

问题汇总

笔记整理人：天国之光（2019 年 6 月 24 日）

说明

- （1） pytorch 使用 0.4 版本的官方文档，安装的是 1.0 版本，其中某些函数的区别，同学在问的时候我会整理好写成 Word 文档。
- （2） 我会把每天学生经常问的问题整理成 Word 文档，先看 Word 文档中有没有解决方案，如果没有再在群里问。整理好的文档会在本篇文章中定期更新，方便大家查阅资料就能解决问题，提高学习效率。
- （3） 项目实战最后一周会进行学习，大家先学基础 API，不然项目也看不懂，打牢基础是最重要的。
- （4） 官方文档有可能有误，具体报错问题可以在群里问，然后会整理到文档中。
- （5） 每天同一个问题大于 2 次，我就会整理到文档中，其他小概率问题忽略。

问题 1：环境配置问题

请参考《1.window+pytorch 配置》文档，只要按步骤来就不会报错，烦请仔细看。

在 Pycharm 中安装 Pytorch

先访问网站：<https://pytorch.org/>

然后根据系统配置，选择下载方式

PyTorch Build	Stable (1.1)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python 2.7	Python 3.5	Python 3.6	Python 3.7
CUDA	9.0	10.0	None	
Run this Command:	<pre> pip3 install https://download.pytorch.org/whl/cu100/torch-1.1.0-cp37-cp37m-win_amd64.whl pip3 install https://download.pytorch.org/whl/cu100/torchvision-0.3.0-cp37-cp37m-win_amd64.whl </pre>			

最后在 Pycharm 的 Terminal 运行命令即可。

问题 2：自动求导机制问题

```

>>> regular_input = Variable(torch.randn(5, 5))
>>> volatile_input = Variable(torch.randn(5, 5), volatile=True)
>>> model = torchvision.models.resnet18(pretrained=True)
>>> model(regular_input).requires_grad
True
>>> model(volatile_input).requires_grad
False
>>> model(volatile_input).volatile
True
>>> model(volatile_input).creator is None
True

```

运行会报错，显示维度不一致：因为 model 需要四维张量（batch,channel,height,width），而文档中是二维张量，所以报错。

注意：去掉 variable 函数，这是 0.4 版本的函数，我们安装的是 1.0 版本

```

regular_input=torch.randn(1,3,224,224)#默认是 True
volatile_input=torch.randn(1,3,224,224)
Model=torchvision.models.resnet18(pretrained=True)
Model(regular_input).requires_grad
with torch.no_grad():
    Model(volatile_input).requires_grad

```

总结：requires_grad=False 时不需要更新梯度，适用于冻结某些层的梯度；volatile=True 相当于 requires_grad=False，适用于推断阶段，不需要反向传

播。这个现在已经取消了，使用 `with torch.no_grad()` 来替代维度不一致问题是函数所需维度和输入维度不一致，这个不是难点，只需根据需求更改即可，比如这个例子 model 需要 `[batch,channel,height,width]=[batch,3,224,224]`，后面讲网络结构会说到。

问题 3：有关 autograd 和 backward 的问题

```
>>> import torch
>>> a=torch.randn(2,3)
>>> a.backward()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "D:\software\install\anaconda\lib\site-packages\torch\tensor.py", line 102, in backward
    torch.autograd.backward(self, gradient, retain_graph, create_graph)
  File "D:\software\install\anaconda\lib\site-packages\torch\autograd\__init__.py", line 90, in backward
    allow_unreachable=True) # allow_unreachable flag
RuntimeError: element 0 of tensors does not require grad and does not have a grad fn
>>> b=torch.randn(2,3,requires_grad=True)
>>> b.backward()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "D:\software\install\anaconda\lib\site-packages\torch\tensor.py", line 102, in backward
    torch.autograd.backward(self, gradient, retain_graph, create_graph)
  File "D:\software\install\anaconda\lib\site-packages\torch\autograd\__init__.py", line 84, in backward
    grad_tensors = _make_grads(tensors, grad_tensors)
  File "D:\software\install\anaconda\lib\site-packages\torch\autograd\__init__.py", line 28, in _make_grads
    raise RuntimeError("grad can be implicitly created only for scalar outputs")
RuntimeError: grad can be implicitly created only for scalar outputs
>>> c=torch.randn(2,3,requires_grad=True)
>>> d=c.mean()
>>> d.backward()
>>> d.grad
```

