系统开发工具基础
调试及性能分析 & 元编程 & PyTorch

王志巍 22020007161

2024 - 09 - 13

目录

1 调试及性能分析

1.1 日志

```python
import logging

# 配置日志记录
logging.basicConfig(
    filename='testlog.log',  # 日志文件名
    level=logging.DEBUG,  # 日志级别
    format='%(asctime)s - %(levelname)s - %(message)s'  # 日志格式
)

def main():
    logging.info('程序开始运行')
    try:
        a = 0
        b = 1
        result = a + b
        logging.debug(f'计算结果: {a} + {b} = {result}')
        # 警告
        if b == 1:
            logging.warning('警告,b 的值为 1')
        # 错误
        if a == 0:
            raise ValueError('a 的值不能为 0')
    except Exception as e:
        logging.error(f'发生错误: {e}')

    logging.info('程序结束运行')

if __name__ == '__main__':
    main()
```

```
ouc@islouc-vm:~/Desktop/debug0$ py test.py
ouc@islouc-vm:~/Desktop/debug0$ cat testlog.log
2024-09-13 12:48:53,855 - INFO - 程序开始运行
2024-09-13 12:48:53,856 - DEBUG - 计算结果: 0 + 1 = 1
2024-09-13 12:48:53,856 - WARNING - 警告,b 的值为 1
2024-09-13 12:48:53,856 - ERROR - 发生错误: a 的值不能为 0
2024-09-13 12:48:53,856 - INFO - 程序结束运行
```

1.2 反汇编工具 GDB

1.2.1 GDB

```
gdb <可执行文件>
```

```
ouc@islouc-vm:~/Desktop/debug0$ gdb bomb
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.2) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
pwndbg: loaded 147 pwndbg commands and 48 shell commands. Type pwndbg [--shell | --all] [filter] for a list.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
Reading symbols from bomb...
------- tip of the day (disable with set show-tips off) -------
The set show-flags on setting will display CPU flags register in the regs context panel
```

1.2.2 disassemble

disassemble 指令可以反汇编代码，显示程序的汇编指令。使用时往往可以用缩写 disass 代替。

```
disass <函数名>
```

```
pwndbg> disass phase_1
Dump of assembler code for function phase_1:
   0x00000000000015b4 <+0>:     endbr64
   0x00000000000015b8 <+4>:     push   rbp
   0x00000000000015b9 <+5>:     mov    rbp,rsp
   0x00000000000015bc <+8>:     lea    rsi,[rip+0x1b8d]        # 0x3150
   0x00000000000015c3 <+15>:    call   0x1af1 <strings_not_equal>
   0x00000000000015c8 <+20>:    test   eax,eax
   0x00000000000015ca <+22>:    jne    0x15ce <phase_1+26>
   0x00000000000015cc <+24>:    pop    rbp
   0x00000000000015cd <+25>:    ret
   0x00000000000015ce <+26>:    call   0x1d6d <explode_bomb>
   0x00000000000015d3 <+31>:    jmp    0x15cc <phase_1+24>
End of assembler dump.
```

1.2.3 内存查询

```
x /<类型> <地址>
/d   以十进制显示整数
/wd  以十进制显示无符号整数
/f   显示浮点数
/c   显示字符
/s   显示字符串
```

