

第四周实验报告

张津侨

2024 年 9 月 13 日

1 练习内容及结果

1.1 查找登录信息及其所执行的指令

```
cecilia-0623@ubuntu:~$ journalctl | grep sudo
Sep 05 18:46:33 ubuntu sudo[53227]: cecilia-0623 : TTY=pts/4 ; PWD=/home/cecilia-0623 ; USER=root ; COMMAND=/usr/bin/apt install tmux
Sep 05 18:46:33 ubuntu sudo[53227]: pam_unix(sudo:session): session opened for user root by (uid=0)
Sep 05 18:46:45 ubuntu sudo[53227]: pam_unix(sudo:session): session closed for user root
cecilia-0623@ubuntu:~$ sudo ls
[sudo] password for cecilia-0623:
abcde.txt  backup.sh  Downloads      hello.sh  Music      sigint.py  try.txt
abcd.txt   Desktop    examples.desktop LaTeX.txt  Pictures   Templates  Videos
abc.txt    Documents  gits           min.sh    Public     test.txt   vinrc
cecilia-0623@ubuntu:~$ journalctl | grep sudo
Sep 05 18:46:33 ubuntu sudo[53227]: cecilia-0623 : TTY=pts/4 ; PWD=/home/cecilia-0623 ; USER=root ; COMMAND=/usr/bin/apt install tmux
Sep 05 18:46:33 ubuntu sudo[53227]: pam_unix(sudo:session): session opened for user root by (uid=0)
Sep 05 18:46:45 ubuntu sudo[53227]: pam_unix(sudo:session): session closed for user root
Sep 10 03:42:00 ubuntu sudo[56461]: cecilia-0623 : TTY=pts/4 ; PWD=/home/cecilia-0623 ; USER=root ; COMMAND=/bin/ls
Sep 10 03:42:00 ubuntu sudo[56461]: pam_unix(sudo:session): session opened for user root by (uid=0)
Sep 10 03:42:00 ubuntu sudo[56461]: pam_unix(sudo:session): session closed for u
```

使用 Linux 上的 journalctl 命令
来获取最近一天中用户的登录信息及其所执行的命令
使用一些无害的命令，例如 sudo ls 后再次查看

1.2 创建虚拟环境并下载 Pytorch

```
(base) C:\Users\ZJQ>conda create -n myenv python=3.8
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: D:\anaconda\envs\myenv

added / updated specs:
 - python=3.8

The following packages will be downloaded:
```

package	build	
ca-certificates-2024.7.2	haa95532_0	128 KB
openssl-3.0.15	h827c3e9_0	7.8 MB
pip-24.2	py38haa95532_0	2.4 MB
python-3.8.19	h1aa4202_0	18.9 MB
setuptools-72.1.0	py38haa95532_0	2.4 MB
vc-14.40	h2eaa2aa_0	10 KB
vs2015_runtime-14.40.33807	h98bb1dd_0	1.3 MB

