第四次作业

黄磊　计 702　2022E8013282156

一、　简答题

1. 试列举几种常见的半监督学习方法。比较有监督学习、无监督学习、半监督学习、 主动学习以及强化学习的异同；

半监督学习：自训练方法、多视角算法、生成模型、转导 SVM，基于图的算法。

监督学习最大的特点就是其数据集带有标签。换句话来说，监督学习就是在带有标签的训练数据中学习模型，然后对某个给定的新数据利用模型预测它的标签。

让学习器不依赖外界交互、自动地利用未标记样本来提升学习性能，就是半监督学习，将大量的无类标签的样例加入到有限的有类标签的样本中一起训练来进行学习，以达到提升性能的目的。

无监督学习的特点是，模型学习的数据没有标签，因此无监督学习的目标是通过对这些无标签样本的学习来揭示数据的内在特性及规律，其代表就是聚类。

自监督学习主要是利用辅助任务（pretext）从大规模的无监督数据中挖掘自身的监督信息，通过这种构造的监督信息对网络进行训练，从而可以学习到对下游任务有价值的表征。

主动学习算法可以交互式地查询用户以用真实标签标记新的数据点，被允许从尚未标记的样本池中主动选择下一个要标记的可用样本子集。其基本思想是：如果允许机器学习算法选择它想要学习的数据，它可以在使用更少的训练标签的同时实现更高的精度。

强化学习（Reinforcement learning，RL）讨论的问题是一个智能体 (agent) 怎么在一个复杂不确定的 环境(environment) 里面去极大化它能获得的奖励。通过感知所处环境的 状态(state) 对 动作(action) 的 反应 (reward)， 来指导更好的动作，从而获得最大的 收益(return)，这被称为在交互中学习，这样的学习方法就被称作强化学习

2. 试给出协同训练的方法步骤；

输入：标记数据集 L，未标记数据集 U。

- 用 L1 训练视图 X1 上的分类器 f1，用 L2 训练视图 X2 上的分类器 f2；
- 用 f1 和 f2 分别对未标记数据 U 进行分类；
- 把 f1 对 U 的分类结果中，前 k 个最置信的数据（正例 p 个反例 n 个）及其分类结果加入 L2；把 f2 对 U 的分类结果中，前 k 个最置信的数据及其分类结果加入 L1；把这 2（p+n）个数据从 U 中移除；
- 重复上述过程，直到 U 为空集。

输出：分类器 f1 和 f2。其中 f1 和 f2 可以是同一种分类器也可以不是同一种分类器。

二、 编程题

SARSA 和 Q-Learning 都是时序差分算法，SARSA 使用 on-policy 策略，而 Q-Learning 使用的是 off-policy 策略。

Sarsa 的思想是先基于当前状态 S，使用$\epsilon-$贪婪法按一定概率选择动作 A，然后得到奖励 R，并更新进入新状态 S′，基于状态 S′，使用$\epsilon-$贪婪法选择 A′（即在线选择，仍然使用同样的$\epsilon-$贪婪），即：

$$Q(S,A) = Q(S,A) + \alpha\big(R + \gamma Q(S',A) - Q(S,A)\big)$$

Q-Learnig 的思想就是，先基于当前状态 S，使用$\epsilon-$贪婪法按一定概率选择动作 A，然后得到奖励 R，并更新进入新状态 S′，基于状态 S′，直接使用贪婪法从所有的动作中选择最优的 A′（离线选择，不使用同样的$\epsilon-$贪婪），即：

$$Q(S,A) = Q(S,A) + \alpha(R + \gamma \max_a Q(S',A) - Q(S,A))$$

为了更加有效地训练，将 reward 重新设置，如果进洞则为-50，如果 win 则为 100，每次行动的 reward 设置为-1。Baseline 设置为：$lr = 0.1,\ \gamma = 0.9, \epsilon = 0.3, Epoch = 2000$，采用控制变量分别测试其性能，即只改变学习率和折扣因子。使用三个性能指标评估各自性能：

a. 训练过程中赢得游戏的次数占比（Win Percentage）；
b. 训练过程中赢得游戏时候的平均步数；
c. 训练过程中赢得游戏时候的步数与理想步数的均方根误差