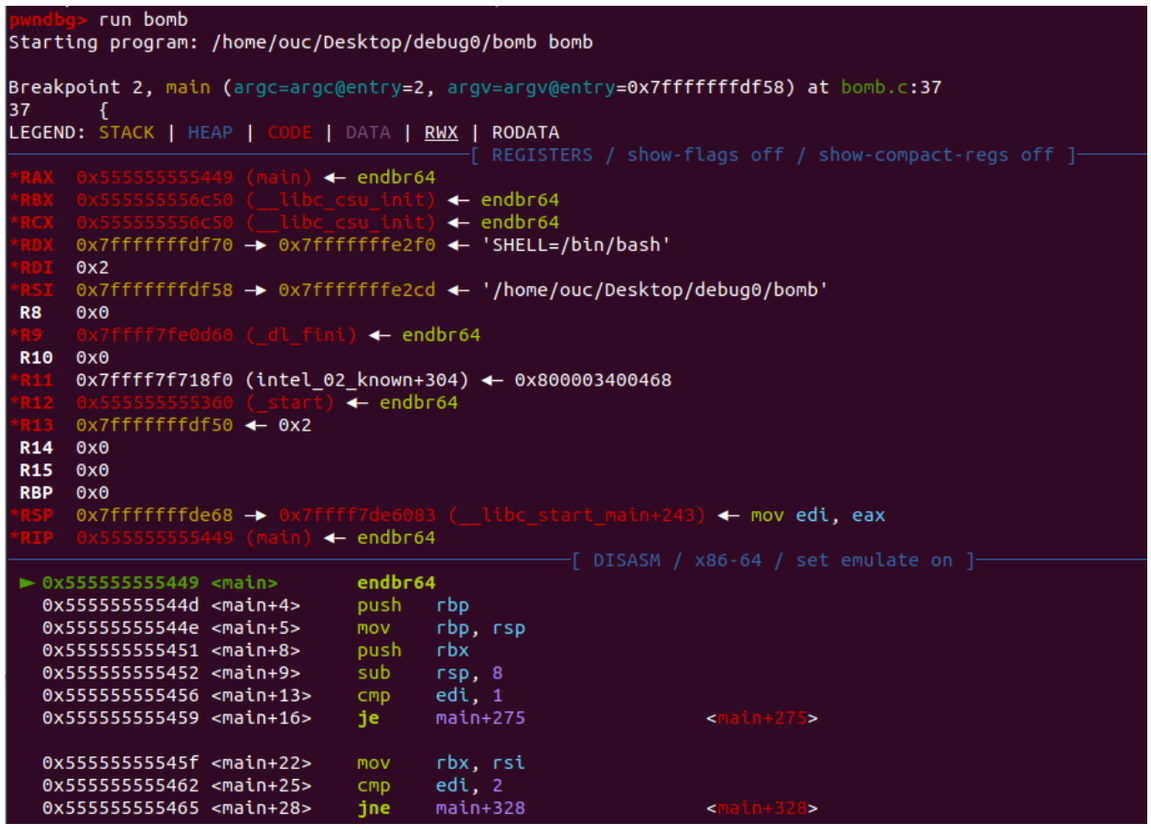
```
pwndbg> x /s 0x3150
0x3150: "When a problem comes along, you must zip it!"
```

1.2.4 断点调试

设置断点后运行可执行文件，gdb 会先运行到第一个断点，之后有程序员控制，逐行执行。

```
break <地址/函数名>
run <文件名>
```

```
pwndbg> break main
Breakpoint 2 at 0x555555555449: file bomb.c, line 37.
```



1.3 性能分析

1.3.1 计时

```
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)
```

```
Sleeping for 100 ms
0.10814547538757324
```

1.3.2 CPU

```
import cProfile
import pstats
import io

def example_function():
    # 示例函数：计算前10000个数的平方和
    total = 0
    for i in range(10000):
        total += i ** 2
    return total
```

```
def profile_function(func):
    pr = cProfile.Profile()
    pr.enable()
    func()
    pr.disable()

    s = io.StringIO()
    sortby = 'cumulative'
    ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
    ps.print_stats()
    print(s.getvalue())

if __name__ == '__main__':
    profile_function(example_function)
```

```
2 function calls in 0.002 seconds
Ordered by: cumulative time
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    1    0.002    0.002    0.002    0.002
        testcpu.py:5(example_function)
    1    0.000    0.000    0.000    0.000
        {method 'disable' of '_lsprof.Profiler' objects}
```

1.3.3 内存

```
from memory_profiler import profile

@profile
def example_function():
    # 大列表占用内存
    large_list = [i for i in range(100000)]
    return large_list

if __name__ == '__main__':
    example_function()
```

```
python3 -m memory_profiler testmem.py

结果:
Filename: testmem.py
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     3     18.7 MiB     18.7 MiB           1   @profile
     4                                         def example_function():
     5                                         # 大列表占用内存
     6     22.6 MiB      3.9 MiB      100003   large_list =
                                               [i for i in range(100000)]
     7     22.6 MiB      0.0 MiB           1   return large_list
```

