
PyTorch

什么是 PyTorch?

- PyTorch 是一个开源的 python 机器学习库，基于 Torch，主要针对两类人群：
  - 作为 NumPy 的替代品，可以利用 GPU 的性能进行计算
  - 作为一个高灵活性、速度快的深度学习平台

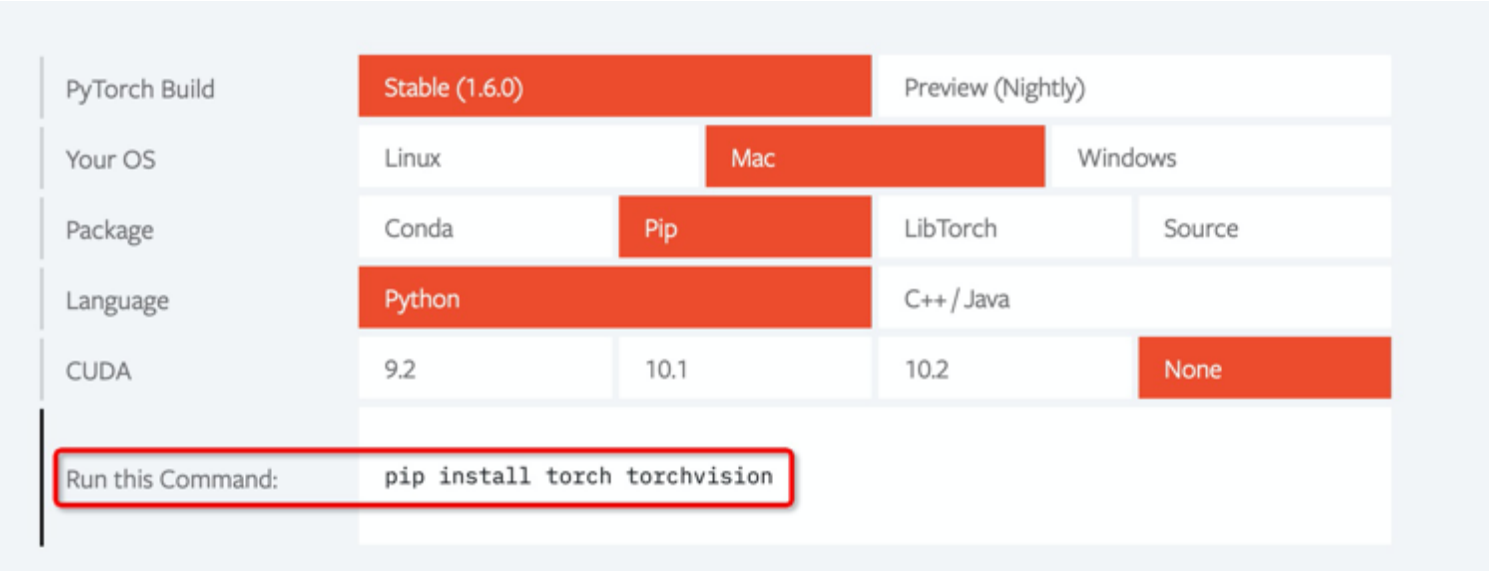
- 特点：
  - 上手简单
  - 调试方便
  - 文档友好

一些特性

- 动态图
- 多 GPU、分布式训练
- JIT、TorchScript（C++ 部署）
- →ONNX（通用）→Caffe2（移动端部署）

安装

- 准备：一台电脑/服务器/Docker、一张高性能 NVIDIA 显卡(可选)
- 官网：https://pytorch.org



推荐先安装 anaconda（https://www.anaconda.com/products/individual），然后通过 pip 安装
安装过慢可以使用其它 pip 源，以阿里源为例：
pip install torch torchvision -i http://mirrors.aliyun.com/pypi/simple/ --trusted-host mirrors.aliyun.com
验证安装成功：进入 python 环境，执行 import torch 不报错

张量

- Tensor(张量）类似于NumPy的ndarray，但还可以在GPU上使用来加速计算。

```
import torch
```

构造一个填满 `0` 且数据类型为 `long` 的矩阵:

```
x = torch.zeros(5, 3, dtype=torch.long)
print(x)
```

