实验报告

23020007067 李子昊

2024 年 9 月 15 日

目录

1 课后练习 **1**

1.1 调试及性能分析练习 1

1.2 元编程练习 4

1.3 PyTorch 编程练习 4

2 实例展示 **6**

2.1 调试及性能分析实例展示 6

2.2 元编程实例展示 11

2.3 PyTorch 编程实例展示 14

3 个人心得 **24**

4 github 链接 **25**

1 课后练习

1.1 调试及性能分析练习

1. 使用 python 中自带的 pdb–help 进行调试。

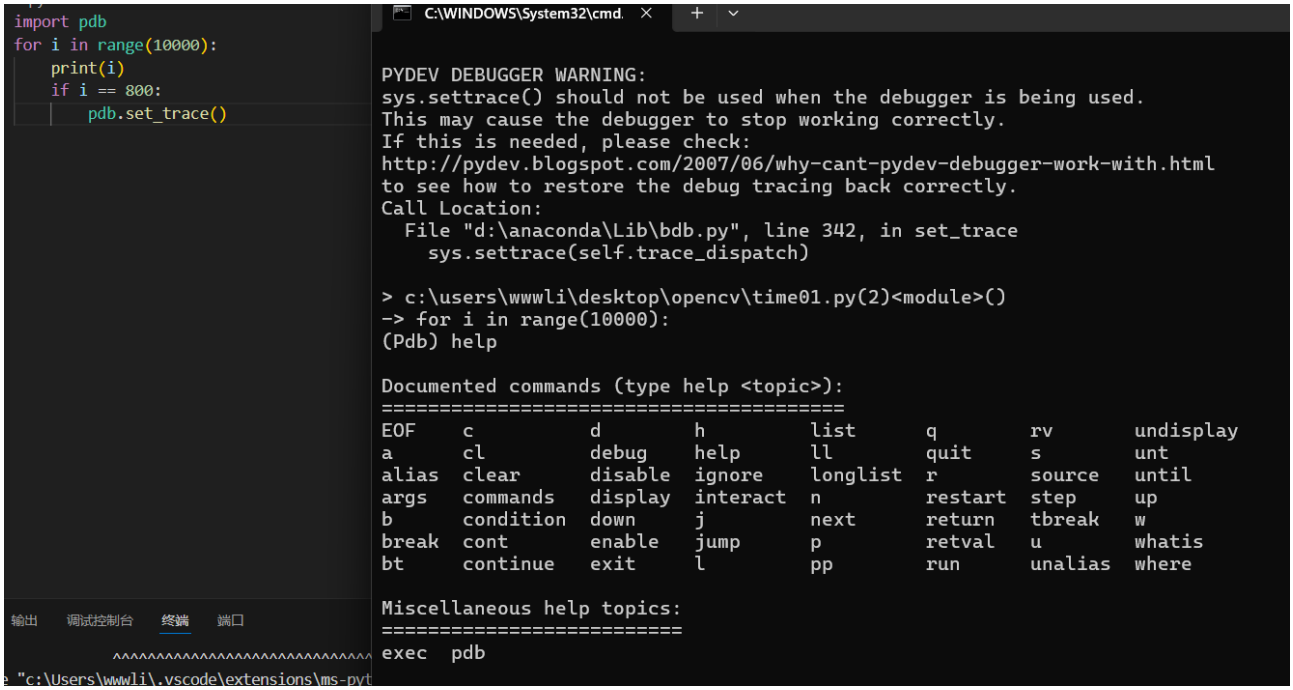

图 1: 题目一

2. 使用 Linux 上的 journalctl 或 macOS 上的 log show 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。如果找不到相关信息，您可以执行一些无害的命令，例如 sudo ls 然后再次查看。

```
li@li-VMware-Virtual-Platform:~$ journalctl | grep sudo
8月 30 17:22:10 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Bound t
o unit sssd.service, but unit isn't active.
8月 30 17:22:10 li-VMware-Virtual-Platform systemd[1]: Dependency failed for sss
d-sudo.socket - SSSD Sudo Service responder socket.
8月 30 17:22:10 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Job sss
d-sudo.socket/start failed with result 'dependency'.
8月 30 17:22:21 li-VMware-Virtual-Platform useradd[1490]: add 'li' to group 'sud
o'
8月 30 17:22:21 li-VMware-Virtual-Platform useradd[1490]: add 'li' to shadow gro
up 'sudo'
8月 31 11:47:00 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Bound t
o unit sssd.service, but unit isn't active.
8月 31 11:47:00 li-VMware-Virtual-Platform systemd[1]: Dependency failed for sss
d-sudo.socket - SSSD Sudo Service responder socket.
8月 31 11:47:00 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Job sss
d-sudo.socket/start failed with result 'dependency'.
8月 31 11:56:18 li-VMware-Virtual-Platform sudo[2709]:       li : TTY=pts/0 ; PW
D=/home/li ; USER=root ; COMMAND=/usr/bin/apt install vim
8月 31 11:56:18 li-VMware-Virtual-Platform sudo[2709]: pam_unix(sudo:session): s
ession opened for user root(uid=0) by li(uid=1000)
8月 31 11:56:44 li-VMware-Virtual-Platform sudo[2709]: pam_unix(sudo:session): s
ession closed for user root
```

```
li@li-VMware-Virtual-Platform:~$ sudo ls
[sudo] li 的密码：
公共  桌面          buggy.sh   function.sh        myfile         test1.sh
模板  abc.sh        case.sh    input.txt          myfile.sh      test.sh
视频  algorithm.sh  debug.sh   input.txt.bak      occurance.txt  vimrc
图片  all.sh        demo       li                 out.log        while.sh
文档  all.txt       demo.c     li.pub             printf.sh
下载  array.sh      etc        marco_history.log  snap
音乐  break.sh      for.sh     marco.sh           sum100.sh
li@li-VMware-Virtual-Platform:~$
```


```
li@li-VMware-Virtual-Platform:~$ journalctl | grep sudo
8月 30 17:22:10 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Bound t
o unit sssd.service, but unit isn't active.
8月 30 17:22:10 li-VMware-Virtual-Platform systemd[1]: Dependency failed for sss
d-sudo.socket - SSSD Sudo Service responder socket.
8月 30 17:22:10 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Job sss
d-sudo.socket/start failed with result 'dependency'.
8月 30 17:22:21 li-VMware-Virtual-Platform useradd[1490]: add 'li' to group 'sud
o'
8月 30 17:22:21 li-VMware-Virtual-Platform useradd[1490]: add 'li' to shadow gro
up 'sudo'
8月 31 11:47:00 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Bound t
o unit sssd.service, but unit isn't active.
8月 31 11:47:00 li-VMware-Virtual-Platform systemd[1]: Dependency failed for sss
d-sudo.socket - SSSD Sudo Service responder socket.
8月 31 11:47:00 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Job sss
d-sudo.socket/start failed with result 'dependency'.
8月 31 11:56:18 li-VMware-Virtual-Platform sudo[2709]:       li : TTY=pts/0 ; PW
D=/home/li ; USER=root ; COMMAND=/usr/bin/apt install vim
8月 31 11:56:18 li-VMware-Virtual-Platform sudo[2709]: pam_unix(sudo:session): s
```


