第四周实验报告

22090032004 高晓琳

2024 年 9 月 15 日

目录

1 调试及性能分析 **1**

1.1 调试 1

1.1.1 显示日志 1

1.1.2 pdb 调试器 2

1.1.3 静态分析工具 2

1.1.4 journalctl 3

1.2 性能分析 3

1.2.1 计时 3

1.2.2 工具：CPU 3

1.2.3 工具：内存 4

1.2.4 工具对比 5

1.2.5 可视化 6

2 pytorch **8**

2.1 张量 8

2.2 操作 8

2.3 Tensor 和 Numpy 数组的转换 9

2.4 Autograd 10

2.4.1 变量 11

2.4.2 梯度 12

2.5 神经网络 12

2.6 一个训练神经网络的运行尝试 14

2.6.1 数据预处理 14

2.6.2 定义卷积神经网络 15

2.6.3 定义一个 Loss 函数和优化器 16

2.6.4 训练网络 16

2.6.5 测试数据 18

3 总结与心得 **18**

1 调试及性能分析

1.1 调试

1.1.1 显示日志

```
lin@xiaolin:/mnt/c/Users/lin/Downloads$ python3 logger.py color
2024-09-15 14:24:53,764 - Sample - CRITICAL - Maximum value reached (logger.py:64)
2024-09-15 14:24:54,065 - Sample - ERROR - Value is 8 - Dangerous region (logger.py:62)
2024-09-15 14:24:54,366 - Sample - INFO - Value is 2 - Everything is fine (logger.py:58)
2024-09-15 14:24:54,667 - Sample - ERROR - Value is 8 - Dangerous region (logger.py:62)
2024-09-15 14:24:54,967 - Sample - CRITICAL - Maximum value reached (logger.py:64)
2024-09-15 14:24:55,268 - Sample - ERROR - Value is 8 - Dangerous region (logger.py:62)
2024-09-15 14:24:55,569 - Sample - INFO - Value is 2 - Everything is fine (logger.py:58)
2024-09-15 14:24:55,869 - Sample - WARNING - Value is 5 - System is getting hot (logger.py:60)
2024-09-15 14:24:56,170 - Sample - INFO - Value is 1 - Everything is fine (logger.py:58)
2024-09-15 14:24:56,471 - Sample - CRITICAL - Maximum value reached (logger.py:64)
2024-09-15 14:24:56,772 - Sample - CRITICAL - Maximum value reached (logger.py:64)
2024-09-15 14:24:57,073 - Sample - ERROR - Value is 8 - Dangerous region (logger.py:62)
2024-09-15 14:24:57,374 - Sample - INFO - Value is 0 - Everything is fine (logger.py:58)
2024-09-15 14:24:57,675 - Sample - CRITICAL - Maximum value reached (logger.py:64)
2024-09-15 14:24:57,976 - Sample - ERROR - Value is 7 - Dangerous region (logger.py:62)
2024-09-15 14:24:58,277 - Sample - ERROR - Value is 7 - Dangerous region (logger.py:62)
2024-09-15 14:24:58,577 - Sample - INFO - Value is 4 - Everything is fine (logger.py:58)
2024-09-15 14:24:58,878 - Sample - WARNING - Value is 5 - System is getting hot (logger.py:60)
2024-09-15 14:24:59,180 - Sample - ERROR - Value is 7 - Dangerous region (logger.py:62)
2024-09-15 14:24:59,480 - Sample - INFO - Value is 4 - Everything is fine (logger.py:58)
2024-09-15 14:24:59,781 - Sample - CRITICAL - Maximum value reached (logger.py:64)
2024-09-15 14:25:00,082 - Sample - CRITICAL - Maximum value reached (logger.py:64)
2024-09-15 14:25:00,383 - Sample - WARNING - Value is 5 - System is getting hot (logger.py:60)
2024-09-15 14:25:00,684 - Sample - CRITICAL - Maximum value reached (logger.py:64)
2024-09-15 14:25:00,984 - Sample - INFO - Value is 3 - Everything is fine (logger.py:58)
2024-09-15 14:25:01,285 - Sample - WARNING - Value is 6 - System is getting hot (logger.py:60)
2024-09-15 14:25:01,587 - Sample - INFO - Value is 1 - Everything is fine (logger.py:58)
2024-09-15 14:25:01,887 - Sample - CRITICAL - Maximum value reached (logger.py:64)
2024-09-15 14:25:02,188 - Sample - CRITICAL - Maximum value reached (logger.py:64)
```

图 1: 打印日志

1.1.2 pdb 调试器

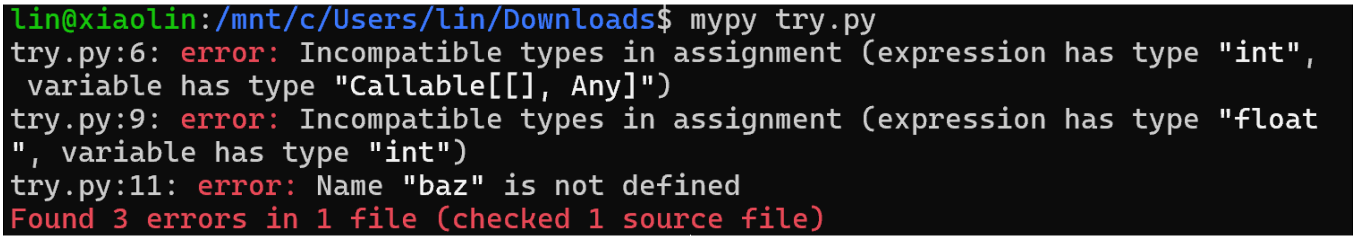
```
lin@xiaolin:/mnt/c/Users/lin/Downloads$ python3 -m pdb try.py
> /mnt/c/Users/lin/Downloads/try.py(1)<module>()
-> def bubble_sort(arr):
(Pdb) l
  1  -> def bubble_sort(arr):
  2         n = len(arr)
  3         for i in range(n):
  4             for j in range(n):
  5                 if arr[j] > arr[j+1]:
  6                     arr[j] = arr[j+1]
  7                     arr[j+1] = arr[j]
  8         return arr
  9
 10     print(bubble_sort([4, 2, 1, 8, 7, 6]))
[EOF]
(Pdb) b 10
Breakpoint 1 at /mnt/c/Users/lin/Downloads/try.py:10
(Pdb) c
> /mnt/c/Users/lin/Downloads/try.py(10)<module>()
-> print(bubble_sort([4, 2, 1, 8, 7, 6]))
(Pdb) c
Traceback (most recent call last):
  File "/usr/lib/python3.10/pdb.py", line 1723, in main
    pdb._runscript(mainpyfile)
  File "/usr/lib/python3.10/pdb.py", line 1583, in _runscript
    self.run(statement)
  File "/usr/lib/python3.10/bdb.py", line 598, in run
    exec(cmd, globals, locals)
  File "<string>", line 1, in <module>
  File "/mnt/c/Users/lin/Downloads/try.py", line 10, in <module>
    print(bubble_sort([4, 2, 1, 8, 7, 6]))
  File "/mnt/c/Users/lin/Downloads/try.py", line 5, in bubble_sort
    if arr[j] > arr[j+1]:
IndexError: list index out of range
Uncaught exception. Entering post mortem debugging
Running 'cont' or 'step' will restart the program
> /mnt/c/Users/lin/Downloads/try.py(5)bubble_sort()
-> if arr[j] > arr[j+1]:
(Pdb)
```

