

# 系统开发工具基础第四次实验报告

张烨 23020007162

2024 年 9 月 14 日

## 目录

一、 实验目的	3
二、 实验内容	3
(一) 调试及性能分析	3
1. python 调试器——pdb	3
2. 高阶 pdb	7
3. 对插入排序和快速排序的性能分析方法 1	7
4. 对插入排序和快速排序的性能分析方法 2	8
(二) 元编程	9
(三) 大杂烩	9
1. GitHub 创建一个议题	9
2. GitHub 拉取请求提交代码更改	10
(四) pytorch	12
1. 张量的声明和定义	12
2. 加法运算操作的三种实现方法	14
3. Tensor 转换为 Numpy 数组	15
4. Numpy 数组转换为 Tensor	16
5. 梯度	16
6. 定义网络	16
7. 损失函数	17
8. 反向传播	18
9. 更新权重	18

目录	2
10. 加载和归一化 CIFAR10 . . . . .	19
11. 构建一个卷积神经网络 . . . . .	19
12. 定义损失函数和优化器 . . . . .	20
13. 训练网络 . . . . .	20
14. 测试 . . . . .	21
三、 心得体会	23
四、 相关练习、报告和代码查看链接	23

## 一、实验目的

1. 调试及性能分析、元编程、大杂烩。
- 2.pytorch 入门，了解基本内容。

## 二、实验内容

### (一) 调试及性能分析

#### 1.python 调试器——pdb

pdb 是 python 的调试器。

(1) l(list) - 展示当前行附近的 11 行，或者是继续之前的展示。

首先在 pdb 中输入 help l, 就可以查看 l(list) 的详细解释：

```
(Pdb) help l
l(list) [first [,last] | .]

List source code for the current file.  Without arguments,
list 11 lines around the current line or continue the previous
listing.  With . as argument, list 11 lines around the current
line.  With one argument, list 11 lines starting at that line.
With two arguments, list the given range; if the second
argument is less than the first, it is a count.

The current line in the current frame is indicated by "-->".
If an exception is being debugged, the line where the
exception was originally raised or propagated is indicated by
">>", if it differs from the current line.

(Pdb) |
```

图 1: pdb——l(list)

在 pdb 调试窗口中，输入 l 可以看到：

```
(Pdb) l
4      def main():
5          print("Add the values of the dice")
6          print("It's really that easy")
7          print("What are you doing with your life.")
8          import pdb;pdb.set_trace() # add pdb here
9  -->      GameRunner.run()
10
11
12      if __name__ == "__main__":
13          main()
[EOF]
(Pdb) |
```

图 2: pdb——l(list)

如果想看整个文件，可以给 list 命令加上范围参数，比如我现在这个文件，一共有 13 行，我就可以输入 l 1,13:

```
(Pdb) l 1, 13
1      from dicegame.runner import GameRunner
2
3
4      def main():
5          print("Add the values of the dice")
6          print("It's really that easy")
7          print("What are you doing with your life.")
8          import pdb;pdb.set_trace() # add pdb here
9      ->  GameRunner.run()
10
11
12      if __name__ == "__main__":
13          main()
(Pdb)
```

图 3: pdb——l(list)

(2) s(tep) - 执行当前行，会停在第一个可能停下的地方。  
调用 step 命令看到:

```
(Pdb) s
--Call--
> c:\users\zy\pdb-tutorial\dicegame\runner.py(21)run()
-> @classmethod
(Pdb) █
```

图 4: pdb——s(tep)

下面用 list 命令来查看前后段:

```
(Pdb) l
16          total = 0
17          for die in self.dice:
18              total += 1
19          return total
20
21      ->  @classmethod
22          def run(cls):
23              # Probably counts wins or something.
24              # Great variable name, 10/10.
25              c = 0
26              while True:
(Pdb)
```

图 5: pdb——l(list)

