

实验报告——调试及性能分析，元编程演示实验，PyTorch 编程

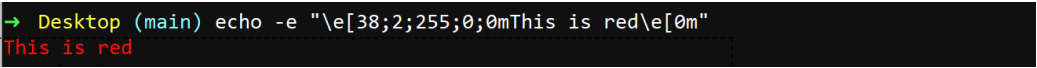
姓名：徐亚齐 学号：23020007136

2024 年 9 月 13 日

1 实验实例

1.1 在终端中打印不同颜色的字

可以使用 shell 改变输出结果的颜色

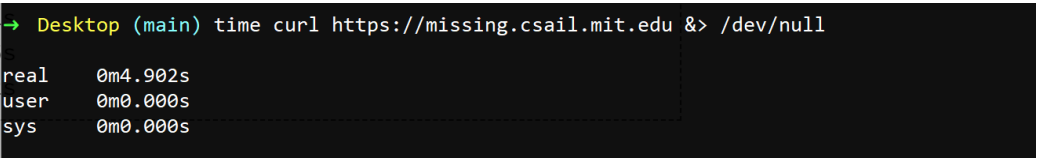


```
→ Desktop (main) echo -e "\e[38;2;255;0;0mThis is red\e[0m"
This is red
```

图 1: 实例

1.2 试着执行一个用于发起 HTTP 请求的命令，并输出此次请求花费的时间信息

发起请求应该用命令 curl，为了统计时间，需要在前面加上 time 前缀，这样就可以输出程序执行花费的真实时间，用户时间和系统时间。

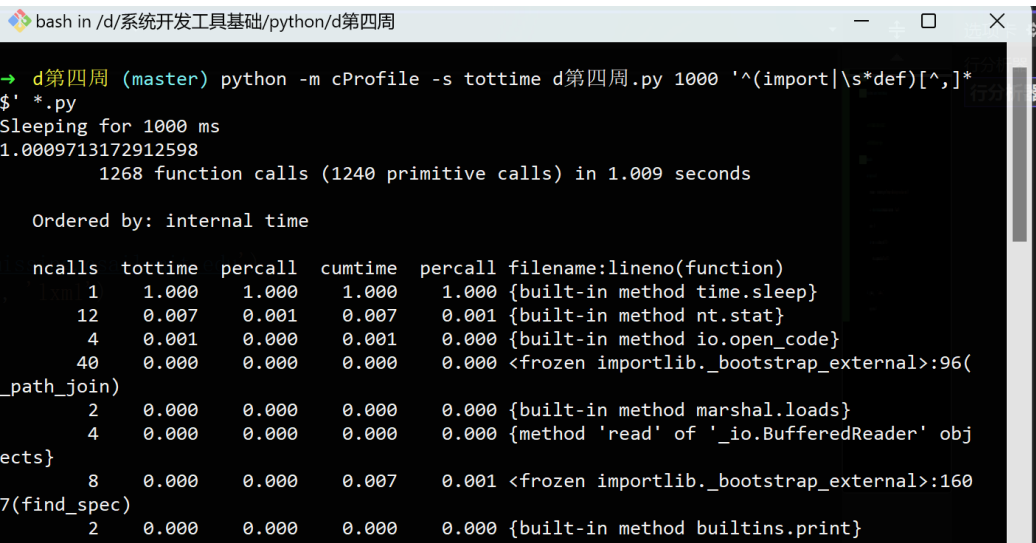


```
→ Desktop (main) time curl https://missing.csail.mit.edu &> /dev/null
real    0m4.902s
user    0m0.000s
sys     0m0.000s
```

