

赞同 47

分享

PYTORCH

PyTorch简明笔记[3]-神经网络的基本组件（Layers、functions）



蛭蛭

深度学习小学生，NLP爱好者。个人分享公众号：SimpleAI

[关注他](#)

47 人赞同了该文章



不断地被人安利PyTorch，终于忍不住诱惑决定入坑了。

当我翻看PyTorch官网的时候，一下子就爱上了它那清晰的文档和友好的入门指南。所以决定好好地系统性地把PyTorch学一学。所以，记一份适合自己的更加清晰简明的笔记，把基础打牢固，就很有必要了。

这份笔记的目的，主要是方便随时查阅，不必去看详细的冗长的原始文档。也方便跟我一样的小白可以迅速入门，快速实践。同时，我来记录笔记的过程中，也会补充深度学习相关的知识，在学习PyTorch框架的时候，也学习/复习深度学习。

本篇是PyTorch简明笔记第[3]篇。

前言：

PyTorch的 `torch.nn` 中包含了各种神经网络层、激活函数、损失函数等等的类。我们通过 `torch.nn` 来创建对象，搭建网络。

PyTorch中还有 `torch.nn.functional`，让我们可以通过调用函数的方式，来直接搭建网络，而不用像 `torch.nn` 一样要先创建对象。

我们可以按照自己的习惯，结合上面两种方法，来搭建网络。

一般情况下，对于像Conv层这种需要定义多个参数的时候，我们采用 `torch.nn` 的方式比较方便，而对于参数比较少的，或者不用设置参数的，尤其是一些函数，我们就可以采用 `torch.nn.functional` 来定义。一般我们 `import torch.nn.functional as F`，这样后面写起来方便一些。

`torch.nn.Module` 是所有神经网络模型的基本类（basic class），所有的模型都应该是它的子类。



赞同 47

分享

```
# 调用nn.Module的初始化方法
super(Model, self).__init__()          # 添加该模型的自定义初始化（主要是定义神经网络层）
self.conv1 = nn.Conv2d(1, 20, 5)
self.conv2 = nn.Conv2d(20, 20, 5)      # 定义模式的输出是怎么计算的
# （主要写各层之间通过什么激活函数、池化等等来连接）
def forward(self, x):
    x = F.relu(self.conv1(x))           return F.relu(self.conv2(x))
```

通过上面的方式定义了模型类之后，我们就可以使用 `nn.Module` 内置的 `.parameters()` 方法来获取模型的参数。我们后面要更新的就是这些参数。

一、常用的神经网络层

这里，我们介绍以下几种layers：

1. 卷积层-Conv2d
2. 全连接层
3. 池化层
4. Dropout
5. BatchNorm

1.卷积层 (2D)

CLASS

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True)
```

- 注意是个类，要创建对象后再使用。
- 参数中的 `kernel_size` , `stride` , `padding` , `dilation` 的值，可以为int，也可以为tuple。是int的时候，就代表长宽相等。
- Input size为(N,C_{in},H,W);
Output size为(N,C_{out},H_{out},W_{out}).
其中，N为batch size，即样本数，C为channel数，H为height，W为width。

举例：

```
1  # 16通道进来，64通道出去，kernel为3×3，stride为2
2  conv1 = nn.Conv2d(16,64,3,stride=2)
3  # 100个16通道的32×32数据
4  input_data = torch.rand(100,16,32,32)
5  output_data = conv1(input_data)
6  print(output_data.size())

torch.Size([100, 64, 15, 15])
```

知乎 @蝈蝈

2.全连接层/线性层

赞同 47

举例：

分享

```
1 # 全连接层，32个特征输入，128个特征输出。
2 # 相当于一个128个神经元神经网络层
3 fc1 = nn.Linear(32, 128)
4 # 100个样本，每个样本有32个特征值
5 input = torch.randn(100, 32)
6 output = fc1(input)
7 print(output.size())
```

torch.Size([100, 128])

知乎 @蛭蛭

3.Pooling (2D) 层

采用CLASS方式

`torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)`

`torch.nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True)`

举例：

```
1 # Maxpooling2D
2 maxpool = nn.MaxPool2d(2, 2)
3 input = torch.randn(100, 3, 64, 64)
4 output = maxpool(input)
5 print(output.size())
```

torch.Size([100, 3, 32, 32])

知乎 @蛭蛭

采用Function方式：（似乎更简洁）

`F.avg_pool2d(input, kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True) → Tensor`

```
1 # Maxpooling2D
2 input = torch.randn(100, 3, 64, 64)
3 output = F.max_pool2d(input, 2, 2)
4 print(output.size())
```

torch.Size([100, 3, 32, 32])

知乎 @蛭蛭

4.Dropout 层

CLASS

`torch.nn.Dropout(p=0.5, inplace=False)`

`torch.nn.Dropout2d(p=0.5, inplace=False)`



举例:

```
1 # Dropout
2 dp = nn.Dropout(p=0.5)
3 input = torch.randn(10,1)
4 print("-----Before dropout:-----\n",list(input))
5 print("-----After dropout:-----\n",list(dp(input)))

