PyTorch接口总结

1.Tensor

1.1 Tensor类型

通常使用`Tensor`类的构造函数返回的是`FloatTensor`类型对象，可以通过在对象上调用`cuda()`返回一个新的`cuda.FloatTensor`类型的对象。

`torch`模块内提供了操作tensor的接口，而`Tensor`类型的对象上也设计了对应的接口。例如：`torch.add()`与`tensor.add()`等价。

想就地修改一个tensor对象，需要使用加后缀下划线的方法。

Data Type	CPU tensor	GPU tensor
32-bit floating point	torch.FloatTensor	torch.cuda.FloatTensor
64-bit floating point	torch.DoubleTensor	torch.cuda.DoubleTensor
16-bit floating point	torch.HalfTensor	torch.cuda.HalfTensor
8-bit integer (unsigned)	torch.ByteTensor	torch.cuda.ByteTensor
8-bit integer (signed)	torch.CharTensor	torch.cuda.CharTensor
16-bit integer (signed)	torch.ShortTensor	torch.cuda.ShortTensor
32-bit integer (signed)	torch.IntTensor	torch.cuda.IntTensor

Data Type	CPU tensor	GPU tensor
64-bit integer (signed)	torch.LongTensor	torch.cuda.LongTensor

1.2 tensor的常见创建接口

方法名	说明
Tensor()	直接从参数构造一个张量，参数支持list， numpy数组
eye(row, column)	创建指定行数、列数的二维单位tensor
linspace(start, end, count)	在区间[s, e]上创建c个tensor
logspace(s, e, c)	在区间[10^s , 10^e]上创建c个tensor
ones(*size)	返回指定shape的张量，元素初始为1
zeros(*size)	返回指定shape的张量， 元素初始为0
ones_like(t)	返回与t的shape相同的张量，且元素初始为1
zeros_like(t)	返回与t的shape相同的张量， 且元素初始为0
arange(s, e, sep)	在区间[s, e)上以间隔sep生成一个序列张量

1.3 随机采样

方法名	说明
rand(*size)	在区间[0, 1)返回一个均匀分布的随机数张量
uniform(s, e)	在指定区间[s, e]上生成一个均匀分布的张量
randn(*size)	返回正态分布N(0, 1)取样的随机数张量
normal(means, std)	返回一个正态分布N(means, std)

1.4 序列化

方法名	说明
save(obj, path)	张量对象的保存，通过pickle进行
load(path)	从文件中反序列化一个张量对象

1.5 数学操作

这些方法均为逐元素处理方法

方法名	说明
abs	绝对值
add	加法
addcdiv(t, v, t1, t2)	t1与t2按元素相除后，乘v加t
addcmul(t, v, t1, t2)	t1与t2按元素相乘后，乘v加t
ceil	向上取整
floor	向下取整
clamp(t, min, max)	将张量元素限制在指定区间
exp	指数
log	对数
pow	幂
mul	逐元素乘法
neg	取反
sign	取符号
sqrt	开根号
sigmoid	
tanh	
relu	

注：这些操作均创建新的tensor， 如果需要就地操作，可以使用下划线。

1.6 归约方法

方法名	说明
cumprod(t, axis)	在指定维度对t进行累积
cumsum(t, axis)	指定维度对t进行累加
dist(a, b, p=2)	返回a, b之间的p阶范数
mean	均值
median	中位数
std	标准数
var	方差
norm(t, p=2)	返回t的p阶范数
prod(t)	返回t所有元素的积
sum(t)	返回t所有元素的和

1.7 比较方法

方法名	说明
eq	比较tnesor是否相等，支持broadcast
equal	比较tensor是否有相同的shape和值
ge/le	大于/小于比较
gt/lt	大于等于/小于等于比较
max/min(t, axis)	返回最值，若指定axis， 则额外返回下标
topk(t, k, axis)	在指定的axis维上取最高k个值

1.8 其他操作

方法名	说明
cat(iterable, axis)	在指定的维度上拼接序列
chunk(tensor, c, axis)	在指定的维度上分割tensor
squeeze(input, dim)	将张量维度为1的dim进行压缩，不指定dim则压缩所有维度为1的维度
unsqueeze(dim)	squeeze的逆操作
transpose(t)	计算矩阵的转置
cross(a, b, axis)	在指定维度上计算向量积
diag	返回对角线元素
hist(t, bins)	计算直方图
trace	返回迹