图 2: 实例

1.3 用 python 写一部分代码，并尝试用 cProfile 模块来分析每次函数调用所消耗的时间

Python 的 cProfile 分析器显示的是每次函数调用的时间，用它来分析程序运行消耗的时间需要在命令中写出 python 的文件名。如果代码里面使用了第三方的函数库，它可能会看上去快到反直觉。



```
bash in /d/系统开发工具基础/python/d第四周
→ d第四周 (master) python -m cProfile -s tottime d第四周.py 1000 '^(import|\s*def)[^,]*$' *.py
Sleeping for 1000 ms
1.0009713172912598
    1268 function calls (1240 primitive calls) in 1.009 seconds

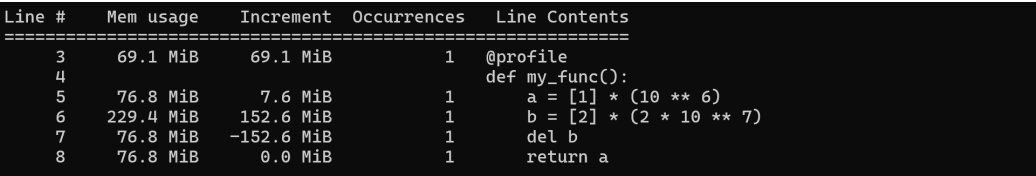
Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    1    1.000    1.000    1.000    1.000 {built-in method time.sleep}
    12    0.007    0.001    0.007    0.001 {built-in method nt.stat}
     4    0.001    0.000    0.001    0.000 {built-in method io.open_code}
    40    0.000    0.000    0.000    0.000 <frozen importlib._bootstrap_external>:96(_path_join)
     2    0.000    0.000    0.000    0.000 {built-in method marshal.loads}
     4    0.000    0.000    0.000    0.000 {method 'read' of '_io.BufferedReader' objects}
     8    0.000    0.000    0.007    0.001 <frozen importlib._bootstrap_external>:1607(find_spec)
     2    0.000    0.000    0.000    0.000 {built-in method builtins.print}
```

图 3: 实例

1.4 用 python 写一段代码，并尝试使用内存分析器对其进行分析

使用内存分析器 memory-profiler 可以用的命令为 python -m memory_profiler example.py，以此来输出代码内存的相关信息。

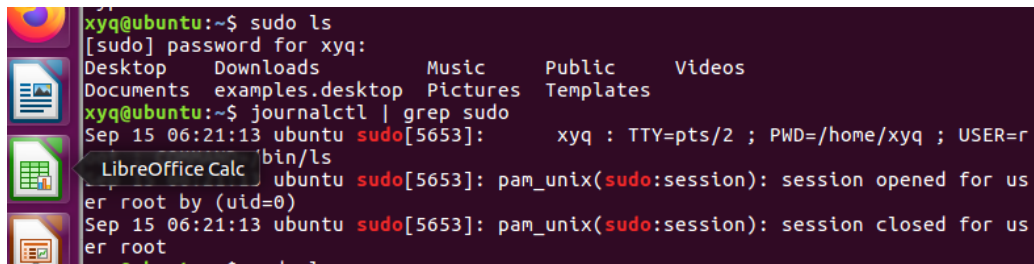


Line #	Mem usage	Increment	Occurrences	Line Contents
3	69.1 MiB	69.1 MiB	1	@profile
4				def my_func():
5	76.8 MiB	7.6 MiB	1	a = [1] * (10 ** 6)
6	229.4 MiB	152.6 MiB	1	b = [2] * (2 * 10 ** 7)
7	76.8 MiB	-152.6 MiB	1	del b
8	76.8 MiB	0.0 MiB	1	return a

图 4: 实例

1.5 使用 Linux 上的 journalctl 命令来获取最近一天中超级用户的登录信息及其所执行的指令。

在 Linux 系统中，可以先执行像 `sudo ls` 一样的无害的命令，再用 `journalctl | grep sudo` 命令来获取登录信息及其所执行的指令。



```
xyq@ubuntu:~$ sudo ls
[sudo] password for xyq:
Desktop  Downloads      Music          Public         Videos
Documents examples.desktop Pictures        Templates
xyq@ubuntu:~$ journalctl | grep sudo
Sep 15 06:21:13 ubuntu sudo[5653]: xyq : TTY=pts/2 ; PWD=/home/xyq ; USER=r
er root by (uid=0)
Sep 15 06:21:13 ubuntu sudo[5653]: pam_unix(sudo:session): session opened for us
er root
Sep 15 06:21:13 ubuntu sudo[5653]: pam_unix(sudo:session): session closed for us
er root
xyq@ubuntu:~$ sudo ls
```