(1) 紫框部分报错是因为默认创建 tensor 的 `requires_grad=False`，而 backward 需要 `requires_grad=True`；

(2) 蓝框部分报错是因为 backward 需要是 scalar 的数据，也就是需要的是标量类型的张量，简单说 `dim=0` 的张量就标量，`dim=1` 的张量就是向量，`dim=2` 的张量就是二维矩阵。

问题 4：torch.normal 参数问题

```
In[2]: import torch
In[3]: torch.normal(mean=torch.arange(1, 11.0), std=torch.arange(1, 0, -0.1))
Out[3]:
tensor([ 0.1176,  3.4049,  2.8217,  4.0664,  4.9989,  6.5216,  6.3855,  7.8626,
          9.4126, 10.0937])
```

(1) 参数问题，0.4 版本是 means, 1.0 版本是 mean

(2) 数据类型问题, mean 和 std 需要 float(浮点型)类型

问题 5: torch.mul() 函数

```
(base) C:\Users\machi>python
Python 3.6.7 (default, Feb 28 2019, 07:28:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import torch
>>> t1=torch.randn(1,4)
>>> t2=torch.randn(4,1)
>>> torch.mul(t1,t2)
tensor([[ -0.1952, -0.6010, -0.4554, -0.1385],
        [-0.0363, -0.1119, -0.0848, -0.0258],
        [ 0.7025,  2.1622,  1.6386,  0.4983],
        [-0.2884, -0.8878, -0.6728, -0.2046]])
>>> torch.mul(t1,10)
tensor([[ 4.9128, 15.1218, 11.4599,  3.4850]])
>>> t3=torch.randn(2,2)
>>> t4=torch.randn(2,2)
>>> torch.mul(t3,t4)
tensor([[ 0.0704, -0.4306],
        [ 0.8794,  0.3230]])
>>>
```

torch.mul(a, b)是矩阵 a 和 b 对应位相乘:

有三种情况:

- (1) 一个 tensor 乘一个常数
- (2) 两个 shape 完全一致的 tensor
- (3) 如果形状维度不一致, 可以触发 broadcast 机制的方法:

比如 `t1=torch.randn(1,4)`, `t2=torch.randn(4,1)`可以触发 broadcast 机制

什么是 broadcast 机制?

参考资料: <https://blog.csdn.net/fendyu/article/details/79955154>

Broadcast 总结起来三句话:

- (1) 如果两个数组在维度的数量上有差异, 那么维度较少的数组的形状就会被用 1 填充在它的前导(左)边。
- (2) 如果两个数组的形状在任何维度上都不匹配, 但等于 1, 那么在这个维度中, 形状为 1 的数组将被拉伸以匹配另一个形状。
- (3) 如果在任何维度上, 大小都不一致, 且两者都不等于 1, 就会出现错误。

问题 6: tensorboardx 可视化安装问题

注意事项: 在可视化部分需要安装 tensorboardx, 但是 tensorboardx 依赖于 tensorflow 的 tensorboard, 使用的时候先安装 tensorflow 和 tensorboard, 具体安装

步骤如下：

- (1) `pip install tensorflow==1.2`(tensorflow 版本太高可能会有问题)
- (2) `pip install tensorboard`
- (3) `pip install tensorboardx`

安装好之后 pytorch 就可以使用 tensorboardx 可视化了。

问题 7: `torch.all()` 函数

`torch.all()` 是判断两个张量各个对应位置的值是否相等，有一个不相等就是 0，全相等返回 1。

`torch.all()` 可以参考 `numpy.all()` 这个函数，两个意义是一样的。

问题 8: pytorch 中默认的网络权重初始化方法是什么?其内置初始化方法中对 relu 激活函数的所有网络中一定是默认权重初始化方法最佳吗?

