Pytorch简介、安装与使用

罗浩
浙江大学

大纲

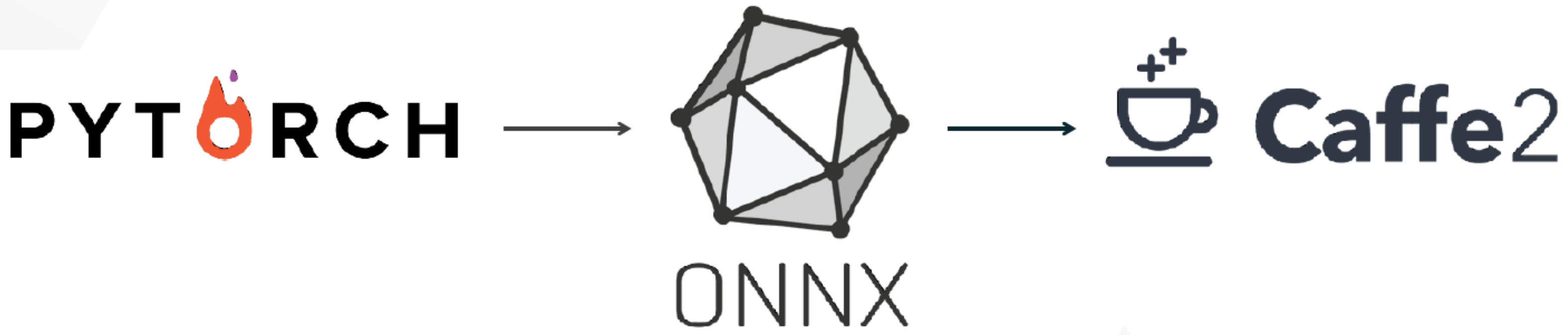
- Pytorch概述
- Pytorch的安装
- Pytorch张量
- Pytorch手写体识别

Pytorch概述

Pytorch

Research to Production at Facebook – Early 2018

ENABLING MODEL OR MODEL FRAGMENT TRANSFER



Pytorch概述

特性

- **易用的API**：像Python一样简单易用；
- **支持Python**：如上所说，PyTorch平滑地与Python数据科学栈集成。它与Numpy如此相似，你甚至都不会注意到其中的差异；
- **动态计算图**：PyTorch没有提供具有特定功能的预定义图，而是提供给我们一个框架用于构建计算图，我们甚至可以在运行时更改它们。这对于一些情况是很有用的，比如我们在创建一个神经网络时事先并不清楚需要多少内存。

Numpy

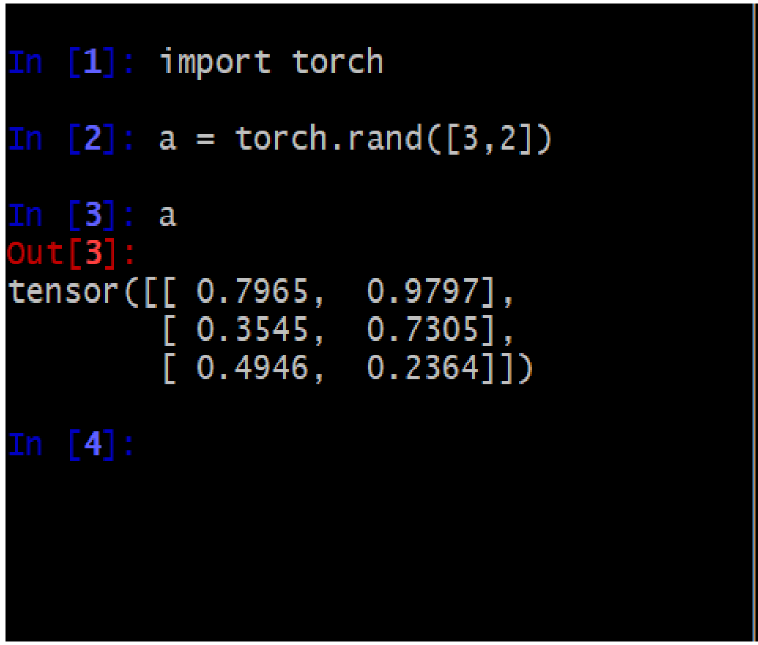
```python
# -*- coding: utf-8 -*-
import numpy as np

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = np.random.randn(N, D_in)
y = np.random.randn(N, D_out)

# Randomly initialize weights
w1 = np.random.randn(D_in, H)
w2 = np.random.randn(H, D_out)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = x.dot(w1)
    h_relu = np.maximum(h, 0)
    y_pred = h_relu.dot(w2)

    # Compute and print loss
    loss = np.square(y_pred - y).sum()
    print(t, loss)

    # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.T.dot(grad_y_pred)
    grad_h_relu = grad_y_pred.dot(w2.T)
    grad_h = grad_h_relu.copy()
    grad_h[h < 0] = 0
    grad_w1 = x.T.dot(grad_h)

    # Update weights
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

PyTorch

```python
import torch


dtype = torch.FloatTensor
# dtype = torch.cuda.FloatTensor # Uncomment this to run on GPU

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = torch.randn(N, D_in).type(dtype)
y = torch.randn(N, D_out).type(dtype)

# Randomly initialize weights
w1 = torch.randn(D_in, H).type(dtype)
w2 = torch.randn(H, D_out).type(dtype)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)

    # Compute and print loss
    loss = (y_pred - y).pow(2).sum()
    print(t, loss)

    # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    # Update weights using gradient descent
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

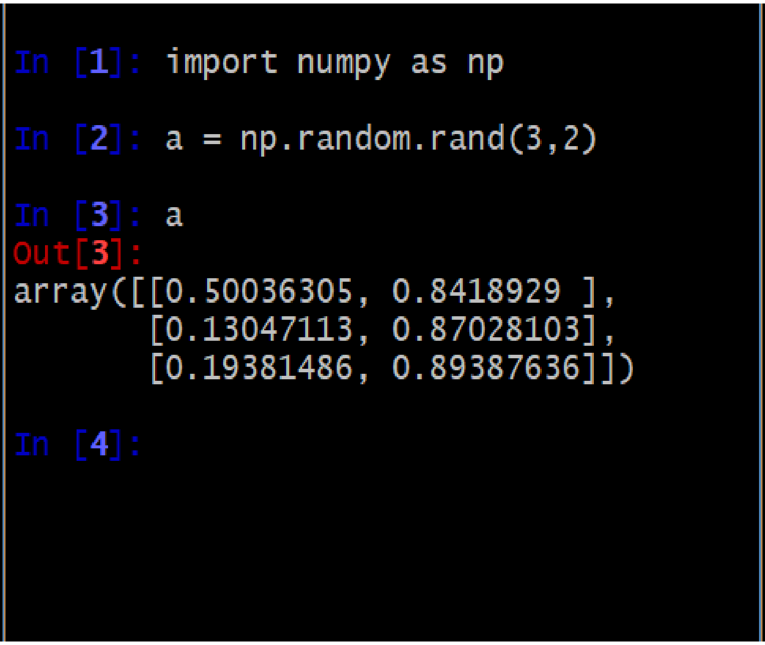
Pytorch概述

Pytorch VS Numpy VS Tensorflow



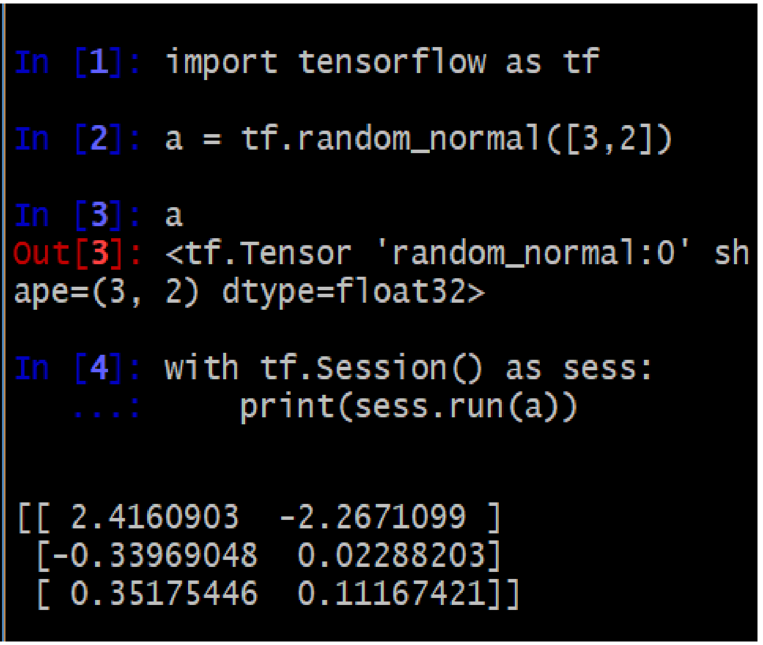
```
In [1]: import torch

In [2]: a = torch.rand([3,2])

In [3]: a
Out[3]:
tensor([[ 0.7965,  0.9797],
        [ 0.3545,  0.7305],
        [ 0.4946,  0.2364]])

In [4]:
```

Pytorch



```
In [1]: import numpy as np

In [2]: a = np.random.rand(3,2)

In [3]: a
Out[3]:
array([[0.50036305, 0.8418929 ],
       [0.13047113, 0.87028103],
       [0.19381486, 0.89387636]])

In [4]:
```

Numpy



```
In [1]: import tensorflow as tf

In [2]: a = tf.random_normal([3,2])

In [3]: a
Out[3]: <tf.Tensor 'random_normal:0' sh
ape=(3, 2) dtype=float32>

In [4]: with tf.Session() as sess:
   ...:     print(sess.run(a))


[[ 2.4160903  -2.2671099 ]
 [-0.33969048  0.02288203]
 [ 0.35175446  0.11167421]]
```

Tensorflow

Pytorch概述

常用类

- torch：类似于 NumPy 的张量库，带有强大的 GPU 支持
- torch.autograd：一个基于 tape 的自动微分库，支持 torch 中的所有的微分张量运算
- torch.nn：一个专为最大灵活性而设计、与 autograd 深度整合的神经网络库
- torch.optim：包含了各种常用的优化器，例如SGD、Adam等
- torch.multiprocessing：Python 多运算，但在运算中带有惊人的 torch 张量内存共享。这对数据加载和 Hogwild 训练很有帮助。
- torch.utils：数据加载器、训练器以及其他便利的实用功能

Pytorch概述

常用库

- **torchvision**：基于Pytorch的计算机视觉库，包含一些常用的图像处理、分类与检测的模型等；
- **torchtext**：基于Pytorch的自然语言处理库，包含一些常用的文本处理、词嵌入、分词等；
- **torchaudio**：基于Pytorch的语音识别库，包含一些常用的语音预处理、语音识别模型等；
- **tensorboardX**：基于Pytorch的tensorboard，可以可视化模型的训练过程与结果；

Pytorch概述

常用链接

- **Pytorch：** https://pytorch.org/
- **Github：** https://github.com/pytorch/pytorch
- **Pytorch API：** https://pytorch.org/docs/master/
- **Pytorch tutorials：** https://pytorch.org/tutorials/

Pytorch的安装

推荐安装

- Ubuntu系统
- 显卡驱动（CUDA、CuDNN ）
- Python, pip（本次课程推荐2.7版本，自行安装)
- Pytorch框架（本次课程推荐0.4.0版本)
- Torchvision等常用库

AI300学院

Pytorch的安装

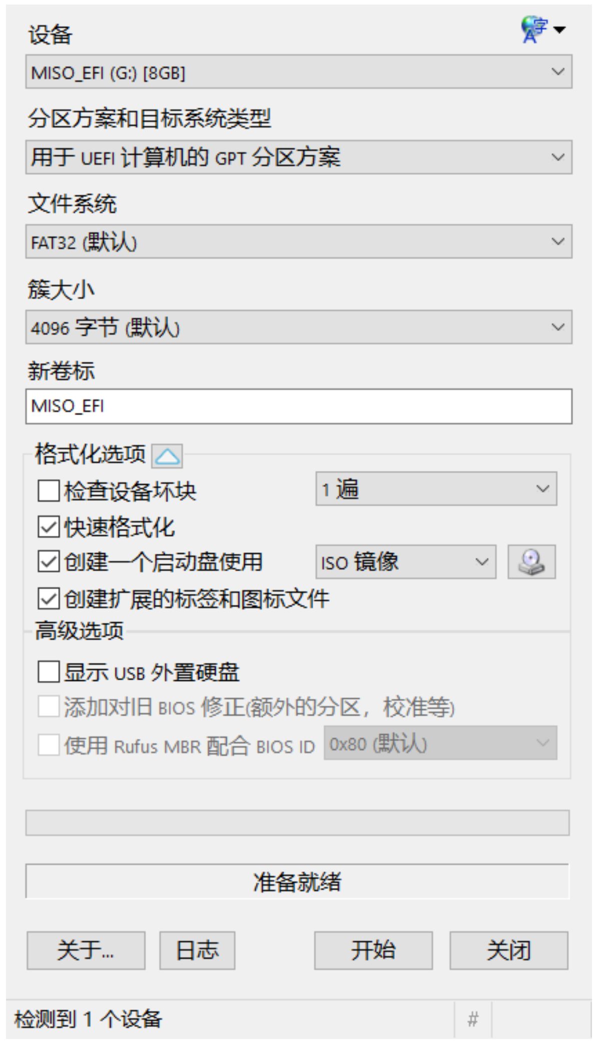
Ubuntu系统安装

- **1.确认主板BIOS是 UEFI 模式**

首先确认下你的主板的BIOS是UEFI模式。进BIOS看，一般是开机按F2键，或者Del键不同的电脑不一样。

- **2.制作一个U盘启动盘**

到Ubuntu官网下载镜像，选择Ubuntu 16.04.4，然后到这里下载Rufus刻录软件，点击 打开，插入U盘，按右图配置，然后写入就完成镜像刻录了。


设备
MISO_EFI (G:) [8GB]
分区方案和目标系统类型
用于 UEFI 计算机的 GPT 分区方案
文件系统
FAT32 (默认)
簇大小
4096 字节 (默认)
新卷标
MISO_EFI
格式化选项
☐ 检查设备坏块　1 遍
☑ 快速格式化
☑ 创建一个启动盘使用　ISO 镜像
☑ 创建扩展的标签和图标文件
高级选项
☐ 显示 USB 外置硬盘
☐ 添加对旧 BIOS 修正(额外的分区，校准等)
☐ 使用 Rufus MBR 配合 BIOS ID　0x80 (默认)
准备就绪