画图的时候对 test_reward 进行了一些比例上调整，使得图像更加直观。

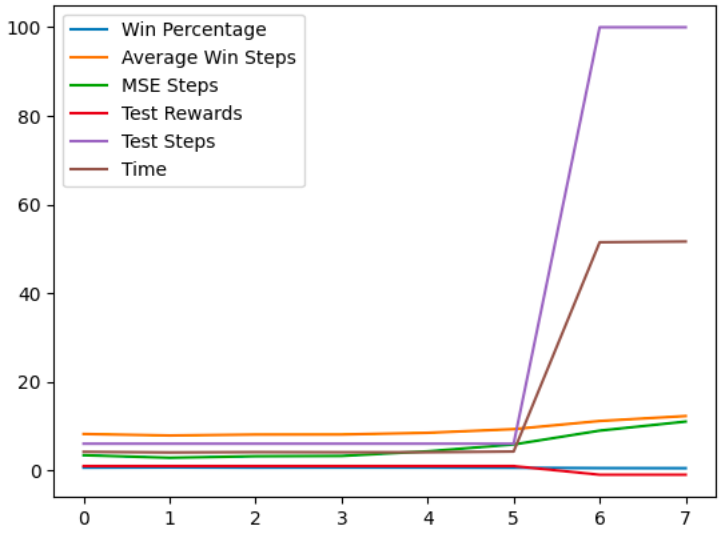
(1) 编程实现 Sarsa 算法实现 Agent 穿越冰湖，分析不同学习率和折扣因子下算法的表现；

1.1 首先，固定 $\gamma = 0.9$，比较不同的 lr 值对性能的影响：

```
================================================ lr =  0.1   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.5915
训练过程中赢游戏时平均步数: 8.191885038038885
训练过程中赢游戏时步数与理想步数的均方根误差: 3.399940330638458
test steps:6,  test reward:95
程序运行时间为:  4.170410871505737
================================================ lr =  0.001   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.637
训练过程中赢游戏时平均步数: 7.858712715855573
训练过程中赢游戏时步数与理想步数的均方根误差: 2.8239834030313564
test steps:6,  test reward:95
程序运行时间为:  4.024261236190796
================================================ lr =  0.01   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.5975
训练过程中赢游戏时平均步数: 8.07112970711297
训练过程中赢游戏时步数与理想步数的均方根误差: 3.1695465140304333
test steps:6,  test reward:95
程序运行时间为:  4.1071014404296875
================================================ lr =  0.1   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.6125
训练过程中赢游戏时平均步数: 8.081632653061224
训练过程中赢游戏时步数与理想步数的均方根误差: 3.2362739976329444
test steps:6,  test reward:95
程序运行时间为:  4.072105646133423
================================================ lr =  0.2   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.608
训练过程中赢游戏时平均步数: 8.460526315789474
训练过程中赢游戏时步数与理想步数的均方根误差: 4.282384357980221
test steps:6,  test reward:95
程序运行时间为:  4.073087692260742
```

```
================================================ lr =  0.5   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.568
训练过程中赢游戏时平均步数: 9.286971830985916
训练过程中赢游戏时步数与理想步数的均方根误差: 5.824154414838887
test steps:6,  test reward:95
程序运行时间为:  4.232595205307007
================================================ lr =  0.75   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.493
训练过程中赢游戏时平均步数: 11.125760649087221
训练过程中赢游戏时步数与理想步数的均方根误差: 8.949939657801952
test steps:100,  test reward:-100
程序运行时间为:  51.46882915496826
================================================ lr =  0.9   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.471
训练过程中赢游戏时平均步数: 12.24416135881104
训练过程中赢游戏时步数与理想步数的均方根误差: 10.997200956367719
test steps:100,  test reward:-100
程序运行时间为:  51.63462209701538
```

可以看出，随着 lr 的增大，SARSA 算法变得更加不稳定，平均步数和均方根误差均有增大，赢游戏概率有下降趋势，甚至会出现游戏失败的情况，这是因为学习率增大有可能导致欠学习的情况；在程序运行时间上差别不大。

