20377199 赵芮箐 第9周作业

作业内容： 生成器和迭代器有两种常见的使用场景。 一. 后项需要前项导出，且无法通过列表推导式生成。例如，时间序列中的“随机游走”便是一种满足上述条件的序列数据。其公式为$$X_t = \mu + X_{t-1} + w_t$$，其中μ为漂移量，w_{t}是满足某种条件的独立同分布的随机变量，这里假设其服从正态分布N（0, σ^2）。本题要求写出实现该功能的迭代器函数。具体要求如下： 1. 实现random_walk生成器，输入参数μ, X_0, σ^2，N，函数将迭代返回N个随机游走生成的变量。 2. 利用zip，实现拼合多个random_walk的生成器，以生成一组时间上对齐的多维随机游走序列。 二. 需要迭代的内容数据量过大，无法一次性加载。例如，在图像相关的深度学习任务中，由于数据总量过大，一次性加载全部数据耗时过长、内存占用过大，因此一般会采用批量加载数据的方法。（注：实际应用中由于需要进行采样等操作，通常数据加载类的实现原理更接近字典，例如pytorch中的Dataset类。）现提供文件FaceImages.zip（http://vis-www.cs.umass.edu/fddb/originalPics.tar.gz，其中包含5000余张人脸图片。要求设计FaceDataset类，实现图片数据的加载。具体要求： 1. 类接收图片路径列表 2. 类支持将一张图片数据以ndarray的形式返回（可以利用PIL库实现）。 3. 实现__iter__方法。 4. 实现__next__方法，根据类内的图片路径列表，迭代地加载并以ndarray形式返回图片数据。 请实现上述生成器和迭代器并进行测试。

任务一：实现random_walk生成器，并利用zip，生成一组时间上对齐的多维随机游走序列

```
# 实现random_walk生成器
def random_walk(mu, x, sigma_square, N):
    for i in range(N):
        yield x
        w_t = np.random.normal(0,math.sqrt(sigma_square))
        x = mu + x + w_t

# 测试
for i in random_walk(1, 1, 4, 10):
    print(i)

# 实现多维随机游走序列生成器，PS:这里以三维为例
def random_walk_vector(mu, x, s1, s2, s3, N):
    vector = []
    walk1 = list(random_walk(mu, x, s1, N))
    walk2 = list(random_walk(mu, x, s2, N))
    walk3 = list(random_walk(mu, x, s3, N))
    for vec in zip(walk1, walk2, walk3):
        vector.append(vec)
    return vector

# 测试
vector = random_walk_vector(1, 1, 0.01, 25, 100, 10)
for vec in vector:
    print(vec)
```

结果展示：

- random_walk 生成器

```
1
4.845186142428609
10.017391759349099
13.748841896359684
13.696579528454796
15.781863387517582
19.008870590438647
21.096945559890194
20.86764977180739
23.872774292635754
```

- 时间上对齐的多维随机游走序列

```
(1, 1, 1)
(2.046604645003928, -0.1888128560612654, -2.0618907236997135)
(3.0213562595546337, -5.311273677728089, -8.78736800028675)
(4.0653940309944865, 2.6715515490867627, 12.017483254899298)
(4.994985149377107, 9.500176833677646, 3.121474647903181)
(5.8537458191229845, 11.947862112394889, -5.535923849995781)
(6.911322201338584, 16.195056091463897, -19.806094671576396)
(7.857864045417101, 15.114393564849234, -30.53532490870269)
(8.82281340648346, 14.727928546968577, -28.798687238013805)
(9.897249692897576, 7.766461324499983, -20.04201206110895)
```