检测到 1 个设备

AI300学院

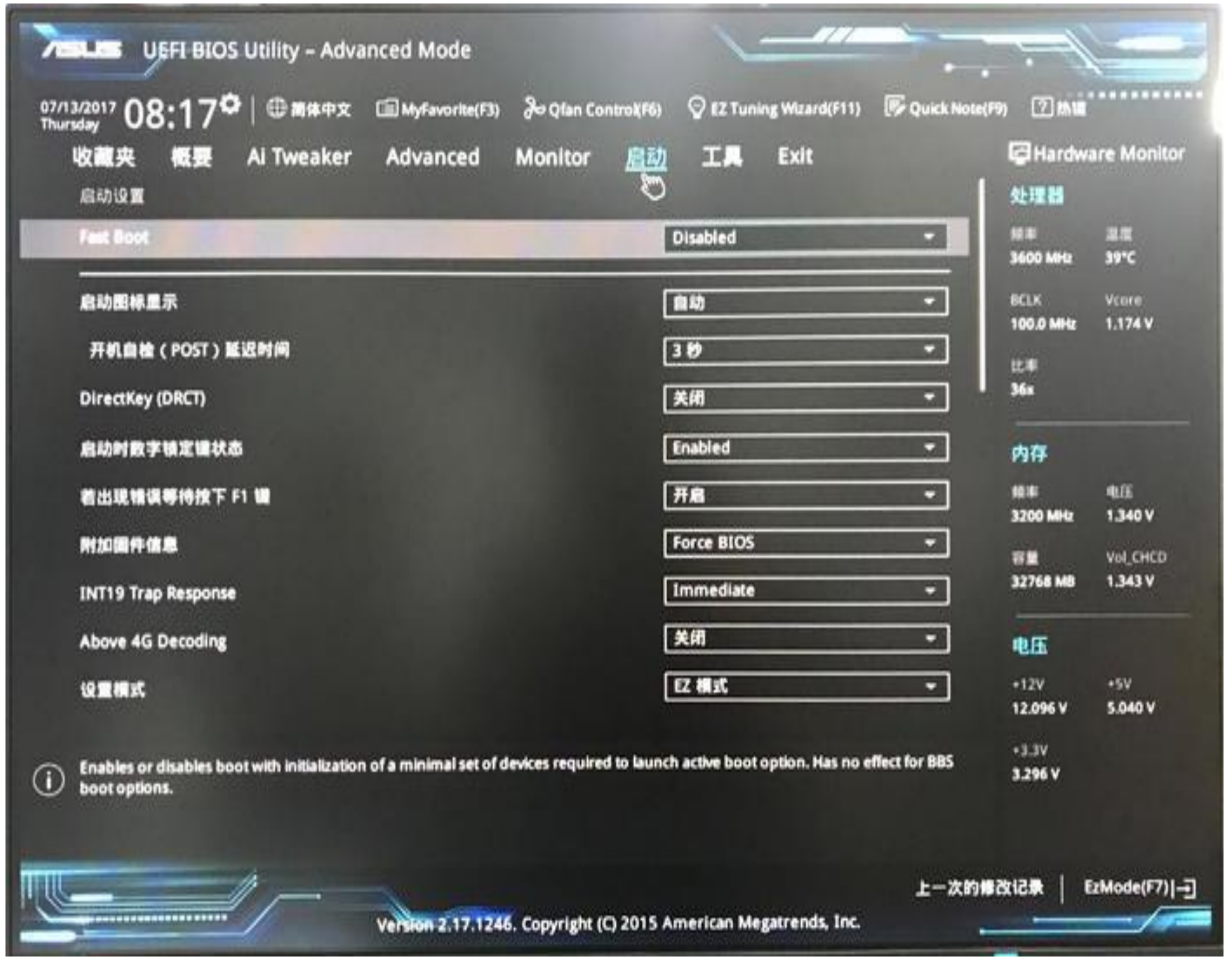
Pytorch的安装

Ubuntu系统安装

3.禁用主板的 Secure Boot

如果你想要装Windows和 Linux 双操作系统，那么必须要禁用主板的 Secure Boot， 因为主板内置的公钥只有微软的。在Ubuntu 下安装 Nvidia 驱动的时候，会警告 UEFI Secure Boot is not compatible with the use of third-party drivers. 如果没有禁用Secure Boot, 安装了显卡驱动后，开机，输入密码时，会进不去系统，循环登录。

- 进入 UEFI BIOS界面：选择 Boot->Secure Boot-> Key Management -> Save Secure Boot Keys，插入U盘，备份key到这个U盘，会有四个文件, PK, KEK, DB 和 DBX 写入到U盘。
- 删除 Platform Key. 选择 Delete PK，点击OK确认删除。删除后回到上一级菜单，可以看到 Secure Boot 已经变成 disabled 了。



启动设置

Fast Boot	Disabled
启动图标显示	自动
开机自检（POST）延迟时间	3 秒
DirectKey (DRCT)	关闭
启动时数字锁定键状态	Enabled
若出现错误等待按下 F1 键	开启
附加固件信息	Force BIOS
INT19 Trap Response	Immediate
Above 4G Decoding	关闭
设置模式	EZ 模式

Enables or disables boot with initialization of a minimal set of devices required to launch active boot option. Has no effect for BBS boot options.

处理器

频率	温度
3600 MHz	39°C
BCLK	Vcore
100.0 MHz	1.174 V
比率	
36x	

内存

频率	电压
3200 MHz	1.340 V
容量	Vol_CHCD
32768 MB	1.343 V

电压

+12V	+5V
12.096 V	5.040 V
+3.3V	
3.296 V	

Version 2.17.1246. Copyright (C) 2015 American Megatrends, Inc.

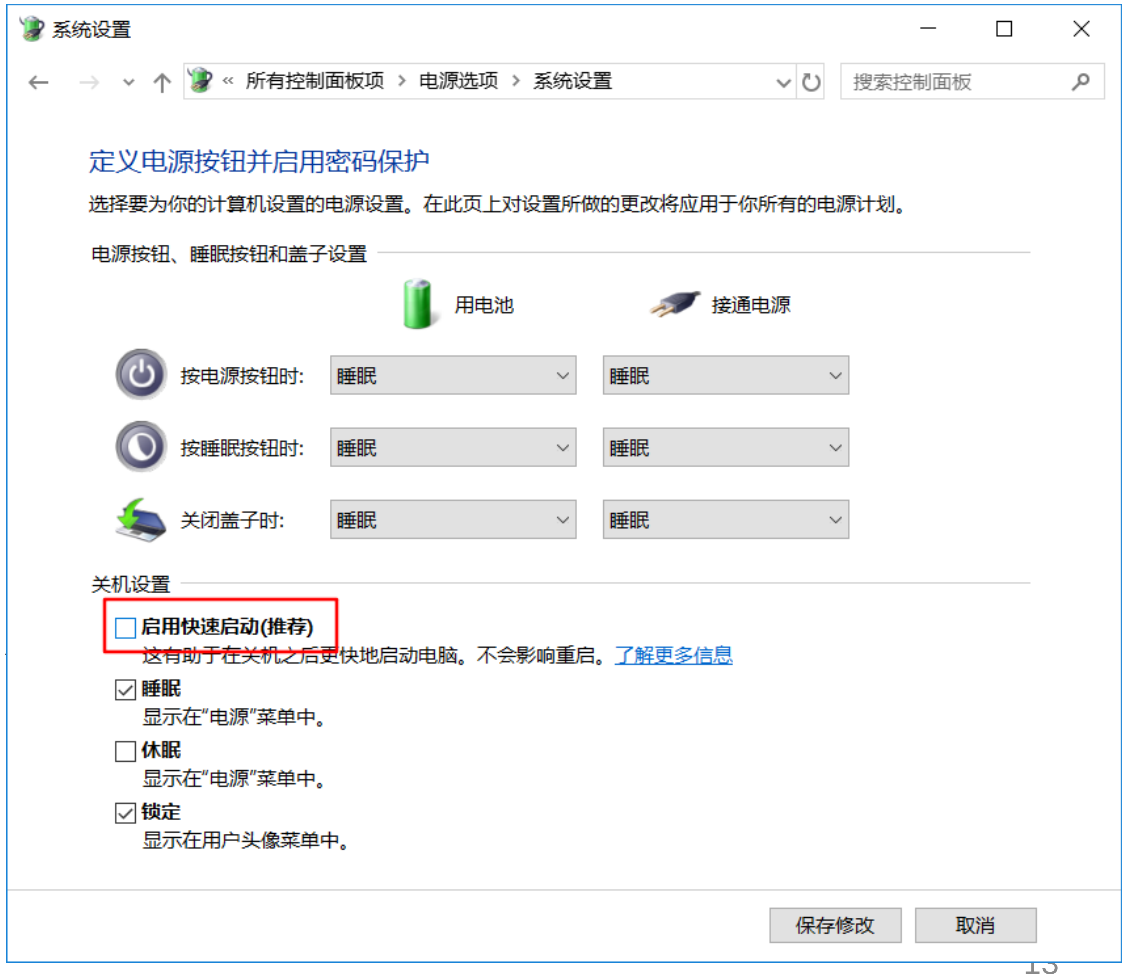
Pytorch的安装

Ubuntu系统安装

4.Windows 和 Ubuntu 安装在同一块硬盘上

开机，按住 DEL 键进入BIOS，点击Boot Menu(F8)菜单，选择从U盘启动，且注意要选择UEFI模式的U盘

- 开始安装Windows, 安装完后，重启，进入Windows 后，关闭Windows的 Fast Startup，即进入控制面板->电源，找到 Fast Startup，禁用掉。
- 把 Ubuntu U盘启动盘插上，开机，按DEL键进入BIOS，选择从这个U盘启动，要选择UEFI模式的U盘，开始安装，安装类型最好选择***其他选项***。


系统设置

所有控制面板项 › 电源选项 › 系统设置

搜索控制面板

定义电源按钮并启用密码保护

选择要为你的计算机设置的电源设置。在此页上对设置所做的更改将应用于你所有的电源计划。

电源按钮、睡眠按钮和盖子设置

	用电池	接通电源
按电源按钮时:	睡眠	睡眠
按睡眠按钮时:	睡眠	睡眠
关闭盖子时:	睡眠	睡眠

关机设置

- [ ] 启用快速启动(推荐)
  这有助于在关机之后更快地启动电脑。不会影响重启。了解更多信息
- [x] 睡眠
  显示在"电源"菜单中。
- [ ] 休眠
  显示在"电源"菜单中。
- [x] 锁定
  显示在用户头像菜单中。

Pytorch的安装

Ubuntu系统安装

5.Windows 和 Ubuntu 分别安装不同的硬盘上

- 关机，把Windows这块硬盘拆下来，当做它不存在。
- 开始安装Ubuntu，安装到另一块硬盘上。选择 Erase disk and install Ubuntu，接下来选择安装到硬盘上。
- 把 Windows 那块硬盘在接入主板。
- 这样，两块硬盘就分别安装了独立的操作系统，具体进入哪个，由UEFI BIOS里的启动优先级来决定。你想默认启动Windows，那就把Windows那块硬盘拖动到第一的位置，如果你想默认启动Ubuntu，那就把Ubuntu那块硬盘拖动到第一的位置。

Pytorch的安装

Ubuntu系统安装

6.注意事项

- 安装Ubuntu分区只用分两个区（内存大于4G）
  - / （主分区）
  - /home（逻辑分区）

- 安装一定要选择安装英文版，因为后续你不会想进行：
  - “cd 桌面” 或者 “cd 下载”
- 如果是单硬盘双系统，那么安装完后一般是grub引导，如果是两个系统安装在不同的硬盘，那么可以手动进BIOS选启动盘。
- 安装ubuntu时可以最后选择把引导安装在 / 目录下对应的分区。比如/dev/sda2。
- 安装完Ubuntu后可能存在一系列问题，比如双系统时间和Windows差8个小时，解决方法可以参考https://wxzs5.github.io/2018/03/01/configure-ubuntu/。

AI300学院

Pytorch的安装

NVIDIA显卡配置

1. 先卸载原有N卡驱动（可选）

- #如果apt-get安装:
  *sudo apt-get remove --purge nvidia**
- #如果是runfile安装:
  *sudo chmod +x *.run*
  sudo ./NVIDIA-Linux-x86_64.run –uninstall*

2. 禁用nouveau驱动

- *sudo vim /etc/modprobe.d/blacklist.conf*在文本最后添加：（禁用nouveau第三方驱动，之后也不需要改回来）
  blacklist nouveau
  options nouveau modeset=0
- 然后执行：*sudo update-initramfs -u*
- 重启后，执行：*lsmod | grep nouveau*。如果没有屏幕输出，说明禁用nouveau成功。

