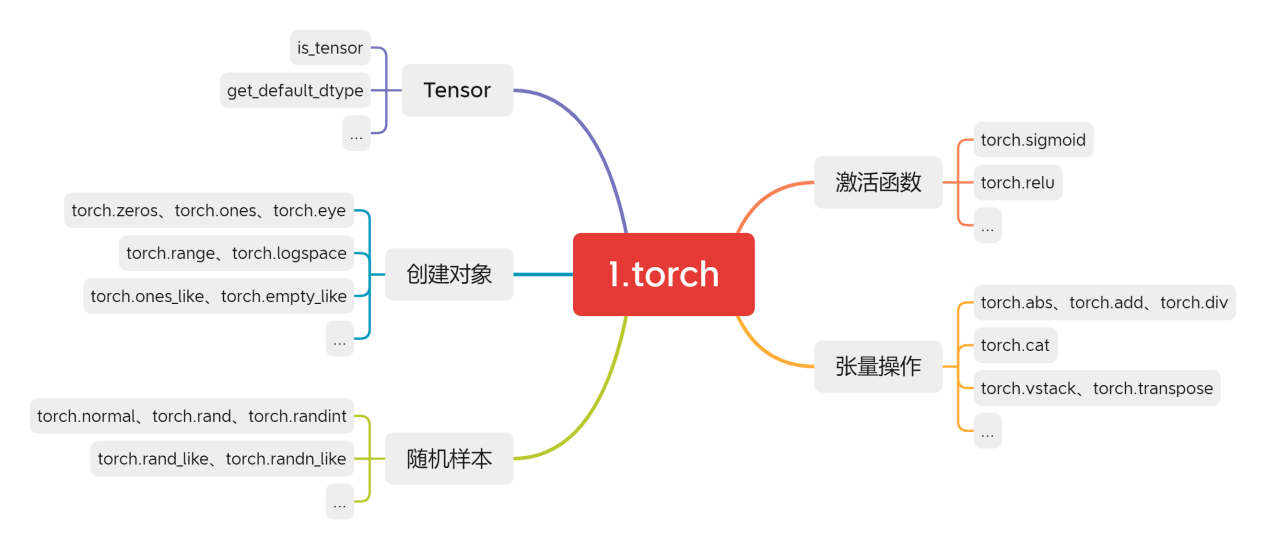
**PyTorch实现卷积神经网络三-Pytorch模块介绍**

上次次推文我们讲述了卷积神经网络的卷积层和池化层的相关知识点，包括卷积操作和池化操作，这也是深度学习的基础，考虑到Pytorch中的模块很多，我们有必要对这些模块进行一些简单的梳理，这样在学习的时候就更加的清晰，在使用的时候也就变得更加顺手，接下来我们就开始这项工作。

**这里要注意两点：**

1. **Pytorch是Python版本的torch的意思。**
2. **PyTorch如果张量的方法后缀带下画线，则在使用该方法会修改张量本身的数据。**
3. **torch模块**

torch模块本身包含了Pytorch很多经常使用的激活函数，比如我们常说的torch.sigmoid、torch.relu等等，其次这个模块中还涉及到很多关于Tensor的操作方法如我们熟悉的torch.add、torch.div等等，最后还有产生一定形状的张量的方法比如torch.zero、torch.randn等等。



1. **torch.Tensor模块**

torch.Tensor模块定义了torch中的张量类型，其中的张量有不同的数值类型，如单精度、双精度浮点、整数类型等，而且张量有一定的维数和形状。同时，张量的类中也包含着一系列的方法，返回新的张量或者更改当前的张量。具体方法大家可以从这个链接中进行查询：

<https://pytorch.org/docs/1.12/tensors.html>

**import** torch  
**import** numpy **as** np  
a = torch.tensor([[1., -1.], [1., -1.]])  
b = torch.tensor(np.array([[1, 2, 3], [4, 5, 6]]))  
print(a,**'\n'**)  
print(a.T,**'\n'**)  
print(a.device,**'\n'**)  
print(a.dtype,**'\n'**)

# 输出

tensor([[ 1., -1.],

[ 1., -1.]])

