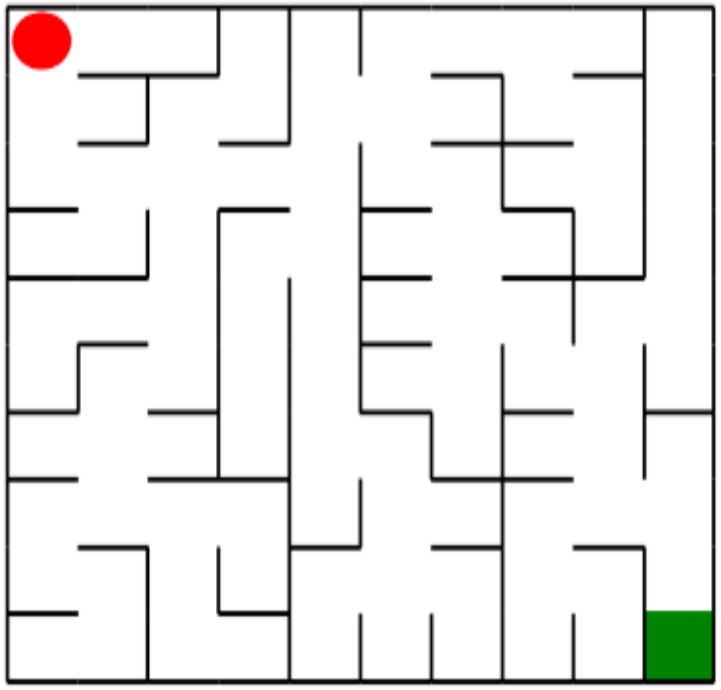
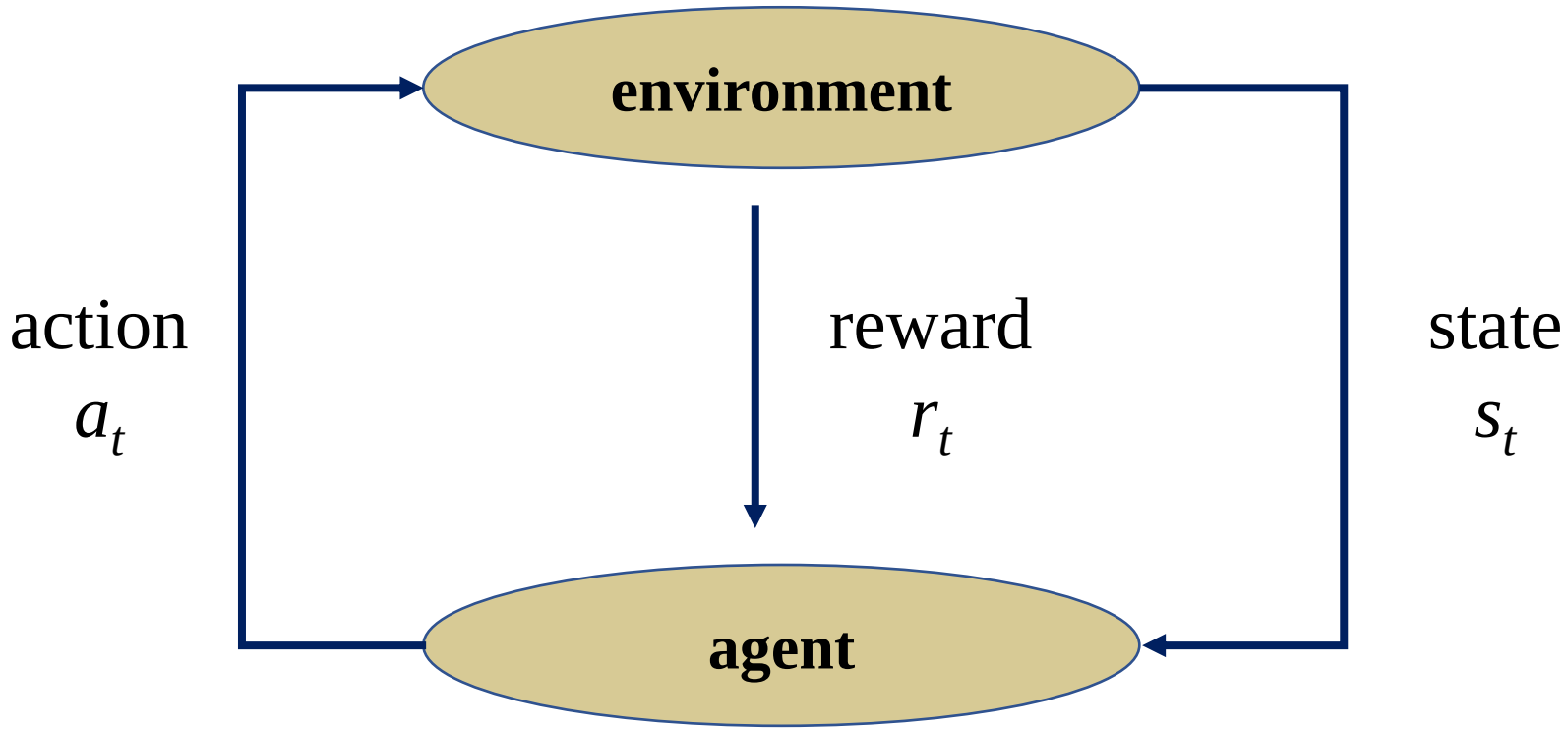
强化学习

人工智能基础 —— 实践课（六）



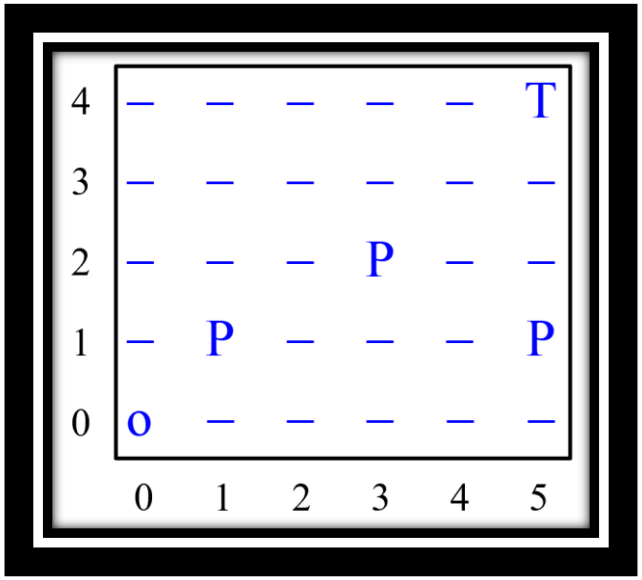
- 在Q-learning中，我们通过维护一张Q值表，通过贝尔曼(Bellman)方程对其进行不停的迭代尝试直至收敛，然后根据Q值表获取Agent在每个状态下的最优策略。但Q值表在状态和动作空间都是有限且低维的时候适用，当状态-动作空间高维且连续时，维护一张无限庞大的Q值表是不现实的。因此。DQN提出将Q-Table的更新问题变成一个函数拟合问题，相近的状态将得到相近的动作输出，即使用神经网络对动作-状态的Q值进行建模估计。
- 了解基础Q-Learning算法。
- 掌握贝尔曼(Bellman)方程意义及运算。
- 了解如何使用pytorch/keras实现DQN任务。

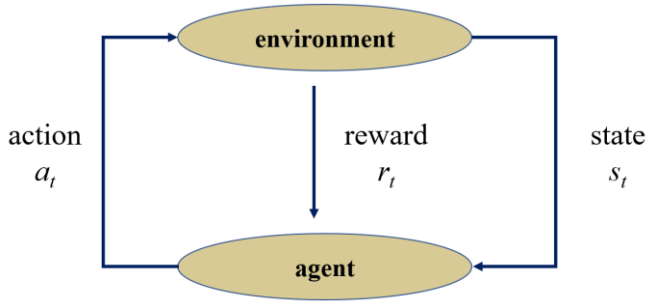
强化学习



environment

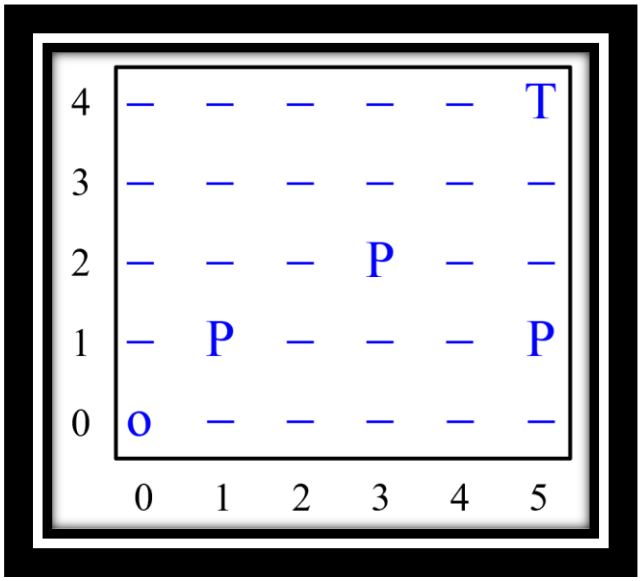
使用强化学习算法设计一个可以在一定大小的迷宫中找到宝藏的agent(智能体)。迷宫可以使用一系列字符表示，字符"o"表示agent的位置，字符"T"表示宝藏的位置，该位置的收益为1，且为游戏的终结状态，字符"P"表示陷阱的位置，该位置的收益为-1，且为游戏的终结状态，字符"_"表示收益为0的中间状态。初始情况下，迷宫如下图表示：