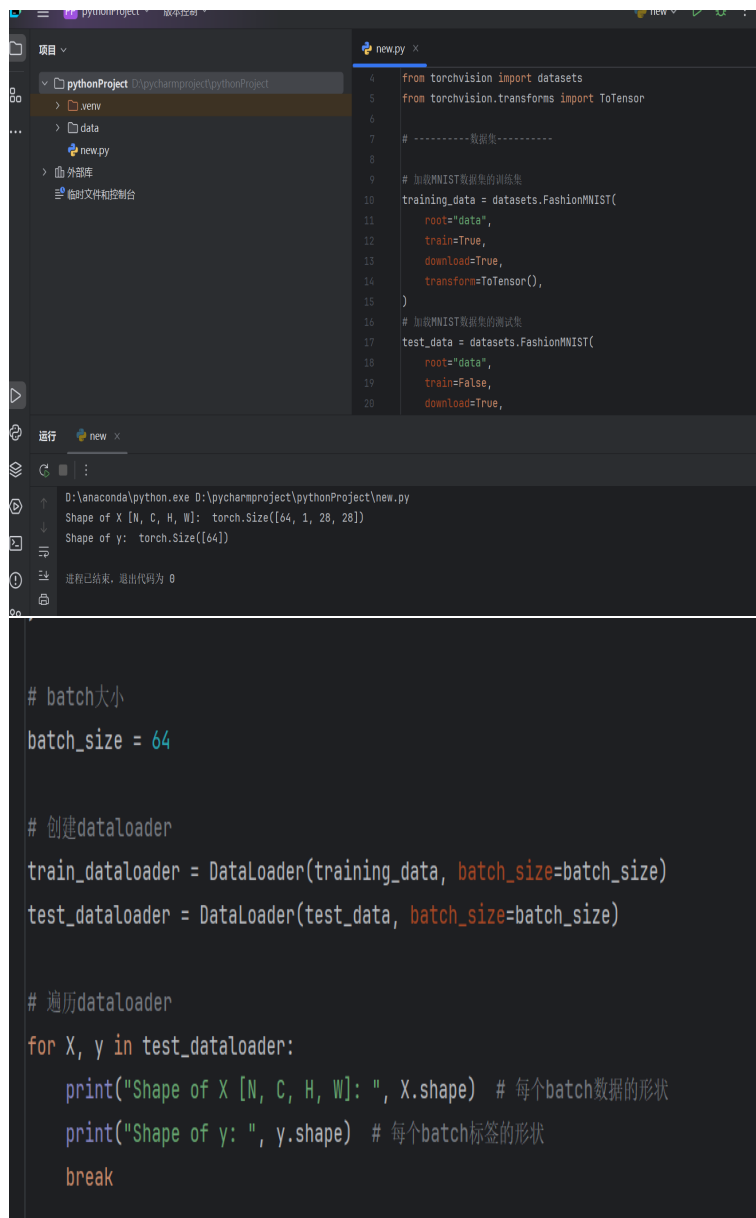
conda create -n myenv python=3.8

创建一个名为 myenv 的虚拟环境，其中 Python 版本为 3.8

并将其激活 conda activate myenv

下载 Pytorch pip install torch torchvision torchaudio

1.3 Pytorch 数据集加载



```
4 from torchvision import datasets
5 from torchvision.transforms import ToTensor
6
7 # -----数据集-----
8
9 # 加载MNIST数据集的训练集
10 training_data = datasets.FashionMNIST(
11     root="data",
12     train=True,
13     download=True,
14     transform=ToTensor(),
15 )
16
17 # 加载MNIST数据集的测试集
18 test_data = datasets.FashionMNIST(
19     root="data",
20     train=False,
21     download=True,
22 )
23
24 # batch大小
25 batch_size = 64
26
27 # 创建dataloader
28 train_dataloader = DataLoader(training_data, batch_size=batch_size)
29 test_dataloader = DataLoader(test_data, batch_size=batch_size)
30
31 # 遍历dataloader
32 for X, y in test_dataloader:
33     print("Shape of X [N, C, H, W]: ", X.shape) # 每个batch数据的形状
34     print("Shape of y: ", y.shape) # 每个batch标签的形状
35     break
```

加载 MNIST 数据集的测试集

设置 batch 大小，创建 dataloader

遍历 dataloader，并输出每个 batch 数据及标签的形状

1.4 shellcheck 的下载及应用

```
root@ubuntu:~# sudo apt-get install shellcheck
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  shellcheck
0 upgraded, 1 newly installed, 0 to remove and 44 not upgraded.
Need to get 1,603 kB of archives.
After this operation, 13.3 MB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu xenial/universe amd64 shellcheck amd64
root@ubuntu:~# git clone https://github.com/koalaman/shellcheck.git
Cloning into 'shellcheck'...
remote: Enumerating objects: 9694, done.
remote: Counting objects: 100% (2321/2321), done.
remote: Compressing objects: 100% (198/198), done.
remote: Total 9694 (delta 2220), reused 2124 (delta 2122), pack-reused 7373 (from 1)
  LibreOfficeImpress 100% (9694/9694), 5.31 MiB | 2.38 MiB/s, done.
Resolving deltas: 100% (5977/5977), done.
Checking connectivity... done.
root@ubuntu:~# cd shellcheck
root@ubuntu:~/shellcheck# cabal update
The program 'cabal' is currently not installed. You can install it by typing:
apt install cabal-install
root@ubuntu:~/shellcheck# vim bad_script.sh
root@ubuntu:~/shellcheck# shellcheck bad_script.sh

In bad_script.sh line 1:
for f in $(ls *.m3u)
^-- SC2148: Tips depend on target shell and yours is unknown. Add a shebang.
^-- SC2045: Iterating over ls output is fragile. Use globs.
^-- SC2035: Use ./*.m3u so names with dashes won't become options.

In bad_script.sh line 3:
grep -qI hq *.mp3 $f && echo -e 'Playlist $f contains a HQ file in mp3 format'
^-- SC2035: Use ./*.mp3 so names with dashes won't become options.
^-- SC2080: Double quote to prevent globbing and word splitting.
^-- SC2016: Expressions don't expand in single quotes, use double quotes for that.

root@ubuntu:~/shellcheck#
```

sudo apt-get install shellcheck

克隆项目仓库, git clone <https://github.com/koalaman/shellcheck.git>

编译并安装, cabal update, cabal install

通过以下命令来检查你的 shell 脚本, shellcheck name.sh

1.5 构建系统

```
root@ubuntu:~/shellcheck# cat paper.tex
\documentclass{article}
\usepackage{graphicx}
\begin{document}
\includegraphics[scale=0.65]{plot-data.png}
\end{document}
root@ubuntu:~/shellcheck# cat plot.py
#!/usr/bin/env python
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import argparse

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

data = np.loadtxt(args.i)
plt.plot(data[:, 0], data[:, 1])
plt.savefig(args.o)
root@ubuntu:~/shellcheck# cat data.dat
1 1
2 2
```

