Introduction to Reinforcement Learning: Part 2

March 9, 2021

Today:

- Learning in RL
- Temporal–Difference
- SARSA
- Q–learning

Recap

Markov Decision Process

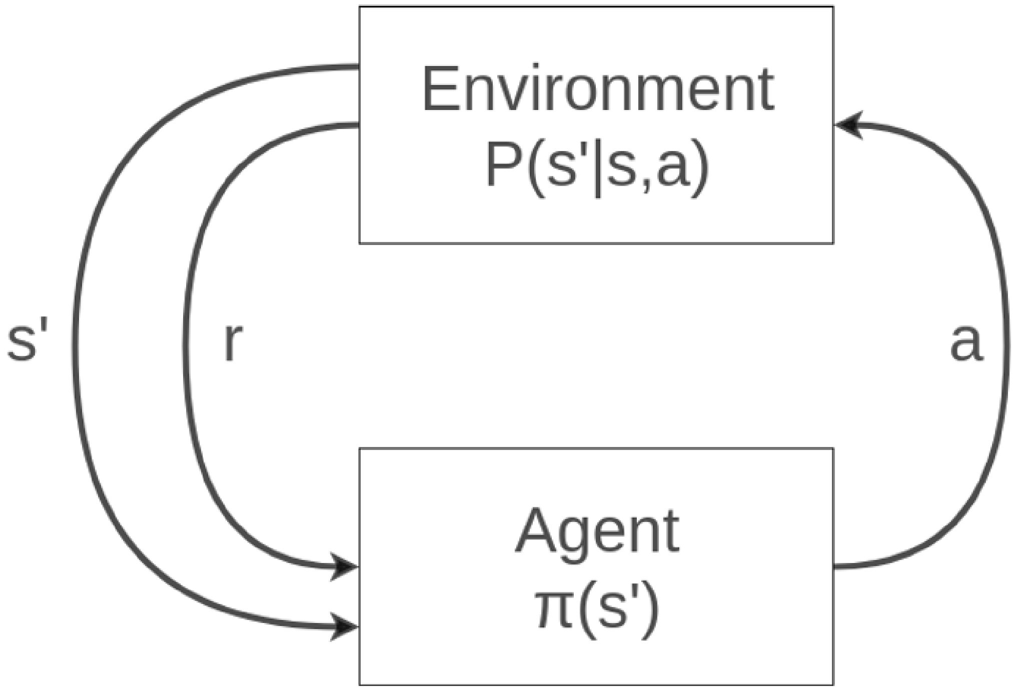
Environment consisting of :

- Set of states $\boldsymbol{S}$
- Set of actions $\boldsymbol{A}$
- Reward function $\boldsymbol{R}(s)$
- A transition model $\boldsymbol{P}(s' \mid s, a)$

Finite MDP

	0	1	2	3
0	-0.04	-0.04	-0.04	-0.04
1	-0.04		-0.04	-1.0
2	-0.04	-0.04	-0.04	+1.0

Agent – Environment in MDPs



Algorithms Recap

Expected discounted utility $U^{\pi}(s) = E\left(R(s) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots + \gamma^n R(s_n)\right)$

Bellman Equations $U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a) U(s')$