> 看的出σ越大，序列在该维度游走的越不稳定，更随机

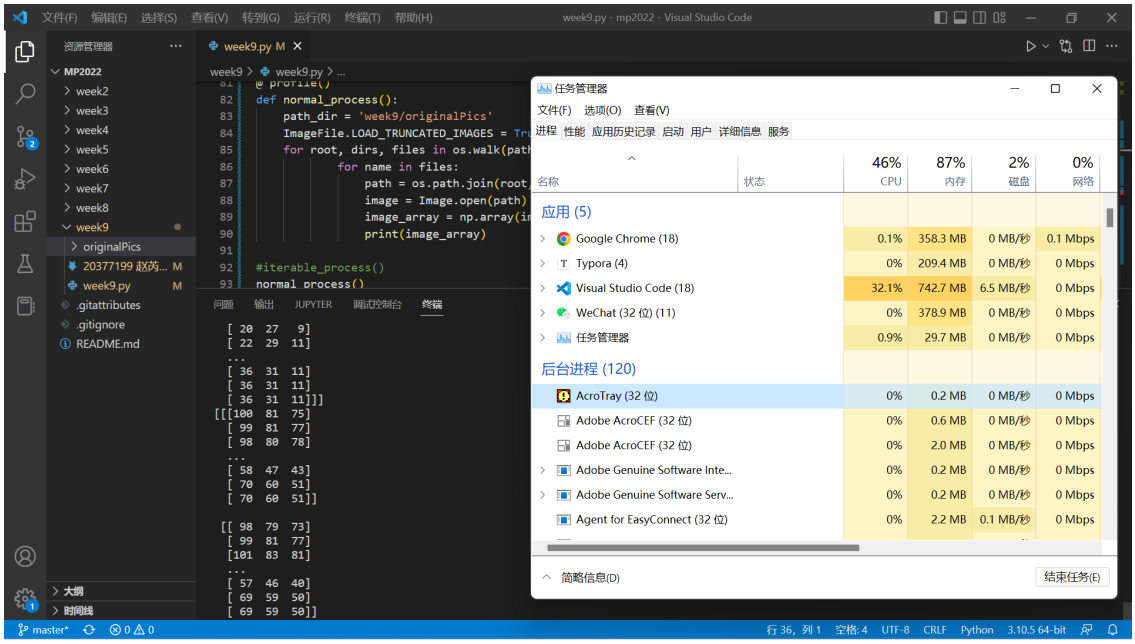
任务二：设计FaceDataset类，实现图片数据的加载

```python
# 设计FaceDataset类，实现图片数据的加载
class FaceDataset():
    def __init__(self, index=0, max=sys.maxsize):
        self._pathlist = []
        self._index = index                             # 从第几张开始加载
        self._max = max                                 # 最多加载多少张

    def load_path(self, path_dir):
        print("-----开始加载图片路径-----")
        for root, dirs, files in os.walk(path_dir): # 生成器，遍历目录
            for name in files:
                self._pathlist.append(os.path.join(root, name))
        print(f"-----图片路径加载完成，共{len(self._pathlist)}张图片-----")
        if self._max > len(self._pathlist):
            self._max = len(self._pathlist)
        self._path = self._pathlist[0]

    def process(self):
        # 当遇到数据截断的图片，PIL不报错，进行下一个
        ImageFile.LOAD_TRUNCATED_IMAGES = True
        image = Image.open(self._path)
        image_array = np.array(image)
        return image_array

    def __iter__(self):
        return self

    def __next__(self):
```

```
        if self._index+1 <= self._max:
            self._path = self._pathlist[self._index]
            self._index += 1
            return self.process()
        else:
            raise StopIteration('{}张图片已处理完毕'.format(self._max))

# 测试
@ profile()
def normal_process():
    path_dir = 'week9/originalPics'
    ImageFile.LOAD_TRUNCATED_IMAGES = True
    for root, dirs, files in os.walk(path_dir):
            for name in files:
                path = os.path.join(root, name)
                image = Image.open(path)
                image_array = np.array(image)
                print(image_array)

@ profile()
def iterable_process():
    path_dir = 'week9/originalPics'
    face = FaceDataset()
    face.load_path(path_dir)
    for img in face:
        print(img)
        #pass

#normal_process()
iterable_process()
```

