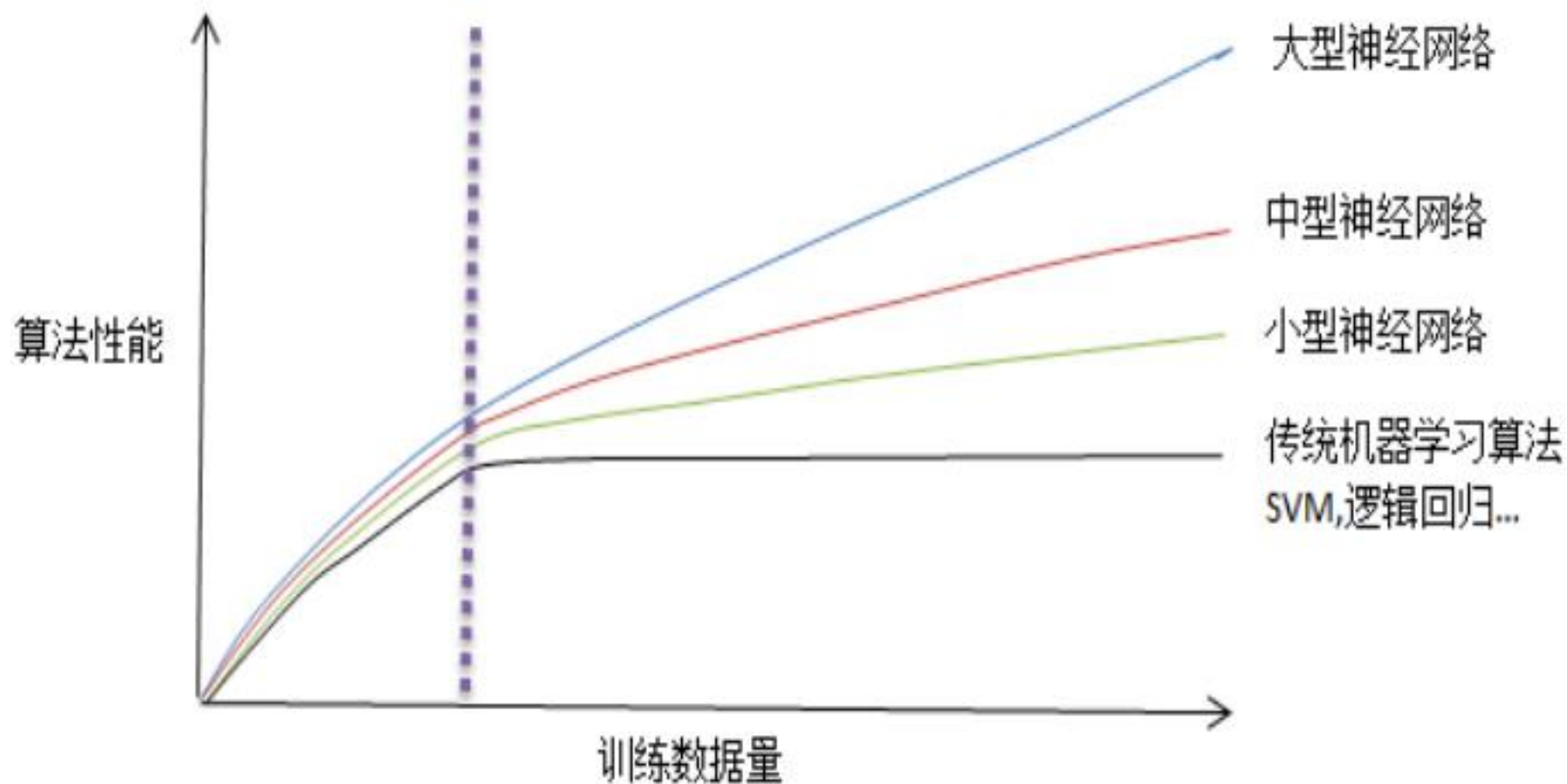
概述



● 深度学习(Deep Learning):

- 一种实现机器学习的技术，是机器学习最重要的分支
- 来源于人工神经网络的研究，它的模型结构是一种含多Hidden layer的神经网络
- 通过组合底层特征形成更加抽象的高层特征





数据、算法、计算力



优点：

1. 学习能力强
2. 覆盖范围广、适应性好
3. 数据驱动、上限高
4. 可移植性好

缺点：

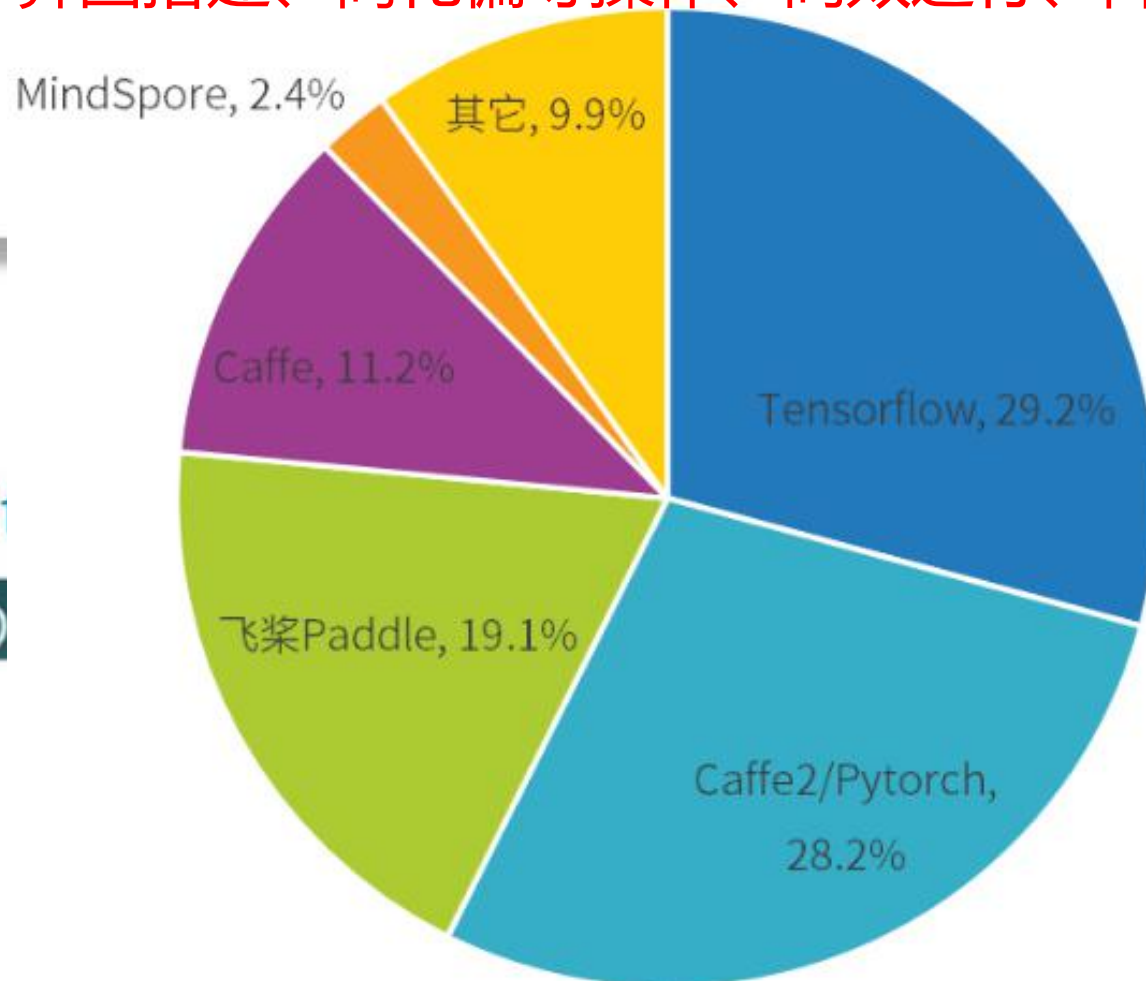
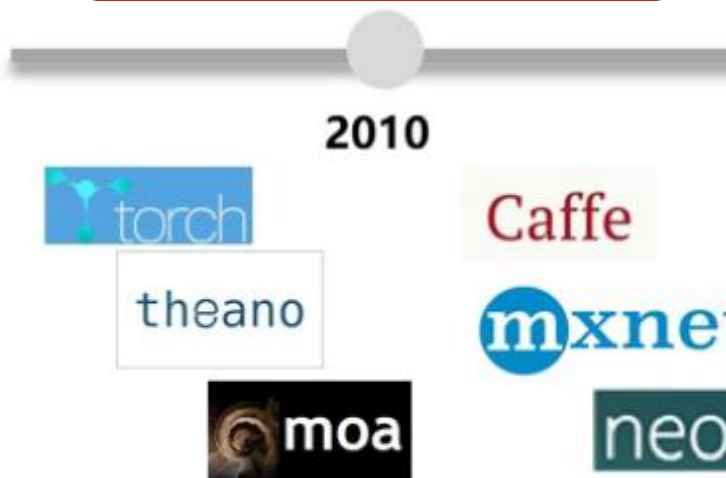
1. 计算量大、便携性差
2. 硬件需求高
3. 模型设计复杂
4. 没有“人性”、容易存在偏见