1.3.4 事件分析

```
import numpy as np
```

```python
def compute():
    matrix = np.random.rand(1000, 1000)
    result = np.dot(matrix, matrix)
    row_sums = np.sum(result, axis=1)
    return row_sums

if __name__ == '__main__':
    result = compute()
    print("Computation completed. Result sum:", np.sum(result))
```

使用 perf 进行事件分析

收集事件

```
perf stat python3 testevt.py
```

```
Computation completed. Result sum: 249899430.0565424

 Performance counter stats for 'python3 testevt.py':

            944.24 msec task-clock                #    0.915 CPUs utilized
               322      context-switches          #  341.016 /sec
                 2      cpu-migrations            #    2.118 /sec
             5,905      page-faults               #    6.254 K/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       1.032439524 seconds time elapsed

       0.831508000 seconds user
       0.113744000 seconds sys
```

记录采样信息

```
perf record python3 testevt.py
```

```
Computation completed. Result sum: 249625070.7951043
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.071 MB perf.data (1409 samples) ]
```

打印数据

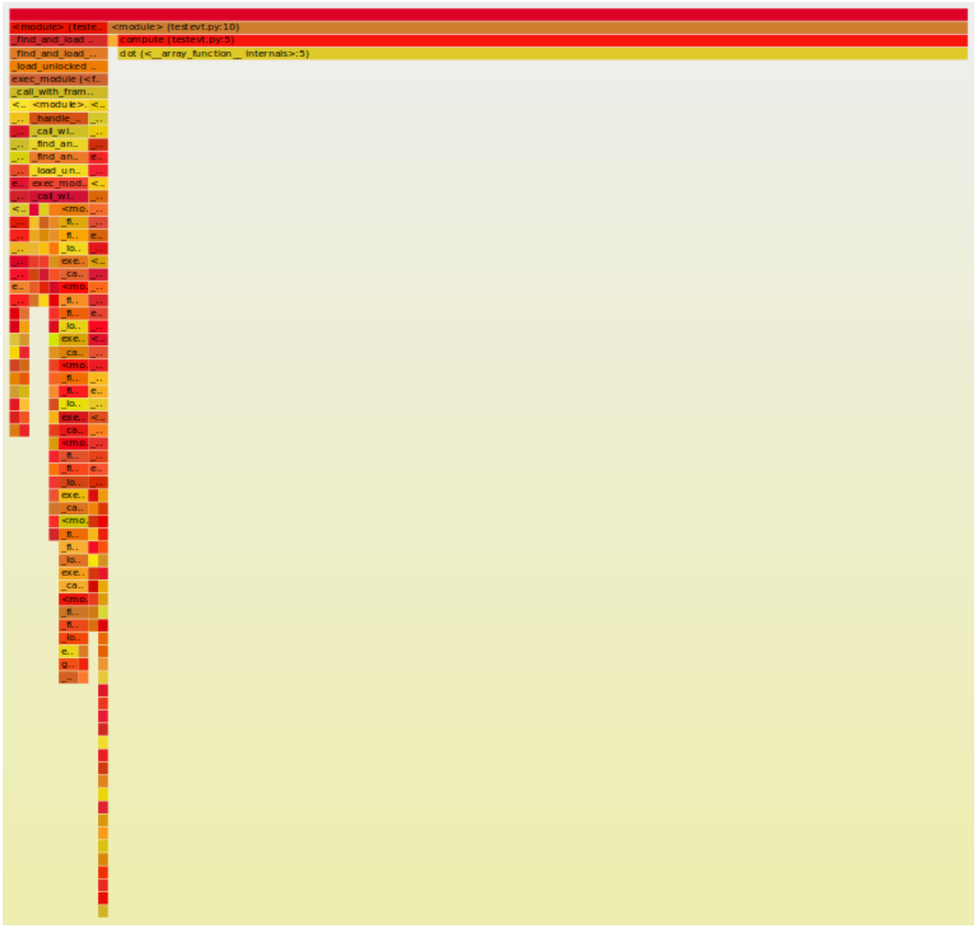
```
perf report
```

```
Samples: 1K of event 'cpu-clock:pppH', Event count (approx.): 352250000
Overhead  Command  Shared Object                                  Symbol
  78.00%  python3  libblas.so.3.9.0                               [.] dgemm_
   6.60%  python3  [kernel.kallsyms]                             [k] clear_page_orig
   1.21%  python3  python3.8                                     [.] _PyEval_EvalFrameDefault
   1.06%  python3  [kernel.kallsyms]                             [k] exit_to_user_mode_prepare
   0.71%  python3  python3.8                                     [.] 0x000000000013d018
   0.28%  python3  [kernel.kallsyms]                             [k] copy_user_generic_unrolled
   0.28%  python3  python3.8                                     [.] PyDict_SetDefault
   0.28%  python3  python3.8                                     [.] 0x00000000001d3d09
