Pytorch调试工具使用手册

|  |
| --- |
| 详细说明：支持在用户python代码中用pdb.set\_trace()来跟踪代码 验收标准：功能验收 TR3交付件：Pytorch调试工具使用手册 |
| 详细说明：用户指定某一层，支持dump这一层的输入（输出）数据到本地 验收标准：功能验收 TR3交付件：Pytorch调试工具使用手册 |
| 详细说明：用户指定某一层，支持dunp这一层的权重数据到本地 验收标准：功能验收 TR3交付件：Pytorch调试工具使用手册 |
| 详细说明：用户指定某一层，支持dunp这一层的前向、后向等计算信息 验收标准：功能验收 TR3交付件：Pytorch调试工具使用手册 |
| 详细说明：支持在终端打印训练相关的信息 验收标准：功能验收 TR3交付件：Pytorch调试工具使用手册 |

1. pdb 调试

可以在python的代码中import pdb:

def train():

current\_batches = 0

de2idx, idx2de = load\_de\_vocab()

en2idx, idx2en = load\_en\_vocab()

enc\_voc = len(de2idx)

dec\_voc = len(en2idx)

# Load data

X, Y = load\_train\_data()

# calc total batch count

num\_batch = len(X) // hp.batch\_size

import pdb

pdb.set\_trace()

model = AttModel(hp, enc\_voc, dec\_voc)

在上面的代码中，可以在想调试代码的上一行加上pdb.set\_trace()

然后运行python train.py 进入调试模式，

n： 下一个， next单步执行

enter：重复上次命令

q: 退出

p <变量>: 打印变量

1. dump 某一层的输入（输出）数据到本地

在python脚本中，可以用numpy.savetxt来保存输入或者输出数据。

输入的dump :

np.savetxt("cpu\_softmax\_input.txt", x.detach().numpy().reshape(-1, 1))

输出的dump：

np.savetxt("cpu\_softmax\_output.txt", out\_cpu.detach().numpy().reshape(-1, 1))

1. 打印权值

index = 0

for name, param in model.named\_parameters():

index = index + 1

name = "tmp\_" + str(index) + "\_" + name

np.savetxt(name, param.detach().numpy().reshape(-1, 1))

1. Dump 前向后向的值

可以用pytorch自带的hook机制打印前向和后向的值。

前向使用register\_forward\_hook

反向使用register\_backward\_hook

import torch

import torch.nn as nn

import torch.nn.functional as F

class MNISTConvNet(nn.Module):

def \_\_init\_\_(self):

super(MNISTConvNet, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(1, 10, 5)

self.pool1 = nn.MaxPool2d(2, 2)

self.conv2 = nn.Conv2d(10, 20, 5)

self.pool2 = nn.MaxPool2d(2, 2)

self.fc1 = nn.Linear(320, 50)

self.fc2 = nn.Linear(50, 10)

def forward(self, input):

x = self.pool1(F.relu(self.conv1(input)))

x = self.pool2(F.relu(self.conv2(x)))

x = x.view(x.size(0), -1)

x = F.relu(self.fc1(x))

x = F.relu(self.fc2(x))

return x

def printnorm(self, input, output):

# input is a tuple of packed inputs

# output is a Tensor. output.data is the Tensor we are interested

print('Inside ' + self.\_\_class\_\_.\_\_name\_\_ + ' forward')

print('')

print('input: ', type(input))

print('input[0]: ', type(input[0]))

print('output: ', type(output))

print('')

print('input size:', input[0].size())

print('output size:', output.data.size())

print('output norm:', output.data.norm())

def printgradnorm(self, grad\_input, grad\_output):

print('Inside ' + self.\_\_class\_\_.\_\_name\_\_ + ' backward')

print('Inside class:' + self.\_\_class\_\_.\_\_name\_\_)

print('')

print('grad\_input: ', type(grad\_input))

print('grad\_input[0]: ', type(grad\_input[0]))

print('grad\_output: ', type(grad\_output))

print('grad\_output[0]: ', type(grad\_output[0]))

print('')

print('grad\_input size:', grad\_input[0].size())

print('grad\_output size:', grad\_output[0].size())

print('grad\_input norm:', grad\_input[0].norm())

net = MNISTConvNet()

input = torch.randn(1, 1, 28, 28)

net.conv2.register\_forward\_hook(printnorm)

out = net(input)

target = torch.tensor([3], dtype=torch.long)

loss\_fn = nn.CrossEntropyLoss()

net.conv2.register\_backward\_hook(printgradnorm)

out = net(input)

err = loss\_fn(out, target)

err.backward()

1. 打印训练相关信息

打印loss，accuracy， learning\_rate，iterations 的值：

print(“loss : “, loss.iterm(), “ acc : “, acc.item(), “learning\_rate : “, lr, “iterations : “, iterations)