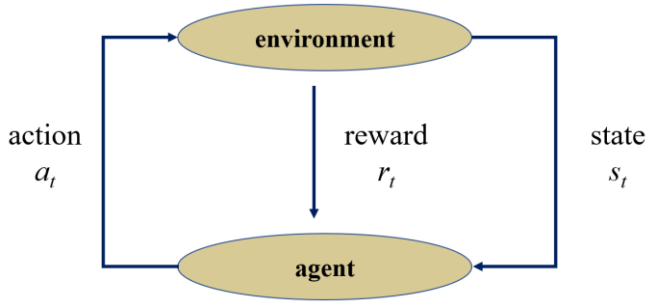




每次行动agent可以选择向不超过迷宫边界的相邻位置移动一格，通过Q-Learning算法希望agent最终找到一条通往宝藏的路径。

Q-learning





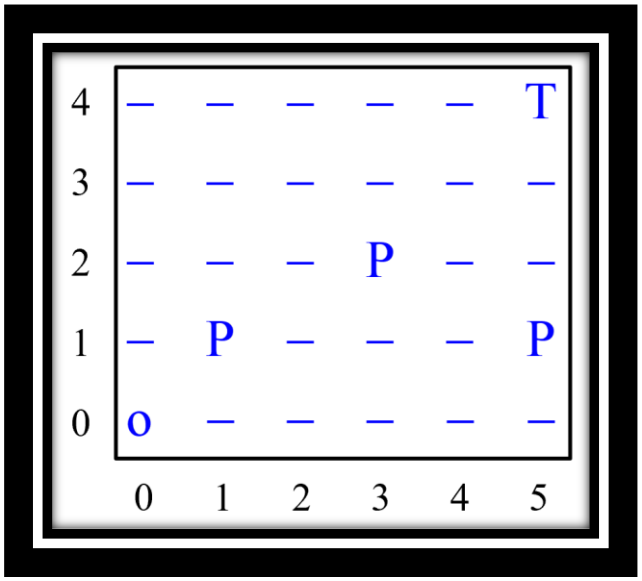
States ：智能体在环境中可能处于的所有状态
Actions: 智能体在环境中所能采取的所有行动

States : {(0, 0), (0, 1), …, (4, 4) , (4, 5)}
Actions: {(+1, 0), (-1, 0), (0, -1), (0, +1)}

Rewards: 智能体所能够获取的奖励

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Q-table



States : {(0, 0), (0, 1), …, (4, 4) , (4, 5)}
Actions: {(+1, 0), (-1, 0), (0, -1), (0, +1)}

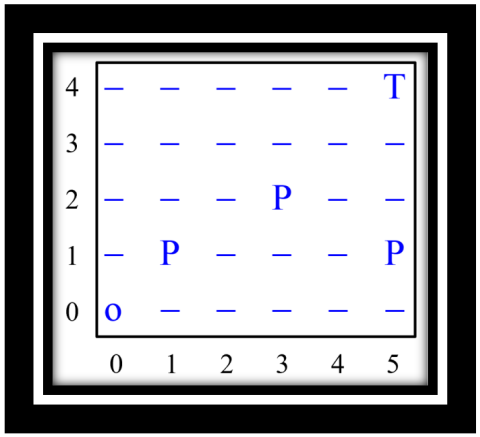
Rewards:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Q-table

	up	down	left	right
0_0	0	0	0	0
0_1	0	0	0	0
0_2	0	0	0	0
0_3	0	0	0	0
0_4	0	0	0	0
0_5	0	0	0	0
…	…	…	…	…
4_0	0	0	0	0
4_1	0	0	0	0
4_2	0	0	0	0
4_3	0	0	0	0
4_4	0	0	0	0
4_5	0	0	0	0

Q-table



	up	down	left	right
0_0	0	0	0	0
0_1	0	0	0	0
0_2	0	0	0	0
0_3	0	0	0	0
0_4	0	0	0	0
0_5	0	0	0	0
…	…	…	…	…
4_0	0	0	0	0
4_1	0	0	0	0
4_2	0	0	0	0
4_3	0	0	0	0
4_4	0	0	0	0
4_5	0	0	0	0

ε	**epsilon**	**0.2**	**贪婪度 greedy**：此实验算法中以epsilon概率进行随机决策，以(1-epsilon)概率进行贪婪决策
α	**alpha**	**0.2**	**学习率**：在强化学习中，学习率α越大,表示采用新的尝试得到的结果比例越大,保持旧的结果的比例越小
γ	**gamma**	**0.8**	**奖励递减值(折现率)**：强化学习中，期望奖励会以奖励乘以奖励递减值的形式体现

1. 探索与利用

以0.2的概率随机选择策略

以(1-0.2)的概率执行贪婪策略

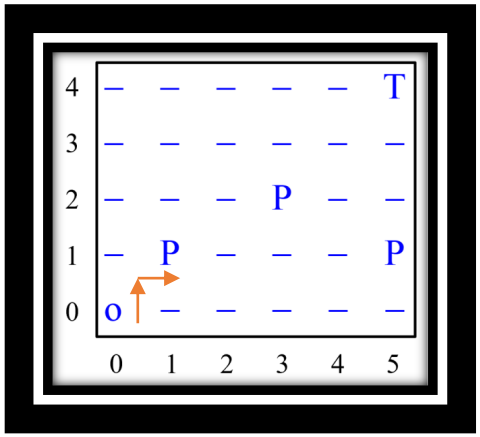
Q-table中对应状态贪婪策略有多个备选action时，从备选action中随机选择action；否则选取最大Q-value的action。

2. 更新Q-table

$$q(s,a) \leftarrow (1-\alpha)q(s,a) + \alpha\left[R + \gamma \max_{a'} q(s',a')\right]$$

$$q(s,a) \leftarrow q(s,a) + \alpha\left[R + \gamma \max_{a'} q(s',a') - q(s,a)\right]$$

Q-table



	up	down	left	right
0_0	0	0	0	0
0_1	-0.2	0	0	0
0_2	0	0	0	0
0_3	0	0	0	0
0_4	0	0	0	0
0_5	0	0	0	0
…	…	…	…	…
4_0	0	0	0	0
4_1	0	0	0	0
4_2	0	0	0	0
4_3	0	0	0	0
4_4	0	0	0	0
4_5	0	0	0	0

ε	**epsilon**	**0.2**	**贪婪度 greedy**：此实验算法中以epsilon概率进行随机决策，以(1-epsilon)概率进行贪婪决策
α	**alpha**	**0.2**	**学习率**：在强化学习中，学习率α越大,表示采用新的尝试得到的结果比例越大,保持旧的结果的比例越小
γ	**gamma**	**0.8**	**奖励递减值(折现率)**：强化学习中，期望奖励会以奖励乘以奖励递减值的形式体现

1. 探索与利用

以0.2的概率随机选择策略

以(1-0.2)的概率执行贪婪策略

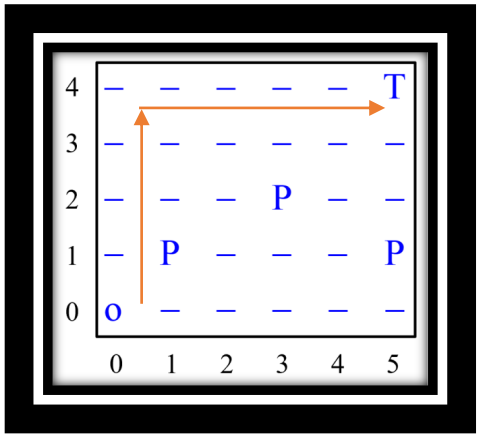
Q-table中对应状态贪婪策略有多个备选action时，从备选action中随机选择action；否则选取最大Q-value的action。