图 2: 题目二

3. 请阅读可逆调试并尝试创建一个可以工作的例子(使用 rr 或 RevPDB)。

```
li@li-VMware-Virtual-Platform:~$ echo 1 | sudo tee /proc/sys/kernel/perf_event_p
aranoid
1
li@li-VMware-Virtual-Platform:~$ gcc -g demo.c -o demo
demo.c: In function ‘main’:
demo.c:5:9: warning: implicit declaration of function ‘printf’ [-Wimplicit-funct
ion-declaration]
    5 |         printf("%d",i);
      |         ^~~~~~
demo.c:2:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
    1 | #include<stdlib.h>
  +++ |+#include <stdio.h>
    2 |
demo.c:5:9: warning: incompatible implicit declaration of built-in function ‘pri
ntf’ [-Wbuiltin-declaration-mismatch]
    5 |         printf("%d",i);
      |         ^~~~~~
demo.c:5:9: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
li@li-VMware-Virtual-Platform:~$ ./demo
```

图 3: 题目三

4. 使用 python 语句判断程序运行了多长时间。

```
time01.py > ...
import time, random
n = random.randint(1, 10) * 100

# 获取当前时间
start = time.time()

# 执行一些操作
print("Sleeping for {} ms".format(n))
time.sleep(n/1000)

# 比较当前时间和起始时间
print(time.time() - start)
```

```
C:\WINDOWS\System32\cmd.

Sleeping for 700 ms
0.7010176181793213
请按任意键继续. . .
```

图 4: 题目四

1.2 元编程练习

1. 大多数的 makefiles 都提供了一个名为 clean 的构建目标，这并不是说我们会生成一个名为 clean 的文件，而是我们可以使用它清理文件，让 make 重新构建。您可以理解为它的作用是“撤销”所有构建步骤。在上面的 makefile 中为 paper.pdf 实现一个 clean 目标。您需要将构建目标设置为 phony。您也许会发现 git ls-files 子命令很有用。其他一些有用的 make 构建目标可以在这里找到。

```
li@li-VMware-Virtual-Platform:~$ cat makefile
paper.pdf: paper.tex plot-data.png
  pdflatex paper.tex

plot-%.png: %.dat plot.py
  ./plot.py -i $*.dat -o $@

.phony: clean
clean:
  mkdir -p Untrack
  rm -f *~ .*~
  git ls-files -o | grep -v Untrack | xargs -r mv -u -t Untrack
li@li-VMware-Virtual-Platform:~$ cat .gitignore
```


```
li@li-VMware-Virtual-Platform:~$ cat makefile
paper.pdf: paper.tex plot-data.png
  pdflatex paper.tex

plot-%.png: %.dat plot.py
  ./plot.py -i $*.dat -o $@

.phony: clean
clean:
  mkdir -p Untrack
  rm -f *~ .*~
  git ls-files -o | grep -v Untrack | xargs -r mv -u -t Untrack
li@li-VMware-Virtual-Platform:~$ cat .gitignore
```


图 5: make

1.3 PyTorch 编程练习

1. 求出 tensor 的一阶导、二阶导和三阶导

```python
import torch


x = torch.tensor(1.0, requires_grad=True)
a = torch.tensor(1.0)
b = torch.tensor(2.0)
c = torch.tensor(3.0)
y = a * torch.pow(x, 4) + b * x + c

#求一阶导
dy_dx = torch.autograd.grad(y, x, create_graph=True)[0]
#求二阶导
dy2_dx2 = torch.autograd.grad(dy_dx, x, create_graph=True)[0]
#求三阶导
dy3_dx3 = torch.autograd.grad(dy2_dx2, x)[0]
print(dy_dx.data, dy2_dx2.data, dy3_dx3)
```

```
C:\WINDOWS\System32\cmd.

tensor(6.) tensor(12.) tensor(24.)
请按任意键继续. . .
```

图 6: torch.autograd.grad

2. 实现张量的拼接 + 升维

```python
p-test6.py > ...
from __future__ import print_function
import torch

import cv2

tensor1=torch.rand(5,3)
tensor2=torch.rand(5,3)
tensor=torch.stack((tensor1,tensor2),dim=-1)
print("final:\n",tensor)
```

```
C:\WINDOWS\System32\cmd.

final:
 tensor([[[0.7880, 0.0787],
         [0.2495, 0.4787],
         [0.2406, 0.4019]],

        [[0.0633, 0.6353],
         [0.0967, 0.9855],
         [0.5667, 0.5589]],

        [[0.3215, 0.7482],
         [0.3823, 0.5792],
         [0.0183, 0.3186]],

        [[0.9575, 0.8257],
         [0.9586, 0.9836],
         [0.8370, 0.2295]],

        [[0.0440, 0.7076],
         [0.0570, 0.9508],
         [0.5255, 0.5889]]])
请按任意键继续. . .
```