平台优势

中国深度学习开源框架用户份额，2021H1

利用硬件资源、简化计算图搭建、简化偏导操作、高效运行、降低入门门槛

发展状况



DL



深度学习平台简介

- 1 Tensorflow
- 2 PyTorch
- 3 PaddlePaddle
- 4 其它平台



1.TensorFlow

TensorFlow是谷歌开源的一款**深度学习框架**，首次发布于2015年，TensorFlow2发布于2019年。

TensorFlow是一个采用**数据流图**进行数值计算的开源软件库，常被应用于各种感知、语言理解、语音识别、图像识别等多项机器深度学习领域。



2015年

谷歌大脑团队发布了TensorFlow并对代码开源

2016年

支持分布式计算；同时增加了对Windows环境的支持

2017年

TensorFlow的1.0版本正式发布，更注重API的稳定性

2018年

TensorFlow生态系统的核心组件发布，更好地建成了生态环境

2019年

TensorFlow的2.0版本发布，简化了模型开发流程，并提供了更强大的跨平台能力



TensorFlow2.0开启默认模式是
采用了动态图机制



TensorFlow2.0使用函数而不是
会话



TensorFlow2.0扩展了一些更高
级的API



TensorFlow1.0更能发挥硬件的
优势

可移植性



TensorFlow可以在CPU和GPU上运行，可以在台式机，服务器，移动设备上运行

多语言支持



支持多种流行语言，如python，java，c++等

可视化训练过程

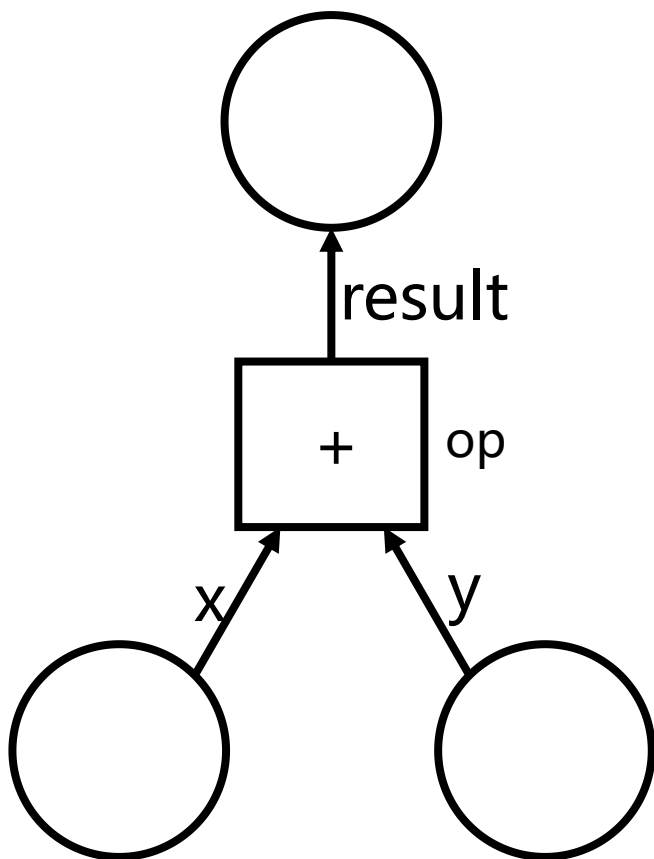


TensorFlow使用TensorBoard对训练过程，模型，数据图等进行可视化

高度的灵活性



只要能够将计算表示成为一个数据流图，那么就可以使用TensorFlow

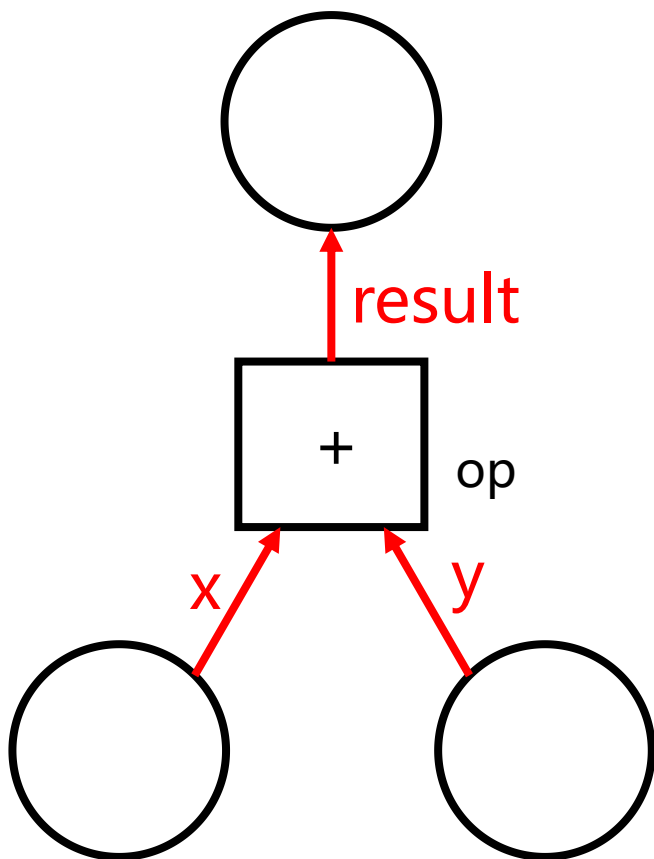


数据流图

TensorFlow中的每一个计算都代表着一个**数据流图**。这个图有两个元素：

- 一系列的tf.Operation，代表**计算单位**

- 一系列的tf.Tensor，代表**数据单位**

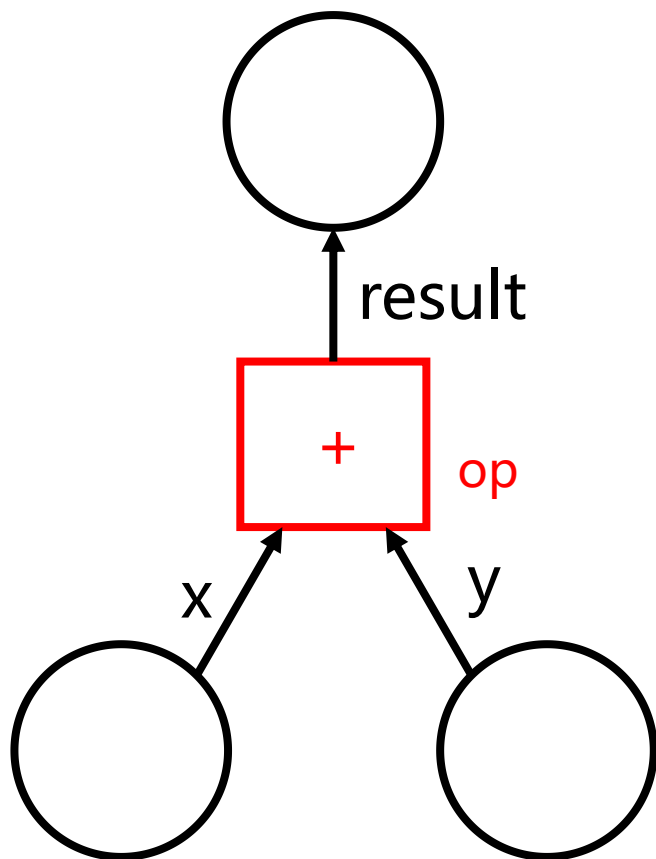


在TensorFlow的数据流图中，线（Edges）表示在节点间相互联系的多维数据数组，即**张量（Tensor）**。一个张量中主要保存了三个属性：**名字（name）**、**维度（shape）**、**类型（type）**。

创建恒定的张量 $x = [1, 3, 6]$, $y = [1, 1, 1]$

```
import tensorflow as tf

x = tf.constant([1, 3, 6])
y = tf.constant([1, 1, 1])
```