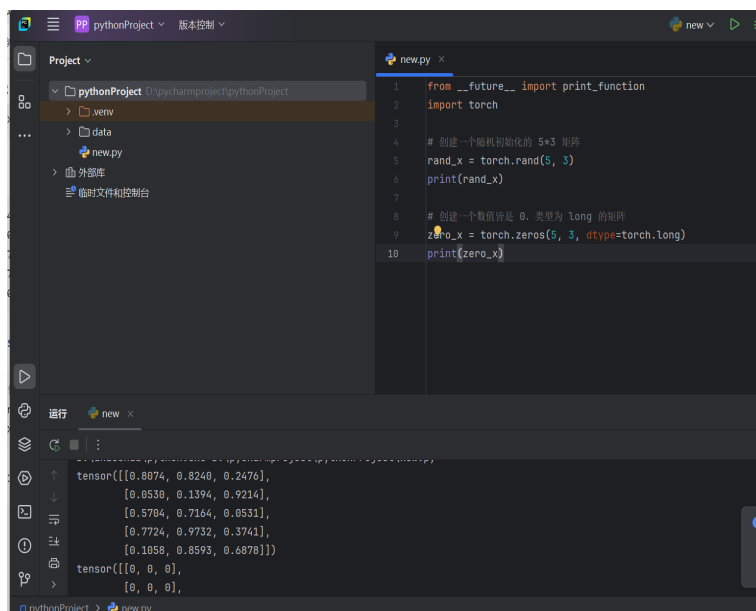
make 会告诉我们，为了构建出 paper.pdf，它需要 paper.tex

构建 plot-data.png 需要源文件 data.dat

当我们执行 make：`./plot.py -i data.dat -o plot-data.png`

`pdflatex paper.tex`

1.6 pytorch 创建矩阵



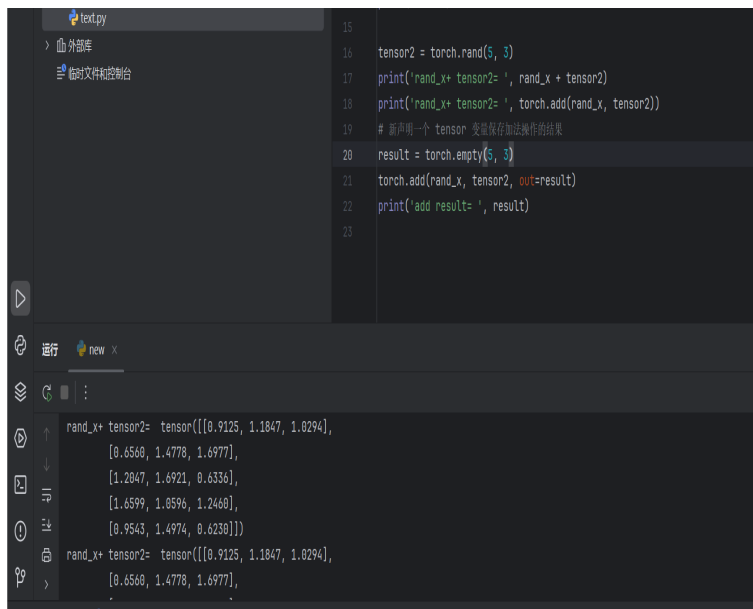
```
1 from __future__ import print_function
2 import torch
3
4 # 创建一个随机初始化的 5*3 矩阵
5 rand_x = torch.rand(5, 3)
6 print(rand_x)
7
8 # 创建一个数值皆 0, 类型为 long 的矩阵
9 zero_x = torch.zeros(5, 3, dtype=torch.long)
10 print(zero_x)
```

```
tensor([[[0.8074, 0.8240, 0.2476],
         [0.0530, 0.1394, 0.9214],
         [0.5704, 0.7164, 0.0531],
         [0.7724, 0.9732, 0.3741],
         [0.1058, 0.8593, 0.6878]])])
tensor([[[0, 0, 0],
         [0, 0, 0],
         [0, 0, 0],
         [0, 0, 0],
         [0, 0, 0]]])
```

`torch.rand()`: 随机初始化一个矩阵

`torch.zeros()`: 创建数值皆为 0 的矩阵

1.7 pytorch 矩阵的运算



```
15
16 tensor2 = torch.rand(3, 3)
17 print('rand_x+ tensor2= ', rand_x + tensor2)
18 print('rand_x+ tensor2= ', torch.add(rand_x, tensor2))
19 # 再声明一个 tensor 变量保存加法操作的结果
20 result = torch.empty(3, 3)
21 torch.add(rand_x, tensor2, out=result)
22 print('add result= ', result)
23
```

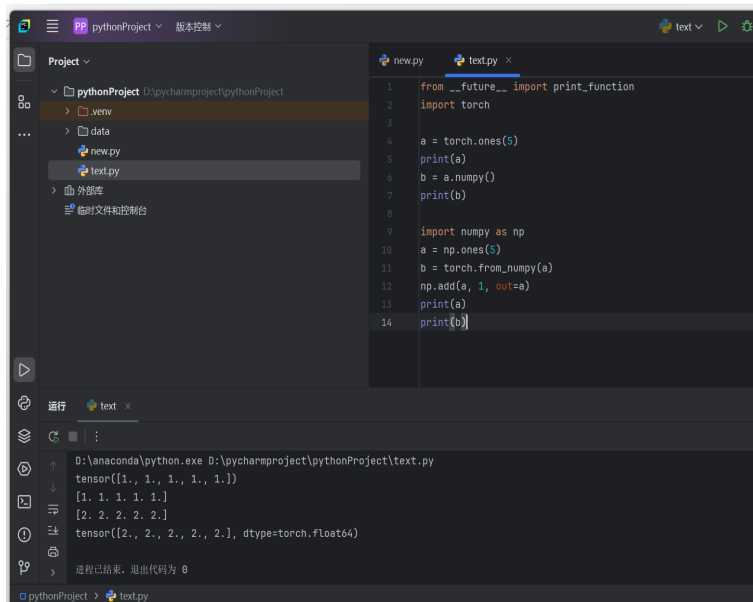
运行 new x

```
↑
rand_x+ tensor2= tensor([[[0.9125, 1.1847, 1.8294],
[0.6568, 1.4778, 1.6977],
[1.2847, 1.6921, 0.6336],
[1.6599, 1.8598, 1.2468],
[0.9543, 1.4974, 0.6238]])
↓
rand_x+ tensor2= tensor([[[0.9125, 1.1847, 1.8294],
[0.6568, 1.4778, 1.6977],
[1.2847, 1.6921, 0.6336],
[1.6599, 1.8598, 1.2468],
[0.9543, 1.4974, 0.6238]])
```

