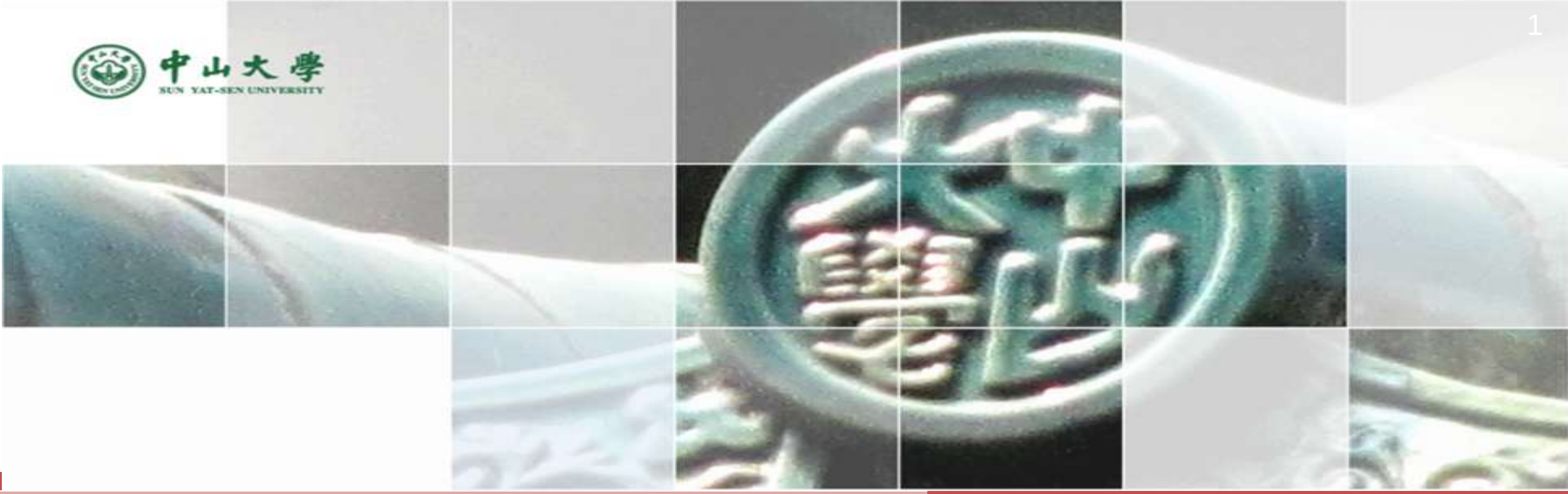


深度强化学习在无人机飞行控制上的应用

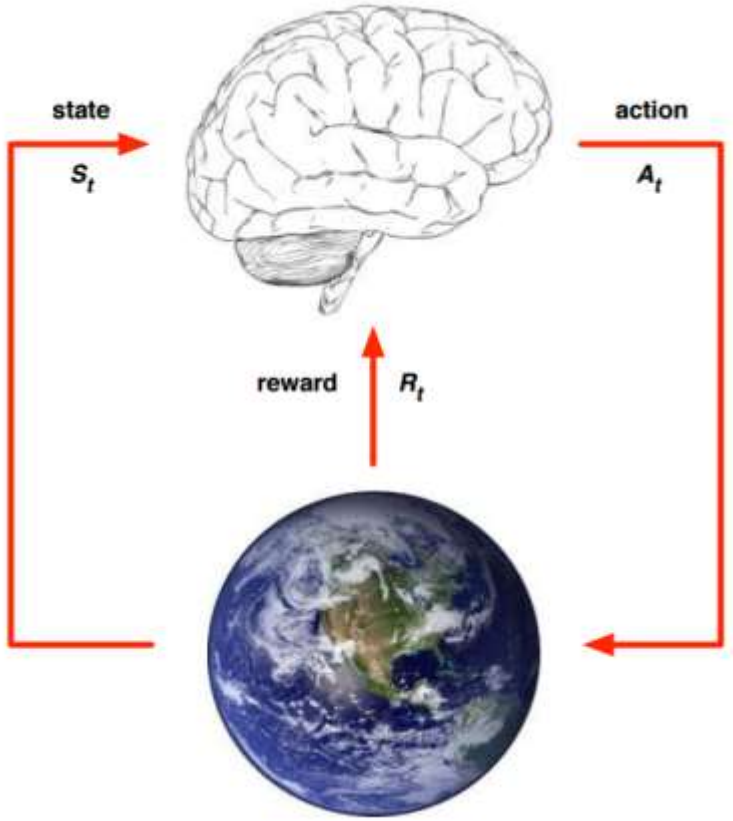
中山大学智能工程学院
古　博
gubo@mail.sysu.edu.cn

内容

- 深度强化学习基础知识介绍
- 深度强化学习在无人机飞行控制上的应用

强化学习建模

- 大脑代表我们的算法执行个体（Agent)，我们可以操作个体来做决策，即选择一个合适的动作（Action）A_t。
- 地球代表我们要研究的环境，它有自己的状态模型。选择了动作A_t后，环境的状态(State)会从S_t变为S_{t+1}，同时得到了采取动作A_t的延时奖励(Reward) R_{t+1}。
- 个体可以继续选择下一个合适的动作，然后环境的状态又会变，又有新的奖励值 ...



强化学习要素（一）

- 环境的状态集S：t时刻环境的状态s_t是环境状态集中某一个状态；
- 个体的动作集A：t时刻个体采取的动作a_t是动作集中某一个动作；
- 环境的奖励R：t时刻个体在状态s_t采取的动作a_t 对应的奖励R_{t+1}会在$t+1$时刻得到；
- 策略(policy)π：它代表个体采取动作的依据，即个体会依据策略π来选择动作。
  - 最常见的策略表达方式是一个条件概率分布$\pi(a|s)$， 即在状态s时采取动作a的概率。即$\pi(a|s)=P(a_t=a|s_t=s)$。此时概率大的动作被个体选择的概率较高。

强化学习要素（二）

- 个体在策略π和状态s时的价值（value），一般用$v_\pi(s)$表示。这个价值一般是一个期望函数。虽然当前动作会给一个延时奖励R_{t+1}，但是光看这个延时奖励是不行的，因为当前的延时奖励高，不代表到了$t+1$，$t+2$，。。。。时刻的后续奖励也高。
  - 比如下象棋，我们可以某个动作可以吃掉对方的车，这个延时奖励是很高，但是接着后面我们输棋了。此时吃车的动作奖励值高但是价值并不高。因此我们的价值要综合考虑当前的延时奖励和后续的延时奖励。
- 价值函数$v_\pi(s)$一般可以表示为下式，不同的算法会有对应的一些价值函数变种，但思路相同：