图 2: 调试冒泡程序

1.1.3 静态分析工具

```
lin@xiaolin:/mnt/c/Users/lin/Downloads$ pyflakes3 try.py
try.py:6:5 redefinition of unused 'foo' from line 3
try.py:11:7 undefined name 'baz'
```

图 3: pyflakes

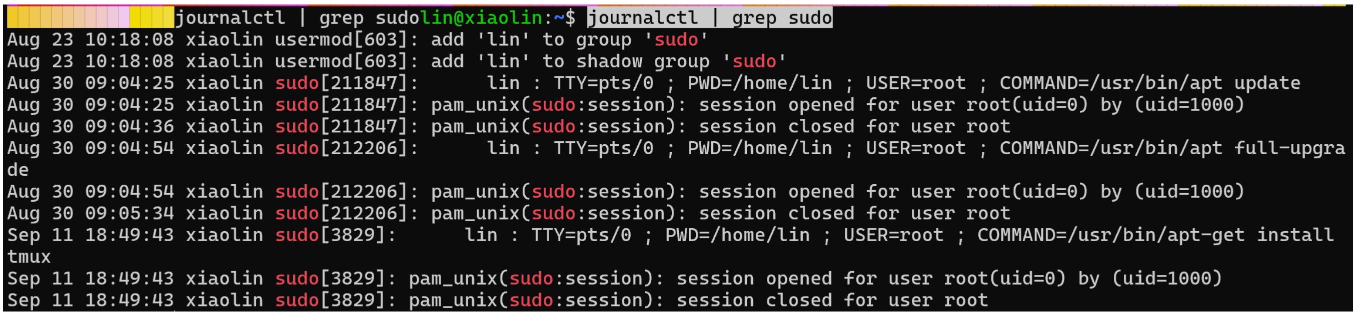


```
lin@xiaolin:/mnt/c/Users/lin/Downloads$ mypy try.py
try.py:6: error: Incompatible types in assignment (expression has type "int",
 variable has type "Callable[[], Any]")
try.py:9: error: Incompatible types in assignment (expression has type "float
", variable has type "int")
try.py:11: error: Name "baz" is not defined
Found 3 errors in 1 file (checked 1 source file)
```

图 4: mypy

1.1.4 journalctl

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。



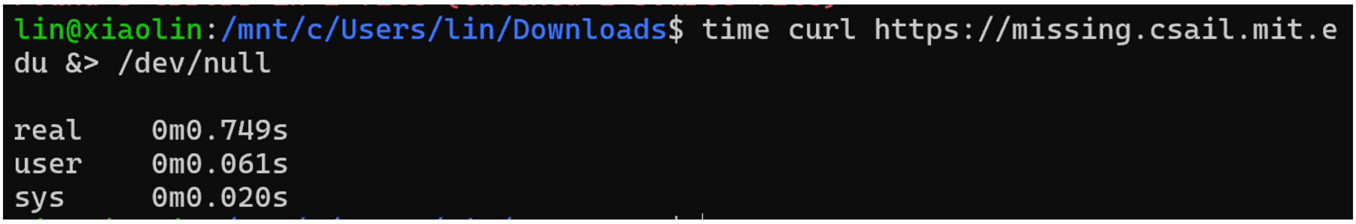
```
journalctl | grep sudolin@xiaolin:~$ journalctl | grep sudo
Aug 23 10:18:08 xiaolin usermod[603]: add 'lin' to group 'sudo'
Aug 23 10:18:08 xiaolin usermod[603]: add 'lin' to shadow group 'sudo'
Aug 30 09:04:25 xiaolin sudo[211847]:      lin : TTY=pts/0 ; PWD=/home/lin ; USER=root ; COMMAND=/usr/bin/apt update
Aug 30 09:04:25 xiaolin sudo[211847]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Aug 30 09:04:36 xiaolin sudo[211847]: pam_unix(sudo:session): session closed for user root
Aug 30 09:04:54 xiaolin sudo[212206]:      lin : TTY=pts/0 ; PWD=/home/lin ; USER=root ; COMMAND=/usr/bin/apt full-upgra
de
Aug 30 09:04:54 xiaolin sudo[212206]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Aug 30 09:05:34 xiaolin sudo[212206]: pam_unix(sudo:session): session closed for user root
Sep 11 18:49:43 xiaolin sudo[3829]:      lin : TTY=pts/0 ; PWD=/home/lin ; USER=root ; COMMAND=/usr/bin/apt-get install
tmux
Sep 11 18:49:43 xiaolin sudo[3829]: pam_unix(sudo:session): session opened for user root(uid=0) by (uid=1000)
Sep 11 18:49:43 xiaolin sudo[3829]: pam_unix(sudo:session): session closed for user root
```

图 5: journalctl 命令

1.2 性能分析

1.2.1 计时

大多数情况下只需要打印两处代码之间的时间即可发现问题。如可使用 Python 的 time 模块。但执行时间也可能会误导用户，因为电脑可能也在同时运行其他进程，也可能在此期间发生了等待。因此需要区分真实时间、用户时间和系统时间。如下图的例子。通常来说，用户时间 + 系统时间代表了进程所消耗的实际 CPU。



```
lin@xiaolin:/mnt/c/Users/lin/Downloads$ time curl https://missing.csail.mit.e
du &> /dev/null

real    0m0.749s
user    0m0.061s
sys     0m0.020s
```