将两个矩阵相加

新声明一个 tensor 变量保存加法操作的结果

1.8 Numpy 数组与 Tensor 的互相转化



```
1 from __future__ import print_function
2 import torch
3
4 a = torch.ones(5)
5 print(a)
6 b = a.numpy()
7 print(b)
8
9 import numpy as np
10 a = np.ones(5)
11 b = torch.from_numpy(a)
12 np.add(a, 1, out=a)
13 print(a)
14 print(b)
```

运行

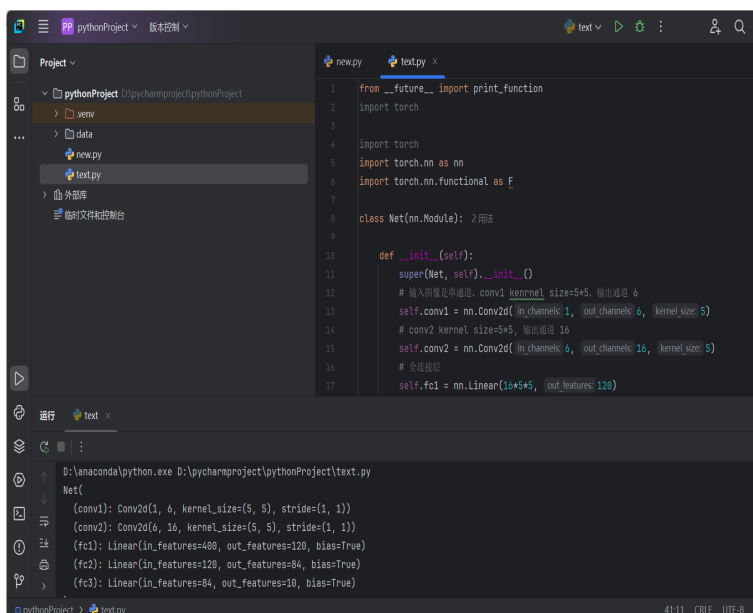
```
D:\anaconda\python.exe D:\pycharmproject\pythonProject\text.py
tensor([1., 1., 1., 1., 1.])
[1. 1. 1. 1. 1.]
[2. 2. 2. 2. 2.]
tensor([2., 2., 2., 2., 2.], dtype=torch.FloatTensor)
```

进程已结束，退出代码为 0

Tensor 转换为 Numpy 数组: `b = a.numpy()`

Numpy 数组转换为 Tensor: `b = torch.from_numpy(a)`

1.9 定义一个神经网络并打印

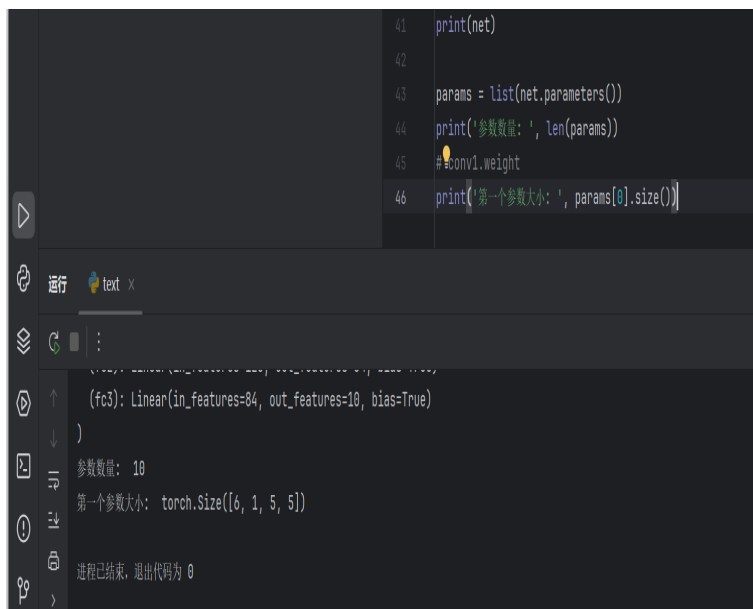


```
1 from __future__ import print_function
2 import torch
3
4 import torch
5 import torch.nn as nn
6 import torch.nn.functional as F
7
8 class Net(nn.Module):
9     def __init__(self):
10         super(Net, self).__init__()
11         # 输入图像是单通道, conv1 kernel size=5*5, 输出通道 6
12         self.conv1 = nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=2)
13         # conv2 kernel size=5*5, 输出通道 16
14         self.conv2 = nn.Conv2d(6, 16, kernel_size=5, stride=1, padding=2)
15         # 全连接层
16         self.fc1 = nn.Linear(16*5*5, out_features=120)
```