-----Before dropout:-----
[tensor([0.0369]), tensor([0.2959]), tensor([-1.3982]), tensor([-1.1381]), tensor([0.1552]), tensor([0.1408]), tensor([2.0603]), tensor([-1.4063]), tensor([-1.2846]), tensor([0.0919])]
-----After dropout:-----
[tensor([0.]), tensor([0.]), tensor([-2.7964]), tensor([-0.]), tensor([0.3105]), tensor([0.1205]), tensor([-2.8126]), tensor([-0.]), tensor([0.]), tensor([0.])]
```

5.BatchNorm (2D)

CLASS

`torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)`

BN层的主要作用是，通过对数据进行标准化，来加速神经网络的训练。

唯一必须设置的参数 `num_features` 要等于输入数据(N,C,H,W)中的C，就是Channel数。

```
1 # BatchNorm
2 input = torch.randn(100,64,128,128)
3 bn = nn.BatchNorm2d(64)
4 output = bn(input)
5 print(output.size())

torch.Size([100, 64, 128, 128])
```

二、常用的激活函数

采用CLASS方式:

`torch.nn.ReLU(inplace=False)`

`torch.nn.Sigmoid`

`torch.nn.Tanh`

`torch.nn.Softmax(dim=None)`

这些很简单，就不解释了。举例:

```
1 # Activation
2 data = torch.randn(10)
3 softmax = nn.Softmax()
4 output = softmax(data)
5 print(data)
6 print(output)

tensor([ 0.9060,  0.1661,  1.0211,  0.4333, -0.1023,  0.1616, -1.0002,  0.4336,
        -1.0474, -0.2065])
tensor([0.1885, 0.0800, 0.2115, 0.1175, 0.0688, 0.0895, 0.0280, 0.1175, 0.0267,
```



```
1 # Activation
2 data = torch.randn(10)
3 output = F.relu(data)
4 print(data)
5 print(output)
```

tensor([-0.0891, 0.7597, -0.0196, -1.7342, 0.3400, -0.4846, -1.7961, -0.4371, 1.1572, 0.2715])

tensor([0.0000, 0.7597, 0.0000, 0.0000, 0.3400, 0.0000, 0.0000, 0.0000, 1.1572, 0.2715])

知乎 @蛭蛭

三、损失函数

MSE

`torch.nn.MSELoss(size_average=None, reduce=None, reduction='mean')`

Cross-Entropy

`torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean')`

用法也很简单，把预测值 (input) 和标签值 (target) 扔进去就行：

```
1 # Loss function
2 loss_f = nn.CrossEntropyLoss()
3 x = torch.randn(8, 3, requires_grad=True) # batch size=8, num_class = 3
4 labels = torch.ones(8, dtype=torch.long) # 一组 1维 的long类型标签
5 loss = loss_f(x, labels)
6 print(loss)
7 loss.backward()
8 x.grad
```

tensor(0.9460, grad_fn=<NllLossBackward>)

tensor([[0.0037, -0.0989, 0.0952],
 [0.0230, -0.0347, 0.0117],
 [0.0099, -0.0401, 0.0302],
 [0.0037, -0.0248, 0.0211],
 [0.0669, -0.0923, 0.0254],
 [0.0735, -0.0911, 0.0176],
 [0.0243, -0.0846, 0.0603],
 [0.0449, -0.0906, 0.0458]])

知乎 @蛭蛭

这里对Cross-entropy的使用有一点需要注意的地方：

Shape:

- Input: (N, C) where $C = \text{number of classes}$, or

$(N, C, d_1, d_2, \dots, d_K)$ with $K \geq 2$ in the case of K -dimensional loss.

- Target: (N) where each value is $0 \leq \text{targets}[i] \leq C - 1$, or

$(N, d_1, d_2, \dots, d_K)$ with $K \geq 2$ in the case of K -dimensional loss.

知乎 @蛭蛭

比如，我们有5个类别， $C=5$ ，那么你给的标签值必须在 $[0,4]$ 之间，不能取其他的数字。

上面的内容列举了最常见的一些layers和functions。我在举例子的时候，主要是采用 `torch.nn` 定义Class的方式，无论是layer还是函数，都是先创建对象，在用对象去进行操作。上面写的每一个，其实在 `torch.nn.functional` 中都有对应，使用起来相当于省掉了创建对象那一步，所以就不赘述了。

下一篇笔记记录如何使用上面的这些组件，去搭建神经网络，做一个图片分类模型。

往期PyTorch系列笔记：

[PyTorch简明笔记\[1\]-Tensor的初始化和基本操作](#)

[PyTorch简明笔记\[2\]-Tensor的自动求导\(AoutoGrad\)](#)

发布于 2018-12-10

[PyTorch](#) [深度学习 \(Deep Learning\)](#) [TensorFlow](#)

文章被以下专栏收录



DeepLearning学习笔记
Andrew Ng的大型深度学习课程笔记，我将一点点认真地写下我的个人理解的笔记， ...

关注专栏



机器学习算法与自然语言处理
公众号[自然语言处理与机器学习] 微信号yizhennotes

关注专栏

推荐阅读



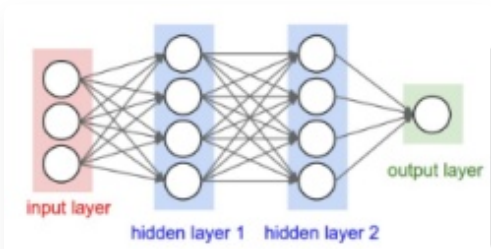
TensorFlow 2.0和PyTorch谁更好？大牛们争了好几天

量子位 发表于量子位



10分钟快速入门PyTorch (4)

Sherl... 发表于深度炼丹



10分钟快速入门PyTorch (3)

Sherl... 发表于深度炼丹

5 条评论

切换为时间排序

写下你的评论...



Mbapey

7 个月前