AI300学院

Pytorch的安装

NVIDIA显卡配置

3. 禁用X-Window服务

- *sudo service lightdm stop* #这会关闭图形界面按Ctrl-Alt+F1进入命令行界面，输入用户名和密码登录即可。在命令行输入：*sudo service lightdm start*，然后按Ctrl-Alt+F7即可恢复到图形界面。

4. 命令行安装驱动

#给驱动run文件赋予执行权限：

sudo chmod +x NVIDIA-Linux-x86_64-.run*

#后面的参数非常重要，不可省略：

sudo ./NVIDIA-Linux-x86_64-.run –no-opengl-files*

–no-opengl-files：表示只安装驱动文件，不安装OpenGL文件。这个参数不可省略，否则会导致登陆界面死循环。

–no-x-check：表示安装驱动时不检查X服务，非必需。

–no-nouveau-check：表示安装驱动时不检查nouveau，非必需。

-Z, --disable-nouveau：禁用nouveau。此参数非必需，因为之前已经手动禁用了nouveau。

-A：查看更多高级选项。

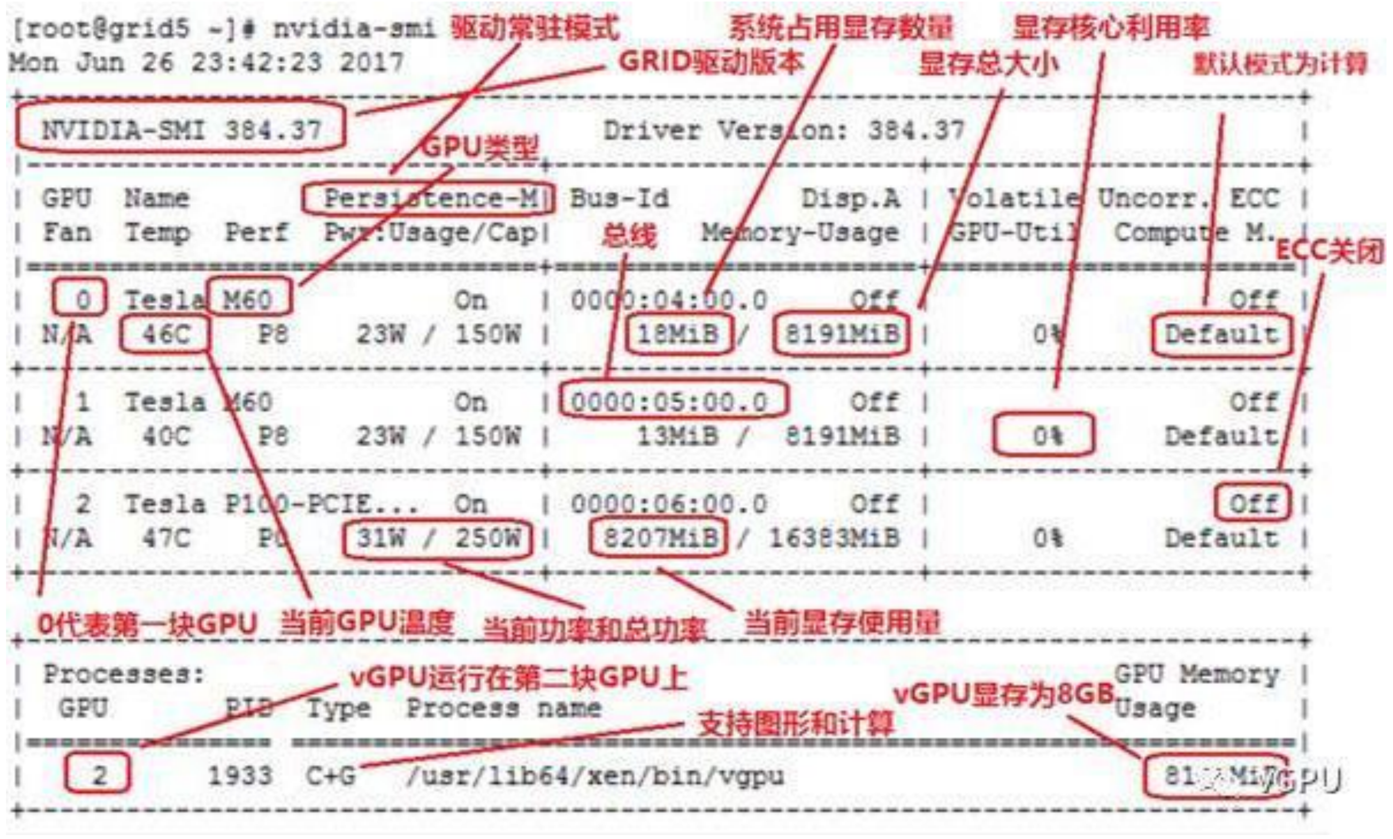
必选参数解释：**因为NVIDIA的驱动默认会安装OpenGL，而Ubuntu的内核本身也有OpenGL、且与GUI显示息息相关，一旦NVIDIA的驱动覆写了OpenGL，在GUI需要动态链接OpenGL库的时候就引起问题。**

之后，按照提示安装，成功后重启即可。 如果提示安装失败，不要急着重启电脑，重复以上步骤，多安装几次即可。

AI300学院

Pytorch的安装

NVIDIA显卡配置



```
[root@grid5 ~]# nvidia-smi
Mon Jun 26 23:42:23 2017
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.37                 Driver Version: 384.37                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla M60           On   | 0000:04:00.0     Off |                  Off |
| N/A   46C    P8    23W / 150W |     18MiB /  8191MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla M60           On   | 0000:05:00.0     Off |                  Off |
| N/A   40C    P8    23W / 150W |     13MiB /  8191MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla P100-PCIE...  On   | 0000:06:00.0     Off |                  Off |
| N/A   47C    P0    31W / 250W |   8207MiB / 16383MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|    2      1933   C+G   /usr/lib64/xen/bin/vgpu                     81[illegible]MiB |
+-----------------------------------------------------------------------------+
```

5.Driver测试：

nvidia-smi #若列出GPU的信息列表，表示驱动安装成功

nvidia-settings #若弹出设置对话框，亦表示驱动安装成功

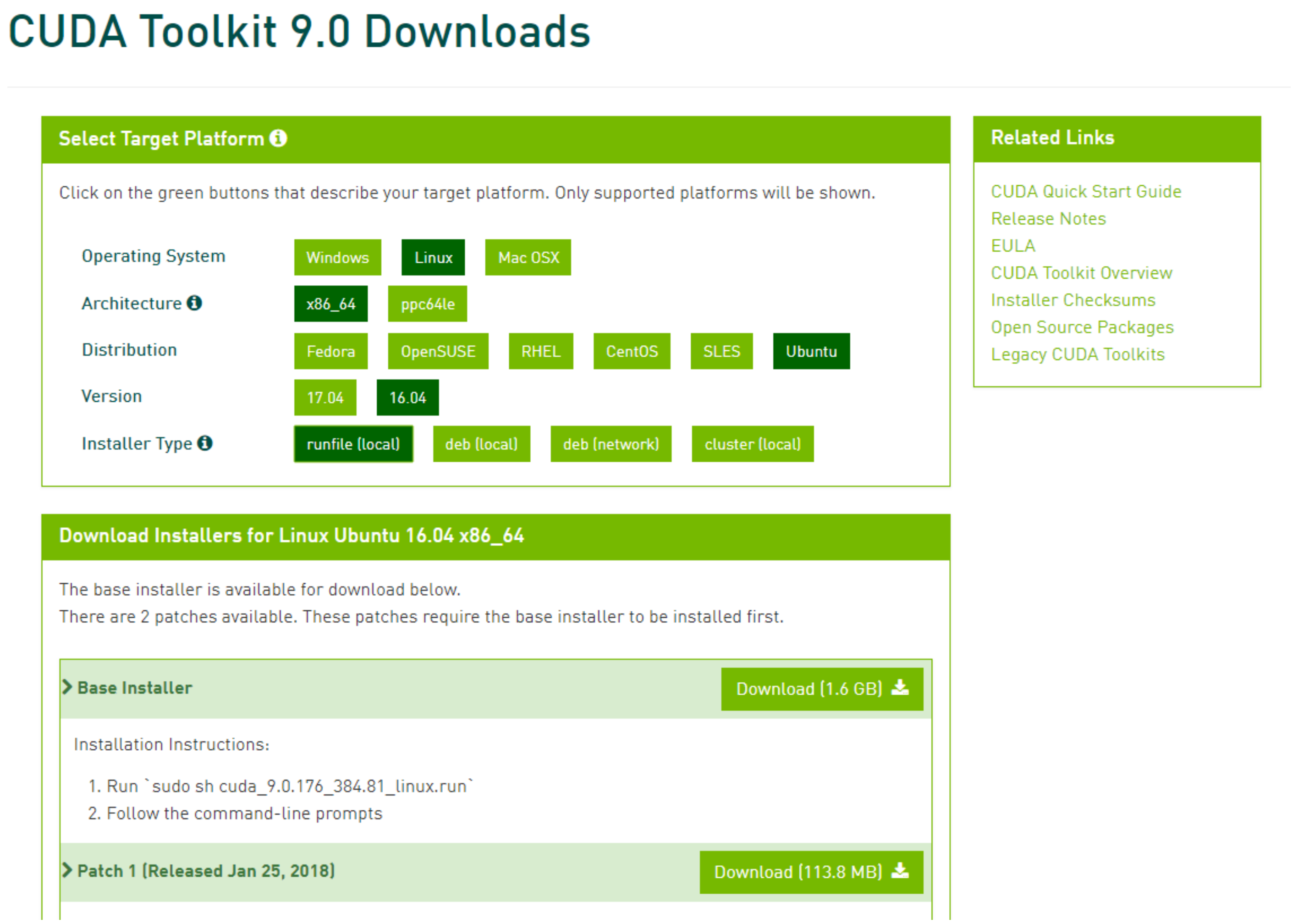
Pytorch的安装

NVIDIA显卡配置

6. 安装CUDA
到官网下载cuda包， runfile

建议安装版本：cuda8.0或cuda 9.0

官网：
https://developer.nvidia.com/cuda-downloads



CUDA Toolkit 9.0 Downloads

Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System	Windows [✓ Linux] Mac OSX
Architecture	[✓ x86_64] ppc64le
Distribution	Fedora OpenSUSE RHEL CentOS SLES [✓ Ubuntu]
Version	17.04 [✓ 16.04]
Installer Type	[✓ runfile (local)] deb (local) deb (network) cluster (local)

Related Links

- CUDA Quick Start Guide
- Release Notes
- EULA
- CUDA Toolkit Overview
- Installer Checksums
- Open Source Packages
- Legacy CUDA Toolkits

Download Installers for Linux Ubuntu 16.04 x86_64

The base installer is available for download below.
There are 2 patches available. These patches require the base installer to be installed first.

Base Installer Download (1.6 GB)

Installation Instructions:

1. Run `sudo sh cuda_9.0.176_384.81_linux.run`
2. Follow the command-line prompts

Patch 1 (Released Jan 25, 2018) Download (113.8 MB)

Pytorch的安装

NVIDIA显卡配置

6. 安装CUDA

sudo ./cuda_*linux.run --no-opengl-libs

--no-opengl-libs：表示只安装驱动文件，不安装OpenGL文件。必需参数，原因同上。注意：不是-no-opengl-files。

--uninstall (deprecated)：用于卸载CUDA Driver（已废弃）。

--toolkit：表示只安装CUDA Toolkit，不安装Driver和Samples。

--help：查看更多高级选项。

之后，按照提示安装即可。依次选择：

accept #同意安装

n #不安装Driver，因为已安装最新驱动

y #安装CUDA Toolkit

<Enter> #安装到默认目录

y #创建安装目录的软链接

n #不复制Samples，因为在安装目录下有/samples

AI300学院

Pytorch的安装

NVIDIA显卡配置

7.CUDA Sample测试：

#编译并测试设备 deviceQuery：

cd /usr/local/cuda-9.0/samples/1_Utilities/deviceQuery

sudo make

./deviceQuery

如果测试的最后结果是Result = PASS， 说明CUDA安装成功啦！

8.把cuda加到环境变量中。

- 在.bashrc中配置
  - *vim ~/.bashrc*
  - *加入以下变量：*
  - *export PATH=/usr/local/cuda-9.0/bin${PATH:+:${PATH}}*
  - *export LD_LIBRARY_PATH=/usr/local/cuda-9.0/lib64\${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}*

