

Reinforcement Learning

Introduction and Model-Free Learning

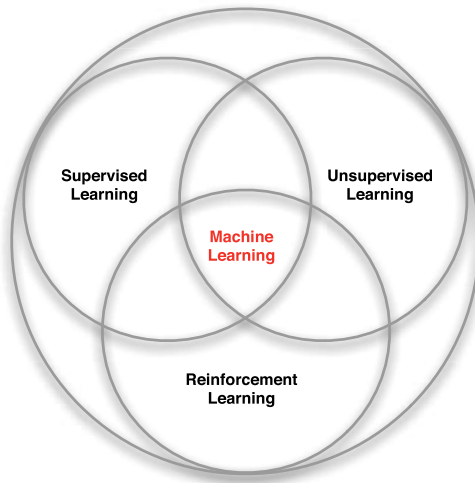
Davide Abati

December 10, 2017

University of Modena and Reggio Emilia

All this material is a free re-arrangement of [David Silver's UCL Course on RL](#).
You are also encouraged to take a look to his [Youtube lectures](#).

What is reinforcement learning?



What makes reinforcement learning different from other machine learning paradigms?

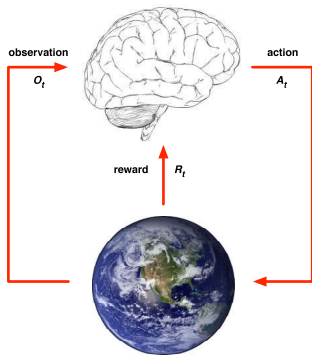
- There is no supervisor, only a *reward* signal.
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d. data)
- Agent is *active*: its actions affect the environment he lives in.

- A **reward** R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximise cumulative reward over an episode

- Fly stunt manoeuvres in a helicopter
 - +ve reward for following desired trajectory
 - -ve reward for crashing
- Defeat the world champion at Go
 - +ve/-ve reward for winning/losing a game
- Make a humanoid robot walk
 - +ve reward for forward motion
 - -ve reward for falling over
- Play Atari games better than humans
 - +ve reward for increasing/decreasing score

Inside a RL agent

- Goal: *select actions to maximise total future reward*
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward.
 - A financial investment may take months to mature
 - Refuelling a helicopter now might prevent a crash in several hours
 - Blocking opponent moves might help winning chances many moves from now



- At each step t the agent:
 - Receives observation O_t
 - Receives scalar reward R_t
 - Executes action A_t
- The environment:
 - Receives action A_t
 - Emits observation O_{t+1}
 - Emits scalar reward R_{t+1}
- t increments at env. step

- The **history** is the sequence of observations, actions, rewards

$$H_t = O_1, R_1, A_1, \dots, A_{t-1}, O_t, R_t$$

- The **state** is the information used to determine what happens next.
 - It is a function of the history:

$$S_t = f(H_t)$$

Agent state S_t^a

whatever information the agent uses
to pick the next action

it is the information used by RL algo-
rithms

Environment state S_t^e

whatever data the environment uses
to pick the next observation/reward

usually not visible by the agent

- **Full observability**: agent directly observes environment state
- **Partial observability**: agent indirectly observes environment state

- An agent may include one or more of these components:
 - Policy: agent's behaviour function
 - Value function: how good is each state and/or action
 - Model: representation of the environment

- A **policy** is the agent's behaviour
- It is a map from state to action
- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

Definition

The return G_t is the total discounted reward from time-step t .

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The discount $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
- γ close to 0 leads to *myopic* evaluation
- γ close to 1 leads to *far-sighted* evaluation

- Value function is a prediction of future reward
- Used to evaluate goodness/badness of states
- And therefore to select between actions

Definition

The *state-value function* $v_{\pi}(s)$ is the expected return starting from state s , and then following policy π

$$v_{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s]$$

Definition

The *action-value function* $q_{\pi}(s, a)$ is the expected return starting from state s , taking action a , and then following policy π

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a]$$

The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

$$\begin{aligned}v_{\pi}(s) &= \mathbb{E}_{\pi}[G_t | S_t = s] \\&= \mathbb{E}_{\pi}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\&= \mathbb{E}_{\pi}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) | S_t = s] \\&= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\&= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s]\end{aligned}$$

