#PyTorch：

import torch

tensor类：包括FloatTensor,DoubleTensor,IntTensor等。和numpy类似，可以用np.array创建，还可以使用zeros,ones,eye,randn等创建.

torch.range(a,b[,c])得到一个tensor

基本运算：可以使用torch.abs(t),torch.add(a,b)等对tensor进行计算。

add+,sub-,mul\*,div/,pow^,**mm矩阵之间的乘法**,**mv矩阵和向量的矩阵乘法,dot点乘**

tensor转换为np.array:

浅拷贝(共享内存):<tensor>.**numpy()**

深拷贝(独立副本):<tensor>.**clone().detach().numpy()**

torch.**clamp**(<tensor>,lower,upper)将所有元素框定边界[lower,upper],越界的重写成边界值

自动求导：

一元函数：

x=torch.tensor(3.0,**requires\_grad=True**) #开启梯度跟踪

y=x\*\*2

y.**backward()** #求导

print(x**.grad**) #求导的结果放在x的.grad属性中

多元函数：对每个自变量求偏导的结果放在各个自变量的.grad属性中.

注：多次调用backward()时梯度会累加，故需要清零 x**.grad.zero\_()**

禁用梯度跟踪：

1.y=x**.detach()**\*\*2

2.**with torch.no\_grad():**

y=x\*\*2

#torch.nn

from torch import nn

实现某个神经网络，只需继承nn.Module，在初始化中定义模型结构与参数，在函数forward()中编写网络前向过程即可。

**import torch**

**import torch.nn as nn**

**import torch.nn.functional as F**

**class Net(nn.Module):**

**def \_\_init\_\_(self):**

**super(Net, self).\_\_init\_\_()**

**# 输入图像channel：1，输出channel：6，5x5卷积核**

**self.conv1 = nn.Conv2d(1, 6, 5)**

**self.conv2 = nn.Conv2d(6, 16, 5)**

**# 全连接层**

**self.fc1 = nn.Linear(16 \* 5 \* 5, 120)**

**self.fc2 = nn.Linear(120, 84)**

**self.fc3 = nn.Linear(84, 10)**

**def forward(self, x):**

**# 使用2x2窗口进行最大池化**

**x = F.max\_pool2d(F.relu(self.conv1(x)), (2, 2))**

**# 如果窗口是方的，只需要指定一个维度**

**x = F.max\_pool2d(F.relu(self.conv2(x)), 2)**

**x = x.view(-1, self.num\_flat\_features(x))**

**x = F.relu(self.fc1(x))**

**x = F.relu(self.fc2(x))**

**x = self.fc3(x)**

**return x**

**def num\_flat\_features(self, x):**

**size = x.size()[1:]  # 获取除了batch维度之外的其他维度**

**num\_features = 1**

**for s in size:**

**num\_features \*= s**

**return num\_features**

**net = Net()**

**print(net)**