图 7: torch.stack

3. 实现张量的行列交换

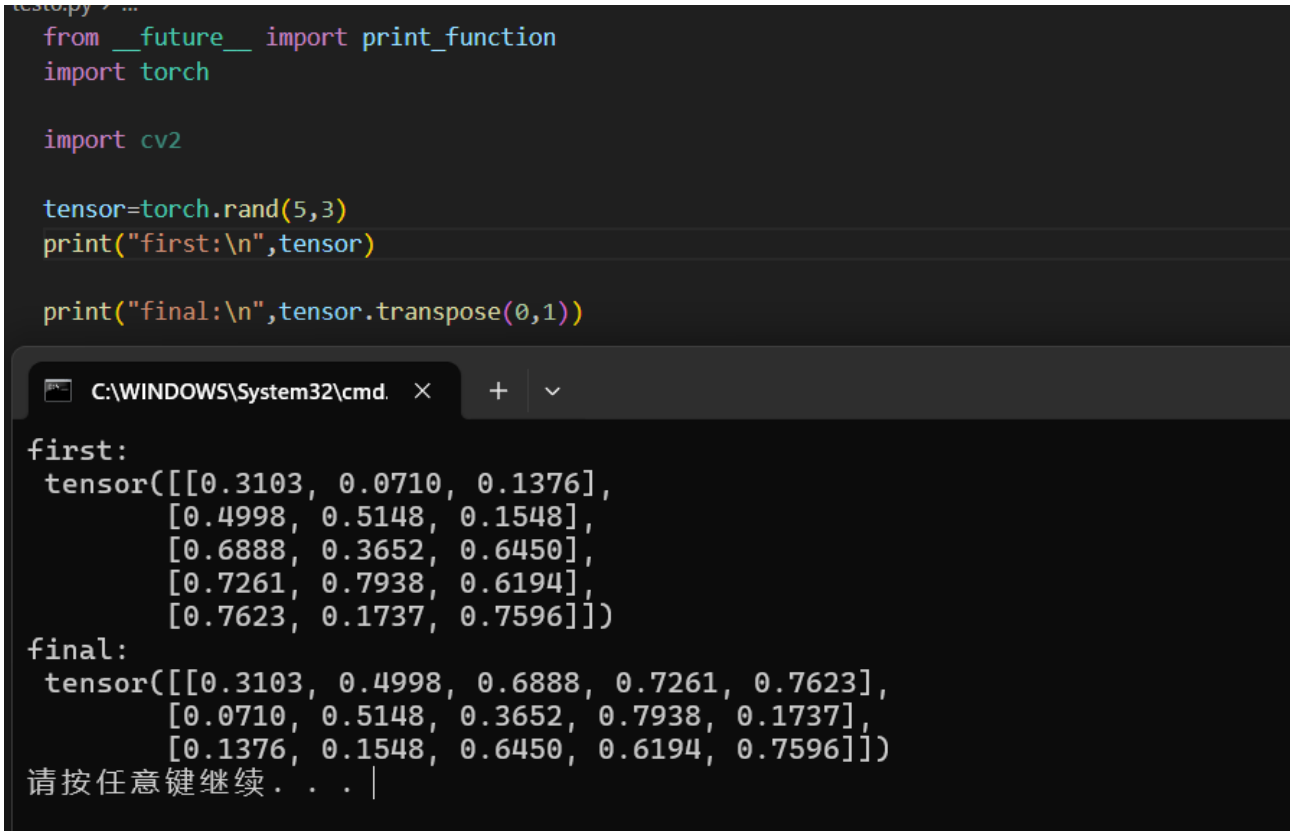

图 8: A.transpose(0，1)

2 实例展示

2.1 调试及性能分析实例展示

表 1: PyTorch 实例展示

1	journalctl | grep sudo	获取登录信息及指令
2	sudo ls	查看文件
3	sudo tee /proc/sys/kernel/perf_event_paranoid	向文件输出内容
4	gcc -g demo.c -o demo	编译运行 demo.c
5	pdb.set_trace()	启动 pdb 调试
6	pdb 中 ll	查看上下文
7	pdb 中 help	查看帮助
8	pdb 中 c	继续执行程序
9	cProfile.run()	查看代码总的效率以及各个部分的效率
10	stress -c 3	创建负载

1.journalctl | grep sudo ： 获取登录信息及指令

```
li@li-VMware-Virtual-Platform:~$ journalctl | grep sudo
8月  30 17:22:10 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Bound t
o unit sssd.service, but unit isn't active.
8月  30 17:22:10 li-VMware-Virtual-Platform systemd[1]: Dependency failed for sss
d-sudo.socket - SSSD Sudo Service responder socket.
8月  30 17:22:10 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Job sss
d-sudo.socket/start failed with result 'dependency'.
8月  30 17:22:21 li-VMware-Virtual-Platform useradd[1490]: add 'li' to group 'sud
o'
8月  30 17:22:21 li-VMware-Virtual-Platform useradd[1490]: add 'li' to shadow gro
up 'sudo'
8月  31 11:47:00 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Bound t
o unit sssd.service, but unit isn't active.
8月  31 11:47:00 li-VMware-Virtual-Platform systemd[1]: Dependency failed for sss
d-sudo.socket - SSSD Sudo Service responder socket.
8月  31 11:47:00 li-VMware-Virtual-Platform systemd[1]: sssd-sudo.socket: Job sss
d-sudo.socket/start failed with result 'dependency'.
8月  31 11:56:18 li-VMware-Virtual-Platform sudo[2709]:       li : TTY=pts/0 ; PW
D=/home/li ; USER=root ; COMMAND=/usr/bin/apt install vim
8月  31 11:56:18 li-VMware-Virtual-Platform sudo[2709]: pam_unix(sudo:session): s
```

图 9: journalctl | grep sudo

2.sudo ls ： 查看文件

```
li@li-VMware-Virtual-Platform:~$ sudo ls
[sudo] li 的密码：
公共  桌面          buggy.sh   function.sh        myfile         test1.sh
模板  abc.sh        case.sh    input.txt          myfile.sh      test.sh
视频  algorithm.sh  debug.sh   input.txt.bak      occurance.txt  vimrc
图片  all.sh        demo       li                 out.log        while.sh
文档  all.txt       demo.c     li.pub             printf.sh
下载  array.sh      etc        marco_history.log  snap
音乐  break.sh      for.sh     marco.sh           sum100.sh
li@li-VMware-Virtual-Platform:~$
```

图 10: sudo ls

3.sudo tee /proc/sys/kernel/perf_event_paranoid

```
li@li-VMware-Virtual-Platform:~$ echo 1 | sudo tee /proc/sys/kernel/perf_event_p
aranoid
[sudo] li 的密码：
1
li@li-VMware-Virtual-Platform:~$
```

图 11: sudo tee /proc/sys/kernel/perf_event_paranoid

4.gcc -g demo.c -o demo: 编译运行 demo.c

```
li@li-VMware-Virtual-Platform:~$ gcc -g demo.c -o demo
demo.c: In function ‘main’:
demo.c:5:9: warning: implicit declaration of function ‘printf’ [-Wimplicit-funct
ion-declaration]
    5 |         printf("%d",i);
      |         ^~~~~~
demo.c:2:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
    1 | #include<stdlib.h>
  +++ |+#include <stdio.h>
    2 |
demo.c:5:9: warning: incompatible implicit declaration of built-in function ‘pri
ntf’ [-Wbuiltin-declaration-mismatch]
    5 |         printf("%d",i);
      |         ^~~~~~
demo.c:5:9: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
```

图 12: gcc -g demo.c -o demo

5.pdb.set_trace(): 启动 pdb 调试

```
import pdb
for i in range(10000):
    print(i)
    if i == 800:
        pdb.set_trace()
```

```
C:\WINDOWS\System32\cmd.

PYDEV DEBUGGER WARNING:
sys.settrace() should not be used when the debugger is being used.
This may cause the debugger to stop working correctly.
If this is needed, please check:
http://pydev.blogspot.com/2007/06/why-cant-pydev-debugger-work-with.html
to see how to restore the debug tracing back correctly.
Call Location:
  File "d:\anaconda\Lib\bdb.py", line 342, in set_trace
    sys.settrace(self.trace_dispatch)

> c:\users\wwwli\desktop\opencv\time01.py(2)<module>()
-> for i in range(10000):
(Pdb) help

Documented commands (type help <topic>):
========================================
EOF     c          d        h         list      q        rv       undisplay
a       cl         debug    help      ll        quit     s        unt
alias   clear      disable  ignore    longlist  r        source   until
args    commands   display  interact  n         restart  step     up
b       condition  down     j         next      return   tbreak   w
break   cont       enable   jump      p         retval   u        whatis
bt      continue   exit     l         pp        run      unalias  where

Miscellaneous help topics:
==========================
exec  pdb
```

图 13: pdb.set_trace()