tensor([[ 1., 1.],

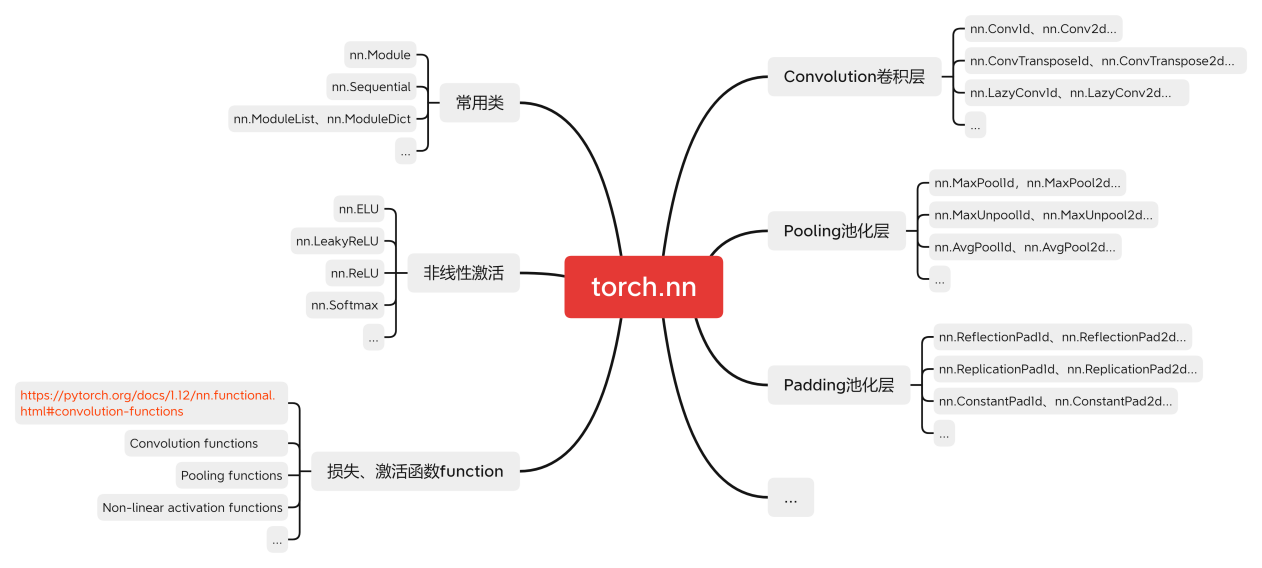
[-1., -1.]])

cpu

torch.float32

1. **torch.nn模块**

torch.nn是Pytorch中一个非常重要的模块，也是经常要使用到的一个模块，是我们在构建神经网络时的核心。这个模块定义了一系列的神经网络层，包括卷积层（nn.ConvNd,N=1,2,3）、线性层（nn.Linear）等等。另外，torch.nn中也定义了一系列的损失函数，包括平方损失函数（torch.nn.MSELoss）、交叉熵损失函数（torch.nn.CrossEntropyLoss）等。



1. **torch.sparse模块**

稀疏矩阵的研究一直是一个热门的研究方向，在工程实践中我们也一直心系稀疏矩阵。在torch中大多数的tensor是连续存储在内存中的，从而可以使用各种算法进行处理。但是在面对稀疏tensor的时候，由于稀疏tensor具有使大部分元素等于零的属性，这意味着如果仅存储或/和处理非零元素，则可以节省大量内存和处理器资源。目前我们可以使用不同的稀疏存储格式（[例如COO，CSR / CSC，LIL等](https://en.wikipedia.org/wiki/Sparse_matrix)）来存储稀疏矩阵元素，这些格式针对稀疏tensor中非零元素的特定结构以及Tensor上的特定操作进行了优化。

在Pytorch中torch.sparse模块定义了稀疏张量，其中构造的稀疏tensor采用的是COO格式（Coordinate），其是使用一个长整型来存储非零元素的位置，用浮点数张量存储对应非零元素的值，当然稀疏tensor之间可以进行s诸如加、减、乘、除运算和矩阵乘法运算。

1. **torch.cuda模块**

torch.cuda 模块定义了与CUDA运算相关的一系列函数，支持了张量在GPU上进行运算。这个模块提供的功能包括：

1. 检查系统CUDA是否可用
2. GPU内存管理
3. 设置GPU计算流(Stream)
4. ...