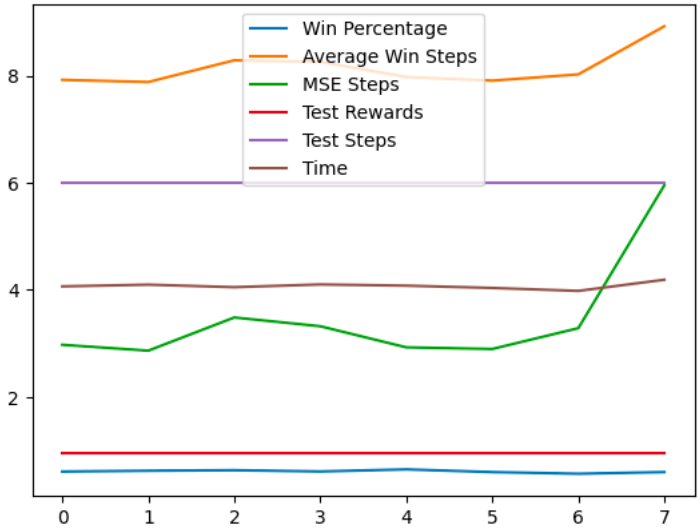


1.2 固定 $lr = 0.1$，比较不同的 γ 值对性能的影响：

```
================================================ lr =  0.1   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.609
训练过程中赢游戏时平均步数: 7.922824302134647
训练过程中赢游戏时步数与理想步数的均方根误差: 2.97554344362392
test steps:6,  test reward:95
程序运行时间为:  4.066453456878662
================================================ lr =  0.1   gamma =  0.01 ================================================
训练过程中赢游戏的占比: 0.624
训练过程中赢游戏时平均步数: 7.882211538461538
训练过程中赢游戏时步数与理想步数的均方根误差: 2.8662771798615183
test steps:6,  test reward:95
程序运行时间为:  4.098522186279297
================================================ lr =  0.1   gamma =  0.1 ================================================
训练过程中赢游戏的占比: 0.634
训练过程中赢游戏时平均步数: 8.287854889589905
训练过程中赢游戏时步数与理想步数的均方根误差: 3.485549257686713
test steps:6,  test reward:95
程序运行时间为:  4.051585912704468
================================================ lr =  0.1   gamma =  0.2 ================================================
训练过程中赢游戏的占比: 0.6105
训练过程中赢游戏时平均步数: 8.25880425880426
训练过程中赢游戏时步数与理想步数的均方根误差: 3.321929715574518
test steps:6,  test reward:95
程序运行时间为:  4.102388858795166
================================================ lr =  0.1   gamma =  0.5 ================================================
训练过程中赢游戏的占比: 0.6495
训练过程中赢游戏时平均步数: 7.9753656658968435
训练过程中赢游戏时步数与理想步数的均方根误差: 2.927925591250281
test steps:6,  test reward:95
程序运行时间为:  4.0797953605651855
```

```
================================================ lr =  0.1   gamma =  0.7 ================================================
训练过程中赢游戏的占比: 0.599
训练过程中赢游戏时平均步数: 7.908180300500835
训练过程中赢游戏时步数与理想步数的均方根误差: 2.8975264334233914
test steps:6,  test reward:95
程序运行时间为:  4.037039756774902
================================================ lr =  0.1   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.5685
训练过程中赢游戏时平均步数: 8.02462620932278
训练过程中赢游戏时步数与理想步数的均方根误差: 3.2861212375745863
test steps:6,  test reward:95
程序运行时间为:  3.982120990753174
================================================ lr =  0.1   gamma =  1 ================================================
训练过程中赢游戏的占比: 0.599
训练过程中赢游戏时平均步数: 8.923205342237061
训练过程中赢游戏时步数与理想步数的均方根误差: 5.953352333490586
test steps:6,  test reward:95
程序运行时间为:  4.190203905105591
```