在TensorFlow的数据流图中，节点（Nodes）表示操作，即operation

定义操作单元

```
import tensorflow as tf

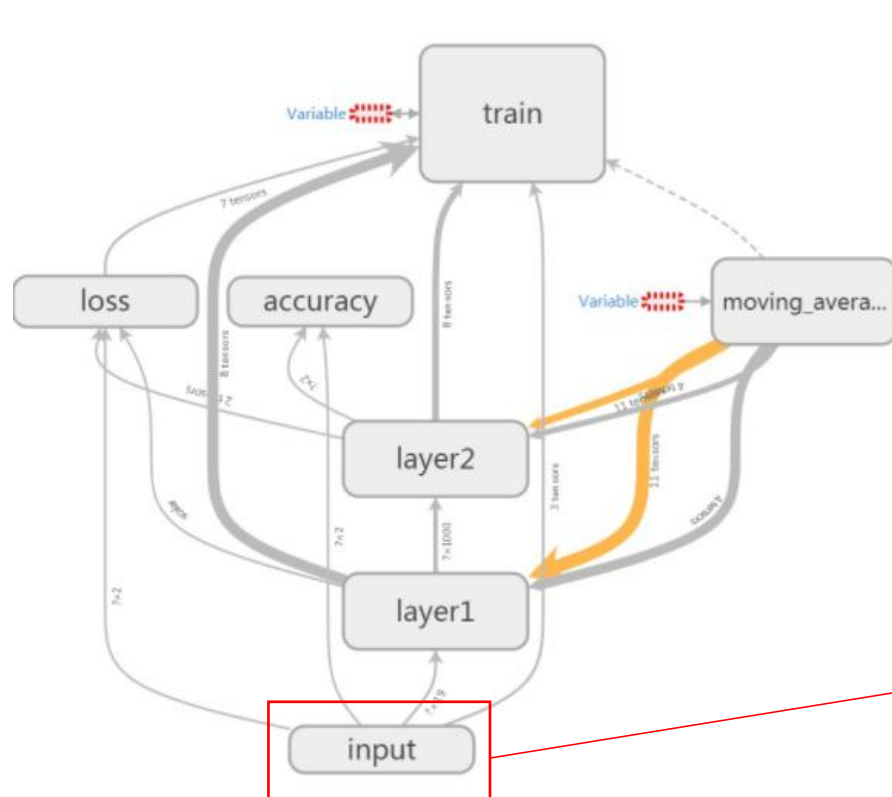
x = tf.constant([1, 3, 6])
y = tf.constant([1, 1, 1])

op = tf.add(x, y)
```

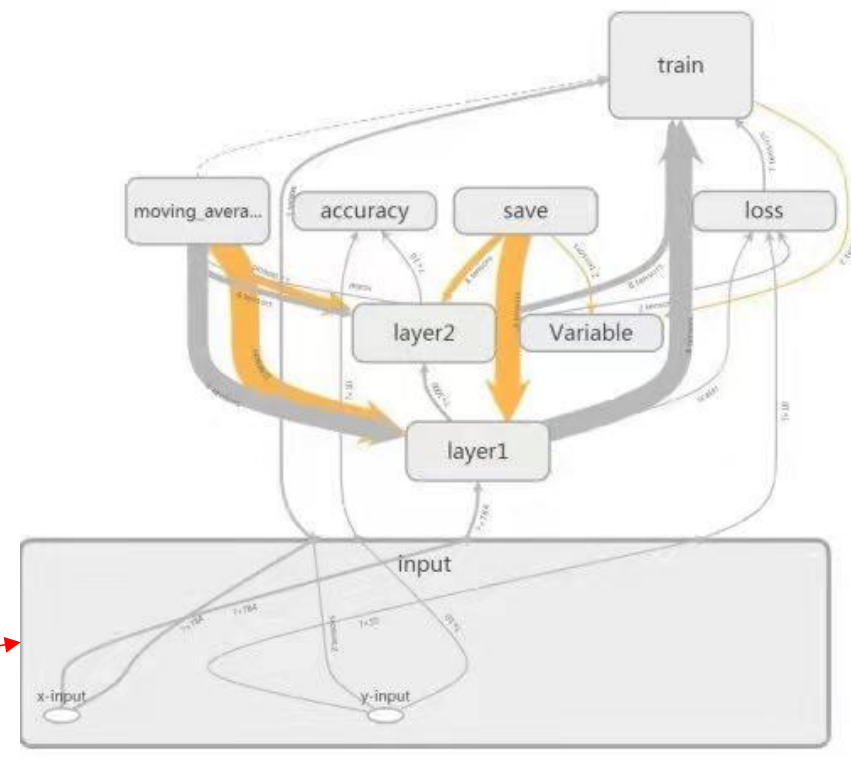


TensorBoard是TensorFlow内置的一个**可视化工具**，它通过将TensorFlow程序输出的日志文件的信息可视化,使得TensorFlow程序的理解、调试和优化更加简单高效。

利用TensorBoard可以更直观的看见整个神经网络的结构，观察模型的结构和训练过程中各个参数的变化。



使用TensorBoard对数据图结构可视化



input节点的深层信息图

TensorFlow Lite是为了解决TensorFlow在移动平台和嵌入式端过于臃肿而定制开发的轻量级解决方案，允许用户在多种设备上运行。TensorFlow Lite模型具有占用空间小，低延迟的特点。

TensorFlow



TensorFlow Lite



以移动端为例

在高性能的电脑上用数据训练出神经网络的参数



用TensorFlow转化器把训练好的模型转化成TensorFlow lite的格式



在手机移动端上，使用TensorFlow lite解析器读取该模型，将模型部署在移动端上并高效的运行它。

英语流利说使用TensorFlow
训练 AI 老师

Airbnb 用 TensorFlow 做
图片分类。系统自动把房东上
传的图片分类成客厅、厨房和
卧室

基于 TensorFlow 的深度神
经网络解决大规模计算问题，
研究极端天气以及它们对地球
的影响



京东基于TensorFlow 做
大规模推荐



可口可乐用 TensorFlow 来
解决抽奖码扫码输入的问题



安徽中医药大学的一位老师，
在用 TensorFlow 来识别
中药材的切片



2. PyTorch

Py + Torch

PyTorch是一个基于Torch的Python**开源机器学习框架**，可加快从研究原型设计到生产部署的过程。

通常被用于计算机视觉、自然语言处理等多种应用程序。

FROM
RESEARCH TO
PRODUCTION

An open source machine learning framework that accelerates the path from research prototyping to production deployment.



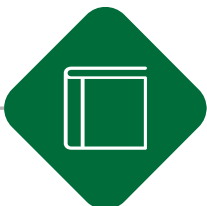
2017年1月正式发布PyTorch

2018年4月更新0.4.0版, **支持Windows系统**

2019年5月更新1.1.0版, **支持TensorBoard**

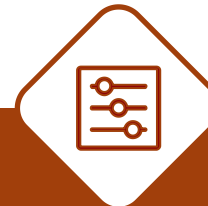
2019年8月更新1.2.0版, 更新**torchvision**

2021年6月更新1.9.0版



简洁优雅

tensor→autograd
→nn.Module



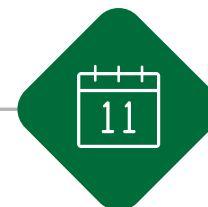
高效快速

保持灵活性的同时
不以速度为代价



易于使用

API设计来源于Torch
最优雅的面向对象设计的框架



社区活跃

作者亲自维护论坛,
FAIR的支持确保持续的开发更新

Tensor

张量是PyTorch里的基本运算单位，
类似于numpy中的ndarray，但Tensor可以使用GPU加速运算。

互相转换

```
► # a --> Tensor  
a = torch.rand(3, 2)  
# Tensor --> numpy  
numpy_a = a.numpy()  
# numpy --> Tensor  
b = torch.from_numpy(numpy_a)|
```

Autograd

Autograd模块实现了深度学习算法中的反向传播求导，在Tensor上的所有操作，Autograd都能为它们自动计算微分。

操作设置

```
▶ x = torch.randn(5, 5, requires_grad = True)
```

通过设置**`requires_grad = True`**来对该张量进行自动求导。计算完成后，调用**`backward()`**方法计算梯度，并保存在**`grad`**中。

OpenAI的大规模文本处理网络GPT-3

可以自动续写代码：

MODEL: GPT-J-6B

[Model on Github](#)

Prompt List

Try a classic prompt
evaluated on other
models

```
import cv2  
  
file_name="test.jpg"  
  
#open it with cv2 and detect the edge
```

TOP-P

0.9

Temperature

0.8

Run the model! ⚡

Powered by [Neuro](#)

RESULT

```
import cv2
```

```
file_name="test.jpg"
```

```
#open it with cv2 and detect the edge
```

```
img=cv2.imread(file_name)
```

```
edges=cv2.Canny(img, 100, 200)
```

```
#get edges from Canny in grayscale image
```

```
cv2.imshow("edges", edges)
```

```
#draw the edges
```

```
cv2.imshow("img", img)
```

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

```
#save the detected edge image and name as "edge.jpg"
```

```
cv2.imwrite("edge.jpg", edges)
```