`Conv2d` 继承自 `_ConvNd`，在 `_ConvNd` 中，可以看到默认参数就是进行初始化

```
class Conv2d(_ConvNd):
    """Applies a 2D convolution over an input signal
    planes.

    In the simplest case, the output value of the l
```

`conv2d` 的默认初始化方式：

```
def reset_parameters(self):
    n = self.in_channels
    init.kaiming_uniform_(self.weight, a=math.sqrt(5))
    if self.bias is not None:
        fan_in, _ = init._calculate_fan_in_and_fan_out(self.weight)
        bound = 1 / math.sqrt(fan_in)
        init.uniform_(self.bias, -bound, bound)
```

`BN` 继承 (`_BatchNorm`) ,默认初始化方式：

```
def reset_parameters(self):  
    self.reset_running_stats()  
    if self.affine:  
        init.uniform_(self.weight)  
        init.zeros_(self.bias)
```

问题 9: torch.no_grad() 和 eval() 的区别?

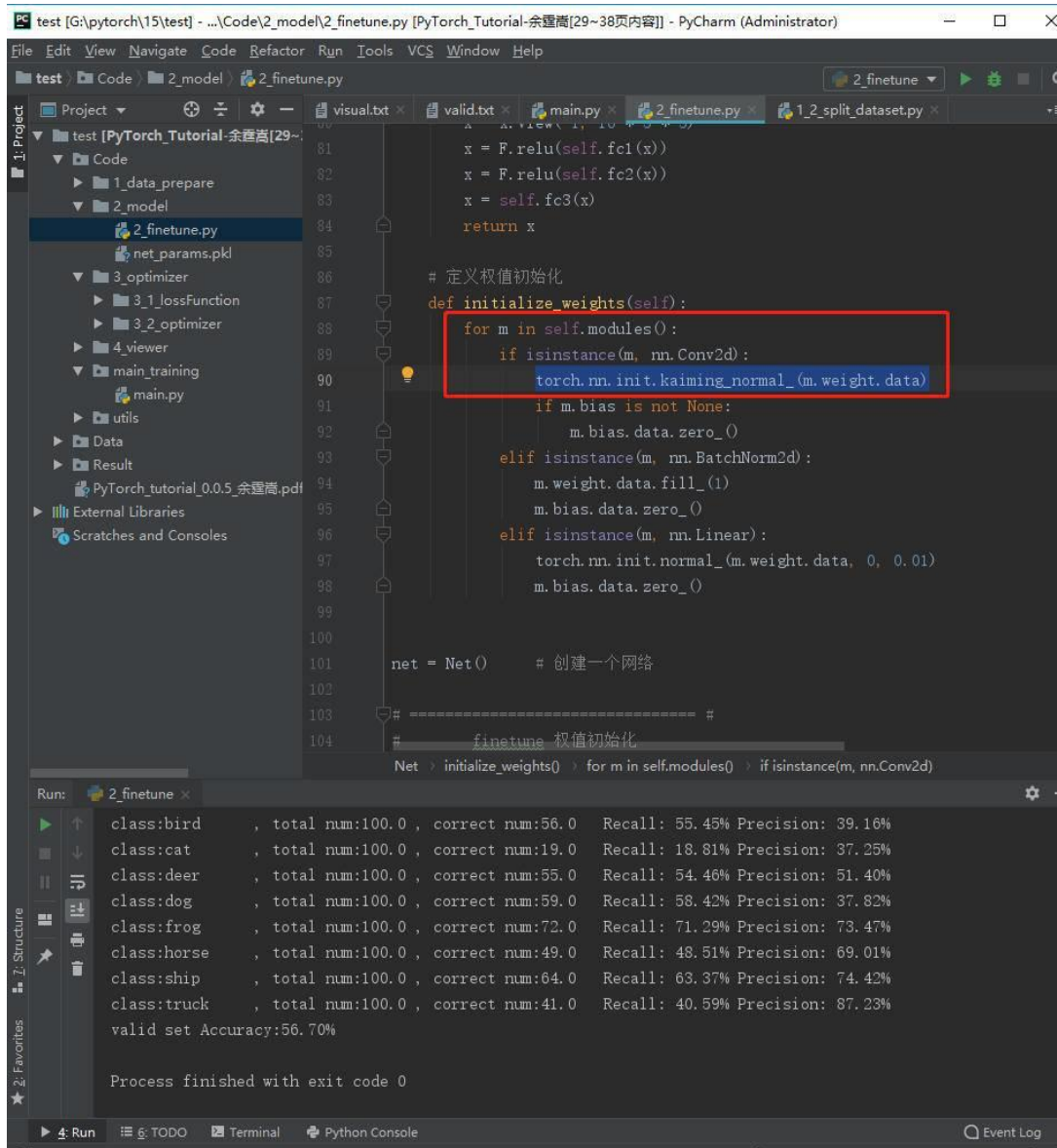
model.eval(): 所有的图层处于评估模式, 这样, batchnorm 或 dropout 图层将在 eval 模型而不是训练模式下工作。

