# 一文搞定Pytorch+CNN讲解

hadxu
不喜欢看kindle的学生不是好程序员
关注他





在折腾过各种神经网络框架之后，我决定入Pytorch坑。

如果你是科研或者学习之用，强烈推荐Pytorch，如果是工业使用，需要大规模部署，请转Tensorflow。

**Pytorch简单入门**

- **Pytorch中最重要的就是Variable模块，该模块集成了围绕一个张量所有的操作，包括前向传播、反向传播的各种求偏导数的数值。**
- **Pytorch所有的网络在nn包里，我们待会会实现经典的Lenet5模型。**
- **Pytorch计算GPU和CPU切换很快，直接使用x.cuda()即可**

**Lenet5模型的实现：**

网上的Lenet5已经烂大街了，为什么还要讲一下呢？原因在于今天我在学习经典的神经网络的时候，发现Lenet5论文中在卷积层之后直接得到120个全连接层，我就一直在考虑120是哪来的？问了很多人，都没有回答我，问了师兄，师兄直接说看ufldl去。于是自己做实验，一步一步研究，终于得出了结果120是你随便设的！如果你和我一样，开始不知道为啥，看了我这篇文章就懂了，涉及概念比较多，我们先剖析代码，用到什么，就解决什么。

**导入各种库**

```
import torch
from torch.autograd import Variable
import numpy as np
import torch.nn as nn
from torchvision import datasets,transforms
```

- *Variable是Pytorch数据格式模块*
- *nn是神经网络模块*
- *torchvision是Pytorch的外围库，该库包含了各种关于图像的各种功能函数*

**读取模型**

```
train_dataset = datasets.MNIST('data/',download=False,train=True,
```

```
test_dataset = datasets.MNIST('data/',download=False,
                              transform=transforms.Compose([
                                  transforms.ToTensor(),
                                  transforms.Normalize((0.1307,), (0.3081,)),
                              ]))
```

- 我们使用MNIST数据集，一定要记得，该数据集的大小为28*28，通道为1(黑白)。
- transform表示了对数据集进行的操作，包括了转成张量，以及规则话，说道理，如果不进行的话，也没有关系，至于(0.1307,), (0.3081,)怎么来的，我也不知道，抄的官网的。

建立数据集迭代器：

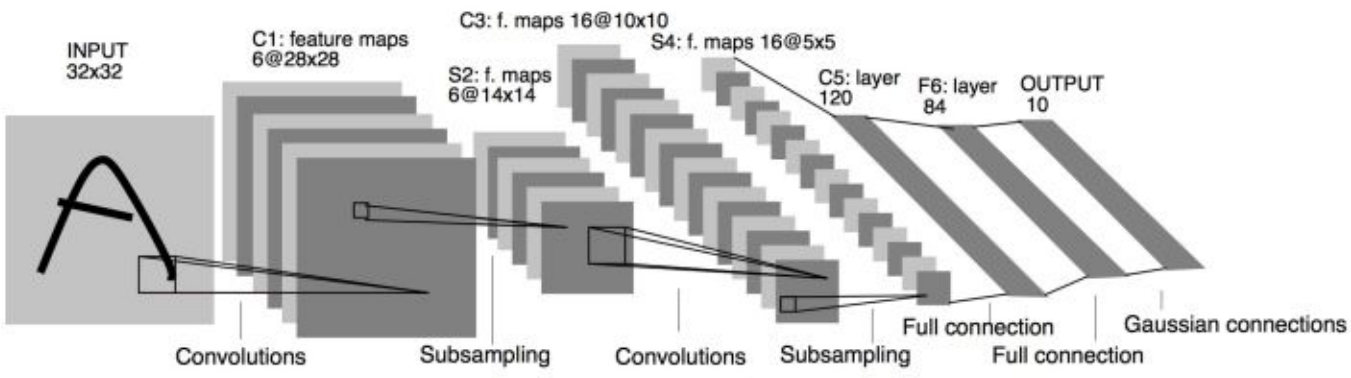
```
train_loader = torch.utils.data.DataLoader(train_dataset,batch_size=64,shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset,batch_size=64,shuffle=True)
```



- Pytorch中，训练最好使用迭代器来进行，不然数据集大，内存吃不消，如果你不知道迭代器是什么（手动再见）
- 我们使用的batch_size为64，有的人好奇为什么使用64，或者32，我的理解是这样的，当我们的数据大小为2的幂次数，计算机会计算的特别快，因为计算机是二进制嘛，如果是2的幂次数的话，计算机很容易通过移位来进行计算。
- shuffle(打乱)将数据集打乱。