图 6: 发起一个 http 请求的计时

1.2.2 工具：CPU

在 Python 中，我们使用 cProfile 模块来分析每次函数调用所消耗的时间，可以看出在该代码中，IO 消耗了大量的时间，编译正则表达式也比较耗费时间。

```
try.py
try.py
try.py
try.py
try.py
         107452 function calls (107433 primitive calls) in 1.808 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1000    0.798    0.001    0.800    0.001 {built-in method io.open}
     1000    0.651    0.001    0.657    0.001 {method 'readlines' of '_io._IO
Base' objects}
     1000    0.292    0.000    0.292    0.000 {method '__exit__' of '_io._IOB
ase' objects}
     1000    0.017    0.000    0.017    0.000 {built-in method builtins.print
}
    19000    0.012    0.000    0.016    0.000 re.py:288(_compile)
     1000    0.012    0.000    1.805    0.002 try.py:5(grep)
    19000    0.008    0.000    0.008    0.000 {method 'search' of 're.Pattern
' objects}
     3000    0.004    0.000    0.006    0.000 codecs.py:319(decode)
    37050    0.004    0.000    0.004    0.000 {built-in method builtins.isins
tance}
        1    0.003    0.003    1.808    1.808 try.py:1(<module>)
    19000    0.003    0.000    0.019    0.000 re.py:249(compile)
     3000    0.002    0.000    0.002    0.000 {built-in method _codecs.utf_8_
decode}
     1000    0.001    0.000    0.002    0.000 codecs.py:309(__init__)
     1000    0.000    0.000    0.000    0.000 codecs.py:260(__init__)
      3/1    0.000    0.000    0.000    0.000 sre_parse.py:494(_parse)
      6/1    0.000    0.000    0.000    0.000 sre_compile.py:87(_compile)
      7/2    0.000    0.000    0.000    0.000 sre_parse.py:175(getwidth)
      2/1    0.000    0.000    0.000    0.000 sre_parse.py:436(_parse_sub)
       44    0.000    0.000    0.000    0.000 sre_parse.py:165(__getitem__)
       27    0.000    0.000    0.000    0.000 sre_parse.py:234(__next)
        2    0.000    0.000    0.000    0.000 sre_compile.py:292(_optimize_ch
arset)
        1    0.000    0.000    0.000    0.000 sre_compile.py:783(compile)
```

图 7: python3 -m cProfile -s tottime try.py 1000 ’ˆ(import|s*def)[ˆ,]*$’ *.py 运行结果

1.2.3 工具：内存

像 C 或者 C++ 这样的语言，内存泄漏会导致您的程序在使用完内存后不去释放它。为了应对内存类的 Bug，我们可以使用类似 Valgrind 这样的工具来检查内存泄漏问题。在 Python 中，我们使用 memory-profiler 进行内存分析。

```
lin@xiaolin:/mnt/c/Users/lin/Downloads$ python3 -m memory_profiler try.py
Filename: try.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     1   40.160 MiB   40.160 MiB           1   @profile
     2                                         def my_func():
     3   47.695 MiB    7.535 MiB           1       a = [1] * (10 ** 6)
     4  200.238 MiB  152.543 MiB           1       b = [2] * (2 * 10 ** 7)
     5   47.859 MiB -152.379 MiB           1       del b
     6   47.859 MiB    0.000 MiB           1       return a
```

图 8: python 内存分析

1.2.4 工具对比

使用 line_profiler 来比较插入排序和快速排序的性能。使用 memory_profiler 来检查内存消耗。

```
lin@xiaolin:/mnt/c/Users/lin/Downloads$ kernprof -l -v sorts.py
Wrote profile results to sorts.py.lprof
Timer unit: 1e-06 s

Total time: 0.064978 s
File: sorts.py
Function: quicksort at line 22

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    22                                           @profile
    23                                           def quicksort(array):
    24     33800       9862.0      0.3     15.2      if len(array) <= 1:
    25     17400       4033.0      0.2      6.2          return array
    26     16400       4215.0      0.3      6.5      pivot = array[0]
    27     16400      18983.0      1.2     29.2      left = [i for i in array[1:] if i < pivot]
    28     16400      18213.0      1.1     28.0      right = [i for i in array[1:] if i >= pivot]
    29     16400       9672.0      0.6     14.9      return quicksort(left) + [pivot] + quicksort(right)
```

```
lin@xiaolin:/mnt/c/Users/lin/Downloads$ kernprof -l -v sorts.py
Wrote profile results to sorts.py.lprof
Timer unit: 1e-06 s

Total time: 0.127145 s
File: sorts.py
Function: quicksort_inplace at line 30

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    30                                           @profile
    31                                           def quicksort_inplace(array, low=0, high=None):
    32     33760       8188.0      0.2      6.4      if len(array) <= 1:
    33        37         14.0      0.4      0.0          return array
    34     33723       6509.0      0.2      5.1      if high is None:
    35       963        231.0      0.2      0.2          high = len(array)-1
    36     33723       6456.0      0.2      5.1      if low >= high:
    37     17343       2851.0      0.2      2.2          return array
    38
    39     16380       3073.0      0.2      2.4      pivot = array[high]
    40     16380       3476.0      0.2      2.7      j = low-1
    41    124794      23978.0      0.2     18.9      for i in range(low, high):
    42    108414      21692.0      0.2     17.1          if array[i] <= pivot:
    43     56459      10680.0      0.2      8.4              j += 1
    44     56459      14901.0      0.3     11.7              array[i], array[j] = array[j], array[i]
    45     16380       4791.0      0.3      3.8      array[high], array[j+1] = array[j+1], array[high]
    46     16380       9130.0      0.6      7.2      quicksort_inplace(array, low, j)
    47     16380       8544.0      0.5      6.7      quicksort_inplace(array, j+2, high)
    48     16380       2631.0      0.2      2.1      return array
```