图 5: 实例

1.6 有一些用于计算斐波那契数列 Python 代码, 它为计算每个数字都定义了一个函数, 将代码拷贝到文件中使其变为一个可执行的程序。首先安装 pycallgraph 和 graphviz。并使用 `pycallgraph graphviz - ./fib.py` 来执行代码并查看 `pycallgraph.png` 这个文件。

首先执行 `pip install "setuptools<58.0.0"` 和 `pip install pycallgraph` 命令，用来安装 `pycallgraph` 和 `graphviz`，安装完成后，将计算斐波那契数列的代码写在一个文件中，然后在终端执行 `pycallgraph graphviz - ./fib.py` 命令，执行后会自动生成 `pycallgraph.png` 文件。执行命令后，输出的结果为 34。

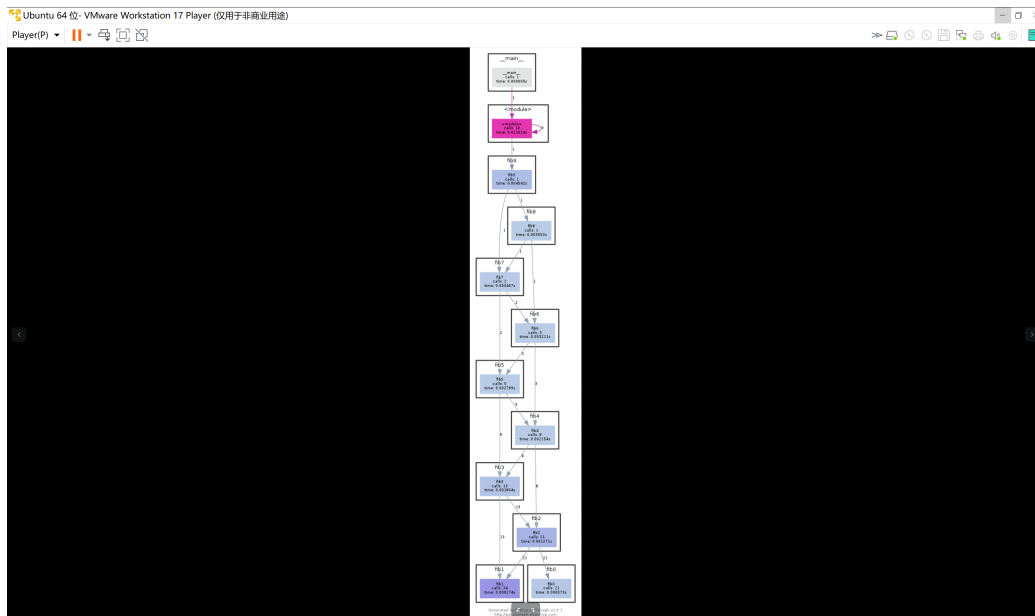


图 6: pycallgraph.png 文件查看

1.7 curl ipinfo.io 命令或执行 HTTP 请求并获取关于您 IP 的信息，并抓取 curl 发起的请求和收到的回复报文。

用 curl ipinfo.io 可以获得 IP 信息。可以使用 curl www.baidu.com 命令，请求百度的首页并过滤除 HTTP 之外的其他报文。

```
→ d第四周 (master) curl ipinfo.io
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100  287  100  287    0    0   400      0 --:--:-- --:--:-- --:--:--  404{
  "ip": "223.104.194.250",
  "city": "Shanghai",
  "region": "Shanghai",
  "country": "CN",
  "loc": "31.2222,121.4581",
  "org": "AS24444 Shandong Mobile Communication Company Limited",
  "postal": "200000",
  "timezone": "Asia/Shanghai",
  "readme": "https://ipinfo.io/missingauth"
}
```