   0.21%  python3  ld-2.31.so                                    [.] _dl_relocate_object
   0.14%  python3  [kernel.kallsyms]                             [k] __d_lookup_rcu
   0.14%  python3  [kernel.kallsyms]                             [k] do_user_addr_fault
   0.14%  python3  libc-2.31.so                                  [.] __memmove_avx_unaligned_erms
   0.14%  python3  libc-2.31.so                                  [.] __memset_avx2_unaligned_erms
   0.14%  python3  libc-2.31.so                                  [.] _int_malloc
   0.14%  python3  libc-2.31.so                                  [.] malloc
   0.14%  python3  mt19937.cpython-38-x86_64-linux-gnu.so        [.] mt19937_gen
   0.14%  python3  mt19937.cpython-38-x86_64-linux-gnu.so        [.] 0x0000000000007f7c
   0.14%  python3  python3.8                                     [.] PyObject_GC_Del
   0.14%  python3  python3.8                                     [.] PyObject_SetAttr
   0.14%  python3  python3.8                                     [.] PyTraceBack_Here
   0.14%  python3  python3.8                                     [.] PyTuple_ClearFreeList
   0.14%  python3  python3.8                                     [.] PyType_Ready
   0.14%  python3  python3.8                                     [.] 0x00000000000ef574
   0.14%  python3  python3.8                                     [.] 0x000000000011865c
   0.14%  python3  python3.8                                     [.] 0x000000000013cf28
   0.14%  python3  python3.8                                     [.] 0x00000000001bb0b1
   0.14%  python3  python3.8                                     [.] 0x00000000001d2a8b
   0.14%  python3  python3.8                                     [.] 0x00000000001d3f7f
   0.07%  python3  [kernel.kallsyms]                             [k] __handle_mm_fault
   0.07%  python3  [kernel.kallsyms]                             [k] __softirqentry_text_start
   0.07%  python3  [kernel.kallsyms]                             [k] _find_next_bit
```