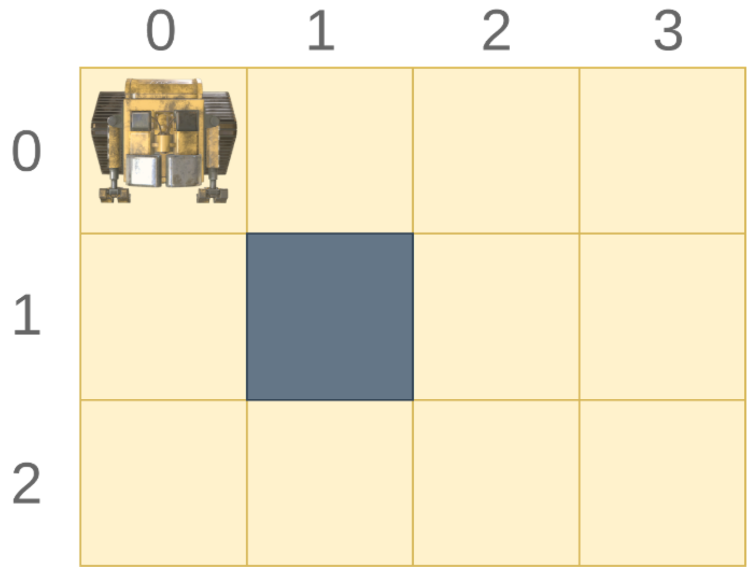
$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U^{\pi}(s')$$

Algorithms:

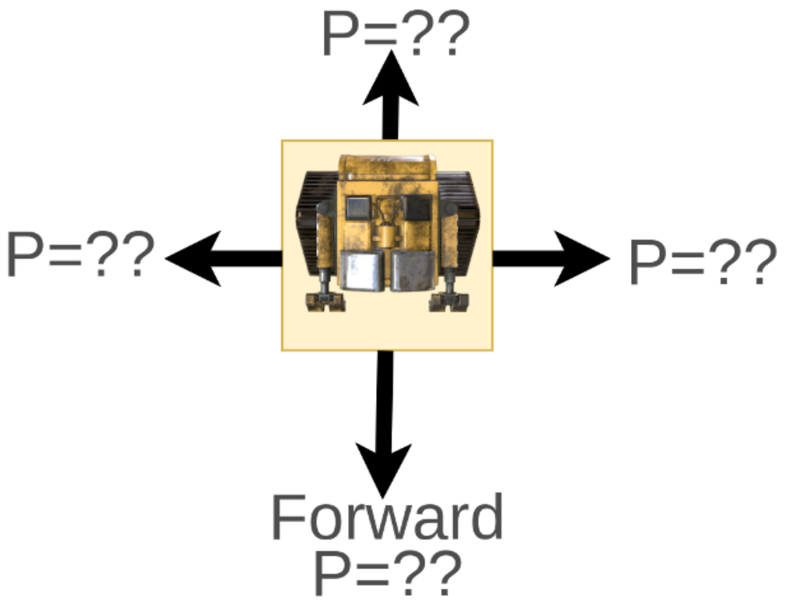
- Value iteration: Finds utilities of optimal policy
- Policy evaluation: Finds utilities of certain policy
- Policy iteration: Finds optimal policy

Transition model is known!

Reinforcement Learning (RL)



Rewards Unknown



Transition Model Unknown

Known Set of States, Set of Actions

Reinforcement Learning (RL)

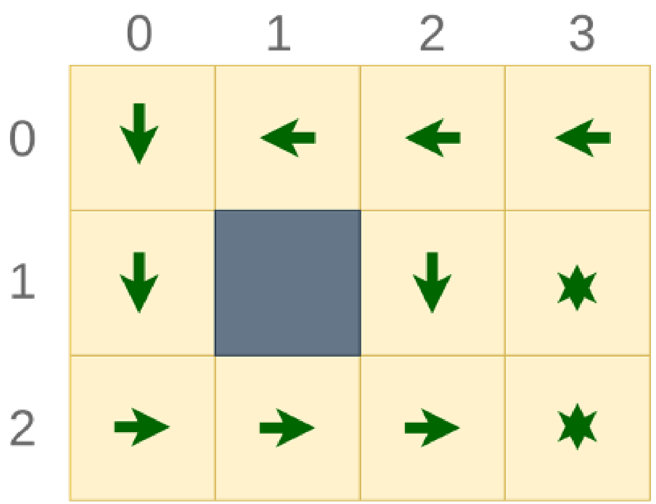
- Learning from interaction with the environment
- Trial and error search

Two approaches:

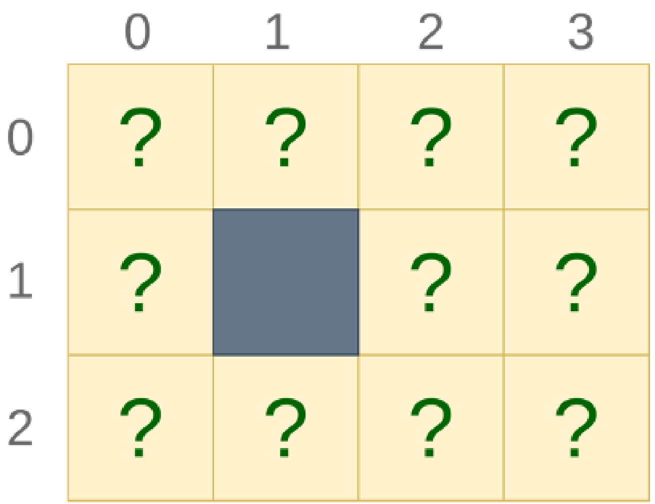
Build a model	Model Free
Estimate $P(s' \mid s, a)$, $R(s)$	Estimate $U^{\pi}(s)$

Reinforcement Learning (RL)

- Passive RL
  - Policy fixed $\pi(s) = a$
  - Not necessarily optimal
  - Goal: Learn utility $U^{\pi}(s)$



- Active RL
  - Policy not fixed
  - Goal: Estimate the optimal policy $\pi(s) = a$

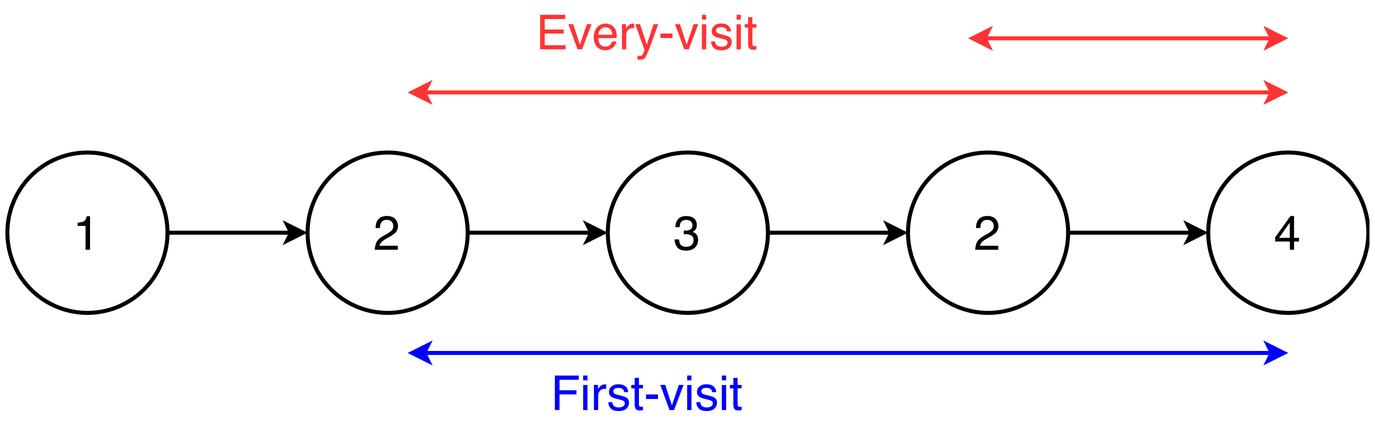


Passive RL: How to estimate utility?