6.pdb 中 ll: 查看上下文

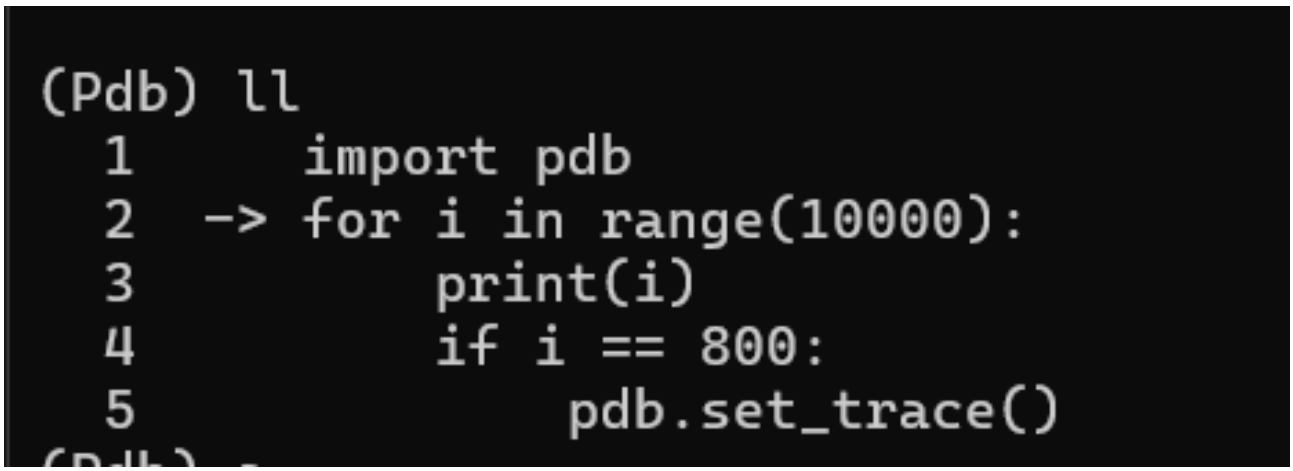

图 14: ll

7.pdb 中 help: 查看帮助

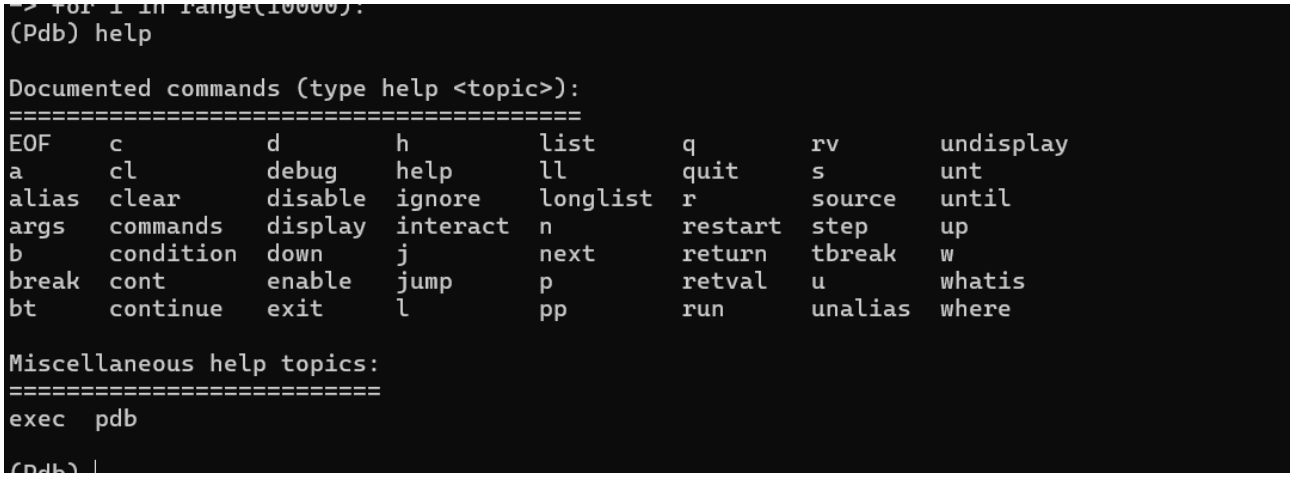

图 15: help

8.pdb 中 c: 继续执行程序

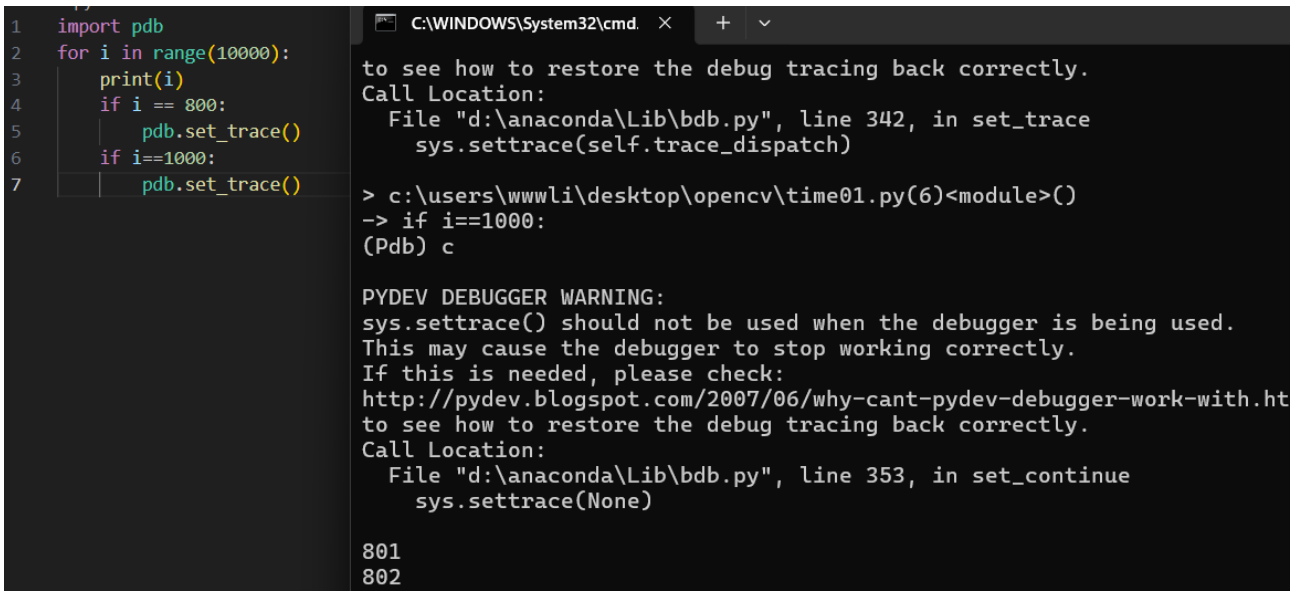

图 16: c

9.cProfile.run(): 查看代码总的效率以及各个部分的效率

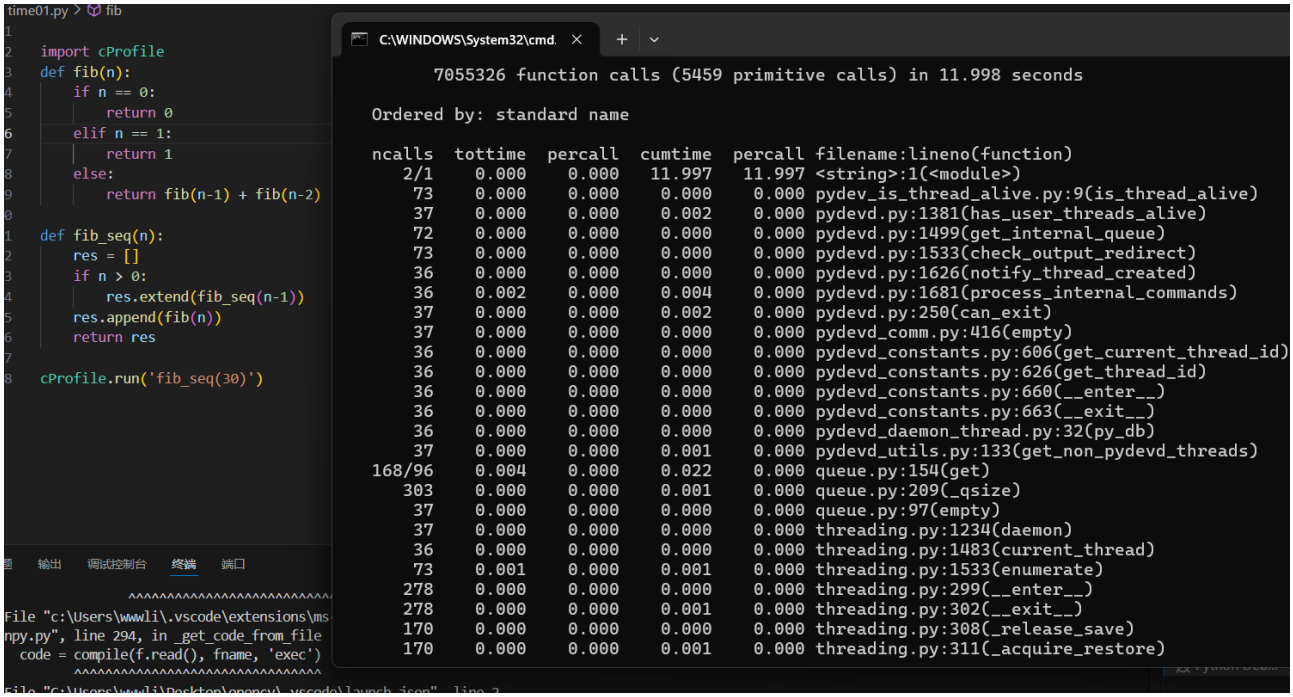

图 17: cProfile.run()

10.stress -c 3: 创建负载

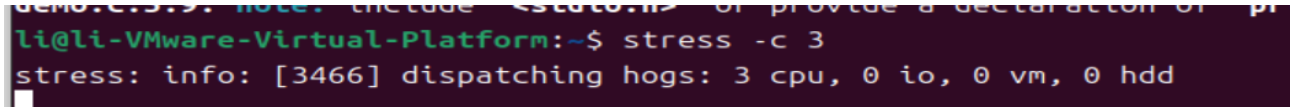

图 18: stress -c 3

2.2 元编程实例展示

表 2: PyTorch 实例展示

1	___init___ 和 ___call___	类的初始化
2	def ___get___(self, instance, cls)	装饰器
3	template<typename T, std::size$_t N >$	混合元编程
4	constexpr T Factorial(T x)	类元编程
5	struct remove$_r eference <_T\ y$	
>	类型元编程	
6	func=decorator（func）	装饰器表示

1. 一个类只能有一个实例

```python
class Singleton(type):
    def __init__(self, *args, **kwargs):
        self._instance = None
        super().__init__(*args, **kwargs)

    def __call__(self, *args, **kwargs):
        if self._instance is None:
            self._instance = super().__call__(*args, **kwargs)
            return self._instance
        else:
            return self._instance


class Spam(metaclass=Singleton):
    def __init__(self):
        print("Spam!!!")
```

图 19: ___init___ 和 ___call___

2.c++ 值元编程

```cpp
//值元编程
#include<iostream>
using namespace std;
constexpr int Factorial(unsigned int n) {
    if (n <= 1) {
        return 1;
    }
    else {
        return n * Factorial(n - 1);
    }
}

int main() {
    static_assert(Factorial(4) == 24, "error");
    printf("值元编程！");
    return 0;
}
```

```
Microsoft Visual Studio 调试控
值元编程！
D:\C++\Project1\x64\Debug\Project1.exe (进程 20592)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

图 20: c++ 值元编程