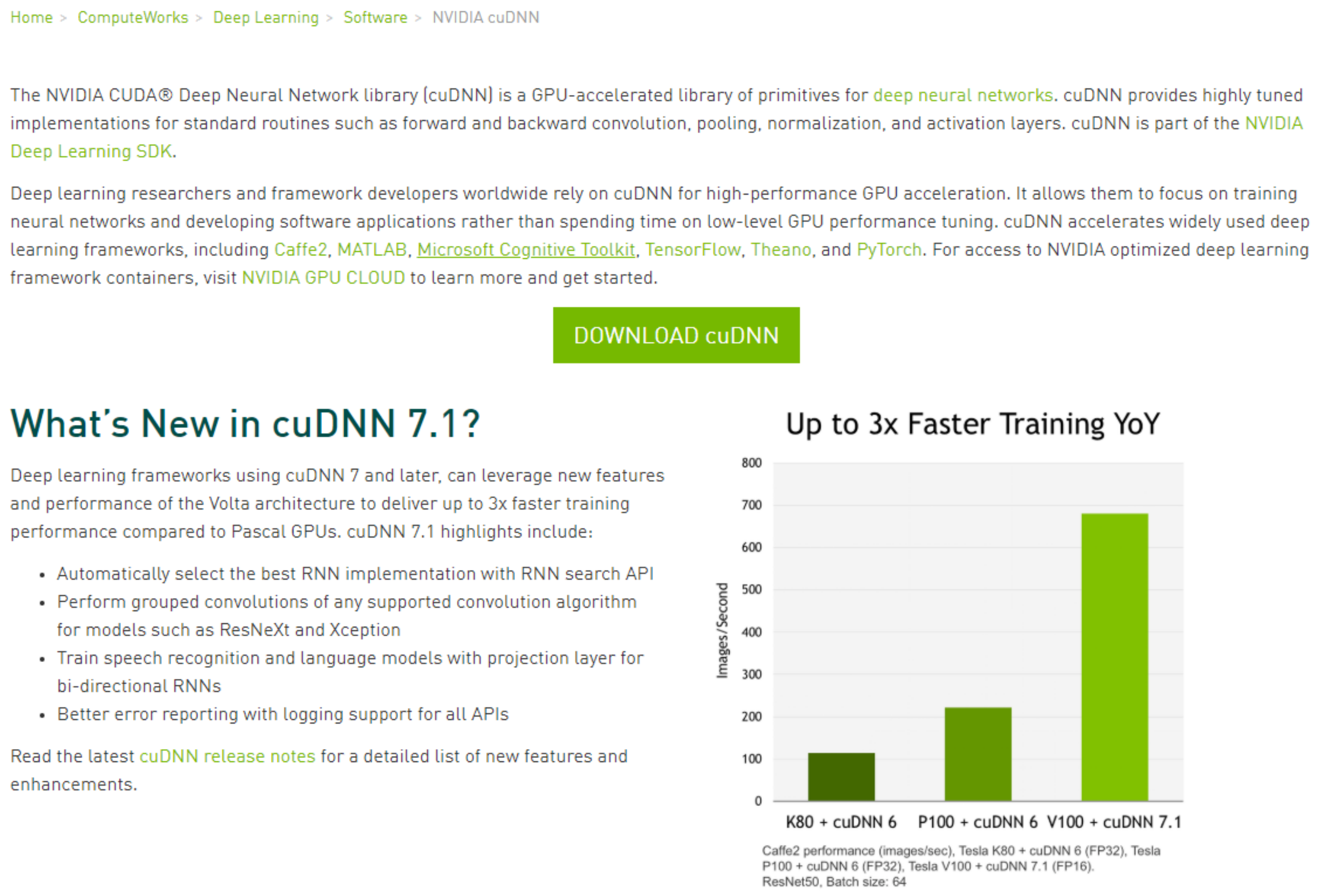
Pytorch的安装

NVIDIA显卡配置

9.cudnn 安装

- 到官网(https://developer.nvidia.com/cudnn)下载cuda包，要注册一个Nvidia账号，然后才可以下载，建议的版本
- cudnn 7.0 for cuda 9.0

- 参考 Nvidia 官方教程（https://docs.nvidia.com/deeplearning/sdk/cudnn-install/index.html）安装，然后关于nvidia驱动的一切就都配置好了。



Home > ComputeWorks > Deep Learning > Software > NVIDIA cuDNN

The NVIDIA CUDA® Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. cuDNN provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. cuDNN is part of the NVIDIA Deep Learning SDK.

Deep learning researchers and framework developers worldwide rely on cuDNN for high-performance GPU acceleration. It allows them to focus on training neural networks and developing software applications rather than spending time on low-level GPU performance tuning. cuDNN accelerates widely used deep learning frameworks, including Caffe2, MATLAB, Microsoft Cognitive Toolkit, TensorFlow, Theano, and PyTorch. For access to NVIDIA optimized deep learning framework containers, visit NVIDIA GPU CLOUD to learn more and get started.

DOWNLOAD cuDNN

What's New in cuDNN 7.1?

Deep learning frameworks using cuDNN 7 and later, can leverage new features and performance of the Volta architecture to deliver up to 3x faster training performance compared to Pascal GPUs. cuDNN 7.1 highlights include:

- Automatically select the best RNN implementation with RNN search API
- Perform grouped convolutions of any supported convolution algorithm for models such as ResNeXt and Xception
- Train speech recognition and language models with projection layer for bi-directional RNNs
- Better error reporting with logging support for all APIs

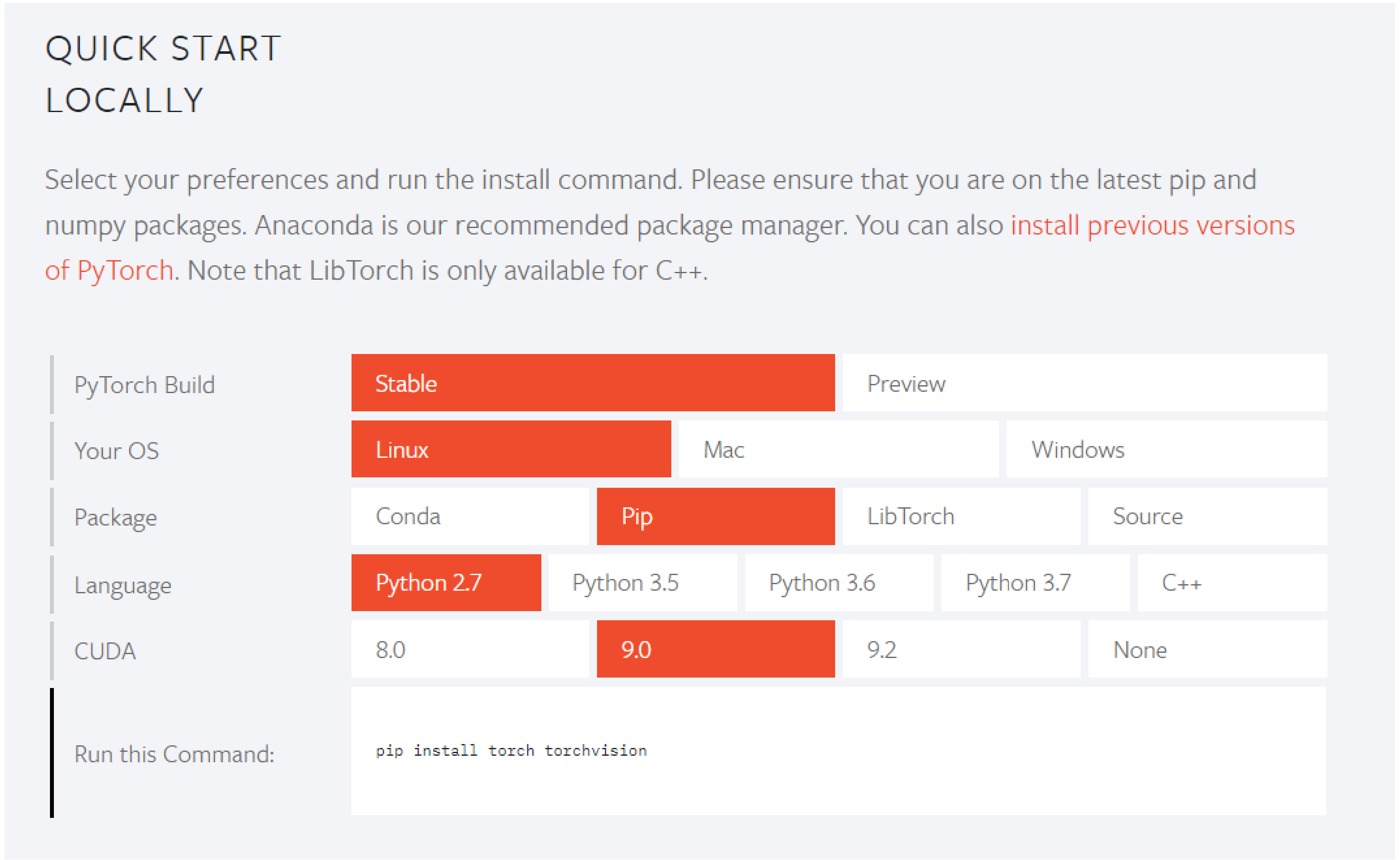
Read the latest cuDNN release notes for a detailed list of new features and enhancements.

Up to 3x Faster Training YoY

Caffe2 performance (images/sec), Tesla K80 + cuDNN 6 (FP32), Tesla P100 + cuDNN 6 (FP32), Tesla V100 + cuDNN 7.1 (FP16). ResNet50, Batch size: 64

Pytorch的安装

Pytorch与torchvision安装



QUICK START
LOCALLY

Select your preferences and run the install command. Please ensure that you are on the latest pip and numpy packages. Anaconda is our recommended package manager. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

PyTorch Build	[✓ Stable]	Preview			
Your OS	[✓ Linux]	Mac	Windows		
Package	Conda	[✓ Pip]	LibTorch	Source	
Language	[✓ Python 2.7]	Python 3.5	Python 3.6	Python 3.7	C++
CUDA	8.0	[✓ 9.0]	9.2	None	

Run this Command:

```
pip install torch torchvision
```

Pytorch：https://pytorch.org/

进入官网选择相对应的版本

Pytorch张量

创建一个张量

```
from __future__ import print_function
import torch
```

Construct a 5x3 matrix, uninitialized:

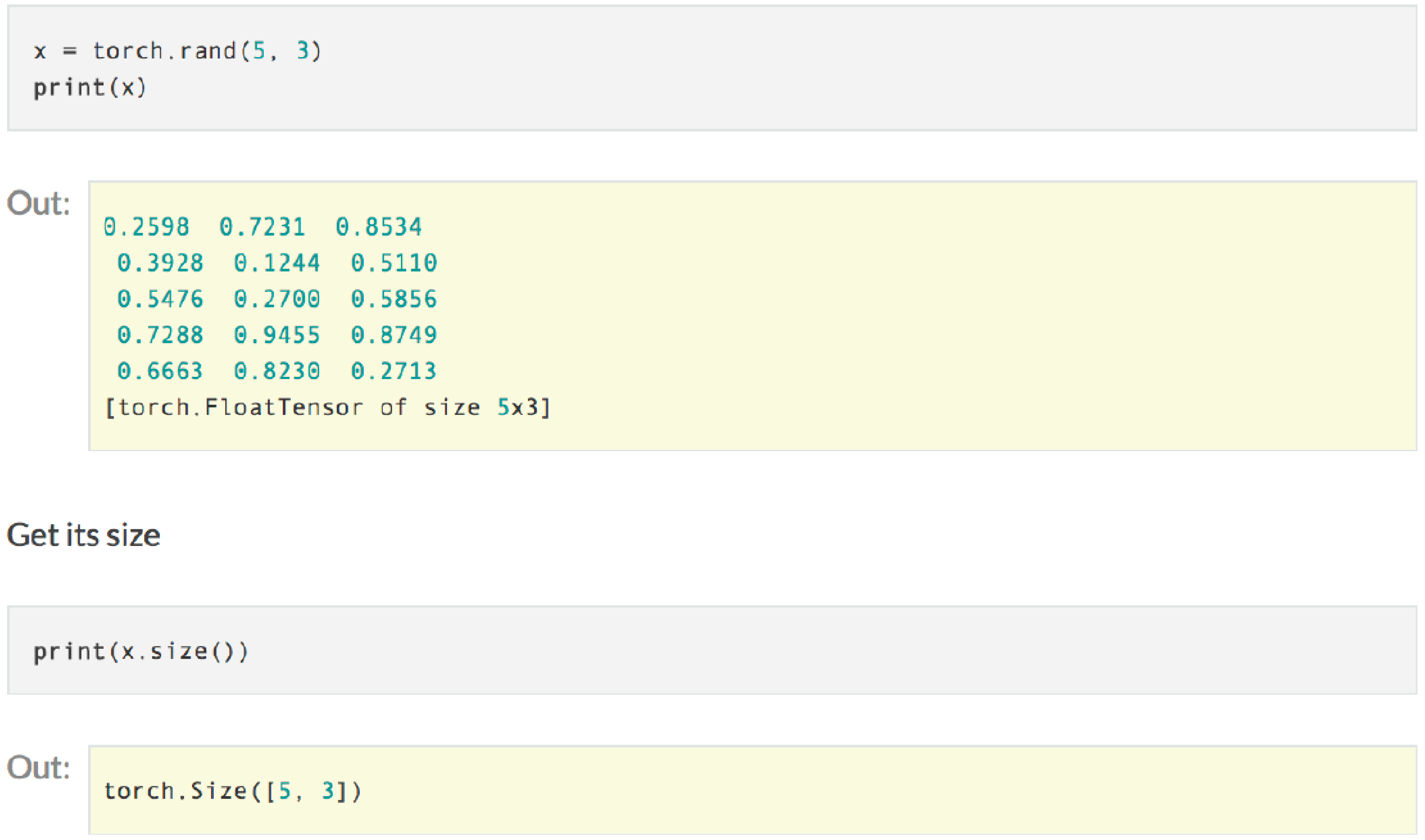
```
x = torch.Tensor(5, 3)
print(x)
```

Out:

```
1.00000e-25 *
  0.4136  0.0000  0.0000
  0.0000  1.6519  0.0000
  1.6518  0.0000  1.6519
  0.0000  1.6518  0.0000
  1.6520  0.0000  1.6519
[torch.FloatTensor of size 5x3]
```