2. 更新Q-table

$$q(s,a) \leftarrow (1-\alpha)q(s,a) + \alpha\left[R + \gamma \max_{a'} q(s',a')\right]$$

$$q(s,a) \leftarrow q(s,a) + \alpha\left[R + \gamma \max_{a'} q(s',a') - q(s,a)\right]$$

Q-table



	up	down	left	right
0_0	0	0	0	0
0_1	-0.2	0	0	0
0_2	0	0	0	0
0_3	0	0	0	0
0_4	0	0	0	0
0_5	0	0	0	0
…	…	…	…	…
4_0	0	0	0	0
4_1	0	0	0	0
4_2	0	0	0	0
4_3	0	0	0	0
4_4	0	0	0	0.2
4_5	0	0	0	0

ε	**epsilon**	**0.2**	**贪婪度 greedy**：此实验算法中以epsilon概率进行随机决策，以(1-epsilon)概率进行贪婪决策
α	**alpha**	**0.2**	**学习率**：在强化学习中，学习率α越大,表示采用新的尝试得到的结果比例越大,保持旧的结果的比例越小
γ	**gamma**	**0.8**	**奖励递减值(折现率)**：强化学习中，期望奖励会以奖励乘以奖励递减值的形式体现

1. 探索与利用

以0.2的概率随机选择策略

以(1-0.2)的概率执行贪婪策略

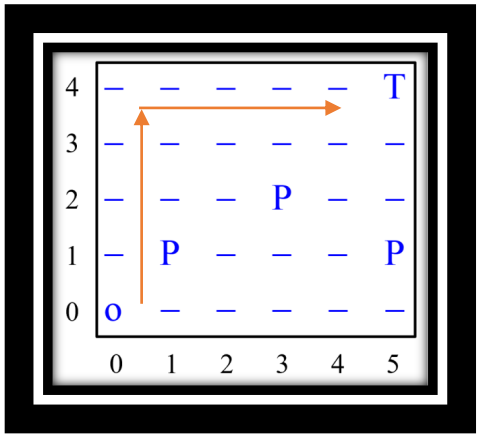
Q-table中对应状态贪婪策略有多个备选action时，从备选action中随机选择action；否则选取最大Q-value的action。

2. 更新Q-table

$$q(s,a) \leftarrow (1-\alpha)q(s,a) + \alpha\left[R + \gamma \max_{a'} q(s',a')\right]$$

$$q(s,a) \leftarrow q(s,a) + \alpha\left[R + \gamma \max_{a'} q(s',a') - q(s,a)\right]$$

Q-table



	up	down	left	right
0_0	0	0	0	0
0_1	-0.2	0	0	0
0_2	0	0	0	0
0_3	0	0	0	0
0_4	0	0	0	0
0_5	0	0	0	0
…	…	…	…	…
4_0	0	0	0	0
4_1	0	0	0	0
4_2	0	0	0	0
4_3	0	0	0	0.032
4_4	0	0	0	0.2
4_5	0	0	0	0

ε	**epsilon**	**0.2**	**贪婪度 greedy**：此实验算法中以epsilon概率进行随机决策，以(1-epsilon)概率进行贪婪决策
α	**alpha**	**0.2**	**学习率**：在强化学习中，学习率α越大,表示采用新的尝试得到的结果比例越大,保持旧的结果的比例越小
γ	**gamma**	**0.8**	**奖励递减值(折现率)**：强化学习中，期望奖励会以奖励乘以奖励递减值的形式体现

1. 探索与利用

以0.2的概率随机选择策略

以(1-0.2)的概率执行贪婪策略

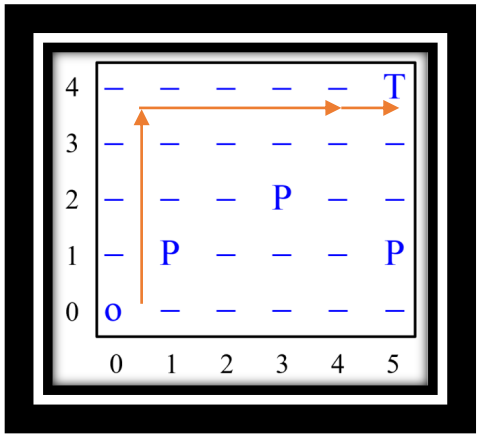
Q-table中对应状态贪婪策略有多个备选action时，从备选action中随机选择action；否则选取最大Q-value的action。

2. 更新Q-table

$$q(s,a) \leftarrow (1-\alpha)q(s,a) + \alpha\left[R + \gamma \max_{a'} q(s',a')\right]$$

$$q(s,a) \leftarrow q(s,a) + \alpha\left[R + \gamma \max_{a'} q(s',a') - q(s,a)\right]$$

Q-table



	up	down	left	right
0_0	0	0	0	0
0_1	-0.2	0	0	0
0_2	0	0	0	0
0_3	0	0	0	0
0_4	0	0	0	0
0_5	0	0	0	0
…	…	…	…	…
4_0	0	0	0	0
4_1	0	0	0	0
4_2	0	0	0	0
4_3	0	0	0	0.032
4_4	0	0	0	0.36
4_5	0	0	0	0

ε	**epsilon**	**0.2**	**贪婪度 greedy**：此实验算法中以epsilon概率进行随机决策，以(1-epsilon)概率进行贪婪决策
α	**alpha**	**0.2**	**学习率**：在强化学习中，学习率α越大,表示采用新的尝试得到的结果比例越大,保持旧的结果的比例越小
γ	**gamma**	**0.8**	**奖励递减值(折现率)**：强化学习中，期望奖励会以奖励乘以奖励递减值的形式体现

1. 探索与利用

以0.2的概率随机选择策略

以(1-0.2)的概率执行贪婪策略

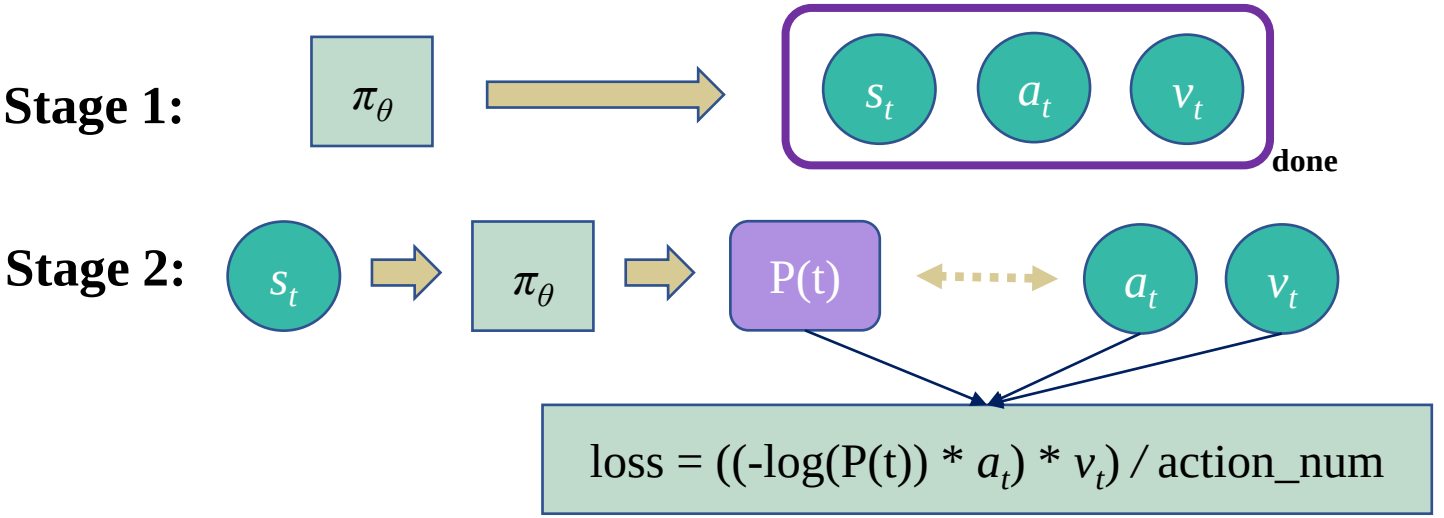
Q-table中对应状态贪婪策略有多个备选action时，从备选action中随机选择action；否则选取最大Q-value的action。

2. 更新Q-table

$$q(s,a) \leftarrow (1-\alpha)q(s,a) + \alpha\left[R + \gamma \max_{a'} q(s',a')\right]$$

$$q(s,a) \leftarrow q(s,a) + \alpha\left[R + \gamma \max_{a'} q(s',a') - q(s,a)\right]$$

