深度学习平台

大数据处理技术

第一组：牛紫旭 张鸿坤 王紫玉 魏耀涛 王思奇

导　师：张华平

时间：2022-3-1

目录 | CONTENTS

1 深度学习平台概述

我们一般通过开始从零学习了使用 Python 和 NumPy 实现深度学习算法来入门深度学习，这样可以很好的理解这些深度学习算法实际上到底是在做什么。但随着入门之后会发现，除非应用更复杂的模型，例如卷积神经网络，或者循环神经网络，或者开始应用很大的模型，否则它就越来越不实用了，至少对大多数人而言，从零开始全部靠自己实现并不现实，尤其是做项目或者做课题等等。

深度学习软件框架应运而生，可以帮助实现这些复杂的模型。它可以做矩阵乘法，并且在建很大的应用时，通过一个数值线性代数库，它会更高效地实现矩阵乘法，利用一些深度学习框架会更加实用，会使工作更加有效。

常见深度学习框架

现在有许多深度学习框架，能让实现神经网络变得更简单。每个框架都针对某一用户或开发群体。

如何选择/衡量框架

- 首先，一个重要的标准就是便于编程，这既包括神经网络的开发和迭代，还包括为产品进行配置，为了成千上百万，甚至上亿用户的实际使用，这取决于你想要做什么。
- 第二个重要的标准是运行速度，特别是训练大数据集时，一些框架能让你更高效地运行和训练神经网络，这取决于你的数据集详细情况。
- 第三个标准是这个框架是否真的开放，要是一个框架真的开放，它不仅需要开源，而且需要良好的管理，比如两大框架——TensorFlow 和 Pytorch。

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

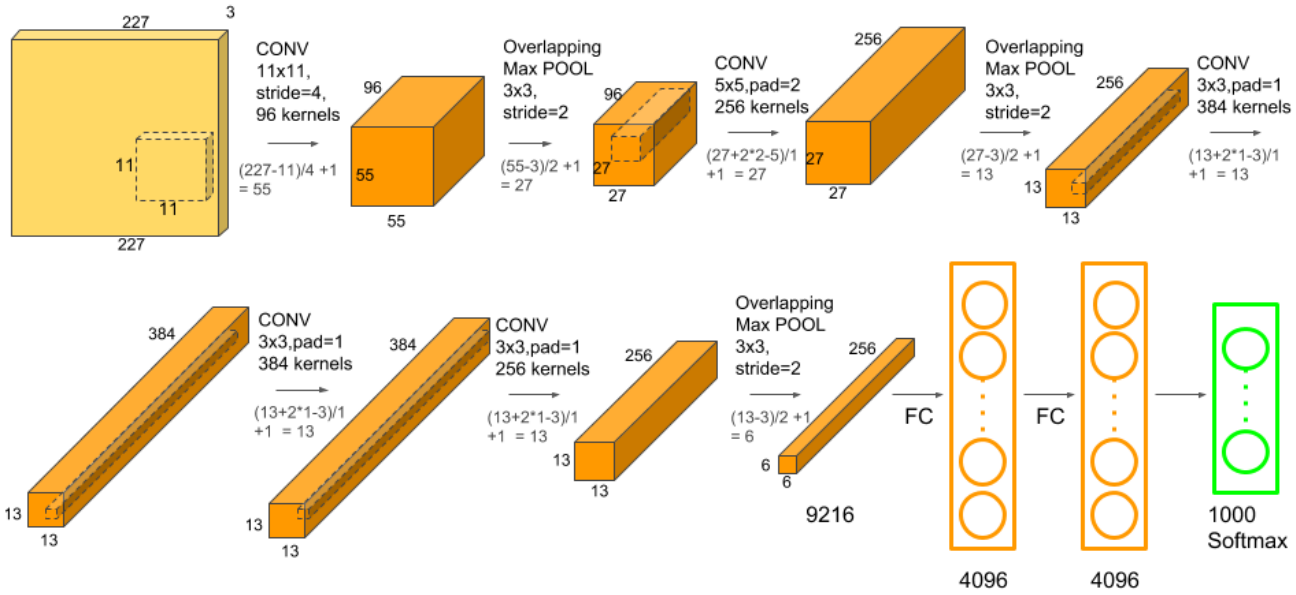
目录

CONTENTS

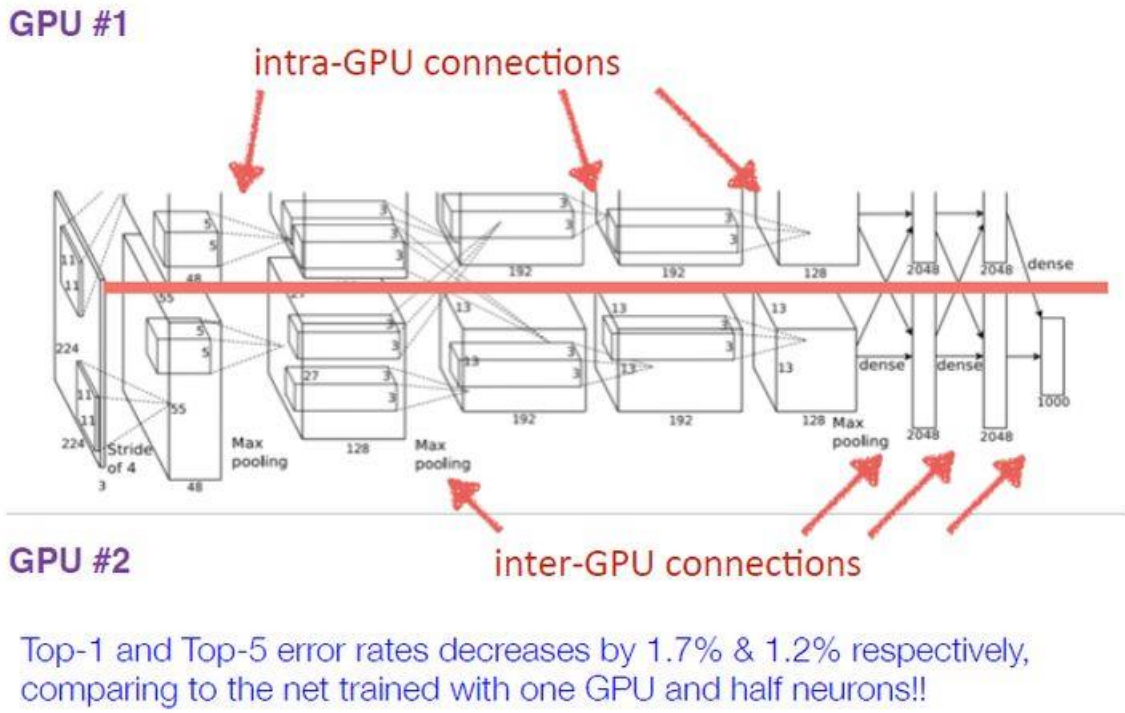
石器时代（21世纪初）

神经网络的概念已经出现一段时间了。在21 世纪初之前，有一些工具可以用来描述和开发神经网络。这些工具包括 MATLAB、OpenNN、Torch 等，它们要么不是专门为神经网络模型开发定制的，要么拥有复杂的用户 api，缺乏 GPU 支持。在此期间，Machine Learning 实践者在使用这些原始的深度学习框架时不得不做很多繁重的工作。