<https://6b.eleuther.ai/>

DeepMind的AlphaFold2，使用PyTorch进行蛋白质结构预测。

**AlphaFold: a solution to
a 50-year-old grand
challenge in biology**





3.PaddlePaddle

PaddlePaddle(飞桨), 是2016年8月百度研发的一款深度学习框架, 以“致力于让深度学习技术的创新和应用更简单”为设计目的。







高度封装

飞桨2.0将自身的API进一步封装，
进一步提升了飞桨的易学易用性



动静转换

飞桨2.0的高层API支持动静转换，
只需要一个函数装饰器
`@paddle.jit.to_static`



大量官方模型库

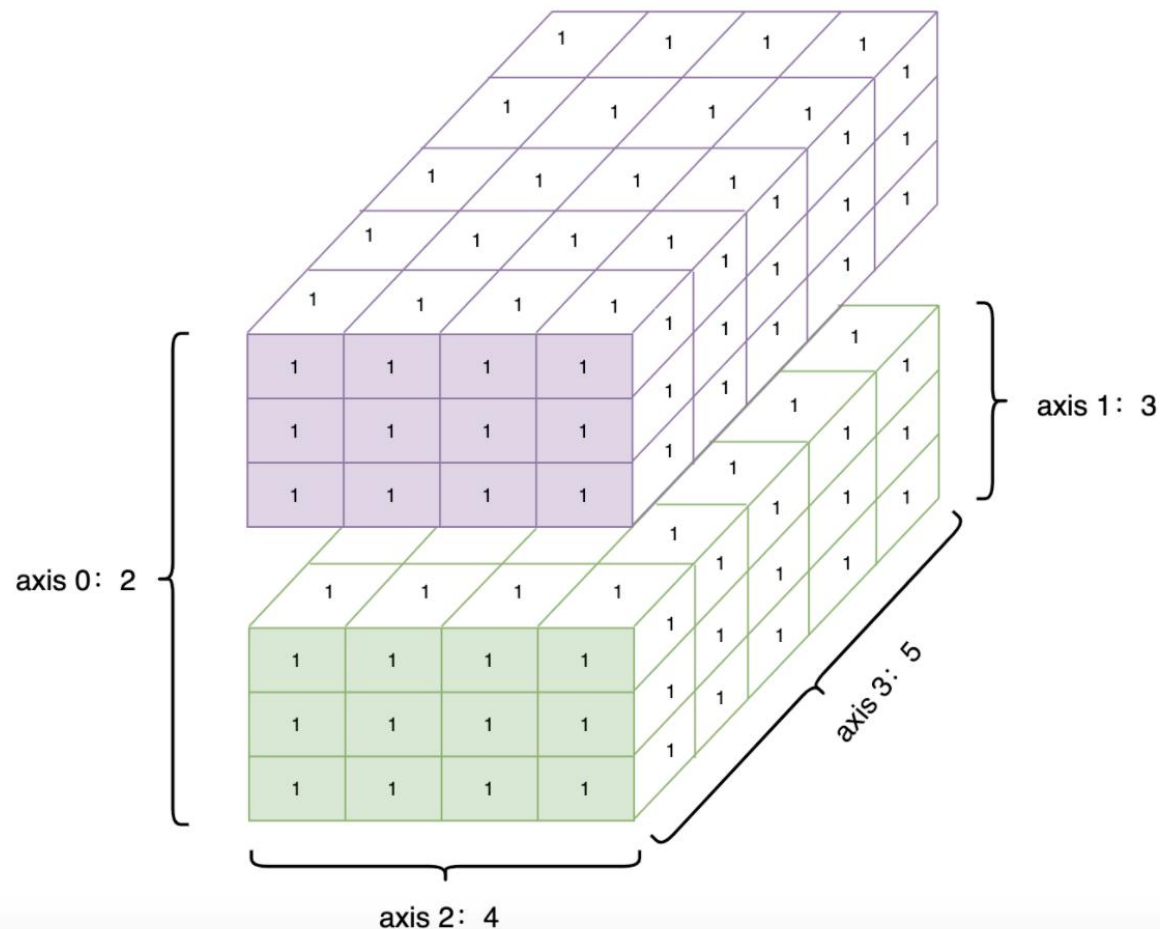
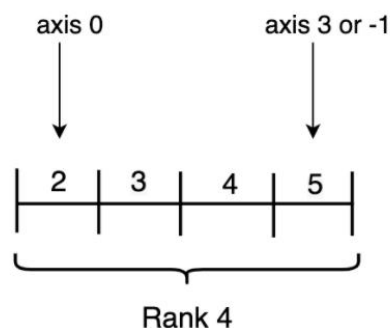
于智能视觉、智能文本处理、智能推荐等部分有着官方模型，稍加修改即可投入使用



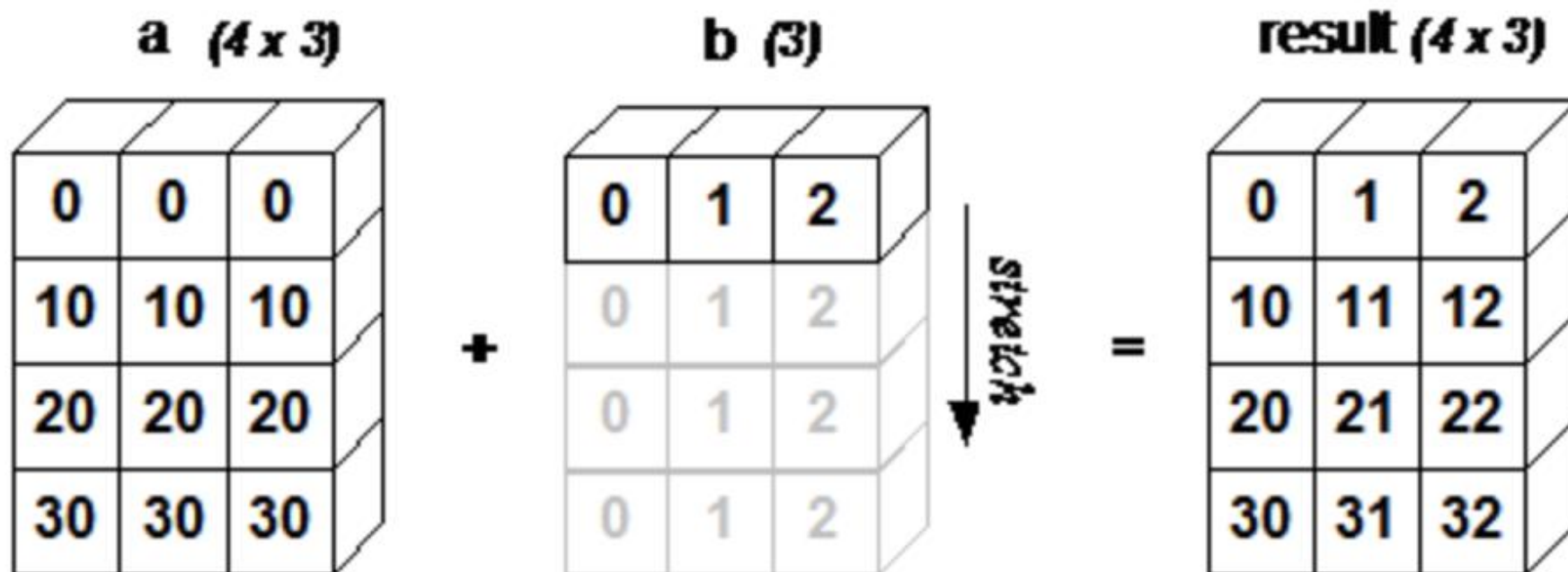
可视化

飞桨可视化分析工具VisualDL，
可帮助用户更清晰直观地理解深度学习模型训练过程及模型结构

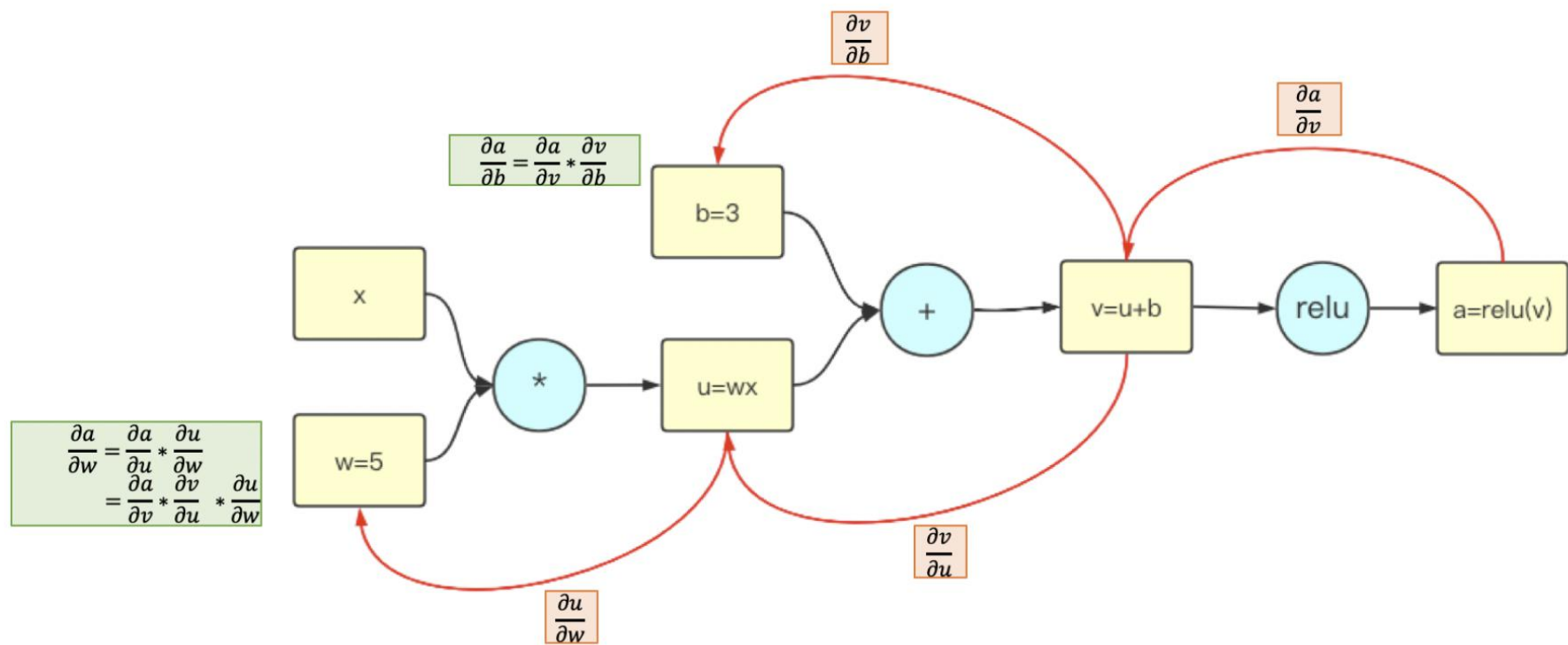
- **shape**: 描述tensor的每个维度上元素的数量
- **ndim**: tensor的维度数量
- **axis**: tensor某个特定的维度
- **size**: tensor中全部元素的个数



