Pytorch

加载数据初认识

- Dataset
  - 提供一种方式去获取数据及其label
  - 如何获取每一个数据及其label
  - 告诉我们总共有多少数据
  - __init__提供全局变量
  - __getitem__ 根据索引返回单个样本（数据和标签）
  - __len__返回数据集的样本总数
  - os.path.join() ： 路径拼接函数
  - os.listdir() 方法用于返回指定的文件夹包含的文件或文件夹的名字的列表

Tensorboard

- SummaryWriter
  - 条目直接写入 log_dir 中的事件文件以供 TensorBoard 使用
  - add_scalar():加一个标量数据（scalar data）到summary中
    - tag：要求是一个string，用以描述 该标量数据图的 标题
    - scalar_value ： 可以简单理解为一个y轴值的列表
    - global_step：可以简单理解为一个x轴值的列表，与y轴的值相对应
  - tensorboard --logdir=logs 可视化
  - add_image()：add_image(tag, img_tensor, global_step=None, walltime=None, dataformats='CHW')
    - tag (string): 数据名称
    - img_tensor (类型 torch.Tensor 或 numpy.array): 图像数据
    - global_step (int, optional): 记录这是第几个子图
    - walltime (float, optional): 记录发生的时间，默认为 time.time()
    - dataformats (string, optional): 图像数据的格式。默认为 'CHW'，即 Channel x Height x Width，还可以是 'CHW'、'HWC' 或 'HW' 等

Transforms

- 结构及用法
  - transform.py工具箱
    - totensor，resize等
    - 创建具体的工具
      - __call__()是一种magic method，在类中实现这一方法可以使该类的实例（对象）像函数一样被调用。
- 常用的Transforms
  - Totensor()
    - 将PIL Image或者 ndarray 转换为tensor，并且归一化至[0-1] 注意事项：归一化至[0-1]是直接除以255，若自己的ndarray数据尺度有变化，则需要自行修改。
  - Normalize()
    - Normalize()函数的作用是将数据转换为标准高斯分布，即逐个个channel的对图像进行标准化（均值变为0，标准差为1），可以加快模型的收敛。输入（channel,height,width）形式的tensor。
  - resize()将输入PIL图像的大小调整为给定大小。
    - size（sequence 或int）：所需的输出大小。如果size是类似（h，w）的序列，则输出大小将与此匹配。如果size是int，则图像的较小边缘将与此数字匹配。即，如果高度>宽度，则图像将重新缩放为（尺寸*高度/宽度，尺寸）
      - 如果 size 是整数：将图像的短边缩放到该值，长边按比例缩放（保持宽高比）。
      - 如果 size 是元组 (h, w)：直接将该图像调整到这个尺寸（可能破坏宽高比）。
    - interpolation（int，optional）：所需的插值。默认是 PIL.Image.BILINEAR
  - compose()
    - Compose 中的操作会按照定义的顺序依次执行
    - 每个变换操作的输入需要与前一个操作的输出兼容。例如：
    - Resize 和 CenterCrop 接受 PIL 图像作为输入。
    - ToTensor 将 PIL 图像转换为张量。
    - Normalize 作用于张量。
  - randomcrop
    - 从图像的任意位置（均匀随机）裁剪出指定尺寸的子图
- 使用方法总结
  - 关注输入和输出
  - 多看官方文档
  - 不知道返回值的时候可以用print()/print(type())

torchvison中的数据集使用

- https://pytorch.org/pytorch-domains

DataLoader的使用

- parameters
  - dataset (必需): 用于加载数据的数据集。通常是torch.utils.data.Dataset的子类实例。
  - batch_size (可选): 每个批次的数据样本数。默认值为1。
  - shuffle (可选): 是否在每个周期开始时打乱数据。默认为False。
  - num_workers (可选): 用于数据加载的子进程数量。默认为0，意味着数据将在主进程中加载。
  - drop_last (可选): 如果数据集大小不能被批次大小整除，是否丢弃最后一个不完整的批次。默认为False。

神经网络骨架-nn.Module的使用

我们在定义自己的网络的时候，需要继承nn.Module类，并重新实现构造函数__init__构造函数和forward这两个方法。但有一些注意技巧：

- （1）一般把网络中具有可学习参数的层（如全连接层、卷积层等）放在构造函数__init__()中，当然我也可以把不具有参数的层也放在里面；
- （2）一般把不具有可学习参数的层(如ReLU、dropout、BatchNormanation层)可放在构造函数中，也可不放在构造函数中，如果不放在构造函数__init__里面，则在forward方法里面可以使用nn.functional来代替
- （3）forward方法是必须要重写的，它是实现模型的功能，实现各个层之间的连接关系的核心。