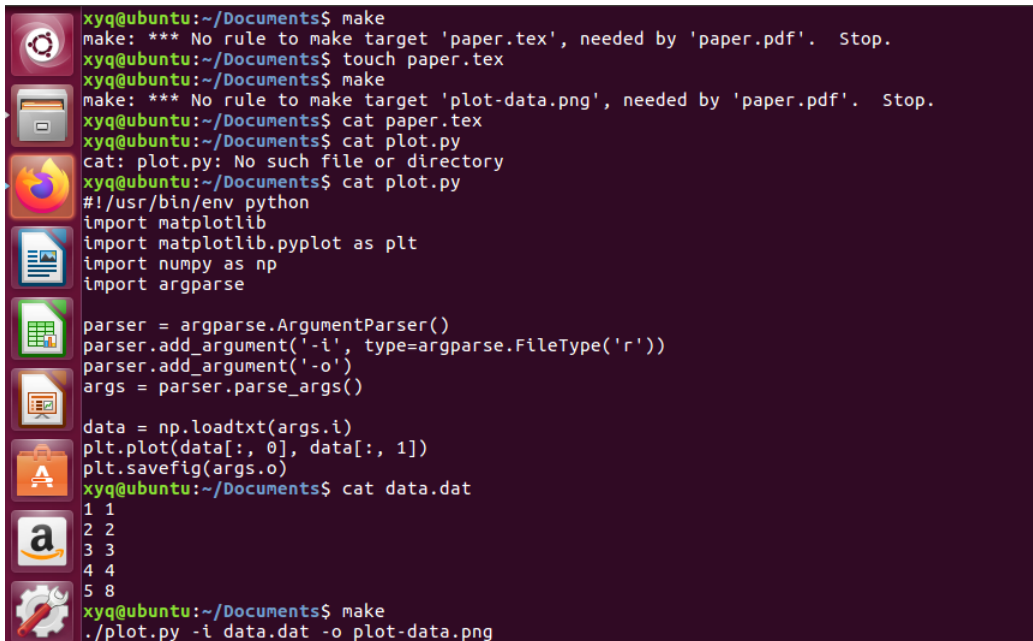
图 7: IP 信息

```
bash in ~/Desktop
→ Desktop (main) curl www.baidu.com
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total      Spent    Left     Speed
100 2381  100 2381    0     0 10563      0 --:--:-- --:--:-- --:--:-- 10822<!DOCTYPE
html>
<!--STATUS OK--><html> <head><meta http-equiv=content-type content=text/html;charset=utf
-8><meta http-equiv=X-UA-Compatible content=IE=Edge><meta content=always name=referrer><
link rel=stylesheet type=text/css href=http://s1.bdstatic.com/r/www/cache/bdorz/baidu.mi
n.css><title>百度一下，你就知道</title></head> <body link=#0000cc> <div id=wrapper> <div
id=head> <div class=head_wrapper> <div class=s_form> <div class=s_form_wrapper> <div id
=lg> <img hidefocus=true src=//www.baidu.com/img/bd_logo1.png width=270 height=129> </di
v> <form id=form name=f action=//www.baidu.com/s class=fm> <input type=hidden name=bdorz
_come value=1> <input type=hidden name=ie value=utf-8> <input type=hidden name=f value=8
> <input type=hidden name=rsv_bp value=1> <input type=hidden name=rsv_idx value=1> <inpu
t type=hidden name=tn value=baidu><span class="bg s_ipt_wr"><input id=kw name=wd class=s
_ipt value maxlength=255 autocomplete=off autofocus></span><span class="bg s_btn_wr"><in
put type=submit id=su value=百度一下 class="bg s_btn"></span> </form> </div> </div> <div
id=u1> <a href=http://news.baidu.com name=tj_trnews class=mnav>新闻</a> <a href=http://
www.hao123.com name=tj_trhao123 class=mnav>hao123</a> <a href=http://map.baidu.com name=
```