**建立神经网络**

**首先我们要实现Lenet5模型，请看(yann.lecun.com/exdb/pub...)**

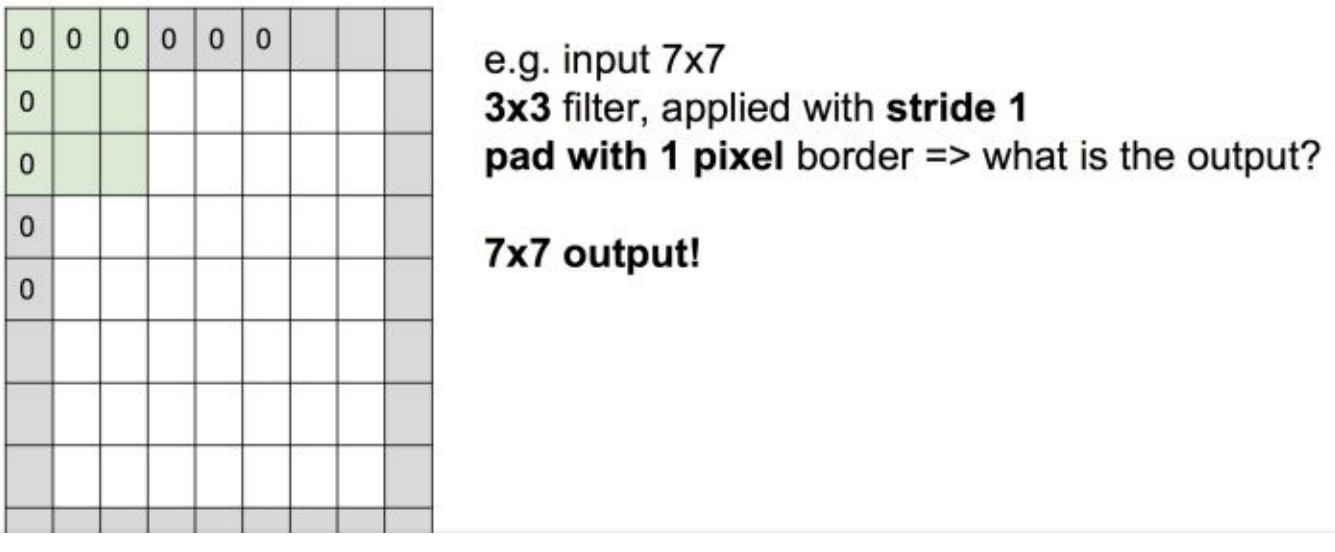



**有没有发现问题?**

- **最大的问题就是，我们的数据集明明是28*28的，怎么在论文中变成32*32了?**
- **按照论文的标记，C表示卷积，S表示池化，但是在S4到C5应该是卷积，怎么变成了全连接层了?**

**是不是同学们都有这个疑问呢?**

**解答： 原因在于使用了padding技术（具体请参考cs231ncs231n.stanford.edu/sli...)**



论文使用的卷积核都是为5*5的，那么根据上图的逻辑，下面的层数应该是这样的：

1. 28*28*1输入,首先padding=2，变成32*32*1
2. 6个卷积核 输出 6@28*28，（28=32-5+1）步长为1
3. 池化 输出 6@14*14，池化步长为2
4. 16卷积核 输出16@10*10
5. 池化 16@5*5

关键点来了：根据论文的实现，下一层应该是卷积，使用了120个卷积核，也就是120@1*1（1=5-5+1），可以发现，进行卷积以后，变成了全连接层，（非常重要，如果不能理解，卷积神经网络展开成全连接就不懂。）同时，我查看了网友的解答，很多朋友在这里说可以将其看成全链接，也就是在S4的时候，下一步直接展开，为（16*5*5=400个神经元，然后在全连接到120个神经元），经过试验，这是正确的，我们待会来看如何试验。

接下来的全连接就简单了，先是120到84的全连接（这里是84的解释）：

```
输出层由欧式径向基函数（Euclidean Radial Basis Function）单元组成，每类一个单元，每个有84个
```

然后就是输出类别84到10的输出。到这里，每一层都讲解完了。

首先我们来实现论文中的网络结构（S4到C5采用卷积神经网络）

```
class Net(nn.Module):
    def __init__(self):
        super(Net,self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5,padding=2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.conv3 = nn.Conv2d(16,120,kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.relu = nn.ReLU()
        self.fc1 = nn.Linear(120,84)
        self.fc2 = nn.Linear(84,10)
        self.logsoftmax = nn.LogSoftmax()
    def forward(self,x):
        in_size = x.size(0)
        out = self.relu(self.mp(self.conv1(x)))
        out = self.relu(self.mp(self.conv2(out)))
        out = self.relu(self.conv3(out))
        out = out.view(in_size, -1)
        out = self.relu(self.fc1(out))
        out = self.fc2(out)
        return self.logsoftmax(out)
```

- 首先定义网络结构，如果对pytorch不熟悉的话，请参考Learning PyTorch with Examples
- 首先定义6个卷积核，padding=2，卷积核大小为5
- 再次定义16个，大小一样
- 再次定义120个，大小一样（**这里的120可以随便设，当时卡了好长时间**）
- 定义max_pooling，大小为2
- 定义Relu函数
- 定义全连接120-84
- 定义全连接84-10

**实现另一种网络结构（lenet5另一种解释S4到C5采用全连接）**

- **在S4层，输出为16**@5*5，那么全连接输出的话就是16*5*5=400个神经元，那么神经网络如下：

```
class Net(nn.Module):
    def __init__(self):
        super(Net,self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5,padding=2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.mp = nn.MaxPool2d(2)
        self.relu = nn.ReLU()
        self.fc1 = nn.Linear(16*5*5,120)  # 必须为16*5*5
        self.fc2 = nn.Linear(120,84)
        self.fc3 = nn.Linear(84,10)
        self.logsoftmax = nn.LogSoftmax()

    def forward(self,x):
        in_size = x.size(0)
        out = self.relu(self.mp(self.conv1(x)))
        out = self.relu(self.mp(self.conv2(out)))
        out = out.view(in_size, -1)
        out = self.relu(self.fc1(out))
        out = self.relu(self.fc2(out))
        out = self.fc3(out)
        return self.logsoftmax(out)
```

这样跑下来的结果是**99.567%**

**两者效果是差不多的，也就证明的我前面的观点：**

**卷积以后如果是1*1的结果，直接拿来作为全连接网络即可。整个代码在我的github中，请多多star(**HadXu/machine-learning**)**

发布于 2017-10-14

PyTorch



**文章被以下专栏收录**