The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

$$v_{\pi}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s]$$

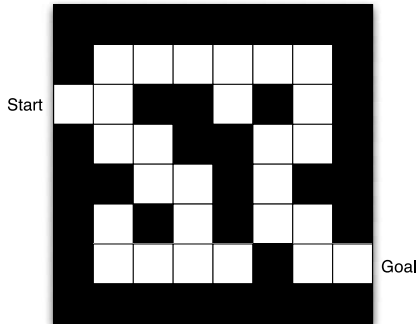
The action-value function can similarly be decomposed,

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

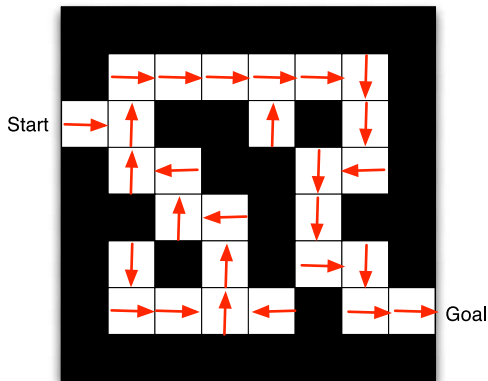
- A **model** predicts what the environment will do next
- \mathcal{P} predicts the next state
- \mathcal{R} predicts the next (immediate) reward

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

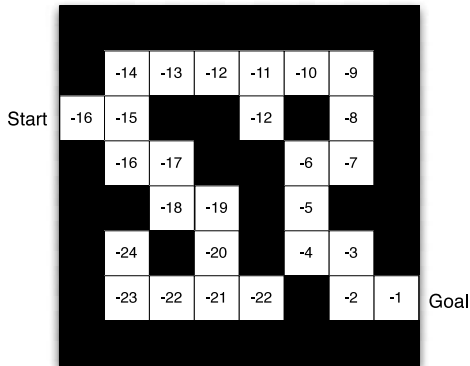


- Rewards: -1 per time-step
- Actions: N, S, W, E
- States: Agent's location

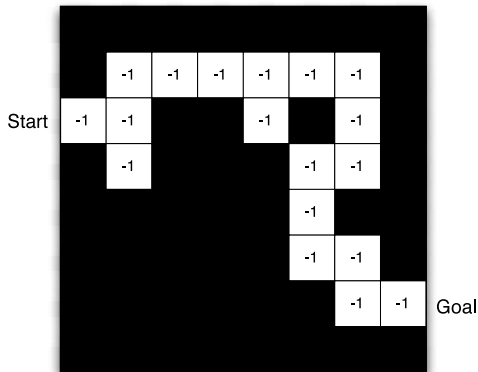


- Arrows represent policy $\pi(s)$ for each state s

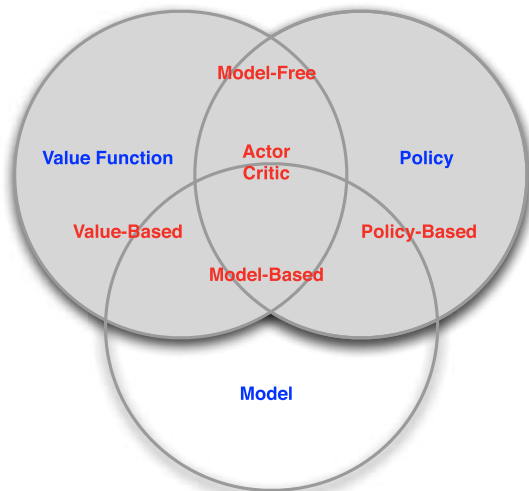
Maze example: value function



- Numbers represent policy $v_{\pi}(s)$ for each state s



- Grid layout represent transition model $\mathcal{P}_{ss'}^a$
- Numbers represent immediate reward R_s^a from each state s (same for all a)



Model-free prediction

- Model-free prediction
 - estimate the value function given a policy in a non-observable environment
 - Monte-Carlo Learning
 - Temporal-Difference Learning