图 8: 请求百度首页并过滤报文

1.8 尝试了解并使用 make 构建系统

首先在目录下编写名为 Makefile 的文件，将所有构建目标、相关依赖和规则都需要在该文件中定义。此时运行 make 不会成功，因为它需要 paper.tex 文件，用 touch 创建好该文件后，再次执行 make 命令，依旧不会成功，因为此时源文件 data.dat 并不存在。当创建好所有需要的文件后，再次运行 make，就能成功运行并生成 PDF。



```
xyq@ubuntu:~/Documents$ make
make: *** No rule to make target 'paper.tex', needed by 'paper.pdf'. Stop.
xyq@ubuntu:~/Documents$ touch paper.tex
xyq@ubuntu:~/Documents$ make
make: *** No rule to make target 'plot-data.png', needed by 'paper.pdf'. Stop.
xyq@ubuntu:~/Documents$ cat paper.tex
xyq@ubuntu:~/Documents$ cat plot.py
cat: plot.py: No such file or directory
xyq@ubuntu:~/Documents$ cat plot.py
#!/usr/bin/env python
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('-i', type=argparse.FileType('r'))
parser.add_argument('-o')
args = parser.parse_args()

data = np.loadtxt(args.i)
plt.plot(data[:, 0], data[:, 1])
plt.savefig(args.o)
xyq@ubuntu:~/Documents$ cat data.dat
1 1
2 2
3 3
4 4
5 8
xyq@ubuntu:~/Documents$ make
./plot.py -i data.dat -o plot-data.png
```

图 9: 实例

1.9 在 Linux 系统中，查看正在运行的所有守护进程

Linux 中的 `systemd` 是最常用的配置和运行守护进程的方法。想要看到正在运行的所有守护进程，只需要运行 `systemctl status` 命令即可。

```
xyq@ubuntu: ~/Documents
xyq@ubuntu:~/Documents$ systemctl status
● ubuntu
  State: running
  Jobs: 0 queued
  Failed: 0 units
  Since: Sun 2024-09-15 05:25:50 PDT; 13h ago
  CGroup: /
    └─user.slice
      └─user-1000.slice
        └─user@1000.service
          └─init.scope
            └─4374 /lib/systemd/systemd --user
              └─4375 (sd-pam)
                └─session-c2.scope
                  └─1257 lightdm --session-child 12 19
                    └─4411 /usr/bin/gnome-keyring-daemon --daemonize --login
                      └─4420 /sbin/upstart --user
                        └─4500 upstart-udev-bridge --daemon --user
                          └─4510 dbus-daemon --fork --session --address=unix:abstract=/tmp/dbus-
                            └─4523 /usr/lib/x86_64-linux-gnu/hud/window-stack-bridge
                              └─4561 upstart-dbus-bridge --daemon --system --user --bus-name system
                                └─4570 upstart-dbus-bridge --daemon --session --user --bus-name sessio
                                  └─4572 upstart-file-bridge --daemon --user
                                    └─4573 /usr/lib/x86_64-linux-gnu/bamf/bamfdaemon
                                      └─4575 /usr/bin/ibus-daemon --daemonize --xim --address unix:tmpdir=/t
                                        └─4585 /usr/lib/gvfs/gvfsd
                                          └─4592 /usr/lib/gvfs/gvfsd-fuse /run/user/1000/gvfs -f -o big_writes
                                            └─4597 /usr/lib/dconf/dconf-service
                                              └─4606 /usr/lib/at-spi2-core/at-spi-bus-launcher
                                                └─4620 /usr/bin/dbus-daemon --config-file=/etc/at-spi2/accessibility.c
                                                  └─4623 /usr/lib/at-spi2-core/at-spi2-registryd --use-gnome-session
                                                    └─4639 /usr/lib/ibus/ibus-dconf
                                                      └─4640 /usr/lib/ibus/ibus-ui-gtk3
                                                        └─4642 /usr/lib/ibus/ibus-x11 --kill-daemon
                                                          └─4668 /usr/lib/ibus/ibus-engine-simple
                                                            └─4697 /usr/lib/x86_64-linux-gnu/hud/hud-service
                                                              └─4699 /usr/lib/unity-settings-daemon/unity-settings-daemon
```