1.3.5 可视化

py-spy 生成火焰图

```
# 安装 py-spy
pip install py-spy
# 运行程序并生成火焰图
py-spy record -o profile.svg -- python3 testevt.py
```



火焰图

pycallgraph 生成调用图

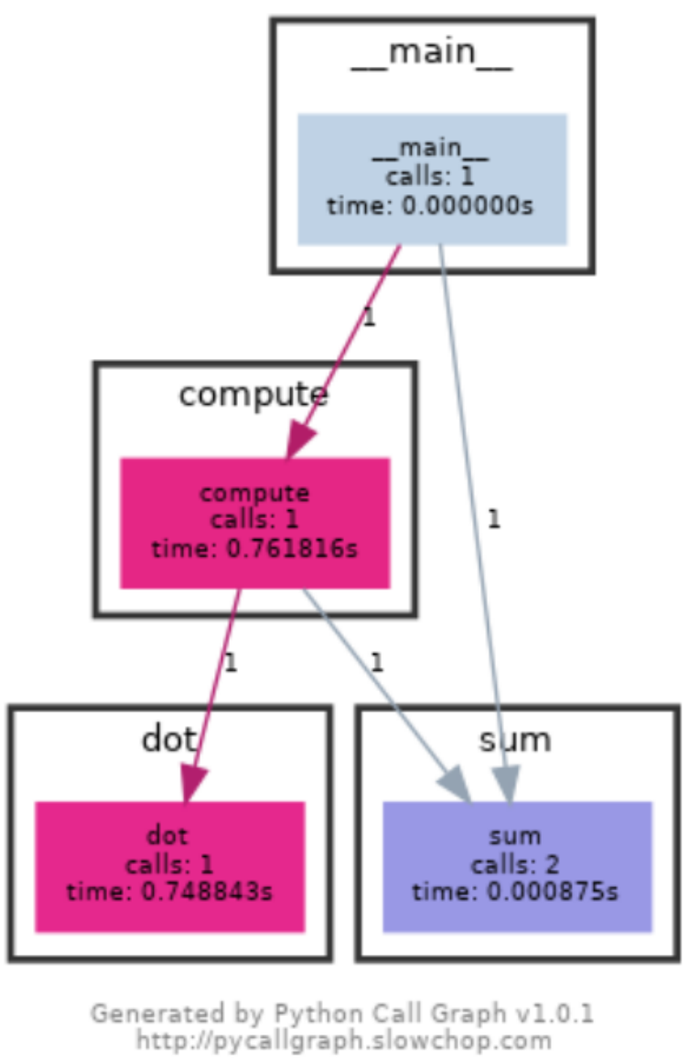
```
# 安装 pycallgraph 和 Graphviz
pip install pycallgraph
sudo apt-get install graphviz
```

```
# 生成调用图
from pycallgraph import PyCallGraph
from pycallgraph.output import GraphvizOutput
import numpy as np

def compute():
    matrix = np.random.rand(1000, 1000)
    result = np.dot(matrix, matrix)
    row_sums = np.sum(result, axis=1)
    return row_sums

if __name__ == '__main__':
    graphviz = GraphvizOutput()
    graphviz.output_file = 'callgraph.png'

    with PyCallGraph(output=graphviz):
        result = compute()
        print("Computation completed. Result sum:", np.sum(result))
```



调用图

2 元编程

2.1 构建系统

```
# Makefile
paper.pdf: paper.tex plot-data.png
    pdflatex paper.tex

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

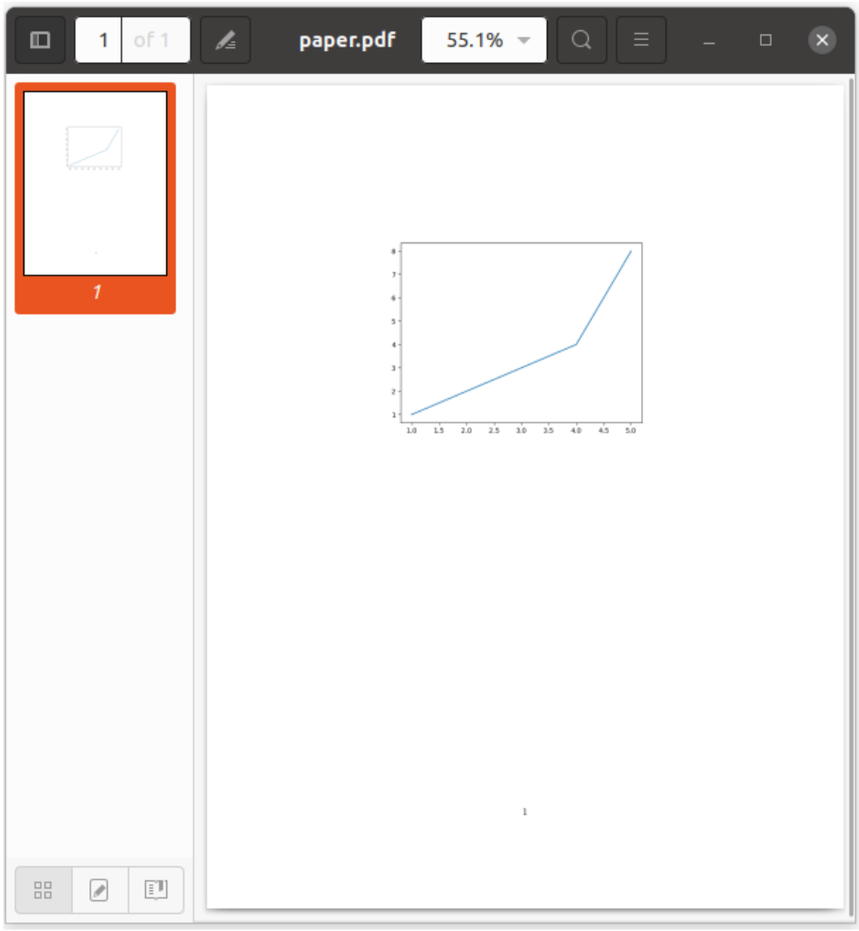
```
# paper.tex
\documentclass{article}
\usepackage{graphicx}
\begin{document}
\includegraphics[scale=0.65]{plot-data.png}
\end{document}
```