- MC methods learn directly from episodes of experience
- MC is *model-free*: no explicit knowledge of environment mechanisms
- MC learns from complete episodes
 - Caveat: can only apply to *episodic* environments (all episodes must terminate).
- MC uses the simplest possible idea: $\text{value} = \text{mean return}$

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

- To evaluate state s
- Every time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

- Update $V(s)$ incrementally after episode $S_1, A_1, R_2, \dots, S_T$
- Compute return G_t
- For each state S_t with return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

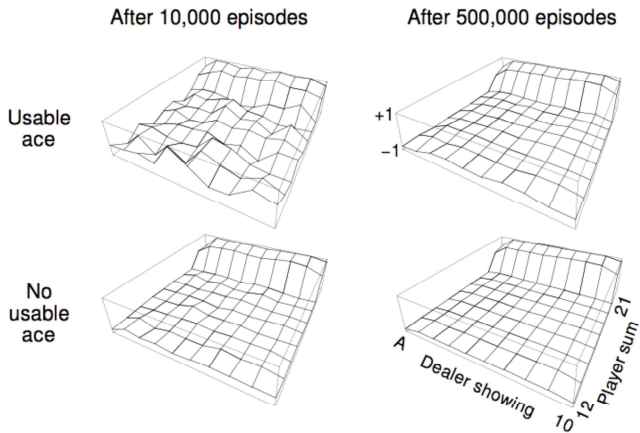
- Usually a running mean is employed, i.e. forget old episodes

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

- States (200 of them):
 - Current sum (12-21)
 - Dealer's showing card (ace-10)
 - Do I have a useable ace? (yes-no)
- Action **stick**: Stop receiving cards (and terminate)
- Action **twist**: Take another card (no replacement)
- Reward for **stick**:
 - +1 if sum of cards $>$ sum of dealer cards
 - 0 if sum of cards = sum of dealer cards
 - -1 if sum of cards $<$ sum of dealer cards
- Reward for **twist**:
 - -1 if sum of cards $>$ 21 (and terminate)
 - 0 otherwise
- Transitions: automatically **twist** if sum of cards $<$ 12



Blackjack Value Function after Monte-Carlo Learning



Policy: **stick** if sum of cards ≤ 20 , otherwise **twist**

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
 - Update value $V(S_t)$ toward actual return G_t

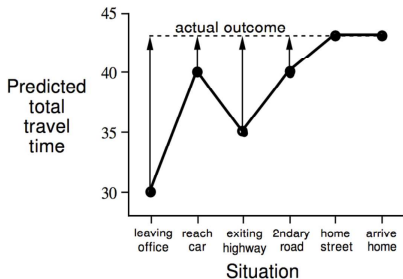
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

- Simplest temporal-difference learning algorithm: TD(0)
 - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

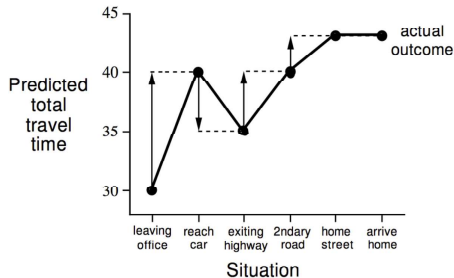
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- $R_{t+1} + \gamma V(S_{t+1})$ is called the TD target
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the TD error

Changes recommended by
Monte Carlo methods ($\alpha=1$)



Changes recommended
by TD methods ($\alpha=1$)



- TD can learn *before* knowing the final outcome
 - TD can learn online after every step
 - MC must wait until end of episode before return is known
- TD can learn without the final outcome
 - TD can learn from incomplete sequences
 - MC can only learn from complete sequences
 - TD works in continuing (non-terminating) environments
 - MC only works for episodic (terminating) environments

- Return $G_t = R_{t+1} + R_{t+2} + \dots + \gamma^{T-1}R_T$ is *unbiased* estimate of $v_\pi(S_t)$
- True TD target $R_{t+1} + v_\pi(S_{t+1})$ is unbiased estimate of $v_\pi(S_t)$
- TD target $R_{t+1} + v(S_{t+1})$ is biased estimate of $v_\pi(S_t)$
- TD target is much lower variance than the return:
 - Return depends on many random actions, transitions, rewards
 - TD target depends on one random action, transition, reward