飞桨和其他框架一样，提供的一些API支持广播(broadcasting)机制，允许在一些运算时使用不同形状的张量，遵循Numpy的广播规则



飞桨的自动微分是通过trace的方式，记录前向OP的执行，并自动创建反向var和添加相应的反向OP，然后来实现反向梯度计算的。



动态图

动态图意味着计算图的构建和计算同时发生（define+by+run）。这种机制由于能够实时得到中间结果的值，使得调试更加容易，同时我们将大脑中的想法转化为代码方案也变得更加容易，对于编程实现来说更友好。

静态图

静态图则意味着计算图的构建和实际计算是分开（define+and+run）的。在静态图中，会事先了解和定义好整个运算流，这样之后再次运行的时候就不再需要重新构建计算图了（可理解为编译），因此速度会比动态图更快，从性能上来说更加高效。

动态图转静态图

飞桨支持用户使用动态图编写组网代码，PaddlePaddle会对用户标识的代码进行分析，将其转换为静态图网络结构，兼顾了动态图易用性和静态图部署性能两方面优势。

该应用基于百度飞桨开源的口罩人脸检测及分类模型，可对未佩戴口罩的人员进行识别并提示，识别准确率达96.5%以上。

口罩检测

金澍科技应用飞桨深度学习开源框架进行训练，分选效率从93%左右提升到97%以上，与人工相差无几

垃圾分类

大恒图像使用基于飞桨开发的语义分割库PaddleSeg，提升30%的缺陷检出率，安装调试周期更是由6到8周缩减到2周。

缺陷检测

连心医疗使用PaddleSeg开发套件研发肺炎筛查系统，在测试数据集上的病灶检测精度和召回率分别达到92%和97%

肺炎筛查



4.其它深度学习框架

Caffe

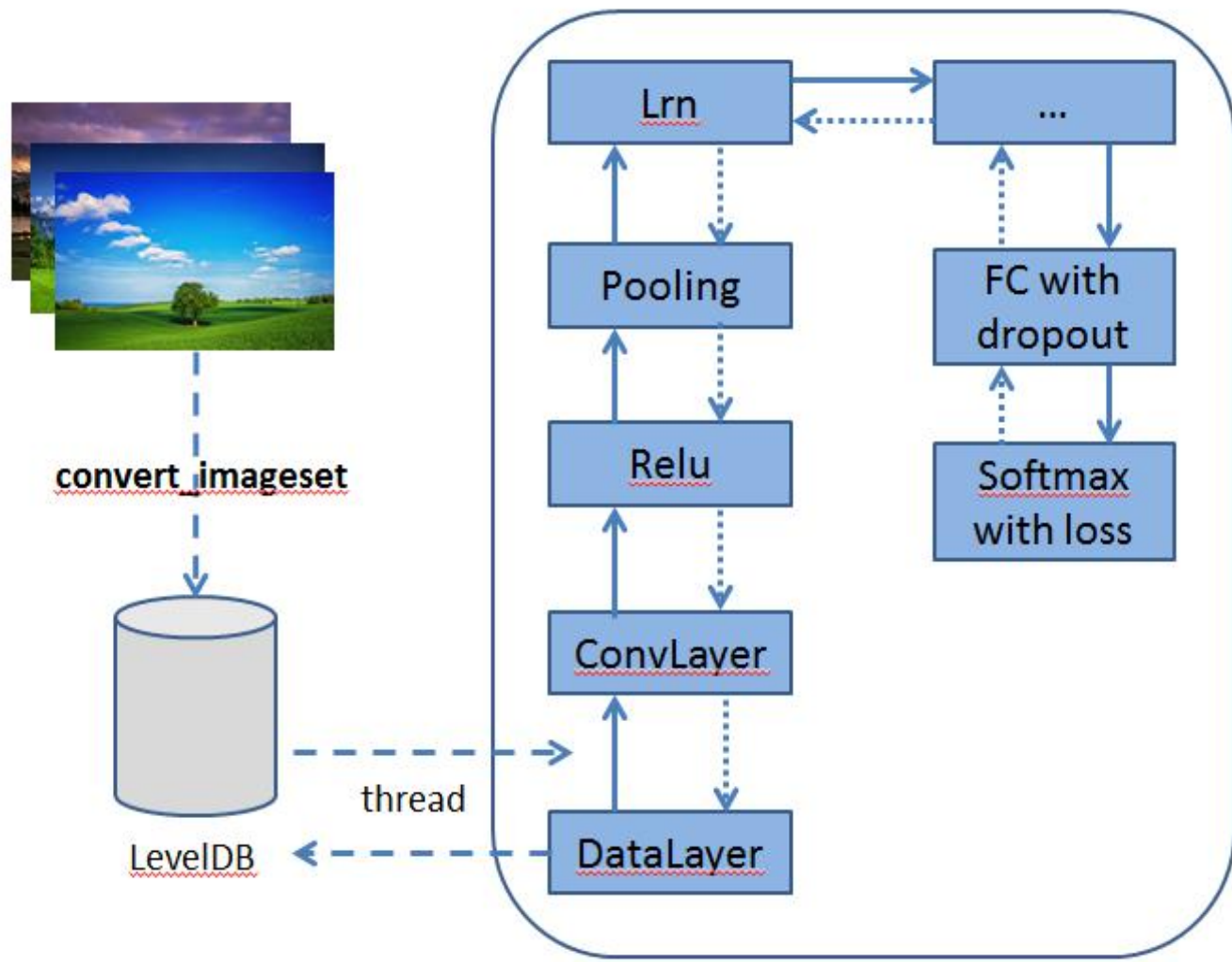
2013年贾扬清在加州大学伯克利分校攻读博士期间创建了Caffe项目。Caffe带有C++、Python和MATLAB接口，在图像处理和视频处理领域有广泛使用。

Caffe

Caffe架构

Caffe的核心模块有三个，分别是Blobs、Layers和Nets。

- Blobs用来进行数据存储、数据交互和处理。
- Layers是神经网络的核心，定义了许多层级结构。
- Nets是一系列Layers的集合，并且这些层结构通过连接形成一个网图。