1.9 矩阵操作

方法名	说明
dot(t1, t2)	计算张量的内积
mm(t1, t2)	计算矩阵乘法
mv(t1, v1)	计算矩阵与向量乘法
qr(t)	计算t的QR分解
svd(t)	计算t的SVD分解

1.10 tensor对象的方法

方法名	作用
size()	返回张量的shape属性值
numel(input)	计算tensor的元素个数
view(*shape)	修改tensor的shape，与np.reshape类似，view返回的对象共享内存
resize	类似于view， 但size超出时会重新分配内存空间
item	若为单元素tensor，则返回python的scalar
from_numpy	从numpy数据填充
numpy	返回ndarray类型数据

tensor对象由两部分组成，tensor的信息与存储，storage封装了真正的data，可以由多个tensor共享。

2.自动求导

变量	作用
requires_grad	表示变量是否需要计算导数
grad_fn	变量的梯度函数
grad	变量对应的梯度

3.神经网络结构

3.1 神经网络层

`torch.nn`模块提供了创建神经网络的基础构件，这些层都继承自Module类。

Layer对应的类	功能说明
Linear(in_dim, out_dim, bias=True)	提供了进行线性变换操作的功能
Dropout(p)	Dropout层，有2D,3D的类型
Conv2d(in_c, out_c, filter_size, stride, padding)	二维卷积层，类似的有Conv1d，Conv3d
ConvTranspose2d()	
MaxPool2d(filter_size, stride, padding)	二维最大池化层
MaxUnpool2d(filter, stride, padding)	逆过程
AvgPool2d(filter_size, stride, padding)	二维平均池化层
FractionalMaxPool2d	分数最大池化
AdaptiveMaxPool2d([h,w])	自适应最大池化
AdaptiveAvgPool2d([h,w])	自适应平均池化
ZeroPad2d(padding_size)	零填充边界
ConstantPad2d(padding_size,const)	常量填充边界
ReplicationPad2d(ps)	复制填充边界
BatchNorm1d()	对2维或3维小批量数据进行标准化操作
RNN(in_dim, hidden_dim, num_layers, activation, dropout, bidi, bias)	构建RNN层

Layer对应的类	功能说明
RNNCell(in_dim, hidden_dim, bias, activation)	RNN单元
LSTM(in_dim, hidden_dim, num_layers, activation, dropout, bidi, bias)	构建LSTM层
LSTMCell(in_dim, hidden_dim, bias, activation)	LSTM单元
GRU(in_dim, hidden_dim, num_layers, activation, dropout, bidi, bias)	构建GRU层
GRUCell(in_dim, hidden_dim, bias, activation)	GRU单元

3.2非线性激活层

激活曾类名	作用
ReLU(inplace=False)	Relu激活层
Sigmoid	Sigmoid激活层
Tanh	Tanh激活层
Softmax	Softmax激活层
Softmax2d	
LogSoftmax	LogSoftmax激活层

3.3 容器类型

容器类型	功能
Module	神经网络模块的基类
Sequential	序列模型，类似于keras，用于构建序列型神经网络
ModuleList	用于存储层，不接受输入
Parameters(t)	模块的属性，用于保存其训练参数
ParameterList	参数列表

3.4 其他层

容器类型	功能
Embedding(vocab_size, feature_dim)	词向量层
Embeddingbag	

3.5 模型的保存

只需要保存各层的权重数据即可，这些数据保存在模块的`state_dict`字典中，只需要序列化与反序列化这个字典即可。

```
# 模型的保存
torch.save(model.state_dict, 'path')
# 模型的加载
model.load_state_dict('path)
```

例如：LeNet神经网络实现：

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

class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

```

4.损失函数与优化方法

4.1 损失函数

由`torch.nn`模块提供损失函数类

类名	功能
MSELoss	均方差损失
CrossEntropyLoss	交叉熵损失
NLLLoss	负对数似然损失
PoissonNLLLoss	带泊松分布的负对数似然损失

4.2 优化方法

由`torch.optim`提供优化方法类

类名	功能
SGD(params, lr=0.1, momentum=0, dampening=0, weight_decay=0, nesterov=False)	随机梯度下降法
Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08, weight_decay=0, amsgrad=False)	Adam
RMSprop(params, lr=0.01, alpha=0.99, eps=1e-08, weight_decay=0, momentum=0, centered=False)	RMSprop
Adadelta(params, lr=1.0, rho=0.9, eps=1e-06, weight_decay=0)	Adadelta
Adagrad(params, lr=0.01, lr_decay=0, weight_decay=0, initial_accumulator_value=0)	Adagrad
lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=10, verbose=False, threshold=0.0001, threshold_mode='rel', cooldown=0, min_lr=0, eps=1e-08)	学习率的控制

例如：对不同层的网络设置不同的学习率：