运行

```
D:\anaconda\python.exe D:\pycharmproject\pythonProject\text.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)
```


1.10 返回网络的训练参数



```
41 print(net)
42
43 params = list(net.parameters())
44 print('参数数量: ', len(params))
45 # conv1.weight
46 print('第一个参数大小: ', params[0].size())
```

运行 text x

(fc3): Linear(in_features=84, out_features=10, bias=True)

参数数量: 10

第一个参数大小: torch.Size([6, 1, 5, 5])

进程已结束, 退出代码为 0

1.11 markdown

Markdown 是一个轻量化的标记语言，致力于将人们编写纯文本时的一些习惯标准化。比如：

用 * 包围的文字表示强调（斜体），或者用 ** 表示特别强调（粗体）；

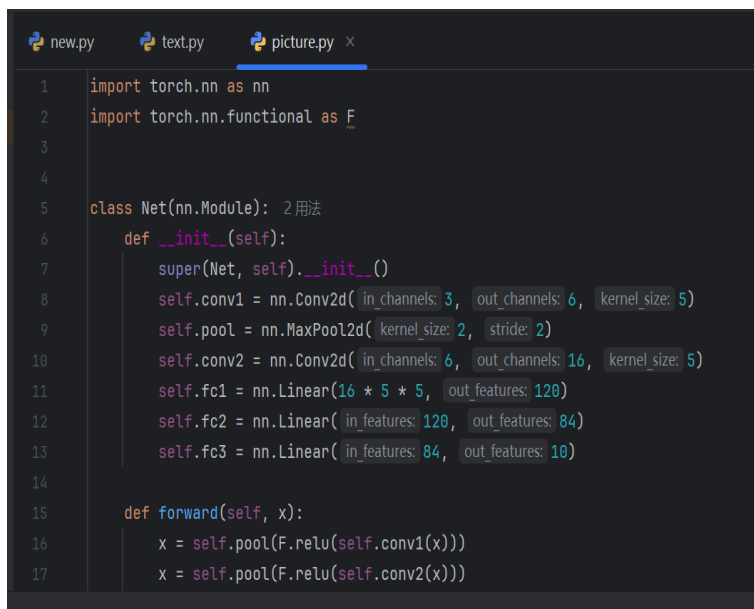
以井号开头的行是标题，井号的数量表示标题的级别，比如：双井号二级标题；

以 - 开头代表一个无序列表的元素。一个数字加.（比如 1.）代表一个有序列表元素；

反引号 `（backtick）包围的文字会以代码字体显示。如果要显示一段代码，可以在每一行前加四个空格缩进，或者使用三个反引号包围整个代码片段：

如果要添加超链接，将需要显示的文字用方括号包围，并在后面紧接着用圆括号包围链接：[显示文字](指向的链接)。

1.12 构建一个卷积神经网络



```
1 import torch.nn as nn
2 import torch.nn.functional as F
3
4
5 class Net(nn.Module): 2用法
6     def __init__(self):
7         super(Net, self).__init__()
8         self.conv1 = nn.Conv2d( in_channels: 3, out_channels: 6, kernel_size: 5)
9         self.pool = nn.MaxPool2d( kernel_size: 2, stride: 2)
10        self.conv2 = nn.Conv2d( in_channels: 6, out_channels: 16, kernel_size: 5)
11        self.fc1 = nn.Linear(16 * 5 * 5, out_features: 120)
12        self.fc2 = nn.Linear( in_features: 120, out_features: 84)
13        self.fc3 = nn.Linear( in_features: 84, out_features: 10)
14
15    def forward(self, x):
16        x = self.pool(F.relu(self.conv1(x)))
17        x = self.pool(F.relu(self.conv2(x)))
```