- Given a policy $\pi(s)$
- Cannot use Bellman updates as before: ~~Value Iteration Algorithm~~
- Don’t have $P(s' \mid s, a)$
- Solution: Execute trials to acquire knowledge and estimate the utility function
- The straightforward way is using Monte Carlo Methods

Monte Carlo Policy Evaluation

- **Goal**: learn $U^{\pi}(s)$
- **Given**: some number of episodes under π which contain s
- **Idea**: Average returns observed after visits to s
- **Every–Visit MC**: average returns for every time s is visited in an episode
- **First–visit MC**: average returns only for first time s is visited in an episode
- Both converge asymptotically



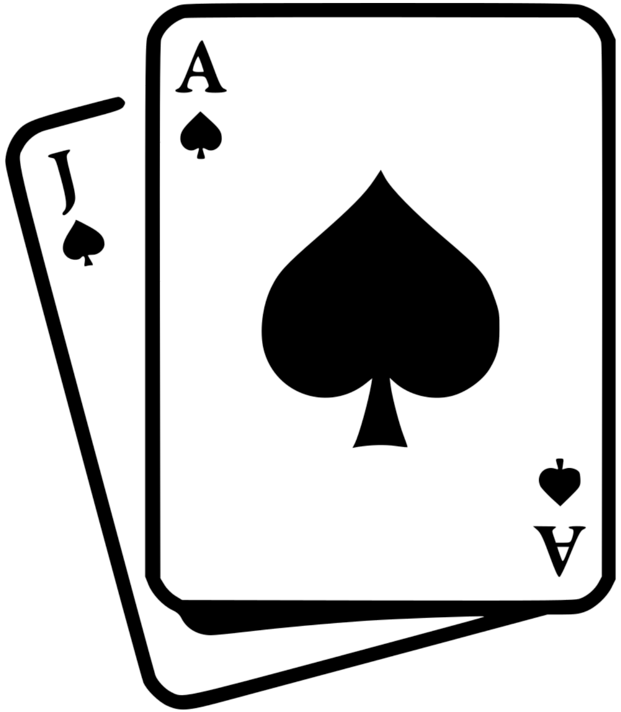
Monte Carlo Policy Evaluation

First–visit Monte Carlo policy evaluation

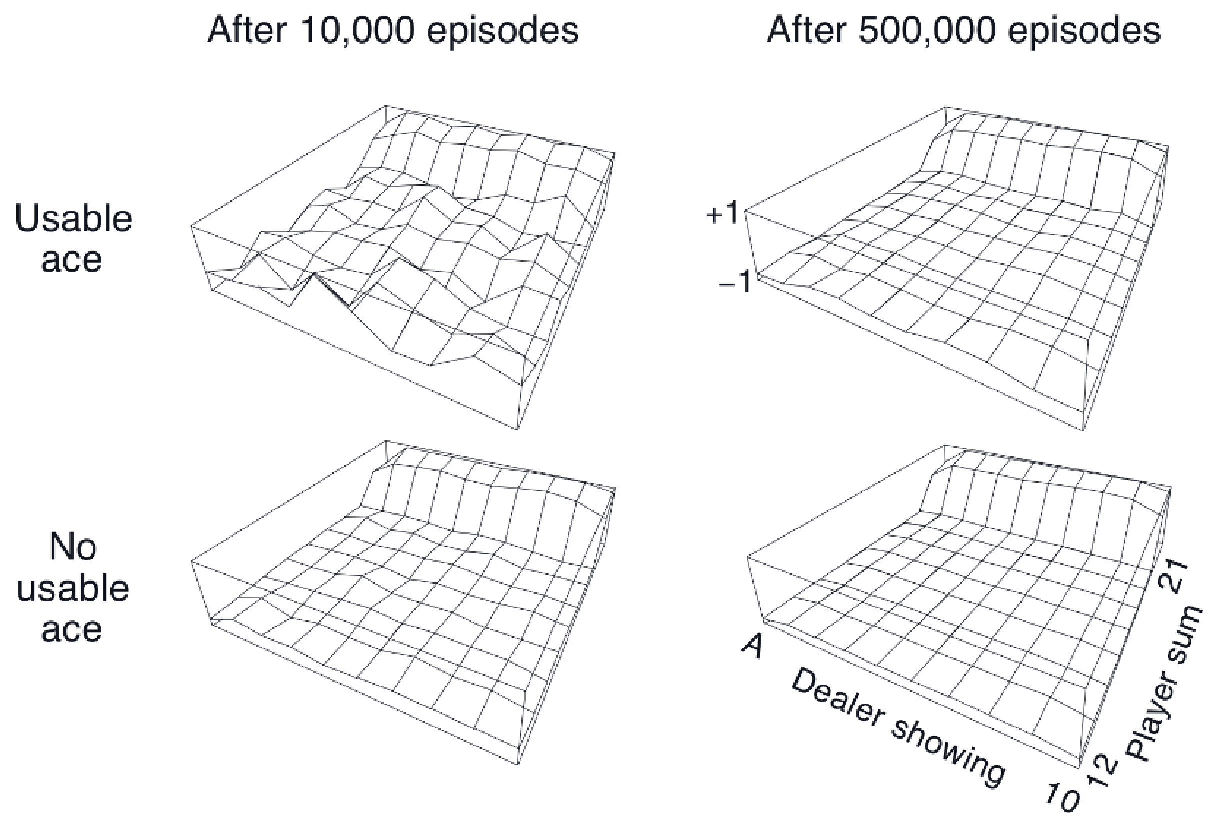
```
Input:
   π ← policy to be evaluated
Initialize:
   U ← an arbitrary utility function
   Returns(s)  ← an empty list for all s
Repeat Forever:
   Generate an episode using π
   For each state s appearing in the episode:
       G ← utility using sequence from first occurence of s
       Append G to Returns(s)
       U(s) ← average(Returns(s))
```