Pytorch张量

随机初始化与张量的size



```
x = torch.rand(5, 3)
print(x)
```

Out:

```
0.2598  0.7231  0.8534
 0.3928  0.1244  0.5110
 0.5476  0.2700  0.5856
 0.7288  0.9455  0.8749
 0.6663  0.8230  0.2713
[torch.FloatTensor of size 5x3]
```

Get its size

```
print(x.size())
```

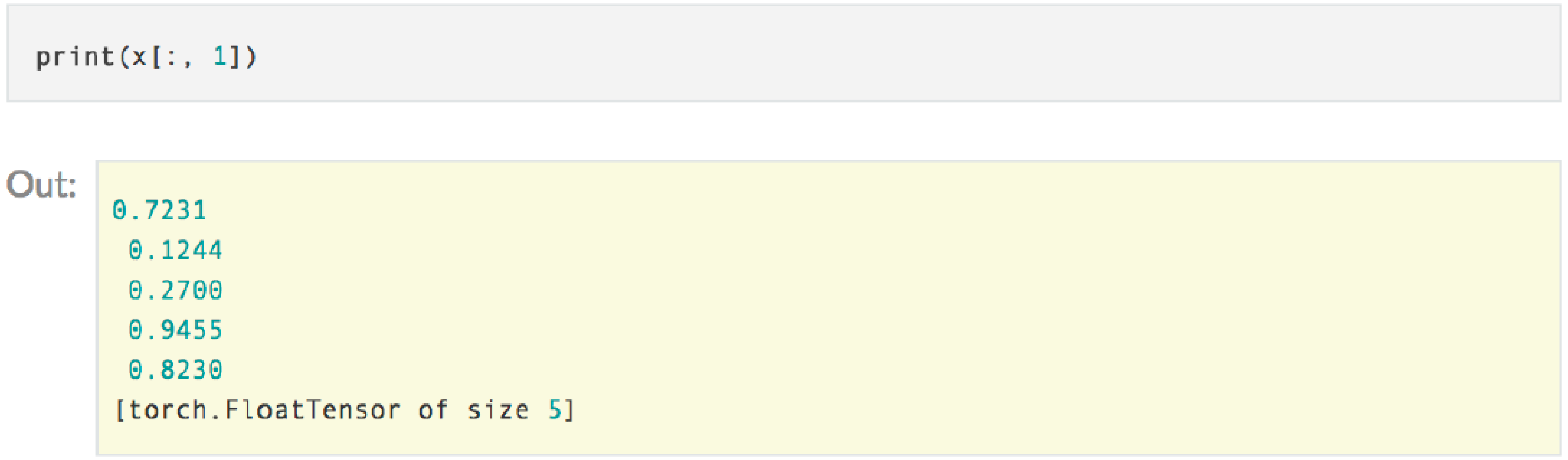
Out:

```
torch.Size([5, 3])
```

AI300学院

Pytorch张量

张量的切片



```
print(x[:, 1])
```

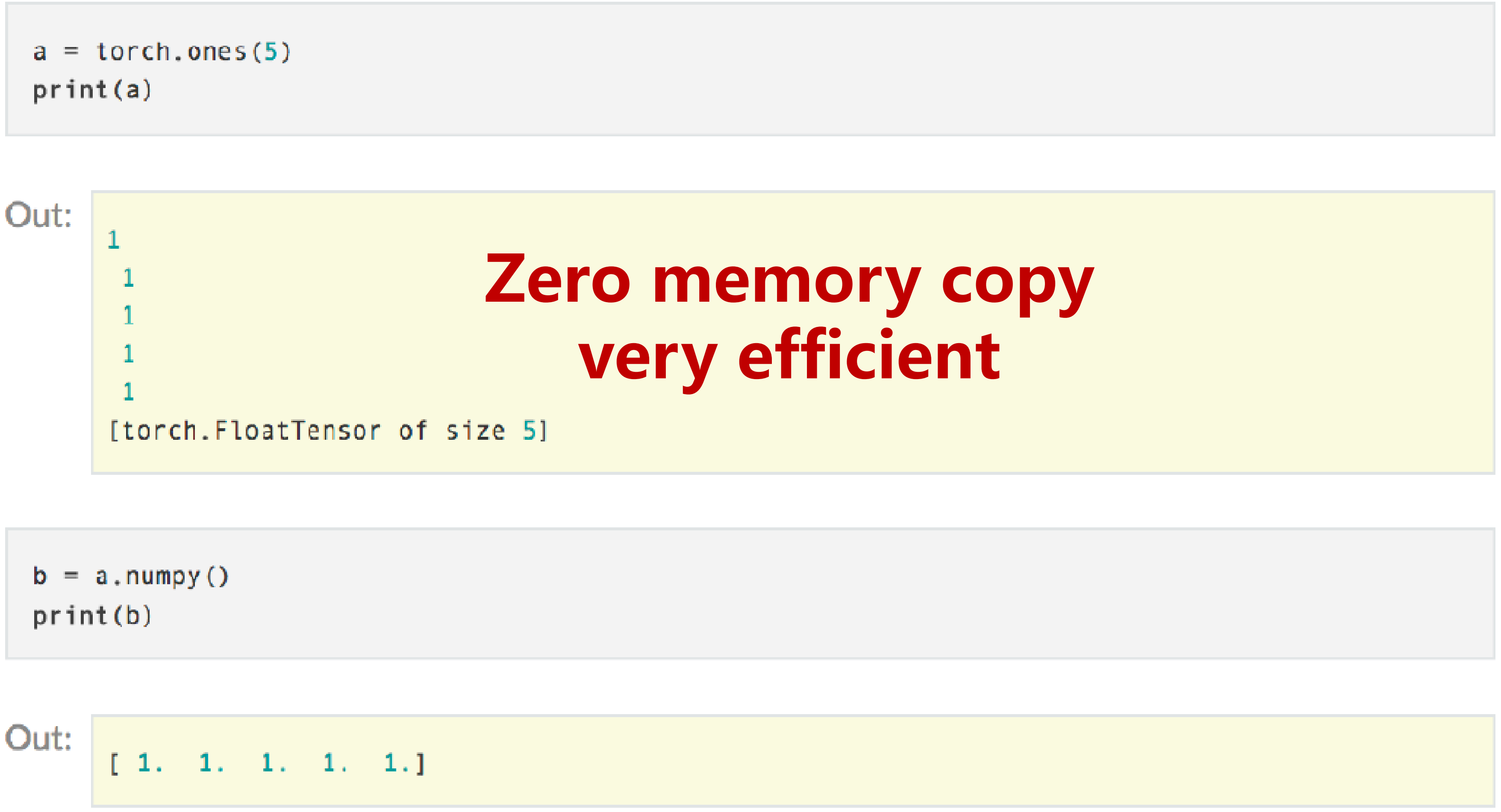
Out:

```
0.7231
 0.1244
 0.2700
 0.9455
 0.8230
[torch.FloatTensor of size 5]
```

与Numpy完全一致

Pytorch张量

Tensor 转 Numpy



```
a = torch.ones(5)
print(a)
```

Out:

```
1
 1
 1
 1
 1
[torch.FloatTensor of size 5]
```

Zero memory copy
very efficient

```
b = a.numpy()
print(b)
```

Out:

```
[ 1.  1.  1.  1.  1.]
```

Pytorch张量

Numpy 转 Tensor

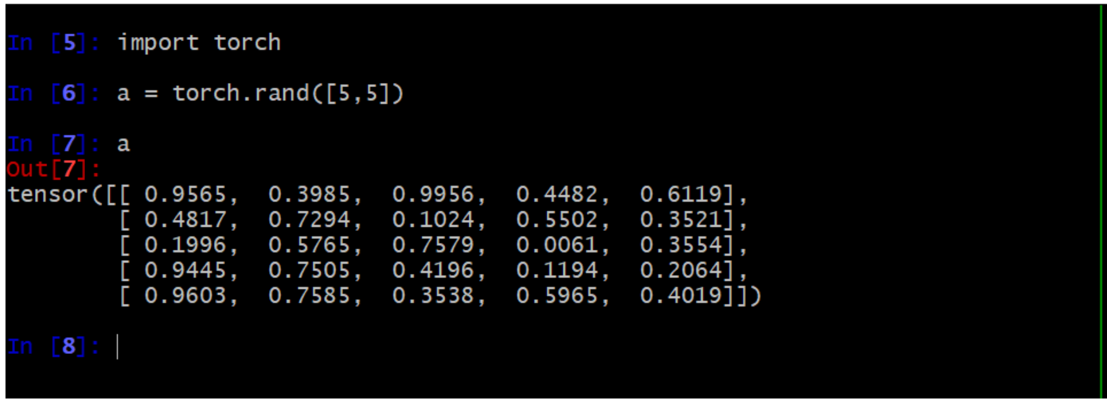
```
import numpy as np
a = np.ones(5)
b = torch.from_numpy(a)
np.add(a, 1, out=a)
print(a)
print(b)
```

Out:

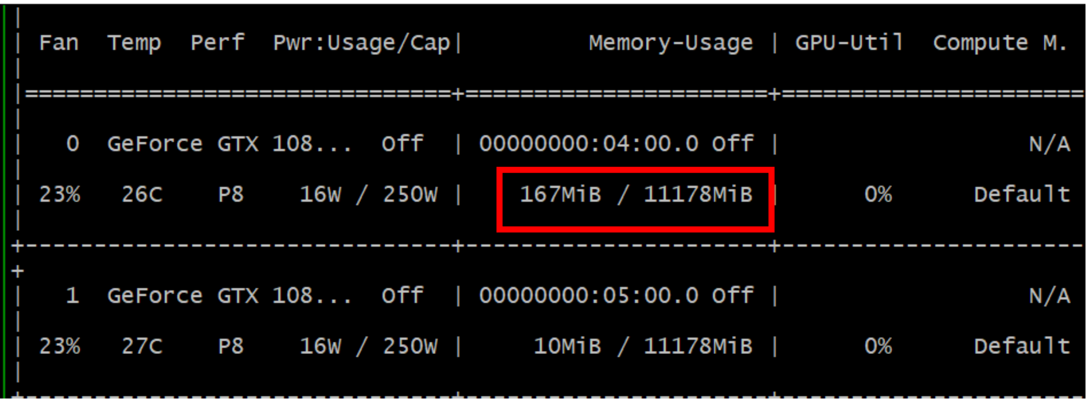
```
[ 2.  2.  2.  2.  2.]

 2
 2
 2
 2
 2
[torch.DoubleTensor of size 5]
```

Pytorch张量

将张量与模型放到GPU

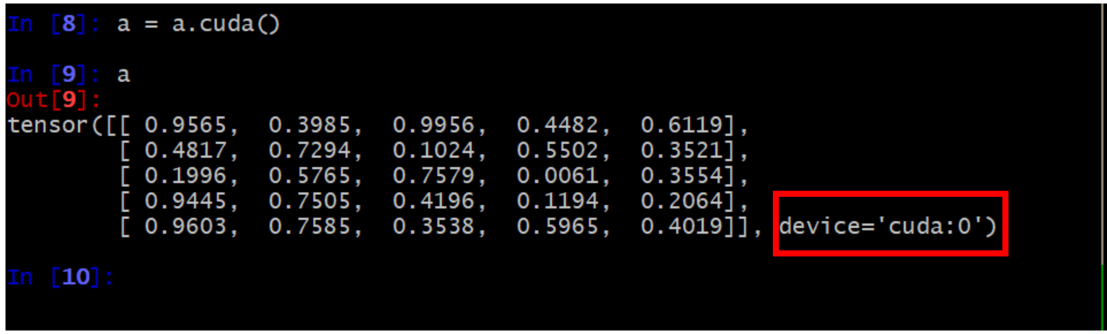


```
In [5]: import torch

In [6]: a = torch.rand([5,5])

In [7]: a
Out[7]:
tensor([[ 0.9565,  0.3985,  0.9956,  0.4482,  0.6119],
        [ 0.4817,  0.7294,  0.1024,  0.5502,  0.3521],
        [ 0.1996,  0.5765,  0.7579,  0.0061,  0.3554],
        [ 0.9445,  0.7505,  0.4196,  0.1194,  0.2064],
        [ 0.9603,  0.7585,  0.3538,  0.5965,  0.4019]])

In [8]: |
```