1.13 在 github 中创建一个 issue

议题可以用来反映软件运行的问题或者请求新的功能。

引用 Issue 的基本格式：

1. 找到 Issue 编号：

每个 issue 在 GitHub 上都有一个唯一的编号。这个编号通常显示在 issue 标题旁边，格式为井号后跟一个数字，例如井号 123。

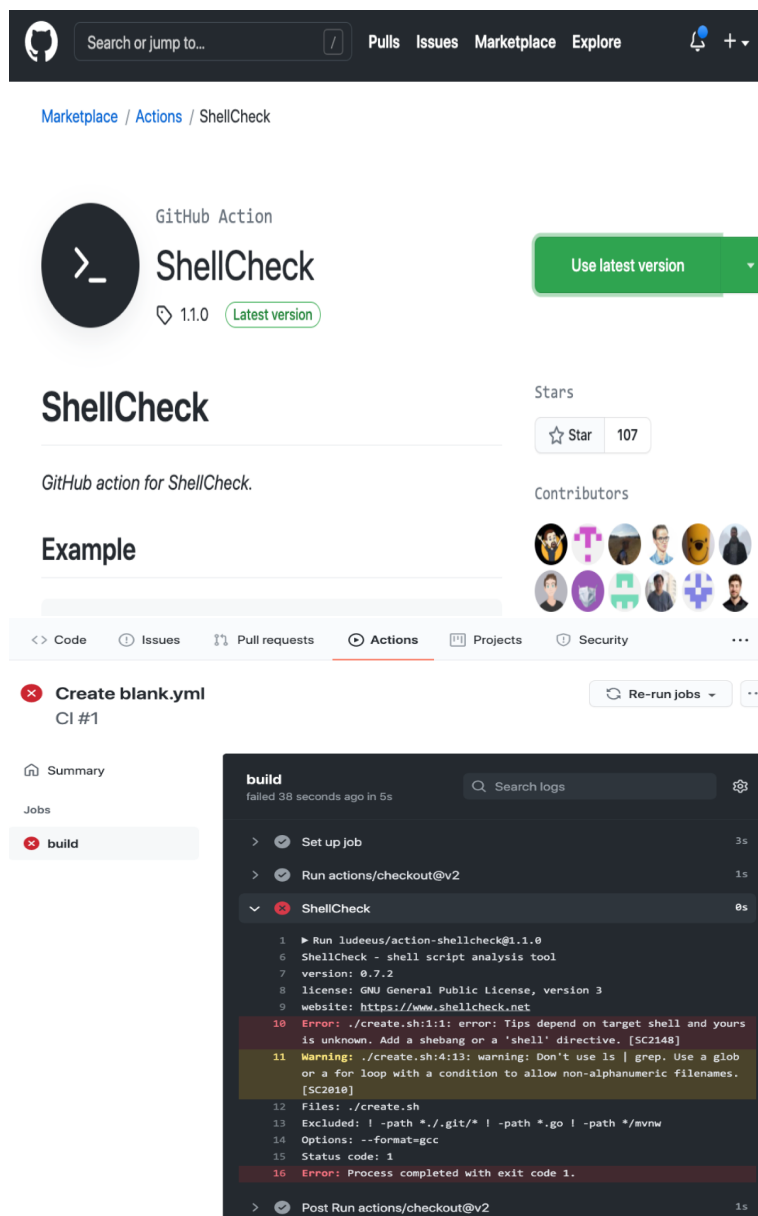
2. 在评论或描述中引用 Issue：

在任何 GitHub 评论、issue 或 PR 的描述中，只需输入井号后面跟上 issue 的编号即可。例如，要引用编号为 123 的 issue，只需写井号 123。

3. GitHub 自动链接：

当您提交评论或描述时，GitHub 会自动将井号 123 转换成指向相应 issue 的链接

1.14 元编程:对 github 仓库中的所有 shell 文件执行 shellcheck



Search or jump to... / Pulls Issues Marketplace Explore

Marketplace / Actions / ShellCheck

GitHub Action

ShellCheck

1.1.0 Latest version

Use latest version

Stars

Star 107

Contributors

Example

<> Code Issues Pull requests **Actions** Projects Security

Create blank.yml

CI #1

Summary

Jobs

build

build

failed 38 seconds ago in 5s

Search logs

Set up job 3s

Run actions/checkout@v2 1s

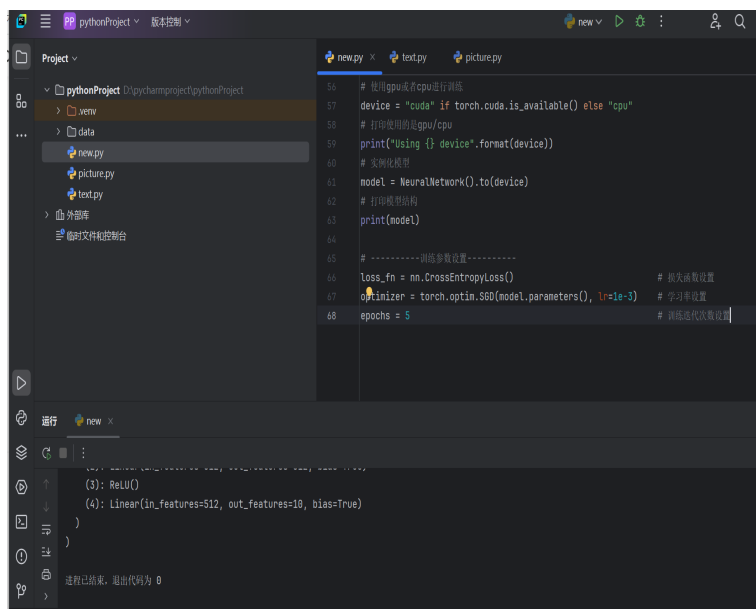
ShellCheck 0s

```
1 ▶ Run ludeeus/action-shellcheck@1.1.0
6 ShellCheck - shell script analysis tool
7 version: 0.7.2
8 license: GNU General Public License, version 3
9 website: https://www.shellcheck.net
10 Error: ./create.sh:11: error: Tips depend on target shell and yours
    is unknown. Add a shebang or a 'shell' directive. [SC2148]
11 Warning: ./create.sh:4:13: warning: Don't use ls | grep. Use a glob
    or a for loop with a condition to allow non-alphanumeric filenames.
    [SC2010]
12 Files: ./create.sh
13 Excluded: ! -path */.git/* ! -path *.go ! -path */mvnw
14 Options: --format=gcc
15 Status code: 1
16 Error: Process completed with exit code 1.
```