torch.no_grad(): 影响 autograd 引擎并停用它, 将无法使用 backprop。

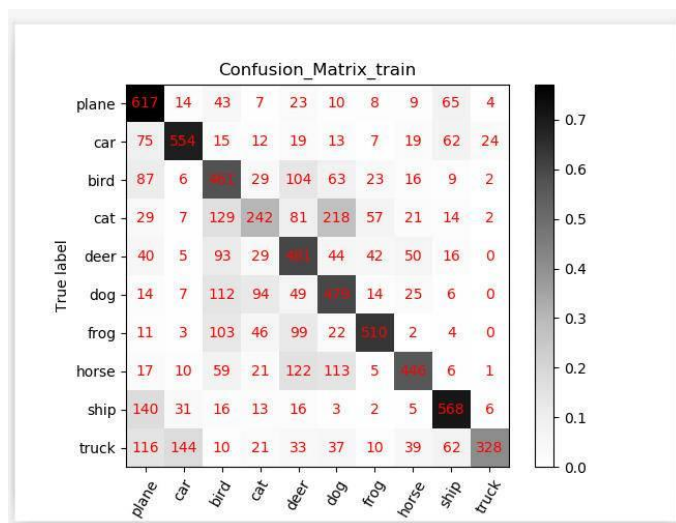
torch.no_grad()主要在测试的时候减少内存和现存的占用, model.eval()主要是因为 BN 和 dropout, 如果没有这两个, 可以不设置成 model.eval()

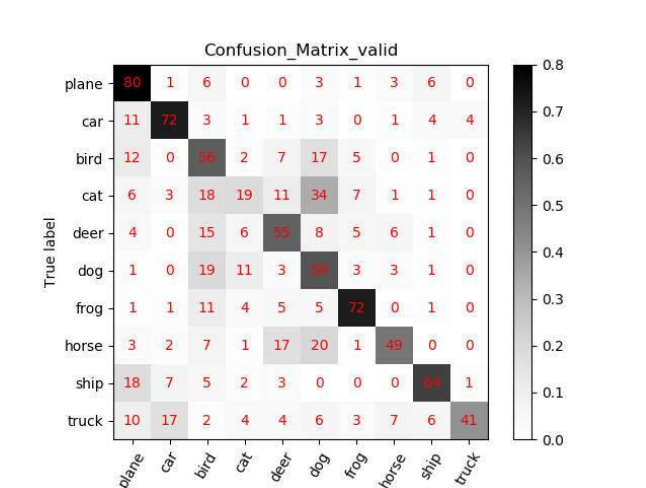
问题 10: 想问下 3.1 作业 2 在 finetune 的时候, 为什么用 xavier 和 kaiming 的正态分布初始化之后训练, 最后准确率都在 50%左右呢? 可是出来的混淆矩阵又是还可以的。