$$v_\pi(s)=E_\pi(R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\ldots|s_t=s)$$

其中γ是第六个模型要素，即奖励衰减因子，在[0，1]之间

强化学习要素（三）

- 环境的状态转化模型，可以理解为一个概率状态机，它可以表示为一个概率模型，即在状态s下采取动作a，转到下一个状态s′的概率，表示为$P^a_{ss'}$。
- 探索率ϵ，这个比率主要用在强化学习训练迭代过程中，由于我们一般会选择使当前轮迭代价值最大的动作，但是这会导致一些较好的但我们没有执行过的动作被错过。因此我们在训练选择最优动作时，会有一定的概率ϵ不选择使当前轮迭代价值最大的动作，而选择其他的动作。

强化学习的分类

- Q-Learning
- Deep Q-Network （DQN）
- Nature DQN
- Double DQN
- …

Q-Learning

- Q-Learning算法是一种使用时序差分(Temporal-Difference, TD)求解强化学习控制问题的方法，可以表示为：
  - 给定强化学习的5个要素：状态集S， 动作集A， 即时奖励R，衰减因子γ， 探索率ϵ， 求解最优的动作价值函数q^*和最优策略π^*
  - 这一类强化学习的问题求解不需要环境的状态转化模型，是不基于模型的强化学习问题求解方法。
  - 对于这类控制问题求解方法为**价值迭代**，即通过价值函数的更新，来更新策略，通过策略来产生新的状态和即时奖励，进而更新价值函数。一直进行下去，直到价值函数和策略都收敛。

冰湖（FrozenLake）问题

- 冰湖环境是一个4*4的网格。格子包含三种类型：起始格，目标格，安全的冰格和危险的冰洞。
- 目标是让一个agent能够自己从起点到终点，并且不会在中途掉入洞中。
- 在任何时候，agent都可以选择从上，下，左，右四个中选择一个方向进行一步移动。而后会使问题变复杂的是，在冰湖环境中，还有风会偶尔把agent吹得偏离到一个并非它移动目标的格子上。因此，在这种环境下agent就不太可能每次都能表现完美，但是学习如何去避免掉入洞中并达到目标格还是可以做到的。agent每走一步的回报（reward）都是0，除了在达到目标时为1。