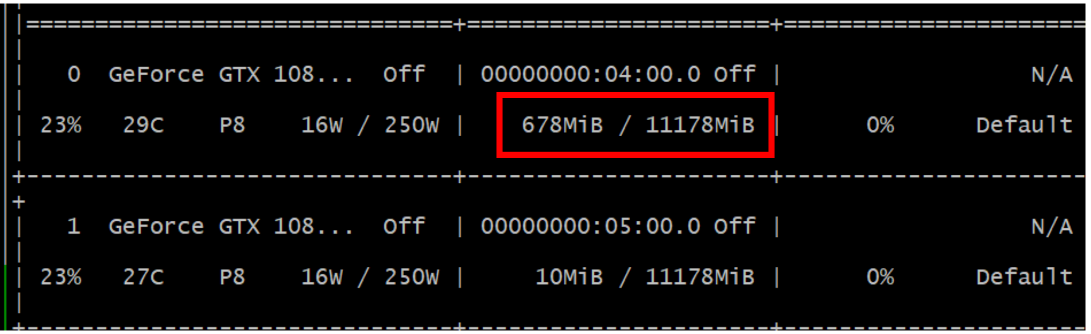


```
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 00000000:04:00.0 Off |                  N/A |
| 23%   26C    P8    16W / 250W |    167MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 108...  Off  | 00000000:05:00.0 Off |                  N/A |
| 23%   27C    P8    16W / 250W |     10MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

在张量和模型等后面加上 .cuda()



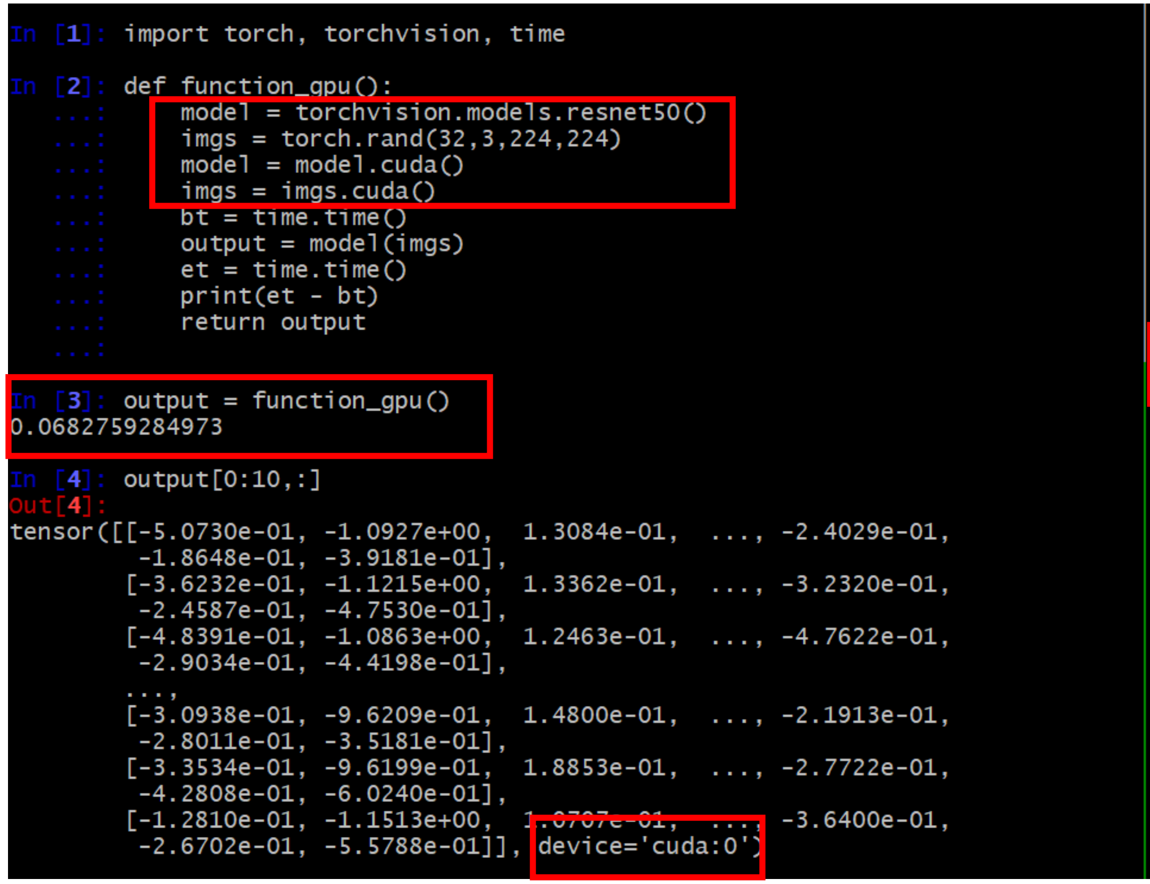
```
In [8]: a = a.cuda()

In [9]: a
Out[9]:
tensor([[ 0.9565,  0.3985,  0.9956,  0.4482,  0.6119],
        [ 0.4817,  0.7294,  0.1024,  0.5502,  0.3521],
        [ 0.1996,  0.5765,  0.7579,  0.0061,  0.3554],
        [ 0.9445,  0.7505,  0.4196,  0.1194,  0.2064],
        [ 0.9603,  0.7585,  0.3538,  0.5965,  0.4019]], device='cuda:0')

In [10]:
```



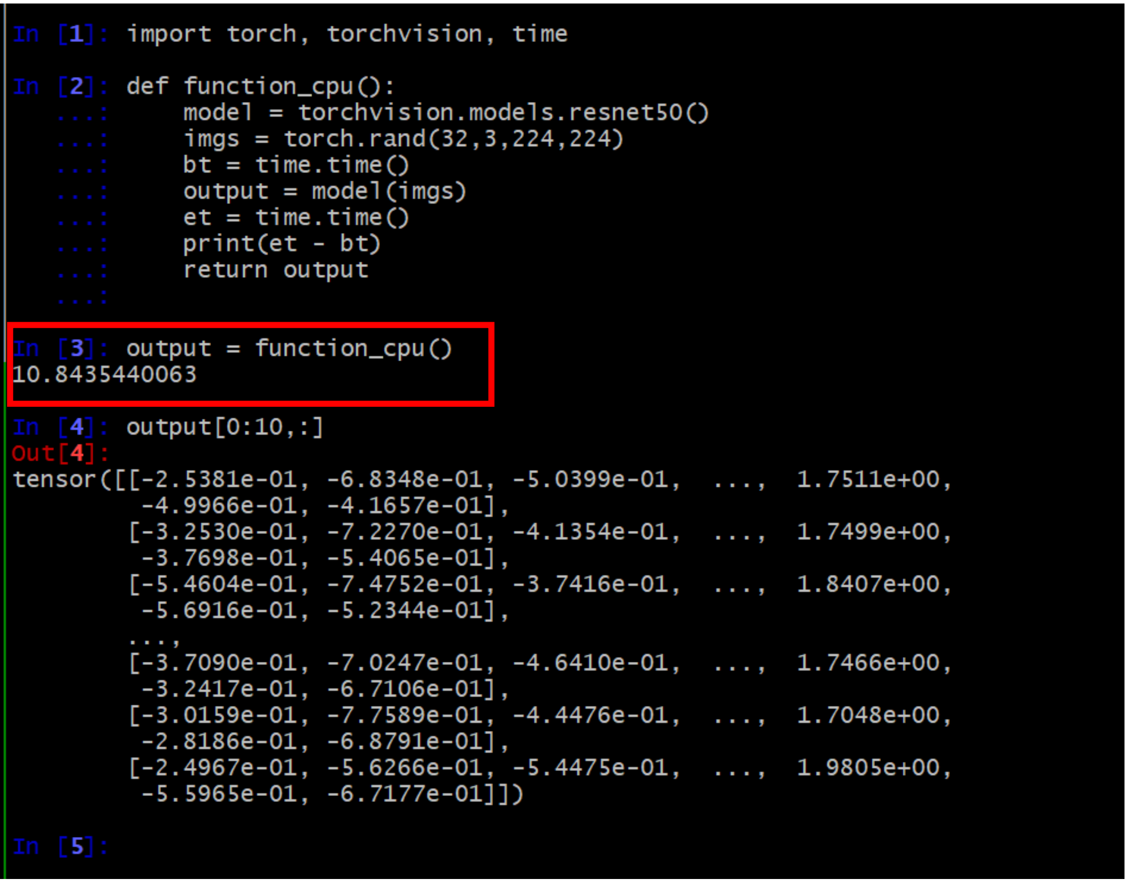
```
|===============================+======================+======================|
|   0  GeForce GTX 108...  Off  | 00000000:04:00.0 Off |                  N/A |
| 23%   29C    P8    16W / 250W |    678MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 108...  Off  | 00000000:05:00.0 Off |                  N/A |
| 23%   27C    P8    16W / 250W |     10MiB / 11178MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

Pytorch张量

CPU与GPU耗时对比



```
In [1]: import torch, torchvision, time

In [2]: def function_gpu():
   ...:     model = torchvision.models.resnet50()
   ...:     imgs = torch.rand(32,3,224,224)
   ...:     model = model.cuda()
   ...:     imgs = imgs.cuda()
   ...:     bt = time.time()
   ...:     output = model(imgs)
   ...:     et = time.time()
   ...:     print(et - bt)
   ...:     return output
   ...:

In [3]: output = function_gpu()
0.0682759284973

In [4]: output[0:10,:]
Out[4]:
tensor([[-5.0730e-01, -1.0927e+00,  1.3084e-01,  ..., -2.4029e-01,
         -1.8648e-01, -3.9181e-01],
        [-3.6232e-01, -1.1215e+00,  1.3362e-01,  ..., -3.2320e-01,
         -2.4587e-01, -4.7530e-01],
        [-4.8391e-01, -1.0863e+00,  1.2463e-01,  ..., -4.7622e-01,
         -2.9034e-01, -4.4198e-01],
        ...,
        [-3.0938e-01, -9.6209e-01,  1.4800e-01,  ..., -2.1913e-01,
         -2.8011e-01, -3.5181e-01],
        [-3.3534e-01, -9.6199e-01,  1.8853e-01,  ..., -2.7722e-01,
         -4.2808e-01, -6.0240e-01],
        [-1.2810e-01, -1.1513e+00,  1.0707e-01,  ..., -3.6400e-01,
         -2.6702e-01, -5.5788e-01]], device='cuda:0')
```



```
In [1]: import torch, torchvision, time

In [2]: def function_cpu():
   ...:     model = torchvision.models.resnet50()
   ...:     imgs = torch.rand(32,3,224,224)
   ...:     bt = time.time()
   ...:     output = model(imgs)
   ...:     et = time.time()
   ...:     print(et - bt)
   ...:     return output
   ...:

In [3]: output = function_cpu()
10.8435440063

In [4]: output[0:10,:]
Out[4]:
tensor([[-2.5381e-01, -6.8348e-01, -5.0399e-01,  ...,  1.7511e+00,
         -4.9966e-01, -4.1657e-01],
        [-3.2530e-01, -7.2270e-01, -4.1354e-01,  ...,  1.7499e+00,
         -3.7698e-01, -5.4065e-01],
        [-5.4604e-01, -7.4752e-01, -3.7416e-01,  ...,  1.8407e+00,
         -5.6916e-01, -5.2344e-01],
        ...,
        [-3.7090e-01, -7.0247e-01, -4.6410e-01,  ...,  1.7466e+00,
         -3.2417e-01, -6.7106e-01],
        [-3.0159e-01, -7.7589e-01, -4.4476e-01,  ...,  1.7048e+00,
         -2.8186e-01, -6.8791e-01],
        [-2.4967e-01, -5.6266e-01, -5.4475e-01,  ...,  1.9805e+00,
         -5.5965e-01, -6.7177e-01]])

In [5]:
```