作业 2: 比较初始化方法和 finetune 方法收敛速度和精度




```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
test > Code > 2_model > 2_finetune.py
Project
  test [PyTorch_Tutorial-余程嵩[29-
    Code
      1_data_prepare
      2_model
        2_finetune.py
        net_params.pkl
      3_optimizer
Run: 2_finetune x
  Training: Epoch[010/010] Iteration[490/500] Loss: 1.2038 Acc:58.76%
  参数组1的学习率:0.001, 参数组2的学习率:0.01
  Training: Epoch[010/010] Iteration[500/500] Loss: 1.0525 Acc:58.94%
  参数组1的学习率:0.001, 参数组2的学习率:0.01
  Valid set Accuracy:59.40%
  Finished Training
  class:plane    , total num:800.0 , correct num:442.0 Recall: 55.18% Precision: 74.04%
  class:car      , total num:800.0 , correct num:692.0 Recall: 86.39% Precision: 59.50%
  class:bird     , total num:800.0 , correct num:358.0 Recall: 44.69% Precision: 56.74%
  class:cat      , total num:800.0 , correct num:329.0 Recall: 41.07% Precision: 42.84%
  class:deer     , total num:800.0 , correct num:346.0 Recall: 43.20% Precision: 60.28%
  class:dog      , total num:800.0 , correct num:375.0 Recall: 46.82% Precision: 58.78%
  class:frog     , total num:800.0 , correct num:650.0 Recall: 81.15% Precision: 58.09%
  class:horse    , total num:800.0 , correct num:575.0 Recall: 71.79% Precision: 58.67%
  class:ship     , total num:800.0 , correct num:557.0 Recall: 69.54% Precision: 69.36%
  class:truck    , total num:800.0 , correct num:508.0 Recall: 63.42% Precision: 68.93%
  train set Accuracy:60.40%
  class:plane    , total num:100.0 , correct num:58.0 Recall: 57.43% Precision: 69.88%
  class:car      , total num:100.0 , correct num:89.0 Recall: 88.12% Precision: 54.94%
  class:bird     , total num:100.0 , correct num:50.0 Recall: 49.50% Precision: 54.35%
  class:cat      , total num:100.0 , correct num:33.0 Recall: 32.67% Precision: 42.31%
  class:deer     , total num:100.0 , correct num:37.0 Recall: 36.63% Precision: 48.05%
  class:dog      , total num:100.0 , correct num:54.0 Recall: 53.47% Precision: 56.25%
  class:frog     , total num:100.0 , correct num:85.0 Recall: 84.16% Precision: 58.62%
  class:horse    , total num:100.0 , correct num:67.0 Recall: 66.34% Precision: 62.04%
  class:ship     , total num:100.0 , correct num:63.0 Recall: 62.38% Precision: 72.41%
  class:truck    , total num:100.0 , correct num:58.0 Recall: 57.43% Precision: 70.73%
  valid set Accuracy:59.40%
  Process finished with exit code 0
  Run TODO Terminal Python Console
```





解答：

(1) 因为本例子只是让大家了解方法流程，所以使用的最简单的 Lenet 网络，而 cifar10 是比较复杂的场景，简单网络模型不足以拟合复杂场景，所以可以看到训练数据集只有 50%

(2) 初始化方法，我们一般推荐默认初始化方法或者 Finetune 的方法，自己设计初始化方法往往达不到比较好的效果。

(3) 默认 conv 初始化方法继承_ConvNd:

```
def reset_parameters(self):
    n = self.in_channels
    init.kaiming_uniform_(self.weight, a=math.sqrt(5))
    if self.bias is not None:
        fan_in, _ = init._calculate_fan_in_and_fan_out(self.weight)
        bound = 1 / math.sqrt(fan_in)
        init.uniform_(self.bias, -bound, bound)
```

其他默认初始化方法可以自行查看。

(4) 测试效果 50%~60%左右，所描述的混淆矩阵也是 50%~60%左右。

问题 11:请问两块 gpu 使用量不同怎么解决？

cmd 命令提示符

C:\Users\Deep Learning>nvidia-smi

Thu Jun 20 21:18:17 2019

NVIDIA-SMI 425.25			Driver Version: 425.25			CUDA Version: 10.1		
GPU	Name	TCC/WDDM	Bus-Id	Disp. A	Volatile	Uncorr.	ECC	
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util	Compute	M.	
0	GeForce RTX 208...	WDDM	00000000:05:00.0	On			N/A	
85%	85C	P2	137W / 260W	9907MiB / 11264MiB	94%		Default	
1	GeForce RTX 208...	WDDM	00000000:09:00.0	Off			N/A	
41%	50C	P2	100W / 260W	9907MiB / 11264MiB	16%		Default	