卷积操作

- torch.nn.functional.conv2d
  - input – input tensor of shape 输入
  - weight – filters of shape 卷积核
  - stride – the stride of the convolving kernel.Can be a single number or a tuple (sH, sW) 步幅
  - padding –implicit paddings on both sides of the input. Can be a string {'valid', 'same'}, single number or a tuple (padH, padW). 填充
- torch.reshape

卷积层

- torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)[source]
  - in_channels (int) – Number of channels in the input image
  - out_channels (int) – Number of channels produced by the convolution
  - kernel_size (int or tuple) – Size of the convolving kernel （只需要定义大小，不需要设定具体数字）
  - stride (int or tuple, optional) – Stride of the convolution. Default: 1
  - padding (int, tuple or str, optional) – Padding added to all four sides of the input. Default: 0
- 作用
  - 卷积层的作用是提取输入图片中的信息，这些信息被称为图像特征，这些特征是由图像中的每个像素通过组合或者独立的方式所体现，比如图片的纹理特征，颜色特征。

池化层

- torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)
  - 当ceil_mode = true时，将保存不足为kernel_size大小的数据保存，自动补足NAN至kernel_size大小；
  - 当ceil_mode = false时，剩余数据不足kernel_size大小时，直接舍弃。
- 作用
  - 1 挑选不受位置干扰的图像信息。
  - 2 对特征进行降维，提高后续特征的感受野，也就是让池化后的一个像素对应前面图片中的一个区域。
  - 3 因为池化层是不进行反向传播的，而且池化层减少了特征图的变量个数，所以池化层可以减少计算量。

非线性激活函数

- ReLU
  - torch.nn.ReLU(inplace=False)
  - inplace参数
    - inplace = False 时,不会修改输入对象的值,而是返回一个新创建的对象.
    - inplace = True 时,会修改输入对象的值
- Sigmoid
  - torch.nn.Sigmoid(*args, **kwargs)
- 作用
  - 1.模拟人类神经元的传递规则
  - 2.限制每层之间的输出值范围
  - 3.为神经网络引入非线性的能力

其它层

- Sparse 层
- Dropout 层
- Transformer 层
- 循环层
- 正则化层
  - BatchNorm2d
    - torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True, device=None, dtype=None)
- 线性层
  - torch.nn.Linear(in_features, out_features, bias=True, device=None, dtype=None)

nn.Sequential

损失函数

- 作用
  - 损失函数使用主要是在模型的训练阶段，每个批次的训练数据送入模型后，通过前向传播输出预测值，然后损失函数会计算出预测值和真实值之间的差异值。也就是损失值。得到损失值之后，模型通过反向传播去更新各个参数，来降低真实值与预测值之间的损失，使得模型生成的预测值往真实值方向靠拢，从而达到学习的目的。
- L1Loss
  - $\ell(x, y)=\frac{1}{N} \sum_{n}^{N}\left|x_{n}-y_{n}\right|$
- MSELoss
  - $\ell(x, y)=L=\{l_1, \ldots, l_N\}^{\top}, \quad l_n=\left(x_n-y_n\right)^2$
  - $\ell(x, y)=\begin{cases}\operatorname{mean}(L), & \text{if reduction} = \text{mean} \\ \operatorname{sum}(L), & \text{if reduction} = \text{sum}\end{cases}$
- CrossEntropyLoss
  - $\mathcal{L}=\frac{1}{N} \sum_{i=1}^{N} \ell_i=\frac{1}{N} \sum_{i=1}^{N}-\log \left(p_{i y_i}\right)$
  - $p_{ic}=\frac{\exp \left(x_{ic}\right)}{\sum_{j=1}^{C} \exp \left(x_{ij}\right)}$

反向传播

优化器

- 参数
  - params： 模型参数
  - lr 学习速率

Example:

```
for input, target in dataset:
    optimizer.zero_grad()
    output = model(input)
    loss = loss_fn(output, target)
    loss.backward()
    optimizer.step()
```

模型操作

- 现有模型的使用以及修改（以vgg_16为例）
  - 使用
    - vgg16_false = torchvision.models.vgg16(pretrained=False)
  - 修改
    - 增加层
      - vgg16_false.classifier.add_module('add_linear',nn.Linear(1000,10))
    - 修改层
      - vgg16_false.classifier[6] = nn.Linear(4096, 10)
- 模型的保存与读取
  - 方式一
    - 保存
      - #保存方式1(模型结构+模型参数)
torch.save(vgg16,"vgg16_method1.pth")
    - 读取
      - #方式1 -》保存方式1，加载模型
model = torch.load("vgg16_method1.pth")
print(model)
  - 方式二
    - 保存
      - #保存方式2(模型参数)(官方推荐)
torch.save(vgg16.state_dict(),"vgg16_method2.pth")
    - 读取
      - #方式2，加载模型
vgg16 = torchvision.models.vgg16(pretrained=False)
vgg16.load_state_dict(torch.load("vgg16_method2.pth"))
print(vgg16)