3.c++ 混合元编程

```python
from functools import wraps

def print_result(func):

    @wraps(func)
    def wrappper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(result)
        return result

    return wrappper

@print_result
def add(x, y):
    return x + y
#相当于:
#add = print_result(add)

add(1, 3)
```

```
C:\WINDOWS\System32\cmd.
4
请按任意键继续. . .
```

图 21: c++ 混合元编程

4. 装饰器

```cpp
//混合元编程
#include <iostream>
#include <array>
using namespace std;

template<typename T, std::size_t N>
struct DotProductT {
    static inline T result(const T* a, const T* b) {
        return (*a) * (*b) + DotProductT<T, N - 1>::result(a + 1, b + 1);
    }
};

template<typename T>
struct DotProductT<T, 0> {
    static inline T result(const  T*, const  T*) {
        return T{};
    }
};
template<typename T, std::size_t N>
auto dotProduct(std::array<T, N> const& x, std::array<T, N> const& y) {
    return DotProductT<T, N>::result(x.data(), y.data());
}

int main() {
    array<int, 3> A{ 1, 2, 3 };

    auto x = dotProduct(A, A);
    cout << x << endl;
    return 0;
}
```

```
Microsoft Visual Studio 调试控

14

D:\C++\Project1\x64\Debug\Project1.exe (进程 15248)已退出，代
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->
按任意键关闭此窗口. . .
```

图 22: 装饰器

5.func=decorator（func）

```python
def attr_upper(cls):
    for attrname,value in cls.__dict__.items():
        if isinstance(value,str):
            if not value.startswith('__'):
                setattr(cls,attrname,bytes.decode(str.encode(value).upper()))
    return cls

@attr_upper
class Person:
    sex = 'man'

print(Person.sex) # MAN
```

```
C:\WINDOWS\System32\cmd.