Policy-Gradients



function REINFORCE

Initialise θ arbitrarily

for each episode $\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

for $t = 1$ to $T - 1$ **do**

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

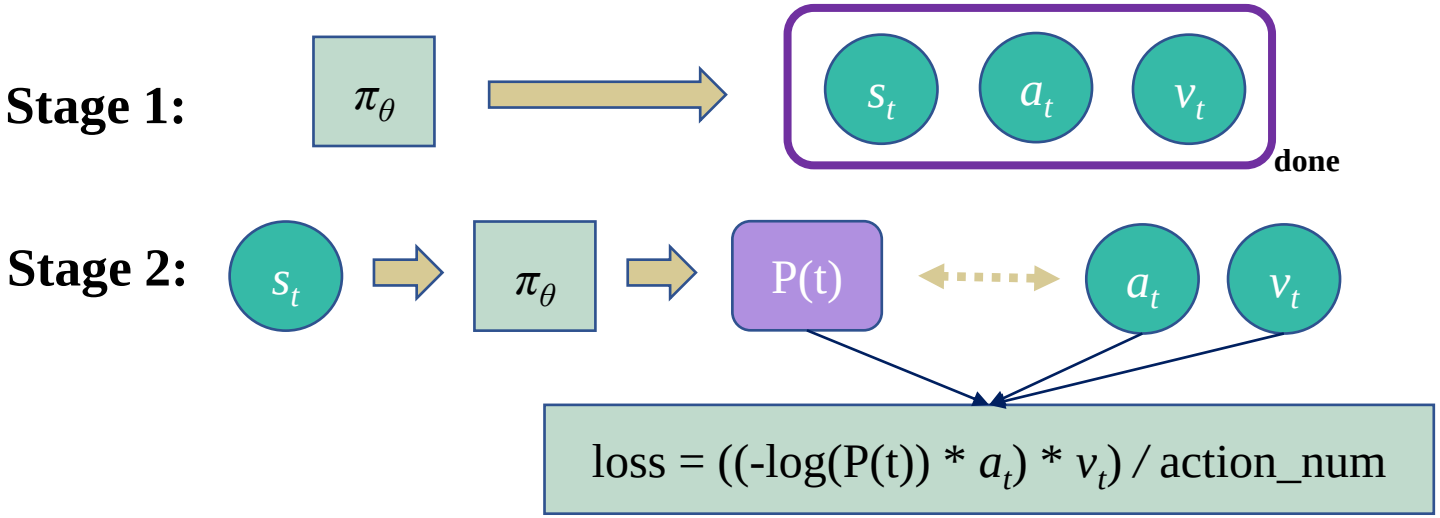
end for

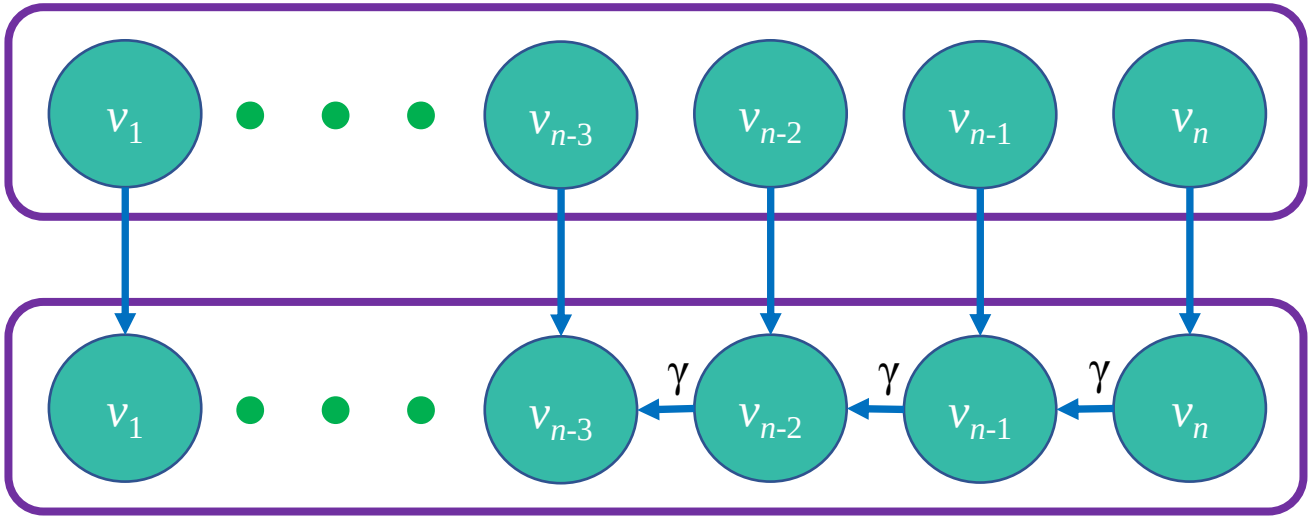
end for

return θ

end function

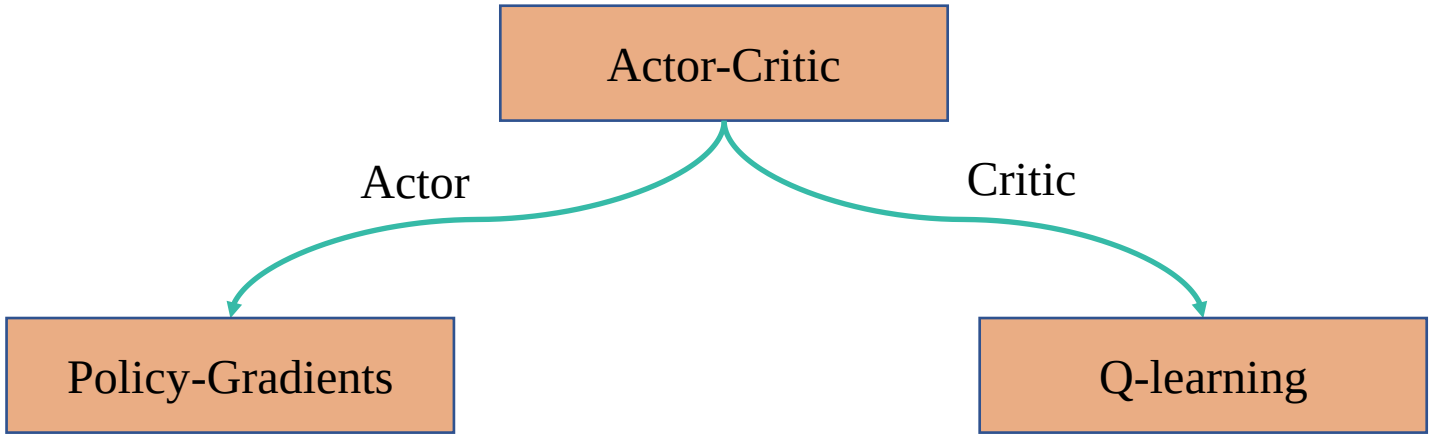
Policy-Gradients





$$v_t = \frac{v_t - mean(v)}{std(v)}$$

Actor-Critic



在连续动作中选择合适的动作

以动作概率为基础

只进行回合更新

能进行单步更新

以值为基础

学习选择动作的概率

学习 reward 奖励函数

function REINFORCE

Initialise θ arbitrarily