AlexNet



细化的结构图



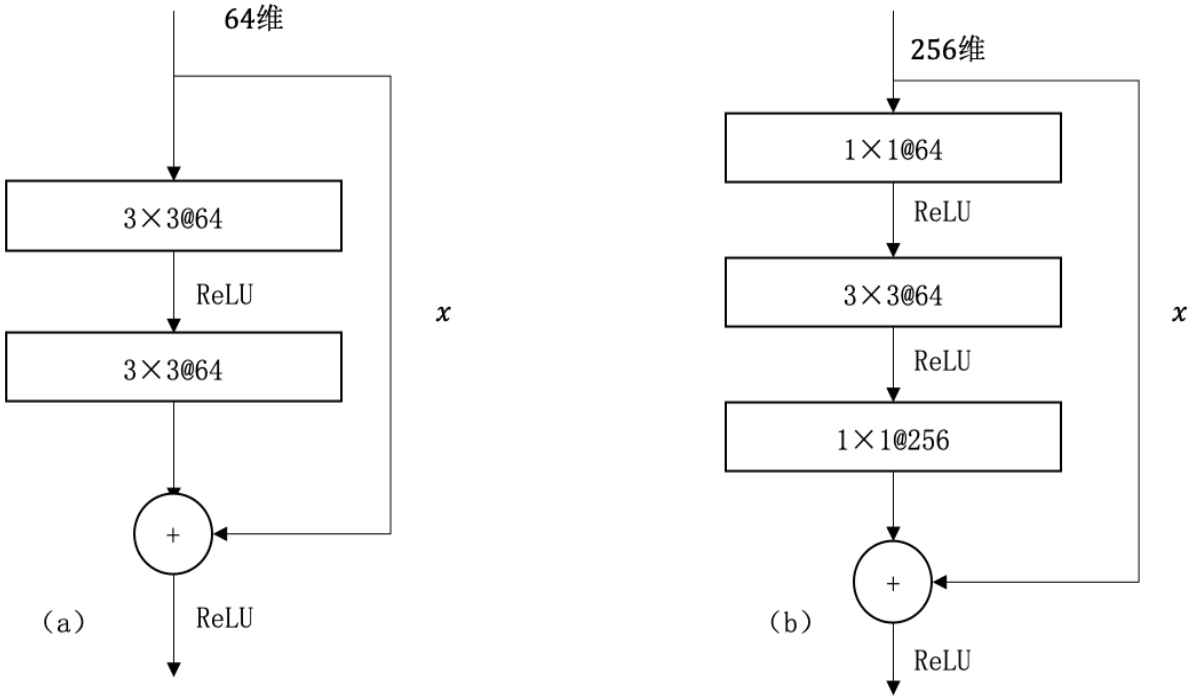
多GPU训练

青铜时代（~2012年）

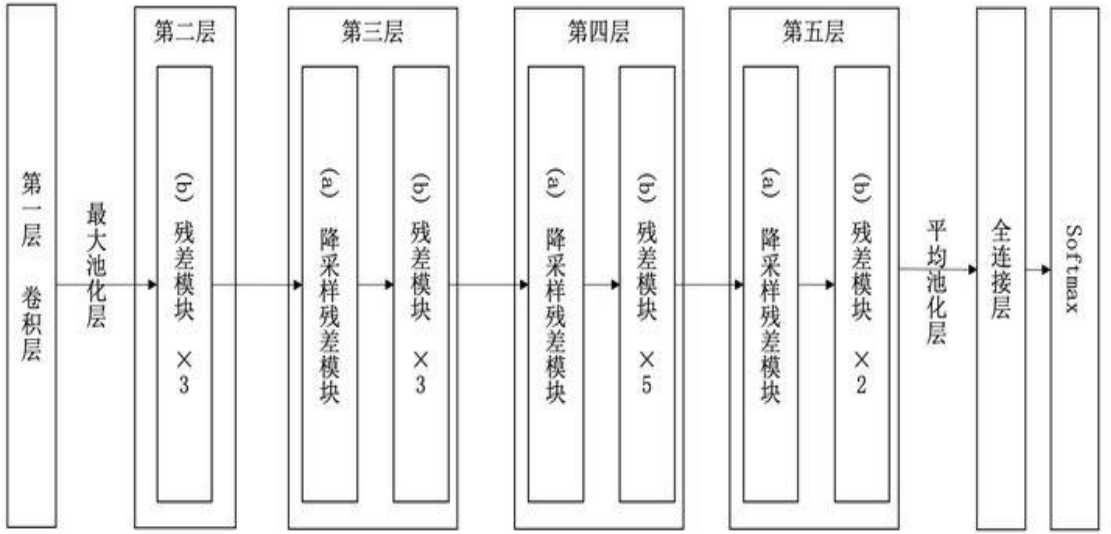
大约在这个时候，一些早期的深度学习框架，如 Caffe、Chainer 和Theano 应运而生。使用这些框架，用户可以方便地建立复杂的深度神经网络模型，如 CNN、RNN、LSTM 等。此外，这些框架还支持多 GPU 训练，这大大减少了对这些模型的训练时间，并且能够对以前无法装入单一 GPU 内存的大型模型进行训练。

在这些框架中，Caffe 和 Theano 使用声明式编程风格，而 Chainer 采用命令式编程风格。这两种不同的编程风格也为即将到来的深度学习框架设定了两条不同的开发路径。

ResNet



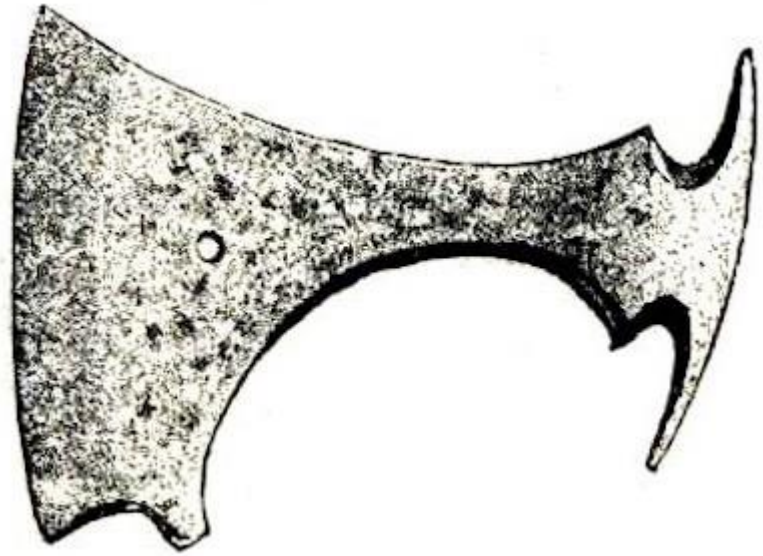
ResNet构建块示意图



34层ResNet模型架构图

铁器时代（2015~2016）

谷歌开源了著名的 TensorFlow 框架，它至今仍是 Machine Learning 领域最流行的深度学习框架。Caffe 的发明者加入了 Facebook 并发布了 Caffe2；与此同时，Facebook AI 研究（FAIR）团队也发布了另一个流行的框架 PyTorch，它基于 Torch 框架，但使用了更流行的 Python api。微软研究院开发了 CNTK 框架。亚马逊采用了 MXNet。



声明式编程

Theano

CNTK

Tensorflow

命令式编程

Torch

Pytorch

杂交

MXNet

罗马时代（2019~2020）

正如人类历史的发展一样，深度学习框架经过一轮激烈的竞争，最终形成了两大「帝国」：TensorFlow 和 PyTorch 的双头垄断，这两大「帝国」代表了深度学习框架研发和生产中95%以上的用例。

2019 年，Chainer 团队_将他们的开发工作转移到 PyTorch；类似地，微软_停止了 CNTK 框架的积极开发，部分团队成员转而支持 Windows 和 ONNX 运行时上的PyTorch。Keras 被 TensorFlow 收编，并在 TensorFlow 2.0 版本中成为其高级 api 之一。在深度学习框架领域，MXNet 仍然位居第三。

该阶段下深度学习框架的发展的两种趋势

大模型训练

分布式技术

复杂模型的诞生

可用性

命令式编程

便捷api

方便用户使用

工业时代（2021+）

深度学习在自动驾驶、个性化推荐、自然语言理解到医疗保健等广泛领域取得了巨大成功，带来了前所未有的用户、开发者和投资者浪潮。这也是未来十年开发深度学习工具和框架的黄金时期。尽管深度学习框架从一开始就有了长足的发展，但它们之于深度学习的地位还远远不如编程语言 JAVA/ c++ 之于互联网应用那样的成熟。还有很多令人兴奋的机会和工作有待探索和完成。