图 10: 实例

1.10 用 markdown 绘制一个简单的表格

markdown 表格每一列之间的内容用 | 隔开，在第二行的语法中可以通过: 来设置左对齐和右对齐。

```
1 |
2 |
3 | header 1 | header 2 | header 3 |
4 | :--- | :-----: | :-----: |
5 | cell 1 | cell 2 | cell 3 |
6 | cell 4 | cell 5 | cell 6 |
7 | cell 7 | cell 8 | cell 9 |
8 |
9 |
```

header 1	header 2	header 3
cell 1	cell 2	cell 3
cell 4	cell 5	cell 6
cell 7	cell 8	cell 9

图 11: 实例

1.11 markdown 的标题分多少级，尝试使用不同等级的标题

Markdown 文档中最多可设置 6 级标题，标题的设置需要用到 #，一级标题为 #，二级标题为 ##，以此类推。



图 12: 实例

1.12 尝试用 markdown 编辑文章内容，两行之间用分割线分开

Markdown 中 ** 是加粗，* 是斜体，可以在一行中用三个及以上的星号、减号、等于号、底线来建立分隔线，行内不能有除空格外的其他东西。



图 13: 实例

1.13 创建一个未初始化的矩阵和一个随机初始化的矩阵

没有初始化的矩阵用 `torch.empty()` 创建，随机初始化的矩阵用 `torch.rand()` 创建，创建时都可以指出行和列的数量。最后用 `print` 打印输出结果。


```
from __future__ import print_function
import torch

x = torch.empty(5, 3)
print(x)

# 创建一个随机初始化的 5*3 矩阵
rand_x = torch.rand(5, 3)
print(rand_x)
```

```
tensor([[1.2292e+04, 2.1468e-42, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 0.0000e+00]])
tensor([[0.2175, 0.7472, 0.3780],
        [0.8236, 0.4882, 0.3118],
        [0.4449, 0.4198, 0.3019],
        [0.3255, 0.1926, 0.2585],
        [0.3282, 0.5379, 0.7327]])
Press any key to continue . . .
```

图 14: 实例

1.14 创建一个数值全为 1 的矩阵，让它与一个随机初始化的矩阵相加，尝试采用不同的求和方法

先用 `torch.ones` 创建一个数值全为 1 的矩阵，再用 `torch.rand()` 创建一个随机初始化的矩阵。两个矩阵相加可以直接用加号；也可以新声明一个 `tensor` 变量保存用 `add` 加法操作的结果；还可以用 `add` 直接修改变量。最后用 `print` 输出每种方法的结果。

```
from __future__ import print_function
import torch

tensor3 = torch.ones(5, 3, dtype=torch.float)
print(tensor3)

tensor4 = torch.rand(5, 3)
print('tensor3 + tensor4= ', tensor3 + tensor4)

result = torch.empty(5, 3)
torch.add(tensor3, tensor4, out=result)
print('add result= ', result)

tensor3.add_(tensor4)
print('tensor3= ', tensor3)
```

```
tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]])
tensor3 + tensor4= tensor([[1.3127, 1.1263, 1.4634],
        [1.0282, 1.8656, 1.0846],
        [1.6852, 1.4324, 1.3756],
        [1.9992, 1.7216, 1.5859],
        [1.6413, 1.6382, 1.5626]])
add result= tensor([[1.3127, 1.1263, 1.4634],
        [1.0282, 1.8656, 1.0846],
        [1.6852, 1.4324, 1.3756],
        [1.9992, 1.7216, 1.5859],
        [1.6413, 1.6382, 1.5626]])
tensor3= tensor([[1.3127, 1.1263, 1.4634],
        [1.0282, 1.8656, 1.0846],
        [1.6852, 1.4324, 1.3756],
        [1.9992, 1.7216, 1.5859],
        [1.6413, 1.6382, 1.5626]])
```