再次执行 step 命令来进入到这个方法内部，然后使用 list 命令查看当前位置:

```
(Pdb) s
> c:\users\zy\pdb-tutorial\dicegame\runner.py(25)run()
-> c = 0
(Pdb) l
20
21     @classmethod
22     def run(cls):
23         # Probably counts wins or something.
24         # Great variable name, 10/10.
25     ->     c = 0
26         while True:
27             runner = cls()
28
29             print("Round {}".format(runner.round))
30
(Pdb)
```

图 6: pdb

(3) `n(ext)` - 继续执行程序到当前函数的下一行或者是到它返回结果。  
下面再输入 `next` 和 `list` 命令：

```
(Pdb) n
> c:\users\zy\pdb-tutorial\dicegame\runner.py(26)run()
-> while True:
(Pdb) l
21     @classmethod
22     def run(cls):
23         # Probably counts wins or something.
24         # Great variable name, 10/10.
25         c = 0
26     ->     while True:
27         runner = cls()
28
29         print("Round {}".format(runner.round))
30
31         for die in runner.dice:
(Pdb)
```

图 7: pdb——`n(ext)`

发现停在 `while True` 语句，下面继续调用 `next` 命令知道程序抛出异常或结束。

```

(Pdb) n
> c:\users\zy\pdb-tutorial\dicegame\runner.py(27)run()
-> runner = cls()
(Pdb) n
> c:\users\zy\pdb-tutorial\dicegame\runner.py(29)run()
-> print("Round {}\n".format(runner.round))
(Pdb) n
Round 1

> c:\users\zy\pdb-tutorial\dicegame\runner.py(31)run()
-> for die in runner.dice:
(Pdb) l
26         while True:
27             runner = cls()
28
29             print("Round {}\n".format(runner.round))
30
31 ->         for die in runner.dice:
32             print(die.show())
33
34             guess = input("Sigh. What is your guess?: ")
35             guess = int(guess)
36
(Pdb) len(runner.dice)
5
(Pdb)

```

图 8: pdb——n(ext)

使用 len 方法得到 runner.dice 的长度为 5。

(4) b(reak) - 设置程序断点 (取决于提供的参数)。例子:

```

(Pdb) b 34
Breakpoint 1 at c:\users\zy\pdb-tutorial\dicegame\runner.py:34

```

图 9: pdb——b(reak)

(5) r(eturn) - 继续执行到当前函数返回结果。

return 可以直接检查函数的最终返回结果，下面是操作:

```

(Pdb) r
Sigh. What is your guess?: 5
Congrats, you can add like a 5 year old...
Wins: 1 Loses 0
Would you like to play again?[Y/n]: n
--Return--
> c:\users\zy\pdb-tutorial\dicegame\runner.py(60)run()->None
-> i_just_throw_an_exception()

```

图 10: pdb——r(eturn)

## 2. 高阶 pdb

(1) ! 命令

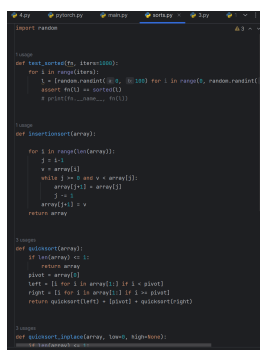
! 命令是在告诉 pdb 接下来的语句是 python 命令而不是 pdb 命令。

(2) post\_mortem() 和 pm()

post\_mortem() 在异常处理时提供了更直接的调试体验，而 pm() 有不同的实现细节。

## 3. 对插入排序和快速排序的性能分析方法 1

首先添加好 sorts.py 文件，如图所示：



```
def bubble_sort(arr):
    for i in range(len(arr)-1):
        for j in range(i+1, len(arr)):
            if arr[i] > arr[j]:
                arr[i], arr[j] = arr[j], arr[i]
    return arr

def insertion_sort(arr):
    for i in range(1, len(arr)):
        j = i
        while j > 0 and arr[j] < arr[j-1]:
            arr[j], arr[j-1] = arr[j-1], arr[j]
            j -= 1
        arr[j] = arr[i]
    return arr

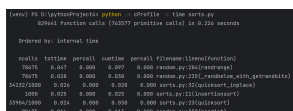
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[0]
    left = [x for x in arr[1:] if x < pivot]
    right = [x for x in arr[1:] if x >= pivot]
    return quicksort(left) + [pivot] + quicksort(right)

def quicksort_insert(arr, low, high):
    if low < high:
        p = partition(arr, low, high)
        quicksort_insert(arr, low, p-1)
        quicksort_insert(arr, p+1, high)
```