Q值表

- Q-Learning会以一个Q值表（Q table，lookup table）的形式呈现。这个表的行代表不同的状态（所在方格的坐标），列代表不同的可采取的行动（上、下、左、右移动一步）。在冰湖环境中，我们有16个状态（每个方格算一个），以及四种动作，因此我们将得到一个16*4的Q值表，这个表的每一个具体值的含义是：该状态下，采取该行为的长期期望回报。
- 我们从初始化一个一致的Q值表（全部赋值为0）开始，再根据我们观察得到的对各种行动的回报来更新Q值表的对应值。

Q-Table		Actions				
		Action 1	Action 2	...	Action N-1	Action N
States	State 1	0.236512	0.165112	...	0.542213	0.45458
	State 2	0.681424	0.445845	...	0.212424	0.554712
	...	...	...	...	...	...
	State M-1	0.788554	0.155514	...	0.554588	0.224245
	State M	0.788954	0.554845	...	0.225255	0.742657

Q值表的更新

- 对Q值表的更新准则是贝尔曼方程（Bellman Equation）：

$$Q^*(s,\ a) = E[R_{t+1} + \gamma \max_{a'} Q^*(s',\ a') | s_t = s,\ a_t = a]$$

- 贝尔曼方程告诉我们：一个给定行动的期望长期回报等于【当前回报】加上【下一个状态下，采取期望中所能带来最优未来长期回报的行为对应的回报】。因此，我们可以在Q表上估计如何对未来行动的Q值进行更新。

The rationality behind Q-learning is that $Q^*(s,\ a)$ can be iteratively estimated based on the sequence of experience samples from actual or simulated interaction with the environment。

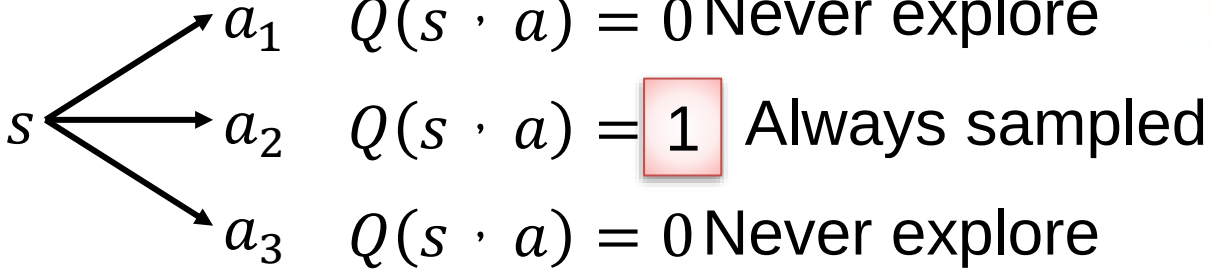
迭代更新

- Let $q(s_t, a_t)$ be the estimated action-value function during the iterative process。 Q-learning update:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_{a} q(s', a') - q(s_t, a_t)]$$

where α∈[0，1] is the learning rate

探索



- The policy is based on Q-function

$$a = arg\max_a Q(s \cdot a)$$

This is not a good way for data collection。

Epsilon Greedy

ε would decay during learning

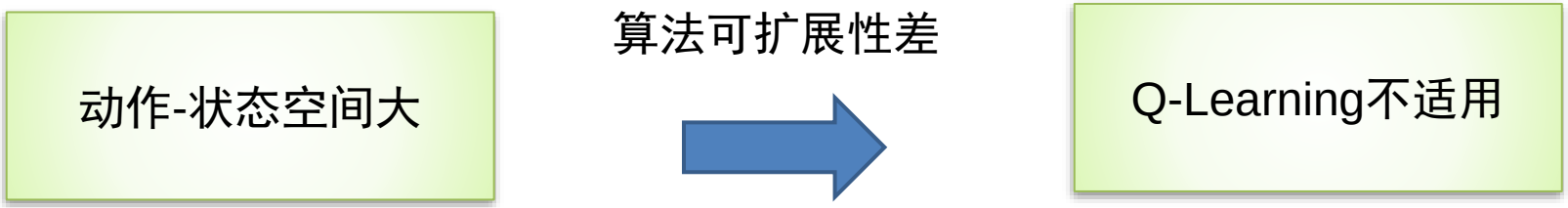
$$a = \begin{cases} arg\max_a Q(s \cdot a) & with\ probability\ 1-\varepsilon \\ random & otherwise \end{cases}$$