从结果看出，折扣因子对于模型性能影响并不大。但是如果折扣因子太小或者太大，模型可能会出现训练不稳定的情况，收敛步数的方差可能会较大。

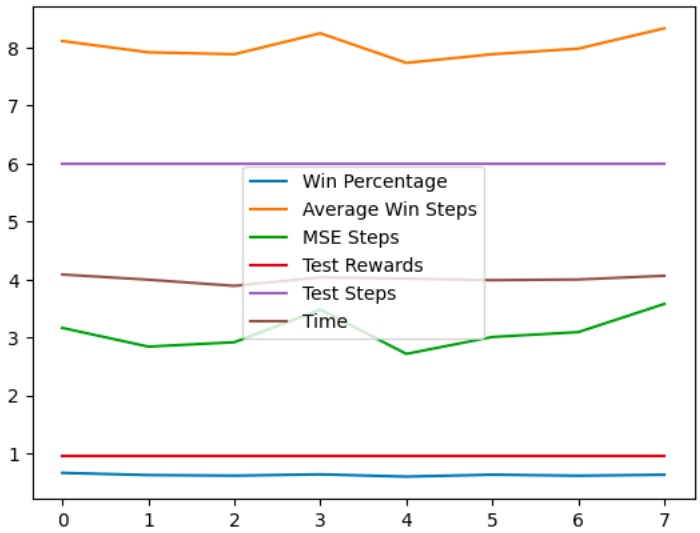


上述结果可以看出来， $lr = 0.01, \gamma = 0.5 \sim 0.75$ 会是比较好的参数选择。

(2) 编程实现 Qlearning 算法实现 Agent 穿越冰湖分析不同学习率和折扣因子下算法的表现；

```
================================================ lr =  0.1   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.663
训练过程中赢游戏时平均步数: 8.116892911010558
训练过程中赢游戏时步数与理想步数的均方根误差: 3.1657337674820574
test steps:6,  test reward:95
程序运行时间为:  4.086863994598389
================================================ lr =  0.001   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.6255
训练过程中赢游戏时平均步数: 7.9224620303757
训练过程中赢游戏时步数与理想步数的均方根误差: 2.8426634362285528
test steps:6,  test reward:95
程序运行时间为:  3.997929334640503
================================================ lr =  0.01   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.6155
训练过程中赢游戏时平均步数: 7.888708367181153
训练过程中赢游戏时步数与理想步数的均方根误差: 2.9179129672565915
test steps:6,  test reward:95
程序运行时间为:  3.8908591270446777
================================================ lr =  0.1   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.6375
训练过程中赢游戏时平均步数: 8.250980392156864
训练过程中赢游戏时步数与理想步数的均方根误差: 3.4810410883561125
test steps:6,  test reward:95
程序运行时间为:  4.037646532058716
================================================ lr =  0.2   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.599
训练过程中赢游戏时平均步数: 7.739565943238731
训练过程中赢游戏时步数与理想步数的均方根误差: 2.716732300144982
test steps:6,  test reward:95
程序运行时间为:  4.015830039978027
```

```
================================================ lr =  0.5   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.632
训练过程中赢游戏时平均步数: 7.889240506329114
训练过程中赢游戏时步数与理想步数的均方根误差: 3.008689945785941
test steps:6,  test reward:95
程序运行时间为:  3.9897499084472656
================================================ lr =  0.75   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.6145
训练过程中赢游戏时平均步数: 7.985353946297803
训练过程中赢游戏时步数与理想步数的均方根误差: 3.092814318195458
test steps:6,  test reward:95
程序运行时间为:  4.001024007797241
================================================ lr =  0.9   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.631
训练过程中赢游戏时平均步数: 8.335974643423137
训练过程中赢游戏时步数与理想步数的均方根误差: 3.5797458054582085
test steps:6,  test reward:95
程序运行时间为:  4.065442800521851
```