图 11: sorts.py

使用 cProfile 进行性能分析：

在终端输入：python -m cProfile -s time sorts.py



```
python -m cProfile -s time sorts.py
python function calls (bottom precision call) in 0.00 seconds

Ordered by: Internal time

ncalls  tottime  percall  filename:line(function)
100000  0.000    0.000    0.000 random.py:261(random)
100000  0.000    0.000    0.000 random.py:261(random)
100000  0.000    0.000    0.000 random.py:261(random)
100000  0.000    0.000    0.000 random.py:261(random)
100000  0.000    0.000    0.000 random.py:261(random)
100000  0.000    0.000    0.000 random.py:261(random)
100000  0.000    0.000    0.000 random.py:261(random)
100000  0.000    0.000    0.000 random.py:261(random)
100000  0.000    0.000    0.000 random.py:261(random)
100000  0.000    0.000    0.000 random.py:261(random)
```

图 12: cProfile

由于太长了，这里只截取部分。

要使用line\_profiler进行分析,首先在终端输入pip install line\_profiler下载：

然后在 quicksort 函数和 insertionsort 函数上用 @profile 标记，然后执行kernprof -l -v sorts.py语句就可以看到：

```
(venv) PS D:\pythonProject4> kernprof -l -v sorts.py
Wrote profile results to sorts.py.lprof
Timer unit: 1e-06 s

Total time: 0.148451 s
File: sorts.py
Function: insertionsort at line 10

Line #      Hits          Time Per Hit   % Time  Line Contents
=====
10             @profile
11             def insertionsort(array):
12
13      25950          4402.1     0.2     3.0      for i in range(len(array)):
14      24950          4922.5     0.2     3.3          j = i-1
15      24950          4348.1     0.2     2.9          v = array[i]
16     230384         52197.6     0.2    35.2          while j >= 0 and v < array[j]:
17     205434         41707.4     0.2    28.1              array[j+1] = array[j]
18     205434         35569.8     0.2    24.0              j -= 1
19      24950         5131.4     0.2     3.5              array[j+1] = v
20      1000           172.5     0.2     0.1          return array
```

图 13: insertionsort

```
Total time: 0.0851783 s
File: sorts.py
Function: quicksort at line 22

Line #      Hits          Time Per Hit   % Time  Line Contents
=====
22             @profile
23             def quicksort(array):
24      33490          8051.0     0.2     9.5      if len(array) <= 1:
25      17245          2602.1     0.2     3.1          return array
26      16245          2839.3     0.2     3.3          pivot = array[0]
27      16245         27488.5     1.7    32.3          left = [i for i in array[1:] if i < pivot]
28      16245         27170.8     1.7    31.9          right = [i for i in array[1:] if i >= pivot]
29      16245         17026.6     1.0    20.0          return quicksort(left) + [pivot] + quicksort(right)
```

图 14: quicksort

可以看到插入排序更耗时。

#### 4. 对插入排序和快速排序的性能分析方法 2

下面使用memory\_profiler进行分析,首先使用pip install memory\_profiler语句来安装,然后输入python -m memory\_profiler sorts.py语句分析两个排序的内存使用情况:



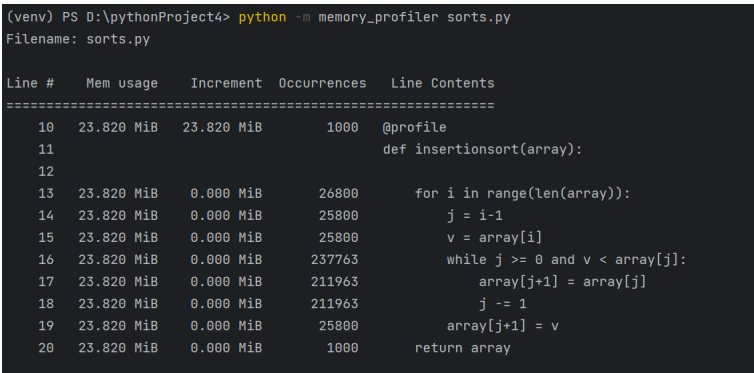


图 15: insertionsort

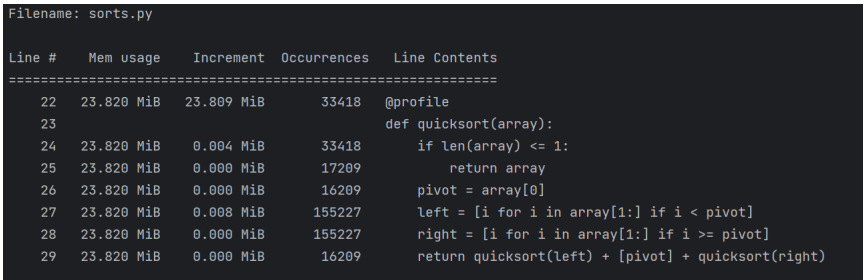


图 16: quicksort

(二) 元编程

(三) 大杂烩

1. GitHub 创建一个议题

下面是从仓库中创建一个议题：

首先到仓库下，点击 Issues：

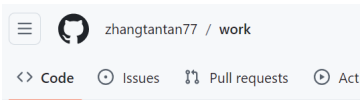


图 17: Issues

然后点击 New issue

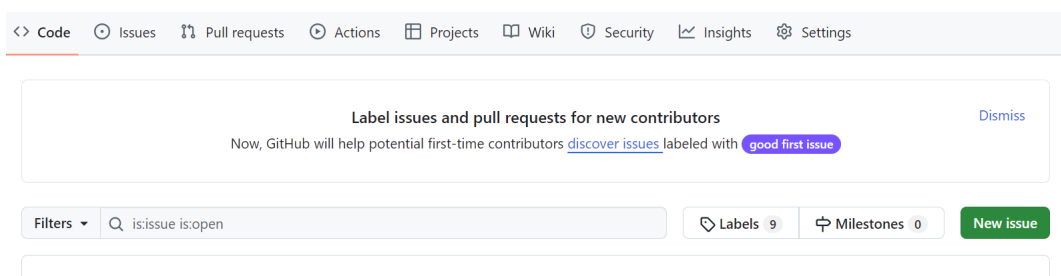


图 18: Issues

进去后填写标题和内容，然后点击 Submit new issue 就可以提交了。这样就成功创建了一个议题。

## 2. GitHub 拉取请求提交代码更改

在 github 上找到要修改的文件，点击右上角一个像铅笔的图标 (Edit this file)

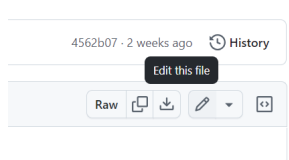


图 19: GitHub

进去后填写完更改说明后点击”Commit changes”

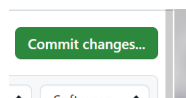
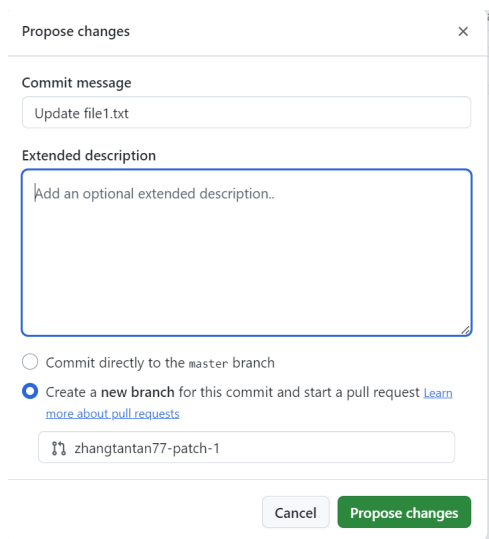