for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

for $t = 1$ to $T - 1$ **do**

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

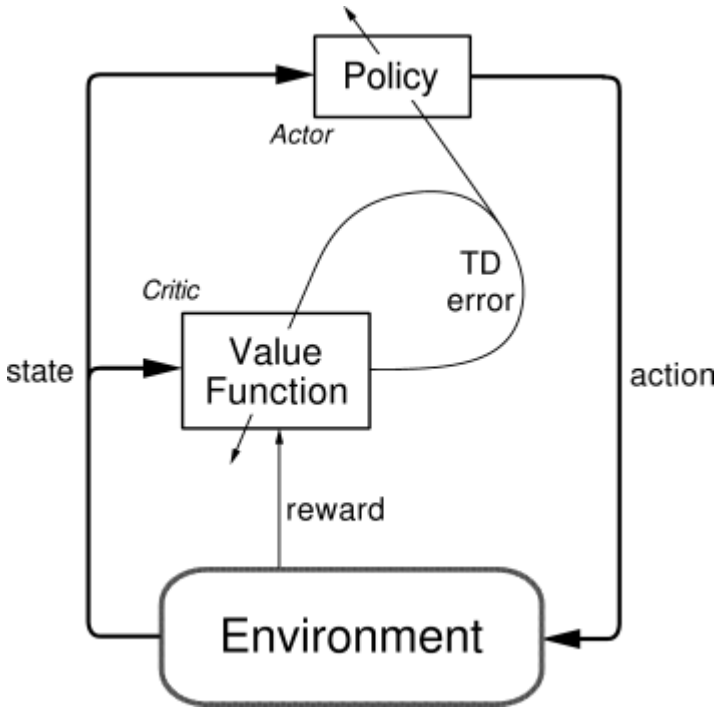
end for

end for

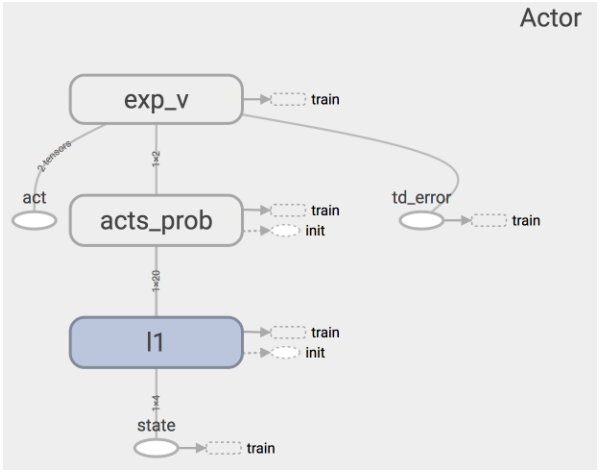
return θ

end function

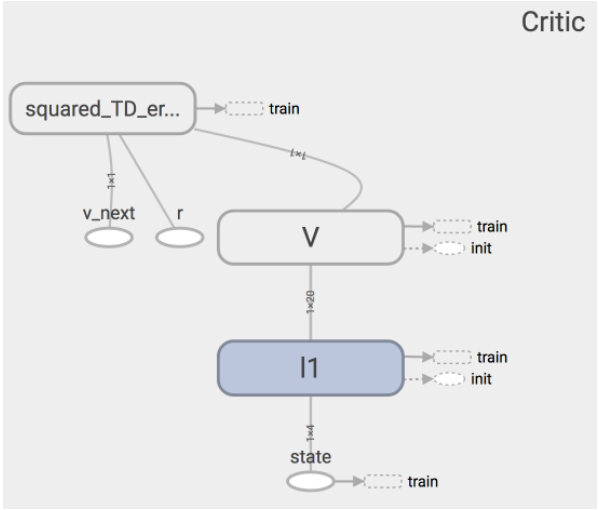
Actor-Critic



Actor: 预测行为的概率



Critic: 预测状态的价值



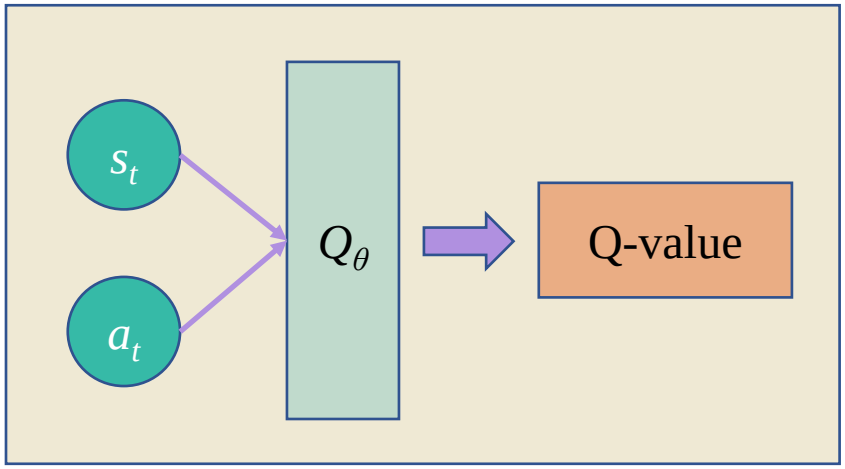
DQN

法宝1 用神经网络计算Q值:

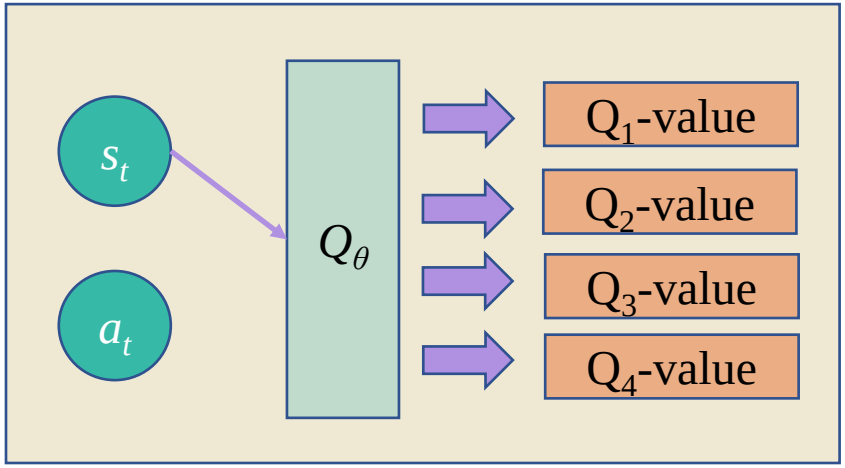
Q-table

	up	down	left	right
0_0	0	0	0	0
0_1	0	0	0	0
0_2	0	0	0	0
0_3	0	0	0	0
0_4	0	0	0	0
0_5	0	0	0	0
…	…	…	…	…
4_0	0	0	0	0
4_1	0	0	0	0
4_2	0	0	0	0
4_3	0	0	0	0
4_4	0	0	0	0
4_5	0	0	0	0

