hw2

- SortData方法, 以学生某个属性(str类型, 是 'name', 'stu_num', 'gender', 'age'的其中之一)作为输入, 将 data按该属性从小到大排序. 可以假定不会输入非学生属性的字符串. 例如, 执行 self.Sort('stu_num')后, data的学生信息按学号从小到大排序, 变为

```
[["Bob", "003", "M", 20],["Aaron", "243", "M", 18], ["Eric", "249", "M", 19]]
```

```
18      def SortData(self,mykey):
19          self.data.sort(key = lambda x:x[self.dict1[mykey]]) #方便起见，用lambda表达式写key函数
```

定义一个函数，输入为x，返回x[dict1[mykey]]。其中 mykey 是传入函数的排序依据(str类型)，dict1的定义如下：

```
9           self.dict1 = {'name':0,'stu_num':1,'gender':2,'age':3} #sortdata 时备用，将字符串转化为下标
```

扩展: sort自定义排序

```python
from functools import cmp_to_key


1 usage
def comp(x, y):
    if x == y:
        return 0
    if str(x) + str(y) < str(y) + str(x):
        return 1
    else:
        return -1


nums = [1, 20, 0, 3, 10]
nums.sort(key=cmp_to_key(comp))
```

排序后nums: [3, 20, 1, 10, 0]

Hw5 遗传算法解决TSP

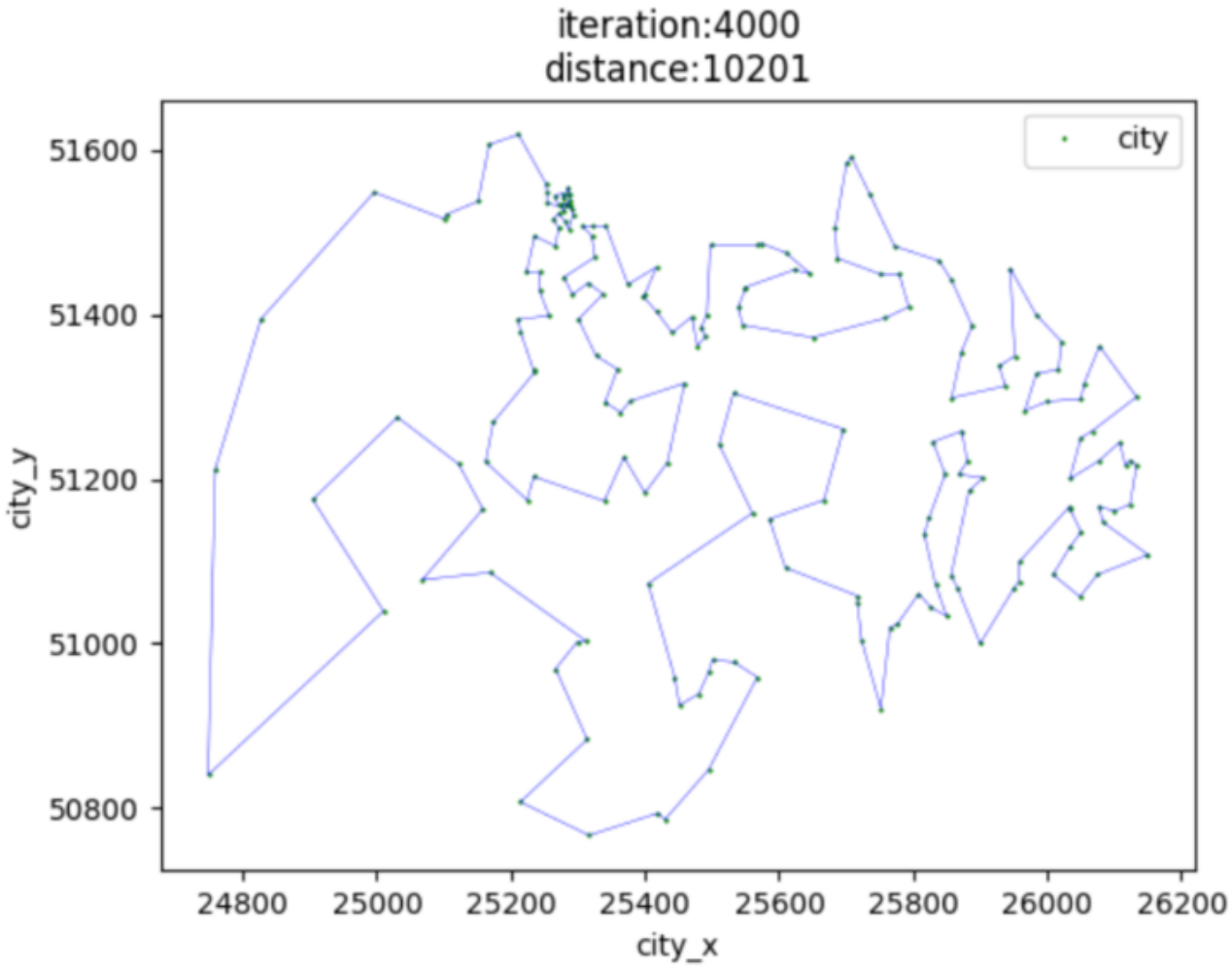
为了方便批改作业, 我们统一用类 `GeneticAlgTSP`来编写遗传算法的各个模块, 并分析算法性能. 该类需包含以下方法:

- 构造函数 `__init__()`, 输入为TSP数据集文件名 `filename`, 数据类型 `str`. 例如 `"dj38.tsp"`是Djibouti的38个城市坐标数据文件; `"ch71009.tsp"`是China的71009个城市坐标数据文件. 我们需要在构造函数中读取该文件中的数据, 存储到类成员 `self.cities`中(数据类型自定, 建议存储为 `numpy`数组). 同时在构造函数中初始化种群, 存储到类成员 `self.population`中(数据类型自定).
- 求解方法 `iterate()`, 输入为算法迭代的轮数 `num_iterations`, 数据类型 `int`. 该方法是基于当前种群 `self.population`进行迭代(不是从头开始), 返回迭代后种群中的一个较优解, 数据类型 `list`, 格式为1-n个城市编号的排列. 例如, 对于n=5的TSP问题, 迭代后返回的较优解形如 `[1,3,4,5,2]`, 表示当前较好的游览城市次序为1-3-4-5-2-1.

可以在类中编写其他方法以方便编写并分析遗传算法的性能. 请在代码注释或实验报告中说明每个方法/模块的功能.

改进点

- 贪心初始化
  - 随机选择一个起始点，每次都选择距离最近的城市，直到回到自己
  - 可以得到一个局部最优解，赢在起跑线上
  - 也有可能因为跳不出局部，比如qa194.tsp就只能收敛到9390
- 顺序交叉
  - 顺序交叉证明是比部分映射交叉更好的方法，但是结合起来用效果更好
- 锦标赛选择
  - 原理：先在群体中随机选择 k 个个体（放回或不放回）进行比较，适应值最好的个体被选择作为生成下一代的父体。
  - 比轮盘赌好
- 多种变异方法
  - 2-opt变异，倒置变异，插入变异，分别给他们一个概率区间，如果随机数位于其中则选择它
  - 概率分别为 `0.01`，`0.01~0.1`， `0.1~0.25`
- 2-opt变异方法
  - 就是随机选取两点交换，判断路径长度有无减小，就这么反复地循环进行直到当前路径长度无法再减小
  - 破局的关键，因为很多时候就是差那么几个点的顺序，而普通变异又很难说刚好变异到那里去，所以采取这种贪心的办法
  - 缺点，极其耗时间，所以把进入它的概率调小为0.01，期望每100次能有1次就行
  - 在实验结果中我将对比使用2-opt与不使用（优化后和优化前），来显示出2-opt的优势及缺点


iteration:4000
distance:10201
city
51600
51400
51200
51000
50800
city_y
24800
25000
25200
25400
25600
25800
26000
26200
city_x