Caffe的特点

Caffe的优点：

- 速度快，谷歌数据标准提升了效率
- 相对稳定
- 迁移性较好
- CPU与GPU的无缝切换

Caffe的缺点：

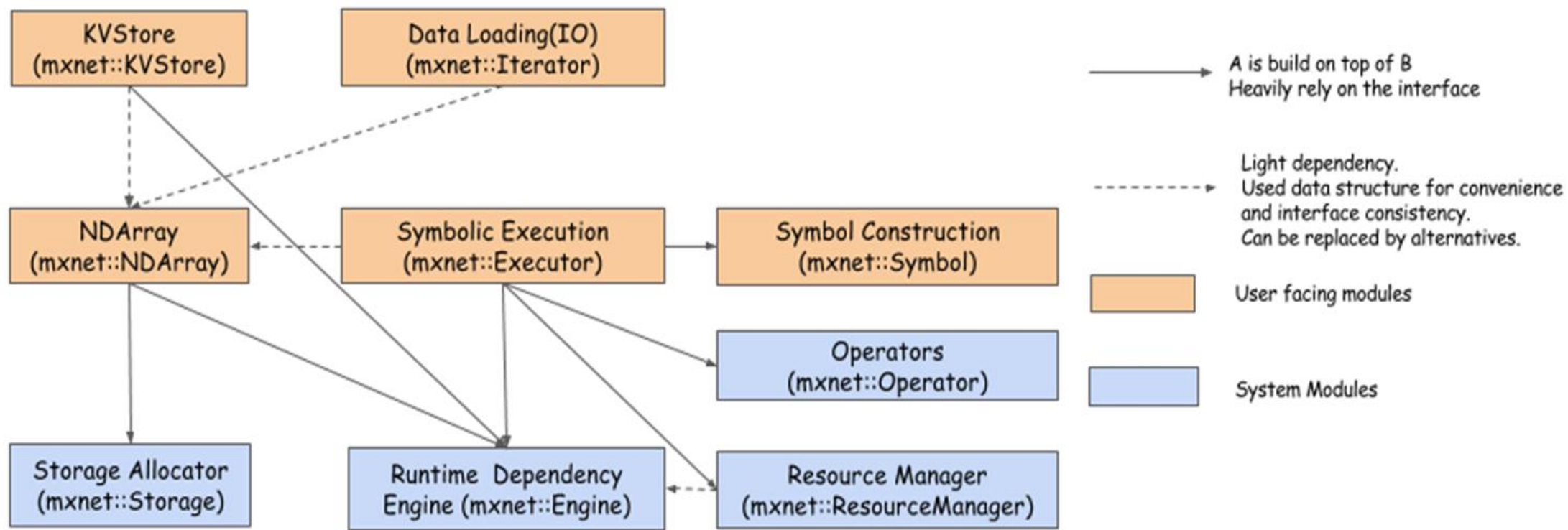
- 各个版本间的兼容性较差
- 不够灵活
- 安装调式比较难

MXNet

MXNet在2015年李沐等人初次发布，如今成了亚马逊的官方框架，有着非常好的分布式支持，占用显存低，同时有Python、C++、R、Matlab和JavaScript等接口。



MXNet架构



MXNet的特点

MXNet的优点：

- 相当快的评测结果
- 非常好的灵活性
- 支持多种设备（包括手机）
- 支持多GPU训练

MXNet的缺点：

- 学习社区小，学习文档较少
- 安装调试较麻烦
- 数据流图比theano差

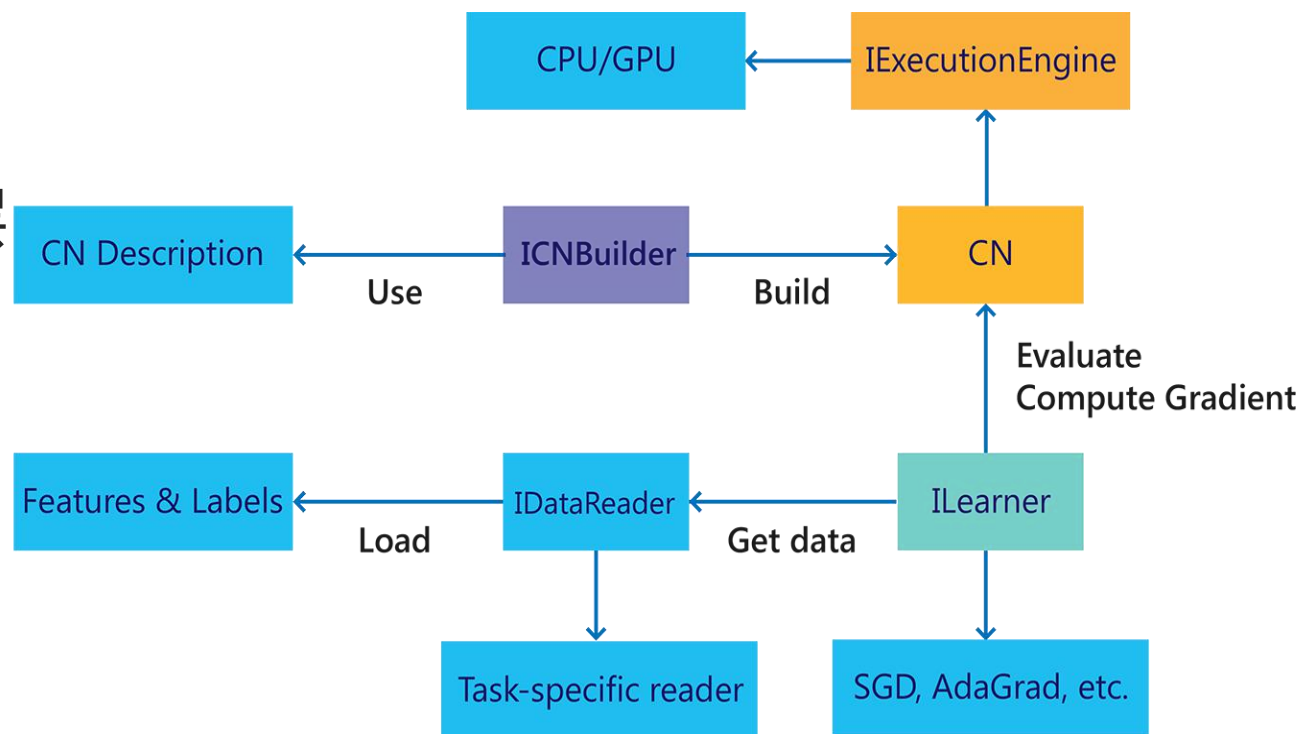
CNTK

CNTK 是一个由微软研究院开发的深度学习算法框架，于2016年在GitHub上开源。它是基于C++语言进行编写，提供Python和C#的接口，相比通用深度学习社区，它在语音识别社区的知名度更高。



CNTK架构

- **IDataReader**: 数据层
- **CN**: 学习层
- **IExecutionEngine**:
预测层





CNTK的特点

CNTK的优点：

- 速度较快
- 很容易上手
- 学习文档较规范
- 支持多GPU训练

CNTK的缺点：

- 学习社区小
- 不支持移动设备

MindSpore

MindSpore是2019年华为推出的新一代全场景AI计算框架，2020年3月开源。该计算框架可满足终端、边缘计算、云全场景需求，能更好保护数据隐私。





MindSpore

全场景统一API
Unified APIs for all scenarios

自动微分
Automatic differentiation

自动并行
Automatic parallelization

自动调优
Automatic tuning

MindSpore IR 计算图表达
MindSpore intermediate representation (IR) for computation graphs

On-Device执行
On-device execution

Pipeline并行
Pipeline parallelism

深度图优化
Deep graph optimization

端-边-云按需协作分布式架构 (部署、调度、通信等)
Device-edge-cloud cooperative distributed architecture (for deployment, scheduling, communications, etc.)