图 9: line_profiler

```
lin@xiaolin:/mnt/c/Users/lin/Downloads$ python3 -m memory_profiler sorts.py
Filename: sorts.py/Users/lin/Downloads$

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    21   40.188 MiB   40.188 MiB       32502   @profile
    22                                         def quicksort(array):
    23   40.188 MiB    0.000 MiB       32502       if len(array) <= 1:
    24   40.188 MiB    0.000 MiB       16751           return array
    25   40.188 MiB    0.000 MiB       15751       pivot = array[0]
    26   40.188 MiB    0.000 MiB      151143       left = [i for i in array[1:] if i < pivot]
    27   40.188 MiB    0.000 MiB      151143       right = [i for i in array[1:] if i >= pivot]
    28   40.188 MiB    0.000 MiB       15751       return quicksort(left) + [pivot] + quicksort(right)
lin@xiaolin:/mnt/c/Users/lin/Downloads$ python3 -m memory_profiler sorts.py

Filename: sorts.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    30   40.164 MiB   40.164 MiB       33302   @profile
    31                                         def quicksort_inplace(array, low=0, high=None):
    32   40.164 MiB    0.000 MiB       33302       if len(array) <= 1:
    33   40.164 MiB    0.000 MiB          38           return array
    34   40.164 MiB    0.000 MiB       33264       if high is None:
    35   40.164 MiB    0.000 MiB         962           high = len(array)-1
    36   40.164 MiB    0.000 MiB       33264       if low >= high:
    37   40.164 MiB    0.000 MiB       17113           return array
    38
    39   40.164 MiB    0.000 MiB       16151       pivot = array[high]
    40   40.164 MiB    0.000 MiB       16151       j = low-1
    41   40.164 MiB    0.000 MiB      122151       for i in range(low, high):
    42   40.164 MiB    0.000 MiB      106000           if array[i] <= pivot:
    43   40.164 MiB    0.000 MiB       55146               j += 1
    44   40.164 MiB    0.000 MiB       55146               array[i], array[j] = array[j], array[i]
    45   40.164 MiB    0.000 MiB       16151       array[high], array[j+1] = array[j+1], array[high]
    46   40.164 MiB    0.000 MiB       16151       quicksort_inplace(array, low, j)
    47   40.164 MiB    0.000 MiB       16151       quicksort_inplace(array, j+2, high)
    48   40.164 MiB    0.000 MiB       16151       return array
```

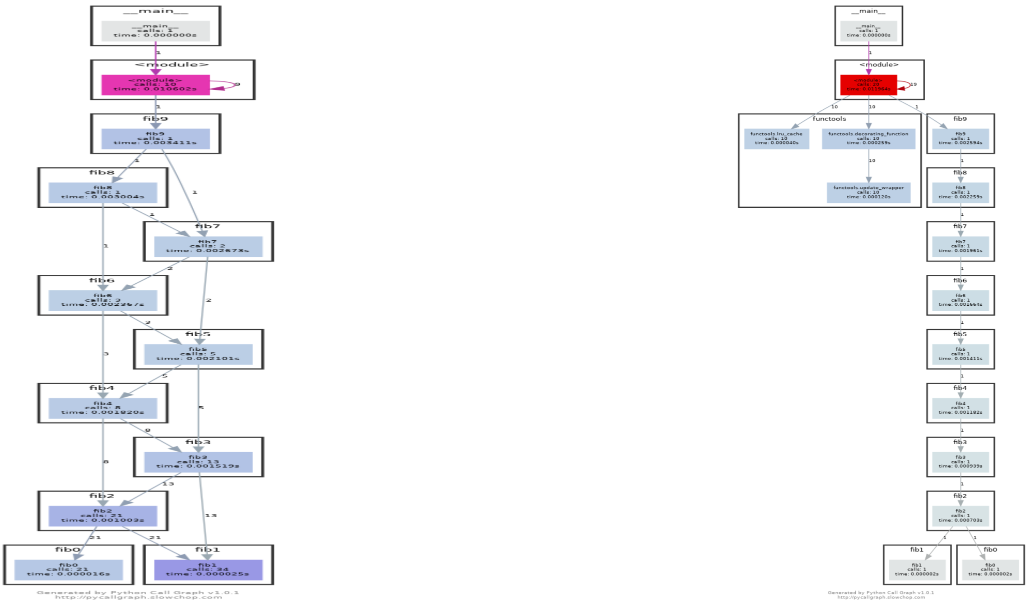
图 10: memory_profiler

1.2.5 可视化

显示注释时和放开注释后 fib0 的调用情况。

```python
#!/usr/bin/env python
def fib0(): return 0

def fib1(): return 1

s = """def fib{}(): return fib{}() + fib{}()"""

if __name__ == '__main__':

    for n in range(2, 10):
        exec(s.format( *args: n, n-1, n-2))
    # from functools import lru_cache
    # for n in range(10):
    #     exec("fib{} = lru_cache(1)(fib{})".format(n, n))
    print(eval("fib9()"))
```

图 11: fib 文件



(a) 有注释 (b) 无注释

图 12: 可视化

2 pytorch

2.1 张量

Numpy 是一个很好的框架，但它不能利用 GPU 加速其数值计算。Pytorch 一大作用就是可以代替 Numpy 库，所首先介绍 Tensors ， 也就是张量，它相当于 Numpy 的多维数组 (ndarrays)。