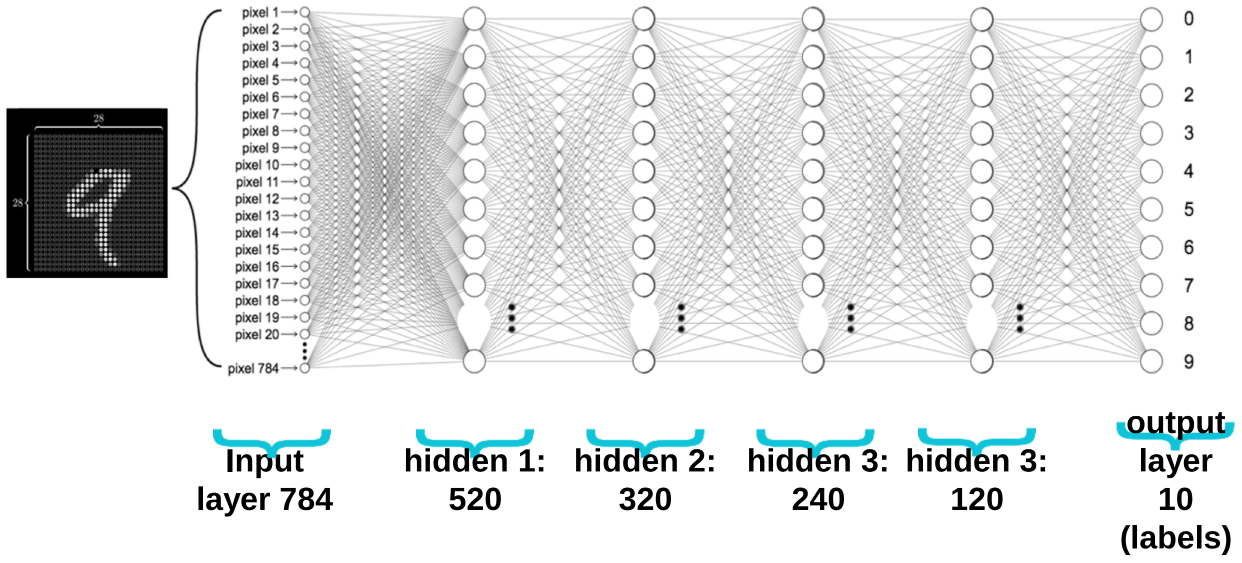
GPU中的运算结果也存放在GPU之中，如果要取回CPU需要加上.cpu()后缀

AI300学院

Pytorch工程组成

模型（网络）

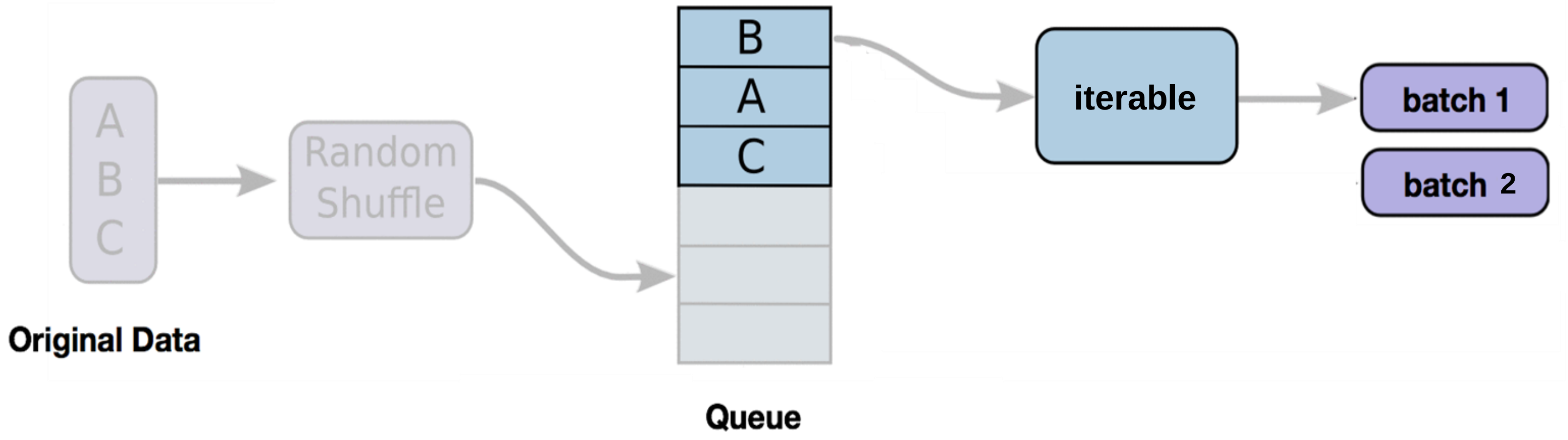
```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.l1 = nn.Linear(784, 520)
        self.l2 = nn.Linear(520, 320)
        self.l3 = nn.Linear(320, 240)
        self.l4 = nn.Linear(240, 120)
        self.l5 = nn.Linear(120, 10)

    def forward(self, x):
       # Flatten the data (n, 1, 28, 28)-> (n, 784)
        x = x.view(-1, 784)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = F.relu(self.l3(x))
        x = F.relu(self.l4(x))
        x = F.relu(self.l5(x))
        return F.log_softmax(x)
```



Pytorch模型分为网络初始化和前向传播两部分，反向传播会自动推导

Pytorch工程组成

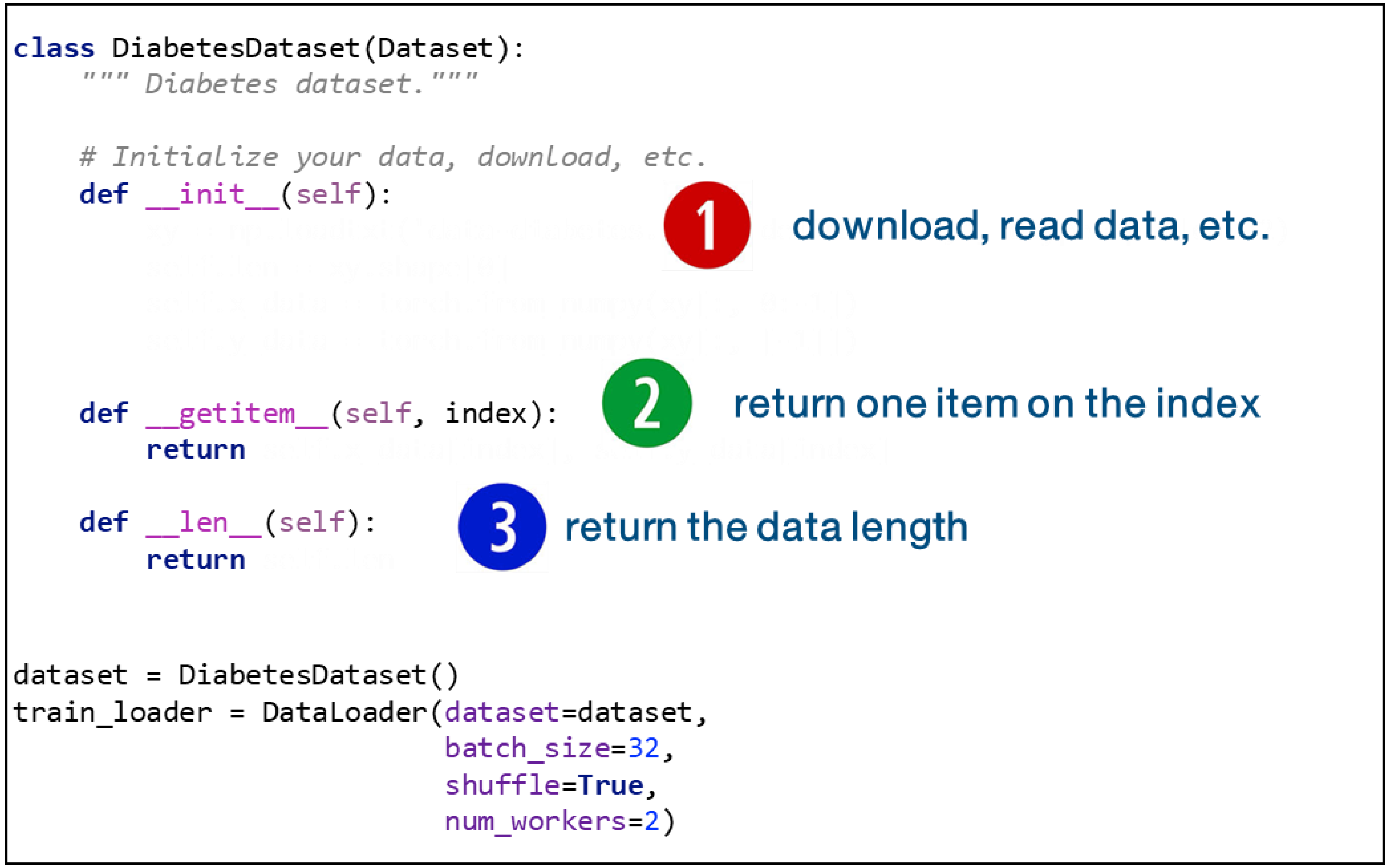
Dataloader（吐数据）



```
for i, data in enumerate(train_loader, 0):
        # get the inputs
        inputs, labels = data

        # wrap them in Variable
        inputs, labels = Variable(inputs), Variable(labels)

        # Run your training process
        print(epoch, i, "inputs", inputs.data, "labels", labels.data)
```

Pytorch工程组成

Custom Dataloader



```python
class DiabetesDataset(Dataset):
    """ Diabetes dataset."""

    # Initialize your data, download, etc.
    def __init__(self):




    def __getitem__(self, index):
        return

    def __len__(self):
        return


dataset = DiabetesDataset()
train_loader = DataLoader(dataset=dataset,
                          batch_size=32,
                          shuffle=True,
                          num_workers=2)
```

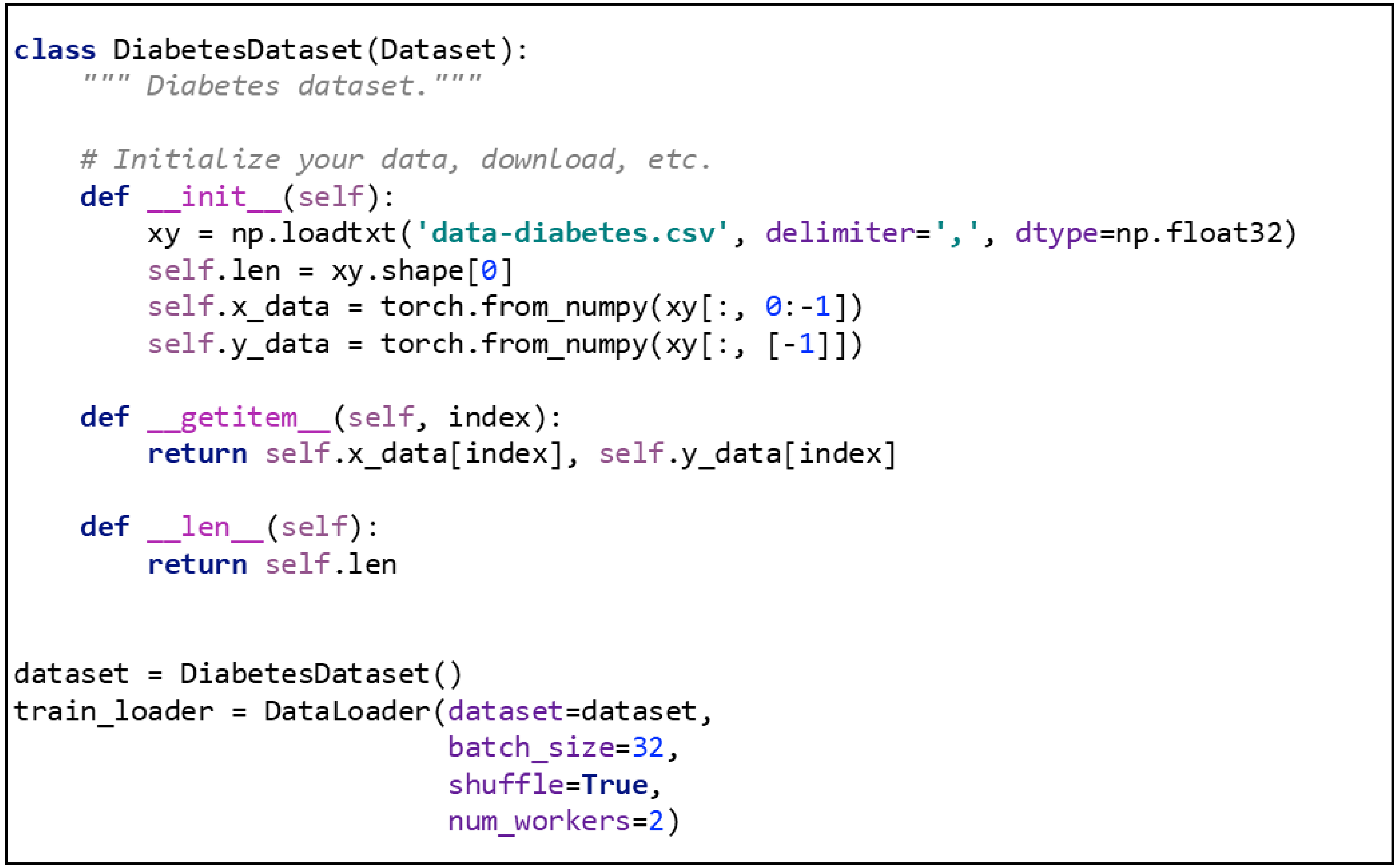
1 download, read data, etc.

2 return one item on the index

3 return the data length

Pytorch工程组成

Custom Dataloader



```python
class DiabetesDataset(Dataset):
    """ Diabetes dataset."""

    # Initialize your data, download, etc.
    def __init__(self):
        xy = np.loadtxt('data-diabetes.csv', delimiter=',', dtype=np.float32)
        self.len = xy.shape[0]
        self.x_data = torch.from_numpy(xy[:, 0:-1])
        self.y_data = torch.from_numpy(xy[:, [-1]])

    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    def __len__(self):
        return self.len


dataset = DiabetesDataset()
train_loader = DataLoader(dataset=dataset,
                          batch_size=32,
                          shuffle=True,
                          num_workers=2)
```