处理器: Ascend、GPU、CPU
Processors: Ascend, GPU, CPU

- 易用性: 统一编译

- 网络和算子统一表达和编译
- 自动并行: 复杂网络自动并行
- 自动微分: E2E的自动微分(网络和算子)
- 精准可视调优

- 性能: 全栈编译加速, 软硬件协同

- MindSporeIR实现图和算子统一融合优化
- 与Ascend软硬件协同, 深度图优化

- 全场景: 端侧推理和云上训练协同

- 轻量化和时延 (IoT设备)
- 根据设备信息自适应模型生成
- 训练时量化, 更好的量化精度, 更小的计算开销

MindSpore的优势



简单的开发体验

只需串行表达就能实现并行训练，降低门槛，简化开发流程。



灵活的调试模式

开发者通过变更一行代码即可切换模式，快速在线定位问题。



充分发挥硬件潜能

最佳匹配昇腾处理器，最大程度地发挥硬件能力。



全场景快速部署

支持云、边和手机的快速部署，让开发者专注于AI应用的创造



深度学习平台实践

1

手写数字识别

2

使用三种框架实现

- PyTorch
- TensorFlow 2.0
- Paddle

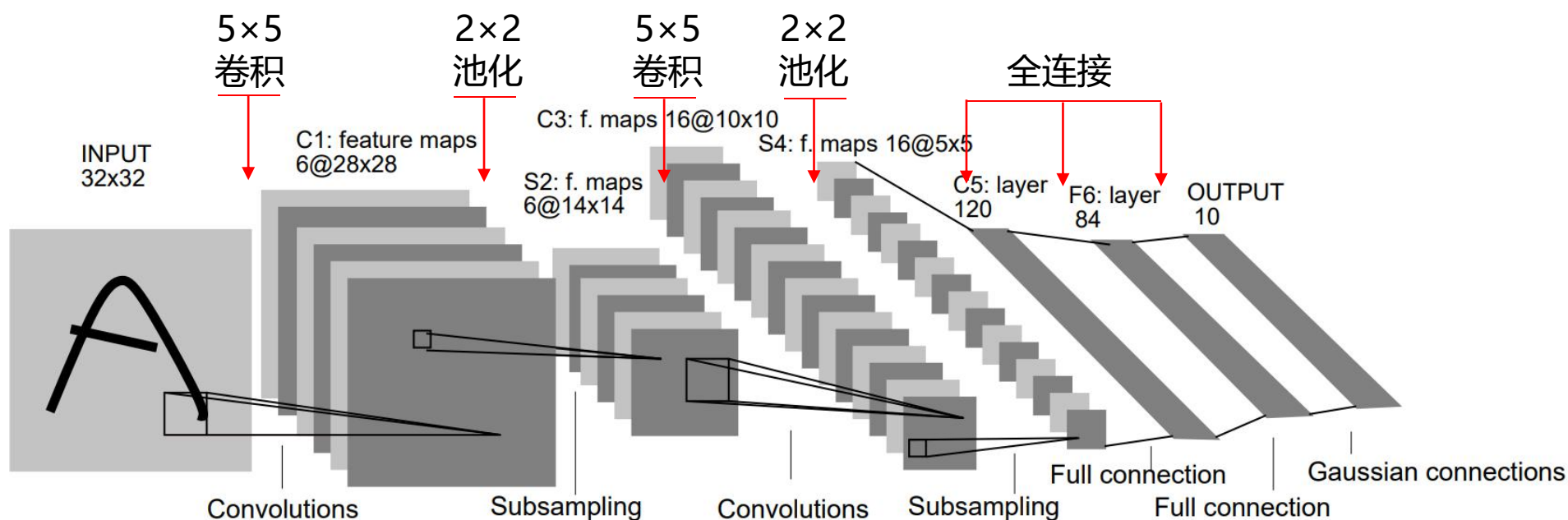
数字识别是典型的图像分类问题，应用于邮政编码识别等领域，缩短了业务处理时间。

使用基于 MNIST 数据集的手写数字识别模型。MNIST是深度学习领域标准、易用的成熟数据集，包含 50000 张训练图像和 10000 张测试图像。



- 任务输入：手写数字图片，其中每张图片为 28×28 的像素矩阵
- 任务输出：对应的 0~9 的数字标签

LeNet-5 由 Yann LeCun 在1998年提出，用于识别手写数字，是经典的卷积神经网络。
LeNet-5 共有7层，包括 2个卷积层、2个池化层、3个全连接层。



LeNet-5 结构

基本步骤

1

数据加载与预处理

2

构建模型

3

训练模型

4

评估模型



- **torch.nn**: 神经网络层和一系列损失函数
- **torch.nn.functional**: 常见的函数
- **torch.optim**: 常见的优化算法
- **torch.utils.data**: 加载和迭代数据
- **torchvision**: 服务于 PyTorch 的图像处理库



```
from torch.util.data import DataLoader
from torchvision import datasets, transforms
```

预处理

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=(0,), std=(1,)),
])
```

训练数据集

```
train_dataset = datasets.MNIST(root='data/',
                                train=True,
                                transform=transform,
                                download=True)
```

测试数据集

```
test_dataset = datasets.MNIST(root='data/',
                                train=False,
                                transform=transform)
```

train Data loader

```
train_loader = DataLoader(dataset=train_dataset,
                           batch_size=batch_size,
                           shuffle=True)
```

test Data loader

```
test_loader = DataLoader(dataset=test_dataset,
                          batch_size=batch_size,
                          shuffle=False)
```



```
class LeNet(nn.Module):
    def __init__(self, num_classes=10):
        """ 在 init 方法指定模型各层, 并初始化 """
        super().__init__()
        # 5*5 卷积
        self.conv1=nn.Conv2d(3,6,kernel_size=5,stride=1)
        # 2*2 池化
        self.pool1=nn.MaxPool2d(kernel_size=2,stride=2)
        # 5*5 卷积
        self.conv2=nn.Conv2d(6,16,kernel_size=5,stride=1)
        # 2*2 池化
        self.pool2=nn.MaxPool2d(kernel_size=2,stride=2)
        # 3个全连接层
        self.linear1=nn.Linear(16*5*5,120)
        self.linear2=nn.Linear(120,84)
        self.linear3=nn.Linear(84, num_classes)
```

```
def forward(self, x):
    """ 在 forward 方法指定前向传播流程 """
    out=self.conv1(x)
    out=F.relu(self.pool1(out))

    out=self.conv2(out)
    out=F.relu(self.pool2(out))

    out=out.reshape(-1, 16*5*5)

    out=self.linear1(out)
    out=self.linear2(out)
    out=self.linear3(out)
    return out
```

```
# 初始化模型, 加载至 GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = LetNet(num_classes).to(device)

# 损失函数, 优化器
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(num_epochs):
    # mini-batch 方式训练模型
    for images, labels in train_loader:
        # 加载数据
        images = images.to(device)
        labels = labels.to(device)

        # 前向传播, 计算损失
        outputs = model(images)
        loss = criterion(outputs, labels)

        # 反向传播, 更新参数
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```



上下文管理器

```
# 评估模型
correct, total = 0, 0
model.eval()
with torch.no_grad():
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Test Accuracy: {100 * correct / total} %')
```



```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets
# 加载 MNIST 数据集
(x, y), (x_test, y_test) = datasets.mnist.load_data() # 返回数组的形状

train_db = tf.data.Dataset.from_tensor_slices((x, y))

train_db = train_db.shuffle(10000) # 数字为缓冲池的大小

# 设置批训练
train_db = train_db.batch(512) # batch size 为 128

# 预处理函数
def preprocess(x, y): # 输入x的shape 为[b, 32, 32], y为[b]
    x = tf.cast(x, dtype=tf.float32) / 255. # 标准化
    x = tf.reshape(x, [-1, 28 * 28])
    y = tf.cast(y, dtype=tf.int32) # 转成整型张量
    y = tf.one_hot(y, depth=10)
    return x, y

# 将数据集传入预处理函数, train_db支持map映射函数
train_db = train_db.map(preprocess)
# 训练20个epoch
train_db = train_db.repeat(20)

# 以同样的方式处理测试集
test_db = tf.data.Dataset.from_tensor_slices((x_test, y_test))
test_db = test_db.shuffle(1000).batch(512).map(preprocess)
```

三种构建模型的方式

- Sequential API: 顺序的线性网络（不支持多输入输出 和 分支结构）
- Functional API: 函数式编程构建模型
- Subclassing API: 面向对象方式构建模型

```
from tensorflow.keras import Sequential
from tensorflow.keras import layers, losses, optimizers

network = Sequential([
    layers.Conv2D(6, kernel_size=3, strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2),
    layers.ReLU(),
    layers.Conv2D(16, kernel_size=3, strides=1),
    layers.MaxPooling2D(pool_size=2, strides=2),
    layers.ReLU(),
    layers.Flatten(),
    layers.Dense(120, activation='relu'),
    layers.Dense(84, activation='relu'),
    layers.Dense(10)
])
```

构建网络模型，给定输入 X 的形状，其中 4 为随意的 *BatchSize*

```
network.build(input_shape=(4, 28, 28, 1))
```



损失函数、优化器

```
criteon = losses.CategoricalCrossentropy(from_logits=True)
optimizer = optimizers.Adam(lr=1e-4)
```

```
for step, (x, y) in enumerate(train_db):
```

```
    x = tf.reshape(x, (-1, 28, 28))
```

上下文管理器

```
    with tf.GradientTape() as tape:
```

```
        x = tf.expand_dims(x, axis=3)    # 插入 1个新维度
```

```
        # 前向传播，计算损失
```

```
        out = network(x)
```

```
        loss = criteon(y, out)
```

```
        # 计算梯度
```

```
        grads = tape.gradient(loss, network.trainable_variables)
```

```
        # 更新参数
```

```
        optimizer.apply_gradients(zip(grads,
network.trainable_variables))
```



```
total, total_correct = 0., 0
correct, total = 0, 0
for x, y in test_db:
    x = tf.reshape(x, (-1, 28, 28))
    x = tf.expand_dims(x, axis=3)
    # 前向传播, 预测
    out = network(x)
    pred = tf.argmax(out, axis=-1)
    y = tf.cast(y, tf.int64)
    y = tf.argmax(y, axis=-1)
    # 统计预测正确数量
    correct += float(tf.reduce_sum(tf.cast(tf.equal(pred, y), tf.float32)))
    total += x.shape[0]

print(f'test acc: {correct / total}')
```

```
import paddle
import paddle.vision.transforms as T

# 数据加载和预处理
# [0-255] -> [0-1]
transform = T.Normalize(mean=[127.5], std=[127.5])

# 训练数据集
train_dataset = paddle.vision.datasets.MNIST(mode='train',
transform=transform)
# 测试数据集
eval_dataset = paddle.vision.datasets.MNIST(mode='test', transform=transform)
```