Model-free control

- Model-free prediction
 - estimate the value function given a policy in a non-observable environment
 - Monte-Carlo Learning
 - Temporal-Difference Learning
- Model-free control
 - find a good policy in a non-observable environment
 - On-Policy Monte-Carlo Control
 - On-Policy Temporal-Difference Learning
 - Off-Policy Learning

- Given a policy π
- Evaluate the policy π

$$v_{\pi}(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$$

$$q_{\pi}(s, a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s, A_t = a]$$

- Improve the policy by acting greedily with respect to v_{π}

$$\pi' = \text{greedy}(v_{\pi})$$

$$\pi' = \text{greedy}(q_{\pi})$$

- In general, need more iterations of improvement / evaluation
- In model-free contexts, action-value function $Q(s, a)$ is our only option



"Behind one door is tenure - behind the other
is flipping burgers at McDonald's."

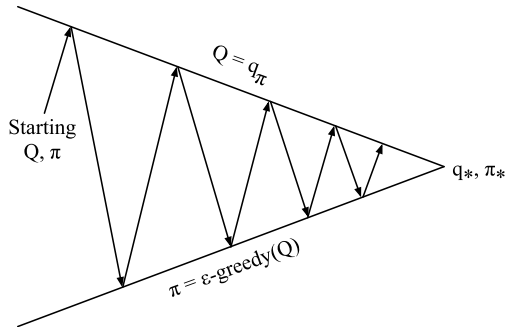
Copyright © 2003 David Farley, d-farley@ibiblio.org

- There are two doors in front of you.
- You open the left door and get reward 0
 $V(\text{left}) = 0$
- You open the right door and get reward +1
 $V(\text{right}) = +1$
- You open the right door and get reward +3
 $V(\text{right}) = +2$
- You open the right door and get reward +2
 $V(\text{right}) = +2$
- \vdots
- Are you sure you've chosen the best door?

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

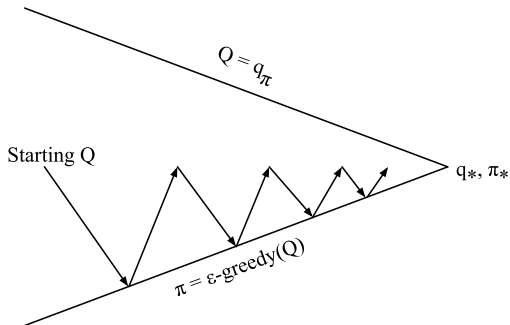
$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{if } a^* = \arg \max_{a \in \mathcal{A}} Q(s, a) \\ \epsilon/m, & \text{otherwise} \end{cases}$$

- On-policy learning
 - “Learn on the job”
 - Learn about policy π from experience sampled from π
- Off-policy learning
 - “Look over someone’s shoulder”
 - Learn about policy π from experience sampled from μ



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement ϵ -greedy policy improvement



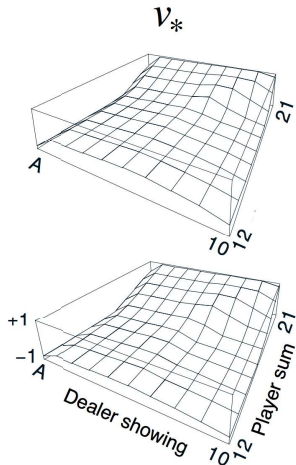
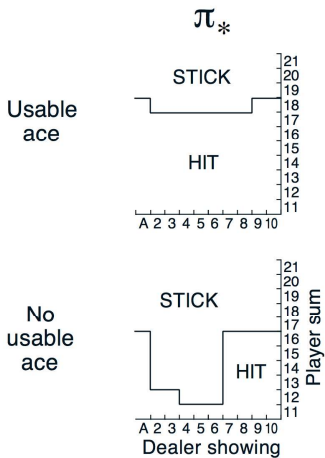
Every episode:

Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

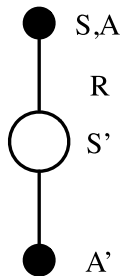
Policy improvement ϵ -greedy policy improvement