torch.empty(): 声明一个未初始化的矩阵

torch.rand()：随机初始化一个矩阵

torch.zeros()：创建数值皆为 0 的矩阵

torch.tensor()：直接传递 tensor 数值来创建

根据已有的 tensor 变量创建新的 tensor 变量：

tensor.new_ones()：new_*() 方法需要输入尺寸大小

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

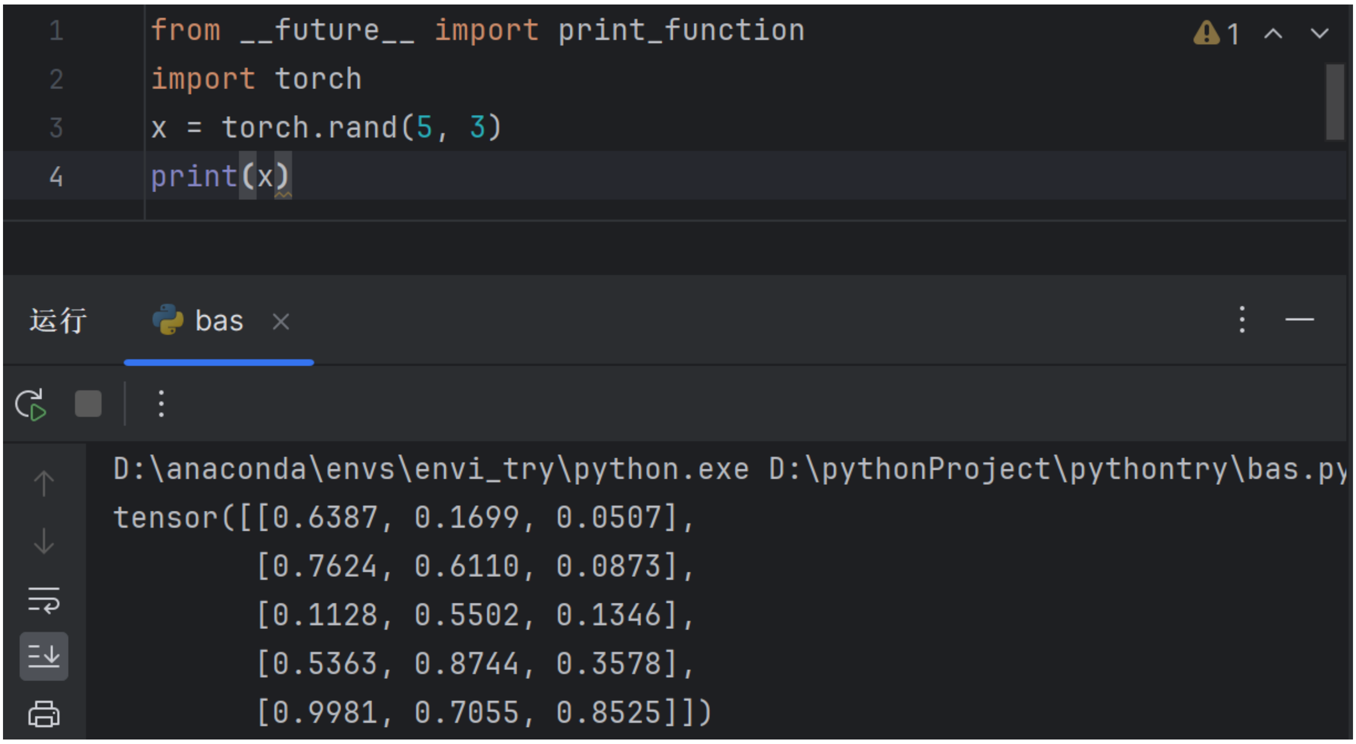


图 13: 随机初始化矩阵

2.2 操作

包括加法、转置、索引、切片、数学计算、线性代数、随机数等。

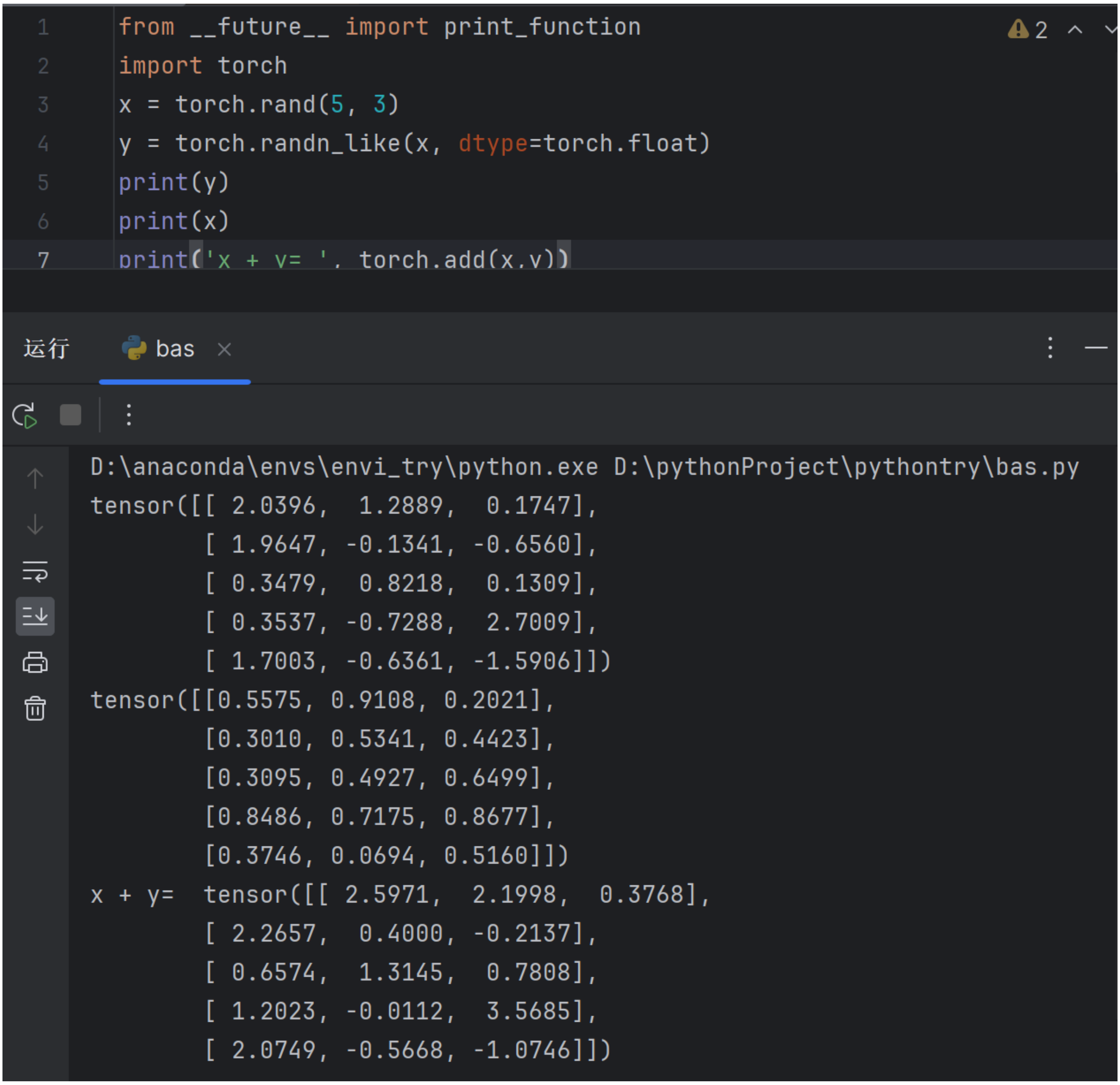


图 14: 加法运算示例

2.3 Tensor 和 Numpy 数组的转换

Tensor 转换为 Numpy 数组：tensor.numpy()；Numpy 数组转换为 Tensor：torch.from_numpy(numpy_array)