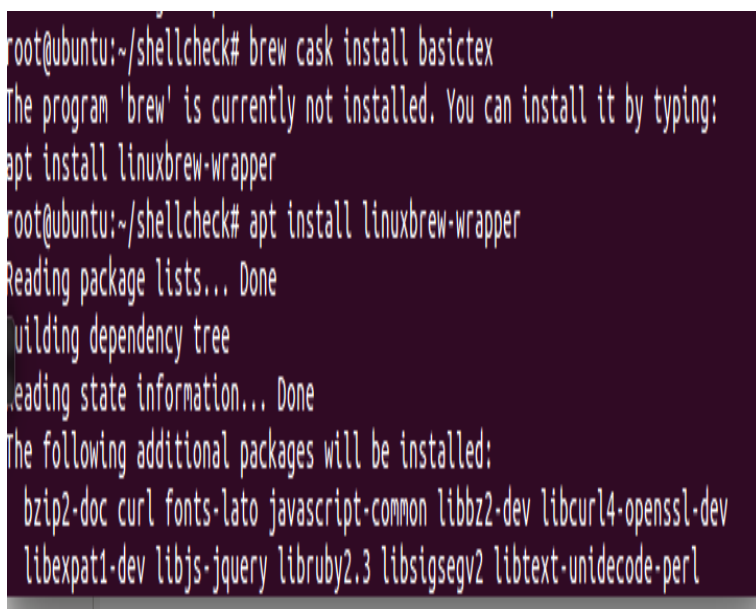
Post Run actions/checkout@v2 1s

基于 GitHub Pages 创建任意一个可以自动发布的页面。
添加一个 GitHub Action 到该仓库，对仓库中的所有 shell 文件执行 shellcheck。

1.15 模型参数设置



1.16 安装 linuxbrew-wrapper(linux)/basictex(MacOS)



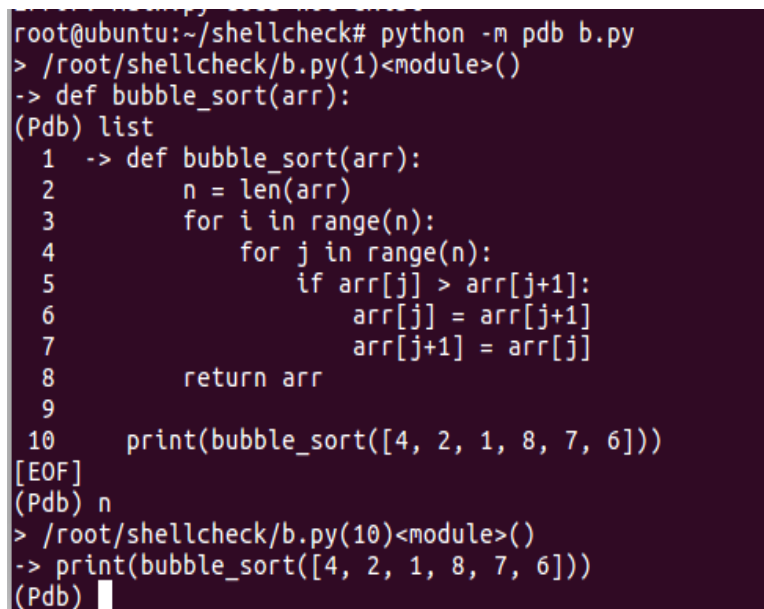
为了编译 LaTeX，首先需要安装
linuxbrew-wrapper(linux)/basictex(MacOS)

1.17 对于守护进程的理解

守护进程是一种在后台运行的计算机程序，通常在系统启动时自动启动，并持续运行以执行特定任务或提供服务，而不需要用户直接交互。

守护进程常用于管理系统资源、处理网络请求、监控系统状态或执行定时任务，它们通常以系统用户身份运行，确保在系统运行期间保持稳定和高效的服务。

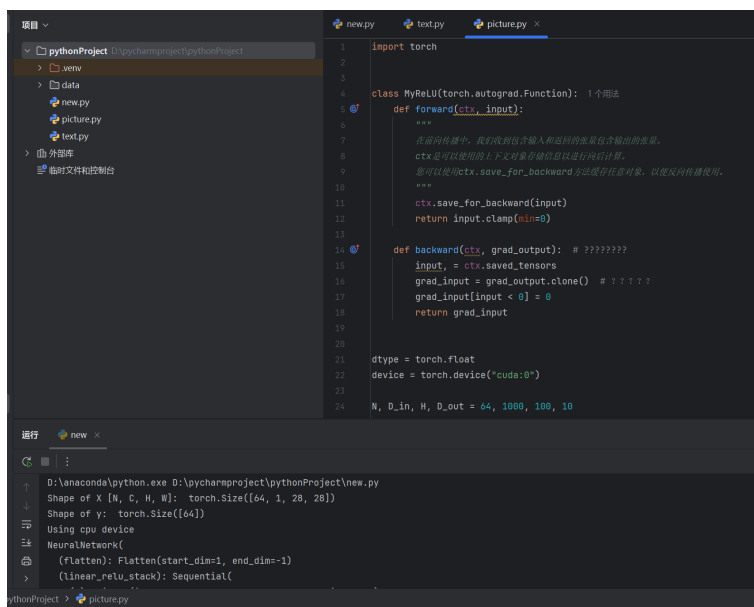
1.18 pdb 调试



```
root@ubuntu:~/shellcheck# python -m pdb b.py
> /root/shellcheck/b.py(1)<module>()
-> def bubble_sort(arr):
(Pdb) list
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) n
> /root/shellcheck/b.py(10)<module>()
-> print(bubble_sort([4, 2, 1, 8, 7, 6]))
(Pdb) █
```