Boltzmann Exploration

$$P(a|s) = \frac{exp\big(Q(s \cdot a)\big)}{\sum_a exp\big(Q(s \cdot a)\big)}$$

Q-Learning缺陷

- 强化学习求解方法，使用的状态都是离散的有限个状态集合S。此时问题的规模比较小，比较容易求解。
- 但是假如我们遇到复杂的状态集合呢？甚至很多时候，状态是连续的，那么就算离散化后，集合也很大，此时我们的传统方法，比如Q-Learning，根本无法在内存中维护这么大的一张Q表。



蒙特卡罗方法Monte-Carlo (MC)

- MC的任务就是从一个状态出发，一直玩游戏直到碰到终结状态，然后得到一个，经历多次实验后，我们就可以用其平均值作为值函数的估计。

具体而言：

1. 从状态s出发，一直到游戏结束，得到回报 $G_t=R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\ldots$
2. 增加计数 $N(s)= N(s) + 1$
3. 增加总的奖励 $S(s)= S(s) + G_t$
4. 用平均值去估计s的值函数 $V(s) = S(s) / N(s)$
5. 不断重复1-4，根据大数定理，$N(s) \to \infty$， $V(s) \to v_\pi(s)$

对比TD和MC

- TD可以在最终结果出来之前进行更新，这种更新又叫做online学习，而MC必须等最终状态到达才行，叫做offline学习。
  - MC是对值函数的无偏(unbiased)估计，而TD是有偏(biased)估计。
  - MC涉及到多个状态和动作以及奖励，因此方差会很大；而TD只和一个动作，奖励和状态有关，因此方差相对较小。
  - MC方法高方差零偏差；有很好的收敛性；对初始值选择不是很敏感从而很稳定；理解和使用很简单。
  - TD方法低方差有偏差；一般比MC更有效率；对初始值更加敏感从而导致算法不稳定。TD算法利用了Markov性质，因此适用于Markov环境。

价值函数的近似表示

- 当问题的动作状态空间大，一个可行的方法是建立价值函数的近似表示。
- 引入一个状态价值函数$\hat{v}$这个函数由参数w描述，并接受状态s作为输入，计算后得到状态s的价值，即我们期望：