结果展示：

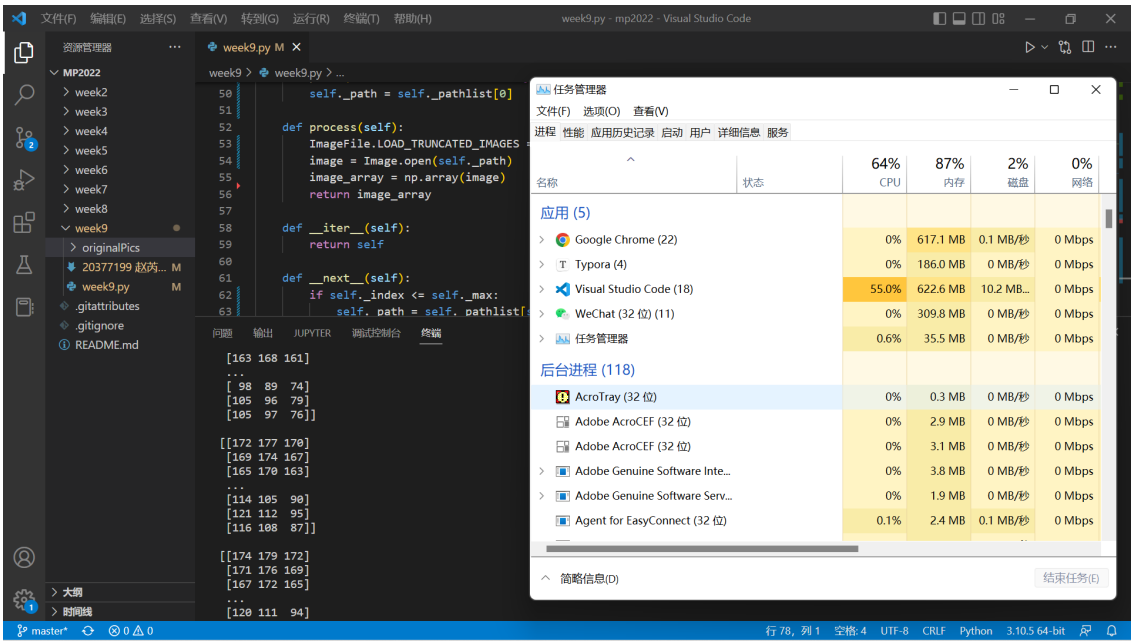
不批量处理：

- 

> 运行CPU占用一直维持在30%-40%左右，内存占用一直维持在750MB左右

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    81     53.4 MiB     53.4 MiB           1   @ profile()
    82                                         def compare():
    83     53.4 MiB      0.0 MiB           1       path_dir = 'week9/originalPics'
    84     53.4 MiB      0.0 MiB           1       ImageFile.LOAD_TRUNCATED_IMAGES = True
    85     59.6 MiB  -1368.1 MiB         809       for root, dirs, files in os.walk(path_dir):
    86     60.1 MiB -48544.6 MiB       29012               for name in files:
    87     60.1 MiB -47737.2 MiB       28204                   path = os.path.join(root, name)
    88     60.1 MiB -55686.4 MiB       28204                   image = Image.open(path)
    89     60.1 MiB -47121.6 MiB       28204                   image_array = np.array(image)
    90     60.1 MiB -47739.0 MiB       28204                   print(image_array)
```

- 利用FaceDataset处理：



> 运行CPU占用一直维持在40%-50%左右，内存占用一直维持在600MB左右

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    72     53.3 MiB     53.3 MiB           1   @ profile()
    73                                         def test():
    74     53.3 MiB      0.0 MiB           1       path_dir = 'week9/originalPics'
    75     53.3 MiB      0.0 MiB           1       face = FaceDataset()
    76     56.4 MiB      3.1 MiB           1       face.load_path(path_dir)
    77     63.6 MiB -363637.4 MiB       28205        for img in face:
    78     63.6 MiB -363629.2 MiB       28204            print(img)
    79                                             #pass
```

用profile()观察发现，如果处理全部数据的话，迭代器其实反而没啥优势啊，直接用for循环处理还内存少点。

于是去了解了一下Dataset 和 DataLoader：

- https://blog.csdn.net/weixin_45901519/article/details/115672355
- https://blog.csdn.net/jx69693678nab/article/details/103819766

为了**不占用过多内存**，我们需要将图片的所有**地址**（并不是所有数字化图片）加载到内存中，需要多少图片数据的时候就从**内存中解析多少图片地址**，这样有效且合理地使用内存，也不会耽误时间。

就好处就是说，因为在深度学习的时候，全部的数据集可能会很大，但我们是拿训练集进行训练，所以，每次批量加载就好了。规定好batch_size，利用迭代器，就可以实现小批量循环迭代式的读取，就避免了耗时太长或者内存有限的问题（所以上面处理全部数据+只是print看不出来个啥）

代码：

https://github.com/rachhhhing/mp2022_python/blob/master/week9/week9.py

Ref：

- 图片的ndarray形式：https://blog.csdn.net/yideqianfenzhiyi/article/details/79193657
- os.walk：https://www.runoob.com/python/os-walk.html