输出:

```
tensor([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])
```

获取张量的形状：

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

输出：

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

直接从数据构造张量：

```
x = torch.tensor([5.5, 3])
print(x)
```

输出：

```
tensor([5.5000, 3.0000])
```

运算

- 超过100种tensor的运算操作，包括转置，索引，切片，数学运算，线性代数，随机数等

加法：形式一

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

加法：形式二

```
print(torch.add(x, y))
```

...

numpy → tensor :

```
import numpy as np
a = np.ones(5)
b = torch.from_numpy(a)
```

tensor → numpy :

```
a = torch.ones(5)
b = a.numpy()
```

CPU上的所有张量(CharTensor除外)都支持与Numpy的相互转换

CUDA 张量

张量可以使用 `.to` 方法移动到任何设备(device) 上:

```
# 当GPU可用时,我们可以运行以下代码
# 我们将使用`torch.device`来将tensor移入和移出GPU
if torch.cuda.is_available():
    device = torch.device("cuda")          # a CUDA device object
    y = torch.ones_like(x, device=device)  # 直接在GPU上创建tensor
    x = x.to(device)                       # 或者使用`.to("cuda")`方法
    z = x + y
    print(z)
    print(z.to("cpu", torch.double))       # `.to`也能在移动时改变dtype
```

输出:

```
tensor([1.0445], device='cuda:0')
tensor([1.0445], dtype=torch.float64)
```

Autograd：自动求导-张量

```
import torch
```

创建一个张量并设置 `requires_grad=True` 用来追踪其计算历史

```
x = torch.ones(2, 2, requires_grad=True)
print(x)
```

输出：

```
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
```

对这个张量做一次运算：

```
y = x + 2
print(y)
```

输出：

```
tensor([[3., 3.],
        [3., 3.]], grad_fn=<AddBackward0>)
```

对y进行更多操作

```
z = y * y * 3
out = z.mean()

print(z, out)
```

输出：

```
tensor([[27., 27.],
        [27., 27.]], grad_fn=<MulBackward0>) tensor(27., grad_fn=<MeanBackward0>)
```

Autograd：自动求导-梯度

现在开始进行反向传播，因为 `out` 是一个标量，因此 `out.backward()` 和 `out.backward(torch.tensor(1.))` 等价。

```
out.backward()
```

输出导数 `d(out)/dx`

```
print(x.grad)
```

输出：

```
tensor([[4.5000, 4.5000],
        [4.5000, 4.5000]])
```

验证：

```
x = torch.ones(2, 2, requires_grad=True)
y = x + 2
z = y * y * 3
out = z.mean()
```

$$out = \frac{\sum_{i=1}^{4} z_i}{4} \qquad z_i = y_i * y_i * 3 \qquad y_i = x_i + 2$$

$$\frac{d(out)}{d(z_i)} = \frac{1}{4} \qquad \frac{d(z_i)}{d(y_i)} = 6y_i = 18 \qquad \frac{d(y_i)}{d(x_i)} = 1$$

$$\frac{d(out)}{d(x_i)} = \frac{d(out)}{d(z_i)} * \frac{d(z_i)}{d(y_i)} * \frac{d(y_i)}{d(x_i)} = 4.5$$

神经网络

- torch.nn.Module
  - 所有网络的基类
  - 编写新的网络
    - 继承 torch.nn.Module
    - 编写 初始化函数
    - 编写 前向传递

torch.nn

- Containers
- Convolution Layers
- Pooling layers
- Padding Layers
- Non-linear Activations (weighted sum, nonlinearity)
- ...

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

## 搭建网络
class Net(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)
    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        return x

## 实例网络对象
net = Net()
## 前向传播
output = net(input)
```

损失函数和优化器

- torch.nn 中有很多不同的损失函数
- torch.optim 中有各种优化器，可以让神经网络按不同的方式更新权重，如SGD、Nesterov-SGD、Adam、RMSProp等

```
## 定义损失函数
criterion = nn.MSELoss()