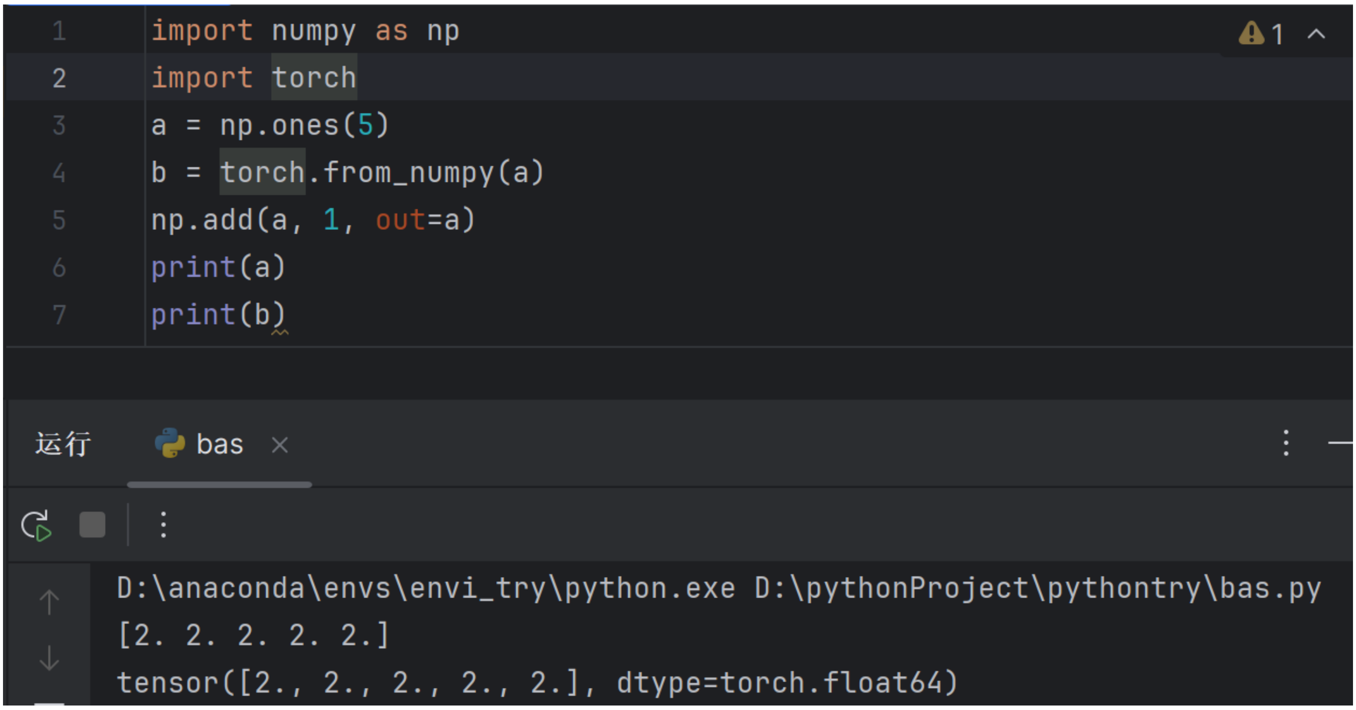
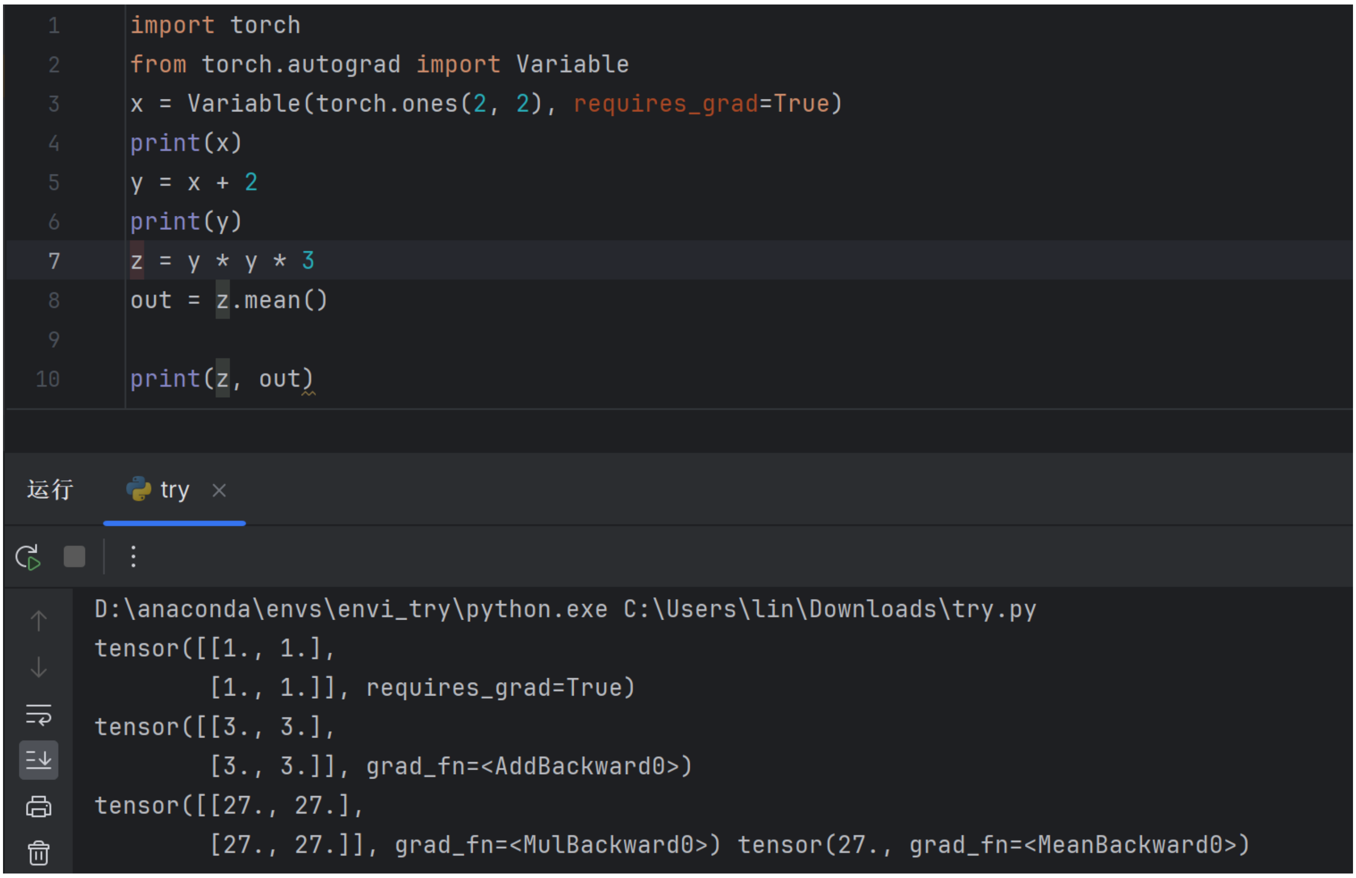