AI300学院

Pytorch工程组成

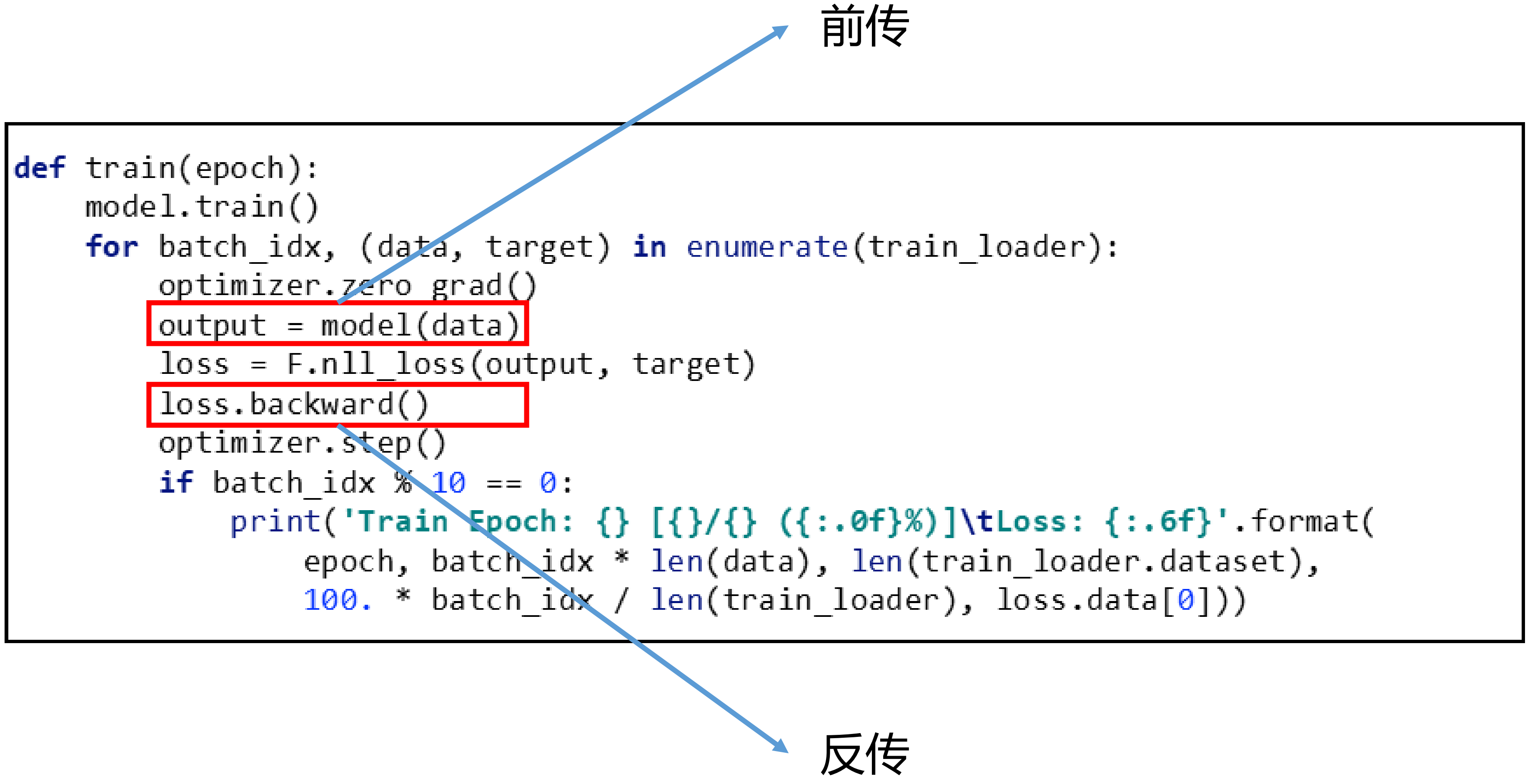
优化器&损失函数

- torch.optim.Adadelta (Python class, in torch.optim)
- torch.optim.Adadelta (Python class, in torch.optim)
- torch.optim.Adadelta.step (Python method, in torch.optim)
- torch.optim.Adadelta.step (Python method, in torch.optim)
- torch.optim.Adagrad (Python class, in torch.optim)
- torch.optim.Adagrad (Python class, in torch.optim)
- torch.optim.Adagrad.step (Python method, in torch.optim)
- torch.optim.Adagrad.step (Python method, in torch.optim)
- torch.optim.Adam (Python class, in torch.optim)
- torch.optim.Adam (Python class, in torch.optim)
- torch.optim.Adam.step (Python method, in torch.optim)
- torch.optim.Adam.step (Python method, in torch.optim)
- torch.optim.Adamax (Python class, in torch.optim)
- torch.optim.Adamax (Python class, in torch.optim)
- torch.optim.Adamax.step (Python method, in torch.optim)
- torch.optim.Adamax.step (Python method, in torch.optim)
- torch.optim.ASGD (Python class, in torch.optim)
- torch.optim.ASGD (Python class, in torch.optim)
- torch.optim.ASGD.step (Python method, in torch.optim)
- torch.optim.ASGD.step (Python method, in torch.optim)

- torch.nn.AdaptiveLogSoftmaxWithLoss (Python class, in torch.nn)
- torch.nn.AdaptiveLogSoftmaxWithLoss.log_prob (Python method, in torch.nn)
- torch.nn.AdaptiveLogSoftmaxWithLoss.predict (Python method, in torch.nn)
- torch.nn.BCELoss (Python class, in torch.nn)
- torch.nn.BCEWithLogitsLoss (Python class, in torch.nn)
- torch.nn.CosineEmbeddingLoss (Python class, in torch.nn)
- torch.nn.CrossEntropyLoss (Python class, in torch.nn)
- torch.nn.CTCLoss (Python class, in torch.nn)
- torch.nn.HingeEmbeddingLoss (Python class, in torch.nn)
- torch.nn.KLDivLoss (Python class, in torch.nn)
- torch.nn.L1Loss (Python class, in torch.nn)
- torch.nn.MarginRankingLoss (Python class, in torch.nn)
- torch.nn.MSELoss (Python class, in torch.nn)
- torch.nn.MultiLabelMarginLoss (Python class, in torch.nn)
- torch.nn.MultiLabelSoftMarginLoss (Python class, in torch.nn)
- torch.nn.MultiMarginLoss (Python class, in torch.nn)
- torch.nn.NLLLoss (Python class, in torch.nn)
- torch.nn.PoissonNLLLoss (Python class, in torch.nn)
- torch.nn.SmoothL1Loss (Python class, in torch.nn)
- torch.nn.SoftMarginLoss (Python class, in torch.nn)
- torch.nn.TripletMarginLoss (Python class, in torch.nn)

Pytorch工程组成

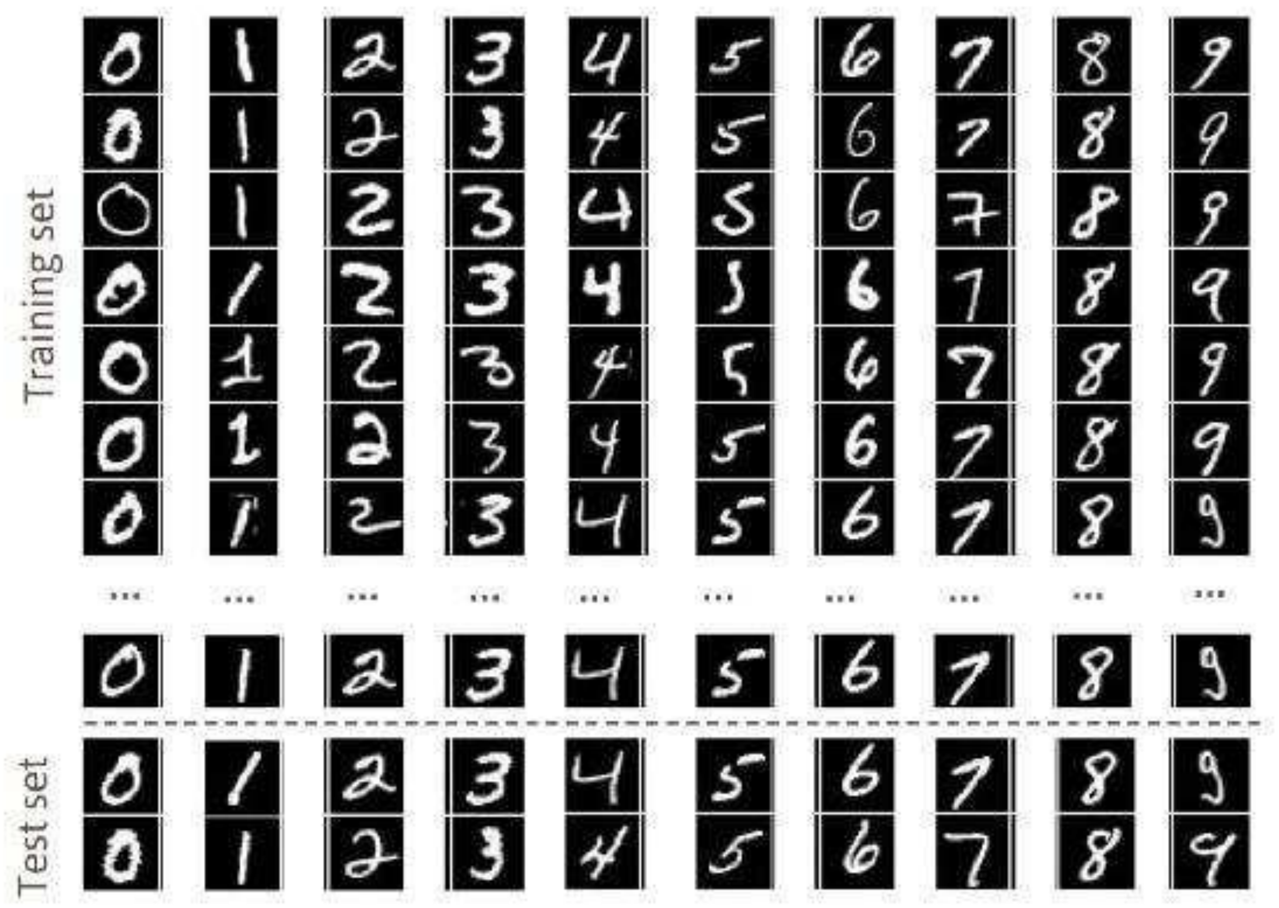
训练过程（前传与反传）



```
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]))
```

Pytorch手写体分类

Mnist手写体分类



Pytorch实现Mnist手写体识别

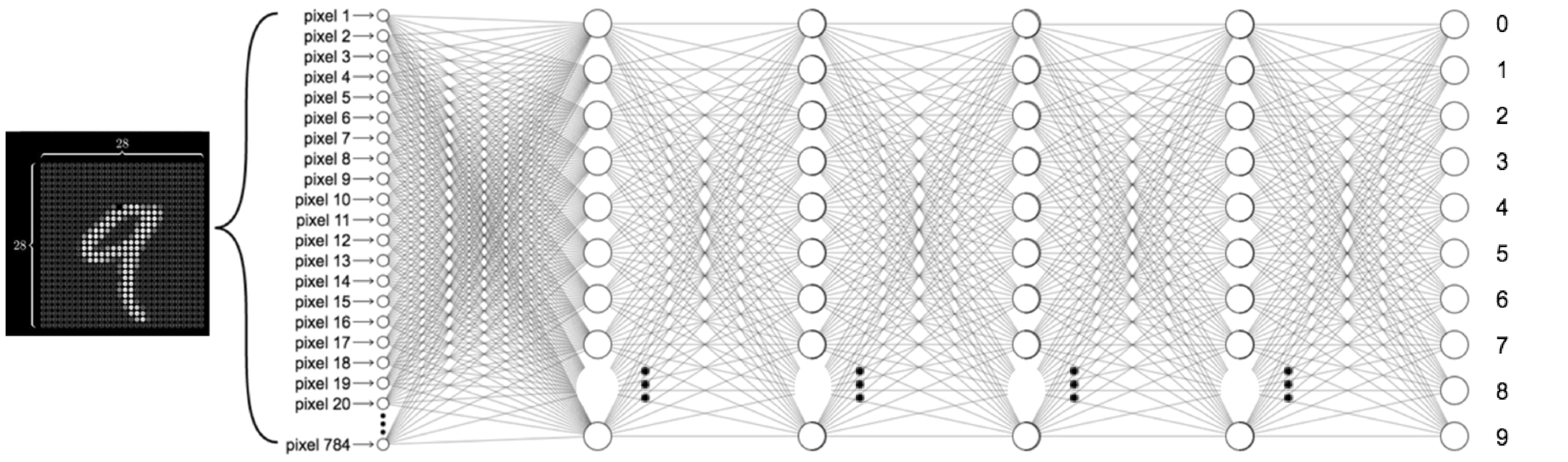
单通道28×28的图像

采用四层全连接网络

采用分类的交叉熵损失

Pytorch手写体分类

网络结构



Input layer 784 hidden 1: 520 hidden 2: 320 hidden 3: 240 hidden 3: 120 output layer 10 (labels)

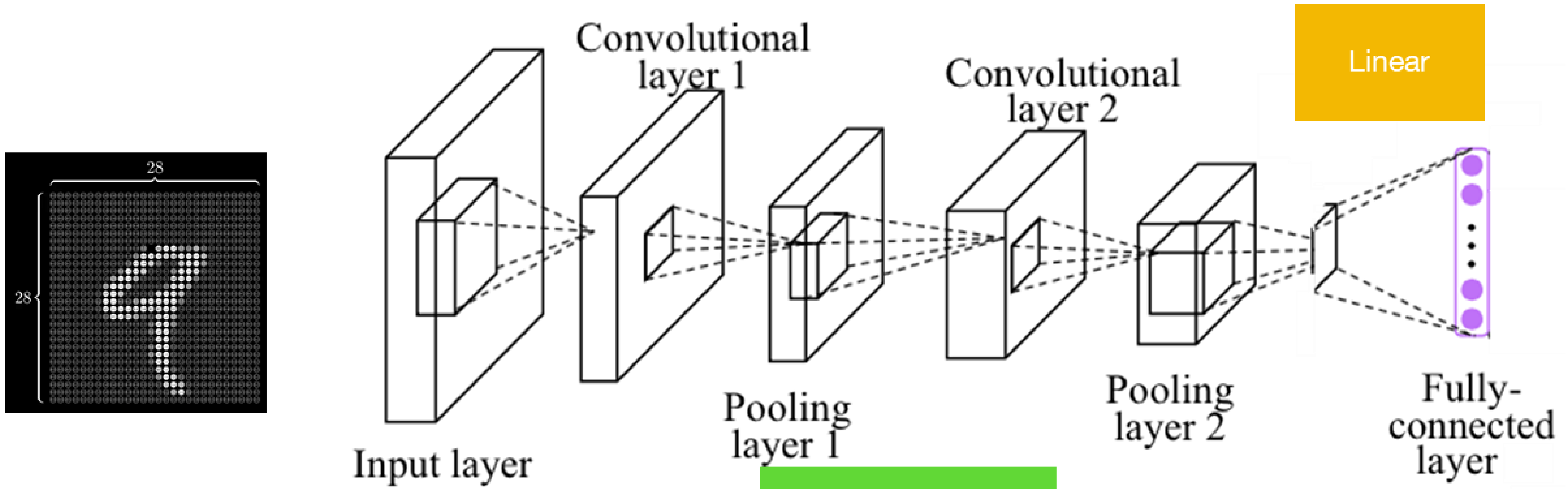
全连接神经网络

Pytorch手写体分类

代码解读

课后思考

将NN改为CNN



torch.nn.Conv2d
(in_channels,
out_channels,
kernel_size)

nn.MaxPool2d
(kernel_size)

欢迎关注AI300学院