## 定义优化器
optimizer = torch.optim.SGD(net.parameters(), args.lr, m
omentum=args.momentum, weight_decay=args.weight_decay)

# 在训练的迭代中:
optimizer.zero_grad()   # 清零梯度缓存
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # 更新参数
```

数据集和加载器

- torch 提供 torch.utils.data.DataLoader 用来加载训练数据
- torch 对于视觉方面还特别创建了一个 torchvision 包，其中包含了针对 Imagenet、CIFAR10、MNIST 等常用数据集，即 torchvision.datasets，以及一些图像变换方法
- 自定义数据集通常通过 datasets.ImageFolder() 方法引入

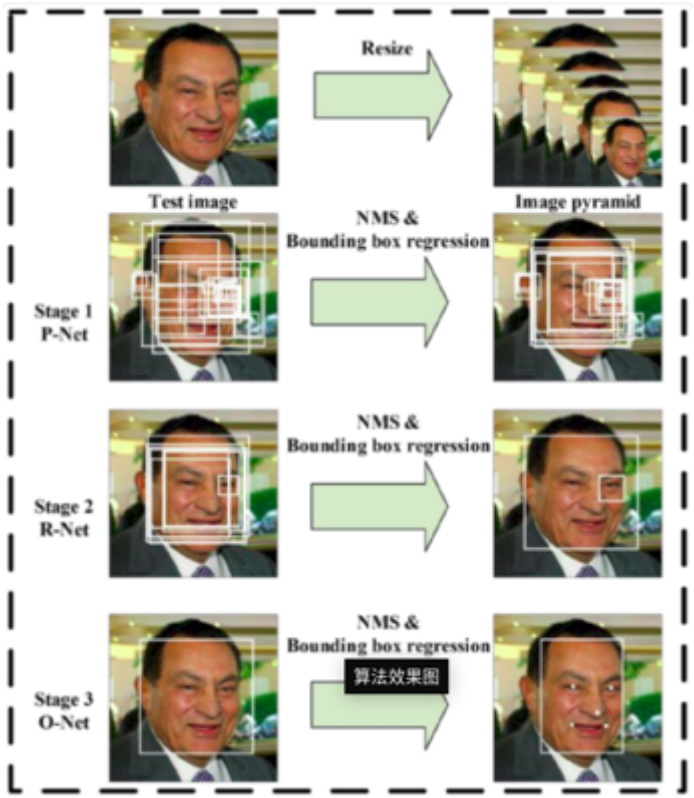
```
# 定义图像转换方法
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
# 准备数据
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4, shuffle=True, num_workers=2)
# 随机获取训练图片
dataiter = iter(trainloader)
inputs, targets = dataiter.next()
```

训练网络

- optimizer.step() 更新权重
- 训练过程：
  - 搭建网络
  - 准备训练数据
  - 网络参数绑定优化器
  - 循环：
    1. 输入数据前向传播
    2. 计算 loss
    3. 反向传播计算梯度
    4. 更新网络权重

```
## 开始训练
for epoch in range(args.start_epoch, args.epochs):
    ## 训练模式
    net.train()
    for i, (input, target) in enumerate(train_loader):
        ## 前向传递，计算 Loss
        out = net(input)
        loss = criterion(out, target)
        ## 反向传播，更新参数
        optimizer.zero_grad()
        loss.backward()
        optimizer.steop()
```

人脸识别



512 维向量

训练：全连接层
应用：相似度量

人脸检测

https://github.com/TreB1eN/InsightFace_Pytorch/tree/master/mtcnn_pytorch

```
from src import detect_faces
from PIL import Image

image = Image.open('image.jpg')
bounding_boxes, landmarks = detect_faces(image)
```

人脸对齐

https://github.com/TreB1eN/InsightFace_Pytorch/blob/master/mtcnn_pytorch/get_aligned_face_from_mtcnn.ipynb

```
src = np.array([
 [30.2946, 51.6963],
 [65.5318, 51.5014],
 [48.0252, 71.7366],
 [33.5493, 92.3655],
 [62.7299, 92.2041] ], dtype=np.float32 )
```