Hw7 手写数字分类

在MNIST数据集上完成手写数字图片分类任务, 具体要求如下:

- 示例代码中已经给出从 `.pth`文件加载数据集的代码(`Tensor`类型), 命名为 `(train_data, train_labels), (test_data, test_labels)`, 分别是训练图像, 训练标签, 测试图像和测试标签. 请基于这些 `Tensor`完成训练任务以及测试任务.
- 用**pytorch**搭建卷积神经网络(在类中 `MyConvNet`来实现), 在训练集上训练模型, 并在测试集上完成分类测试.
- 为了方便批改作业, `MyConvNet`的构造函数请不要使用任何形参.
- 测试时至少用分类正确率来衡量性能(可以添加其他指标来衡量性能并在报告中呈现).
- 训练结束后, 务必使用 `torch.save()`**保存模型**(即神经网络的参数等信息). **此次作业需要额外上传模型.** 模型的文件名格式为hw7_学号_姓名拼音.pth, 例如 `hw7_21000000_zhangsan.pth`.
- 所有内容在同一个 `.py`代码文件上实现.
- 作业提交时将 `.py`代码文件和 `.pth`模型文件提交到 `本科生实验hw7_code`文件夹中, 实验报告提交到 `本科生实验hw7_report`文件夹中. 请不要提交其他无关文件.

卷积层数的影响

训练结果(卷积层数)

评价指标\层数	1	2	3	4
损失	0.41	0.33	0.41	0.81
准确率	0.86	0.88	0.88	0.74

测试结果(卷积层数)

评价指标\层数	1	2	3	4
损失	0.28	0.17	0.19	0.46
准确率	0.92	0.95	0.94	0.91

结论(卷积层数)

在本次实验中，2层的网络表现是最好的，随着层数增多，模型可能会损失一些关键特征。 这一点在4层网络的数据中体现的最为明显，4层网络有625个通道，但池化后图像大小为1×1，图像太小了，损失了许多特征，使得后面的全连接层效果也不好，这是靠通道数无法解决的。

全连接层大小的影响

在本次实验中，我设置了两个全连接层，接下来，我将探究第一层全连接层输出维数的大小对输出的影响。 全连接层大小示例图

训练结果(全连接层大小)

评价指标\大小	50	100	300	500	700	900	1100	1300	1500	1700
损失	0.33	0.16	0.05	0.03	0.03	0.02	0.02	0.02	0.02	0.11
准确率	0.88	0.95	0.98	0.99	0.99	1.00	1.00	0.99	0.99	0.97

测试结果(全连接层大小)

评价指标\大小	50	100	300	500	700	900	1100	1300	1500	1700
损失	0.21	0.19	0.14	0.15	0.14	0.19	0.14	0.15	0.15	0.22
准确率	0.94	0.95	0.96	0.96	0.96	0.95	0.96	0.96	0.96	0.93

可以看到，增加了全连接层后，模型的记忆功能、数据处理功能显著增强。 这也让训练的效果直接有了大幅提升，最终能够几乎完全拟合训练数据。 而且测试效果也很好，没有出现严重过拟合的现象。 最终在**1100**模型性能最好。 关于具体原因，我想是因为卷积层会输出许多特征数据，而这时候全连接层不能直接降维降到很低，否则无法很好地利用这些特征数据。 卷积层应该逐步降维，甚至可以多加一些层，以保证这些特征数据的充分利用。而在本例中，将该维度从50变为100、从100变为300，模型从收敛速度到最终结果都有了一个非常好的提升，我想就可以反映这一点。