MAN
请按任意键继续. . .
```

图 23: func=decorato（func）

2.3 PyTorch 编程实例展示

表 3: PyTorch 实例展示

1	torch.empty()	声明一个未初始化的矩阵
2	torch.rand()	随机初始化一个矩阵
3	torch.zeros()	创建数值皆为 0 的矩阵
4	torch.tensor()	直接传递 tensor 数值来创建
5	tensor.new_ones()	根据已有的 tensor 变量创建新的 tensor 变量
6	torch.randn_like(old_tensor)	保留相同的尺寸大小
7	tensor1.add_(tensor2)	直接修改 tensor 变量
8	tensor3.size()	对 tensors 的尺寸大小获取
9	A.backward(torch.tensor(1.))	计算梯度
10	torch.range(begin，end，step)	起始位置，终止位置和步长
11	torch.mv(A,B)	矩阵与向量相乘
12	A.numpy()	将 Tensor 转化为 Numpy 类型
13	torch.diag(A)	取 A 对角线元素形成一个一维向量
14	A.transpose(0，1)	行列交换
15	torch.stack	拼接 + 升维
16	autograd.grad	求导

1.torch.empty()：声明一个未初始化的矩阵

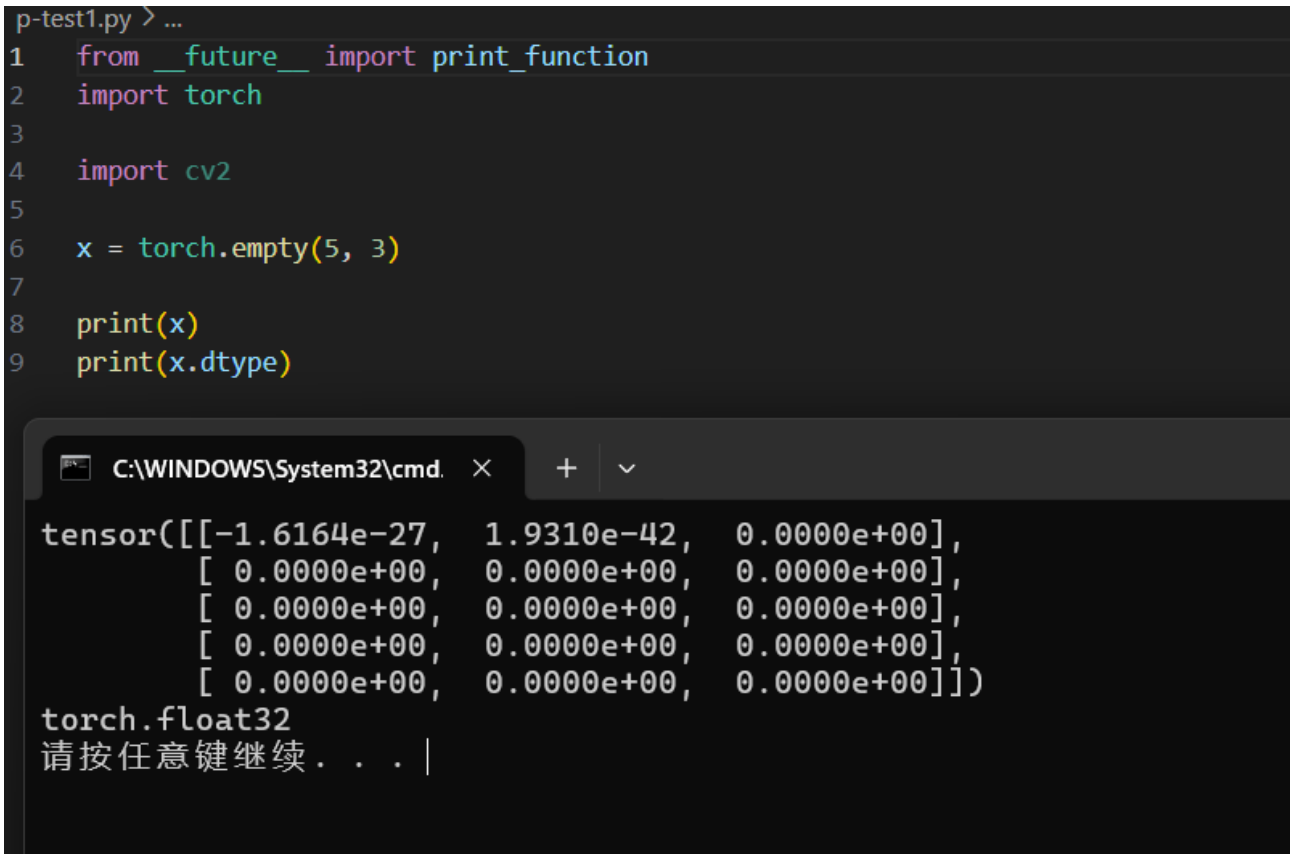

图 24: torch.empty()

2.torch.rand()：随机初始化一个矩阵

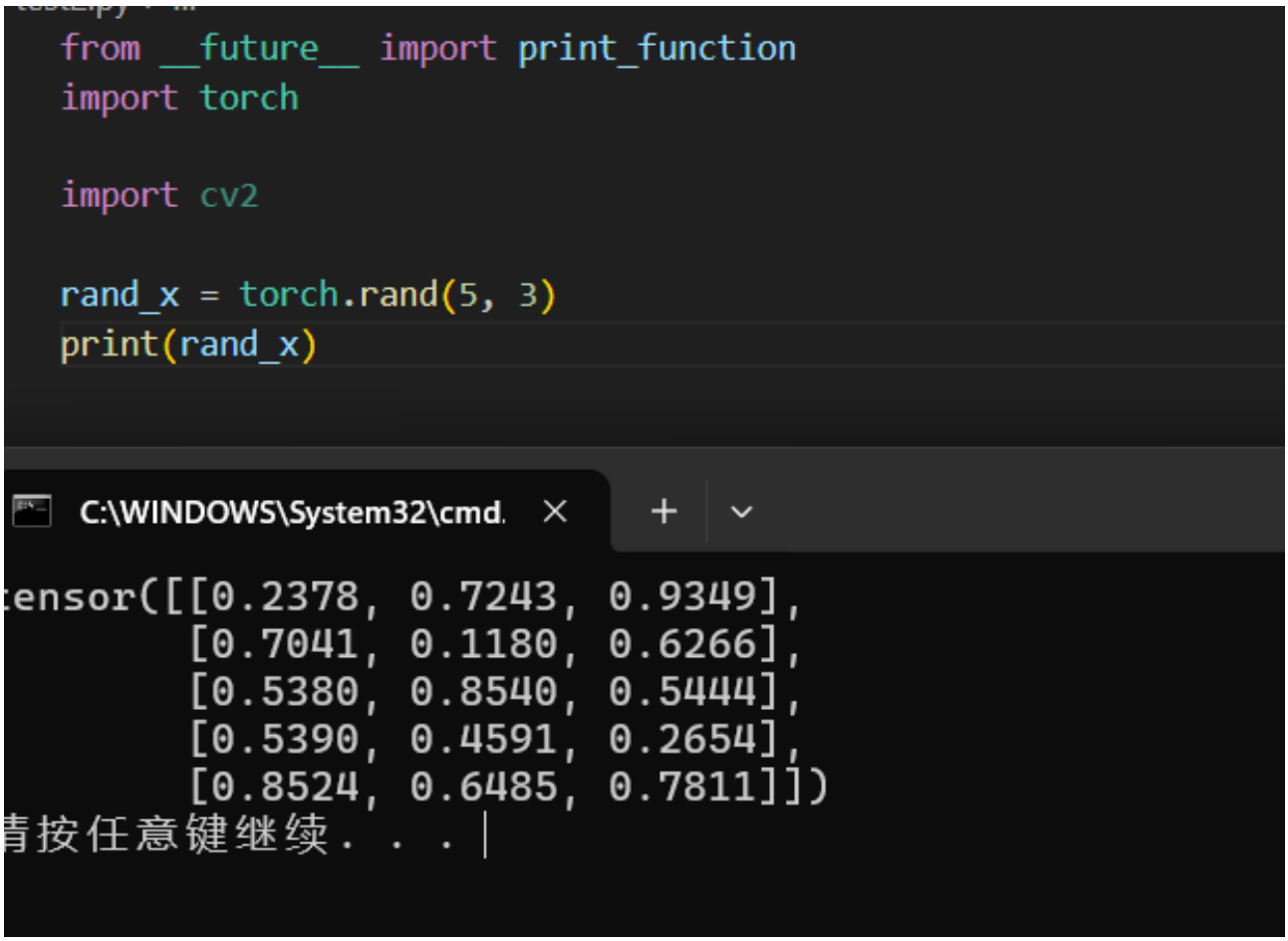

图 25: torch.rand()

3.torch.zeros()：创建数值皆为 0 的矩阵

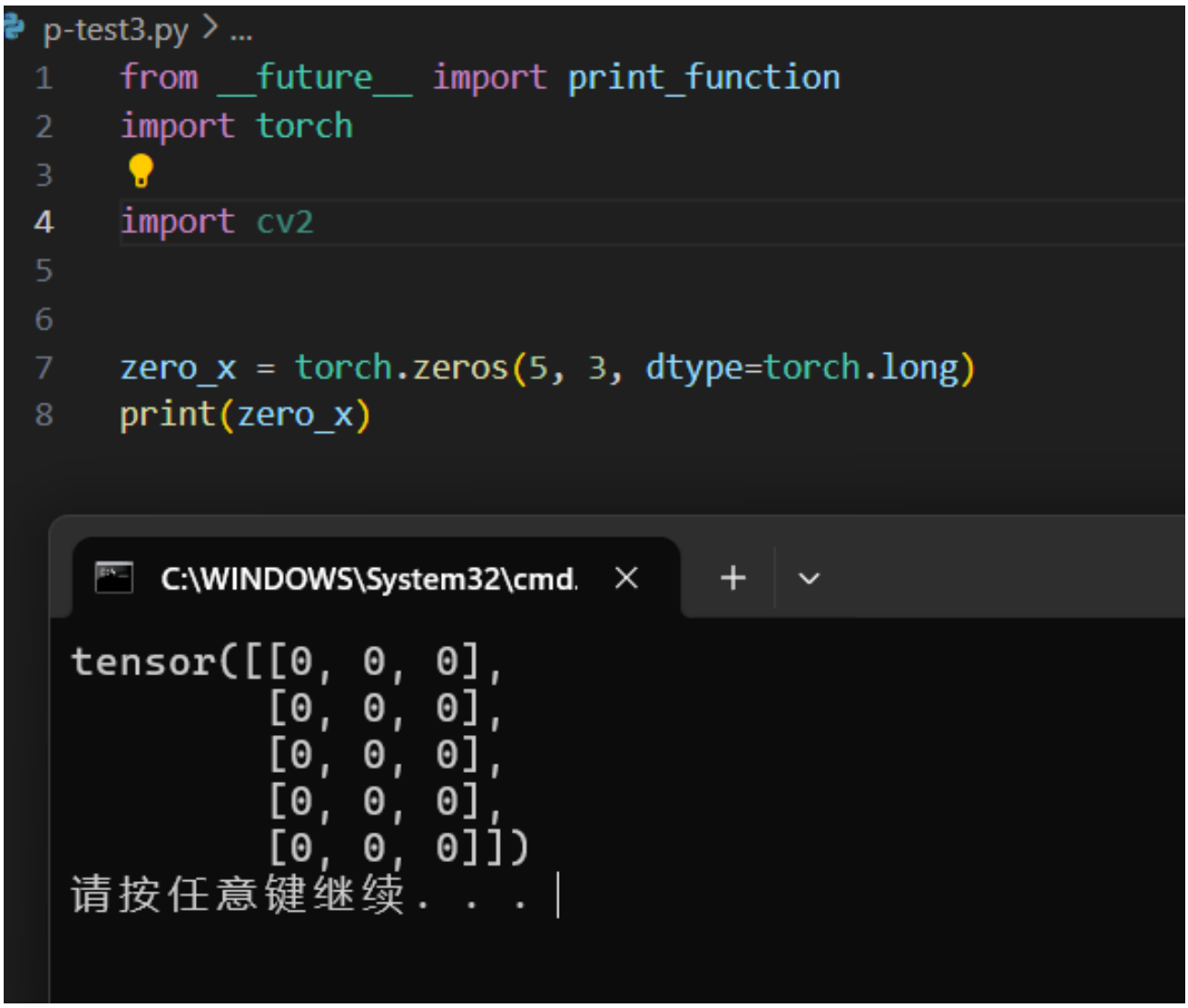

图 26: torch.zeros()

4.torch.tensor()：直接传递 tensor 数值来创建