图 20: GitHub

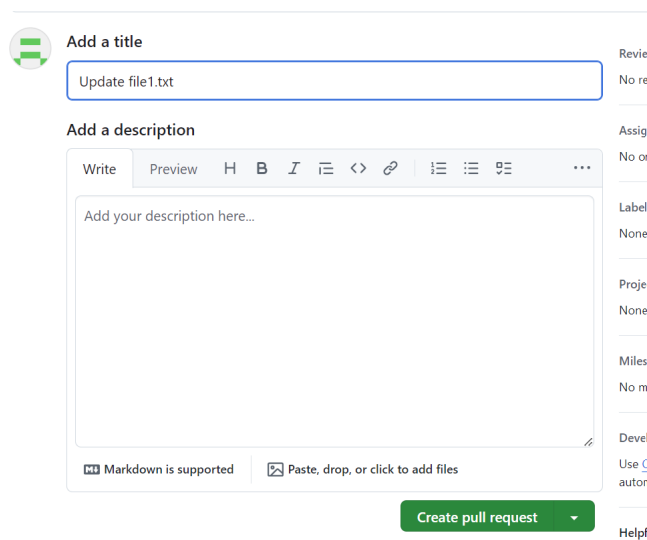
然后选择”Create a new branch for this commit and start a pull request”



The image shows a 'Propose changes' dialog box from GitHub. It has a title bar with a close button (X). The main content area includes a 'Commit message' field with the text 'Update file1.txt', an 'Extended description' text area with placeholder text 'Add an optional extended description..', and two radio buttons. The first radio button is 'Commit directly to the master branch'. The second radio button is selected and labeled 'Create a new branch for this commit and start a pull request', with a link 'Learn more about pull requests' next to it. Below the radio buttons is a text field containing the branch name 'zhangtantan77-patch-1'. At the bottom right, there are two buttons: 'Cancel' and 'Propose changes'.

图 21: GitHub

然后点击“Create pull request”



The image shows the 'Create pull request' form in GitHub. It has a title bar with a GitHub logo and a title 'Add a title'. Below the title bar is a text field with the text 'Update file1.txt'. To the right of the text field is a 'Review' button. Below the text field is a section titled 'Add a description' with a 'Write' tab and a 'Preview' tab. The 'Write' tab is active, showing a text area with placeholder text 'Add your description here...'. Below the text area are two buttons: 'Markdown is supported' and 'Paste, drop, or click to add files'. At the bottom right, there is a green button labeled 'Create pull request' with a dropdown arrow. To the right of the form is a sidebar with various options: 'Assign', 'Labels', 'Project', 'Milestones', 'Development', 'Use', and 'Help'.