Q-function

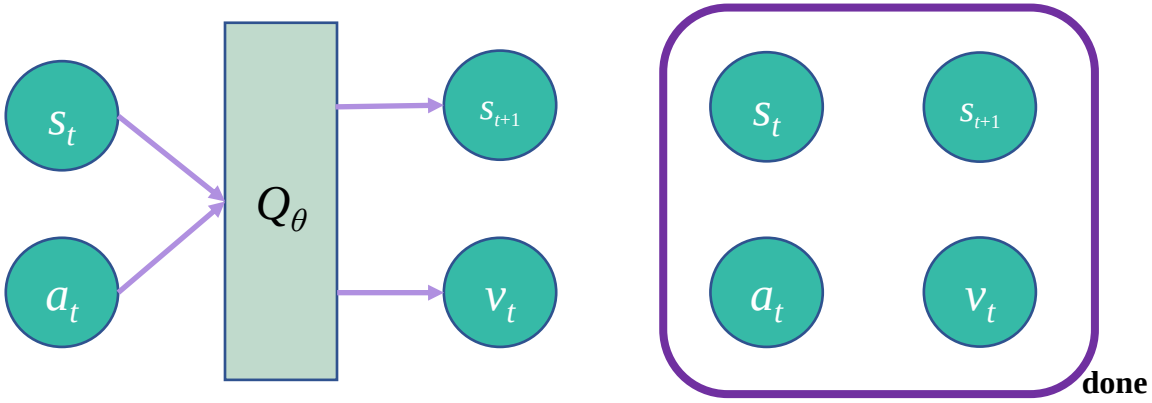


or

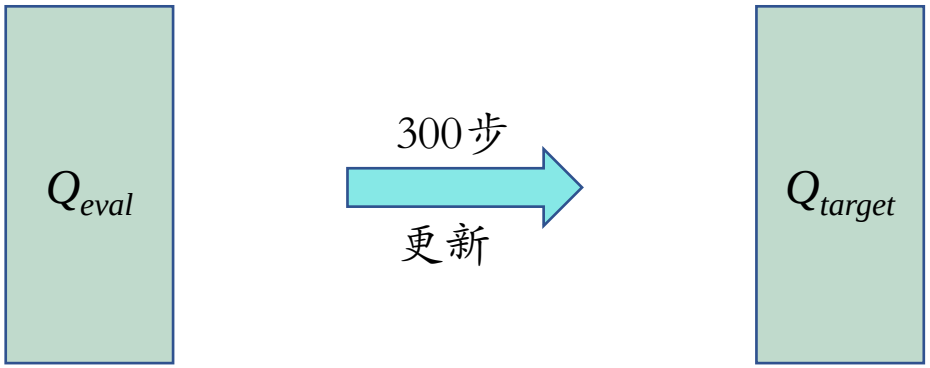


DQN

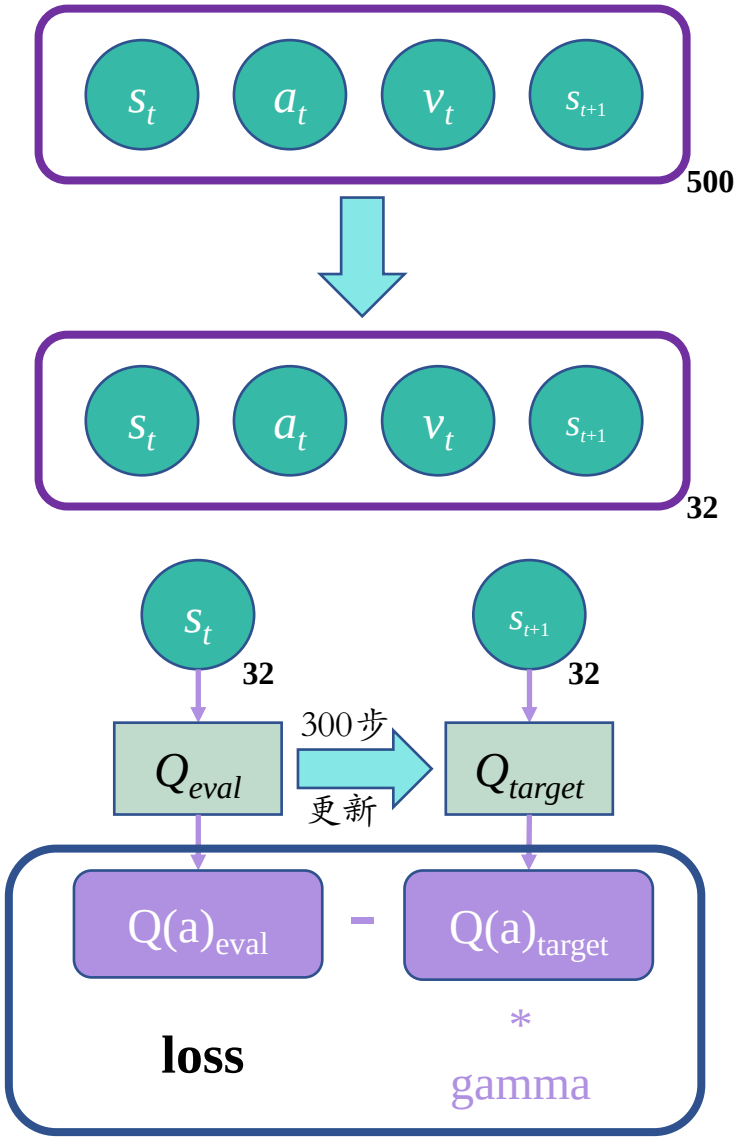
法宝2 记忆库(用于重复学习):



法宝3 冻结Q-target参数(切断相关性):



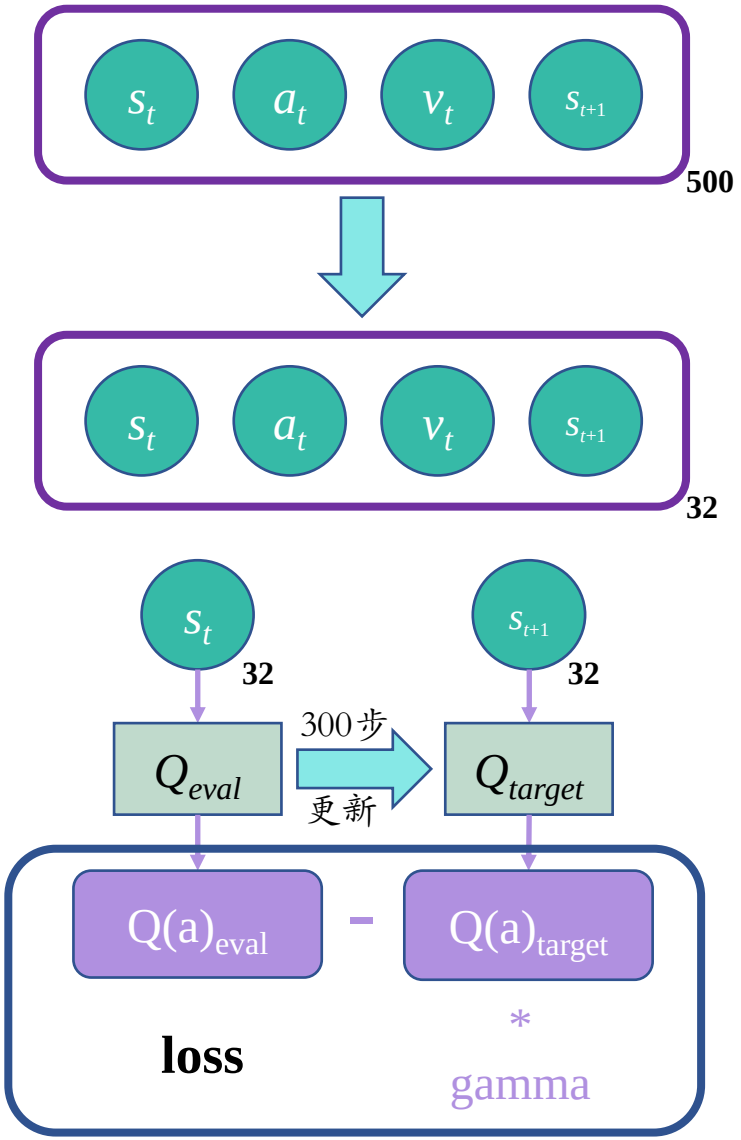
DQN



ε	**epsilon**	**0.2**	**贪婪度 greedy**：此实验算法中以epsilon概率进行随机决策，以(1-epsilon)概率进行贪婪决策
α	**alpha**	**0.2**	**学习率**：在强化学习中，学习率α越大,表示采用新的尝试得到的结果比例越大,保持旧的结果的比例越小
γ	**gamma**	**0.8**	**奖励递减值(折现率)**：强化学习中，期望奖励会以奖励乘以奖励递减值的形式体现
	target iteration	**300**	**Q-target更新率**：每300步用eval网络参数更新一次target网络
	memory size	**500**	**记忆库**：用于重复学习的记忆库
	batch size	**32**	**批大小**：一次训练的样本数目

$$Q(a)_{target} = reward + gamma * max\ Q(s\_\{t+1\})$$

DQN



Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

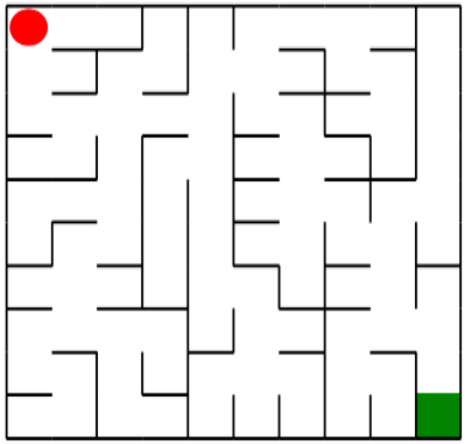
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

机器人自动走迷宫



如上图所示，左上角的**红色椭圆**既是起点也是机器人的初始位置，右下角的**绿色方块**是出口。
游戏规则为：从起点开始，通过错综复杂的迷宫，到达目标点(出口)。

- 在任一位置可执行动作包括：向上走'u'、向右走'r'、向下走'd'、向左走'l'。

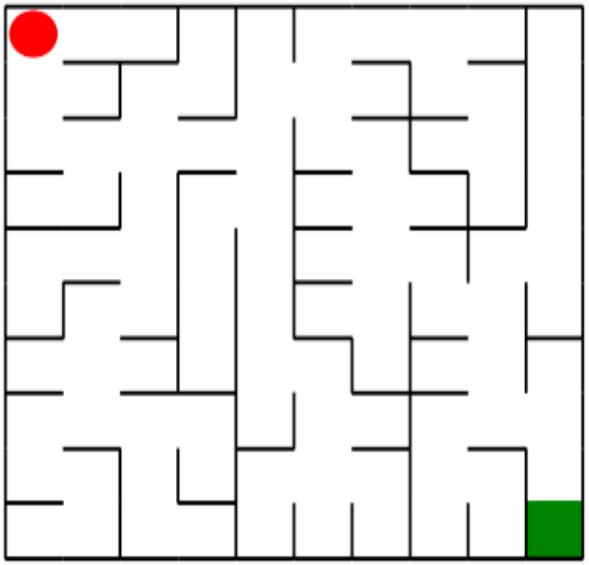
- 执行不同的动作后，根据不同的情况会获得不同的奖励，具体而言，有以下几种情况。
  - 撞墙
  - 走到出口
  - 其余情况

机器人自动走迷宫

迷宫类: class Maze(object)