2 Tensorflow

TensorFlow简介

简介

TensorFlow 是谷歌基于DistBelief进行研发的第二代人工智能学习系统。TensorFlow是将复杂的数据结构传输至人工智能神经网络中进行分析和处理，里面有完整的数据流向和处理机制，同时还封装了大量高效可用的算法及神经网络搭建方面的函数，可以在此基础上进行深度学习的开发与研究。

- 高度的灵活性
- 多语言支持
- 可移植性
- 丰富的算法库

TensorFlow历史版本

张量

张量可以看作是一个多维的数组或者列表，它是对矢量和矩阵的更高维度的泛化，由“tf.Tensor”类定义。

在一个张量中主要保存了三个属性，分别为：

- 名字
- 维度
- 数据类型

```
import tensorflow as tf
a=tf.constant([1.0,2.0],name=“a”)
b=tf.constant([3.0,4.0],name=“b”)
res=tf.add(a,b,name=“add”)
print(res)
```

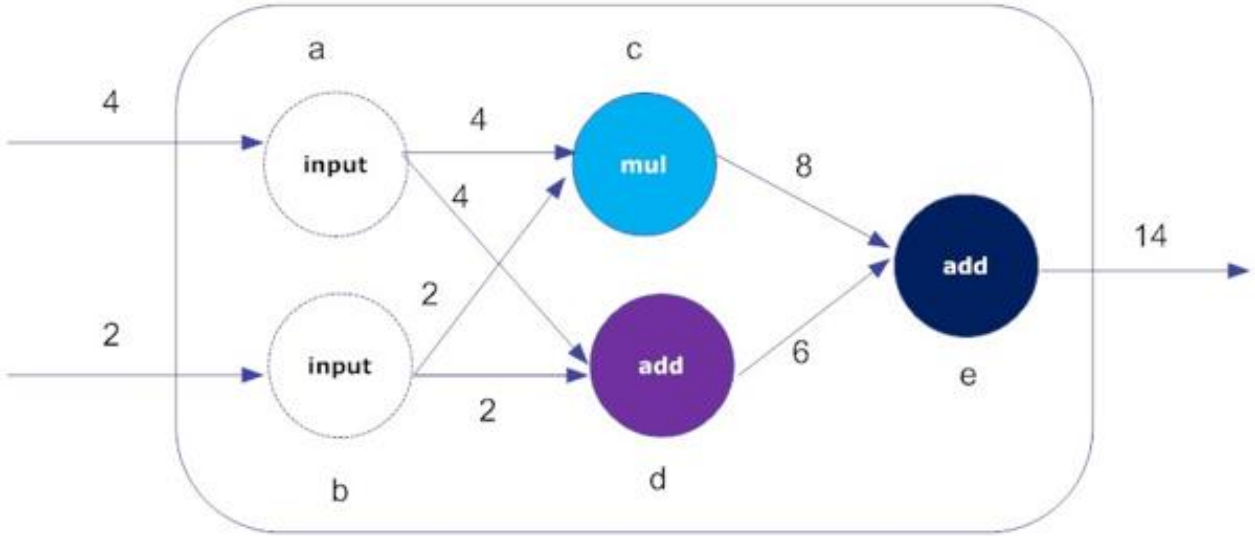
运行结果

```
Tensor(“add:0”,shape=(2,),dtype=float32)
```

数据流图

数据流图是一个有向图，是对TensorFlow中计算任务的抽象描述。

TensorFlow使用数据流图将计算表示成独立指令之间的依赖关系。



TensorFlow在GitHub上的主项目下还有类似models这样的项目，里面包含了许多应用领域的最新研究算法的代码实现，如图象识别领域效果最好的 Inception网络和残差网络，能够让机器自动用文字描述一张图片的应用im2txt项目；自然语言某些处理领域达到人类专家水平的syntaxnet项目等。对于TensorFlow的使用者，可以很方便地借鉴这些已经实现的高质量的项目，快速构建自己的深度学习应用。