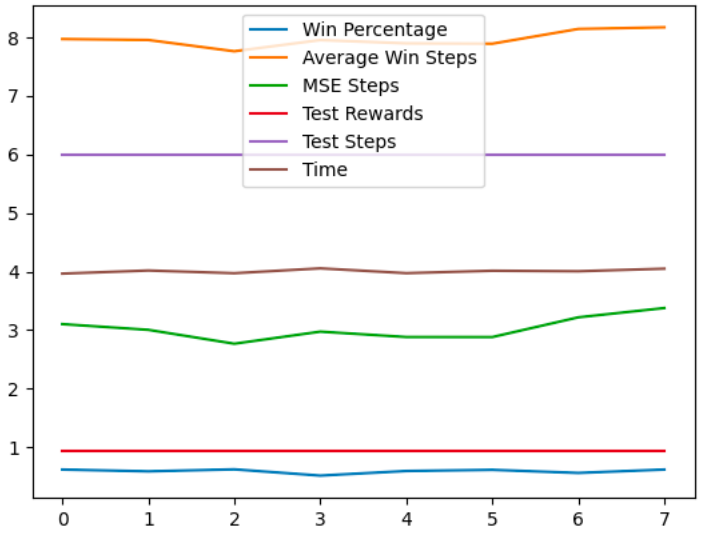


QLearning 算法会更加稳定一些，lr 的改变对整体性能影响并不算很大。但 lr 不能过大，不然会出现模型不稳定的趋势。

固定 $lr = 0.1$，比较不同的 γ 值对性能的影响:

```
================================================ lr =  0.1   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.6235
训练过程中赢游戏时平均步数: 7.964715316760224
训练过程中赢游戏时步数与理想步数的均方根误差: 3.103662115060006
test steps:6,  test reward:95
程序运行时间为:  3.9646637439727783
================================================ lr =  0.1   gamma =  0.01 ================================================
训练过程中赢游戏的占比: 0.593
训练过程中赢游戏时平均步数: 7.948566610455312
训练过程中赢游戏时步数与理想步数的均方根误差: 3.00603664711151
test steps:6,  test reward:95
程序运行时间为:  4.017669916152954
================================================ lr =  0.1   gamma =  0.1 ================================================
训练过程中赢游戏的占比: 0.627
训练过程中赢游戏时平均步数: 7.755980861244019
训练过程中赢游戏时步数与理想步数的均方根误差: 2.7691626104290896
test steps:6,  test reward:95
程序运行时间为:  3.971527576446533
================================================ lr =  0.1   gamma =  0.2 ================================================
训练过程中赢游戏的占比: 0.5205
训练过程中赢游戏时平均步数: 7.9471661863592695
训练过程中赢游戏时步数与理想步数的均方根误差: 2.9752420610441033
test steps:6,  test reward:95
程序运行时间为:  4.0543739795684814
================================================ lr =  0.1   gamma =  0.5 ================================================
训练过程中赢游戏的占比: 0.599
训练过程中赢游戏时平均步数: 7.890651085141903
训练过程中赢游戏时步数与理想步数的均方根误差: 2.88352061756455
test steps:6,  test reward:95
程序运行时间为:  3.97245454788208
```

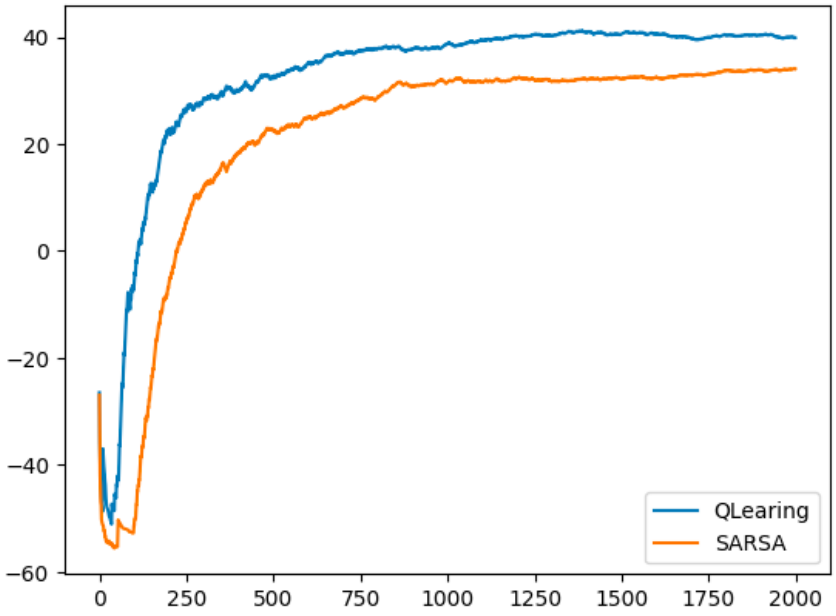
```
================================================ lr =  0.1   gamma =  0.7 ================================================
训练过程中赢游戏的占比: 0.6185
训练过程中赢游戏时平均步数: 7.885206143896524
训练过程中赢游戏时步数与理想步数的均方根误差: 2.8827813749817714
test steps:6,  test reward:95
程序运行时间为:  4.013567924499512
================================================ lr =  0.1   gamma =  0.9 ================================================
训练过程中赢游戏的占比: 0.566
训练过程中赢游戏时平均步数: 8.136925795053003
训练过程中赢游戏时步数与理想步数的均方根误差: 3.219447311169937
test steps:6,  test reward:95
程序运行时间为:  4.004909038543701
================================================ lr =  0.1   gamma =  1 ================================================
训练过程中赢游戏的占比: 0.6235
训练过程中赢游戏时平均步数: 8.16439454691259
训练过程中赢游戏时步数与理想步数的均方根误差: 3.3777183949722343
test steps:6,  test reward:95
程序运行时间为:  4.048730850219727
```



γ 的值对模型性能影响也不是很大。但如果设置过小或过大的话，同样可能会有模型不太稳定、收敛慢的问题。

(3) 分析并实验对比上述两个算法的性能表现

得到其每个轮次前的平均回报图像：



从上述分析可以看出，QLearning 算法相较于 SARSA 更加稳定、收敛更快。本质上是因为 Q-Learning 的 Target Policy 是绝对的 greedy 策略，绝对的 greedy 策略保证了 Agent 在 Q 值进入收敛后不会也不可能再记录可能失败的状态动作的 Q 值，也可以说是 Off-Policy 类的控制方法并不会受到 greedy 策略无探索性的影响，所以才能够产生 Optimal Policy。然而 SARSA 算法使用的 epsilon-greedy 策略来更新 Q 值，也就是说不论何时，Agent 总是有 epsilon 的概率选择非最优的动作，其中失败的动作也始终有一定的概率被选中，并在 Q 函数更新时被记录下来。