Monte Carlo Policy Evaluation:Blackjack Example

- Objective: Have your card sum be greater than the dealer's without exceeding 21.
- States (200 of them):
  - current sum (12–21)
  - dealer's showing card (ace–10)
  - do I have a useable ace?
- Reward: +1 for winning, 0 for a draw, −1 for losing
- Actions: stick (stop receiving cards), hit (receive another card)
- Policy: Stick if my sum is 20 or 21, else hit
- No discounting ($\gamma = 1$)



Monte Carlo Policy Evaluation:Blackjack Example



Monte Carlo Policy Evaluation

Question:

- Can we extend algorithm to do better with same number of episodes?

Issues:

- Utility updated only at the end of an episode

(inefficient for applications with long episodes)

- Termination not guaranteed

Temporal–Difference Learning

$$Generalrule : NewEstimate \leftarrow OldEstimate + StepSize(Target - OldEstimate)$$

In our case the target is

$$\begin{aligned} U^{\pi}(s) &= E\left(R(s) + \gamma R(s_1 + \gamma^2 R(s_2) + \cdots + \gamma^n R(s_n))\right) \\ &= E\left(R(s) + \gamma U^{\pi}(s_1)\right) \end{aligned}$$

Thus TD: $U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha\left(R(s) + \gamma U^{\pi}(s_1) - U^{\pi}(s)\right)$

In better notation:

$$U(s_t) \leftarrow U(s_t) + \alpha(R(s_t) + \gamma U(s_{t+1}) - U(s_t))$$

Temporal–Difference Learning

Conventions:

- Artificial Intelligence: A Modern Approach, Russell and Norvig

Utility: $U^{\pi}(s) = E\left(R(s) + \gamma R(s_1 + \gamma^2 R(s_2) + \cdots + \gamma^n R(s_n)\right)$

TD: $U(s_t) \leftarrow U(s_t) + \alpha(R(s_t) + \gamma U(s_{t+1}) - U(s_t))$

- Reinforcement Learning: An Introduction, Sutton and Barto

Value: $U^{\pi}(s) = E\left(R(s_1) + \gamma R(s_2 + \gamma^2 R(s_3) + \cdots + \gamma^n R(s_n)\right)$