Processes:				CPU Memory Usage
GPU	PID	Type	Process name	
0	1292	C+G	Insufficient Permissions	N/A
0	1416	C+G	...x64_8wekyb3d8bbwe\Microsoft.Photos.exe	N/A
0	1712	C+G	E:\software\WeChat\WeChatWeb.exe	N/A
0	6148	C+G	...46.60.0_x64_kzf8qxf38zg5c\SkypeApp.exe	N/A
0	6248	C+G	C:\Windows\explorer.exe	N/A
0	6296	C+G	...6)\Google\Chrome\Application\chrome.exe	N/A
0	7104	C+G	...dows.Cortana_cw5nlh2txyewy\SearchUI.exe	N/A
0	7304	C+G	..._cw5nlh2txyewy\PeopleExperienceHost.exe	N/A
0	7356	C+G	...t_cw5nlh2txyewy\ShellExperienceHost.exe	N/A
0	7620	C+G	...les (x86)\CorsairLink4\CorsairLink4.exe	N/A
0	7784	C	E:\Anaconda3\envs\dl\python.exe	N/A

推荐这篇博客，讲的比较详细：

<https://www.cnblogs.com/ranjiewen/p/10113532.html>

问题 12: 为什么 torch.nn.init 的初始化函数都在后面加了下划线?有什么含义吗?

```
torch.nn.init.normal_(tensor, mean=0.0, std=1.0)
```

in-place operation 在 pytorch 中是指改变一个 tensor 的值的时候，不经过复制操作，而是直接在原来的内存上改变它的值。可以把它成为原地操作符。在 pytorch 中经常加后缀“_”来代表原地 in-place operation。normal 和 normal_ 功能一样。

问题 13: 有些 py 文件里的函数并没有实际定义，全都是注释，这是什么意思？

```
py × 2_finetune.py × builtins.py ×
    Calling help(thing) prints help for the python object 'thing'.
    """
    pass

* def hex(*args, **kwargs): # real signature unknown; NOTE: unreliably restored from __doc__
    """
    Return the hexadecimal representation of an integer.

    >>> hex(12648430)
    '0xc0ffee'
    """
    pass

* def id(*args, **kwargs): # real signature unknown
    """
    Return the identity of an object.

    This is guaranteed to be unique among simultaneously existing objects.
    (CPython uses the object's memory address.)
    """
    pass

* def input(*args, **kwargs): # real signature unknown
    """
    Read a string from standard input. The trailing newline is stripped.

    The prompt string, if given, is printed to standard output without a
    trailing newline before reading input.

    If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError.
    On *nix systems, readline is used if available.
    """
    pass
```

这些只是提供一些接口，python 封装了 c、c++，具体实现是 c/c++ 中。

问题 14: 请问课程中用的是什么时候的官方文档？volatile
在新版本中应该怎样被替换掉呢？

[illegible]

(1) 目前使用的 0.4 版本，因为 1.0 版本没有中文文档，考虑大家阅读英文文档可能不太方便，所以使用的 0.4 版本 (2) 使用 `with torch.no_grad` 替代了 `volatile`

问题 15: 3.2 作业在同一个模型上比较不同 loss 函数精度的问题

在余大神给的那个分类模型上，把 loss 从交叉熵改成 MSE 会报错，这两个 loss 函数可以比较吗？是不是 MSE 就不能用在余大神那个分类模型中啊？

解答：MSE 均方误差函数一般评价相似度或者距离的损失函数，而在分类模型中我们常用交叉熵损失函数。

问题 16: add 函数和 add () 函数的调用问题

pytorch 官网中示例代码:

```
# but:
>>> x=torch.empty(1,3,1)
>>> y=torch.empty(3,1,7)
>>> (x.add_(y)).size()
RuntimeError: The expanded size of the tensor (1) must match the existing size (7) at non-
singleton dimension 2.
```

改成下图依然报错求问原因？

```
1 import torch
2 ...
5 x = torch.empty(1,3,7)
6 y = torch.empty(3,1,7)
7 print(x.add_(y)).size()
8
test (2)
D:\MM_work\anaconda_bao\python.exe D:/MM_work/python_test/pycharm-test/pyTorch/test.py
Traceback (most recent call last):
  File "D:/MM_work/python_test/pycharm-test/pyTorch/test.py", line 9, in <module>
    print((x.add_(y)).size())
RuntimeError: output with shape [1, 3, 7] doesn't match the broadcast shape [3, 3, 7]

Process finished with exit code 1
```

解答 :需要了解 add 函数和 add_的区别 ,也就是 in_place 操作(见[问题 12](#)) :
in-place operation 在 pytorch 中是指改变一个 tensor 的值的的时候 ,不经过复制操作 ,而是直接在原来的内存上改变它的值。可以把它成为原地操作符。在 pytorch 中经常加后缀 “_” 来代表原地 in-place operation。

问题 17: 关于 nn.Linear 的参数设置问题

```
self.fc1 = nn.Linear(16 * 5 * 5, 120)
```

```
self.fc2 = nn.Linear(120, 84)
```

```
self.fc3 = nn.Linear(84, 10)
```

这里为啥是 120、84, 不能改成其他的?