$$\hat{v}(s, w) \approx v_\pi(s)$$

- 类似的，引入一个动作价值函数$\hat{q}$，这个函数由参数w描述，并接受状态s与动作a作为输入，计算后得到动作价值，即我们期望：

$$\hat{q}(s, a, w) \approx q_\pi(s, a)$$

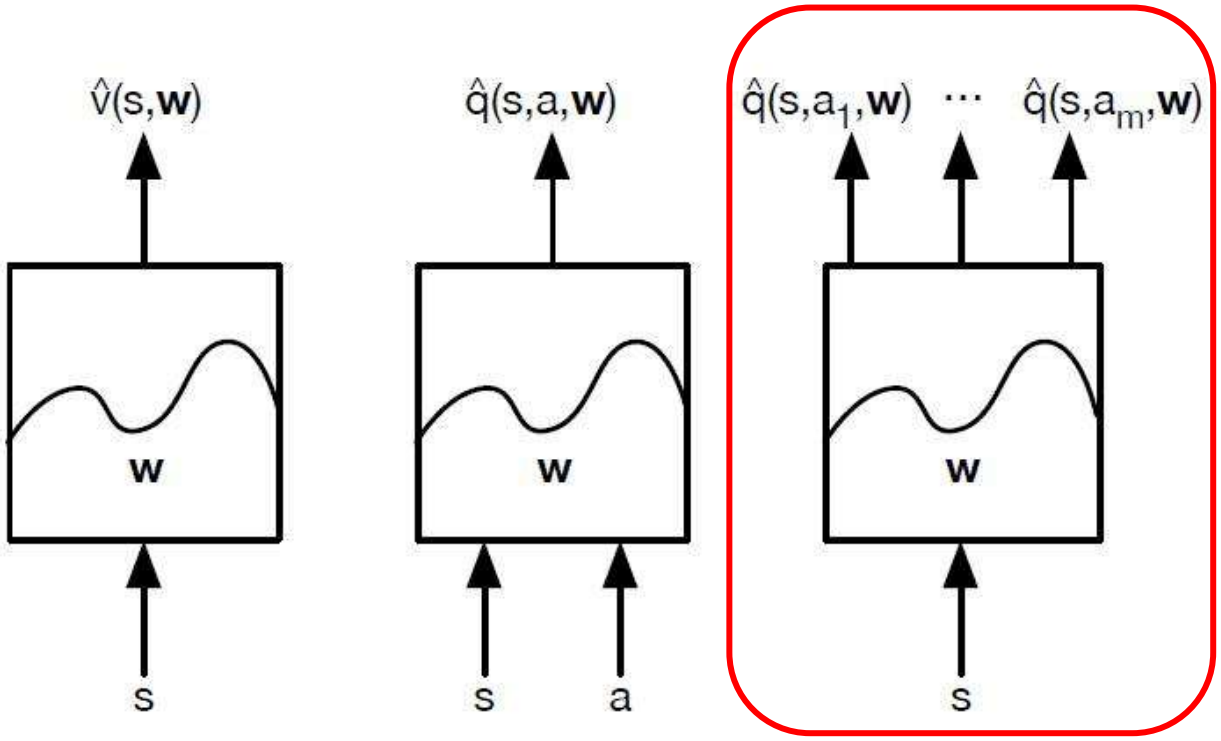
价值函数的近似表示

- 价值函数近似的方法很多，比如最简单的线性表示法，用$\phi(s)$表示状态s的特征向量，则此时我们的状态价值函数可以近似表示为：

$$v_\pi(s,\ w) = \phi(s)^T w$$

- 当然，除了线性表示法，我们还可以用决策树，最近邻，傅里叶变换，神经网络来表达我们的状态价值函数。而最常见，应用最广泛的表示方法是神经网络。

- 对于神经网络，可以使用DNN，CNN或者RNN。没有特别的限制。如果把我们计算价值函数的神经网络看做一个黑盒子，那么整个近似过程可以看做下面这三种情况：



Deep Q-Learning

输入状态s的特征向量，动作集合有多少个动作就有多少个输出$\hat{q}(s, a_i, w)$。这里隐含了我们的动作是有限个的离散动作。

Deep Q-Network (DQN)

- DQN主要使用的技巧是经验回放（experience replay），即将每次和环境交互得到的奖励与状态更新情况都保存起来，用于后面目标Q值的更新。
- 通过经验回放得到的目标Q值和通过Q网络计算的Q值肯定是有误差的，那么我们可以通过梯度的反向传播来更新神经网络的参数w，当w收敛后，我们的就得到的近似的Q值计算方法，进而贪婪策略也就求出来了。

DQN缺陷

- 目标Q值的计算方式：

$$y_j = \begin{cases} R_j & is_{end_j}\ is\ \text{true} \\ R_j + \gamma \max\limits_{a_j} Q\left(\phi\left(s_j'\right),\ a_j,\ w\right) & is_{end_j}\ is\ false \end{cases}$$

- 这里目标Q值的计算使用到了当前要训练的Q网络参数来计算$Q\left(\phi\left(s_j'\right),\ a_j,\ w\right)$，而实际上，我们又希望通过$y_j$来后续更新Q网络参数。这样两者循环依赖，迭代起来两者的相关性就太强了。不利于算法的收敛。

Nature DQN

- 改进版的DQN: Nature DQN尝试用两个Q网络来减少目标Q值计算和要更新Q网络参数之间的依赖关系。
  - Nature DQN使用了两个Q网络，一个当前Q网络Q用来更新模型参数，另一个目标Q网络Q'用于计算目标Q值。
  - 目标Q网络的网络参数不需要迭代更新，而是每隔一段时间从当前Q网络Q复制过来，即延时更新，这样可以减少目标Q值和当前的Q值相关性。
  - 两个Q网络的结构是一模一样的。这样才可以复制网络参数。
  - Nature DQN和上一篇的DQN相比，除了用一个新的相同结构的目标Q网络来计算目标Q值以外，其余部分基本是完全相同的。

算法输入：迭代轮数T，状态特征维度n， 动作集A， 步长α，衰减因子γ， 探索率ϵ， 当前Q网络Q，目标Q网络Q'， 批量梯度下降的样本数m，目标Q网络参数更新频率C。