图 15: Numpy 数组转换为 Tensor

2.4 Autograd

autograd 包是 PyTorch 所有神经网络的核心，为 Tensors 上的所有操作提供了自动区分。autograd.Variable 是包的中央类。它包含一个 Tensor，并支持几乎所有定义的操作。完成计算后可以调用.backward() 并自动计算所有梯度。还可以通过.data 属性访问原始张量，而将此变量的梯度累加到.grad。a Function 对于 autograd 实现也非常重要。Variable 并被 Function 互连并建立一个非循环图，编码完整的计算历史。每个变量都有一个.creator 引用 Function 已创建的属性的属性 Variable。如果想计算导数，可以在一个 Variable 中使用.backward()。如果 Variable 是标量（即它保存一个元素数据），则不需要指定任何参数 backward()，但是如果它具有更多元素，则需要指定一个 grad_output 作为匹配形状的张量的参数。

2.4.1 变量



```python
import torch
from torch.autograd import Variable
x = Variable(torch.ones(2, 2), requires_grad=True)
print(x)
y = x + 2
print(y)
z = y * y * 3
out = z.mean()

print(z, out)
```



```
D:\anaconda\envs\envi_try\python.exe C:\Users\lin\Downloads\try.py
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
tensor([[3., 3.],
        [3., 3.]], grad_fn=<AddBackward0>)
tensor([[27., 27.],
        [27., 27.]], grad_fn=<MulBackward0>) tensor(27., grad_fn=<MeanBackward0>)
```

图 16: 变量

2.4.2 梯度

```python
import torch
from torch.autograd import Variable
x = Variable(torch.ones(2, 2), requires_grad=True)
# print(x)
y = x + 2
# print(y)
z = y * y * 3
out = z.mean()
# print(z, out)
a = torch.randn(2, 2)
a = ((a * 3) / (a - 1))
print(a.requires_grad)
a.requires_grad_(True)
print(a.requires_grad)
b = (a * a).sum()
print(b.grad_fn)
out.backward()
print(x.grad)
x = torch.randn(3, requires_grad=True)
y = x * 2
while y.data.norm() < 1000:
    y = y * 2
print(y)
gradients = torch.tensor([0.1, 1.0, 0.0001], dtype=torch.float)
y.backward(gradients)
print(x.grad)
print(x.requires_grad)
print((x ** 2).requires_grad)
with torch.no_grad():
    print((x ** 2).requires_grad)
```

```
D:\anaconda\envs\envi_try\python.exe C:\Users\lin\Downloads\try.py
False
True
<SumBackward0 object at 0x000001422226F040>
tensor([[4.5000, 4.5000],
        [4.5000, 4.5000]])
tensor([-917.6904,  383.7716, -188.3755], grad_fn=<MulBackward0>)
tensor([5.1200e+01, 5.1200e+02, 5.1200e-02])
True
True
False
```

图 17: 梯度

2.5 神经网络

使用 torch.nn 包构建神经网络。神经网络的典型训练过程如下：1 定义具有一些可学习参数（或权重）的神经网络；2 迭代输入数据集；3 通过网络处理输入；4 计算损失（输出距离是正确的）；5 传播梯度回到网络的

参数；6 更新网络的权重，通常使用简单的更新规则：weight = weight - learning_rate * gradient

对如下的代码：定义了三个卷积层（self.conv1, self.conv2）和两个全连接层（self.fc1, self.fc2, self.fc3）。self.conv1 是第一个卷积层，输入通道数为 1（例如，灰度图像），输出通道数为 6，卷积核大小为 5x5。self.conv2 是第二个卷积层，输入通道数为 6（来自第一个卷积层的输出），输出通道数为 16，卷积核大小也为 5x5。self.fc1、self.fc2、self.fc3 是全连接层，分别具有 120、84、10 个输出单元。这些层的输入特征数量是根据前面层的输出动态计算的，在这里，self.fc1 的输入特征数被硬编码为 16 * 5 * 5，这是基于假设在通过两个卷积层和池化层后，特征图（feature map）的尺寸会变为 5x5（实际情况取决于输入图像尺寸和步长/填充等参数）。forward 方法定义了数据通过网络的前向传播路径。输入 x 首先通过第一个卷积层 self.conv1，然后应用 ReLU 激活函数和 2x2 的最大池化。接着，输出通过第二个卷积层 self.conv2，再次应用 ReLU 激活函数和 2x2 的最大池化。注意，这里的池化层使用了简写形式，仅指定了池化窗口大小。后使用 num_flat_features 方法计算特征图展开后的一维向量的长度，并将特征图 x 转换为这个一维向量。最后，这个一维向量通过三个全连接层 self.fc1、self.fc2、self.fc3，每个层之间都应用 ReLU 激活函数（除了最后一个全连接层外，它通常用于输出层，可能直接输出分数或经过 softmax 激活的概率）。

```python
import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.conv1 = nn.Conv2d( in_channels: 1, out_channels: 6, kernel_size: 5)
        self.conv2 = nn.Conv2d( in_channels: 6, out_channels: 16, kernel_size: 5)
        # an affine operation: y = Wx + b
        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 = F.max_pool2d(F.relu(self.conv1(x)), kernel_size: (2, 2))
        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

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
net = Net()
```

```
D:\anaconda\envs\envi_try\python.exe C:\Users\lin\Downloads\try.py
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 18: 一个简单的 CNN 模型

2.6 一个训练神经网络的运行尝试

2.6.1 数据预处理

一般来说，当您必须处理图像，文本，音频或视频数据时，可以使用将数据加载到 numpy 数组中的标准 python 包。然后将数组转换成一个 torch.*Tensor。