解答 :这些都是超参数 ,可以自己更改 ,只是 Lenet 作者当时在做 mnist 数据分类时 ,在参数为 120,84 时效果比较好 (2) 另外一个原因不能太大 ,比如 (1200,840) 时导致参数太多 ,过拟合 ,而参数太少 ,比如 (12,8) 又会导致欠拟合 ,所以这些参数的设置一般都是根据经验设定。

问题 18: pytorch 要怎么实现模型融合?

解答 :问题中的 “模型融合” 是指多卷积核网络模型融合还是类似 FPN 前后的

网络融合？如果是第一个情况参考 inception 网络 ,如果第二种参考 darknet53

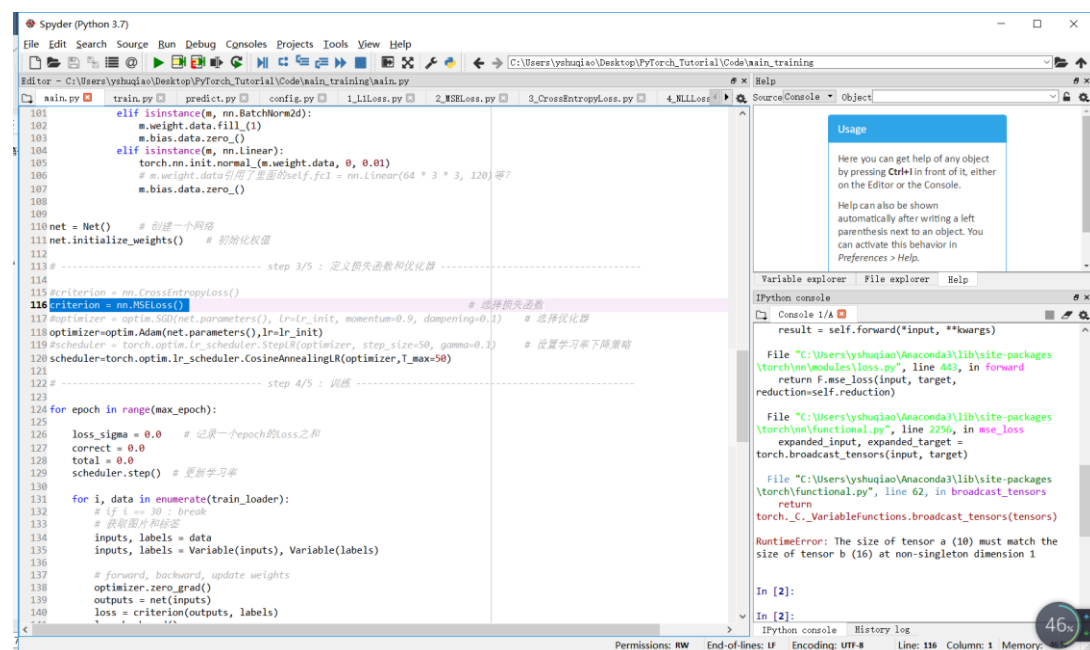
网络模型

问题 19: pytorch 预训练模型怎么得到的？如何从 0 开始训练网络？

解答：（1）预训练模型网络下载得到，也可以自己训练一个大数据集作为预训练模型来训练自己的小数据。（2）不使用预训练模型就是从 0 训练。

问题 20: 关于 nn.MSELoss 的参数调整问题

试着把 nn.CrossEntropyLoss 改为 nn.MSELoss，运行出错，看提示，大概是这个损失函数的输入和标签的 tensor 不相符，可能需要调整数据导入时的 batch_size 和标准化的参数(transform)?但还不会调。



```
101 elif isinstance(m, nn.BatchNorm2d):
102     m.weight.data.fill_(1)
103     m.bias.data.zero_()
104 elif isinstance(m, nn.Linear):
105     torch.nn.init.normal_(m.weight.data, 0, 0.01)
106     # m.weight.data 仿照了上面的self.fc1 = nn.Linear(64 * 3 * 3, 120) 等?
107     m.bias.data.zero_()
108
109
110 net = Net() # 构造一个网络
111 net.initialize_weights() # 初始化权重
112
113 # ----- step 3/5 : 定义损失函数和优化器 -----
114 criterion = nn.CrossEntropyLoss() # 选择损失函数
115 optimizer = optim.Adam(net.parameters(), lr=lr_init, momentum=0.9, dampening=0.1) # 选择优化器
116 scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.1) # 设置学习率下降策略
117 scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=50)
118
119 # ----- step 4/5 : 训练 -----
120
121 for epoch in range(max_epoch):
122     loss_sum = 0.0 # 记录一个epoch的loss之和
123     correct = 0.0
124     total = 0.0
125     scheduler.step() # 更新学习率
126
127     for i, data in enumerate(train_loader):
128         # 获取图片和标签
129         inputs, labels = data
130         inputs, labels = Variable(inputs), Variable(labels)
131
132         # forward, backward, update weights
133         optimizer.zero_grad()