比如有这些方法：

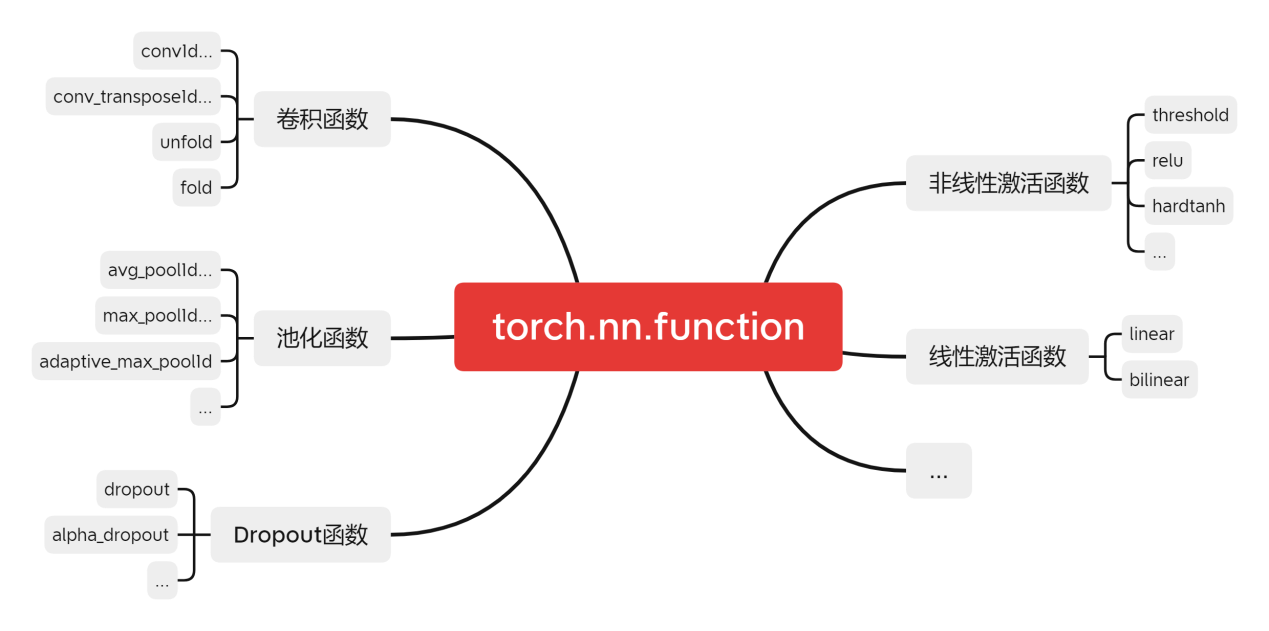
**import** torch  
  
*# 检查当前设备的GPU可用情况*print(torch.cuda.is\_available())  
*# 获取GPU的数量*print(torch.cuda.device\_count())  
*# 获取GPU的名称*print(torch.cuda.get\_device\_name())  
*# 显存的使用量*print(torch.cuda.memory\_usage())  
*# 清空缓存*print(torch.cuda.empty\_cache())

1. **torch.nn.function模块**

torch.nn.functional是PyTorch的函数模块，严格意义上来说是nn模块中的函数。torch.nn.functional提供了一系列神经网络相关的函数，包括卷积函数、池化函数和非线性激活函数（torch.nn.functional.relu6）等，这些函数也是深度学习模型构建的基础。

torch.nn中定义的模块一般会调用torch.nn.functional里的函数，比如，nn.ConvNd模块（N=1，2，3）会调用torch.nn.functional.convNd函数（N=1，2，3），在conv.py文件中我们可以找到答案：





1. **torch.nn.autograd模块**

torch.autograd模块是PyTorch的自动微分算法模块，这个模块提供实现任意标量值函数的自动微分的类和函数。torch.autograd.backward函数，主要用于在求得损失函数之后进行反向梯度传播，torch.autograd.grad函数用于一个标量张量（即只有一个分量的张量）对另一个张量求导，另外该模块还有其他的功能函数。

torch.autograd.backward：计算给定张量相对于计算图的叶子节点的梯度之和。

torch.autograd.grad：计算并返回输出相对于输入的梯度之和。

1. **torch.nn.distributed模块**

torch.distributed是PyTorch的分布式计算模块，主要功能是提供PyTorch并行运行环境，其主要支持的后端有MPI、Gloo和NCCL三种,不同的后端有不同的功能。PyTorch分布式包支持Linux(稳定)、MacOS(稳定)和Windows(原型)。默认情况下，对于Linux, Gloo和NCCL后端被构建并包含在PyTorch分发版中(NCCL仅在使用CUDA构建时)。MPI是一个可选的后端，只有从源代码构建PyTorch时才能包含它。(例如，在安装了MPI的主机上构建PyTorch。)

torch.nn.distributed模块不但提供了后端的一个包装，还提供了一些启动方式来启动多个进程，包括但不限于通过网络（TCP）、通过环境变量、通过共享文件等。

1. **torch.fx模块**

PyTorch1.9版本正式发布了[torch.fx](https://link.zhihu.com/?target=https://pytorch.org/docs/stable/fx.html" \t "https://zhuanlan.zhihu.com/p/_blank)预览版，该模块是开发人员用于转换nn的工具箱。fx由三个主要组件组成:符号跟踪器、中间表示和Python代码生成。具体有什么用，大家可以参数文章：

<https://www.cnblogs.com/bigoldpan/p/16296035.html>

1. **torch.hub模块**

torch.hub 提供了一系列预训练的模型供用户使用。支持通过添加一个简单的文件将预先训练的模型（模型定义和预先训练的权重）发布到github存储库。

Pytorch Hub 提供了方便的 API，用于通过 [torch.hub.list（）](https://pytorch.org/docs/1.12/hub.html" \l "torch.hub.list" \o "torch.hub.list) 探索 hub 中的所有可用模型，通过 [torch.hub.help（）](https://pytorch.org/docs/1.12/hub.html" \l "torch.hub.help" \o "torch.hub.help) 显示文档字符串和示例，并使用 [torch.hub.load（）](https://pytorch.org/docs/1.12/hub.html" \l "torch.hub.load" \o "torch.hub.load) 加载预训练模型。