TensorFlow Lite

- 有了TensorFlow Lite，应用开发者可以在移动设备上部署人工智能。

- TensorFlow Lite模型具有占用空间小，低延迟的特点。

TensorFlow

TensorFlow更适合大规模项目的部署和嵌入式部署，且拥有良好的跨平台性。

TensorFlow自带的可视化工具TensorBoard可随时查看机器学习训练过程中数据的变化和模型的训练曲线与验证结果。

TensorFlow在图像识别、语音识别、NLP中都取得了较好的成果。

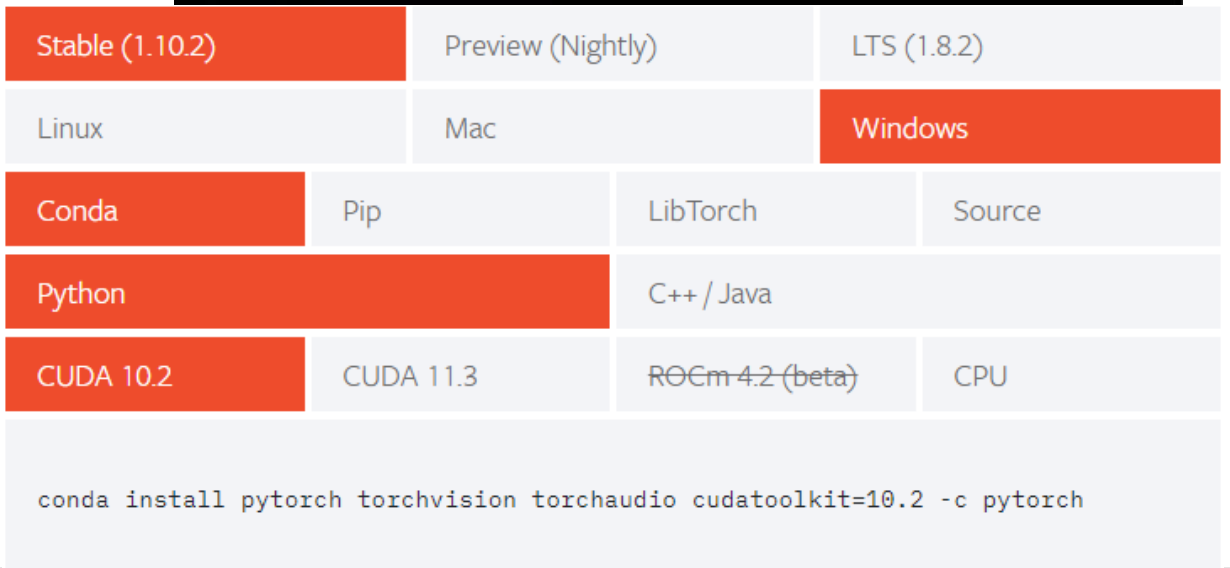
Pytorch

Pytorch更适合轻量级、小规模的开发。

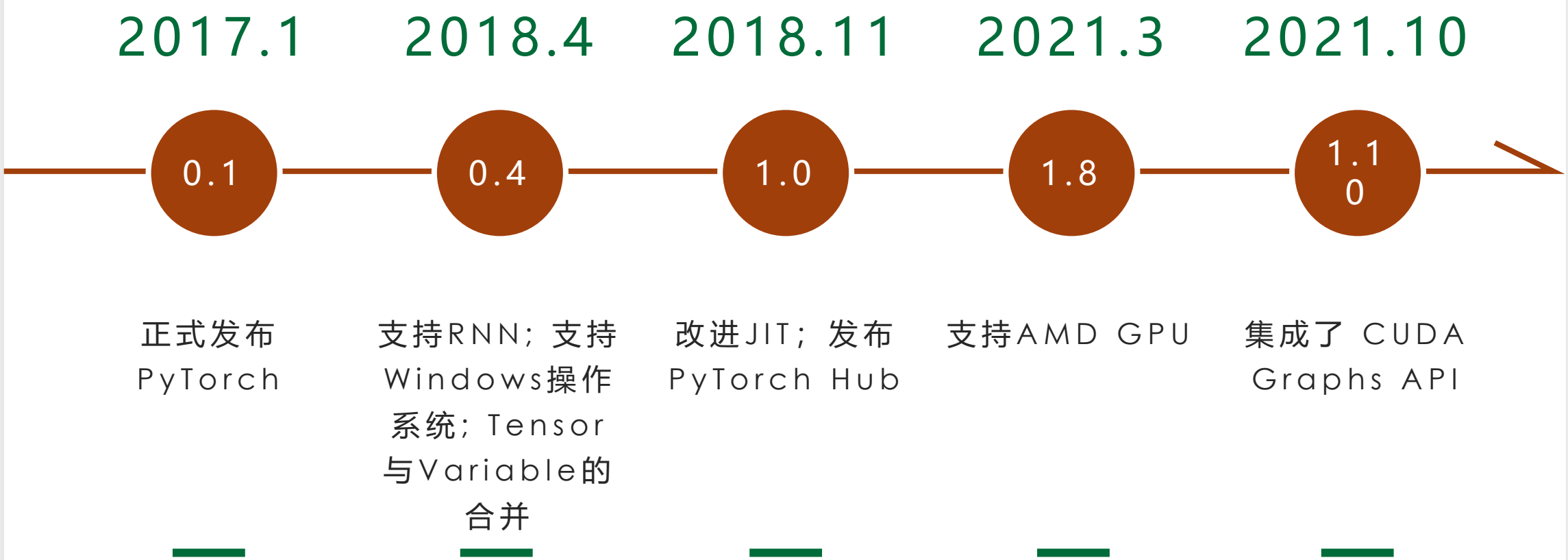
更多……

3 Pytorch

今天是PyTorch公开发布5周年！我们没想到会走这么远，但我们现在达成了这些成就——**2000**贡献者，**9万**项目，GitHub上**390万行**“import torch”



版本更迭


2017.1
2018.4
2018.11
2021.3
2021.10
0.1
0.4
1.0
1.8
1.10
正式发布PyTorch
支持RNN；支持Windows操作系统；Tensor与Variable的合并
改进JIT；发布PyTorch Hub
支持AMD GPU
集成了 CUDA Graphs API