```python
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize( mean: (0.5, 0.5, 0.5),  std: (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=0)

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=0)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

def imshow(img):
    img = img / 2 + 0.5  # 反归一化
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg,  axes: (1, 2, 0)))
    plt.show()  # 显示图像

dataiter = iter(trainloader)
images, labels = next(dataiter)

grid_image = torchvision.utils.make_grid(images)
imshow(grid_image)

print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```

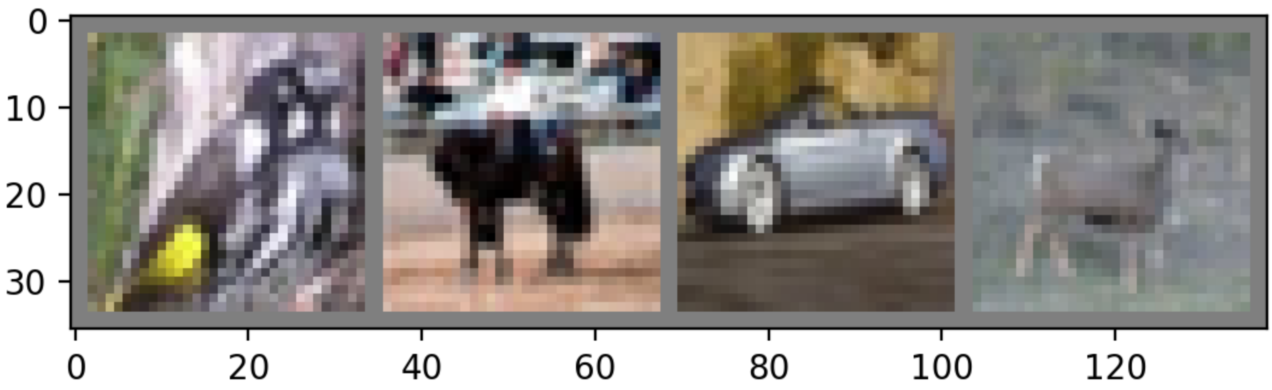


图 19: 加载和归一化 CIFAR10 并展示一些训练图像

2.6.2 定义卷积神经网络

获取 3 通道图像。

```python
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 20: 定义卷积神经网络

2.6.3 定义一个 Loss 函数和优化器

import torch.optim as optim

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), lr=0.001,momentum=0.9)

2.6.4 训练网络

遍历数据迭代器，并将输入馈送到网络并进行优化。

```python
for epoch in range(2):

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data

        optimizer.zero_grad()

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

        running_loss += loss.item()
        if i % 2000 == 1999:
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

```
D:\anaconda\envs\envi_try\python.exe C:\Users\lin\Downloads\try.py
Files already downloaded and verified
Files already downloaded and verified
[1,  2000] loss: 2.175
[1,  4000] loss: 1.878
[1,  6000] loss: 1.658
[1,  8000] loss: 1.572
[1, 10000] loss: 1.515
[1, 12000] loss: 1.472
[2,  2000] loss: 1.396
[2,  4000] loss: 1.358
[2,  6000] loss: 1.336
[2,  8000] loss: 1.339
[2, 10000] loss: 1.327
[2, 12000] loss: 1.290
Finished Training
```

图 21: 训练过程

2.6.5 测试数据

我们将通过预测神经网络输出的类标签来检查，并根据地面实况进行检查。如果预测是正确的，将样本添加到正确预测列表中。

```
dataiter = iter(testloader)
images, labels = next(dataiter)

imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
```

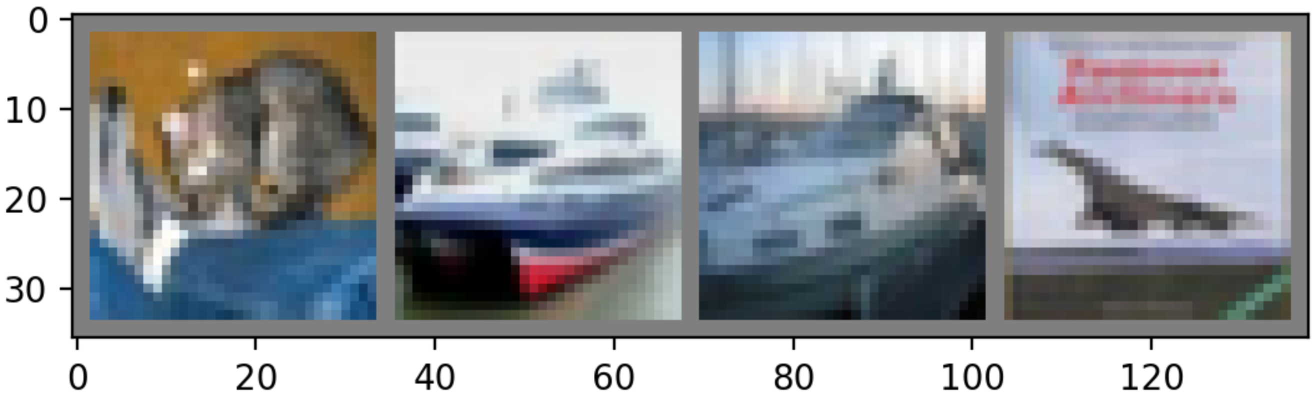


图 22: 输出为 GroundTruth: cat ship ship plane

3 总结与心得

在学习调试及性能分析的过程中，我了解到调试不仅仅是找出代码中的错误，更是对程序逻辑、数据流动及执行流程的深入理解。我学会了使用调试工具，如断点、观察变量、单步执行等。性能分析也十分重要。通过了解程序的运行时间、内存占用等关键指标，可以更准确地定位性能问题，进而优化代码。元编程是指编写那些能够操作或改变其他程序（包括它们自身）的程序或代码，这个领域很复杂，需要不断学习和实践。PyTorch 是一个功能强大的深度学习框架，我了解了它的基本概念，并尝试使用 PyTorch 构建简单的神经网络模型，并进行训练和测试。