```
p-test4.py > ...
from __future__ import print_function
import torch

import cv2



tensor1 = torch.tensor([5.5, 3])
print(tensor1)
```

```
C:\WINDOWS\System32\cmd.
tensor([5.5000, 3.0000])
请按任意键继续. . .
```

图 27: torch.tensor()

5.tensor.new_ones()：new_*() 方法需要输入尺寸大小

```
from __future__ import print_function
import torch

import cv2



tensor1 = torch.tensor([5.5, 3])

tensor2 = tensor1.new_ones(5, 3, dtype=torch.double)  # new_* 方法需要输入 tensor 大小
print(tensor2)
```

```
C:\WINDOWS\System32\cmd.
tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]], dtype=torch.float64)
请按任意键继续. . .
```

图 28: torch.new_ones()

6.torch.randn_like(old_tensor)：保留相同的尺寸大小

```
p-test4.py > ...
from __future__ import print_function
import torch
import cv2

tensor1 = torch.tensor([5.5, 3])

tensor2 = tensor1.new_ones(5, 3, dtype=torch.double)  # new_* 方法需要输入 tensor 大小

tensor3 = torch.randn_like(tensor2, dtype=torch.float)
print('tensor3: ', tensor3)
```

```
C:\WINDOWS\System32\cmd.

tensor3:  tensor([[ 0.3059, -0.8519,  0.1292],
        [ 0.3454,  1.6221, -0.1320],
        [-0.9573, -0.3025, -2.5566],
        [-0.0116, -0.3634,  0.8308],
        [ 1.0384,  1.6286, -0.8239]])
请按任意键继续. . .
```

图 29: torch.randn_like(old_tensor)

7.tensor1.add_(tensor2)：直接修改 tensor 变量

```
test4.py > ...
from __future__ import print_function
import torch
import cv2

tensor1 = torch.rand(5, 3)
tensor2 = torch.rand(5, 3)

print("last_tensor1:",tensor1)
tensor1.add_(tensor2)
print("now_tensor1:",tensor1)


#tensor3 = torch.randn_like(tensor2
#print(tensor3.size())
```

```
C:\WINDOWS\System32\cmd.

last_tensor1: tensor([[0.2647, 0.7429, 0.9674],
        [0.7013, 0.1386, 0.5230],
        [0.9341, 0.2944, 0.8508],
        [0.5605, 0.0502, 0.8998],
        [0.1302, 0.8815, 0.6653]])
now_tensor1: tensor([[0.8252, 1.5584, 1.5770],
        [1.0812, 0.3288, 0.6098],
        [1.0096, 0.3582, 0.9276],
        [1.0598, 0.8433, 1.6109],
        [0.2369, 1.5122, 1.1716]])
请按任意键继续. . .
```

图 30: tensor1.add_(tensor2)

8.tensor3.size()：对 tensors 的尺寸大小获取

```python
from __future__ import print_function
import torch
import cv2

tensor1 = torch.tensor([5.5, 3])

tensor2 = tensor1.new_ones(5, 3, dtype=torch.double)  # new_* 方法需要输入 t

tensor3 = torch.randn_like(tensor2, dtype=torch.float)
print(tensor3.size())
```

```
torch.Size([5, 3])
请按任意键继续. . .
```

图 31: torch.rand()

9.A.backward(torch.tensor(1.)): 计算梯度

```python
import torch
import torch.nn as nn

x = torch.tensor([2, 3, 4], dtype=torch.float, requires_grad=True)
print(x)
y = x * 2
while y.norm() < 1000:
    y = y * 2
print(y)

y.backward(torch.ones_like(y))
print(x.grad)
```

```
tensor([2., 3., 4.], requires_grad=True)
tensor([ 512.,  768., 1024.], grad_fn=<MulBackward0>)
tensor([256., 256., 256.])
请按任意键继续. . .
```

图 32: A.backward

10.torch.range(begin，end，step): 起始位置，终止位置和步长

```python
from __future__ import print_function
import torch
import cv2

print(torch.range(1,10,2))
```

```
C:\WINDOWS\System32\cmd.