```
# plot.py
#!/usr/bin/env python3
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)
```

```
# data.dat
1 1
2 2
3 3
4 4
5 8
```



3 PyTorch

3.1 张量

3.1.1 创建张量

Pytorch 库的 Tensors（张量）对应 NumPy 库的多维数组（ndarrays）

```
import torch
# 直接创建张量
x = torch.tensor([1.0, 2.0, 3.0])
print("x:", x)
# 复制张量
y = x.clone()
print("y:", y)
# 随机生成5*3的张量
rand_x = torch.rand(5, 3)
print("rand_x:", rand_x)
```

```
x: tensor([1., 2., 3.])
y: tensor([1., 2., 3.])
rand_x: tensor([[0.6925, 0.4748, 0.9268],
        [0.3164, 0.6203, 0.6774],
        [0.2891, 0.8857, 0.3549],
        [0.7368, 0.7631, 0.5848],
        [0.5198, 0.6783, 0.2744]])
```

3.1.2 张量运算

PyTorch 库支持张量的多种运算，包括加减乘除、矩阵乘法、求和、极值、均值等。

```python
import torch

a = torch.tensor([1.0, 2.0, 3.0])
b = torch.tensor([4.0, 5.0, 6.0])
# 加
add = a + b
print("加:", add)
# 减
sub = a - b
print("减:", sub)
# 乘
mul = a * b
print("乘:", mul)
# 除
div = a / b
print("除:", div)
# 矩阵乘
mat1 = torch.tensor([[1, 2], [3, 4]])
mat2 = torch.tensor([[5, 6], [7, 8]])
matmul = torch.matmul(mat1, mat2)
print("mat1 * mat2:\n", matmul)
# 求和
sum = torch.sum(a)
print("求和:", sum)
# 均值
mean = torch.mean(a)
print("均值:", mean)
# 最大值
max = torch.max(a)
print("最大值:", max)
# 最小值
min = torch.min(a)
print("最小值:", min)
```

```
加: tensor([5., 7., 9.])
减: tensor([-3., -3., -3.])
乘: tensor([ 4., 10., 18.])
除: tensor([0.2500, 0.4000, 0.5000])
mat1 * mat2:
 tensor([[19, 22],
        [43, 50]])
求和: tensor(6.)
均值: tensor(2.)
最大值: tensor(3.)
最小值: tensor(1.)
```

3.2 梯度

3.2.1 张量的梯度

```python
import torch

x = torch.tensor([2.0, 3.0], requires_grad=True)
# 定义函数
```

```
y = x + 1
print("y:", y)
z = y * y * 3
print("z:", z)
out = z.mean()
print("out:", out)
# 计算梯度
out.backward()
print("grad:", x.grad)
```

```
y: tensor([3., 4.], grad_fn=<AddBackward0>)
z: tensor([27., 48.], grad_fn=<MulBackward0>)
out: tensor(37.5000, grad_fn=<MeanBackward0>)
grad: tensor([ 9., 12.])
```

3.2.2 标量的梯度

在尝试各种梯度计算时，我发现一个神奇的的现象，torch 可以计算标量的梯度。

```
import torch

x = torch.tensor([2.0, 3.0], requires_grad=True)
y = x[0]**2 + x[1]**3
print("y:", y)
y.backward()
print("grad:", x.grad)
```