图 22: GitHub

然后点击“Merge pull request”，然后点击“Confirm merge” 完成合并。

## Update file1.txt #1

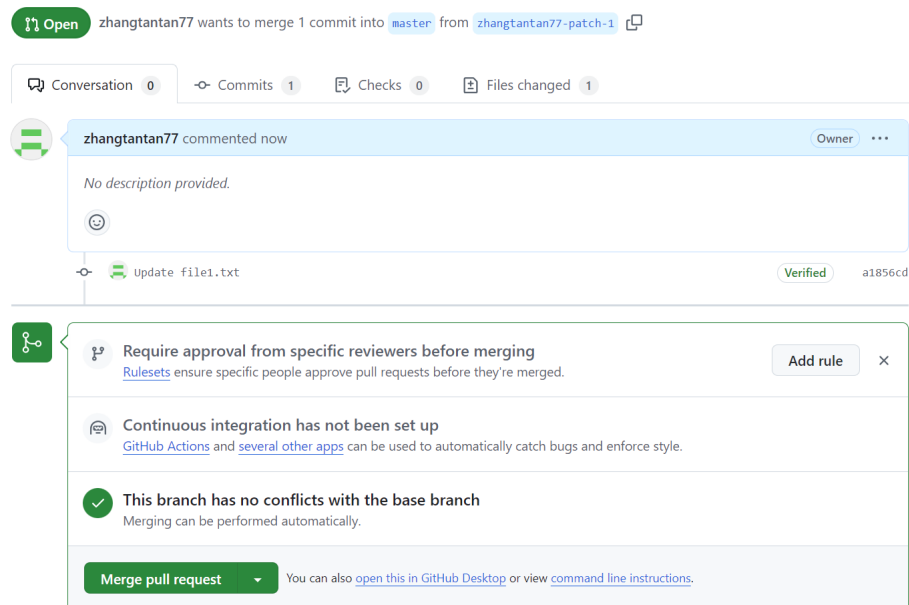


图 23: GitHub

## (四) pytorch

## 1. 张量的声明和定义

首先导入 torch 库:

```
from __future__ import print_function
import torch
```

图 24: 张量

1.torch.empty(): 声明一个未初始化的矩阵。例如, 创建一个 5\*3 的矩阵:

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

图 25: empty

```
tensor([[6.4046e+01, 8.6881e-43, 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]])
```

图 26: empty

2.torch.rand(): 随机初始化一个矩阵。例如：创建一个随机初始化的 5\*3 矩阵：

```
rand_x = torch.rand(5, 3)
print(rand_x)
```

图 27: rand

```
tensor([[0.3849, 0.3302, 0.9822],
        [0.7147, 0.3160, 0.8682],
        [0.9374, 0.7169, 0.9519],
        [0.8994, 0.3829, 0.8974],
        [0.9593, 0.8523, 0.4214]])
```

图 28: rand

3.torch.zeros(): 创建数值皆为 0 的矩阵。例如：创建一个数值皆是 0，类型为 long 的矩阵：

```
zero_x = torch.zeros(5, 3, dtype=torch.long)
print(zero_x)
```

图 29: zeros

```
tensor([[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]])
```

图 30: zeros

4.torch.tensor(): 直接传递 tensor 数值来创建。例如数值是 [5.5,3]:

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

图 31: tensor

```
tensor([5.5000, 3.0000])
```

图 32: tensor

5.tensor.new\_ones(): new\_\*() 方法需要输入尺寸大小。例如：显示定义新的尺寸是 5\*3，数值类型是 torch.double

```
tensor2 = tensor1.new_ones(5, 3, dtype=torch.double) # n  
print(tensor2)
```

图 33: new

```
tensor([[1., 1., 1.],  
        [1., 1., 1.],  
        [1., 1., 1.],  
        [1., 1., 1.],  
        [1., 1., 1.]], dtype=torch.float64)
```

图 34: new

6.torch.randn\_like(old\_tensor): 保留相同的尺寸大小。

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

图 35: randn

```
tensor3: tensor([[ 1.0903,  0.0773, -0.5029],  
                 [-0.7443,  0.1307, -1.6252],  
                 [ 0.2341,  0.8044, -1.6685],  
                 [ 0.8921,  0.6618, -1.4233],  
                 [-1.7011,  0.4179, -0.3639]])
```

图 36: randn

## 2. 加法运算操作的三种实现方法

对于加法的操作，有几种实现方式：

## 1.+ 运算符

2.torch.add(tensor1, tensor2, [out=tensor3])

3.tensor1.add\_(tensor2): 直接修改 tensor 变量

```
tensor4 = torch.rand(5, 3)
print('tensor3 + tensor4= ', tensor3 + tensor4)
print('tensor3 + tensor4= ', torch.add(tensor3, tensor4))
# 新声明一个 tensor 变量保存加法操作的结果
result = torch.empty(5, 3)
torch.add(tensor3, tensor4, out=result)
print('add result= ', result)
# 直接修改变量
tensor3.add_(tensor4)
print('tensor3= ', tensor3)
```