上手快速

深入集成到Python中。

简洁灵活

模型定义十分直观易懂。易于Debug。

优秀的 API

API规范，易于理解查找。

社区支持

Facebook的FAIR强力支持.GitHub上广泛讨论。

Pytorch：Tensors

大家可以把Pytorch中的Tensor理解为可以在GPU上计算的Numpy array。相比array，tensor提供了其他的一些与深度学习息息相关的操作。

Pytorch自动微分器：Autograd

使用autograd时，神经网络的前向传播会定义一个计算图，该图中，node是tensor，edge是函数；在这个graph中进行反向传播可以很容易计算梯度。

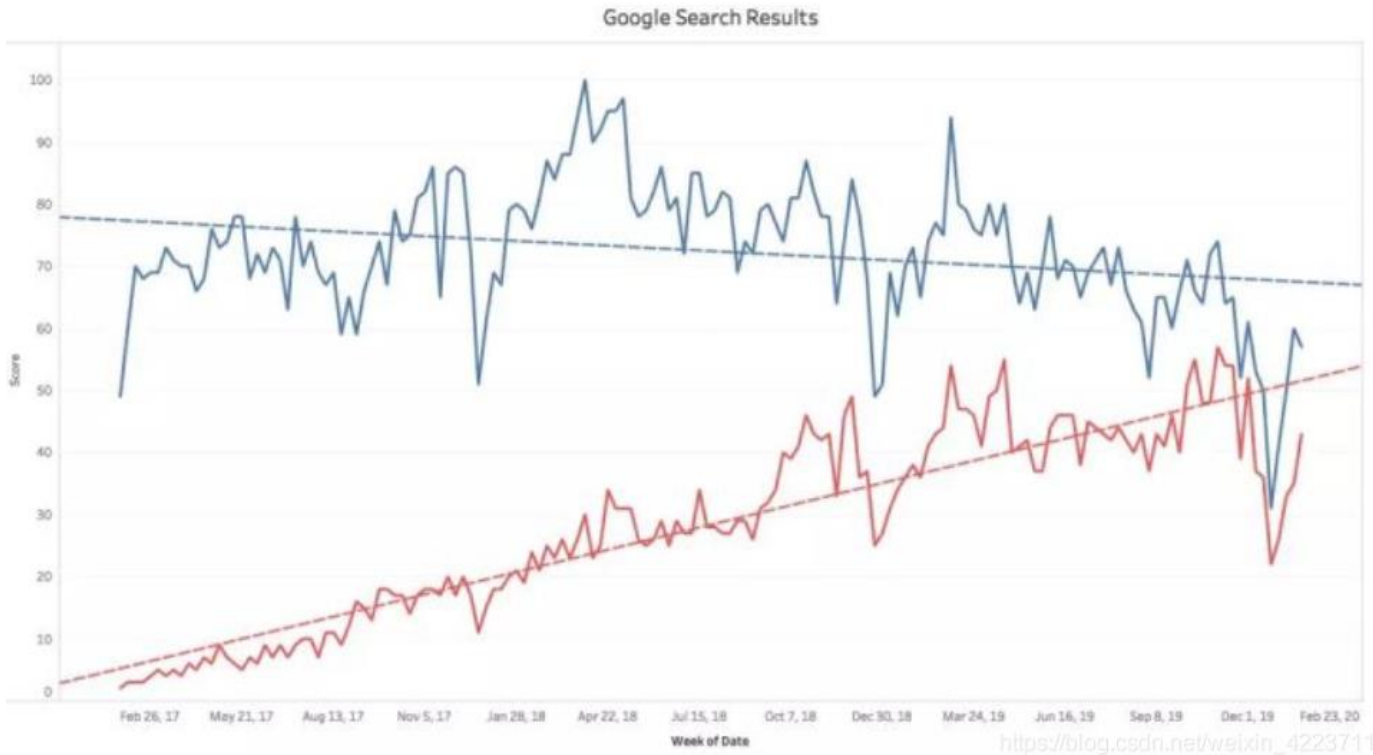
Pytorch：nn module

把计算图打包成layer，layer里有一些待学习的参数，这些参数将会在后面基于梯度信息更新。

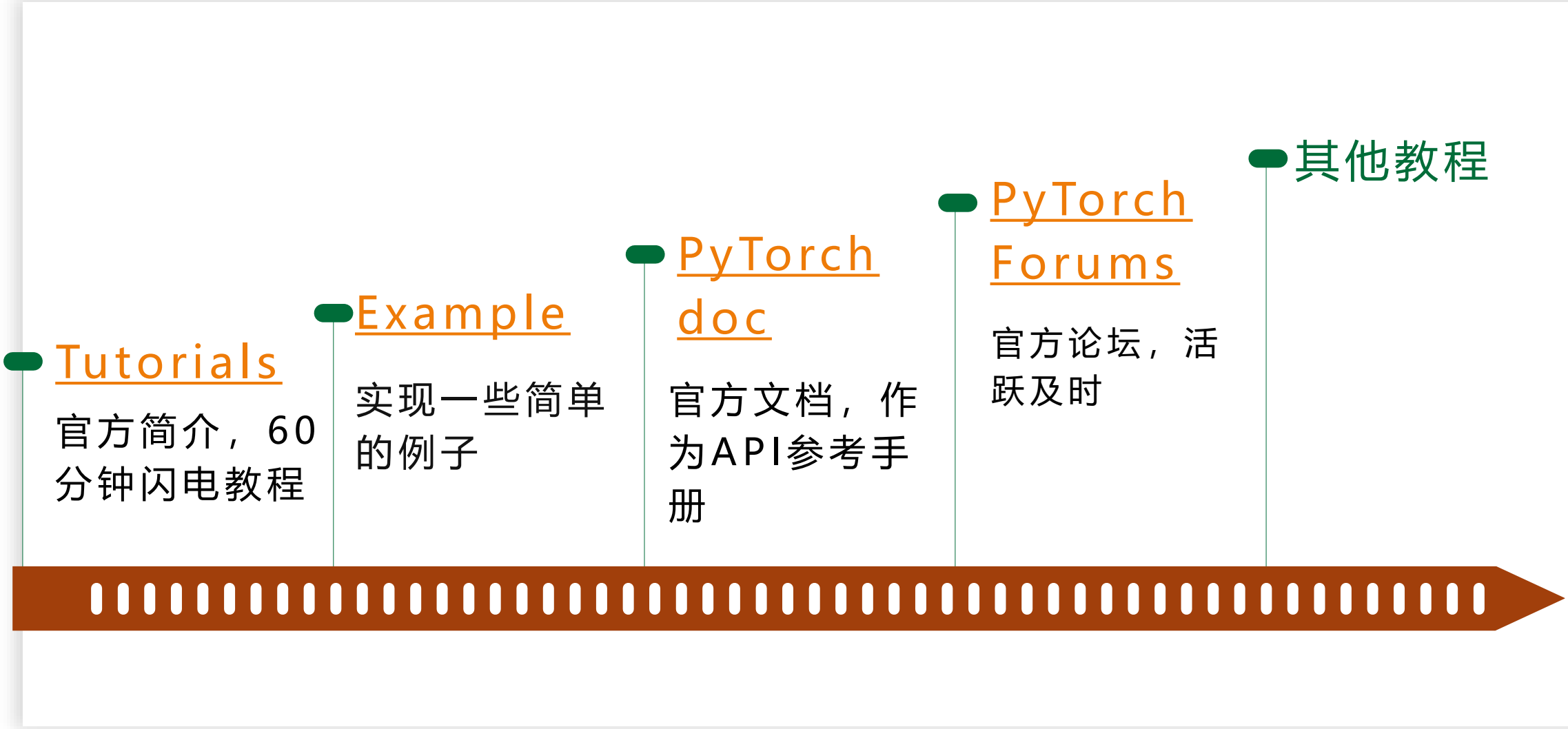
Pytorch：Custom nn Module

pytorch允许通过继承nn module来重新定制自己的模型。

- 称霸顶会。
- 广为讨论。
- 职场优势。



如何入门


Tutorials
官方简介，60分钟闪电教程
Example
实现一些简单的例子
PyTorch doc
官方文档，作为API参考手册
PyTorch Forums
官方论坛，活跃及时
其他教程

4 PaddlePaddle

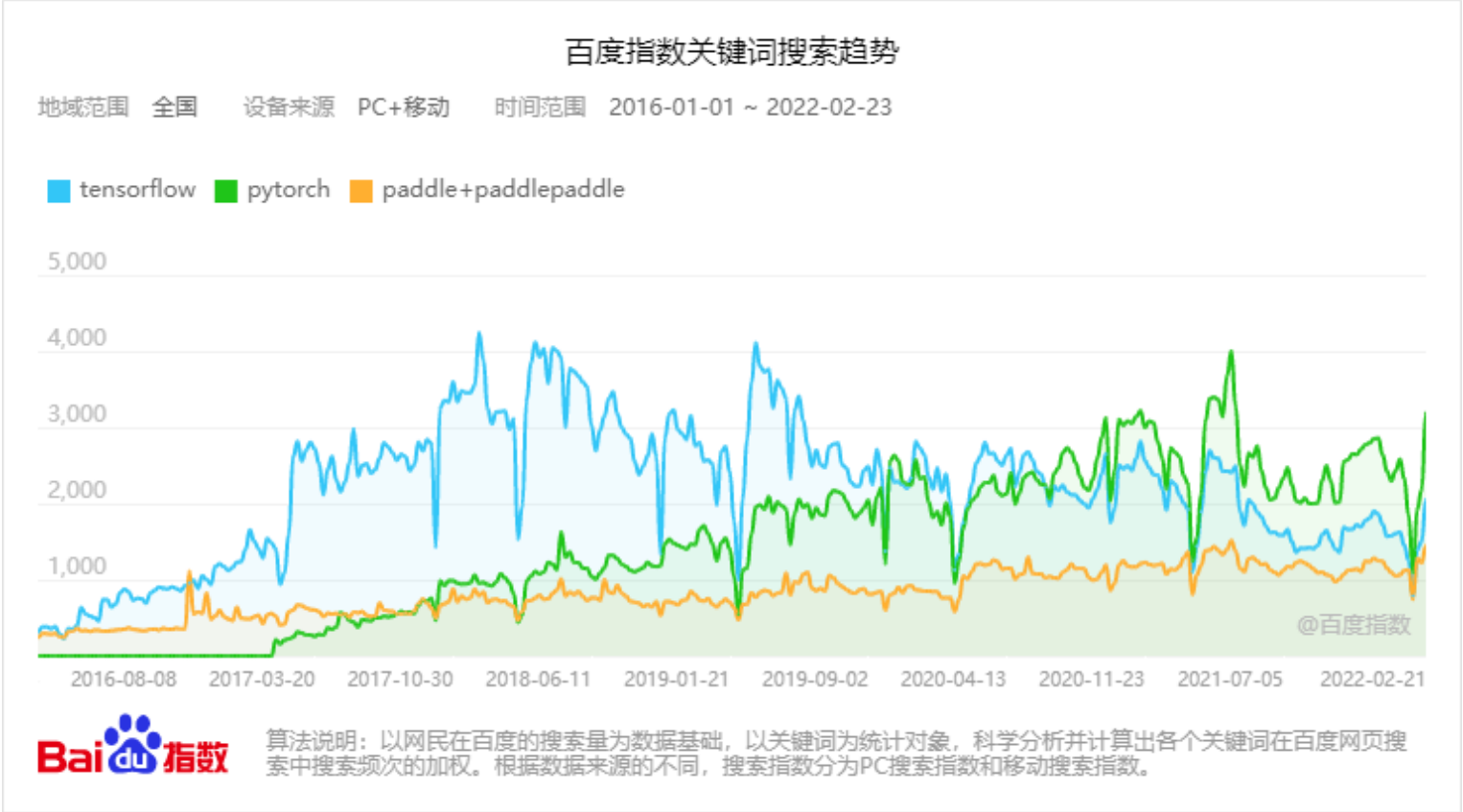
PaddlePaddle（飞桨）

PaddlePaddle（中文名飞桨）是百度出品的国内首个全面开源开放、技术领先、功能完备的产业级深度学习平台

现况

据国际数据公司（IDC）发布的2021年上半年深度学习框架平台市场份额报告显示，百度跃居中国深度学习平台市场综合份额第一。

在2021世界互联网大会乌镇峰会上，百度公布了飞桨深度学习开源开放平台的最新进展：汇聚开发者数量达370万，服务 14 万企事业单位，产生了42.5 万个模型。



发展历程

2018年7月，开源框架 v0.14发布
2018年10月，开源框架 v1.0稳定版本发布
2018年11月，开源框架 v1.1发布
2018年12月，开源框架 v1.2发布
2019年3月，开源框架 v1.3发布
2019年4月，开源框架 v1.4发布
2019年7月，开源框架 v1.5发布
2019年11月，开源框架 v1.6发布
2020年2月，开源框架 v1.7发布
2020年5月，开源框架 v1.8发布
2021年3月，开源框架v2.0发布
2021年5月，开源框架v2.1发布