```
bounding_boxes, landmarks = detect_faces(img)
```

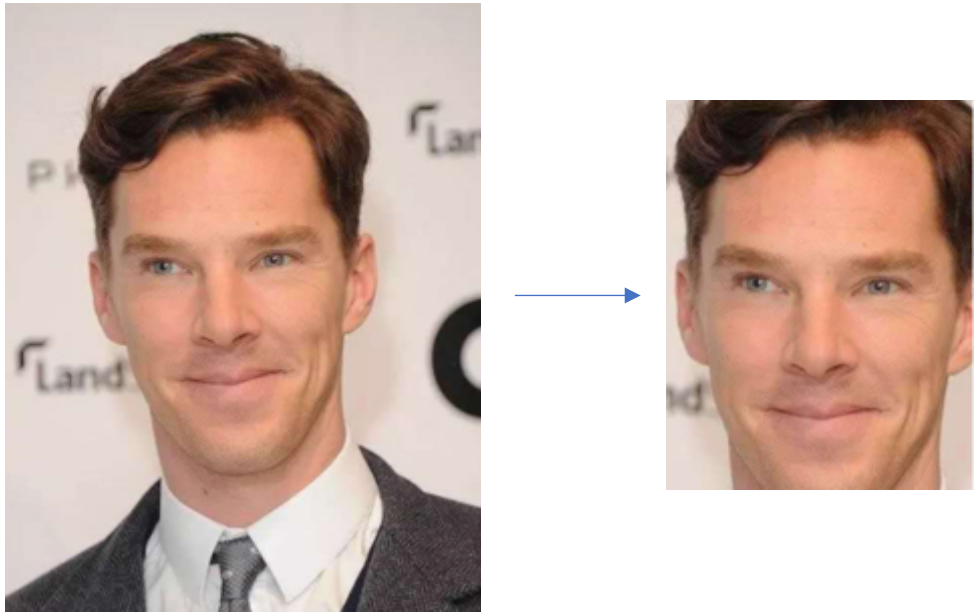
```
dst = landmarks[0].astype(np.float32)

facial5points = [[dst[j],dst[j+5]] for j in range(5)]

from skimage import transform as trans
tform = trans.SimilarityTransform()


tform.estimate(np.array(facial5points), src)

M = tform.params[0:2,:]
```



```
warped = cv2.warpAffine(img_cv2,M,(img.size[0],img.size[1]), borderValue = 0.0)
```

用 PyTorch 训练识别器

- 任务：正确分类 10M 人脸图片，包含 100K 人
- 步骤：
  - 1. 准备数据集
  - 2. 搭建网络
  - 3. 编写训练、验证代码

数据集

- torchvision.datasets
  - MNIST
  - Fashion-MNIST
  - KMNIST
  - EMNIST
  - QMNIST
  - FakeData
  - COCO
  - LSUN
  - ImageFolder
  - DatasetFolder
  - ImageNet
  - CIFAR
  - STL10
  - SVHN
  - PhotoTour
  - SBU
  - Flickr
  - VOC
  - Cityscapes
  - SBD
  - USPS
  - Kinetics-400
  - HMDB51
  - UCF101
  - CelebA

```
root/dog/xxx.png
root/dog/xxy.png
root/dog/xxz.png

root/cat/123.png
root/cat/nsdf3.png
root/cat/asd932_.png
```

- MS-Celeb1M
  - ID1
  - ID2
    - 1.jpg
    - 2.jpg
    - 3.jpg

数据集-ImageFolder

```
import torch
import torchvision.datasets as datasets
import torchvision.transforms as transforms

traindir = '/ssd/Dataset/MS-Celeb-1M'
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406])
train_dataset = datasets.ImageFolder(
    traindir,
    transforms.Compose([
        transforms.Resize(224),
        transforms.CenterCrop(224),
        transforms.RandomHorizontalFilp(),
        transforms.ToTensor(),
        normalize,
    ])
)
```

搭建网络-ResNet

- torchvision.models
  - VGG
  - ResNet
  - AlexNet

```
## 载入现有网络
net = torchvision.models.resnet34(pretrained=False)
## 重写最后一层 FC
net.fc = torch.nn.Linear(512, 100000, bias=False)

## Loss 公式 以及 优化器
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), args.lr,
                            momentum=args.momentum,
                            weight_decay=args.weight_decay)
```

编写训练代码

```
## 数据读取器
train_loader = torch.utils.data.DataLoader(
        train_dataset, batch_size=args.batch_size ,
        shuffle=(train_sampler is None), num_workers=args.workers,
        pin_memory=True, sampler=train_sampler)
## 开始训练
for epoch in range(args.start_epoch, args.epochs):
    ## 训练模式
    net.train()
    for i, (input, target) in enumerate(train_loader):
        ## 前向传递，计算 Loss
        out = net(input)
        loss = criterion(out, target)
        ## 反向传播，更新参数
        optimizer.zero_grad()
        loss.backward()
        optimizer.steop()
```

资源

- 官方文档：https://pytorch.org/docs/stable/index.html
- 官方论坛：https://discuss.pytorch.org/
  - Bug
  - Method
- GitHub：https://github.com/pytorch/
  - Code
  - Example
- 人脸识别：
  - InsightFace_Pytorch：https://github.com/TreB1eN/InsightFace_Pytorch