输出：Q网络参数

1. 随机初始化所有的状态和动作对应的价值Q。 随机初始化当前Q网络的所有参数w，初始化目标Q网络Q'的参数$w'=w$。清空经验回放的集合D。

2. for i from 1 to T，进行迭代。

a) 初始化S为当前状态序列的第一个状态， 拿到其特征向量$\phi(S)$

b) 在Q网络中使用$\phi(S)$作为输入，得到Q网络的所有动作对应的Q值输出。用$\epsilon-$贪婪法在当前Q值输出中选择对应的动作A

c) 在状态S执行当前动作A，得到新状态S'对应的特征向量$\phi(S')$和奖励R，是否终止状态is_end

d) 将$\{\phi(s_j), a_j, R_j, \phi(s_j'), is_end\}$这个五元组存入经验回放集合$D$

e) $S=S'$

f) 从经验回放集合D中采样m个样本$\{\phi(s_j), a_j, R_j, \phi(s_j'), is_end\}, j=1,\dots,m$，计算当前目标Q值$yj$:

$$y_j=\begin{cases} R_j & is_{end_j}\ is\ \text{true} \\ R_j+\gamma \max\limits_{a_j'} Q'(\phi(s_j'),\ a_j',\ w') & is_{end_j}\ is\ false \end{cases}$$

g) 使用均方差损失函数$\frac{1}{m}\sum_{j=1}^{m}\left(y_j-Q(\phi(s_j),\ a_j,\ w)\right)^2$，通过神经网络的梯度反向传播来更新Q网络的所有参数w

h) 如果T%C=1，则更新目标Q网络参数$w'=w$

i) 如果S'是终止状态，当前轮迭代完毕，否则转到步骤b)

Nature DQN缺陷-过度估计

- 比如对于Nature DQN，虽然用了两个Q网络并使用目标Q网络计算Q值，其第j个样本的目标Q值的计算还是贪婪法得到的，计算如下式：

动作选择：目标网络
动作估计（计算目标值y）：目标网络

$$y_j = \begin{cases} R_j & is_{end_j}\ is\ \text{true} \\ R_j + \gamma \max\limits_{a_j'} Q'\left(\phi\left(s_j'\right),\ a_j',\ w'\right) & is_{end_j}\ is\ false \end{cases}$$

- 使用max虽然可以快速让Q值向可能的优化目标靠拢，但是很容易过犹不及，导致过度估计(Over Estimation)，所谓过度估计就是最终我们得到的算法模型有很大的偏差(bias)。
- 为了解决这个问题， DDQN通过解耦目标Q值动作的选择和目标Q值的计算这两步，来达到消除过度估计的问题。

Nature DQN缺陷-过度估计

- By calculating the target with single deep network, we face a simple problem:
- how are we sure that the best action for the next state is the action with the highest Q-value?

Nature DQN缺陷-过度估计

- At the beginning of the training we don’ t have enough information about the best action to take. Therefore, taking the maximum Q value (which is noisy) as the best action to take can lead to false positives. If non-optimal actions are regularly given a higher Q value than the optimal best action, the learning will be complicated.
- The solution is: when we compute the Q target, we use two networks to decouple the action selection from the target Q value generation.
  - We use our DQN network to select what is the best action to take for the next state (the action with the highest Q value).
  - use our target network to calculate the target Q value of taking that action at the next state.

DDQN

- DDQN和Nature DQN一样，也有一样的两个Q网络结构。在Nature DQN的基础上，通过解耦目标Q值动作的选择和目标Q值的计算这两步，来消除过度估计的问题。
- Nature DQN对于非终止状态，其目标Q值的计算：

动作选择：目标网络
动作估计（计算目标值y）：目标网络

$$y_j = R_j + \gamma \max_{a'_j} Q'\left(\phi\left(s'_j\right),\ a'_j,\ w'\right)$$