1. **torch.jit模块**

JIT全称是 Just In Time Compilation，也就是即时编译。基于JIT可以将Python模型转换为 TorchScript Module，且使用TorchScript的方式可以创建序列化和可优化的模型，基于Tracing和Script两种方式，可以将一个Python代码转化为TorchScript代码，继而将导出相应的模型进行不断的优化，或者被C++所调用，最终实现对生产环境下的支持，毕竟Python并不适合做部署。

总而言之torch.jit这个模块存在的意义是把PyTorch的动态图转换成可以优化和序列化的静态图，其主要工作原理是通过输入预先定义好的张量，追踪整个动态图的构建过程，得到最终构建出来的动态图，然后转换为静态图（通过中间表示，即IntermediateRepresentation，来描述最后得到的图），另外JIT也可以用来生成ONNX等其他格式的神经网络描述文件。

1. **torch.nn.init模块**

该模块用于神经网络权重的初始化。神经网络权重的初始值对神经网络的训练以及模型收敛有很大程度的影响。如果初始的权重取值不合适，重则导致后续的优化过程收敛很慢，甚至不收敛,因此官方就提供了一个非常方便的模块，比如该模块就有这些函数：

1. torch.nn.init.calculate\_gain
2. torch.nn.init.uniform\_
3. torch.nn.init.normal
4. ...
5. **torch.optim模块**

torch.optim模块定义了一系列的优化器，如：

1. torch.optim.SGD（随机梯度下降算法）
2. torch.optim.Adagrad（AdaGrad算法）
3. torch.optim.RMSprop（RMSProp算法）
4. torch.optim.Adam（Adam算法）
5. ...

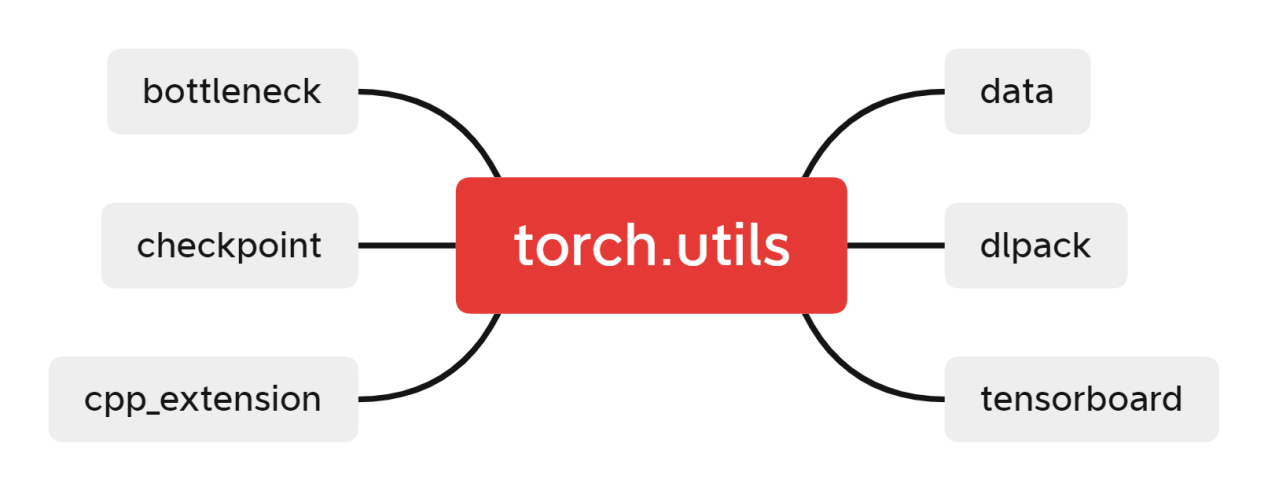
等。当然，这个模块还包含学习率衰减的算法的子模块，即torch.optim.lr\_scheduler.

1. **torch.onnx模块**

torch.onnx定义了PyTorch导出和载入ONNX格式的深度学习模型描述文件。ONNX格式的存在是为了方便不同深度学习框架之间进行交换模型。引入这个模块可以方便PyTorch导出模型给其他深度学习框架使用，或者让PyTorch可以载入其他深度学习框架构建的深度学习模型。

1. **torch.utils模块**

torch.utils 提供了一些列的辅助模块,我们来看一下：



参考资料：

<https://pytorch.org>

<https://zhuanlan.zhihu.com/p/334788042>

<https://www.cnblogs.com/liulunyang/p/14356789.html>

<https://blog.csdn.net/m0_46175303/article/details/113880749>