数据增强

评价指标\层数	准确率	损失
只有原数据	0.96	0.14
原数据与所有增强数据	0.95	0.17
原数据、旋转、水平翻转、剪切、缩放	0.96	0.13
原数据、旋转、剪切、缩放	0.97	0.11
原数据、水平翻转、剪切、缩放	0.96	0.14
原数据、旋转、剪切	0.97	0.10
原数据、缩放	0.96	0.13

结论(数据增强)

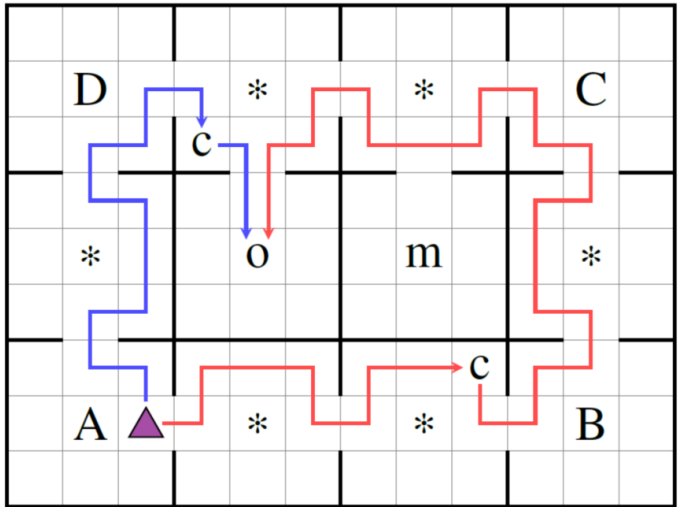
当网络更加复杂、模型对训练数据拟合地更好的时候，数据增强会更加有效。 数据增强能够增大数据的多样性，使得模型能够在更广阔的的域中收敛，进而在测试集中有更好的表现。

数据增强选择数据的标准应该使选择的数据更加接近测试集，也就是说更加接近真实世界的情况。 根据以上数据，我认为，在训练集有较好分布的背景下，所选的增强数据在训练时的表现应该略低于原始数据，原因如下。

1. 一方面能够避免模型过拟合；
2. 一方面也能够增加多样性，使得测试集中的一些特殊数据也能被考虑到。

Hw9 表格型Q-Learning

如下图所示, 本次作业的OfficeWorld环境为9*12网格, 其中三角形表示智能体, 黑色粗线表示无法跨越的墙壁, A,B,C,D为四个地标, c表示coffee, m表示mail, o表示office, *表示盆栽. 该环境的可选任务有 'to_a', 'to_b', 'to_c', 'to_d','get_mail', 'get_coffee', 'to_office', 分别表示在不触碰盆栽的前提下到达地标或者物品所在的网格. 下图的蓝色路径表示了任务 'to_office' 的最优解, 而红色路径是一个次优解.



任务	QL结果	稳定episode数
get_coffee	0.92	117
to_office	0.86	368
get_mail	0.81	765
to_a	1.00	1
to_b	0.89	208
to_c	0.81	660
to_d	0.92	104

思考题3

3. 如果将任务修改为"先去地标 B, 然后去地标 D", 那么 Q-Learning 算法理论上可以学出来吗?

理论上可以学出来。然而，这需要正确的奖励设置和充足的训练时间。

我们可以设置当智能体到达地标 B 时获得一定的奖励，并且只有在到达地标 B 之后，才能通过到达地标 D 获得更大的奖励。这样，智能体就有动力先去地标 B，然后再去地标 D。

Q-Learning 算法理论上是可以学习出最优策略的。只需要将地标 B 和地标 D 都看作各自的终止状态，转化为两个子任务，而对于每个子任务，Q-Learning 都可以学习一组局部 Q 值。之后可以将这些子任务的策略合并，使得智能体遵循的全局策略可以贯穿本任务的全过程，从而达到最优解。

在任务被修改为"先去地标B，然后去地标D"时，Q-Learning算法仍然可以学习到策略，但是它可能需要更多的训练时间和更复杂的表示来捕捉到这个新的任务结构。

此时，可以通过对状态进行扩展来表示任务的顺序。可以将状态定义为当前位置和剩余要访问的地标，如（位置A，剩余地标：B，D）。然后，可以对`QLAgent`进行相应的修改，使其能够考虑到这个顺序约束。

思考题4

4、如果任务是"去地标 A 或者 C"，并且到达地标 A 的奖励是 1，到达地标 C 的奖励是 10，这时候 Q-Learning 的表现如何？若无法学出到达地标 C 的策略，可能是什么原因?