工具齐全

集深度学习核心框架、基础模型库、端到端开发套件、工具组件和服务平台于一体

动静兼容

模型开发时采用动态图编程；训练或者推理部署时，添加一行装饰器 @to_static，即可化动为静

环境友好

详细的中文文档和良好的中文社区环境，百度打造的人工智能学习与实训社区AI Studio包含了丰富资源

Paddle Inference

Paddle Inference 是飞桨的原生推理库，作用于服务器端和云端，提供高性能的推理能力，支持服务器端 X86 CPU 、 NVIDIA GPU 芯片，兼容Linux/Mac/Windows系统。支持所有飞桨训练产出的模型，完全做到即训即用。同时支持 C++，Python，C，Golang，接口简单灵活。

Paddle Lite

Paddle Lite 是一组帮助开发者在移动设备、嵌入式设备和IoT设备上运行模型的工具，其针对移动端设备的机器学习进行优化，压缩模型和二进制文件体积，推理高效，降低内存消耗。支持的平台涵盖 Android、iOS、嵌入式 Linux 设备、Windows、macOS 和 Linux 主机，支持的语言包括Java、Python、C++。

两段文字


农业
智能桃子分拣机
人力成本节约90%
零售
生鲜进货量预测
单店生鲜报损降低30%
工业
精密零件智能分拣
单人工作量从1天缩短为1小时
各领域应用
APPLICATION IN
VARIOUS FIELDS
人力
AI简历匹配系统
面试成功率提升5倍
林业
林区病虫害周期性监测
每月人工采集变为每半小时自动采集
地产
智能楼宇管理
制冷系统节电20%

5 技术实现

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

深度学习平台

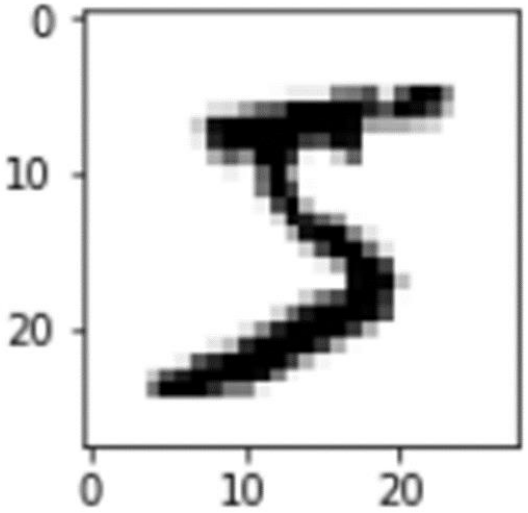
CONTENTS

1 Pytorch

2 Tensorflow

3 PaddlePaddle

手写数字识别

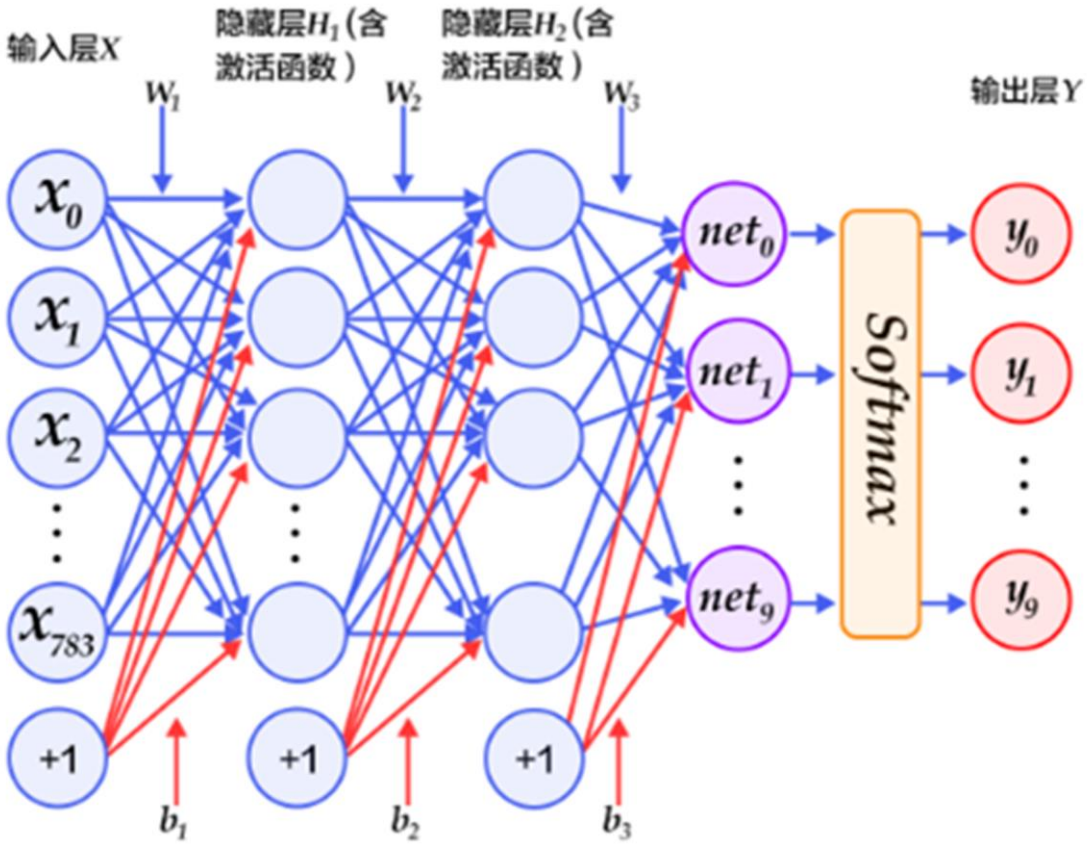


Mnist数据集

MNIST数据集包含60000个训练集和10000测试数据集。分为图片和标签，图片是28*28的像素矩阵，标签为0~9共10个数字。

MLP网络结构

简单的多层感知机，一共有三层，两个大小为100的隐层和一个大小为10的输出层，因为MNIST数据集是手写0到9的灰度图像，类别有10个，所以最后的输出大小是10。最后输出层的激活函数是Softmax，所以最后的输出层相当于一个分类器。加上一个输入层的话，多层感知器的结构是：输入层-->>隐层-->>隐层-->>输出层。



参数设置为：
epochs = 10
Batch_size = 64
Learning_rate = 0.001
Optimizer为adam
损失为交叉熵
运行环境为RAM8G,Disk100G