图 37: 加法

```
[ 0.3324, 0.4988, 1.1370]]
tensor3 + tensor4=  tensor([[ 0.7792,  1.1820,  2.8583],
 [-0.7121,  0.5061, -0.0921],
 [-0.1988, -0.2194,  0.6423],
 [ 1.0934,  0.5587,  0.5177],
 [ 0.5429,  0.3016,  1.5071]])
tensor3 + tensor4=  tensor([[ 0.7792,  1.1820,  2.8583],
 [-0.7121,  0.5061, -0.0921],
 [-0.1988, -0.2194,  0.6423],
 [ 1.0934,  0.5587,  0.5177],
 [ 0.5429,  0.3016,  1.5071]])
add result=  tensor([[ 0.7792,  1.1820,  2.8583],
 [-0.7121,  0.5061, -0.0921],
 [-0.1988, -0.2194,  0.6423],
 [ 1.0934,  0.5587,  0.5177],
 [ 0.5429,  0.3016,  1.5071]])
tensor3=  tensor([[ 0.7792,  1.1820,  2.8583],
 [-0.7121,  0.5061, -0.0921],
 [-0.1988, -0.2194,  0.6423],
 [ 1.0934,  0.5587,  0.5177],
 [ 0.5429,  0.3016,  1.5071]])
```

图 38: 加法

## 3.Tensor 转换为 Numpy 数组

实现 Tensor 转换为 Numpy 数组需要调用 tensor.numpy():

```
a = torch.ones(5)
print(a)
b = a.numpy()
print(b)
```

图 39: 转换

```
tensor([1., 1., 1., 1., 1.])  
[1. 1. 1. 1. 1.]
```

图 40: 转换

#### 4.Numpy 数组转换为 Tensor

转换的操作是调用 `torch.from_numpy(numpy_array)` 方法:

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

图 41: 转换

```
[2. 2. 2. 2. 2.]  
tensor([2., 2., 2., 2., 2.], dtype=torch.float64)
```

图 42: 转换

#### 5. 梯度

梯度用 `out.backward()`:

```
out.backward()  
# 输出梯度 d(out)/dx  
print(x.grad)
```

图 43: 梯度

```
tensor([[4.5000, 4.5000],  
        [4.5000, 4.5000]])
```

图 44: 梯度

#### 6. 定义网络

下面是一个 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
2 usages
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        # 输入图像是单通道, conv1 kernel size=5*5, 输出通道 6
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
        # conv2 kernel size=5*5, 输出通道 16
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        # 全连接层
        self.fc1 = nn.Linear(16*5*5, out_features=120)
        self.fc2 = nn.Linear(in_features=120, out_features=84)
        self.fc3 = nn.Linear(in_features=84, out_features=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)), kernel_size=(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)), kernel_size=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

1 usage
    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)

```

图 45: 网络

```

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)
)

```

图 46: 网络

## 7. 损失函数

下面是简单的均方误差:

```
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)
```

图 47: 均方误差

```
tensor(1.4683, grad_fn=<MseLossBackward0>)
```

图 48: 均方误差

## 8. 反向传播

```
# 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)
```

图 49: 反向传播

```
conv1.bias.grad before backward
None
conv1.bias.grad after backward
tensor([-0.0014,  0.0015,  0.0086,  0.0110,  0.0086,  0.0034])
```

图 50: 反向传播

## 9. 更新权重

采用随机梯度下降 (Stochastic Gradient Descent, SGD) 方法的最简单的更新权重规则如下:

$$\text{weight} = \text{weight} - \text{learning\_rate} * \text{gradient}$$

```
# 简单实现权重的更新例子
learning_rate = 0.01
for f in net.parameters():
    f.data.sub_(f.grad.data * learning_rate)
```

图 51: 更新权重

## 10. 加载和归一化 CIFAR10

首先下载在 pycharm 中下载 torchvision 库:

```
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np

def main():
    # 将图片数据从 [0,1] 归一化为 [-1,1] 的取值范围
    transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])])
    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                             download=True, transform=transform)
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                              shuffle=True, num_workers=2)
    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                             download=True, transform=transform)
    testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                              shuffle=False, num_workers=2)
    classes = ('plane', 'car', 'bird', 'cat',
               'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

    # 显示图片的函数
    def imshow(img):
        img = img / 2 + 0.5 # 归一化
        npimg = img.numpy()
        plt.imshow(np.transpose(npimg, (1, 2, 0)))
        plt.show()