- __init__(self, maze_size=5): 随机生成maze_size大小的迷宫
- sense_robot(self):获取机器人在迷宫中目前的位置
  - return: 机器人在迷宫中目前的位置
- move_robot(self, direction):根据输入方向移动默认机器人，若方向不合法则返回错误信息
  - direction：移动方向, 如:"u", 合法值为： ['u', 'r', 'd', 'l']
  - return：执行动作的奖励值
- can_move_actions(self, position):获取当前机器人可以移动的方向
  - position：迷宫中任一处的坐标点
  - return：该点可执行的动作，如：['u','r','d']
- is_hit_wall(self, location, direction):判断该移动方向是否撞墙
  - location, direction：当前位置和要移动的方向，如(0,0) , "u“
  - return：True(撞墙) / False(不撞墙)
- draw_maze(self):画出当前的迷宫

机器人自动走迷宫

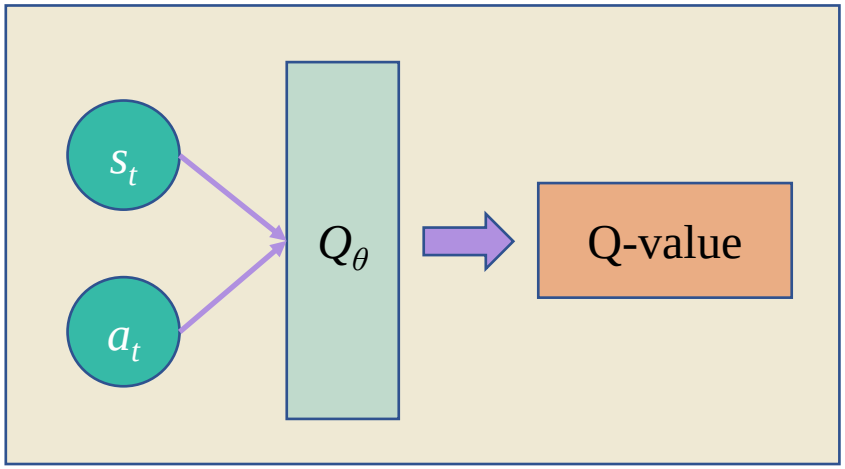


Rewards: 智能体所能够获取的奖励

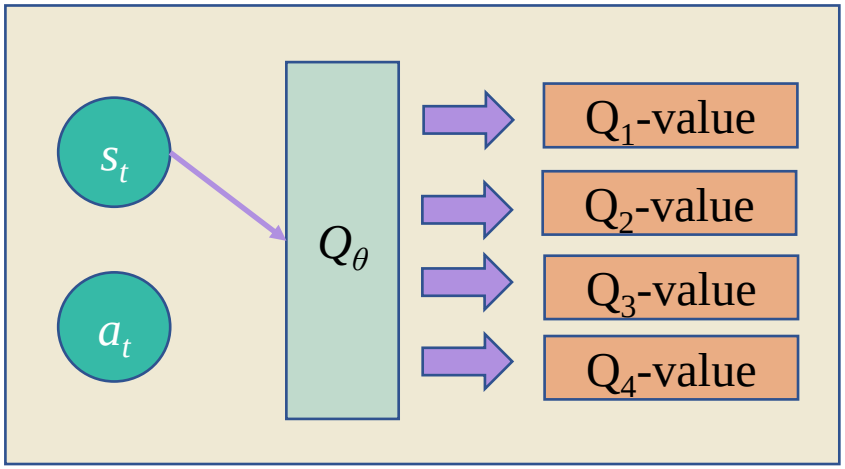
```
self.reward = {
    "hit_wall": -10.,
    "destination": 50.,
    "default": -0.1,
}
```

States : {(0, 0), (0, 1), …, (9, 8) , (9, 9)}
Actions: {(+1, 0), (-1, 0), (0, -1), (0, +1)}

Q-function

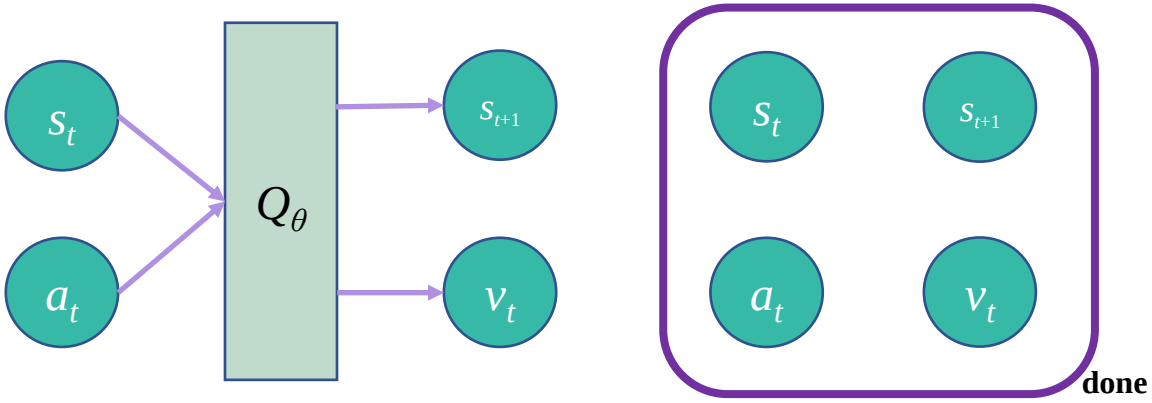


or

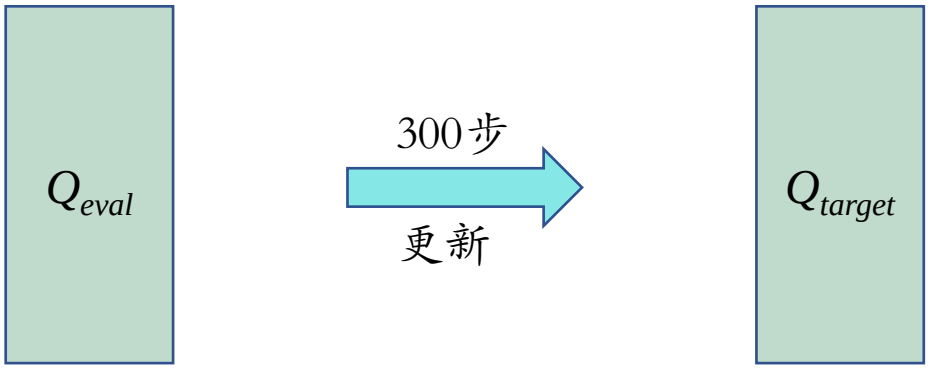


机器人自动走迷宫

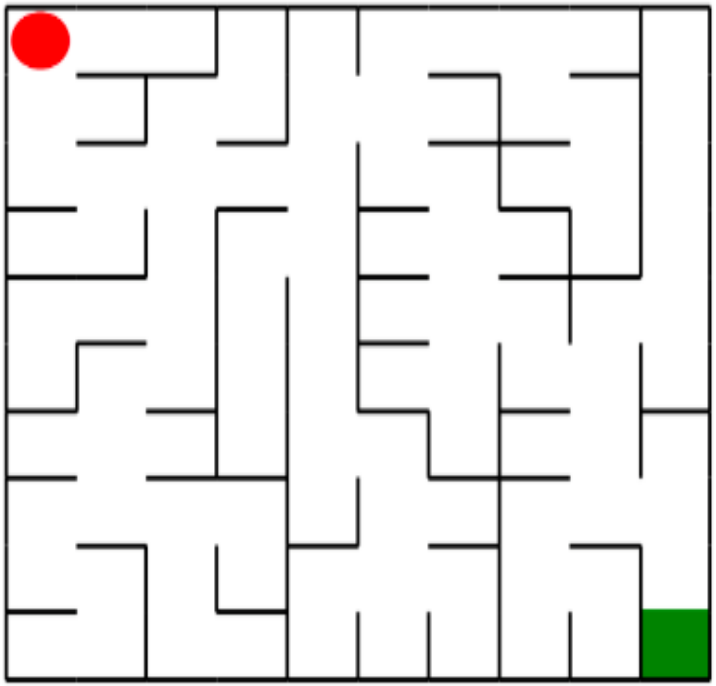
法宝2 记忆库(用于重复学习):