两种构建模型的方式：

- Sequential API: 顺序的线性网络（不支持多输入输出 和 分支结构）
- Subclassing API: 面向对象方式构建模型

```
net = paddle.nn.Sequential(  
    # 卷积层  
    ('C1', paddle.nn.Conv2D(in_channels=1, out_channels=6, kernel_size=3, padding=1, stride=1)),  
    # 池化层 6x28x28  
    ('ReLU1', paddle.nn.ReLU()),  
    ('S2', paddle.nn.MaxPool2D(kernel_size=2, stride=2, ceil_mode=True)),  
    # 卷积层 6x14x14  
    ('C3', paddle.nn.Conv2D(6, 16, kernel_size=5, stride=1, padding=0)),  
    # 池化层 16x10x10  
    ('ReLU2', paddle.nn.ReLU()),  
    ('S4', paddle.nn.MaxPool2D(kernel_size=2, stride=2, ceil_mode=True)),  
    # 全连接层 16x5x5  
    ('C5', paddle.nn.Conv2D(16, 120, kernel_size=5, stride=1, padding=0)),  
    # 120x1x1  
    ('ReLU3', paddle.nn.ReLU()),  
    ('ReLU4', paddle.nn.Flatten()),  
    # 全连接层 120  
    ('F6', paddle.nn.Linear(120, 84)),  
    # 全连接层 84  
    ('ReLU5', paddle.nn.ReLU()),  
    ('OUTPUT', paddle.nn.Linear(84, 10))  
)
```

```
# 配置优化器, 损失函数, 评估指标
loss = paddle.nn.CrossEntropyLoss()
optimizer = paddle.optimizer.Adam(learning_rate=0.001,
parameters=net.parameters())
model.prepare(optimizer=optimizer,
               loss=loss,
               metrics=paddle.metric.Accuracy())

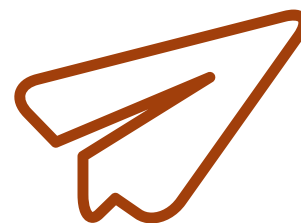
# 启动模型全流程训练
model.fit(train_data=train_dataset,
          eval_data=eval_dataset,
          batch_size=64,
          epochs=5,
          verbose=1,
          shuffle=True)
```

```
result = model.evaluate(eval_dataset, verbose=1)

print(result)
```

输出:

```
Eval begin...
The loss value printed in the log is the current batch, and the metric is
the average value of previous step.
step 10000/10000 [=====] - loss: 2.0623e-05 -
acc: 0.9884 - 4ms/step
Eval samples: 10000
{'loss': [2.0622994e-05], 'acc': 0.9884}
```



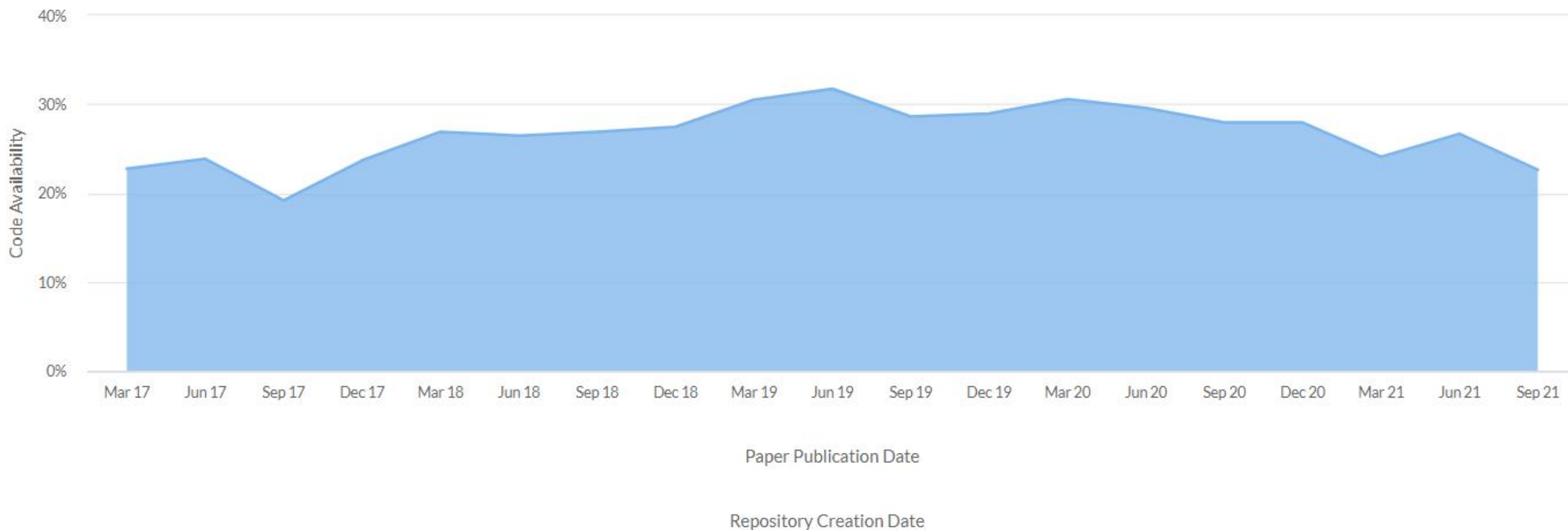
总结

模型对比总结

	Tensorflow2.0	PyTorch	PaddlePaddle	MindSpore
支持语言	Python、Java、C++	Python、C++、CUDA	Python	Python
硬件支持	CPU、GPU	CPU、GPU	CPU、GPU、海光DCU、昆仑XPU、昇腾NPU	CPU、GPU、Ascend、NPU
优势	可视化训练过程、高度灵活性、 运行性能优越	简洁优雅、高效快速、易用性好、 社区活跃	可视化、 高度封装 、官方模型库、 推理引擎一体化	支持 昇腾芯片 且性能好、易用性好、 全场景快速部署

模型生态情况

PvTorch (Solid) vs TensorFlow (Dotted) Raw Counts



使用推荐

- **PyTorch**: 适用于研究需求, 容易快速出算法原型, 便于调试
- **TensorFlow**: 适用于大规模场景, 工业级应用
- **PaddlePaddle**: 适用于使用国产芯片等场景(注: 飞浆论文复现挑战赛
总奖金 ¥ 1500000)
- **MindSpore**: 适用于边缘计算等场景, 与华为昇腾芯片完美结合, 满足云、边、终端全场景需求

TensorFlow相关资料： [TensorFlow官网](#)、[TensorFlow教程](#)

PyTorch相关资料： [PyTorch官网](#)、[ApacheCN深度学习译文集](#)

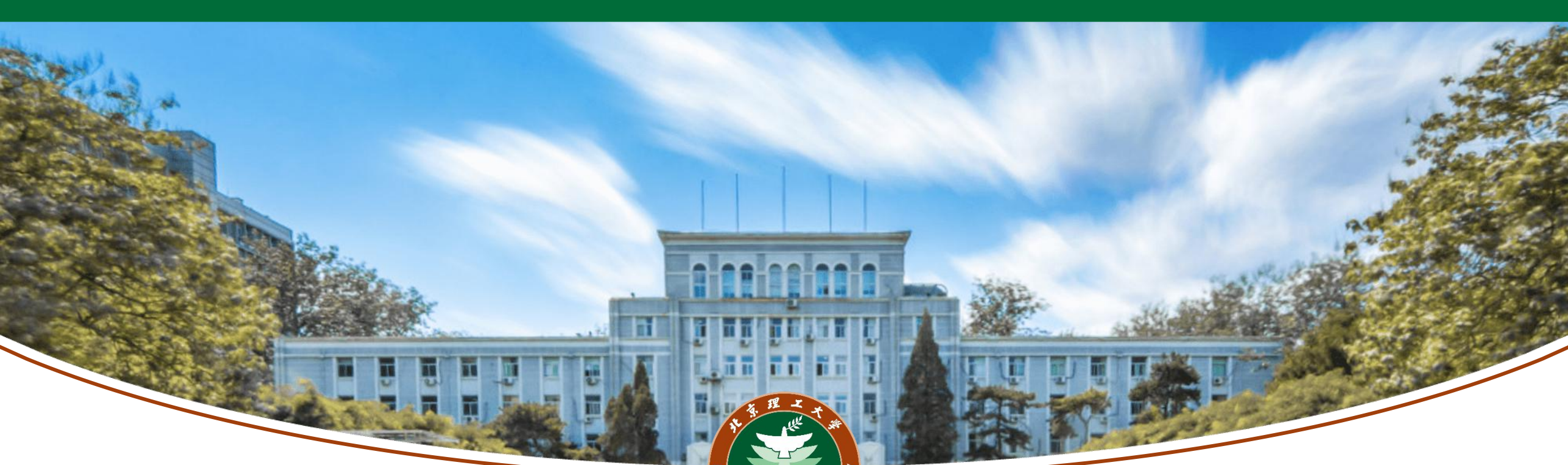
PaddlePaddle相关资料： [Paddle文档](#)、[Paddle教程](#)

Caffe相关资料： [Caffe官网](#)、[Caffe GitHub地址](#)、[Caffe教程](#)

MXNet相关资料： [MXNet官网](#)、[MXNet GitHub地址](#)、[MXNet教程](#)

CNTK相关资料： [CNTK官网](#)、[CNTK GitHub地址](#)、[CNTK教程](#)

MindSpore相关资料： [MindSpore官网](#)、[MindSpore Gitee](#)、[MindSpore 教程](#)



感谢老师的悉心指导

Thanks for Your Attention