Tensorflow

北京理工大學 BEIJING INSTITUTE OF TECHNOLOGY

```python
#!/usr/bin/env  python
#  coding:  utf-8
import  tensorflow  as  tf
import  time
mnist=tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test)=mnist.load_data()     #加载数据


x_train=tf.keras.utils.normalize(x_train,axis=1)
x_test=tf.keras.utils.normalize(x_test,axis=1)
start  =  time.time()

model  =  tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(128,activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(128,activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(10,activation=tf.nn.softmax))
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])


model.fit(x_train,y_train,epochs=10)
print("运行时间：",time.time()-start)
```

```
Epoch 1/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2683 - accuracy: 0.9204
Epoch 2/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1115 - accuracy: 0.9655
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0754 - accuracy: 0.9765
Epoch 4/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0546 - accuracy: 0.9826
Epoch 5/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0425 - accuracy: 0.9862
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0326 - accuracy: 0.9893
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0261 - accuracy: 0.9916
Epoch 8/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0222 - accuracy: 0.9928
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0179 - accuracy: 0.9939
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0157 - accuracy: 0.9944
运行时间：  82.44712948799133
```

```
import torch
import numpy as np
from torchvision.datasets import mnist
from torch import nn
from torch.autograd import Variable
import matplotlib.pyplot as plt
import torch.nn.functional as F
from torch.utils.data import DataLoader
%matplotlib inline
def data_transform(x):
    x = np.array(x, dtype = 'float32') / 255
    x = (x - 0.5) /0.5
    x = x.reshape((-1, ))
    x = torch.from_numpy(x)
    return x
trainset = mnist.MNIST('./dataset/mnist', train=True, transform=data_transform, download=True)
testset = mnist.MNIST('./dataset/mnist', train = False, transform=data_transform, download=True)
train_data = DataLoader(trainset, batch_size=64, shuffle=True)
test_data = DataLoader(testset, batch_size=128, shuffle=False)
```

```
start = time.time()
class MLP(nn.Module):
        def __init__(self):
            super(MLP, self).__init__()
            self.fc1 = nn.Linear(28*28, 100)
            self.fc2 = nn.Linear(100, 100)
            self.fc3 = nn.Linear(100, 10)

        def forward(self, x):
            x = F.relu(self.fc1(x))
            x = F.relu(self.fc2(x))
            x = F.softmax(self.fc3(x))
            return x

mlp = MLP()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(mlp.parameters(), 1e-3)
losses = []
acces = []
eval_losses = []
eval_acces = []
```

```
for e in range(10):
    train_loss = 0
    train_acc = 0#开始训练
    mlp.train()
    for im, label in train_data:
        im = Variable(im)
        label = Variable(label)
        # 前向传播
        out = mlp(im)
        loss = criterion(out, label)
        # 反向传播
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        # 记录误差
        train_loss += loss.item()
        # 计算分类的准确率
        _, pred = out.max(1)
        num_correct = (pred == label).sum().item()
        acc = num_correct / im.shape[0]
        train_acc += acc

    losses.append(train_loss / len(train_data))
    acces.append(train_acc / len(train_data))
```

```
epoch: 1, Loss: 1.693075, Acc: 0.777202
epoch: 2, Loss: 1.624833, Acc: 0.837903
epoch: 3, Loss: 1.560809, Acc: 0.902735
epoch: 4, Loss: 1.519824, Acc: 0.943114
epoch: 5, Loss: 1.511554, Acc: 0.950610
epoch: 6, Loss: 1.508789, Acc: 0.953108
epoch: 7, Loss: 1.503639, Acc: 0.958605
epoch: 8, Loss: 1.501048, Acc: 0.960621
epoch: 9, Loss: 1.499963, Acc: 0.961454
epoch: 10, Loss: 1.497646, Acc: 0.963969
运行时间： 68.1426739692688
```

Paddle


```
import numpy as np
import paddle as paddle
import paddle.fluid as fluid
from PIL import Image
import matplotlib.pyplot as plt
import os
from paddle.fluid.dygraph import Linear

import time

from paddle.vision.transforms import Compose, Normalize
transform = Compose([Normalize(mean=[127.5],std=[127.5],data_format='CHW')])

train_dataset = paddle.vision.datasets.MNIST(mode='train', transform=transform)
test_dataset = paddle.vision.datasets.MNIST(mode='test', transform=transform)
```

```
start = time.time()
class mnist(paddle.nn.Layer):
    def __init__(self):
        super(mnist,self).__init__()
        self.fc1 = paddle.fluid.dygraph.Linear(input_dim=28*28, output_dim=100, act='relu')
        self.fc2 = paddle.fluid.dygraph.Linear(input_dim=100, output_dim=100, act='relu')
        self.fc3 = paddle.fluid.dygraph.Linear(input_dim=100, output_dim=10,act="softmax")

    def forward(self, input_):
        x = fluid.layers.reshape(input_, [input_.shape[0], -1])
        x = self.fc1(x)
        x = self.fc2(x)
        y = self.fc3(x)
        return y

from paddle.metric import Accuracy

# 用Model封装模型
model = paddle.Model(mnist())

# 定义损失函数
optim = paddle.optimizer.Adam(learning_rate=0.001, parameters=model.parameters())

# print(model.parameters())
# 配置模型
model.prepare(optim,paddle.nn.CrossEntropyLoss(),Accuracy())

# 训练保存并验证模型
model.fit(train_dataset,train_dataset,epochs=10,batch_size=64,verbose=1)
print("训练时间：",time.time()-start)
```

```
The loss value printed in the log is the current step, and the metric is the average value of previous step.
Epoch 1/10
step 938/938 [==============================] - loss: 1.6440 - acc: 0.7748 - 9ms/step
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous step.
step 938/938 [==============================] - loss: 1.6175 - acc: 0.8388 - 7ms/step
Eval samples: 60000
Epoch 2/10
step 938/938 [==============================] - loss: 1.5149 - acc: 0.8782 - 9ms/step
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous step.
step 938/938 [==============================] - loss: 1.4627 - acc: 0.9409 - 7ms/step
Eval samples: 60000
Epoch 3/10
step 938/938 [==============================] - loss: 1.4818 - acc: 0.9429 - 10ms/step
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous step.
step 938/938 [==============================] - loss: 1.4626 - acc: 0.9455 - 7ms/step
Eval samples: 60000
Epoch 4/10
step 938/938 [==============================] - loss: 1.4616 - acc: 0.9502 - 10ms/step
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous step.
step 938/938 [==============================] - loss: 1.4930 - acc: 0.9578 - 7ms/step
Eval samples: 60000
```

```
The loss value printed in the log is the current batch, and the metric is the average value of previous step.
step 938/938 [==============================] - loss: 1.4704 - acc: 0.9634 - 6ms/step
Eval samples: 60000
Epoch 7/10
step 938/938 [==============================] - loss: 1.4869 - acc: 0.9613 - 7ms/step
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous step.
step 938/938 [==============================] - loss: 1.4619 - acc: 0.9679 - 6ms/step
Eval samples: 60000
Epoch 8/10
step 938/938 [==============================] - loss: 1.4641 - acc: 0.9640 - 7ms/step
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous step.
step 938/938 [==============================] - loss: 1.4762 - acc: 0.9660 - 6ms/step
Eval samples: 60000
Epoch 9/10
step 938/938 [==============================] - loss: 1.4612 - acc: 0.9665 - 7ms/step
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous step.
step 938/938 [==============================] - loss: 1.4622 - acc: 0.9716 - 6ms/step
Eval samples: 60000
Epoch 10/10
step 938/938 [==============================] - loss: 1.5185 - acc: 0.9675 - 7ms/step
Eval begin...
The loss value printed in the log is the current batch, and the metric is the average value of previous step.
step 938/938 [==============================] - loss: 1.4629 - acc: 0.9715 - 6ms/step
Eval samples: 60000
训练时间： 126.208078622818
```

运行实录

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

```
[]  1  模型训练
```

```
1  import numpy as np
2  import paddle as paddle
3  import paddle.fluid as fluid
4  from PIL import Image
5  import matplotlib.pyplot as plt
6  import os
7  from paddle.fluid.dygraph import Linear
8  import time
9  from paddle.vision.transforms import Compose, Normalize
10 transform = Compose([Normalize(mean=[127.5],std=[127.5],data_format='CHW')])
11
12 train_dataset = paddle.vision.datasets.MNIST(mode='train', transform=transform)
13 test_dataset = paddle.vision.datasets.MNIST(mode='test', transform=transform)
14
15 start = time.time()
16 class mnist(paddle.nn.Layer):
17     def __init__(self):
18         super(mnist,self).__init__()
19         self.fc1 = paddle.fluid.dygraph.Linear(input_dim=28*28, output_dim=100, act='relu')
20         self.fc3 = paddle.fluid.dygraph.Linear(input_dim=100, output_dim=10,act="softmax")
21
22     def forward(self, input_):
23         x = fluid.layers.reshape(input_, [input_.shape[0], -1])
24         x = self.fc1(x)
25         y = self.fc3(x)
26         return y
```