图 15: 实例

1.15 尝试输出某个矩阵第一列的数据

先创建一个矩阵，然后用 `print(tensor[:, 0])` 就可以只输出该矩阵第一列的数据。

```
from __future__ import print_function
import torch

# 创建一个随机初始化的 5*3 矩阵
rand_x = torch.rand(5, 3)
print(rand_x)

print(rand_x[:, 0])

tensor([[0.9598, 0.1190, 0.0859],
        [0.8084, 0.0727, 0.3432],
        [0.2340, 0.0995, 0.6654],
        [0.3356, 0.8242, 0.0688],
        [0.1361, 0.8347, 0.3665]])
tensor([0.9598, 0.8084, 0.2340, 0.3356, 0.1361])
```

图 16: 实例

1.16 尝试对矩阵的尺寸进行修改

想要修改矩阵的尺寸，可以用 `torch.view()` 来实现。然后再用 `print` 输出结果查看修改情况。

```
from __future__ import print_function
import torch

x = torch.randn(4, 4)
y = x.view(16)
# -1 表示除给定维度外的其余维度的乘积
z = x.view(-1, 8)
print(x.size(), y.size(), z.size())

torch.Size([4, 4]) torch.Size([16]) torch.Size([2, 8])
Press any key to continue . . .
```

图 17: 实例

1.17 尝试将 Tensor 转换为 Numpy 数组, 然后修改 tensor 变量, 看看从 tensor 变量转换得到的 Numpy 数组变量是否发生变化。

想要实现将 Tensor 转换为 Numpy 数组，可以调用 `tensor.numpy()` 来实现转换操作，转换完成后输出结果。因为两者是共享同个内存空间的，所以修改 `tensor` 变量后，输出两个变量后发现，转换得到的 Numpy 数组变量也发生变化。


```
import numpy as np
a = np.ones(5)
b = torch.from_numpy(a)
np.add(a, 1, out=a)
print(a)
print(b)

[2. 2. 2. 2. 2.]
tensor([2., 2., 2., 2., 2.], dtype=torch.float64)
Press any key to continue . . .
```

图 20: 实例

1.20 创建一个 tensor, 并让 requires_grad=True 来追踪该变量, 执行任意计算操作并再次输出结果。然后尝试计算梯度 $d(out)/dx$, 并输出

首先用 `import torch` 导入必须的库, 创建一个 tensor, 并让 `requires_grad=True` 来追踪该变量相关的计算操作, 在计算得到 `out` 后, 因为它是一个标量, 因此用 `out.backward()` 相当于 `out.backward(torch.tensor(1.))`, 最后用 `grad` 得到梯度。

```
import torch
x = torch.ones(2, 2, requires_grad=True)
print(x)

y = x + 2
print(y)

z = y * y * 3
out = z.mean()
out.backward()
# 输出梯度 d(out)/dx
print(x.grad)
```

```
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
tensor([[3., 3.],
        [3., 3.]])
tensor([[[4.5000, 4.5000],
         [4.5000, 4.5000]]])
Press any key to continue . . .
```

图 21: 实例

2 实验结果

实验链接: <https://github.com/XY568/four.git>

3 心得体会

在学习了本周的实验内容后, 我发现自己还有许多的知识和工具需要掌握, 例如 markdown, pytorch 等。若能不断地学习新的工具和内容, 多练习并熟练应用它们, 可以大幅度提高工作效率。在学习的过程中, 会遇

到许多的难题，需要自己在网上查找相关解决方法或者询问他人，只有不被这些困难吓退，才能学到越来越多的实用的工具。即使课程结束了，我也不会放弃这门课上学到的内容，而是在空闲时间或学习中尝试应用它们，逐渐将它们的内容牢记于心。