- DDQN对于非终止状态，其目标Q值的计算：

动作选择：当前网络
动作估计（计算目标值y）：目标网络

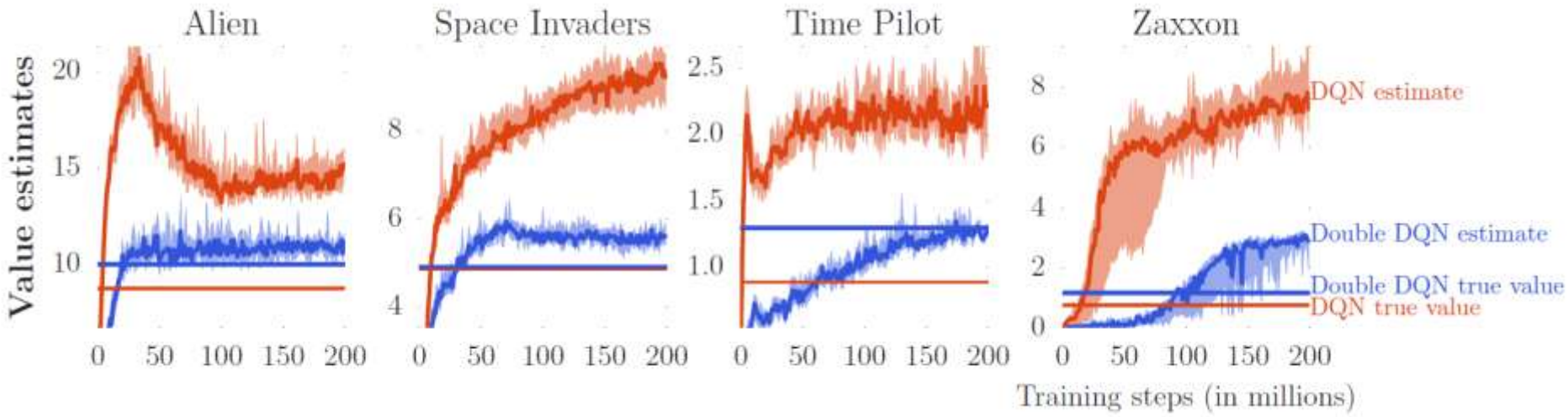
$$y_j = R_j + \gamma\, Q'\left(\phi\left(s'_j\right),\ \arg\max_{a'_j} Q\left(\phi\left(s'_j\right),\ a'_j,\ w\right),\ w'\right)$$

If Q over-estimate *a*, so it is selected。 Q’ would give it proper value。

How about Q’ overestimate? The action will not be selected by Q。

Double DQN

- Q value is usually over-estimated



内容

- 深度强化学习基础知识介绍
- 深度强化学习在无人机飞行控制上的应用

Gym: OpenAI 开源强化学习库

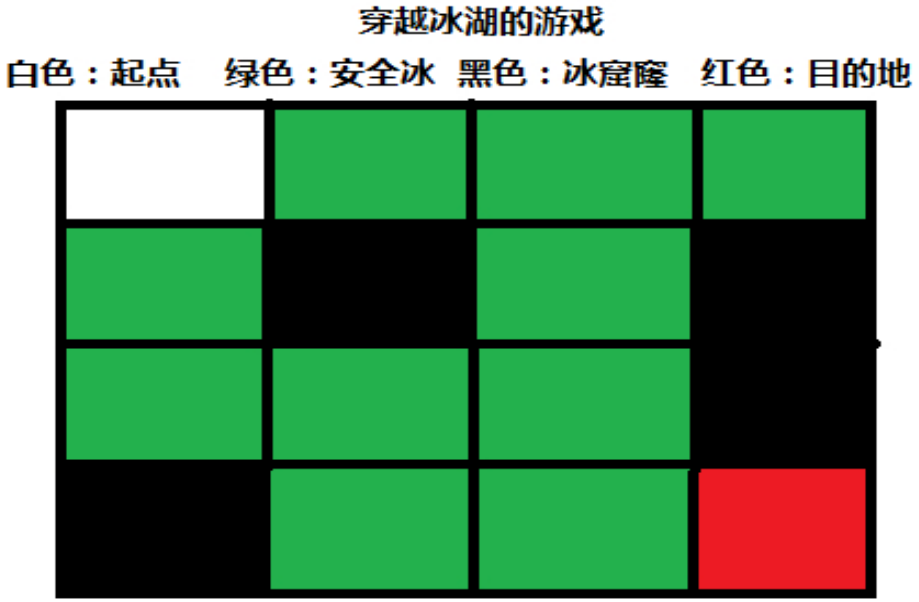
- Frozen Lake 指在一块冰面上有四种状态：

S: initial stat 起点

F: frozen lake 冰湖

H: hole 窟窿

G: the goal 目的地



Agent （无人机）要学会从起点走到目的地，并且不要掉进窟窿（障碍物）。

创建冰湖环境

```
1  # frozen-lake-ex1.py
2  import gym # loading the Gym library
3
4  env = gym.make("FrozenLake-v0")
5  env.reset()
6  env.render()
```