    # 加载训练数据
    dataloader = iter(trainloader)
    images, labels = next(dataloader)
    # 显示图片
    imshow(torchvision.utils.make_grid(images))
    # 打印图片类别标签
    print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
if __name__ == '__main__':
    main()
```

图 52: 1

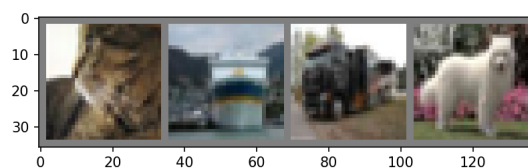


图 53: 1

## 11. 构建一个卷积神经网络

利用之前定义的网络, 修改 conv1 的输入通道, 从 1 变为 3:

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

2 usages

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d( in_channels: 3, out_channels: 6, kernel_size: 5)
        self.pool = nn.MaxPool2d( kernel_size: 2, stride: 2)
        self.conv2 = nn.Conv2d( in_channels: 6, out_channels: 16, kernel_size: 5)
        self.fc1 = nn.Linear(16 * 5 * 5, out_features: 120)
        self.fc2 = nn.Linear( in_features: 120, out_features: 84)
        self.fc3 = nn.Linear( in_features: 84, out_features: 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
```

图 54: 3

## 12. 定义损失函数和优化器

这里采用类别交叉熵函数和带有动量的 SGD 优化方法:

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

图 55: 损失函数和优化器

## 13. 训练网络

指定需要迭代的 epoch, 然后输入数据, 指定次数打印当前网络的信息, 现在假设每 2000 次迭代打印一次信息, 添加代码:

```
net: Net = Net().to(device)
start = time.time()
for epoch in range(2):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 2000 == 1999: # 调整为200以便更频繁打印
            print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, running_loss))
            running_loss = 0.0

print('Finished Training! Total cost time: ', time.time() - start)
```

图 56: 训练网络

```
[1, 2000] loss: 21.613
[1, 4000] loss: 18.176
[1, 6000] loss: 16.315
[1, 8000] loss: 15.900
[1, 10000] loss: 15.338
[1, 12000] loss: 14.904
[2, 2000] loss: 14.189
[2, 4000] loss: 14.073
[2, 6000] loss: 13.854
[2, 8000] loss: 13.445
[2, 10000] loss: 13.289
[2, 12000] loss: 13.105
Finished Training! Total cost time: 71.51499629020691
```

图 57: 训练网络

可以看到，耗时大概 71 秒。

## 14. 测试

首先先对四张图片进行测试：

添加代码：

```
# 展示测试集中的图片
dataiter = iter(testloader)
images, labels = next(dataiter)
# 展示测试图片
imshow(torchvision.utils.make_grid(images))
# 打印测试图片类别标签
print('Test GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
```

图 58: 测试

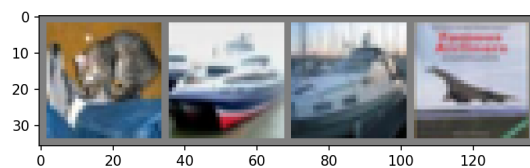


图 59: 测试

```
Test GroundTruth:   cat  ship  ship plane
```

图 60: 测试

看到这四张图片分别是:cat ship ship plane

下面用这四张图片输入网络，查看网络预测结果：

```
Predicted:   cat  car  car plane
```

图 61: 测试

可以看到，预测对了两个。

下面，查看整个测试集上的准确率。先添加下面代码：

```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data

        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (100 * correct / total))
```

图 62: 测试

```
Accuracy of the network on the 10000 test images: 53 %
```

图 63: 测试

可以看到，结果是 53%.

### 三、 心得体会

这次学习了调试和性能分析，了解了元编程，并且学习了大杂烩模块里面的内容。对于 pytorch，也有了一定的了解，并且能够完成一些简单操作，收获非常大。

### 四、 相关练习、报告和代码查看链接

本次报告相关练习、报告和代码均可以在 <https://kkgithub.com/zhangtantan77/work> 查看