```
class model(nn.Module):
    def __init__():
        super(model,self).__init__()
        self.base = Sequencial()
        # code for base sub module
        self.classifier = Sequencial()
        # code for classifier sub module

optim.SGD([
            {'params': model.base.parameters()},
            {'params': model.classifier.parameters(), 'lr': 1e-3}
            ], lr=1e-2, momentum=0.9)
```

4.3 参数初始化

PyTorch中的参数通常由默认的初始化策略，不需要自己指定，但保留有接口。

初始化方法	说明
xavier_uniform_	
xavier_normal_	
kaiming_uniform_	

使用方法：

```
from torch.nn import init

# net的类定义
...

# 初始化各层权重
for name, params in net.named_parameters():
    init.xavier_normal(param[0])
    init.xavier_normal(param[1])
```

5. 数据集与数据加载器

5.1 Dataset与DataLoader

`torch.util.data`模块提供了`DataSet`类用于描述一个数据集。定义自己的数据集需要继承自`DataSet`类，且实现`__getitem__()`与`__len__()`方法。
`__getitem__`方法返回指定索引处的tensor与其对应的label。

为了支持数据的批量及随机化操作，可以使用data模块下的DataLoader类型来返回一个加载器：
`DataLoader(dataset, batch_size=1, shuffle=False, sampler=None, batch_sampler=None, num_workers=0)`。

5.2 torchvision

torchvision是配合pytorch的独立计算机视觉数据集的工具库，下面介绍其中常用的数据集类型。

`torchvision.datasets.ImageFolder(dir, transform, label_map,loader)`
提供了从一个目录初始化出来一个图片数据集的便捷方法。
要求目录下的图片分类存放，每一类的图片存储在以类名为目录名的目录下，方法会将每个类名映射到唯一的数字上，如果你对数字有要求，可以用`label_map`来定义目录名到数字的映射。

`torchvision.datasets.DatasetFolder(dir,transform, label_map, loader, extensions)`
提供了从一个目录初始化一般数据集的便捷方法。目录下的数据分类存放，每类数据存储在class_xxx命名的目录下。

此外`torchvision.datasets`下实现了常用的数据集，如CIFAR-10/100, ImageNet, COCO, MNIST, LSUN等。

除了数据集，torchvision的`model`模块提供了常见的模型实现，如Alex-Net, VGG, Inception, Resnet等。

5.3torchvision提供的图像变换工具

torchvision的`transforms`模块提供了对`PIL.Image`对象和`Tensor`对象的常见操作。如果需要连续应用多个变换，可以使用`Compose`对象组装多个变换。

转换操作	说明
Scale	PIL图片进行缩放
CenterCrop	PIL图片从中心位置剪切
Pad	PIL图片填充
ToTensor	PIL图片转换为Tensor且归一化到[0,1]
Normalize	Tensor标准化
ToPILImage	将Tensor转为PIL表示

例如：

```
import torchvision.tranforms as Trans
tranform = Trans.Compose([
                           T.Scale(28*28),    # 图片缩放
                           T.ToTensor(),    # 转为tensor并归一化[0, 1]
                           T.Normalize([0.5],[0.5])  # 标准化
                          ])
```

6. 训练过程可视化

使用Tensorborad

通过使用第三方库tensorboard_logger，将训练过程中的数据保存为日志，然后便可以通过Tensorboard来查看这些数据了。其功能支持相对有限，这里不做过多介绍。

使用visdom

visdom是facebook开源的一个可视工具，可以用来完成pytorch训练过程的可视化。
安装可以使用pip install visdom
启动类似于tb，在命令行上执行：python -m visdom.server
服务启动后可以使用浏览器打开http://127.0.0.1:8097/即可看到主面板。

visdom的绘图API类似于plot，通过API将绘图数据发送到基于tornado的web服务器上并显示在浏览器中。更详细内容参见visdom的github主页