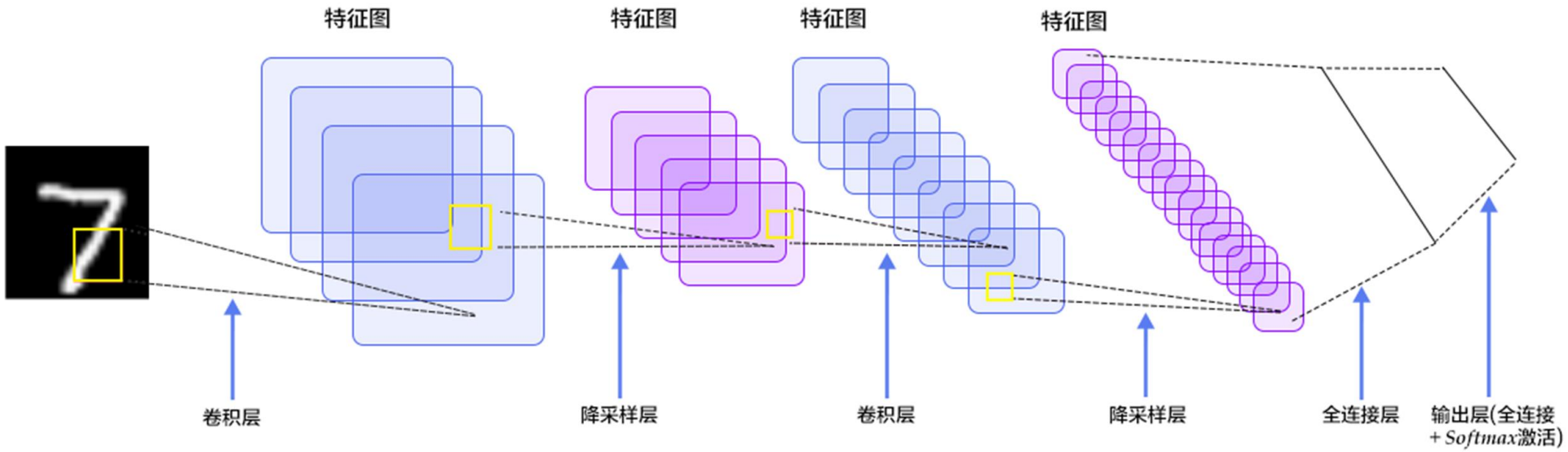
运行时间对比

10次平均运行时间：87s

10次平均运行时间：74s

10次平均运行时间： 110s

LeNet-5网络结构



参数设置为：
epochs = 10
Batch_size = 64
Learning_rate = 0.001
Optimizer为adam
损失为交叉熵
运行环境为Tesla V100，显存100G

运行时间对比

10次平均运行时间：143s

10次平均运行时间：120s

10次平均运行时间： 252s

6 总结及展望

深度学习框架对比

框　架	机　构	支持语言	Stars	Forks	Contributors
TensorFlow	Google	Python/C++/Go/...	41628	19339	568
Caffe	BVLC	C++/Python	14956	9282	221
Keras	fchollet	Python	10727	3575	322
CNTK	Microsoft	C++	9063	2144	100
MXNet	DMLC	Python/C++/R/...	7393	2745	241
Torch7	Facebook	Lua	6111	1784	113
Theano	U. Montreal	Python	5352	1868	271
Deeplearning4J	DeepLearning4J	Java/Scala	5053	1927	101
Leaf	AutumnAI	Rust	4562	216	14
Lasagne	Lasagne	Python	2749	761	55
Neon	NervanaSystems	Python	2633	573	52

Theano

由于Theano已经停止开发，不建议作为研究工具继续学习。

Tensorflow

不完美但最流行的深度学习框架，社区强大，适合工业生产环境。

Keras

入门最简单，但是不够灵活，使用受限。

Caffe

文档不够完善，但性能优异，几乎全平台支持（Caffe2），适合生产环境。

MXNet

文档略混乱，但分布式性能强大，语言支持最多，适合AWS云平台使用。

CNTK

社区不够活跃，但是性能突出，擅长语音方面的相关研究。

Pytorch

简洁，速度快，使用方便，社区活跃，备受学术界推崇。

1.简介

天元是是旷视自主研发的AI生产力平台Brain++的核心深度学习框架。天元是首个由中国AI公司研发的国产深度学习框架。

优势

训练推理一体化

支持算法训练，训练产物可以直接用于推理封装。

动静合一

整合了动态图与静态图各自的优势。

兼容并包

具备Pythonic的API。支持PyTorch Module功能。

灵活高效

多平台多设备适应能力。

1.研发背景

一.对研发过程中会有很多和社区不一样的需求及时提供技术支持；

二.在算法和技术上有底气选择自己的发展道路，而不受制于开源框架提供的能力；

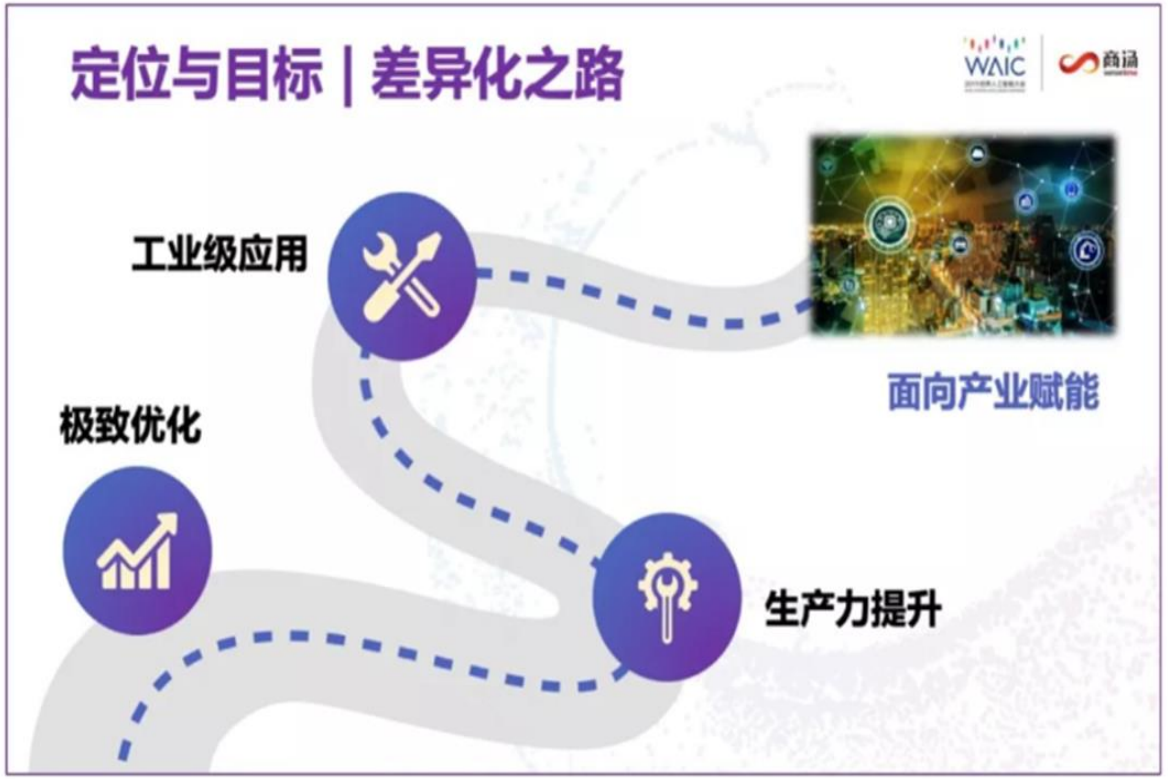
三.期望在研发训练框架的过程中，不断建立在基础系统层面的核心技术积累，让我们不仅在算法层面也在系统层面走在前列。

2.研发方向

一.首先，全面支持业务范畴内的工业级应用，特别的是支持工业级模型，包括具有复杂逻辑的动态模型的大规模训练；

二.针对商汤的业务场景，进行极致优化，建立产品级的核心竞争力；

三.我关注研究员的生产力，全面提高研发和产品迭代的速度。



感谢聆听