C:\Users\wwwli\Desktop\opencv\p-test3.py:5: UserW
se because its behavior is inconsistent with Pyth
 [start, end).
  print(torch.range(1,10,2))
tensor([1., 3., 5., 7., 9.])
请按任意键继续. . .
```

图 33: torch.range(begin，end，step)

11.torch.mv(A,B): 矩阵与向量相乘

```python
from __future__ import print_function
import torch
import cv2

a = torch.tensor([
    [1, 0, 1],
    [0, 1, 0],
    [1, 0, 1],
])

b = torch.tensor([
    [3, 1, 3],
    [1, 0, 1],
    [3, 1, 3],
])
c = torch.tensor([
    [1, 1, 1],
    [0, 1, 1],
    [0, 0, 1],
])
w1 = torch.tensor([1, 2, 1])
w2 = torch.tensor([3, 4, 3])

res5 = torch.mv(a, w1)
res55 = torch.mv(a, w1.t())
print("torch.mv(a, w1)={}\n".format(res5))
print("torch.mv(a, w1.t())={}\n".format(res55))
```

```
C:\WINDOWS\System32\cmd.

torch.mv(a, w1)=tensor([2, 2, 2])

torch.mv(a, w1.t())=tensor([2, 2, 2])

请按任意键继续. . .
```

图 34: torch.mv(A,B)

12.Tensor 转换为 Numpy 数组

```
from __future__ import print_function
import torch

import cv2

tensor=torch.rand(5,3)
print(tensor.numpy)
```

```
C:\WINDOWS\System32\cmd.

<built-in method numpy of Tensor object at 0x000002120E8823A0>
请按任意键继续. . .
```

图 35: torch.rand()

13.torch.diag(A): 取 A 对角线元素形成一个一维向量

```
from __future__ import print_function
import torch

import cv2

tensor=torch.rand(5,3)
print("first:\n",tensor)

print("final:\n",torch.diag(tensor))
```

```
C:\WINDOWS\System32\cmd.

first:
 tensor([[0.4010, 0.1481, 0.1687],
        [0.8863, 0.5285, 0.2955],
        [0.8623, 0.9473, 0.8841],
        [0.4574, 0.5802, 0.5688],
        [0.8637, 0.4272, 0.6715]])
final:
 tensor([0.4010, 0.5285, 0.8841])
请按任意键继续. . .
```

图 36: torch.diag(A)

14.A.transpose(0，1): 行列交换

```python
from __future__ import print_function
import torch

import cv2

tensor=torch.rand(5,3)
print("first:\n",tensor)

print("final:\n",tensor.transpose(0,1))
```

```
C:\WINDOWS\System32\cmd.

first:
 tensor([[0.3103, 0.0710, 0.1376],
        [0.4998, 0.5148, 0.1548],
        [0.6888, 0.3652, 0.6450],
        [0.7261, 0.7938, 0.6194],
        [0.7623, 0.1737, 0.7596]])
final:
 tensor([[0.3103, 0.4998, 0.6888, 0.7261, 0.7623],
        [0.0710, 0.5148, 0.3652, 0.7938, 0.1737],
        [0.1376, 0.1548, 0.6450, 0.6194, 0.7596]])
请按任意键继续. . .
```

图 37: A.transpose(0，1)

15.torch.stack: 拼接 + 升维

```python
p-test6.py > ...
from __future__ import print_function
import torch

import cv2

tensor1=torch.rand(5,3)
tensor2=torch.rand(5,3)
tensor=torch.stack((tensor1,tensor2),dim=-1)
print("final:\n",tensor)
```

```
C:\WINDOWS\System32\cmd.

final:
 tensor([[[0.7880, 0.0787],
         [0.2495, 0.4787],
         [0.2406, 0.4019]],

        [[0.0633, 0.6353],
         [0.0967, 0.9855],
         [0.5667, 0.5589]],

        [[0.3215, 0.7482],
         [0.3823, 0.5792],
         [0.0183, 0.3186]],

        [[0.9575, 0.8257],
         [0.9586, 0.9836],
         [0.8370, 0.2295]],

        [[0.0440, 0.7076],
         [0.0570, 0.9508],
         [0.5255, 0.5889]]])
请按任意键继续. . .
```

图 38: torch.stack

16.torch.autograd.grad: 求导

```python
import torch


x = torch.tensor(1.0, requires_grad=True)
a = torch.tensor(1.0)
b = torch.tensor(2.0)
c = torch.tensor(3.0)
y = a * torch.pow(x, 4) + b * x + c

#求一阶导
dy_dx = torch.autograd.grad(y, x, create_graph=True)[0]
#求二阶导
dy2_dx2 = torch.autograd.grad(dy_dx, x, create_graph=True)[0]
#求三阶导
dy3_dx3 = torch.autograd.grad(dy2_dx2, x)[0]
print(dy_dx.data, dy2_dx2.data, dy3_dx3)
```

```
C:\WINDOWS\System32\cmd.

tensor(6.) tensor(12.) tensor(24.)
请按任意键继续. . .
```

图 39: torch.autograd.grad

3 个人心得

在深度学习中，性能调试对于提升模型训练效率和节省计算资源至关重要。通过合理的性能优化，能显著减少训练时间并降低硬件压力。尤其是使用 PyTorch 时，了解其 GPU 加速、混合精度训练和数据加载优化等功能，可以有效提升模型的运行速度，避免资源浪费。

PyTorch 提供了多种性能优化手段。首先，充分利用 GPU 并减少 CPU 和 GPU 之间的数据传输是基本操作。其次，通过梯度累加和混合精度训练能减少显存消耗，提升训练速度。此外，优化数据加载，增加并行处理的线程数量，也能明显提升效率。

性能调试工具是优化的关键。PyTorch Profiler 是官方提供的性能分析工具，能深入分析每个操作的执行时间和内存占用。结合 TensorBoard 等可视化工具，可以直观地查看模型在训练中的表现，并快速定位瓶颈，从而针对性地优化。

学习性能调试时，应先理解模型训练中的基本原理，逐步从小规模实验开

始，然后扩展到复杂模型。同时，保持对 PyTorch 社区更新的关注，有助于学习到新的调试和优化方法，以便在实际应用中取得更好的效果。

4 github 链接

https://github.com/newbeginnerlzh/git-task01.git