- gym.make() 用于创建冰湖环境
- env.reset() 将环境设置为初始状体
- env.render() 打印当前状态

打印状态空间和动作空间大小：

```
1  # frozen-lake-ex1.py
2
3  print("Action space: ", env.action_space)
4  print("Observation space: ", env.observation_space)
```

就缺省4×4环境而言，state 有0-15共16个，action有4个

随机选择动作

```
# frozen-lake-ex2.py
import gym

MAX_ITERATIONS = 10

env = gym.make("FrozenLake-v0")
env.reset()
env.render()
for i in range(MAX_ITERATIONS):
    random_action = env.action_space.sample()
    new_state, reward, done, info = env.step(
       random_action)
    env.render()
    if done:
        break
```

- 在不采用任何算法时，env可以通过sample() 方法在每个state上随机选择 action；
- 将选择的 action 输入给 env.step()，可以得到new_state, reward, done, info 这四个返回值；
- 使用 env.render() 能将当前新的环境布局展示出来。

模式选择：Stochastic vs Deterministic

- **Stochastic 模式：**

真实世界中，agent采用/执行一个动作后，环境的状态转移是一个随机过程，即状态转换的最终结果不仅仅取决于动作，还受到环境的影响（例如，风、冰面打滑等）。

- **Deterministic模式**

```
1  # frozen-lake-ex4.py
2  env = gym.make("FrozenLake-v0", is_slippery=False)
```

通过关闭slippery surface，可以使环境总是执行agent所采取的动作。

定制地图

- 缺省大小：4×4，修改为8×8

```python
# frozen-lake-ex5.py
env = gym.make("FrozenLake8x8-v0")
env.reset()
env.render()
```

```python
# frozen-lake-ex5.py
env = gym.make('FrozenLake-v0', map_name='8x8')
env.reset()
env.render()
```

- 定制地图

```python
custom_map = [
    'SFFHF',
    'HFHFF',
    'HFFFH',
    'HHHFH',
    'HFFFG'
]

env = gym.make('FrozenLake-v0', desc=custom_map)
env.reset()
env.render()
```

参考资料

- 强化学习：

https://reinforcement-learning4.fun/2019/06/03/how-ai-learns-play-games/

- 关于冰湖游戏以及强化学习

https://reinforcement-learning4.fun/2019/06/09/introduction-reinforcement-learning-frozen-lake-example/

- 如何安装Gym：

配套实验手册、https://reinforcement-learning4.fun/2019/05/24/how-to-install-openai-gym/

- 示例代码：

https://github.com/rodmsmendes/reinforcementlearning4fun/tree/master/gym-tutorial-frozen-lake

内容

- 强化学习基础知识介绍
- 冰湖问题介绍
- 实验内容介绍

大作业内容

- 熟悉使用Gym库；
- 熟悉强化学习代码；
- 在Stochastic 和 Deterministic两种模式下完成仿真模型训练，生成无人机飞行路径；
- 在Stochastic 和 Deterministic模式下，用python控制无人机飞越冰湖，可视化结果，并完成验收；

开放式课题，可自我提升难度，如给每个冰格加上路径权重，训练寻找最短安全飞行路径；尝试使用Q-Learning以外的强化学习模型等

文档规范、课题难度及完成度都是评分时的加分项

实验验收

考核方式

- 每组提交一份实验报告
- 报告内容包含算法介绍，实验结果与分析以及实验过程中遇到的问题与解决办法，附实验代码
- 期末考试周之前提交至邮箱：glab_network@yeah.net
- 邮件名：智能机器人技术-全体组员姓名

注意事项

- 实验报告中需要介绍如何运行代码，方便老师和助教对实验结果进行核对
- 无人机飞行过程中可能存在些许误差，可以通过降低速度，消除风力干扰，代码优化等手段尽量减少误差

评分标准

- 完成仿真模型训练，代码运行成功，20 分；
- 参照示例视频，完成实机飞行演示，20 分；
- 在完成实机飞行演示的过程中，电脑端能够实时显示飞机当前所在位置， 20 分；
- 对实验结果有深入分析，能指出强化学习算法的优缺点或者可改进之处，20 分
- 实验报告格式整洁、思路清晰，20 分；

谢谢!