134         outputs = net(inputs)
135         loss = criterion(outputs, labels)
```

解答：前两天有同学问这个问题，请参考前两天的。

问题 21: SGD 优化器调参经验

比较 SGD 优化器与原来的 Adam 优化器，差好多，可能要调整参数才能得

到比较合适的结果，但还没会调，比如 lr(初始学习率)可以不设置为原来固定的 0.001，让它默认缺省值会好点？（调参方法大家总是有疑惑，请老师传授经验）

解答：学习率属于超参数的范畴，本身设置就是依靠经验，我一般常用 adam+cos 的方法调整学习率。

问题 22: fine tuning 代码的一些问题

余霆嵩老师 code 的第二个，fine tuning 代码，他把模型的全连接层 3 给移除了，并且单独对该层设置学效率及优化方法，请问为什么要这么做？以及对单独一层设置学效率和优化方法（SGD）有什么用？学习率以及优化方法的设置，是只需要对单独一层设置即可了吗？

解答：（1）单独设置学习率以及优化算法的目的是给大家举例说明可以这样设置每层的学习率和优化算法,在以后自己应有中清楚模型初始化方法除了默认和 finetune,每层还可以单独设置。

（2）单独对一层设置学习率和优化算法的作用是只针对这一层使用特定的学习率和优化算法,其他层的学习率和优化算法和指定层不同,比如我们在 finetune 时最后一层全连接层需要单独设置,因为只学习最后一层的模型参数,其他层的参数不变。

问题 23: 为什么 trainset 做 augmentation 之后，原始的 trainset 不会一起扔到网络里训练？

解答：因为源码里 transform 对应的是所有对象在 dataset 的 __get__ 里，就像你写 Totensor 一样，难道还希望一部分不变？想要保留的话，写两个不同的 transorm 结果在一加就行了。

问题 24: 张量的 split 问题

```
print("torch.normal: ", torch.normal(mean= torch.arange(1,6) ) )
```

```
RuntimeError                                Traceback (most recent call last)
<ipython-input-263-7ffa1157689d> in <module>()
--> 1 print("torch.normal: ", torch.normal(mean= torch.arange(1,6) ) )

RuntimeError: _th_normal not supported on CPUType for Long
```

```
print("torch.normal: ", torch.normal(mean= torch.arange(1.,6.) ) )
```

如何把这个分成 tensor([1]),tensor([2]),tensor([3])?

解答: 首先明白 torch.split 是拆分张量的 API: torch.split(tensor, split_size, dim=0), 需要确保将输入张量分割成相等形状的 chunks (如果可分)。如果沿指定维的张量形状大小不能被 split_size 整分, 则最后一个分块会小于其它分块。

使用举例:

```
x = torch.randn(3, 10, 6)
a, b, c = x.split(1, 0) # 在 0 维进行间隔维 1 的拆分
a.size(), b.size(), c.size()
(torch.Size([1, 10, 6]), torch.Size([1, 10, 6]), torch.Size([1, 10, 6]))
```

推荐阅读链接:

https://blog.csdn.net/weixin_44613063/article/details/895768