（1）理论上，只要 episode 趋于无穷且 epsilon 大于 0，可以学出到达地标 C 的策略，因为到达地标 C 奖励大于到达地标 A 奖励。也即理论上 Q-Learning 可以得到最优解。

但是如果地标 A 距离起点较近且地标 C 距离起点较远，且 epsilon 值较小、学习率较小时，可能过度拟合到局部最优。这时 Q-Learning 的性能表现就很差了，因为它需要更多的 episode 以得到最优解，进而浪费大量时间。

（2）若无法学出到达地标 C 的策略，可能是地标 A 距离起点较近且地标 C 距离起点较远，且 epsilon 值较小使得过度拟合到局部最优、学习率较小使得偶然随机得到的最优解无法较大程度地改变原有局部最优解。

原因如下所示：

1. Q-Learning算法在探索和利用之间需要平衡。如果智能体在探索时由于步数限制等原因没有足够地探索到地标C，或者在早期阶段选择了前往地标A的动作并陷入局部最优解，它可能会没有机会学习到到达地标C的策略。
2. Q-Learning算法的初始条件和随机性对学习过程和结果有影响。如果智能体的初始状态或初始Q值设置不合适，或者学习过程中的随机性导致无法发现到达地标C的优势，那么它可能无法学习到最优策略。

hw10 深度强化学习实践

使用pytorch, 在OpenAI gym的"CartPole-v1"环境实现一种深度强化学习算法(包括但不限于: DQN, Double DQN, Dueling DQN, AC, PPO, SAC等).

超参数调整建议

AI对于解决断崖式下降的建议

DQN训练过程在达到峰值后容易出现断崖式下降的原因可能有以下几点：

- 探索率（epsilon）的衰减速度过快，导致智能体在训练过程的早期就停止了探索和利用，这可能会导致智能体陷入次优策略并无法从中恢复。
- 用于经验回放的缓冲区大小相对较小（5000），这可能不足以让智能体学习到一个好的策略。增加缓冲区大小可能有助于智能体从更多样化的经验中学习，并避免过度拟合到一小组经验中。
- 目标网络更新的频率过低（每10步），这可能会导致训练过程不稳定。增加目标网络更新的频率可能有助于稳定训练过程，并防止智能体高估Q值。

为了解决这些问题，您可以尝试以下修改：

- 减缓epsilon衰减的速度，或使用不同的衰减计划，以便在训练的早期阶段进行更多的探索。
- 增加缓冲区大小，以便智能体从更多样化的经验中学习。
- 增加目标网络更新的频率，以稳定训练过程。

通过进行这些修改，您可能能够改善DQN训练过程的稳定性和性能，并避免出现您目前遇到的断崖式性能下降

Reward Shaping方法

```
reward = reward - abs(next_state[0])/5
```

将训练过程的 r 改为如下 reward，即使靠中心位置以及角度好的奖励值增大，能使训练过程的稳定性加强并加快训练速度。也有缺陷：该训练结果偏向在中心位置的稳定，对靠边界的情况没有得到较好训练，若实际测试过程出现移向一旁的情况往往不能继续保持稳定。