python -m pdb b.py 进入调试，
list(l) 列出文件内容，next(n) 执行当前行程序

1.19 pytorch: 定义新的自动求导函数



```
1 import torch
2
3
4 class MyReLU(torch.autograd.Function): 1个用法
5     """
6     在正向传播中，我们得到包含输入和返回的张量包含输出的张量。
7     ctx 是可以使用上下文对象存储信息以进行反向计算。
8     您可以使用 ctx.save_for_backward 方法保存任意对象，以便反向传播使用。
9     """
10
11     def forward(ctx, input):
12         return input.clamp(min=0)
13
14     def backward(ctx, grad_output): # ????????
15         input, = ctx.saved_tensors
16         grad_input = grad_output.clone() # ?????
17         grad_input[input < 0] = 0
18         return grad_input
19
20
21 dtype = torch.float
22 device = torch.device("cuda:0")
23
24 N, D_in, H, D_out = 64, 1000, 100, 10
```

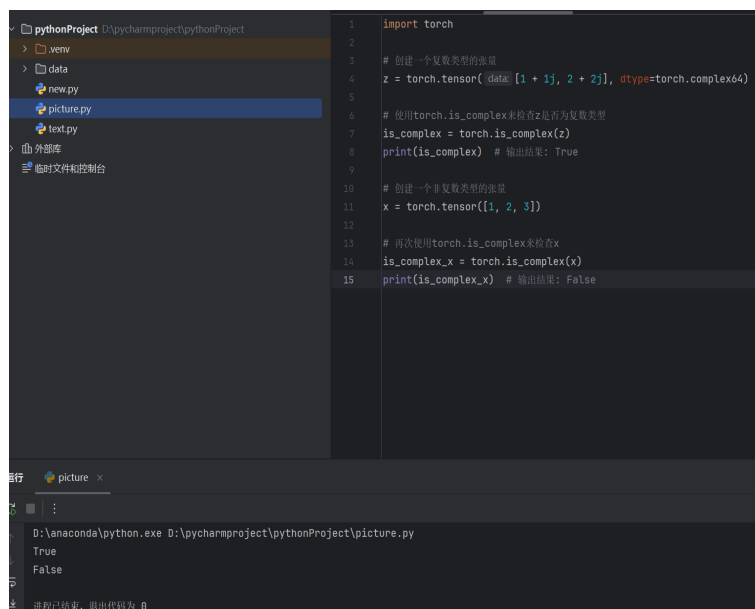
运行 new

```
D:\anaconda\python.exe D:\pycharmproject\pythonProject\new.py
Shape of X [N, C, H, W]: torch.Size([64, 1, 28, 28])
Shape of y: torch.Size([64])
Using cpu device
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    ...
  )
)
```

自动求导的本质：forward 函数计算从输入 Tensors 获得的输出 Tensors。
而 backward 函数接收输出 Tensors 对于某个标量值的梯度，并且计算输入
Tensors 相对于该相同标量值的梯度。

定义 torch.autograd.Function 的子类并实现 forward 和 backward 函数

1.20 pytorch : is-complex 函数运用



```
1 import torch
2
3 # 创建一个复数类型的张量
4 z = torch.tensor([1 + 1j, 2 + 2j], dtype=torch.complex64)
5
6 # 使用torch.is_complex来检查z是否为复数类型
7 is_complex = torch.is_complex(z)
8 print(is_complex) # 输出结果: True
9
10 # 创建一个非复数类型的张量
11 x = torch.tensor([1, 2, 3])
12
13 # 再次使用torch.is_complex来检查x
14 is_complex_x = torch.is_complex(x)
15 print(is_complex_x) # 输出结果: False
```

运行 picture x

D:\anaconda\python.exe D:\pycharmproject\pythonProject\picture.py

True

False

进程已结束, 退出代码为 0

`torch.is-complex` 是 PyTorch 中的一个函数, 用于判断给定的张量是否是复数数据类型。

2 解题感悟

夏季学期的第四周, 我简单学习了调试与性能分析, 元编程, pytorch 在学习调试与性能分析时, 我发现掌握工具的使用至关重要。PyTorch 提供了强大的调试工具, 此外, 使用 `torch.profiler` 进行性能分析, 可以识别瓶颈, 优化模型的训练速度。

元编程方面, PyTorch 的动态计算图特性让我能够灵活地构建和修改模型。这种灵活性使得实验新想法变得更加高效, 尤其是在研究阶段。

在学习 PyTorch 的过程中, 我通过运行不同的模型和算法, 加深了对深度学习的理解。社区资源丰富, 文档详尽, 结合实际项目的练习, 使我在短时间内了解了 PyTorch 的核心概念和应用。

虽然对于这几个工具我只是进行了初步学习, 但是我的视野得到了拓展, 此后有对于相关工具的使用需要时也能更快地上手操作。

3 gitbit 账号链接:<https://github.com/Cecilia-hub>