Back to the blackjack example

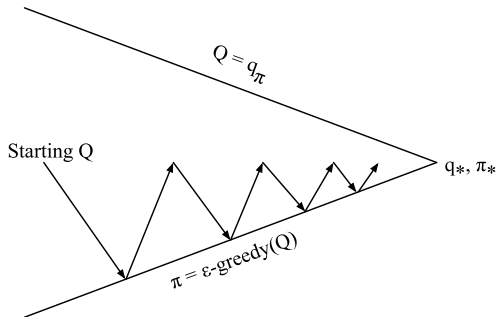




- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
 - Lower variance
 - Online
 - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
 - Apply TD to $Q(S, A)$
 - Use ϵ -greedy policy improvement
 - Update every time-step



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

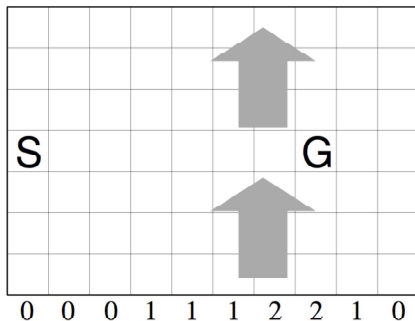


Every time-step:

Policy evaluation with Sarsa, $Q \approx q_\pi$

Policy improvement with ϵ -greedy policy improvement.

```
Initialize  $Q(s, a), \forall s \in S, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal} - \text{state}, \cdot) = 0$   
for each episode do  
  Initialise  $S$   
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)  
  for each step of episode do  
    Take action  $A$ , observe  $R, S'$   
    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)  
     $Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma(Q(S', A') - Q(S, A)))$   
     $S \leftarrow S'; A \leftarrow A'$   
  end for  
end for
```

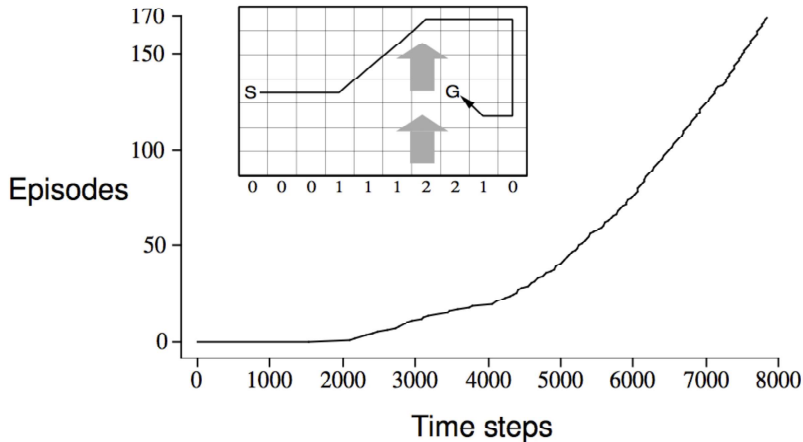


standard
moves



king's
moves

- Reward = -1 per time step until reaching goal
- Undiscounted



- Evaluate target policy $\pi(a, s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$
- While following behaviour policy $\mu(a|s)$

$$\{S_1, A_1, R_2, \dots, S_T\} \sim \mu$$

- Why is this important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, \dots, \pi_{t-1}$
- Learn about *optimal* policy while following *exploratory* policy
- Learn about *multiple* policies while following *one* policy

- We now consider off-policy learning of action-values $Q(s, a)$
- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + Q(S_{t+1}, A') - Q(S_t, A_t))$$

- We now allow both behaviour and target policies to **improve**
- The target policy π is **greedy** w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = \arg \max_{a'} Q(S_{t+1}, a')$$

- The behaviour policy μ is **ϵ -greedy** w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ &= R_{t+1} + Q(S_{t+1}, \arg \max_{a'} Q(S_{t+1}, a')) \\ &= R_{t+1} + \max_{a'} Q(S_{t+1}, a') \end{aligned}$$

Initialize $Q(s, a), \forall s \in S, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\text{terminal} - \text{state}, \cdot) = 0$

for each episode **do**

 Intialise S

for each step of episode **do**

 Choose A from S using policy derived from Q (e.g., ϵ -greedy)

 Take action A , observe R, S'

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

$S \leftarrow S'$

end for

end for