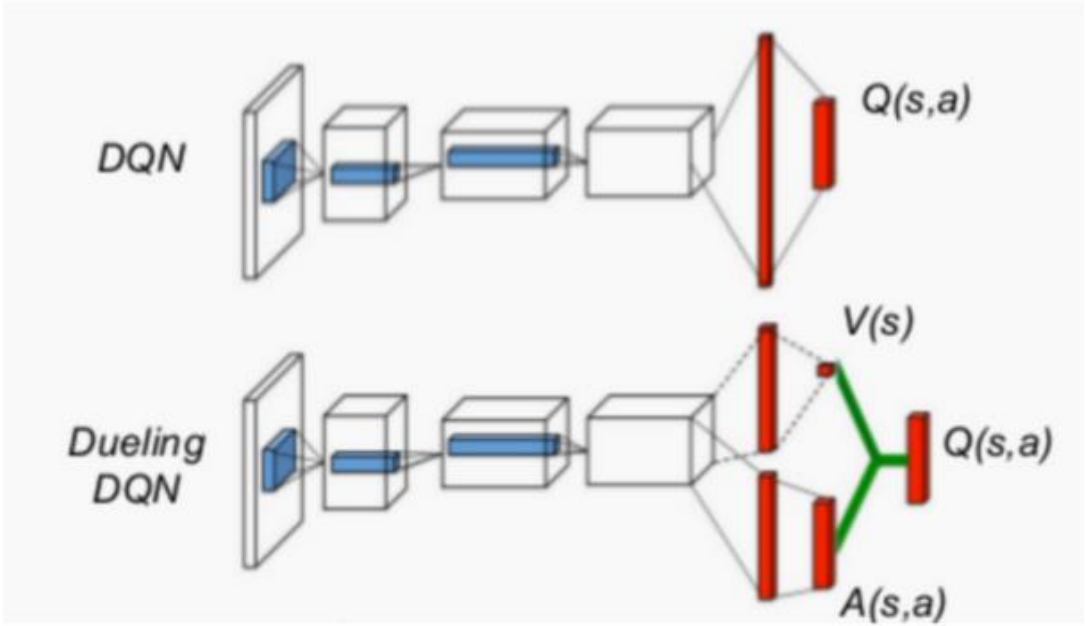
```
#r1 = (env.x_threshold - abs(s[0])) / env.x_threshold - 0.8
#r2 = (env.theta_threshold_radians - abs(s[2])) /env.theta_threshold_radians - 0.5
#reward = r1 + r2
```

```
s2, reward, done, _, _ = env.step(a)
r_total += reward
# 优化
cart_position, cart_velocity, pole_angle, pole_angular_velocity = s2
if abs(cart_position) > agent.Cart_Position_bound:
    r1 = 0.5 * agent.negative_reward
else:
    r1 = agent.negative_reward * abs(cart_position) / agent.Cart_Position_bound + \
         0.5 * (-agent.negative_reward)
if abs(pole_angle) > (agent.Pole_Angle_bound / 2):  # 严苛的角度要求
    r2 = 0.5 * agent.negative_reward
else:
    r2 = agent.negative_reward * abs(pole_angle) / (agent.Pole_Angle_bound / 2) + \
         0.5 * (-agent.negative_reward)
reward += r1 + r2
```

Dueling DQN图示

Dueling DQN 算法原理

1. `Network` 架构:
   - 动作值函数分解为状态值函数和优势函数
   - 使用两个全连接层，单独估计状态值函数和优势函数，最后汇总输出单一Q值



   - θ 为共享网络层参数
   - α 为优势网络函数网络层参数
   - β 为状态值函数网络层参数

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)$$

Dueling DQN网络架构

```
class Net1(nn.Module):
    def __init__(self) -> None:
        super().__init__()
        self.feature = nn.Sequential(
            nn.Linear(n_states, 128),
            nn.ReLU()
        )

        self.advantage = nn.Sequential(
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, n_actions)
        )

        self.value = nn.Sequential(
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, 1)
        )

    def forward(self, x):
        x = self.feature(x)
        advantage = self.advantage(x)
        value = self.value(x)
        return value + advantage - advantage.mean()
```