

torch 笔记

徐世桐

1 import

```
from torch import nn, optim
import torch.nn.functional as F
from torchvision import models, transforms
from PIL import Image
```

2 tensor 使用 GPU

```
if torch.cuda.is_available():
    dev = "cuda:0"
else:
    dev = "cpu"
device = torch.device(dev)
```

得到 device 数据类型，定义使用 CPU 或使用哪一 GPU

`Tensor.get_device()`

查看张量存在的 CPU/GPU

3 tensor 数据类型

`torch.arange`

`torch.arange(a)` 得到 `Tensor[0, 1, ..., [a]]`

`torch.arange(a, b)` 得到 `Tensor[a, a + 1, ..., a + n]`, n 为整数且 $a + n < b$

`torch.arange(a, b, c)` 得到 `Tensor[a, a + c, ..., a + nc]`, n 为整数且 $a + nc < b$

`torch.arange(..., requires_grad=True)` 分配空间记录斜率，同 mxnet 的 `attach_grad()`

`torch.arange(..., device=device 数据类型)` 将张量分配进指定 CPU/GPU

`torch.tensor([], REQUIRES_GRAD, DEVICE)`

通过 python 数组创建 Tensor

`REQUIRES_GRAD = True` 分配空间记录斜率

`DEVICE = device` 数据类型将张量分配进指定 CPU/GPU

`torch.numpy()` 得到 `NDArray`

`torch.tolist()` 得到 python 数组

`torch.from_numpy(NDArray)` 从 `NDArray` 创建 Tensor

`torch.mm(Tensor, Tensor)` 矩阵乘法

`+-*/` 同 `NDArray` 使用广播机制

`Tensor.reshape()` 改变形状, 新形状元素数必须等于输入元素数

`Tensor.permute(SHAPE)` 等同 `mxnet` 的 `transpose`, 更改纬度顺序

`Tensor.transpose(dim0, dim1)` 交换两个纬度

`Tensor.to(device)` 将张量分配进指定 CPU/GPU

`torch.random(MEAN, STD, SIZE*)`

`size=(x_1, x_2, \dots)` 限定输出张量形状

`mean=Tensor, std=Tensor/const` 当没有限定 `size` 时 `mean` 必为 `float Tensor`, 形状和输出形状相同。

`mean=Tensor/const, std=Tensor/const` 当限定 `size` 后 `mean, std` 可为 `const` 或单个值的 `Tensor`
`torch.rand(SIZE*)`

得到 `SIZE` 形状的随机数张量, 每一元素 $\in [0, 1)$ 。 `SIZE` 无定义则得到 `const` 随机数

代替 `torch.uniform` 功能

`dataset = torch.utils.data.TensorDataset(样本 Tensor, 标签 Tensor)`

`dataiter = torch.utils.data.DataLoader(dataset, batch_size= 批量大小, shuffle=True)`

使用 `torch` 进行批量迭代

`dataiter` 输出的 `feature, label` 使用的 CPU/GPU 和样本 `Tensor`, 标签 `Tensor` 使用的 CPU/GPU 分别对应

`torch.save(Tensor, ' 文件名')` 文件中保存一张量

`Tensor = torch.load(' 文件名')` 读取文件中张量

`torch.cuda.synchronize()` 等待异步计算结束, 打印结果同样等待异步计算

4 torch 神经网络

`net = nn.Sequential()`

神经网络定义: `net.add_module(' 层名', 层)`

层定义:

`nn.Linear(输入节点数, 输出节点数)` 定义全连接层

可使用层直接进行前向计算, 训练函数中使用 `[layer.weight, net.bias]` 传入参数

前向计算为 ($|B|$, 特征数) 和 权重 矩阵相乘

使用 GPU 时层定义后需加 `.to(device)`, 并不可以使用 `device=` 赋 GPU

`net.weight/bias.data.fill_(值)` 对层中所有权重/偏差赋值

`nn.init.xavier_uniform(net.weight/.bias)` 对层中所有权重/偏差使用 `xavier` 初始化

`loss`

`= nn.MSELoss(REDUCTION*)` 平方代价函数

`REDUCTION = 'none' | 'mean' | 'sum'` 得到每一样本代价值向量 | 得到平均代价 | 得到代价和。默认为 'mean'

`= nn.CrossEntropyLoss()` 交叉熵损失函数, 已经包含 `softmax` 计算

`trainer`

```

    = optim.SGD(net.parameters(), lr= 学习率) SGD 迭代函数
    = optim.Adam(net.parameters(), lr= 学习率) Adam-SGD 迭代
    trainer.step() 进行迭代
    每一迭代中 trainer.grad_zero() 清零斜率, 否则训练斜率为随机值, 代价值在某一高值波动
net.parameters() 得到权重
    list(net.parameters()) 得到 param 类型数组, 包含 [第一层权重, 第一层偏差, ..., 最后一
层参数]
    param 类型数组.data 得到参数张量
    param 类型数组.name 得到所属层名, 可为空
loss(y_hat, y).backward() 得到代价函数值, 求导
    不会调用.sum() 或.mean(), 求和方法在 loss 函数中定义
class out_image(nn.Module):
    def __init__(self):
        super().__init__()
    def forward(self, x):
        自定义神经网络
net = models.NET_NAME(pretrained=True) 得到预训练的神经网络

```

5 读取图像

```

image = Image.open(' 图像路径')
    得到图片, 显示图片直接调取 image.show(), 显示结果不阻断 python 程序。
transform = transforms.Compose([trans1, trnas2, ...])
    合并多个对图像的变换
transforms.Resize(图片形状) 缩放图片
transforms.ToTensor() 图片变为张量
transforms.Normalize(MEAN, STD) 对图片的张量输入, 求标准化。MEAN, STD 可为张量
    内部实现: 对 RGB 3 通道上的像素分别使用 (3, ) 形状的 MEAN, STD 值求标准化
显示图片
    image = transforms.ToPILImage()(image_tensor)
    image.show()
    或使用自定义包, 支持反标准化
import sys
sys.path.append('../machine_learning/')
from utils.functions import show_tensor_image, un_normalize_image
    show_tensor_image(un_normalize_image(image.reshape((3, height, width))), image_mean,
image_std))

```

6 常见错误

调用 `trainer.zero_grad()`

否则参数代价值高，并迭代后不下降

使用网络层作为权重，不参与斜率计算时 调用 `layer.requires_grad_(False)`

否则调用 `loss.backward()` 时提示需要 `retain_graph=True`

`transforms.ToPILImage()` 不保证像素值在 `[0,1]` 区间,需调用 `image_tensor.clamp(min=0, max=1)`

`d2lzh` 自动对图像做 `clip`，保证值在 `[0,1]` 区间

否则图像中包含突出像素点，如红 紫 蓝像素。

循环中更改 `Tensor` 值并将 `Tensor` 加入数组，使用 `Tensor.clone()` 复制斜率

否则下一迭代可能更改上一迭代已经加入数组的张量

前向计算中不能调用 `Tensor.detach()`

否则此项无法求斜率，无法进行迭代