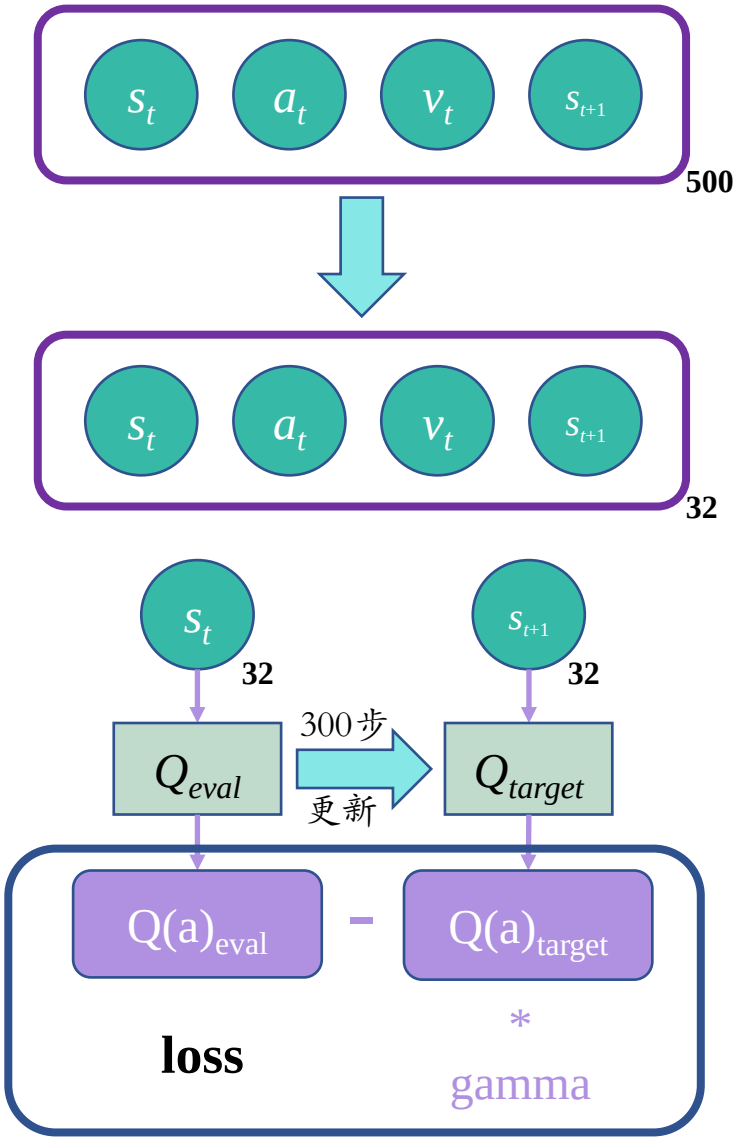


法宝3 冻结Q-target参数(切断相关性):

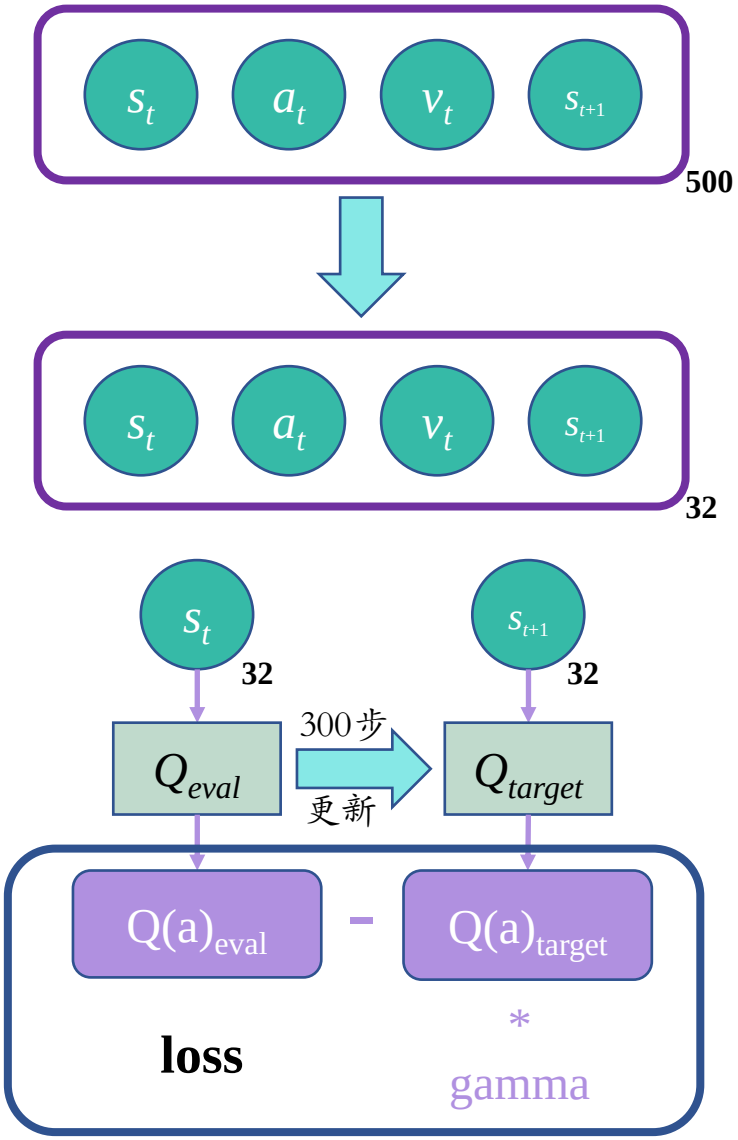


机器人自动走迷宫




s_t
a_t
v_t
s_{t+1}
500
s_t
a_t
v_t
s_{t+1}
32
s_t
32
s_{t+1}
32
300步
更新
Q_{eval}
Q_{target}
$Q(a)_{eval}$
-
$Q(a)_{target}$
loss
*
gamma

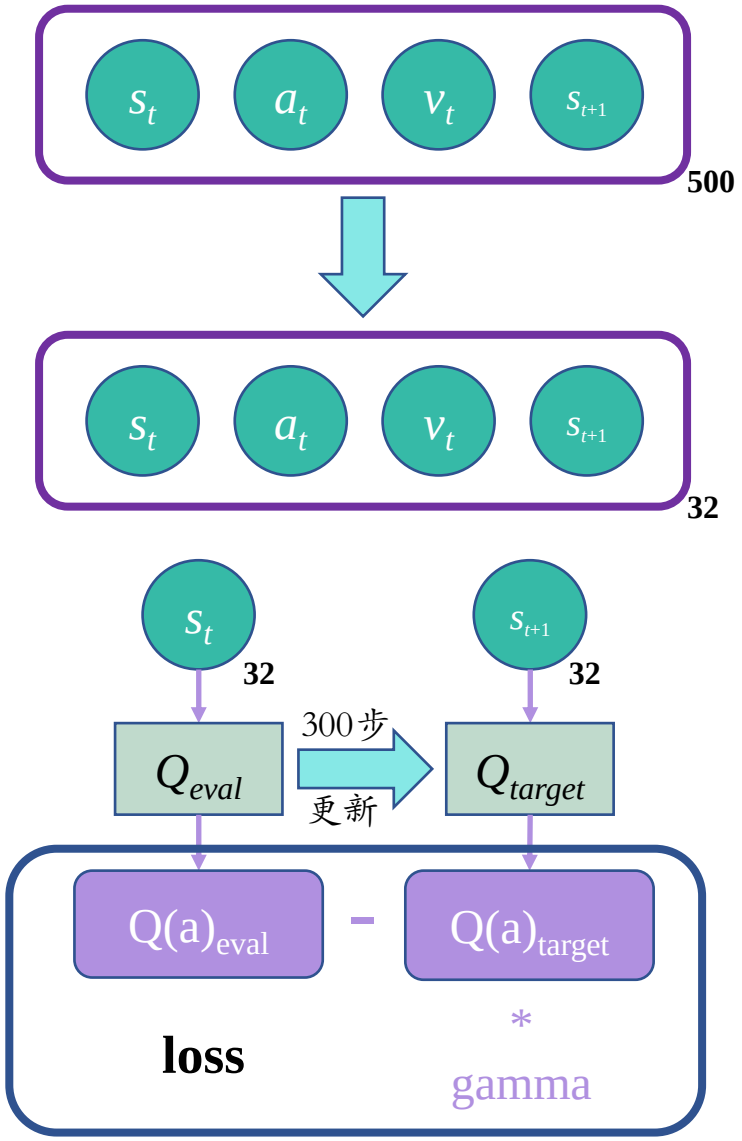
机器人自动走迷宫



ε	**epsilon**	**0.2**	**贪婪度 greedy**：此实验算法中以epsilon概率进行随机决策，以(1-epsilon)概率进行贪婪决策
α	**alpha**	**0.2**	**学习率**：在强化学习中，学习率α越大,表示采用新的尝试得到的结果比例越大,保持旧的结果的比例越小
γ	**gamma**	**0.8**	**奖励递减值(折现率)**：强化学习中，期望奖励会以奖励乘以奖励递减值的形式体现
	target iteration	**300**	**Q-target更新率**：每300步用eval网络参数更新一次target网络
	memory size	**500**	**记忆库**：用于重复学习的记忆库
	batch size	**32**	**批大小**：一次训练的样本数目

$$Q(a)_{target} = reward + gamma * max\ Q(s\_\{t+1\})$$

机器人自动走迷宫



Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

截止时间：2021/01/12 24:00