```
y: tensor(31., grad_fn=<AddBackward0>)
grad: tensor([ 4., 27.])
```

可以发现，backward() 方法计算出了 y 中 x 的梯度。再看对 y 的输出，y 被认为是一个张量，"grad_fn=<AddBackward0>" 表明 y 是一个由加法操作生成的张量，并且可以追踪其梯度。PyTorch 使用自动微分来计算这些梯度，而在自动微分中，标量函数的梯度是相对于其输入变量的偏导数。

3.3 神经网络

3.3.1 定义网络

定义一个 5 层的卷积神经网络，包含两层卷积层和三层全连接层。

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 输入图像是单通道，conv1 kenrnel size=5*5，输出通道 6
        self.conv1 = nn.Conv2d(1, 6, 5)
        # conv2 kernel size=5*5，输出通道 16
        self.conv2 = nn.Conv2d(6, 16, 5)
        # 全连接层
        self.fc1 = nn.Linear(16*5*5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

```
        def forward(self, x):
            # max-pooling 采用一个 (2,2) 的滑动窗口
            x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
            # 核(kernel)大小是方形的话，可仅定义一个数字，如 (2,2) 用 2 即可
            x = F.max_pool2d(F.relu(self.conv2(x)), 2)
            x = x.view(-1, self.num_flat_features(x))
            x = F.relu(self.fc1(x))
            x = F.relu(self.fc2(x))
            x = self.fc3(x)
            return x

        def num_flat_features(self, x):
            # 除了 batch 维度外的所有维度
            size = x.size()[1:]
            num_features = 1
            for s in size:
                num_features *= s
            return num_features

    net = Net()
    print(net)
```

```
    # 输出其网络结构
    Net(
      (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
      (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
      (fc1): Linear(in_features=400, out_features=120, bias=True)
      (fc2): Linear(in_features=120, out_features=84, bias=True)
      (fc3): Linear(in_features=84, out_features=10, bias=True)
    )
    tensor([[ 0.0707,  0.1551,  0.0098,  0.0912, -0.0992, -0.0113,
        -0.0362,  0.0533,-0.0310,  0.0893]], grad_fn=<AddmmBackward0>)
```

3.3.2 损失函数

损失函数的输入是 (output, target)，即网络输出和真实标签对的数据，然后返回一个数值表示网络输出和真实标签的差距。

```
output = net(input)
    # 定义伪标签
    target = torch.randn(10)
    # 调整大小，使得和 output 一样的 size
    target = target.view(1, -1)
    criterion = nn.MSELoss()

    loss = criterion(output, target)
    print(loss)
```

```
    tensor(1.2806, grad_fn=<MseLossBackward0>)
```

3.3.3 反向传播

反向传播的实现只需要调用 loss.backward() 即可，当然首先需要清空当前梯度缓存，即.zero_grad() 方法，否则之前的梯度会累加到当前的梯度，这样会影响权值参数的更新。

```
# 清空所有参数的梯度缓存
net.zero_grad()
print('conv1.bias.grad before backward')
print(net.conv1.bias.grad)

loss.backward()

print('conv1.bias.grad after backward')
print(net.conv1.bias.grad)
```

```
conv1.bias.grad before backward
None
conv1.bias.grad after backward
tensor([-0.0116,  0.0042,  0.0014, -0.0101,  0.0048,  0.0025])
```

3.3.4 更新权重

随机梯度下降 Stochastic Gradient Descent

```
weight = weight - learning_rate * gradient
```

```
# 简单权重更新
learning_rate = 0.01
for f in net.parameters():
    f.data.sub_(f.grad.data * learning_rate)
```

torch.optim 库提供的优化算法

```
import torch.optim as optim
# 创建优化器
optimizer = optim.SGD(net.parameters(), lr=0.01)

# 在训练过程中执行下列操作
optimizer.zero_grad() # 清空梯度缓存
output = net(input)
loss = criterion(output, target)
loss.backward()
# 更新权重
optimizer.step()
```

4 实验心得

通过本次实验学习了 Linux 下调试及性能分析相关的工具，对之后学习中的代码调试、优化等有所帮助。此外，学习使用了 PyTorch 相关的深度学习库，对神经网络等技术的本质有了更多的了解。

5 Github 链接

```
https://github.com/Starry-Sky-OUC/gitTest
```