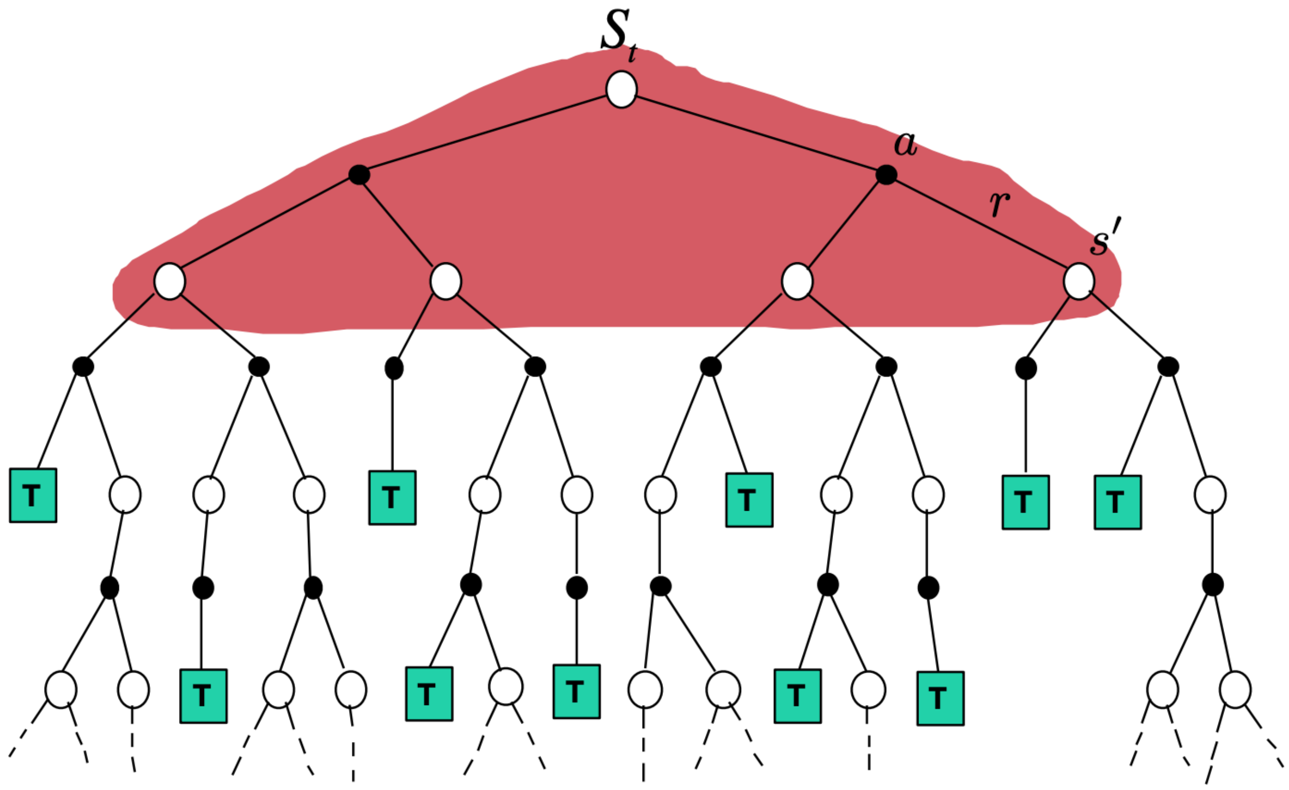
TD: $U(s_t) \leftarrow U(s_t) + \alpha(R(s_{t+1}) + \gamma U(s_{t+1}) - U(s_t))$

Temporal-Difference Algorithm

```
|Input:
|   π ← policy to be evaluated
|Initialize:
|   U ← an arbitrary utility function
Repeat (for each step of episode):
|   A ← action given by π  for S
|   Take action A, observe R', S'
|   U(S) ← U(S) + α [R + γ U(S') −U(S)]
|   S ← S'
|   R ← R'
|   until S is terminal
```

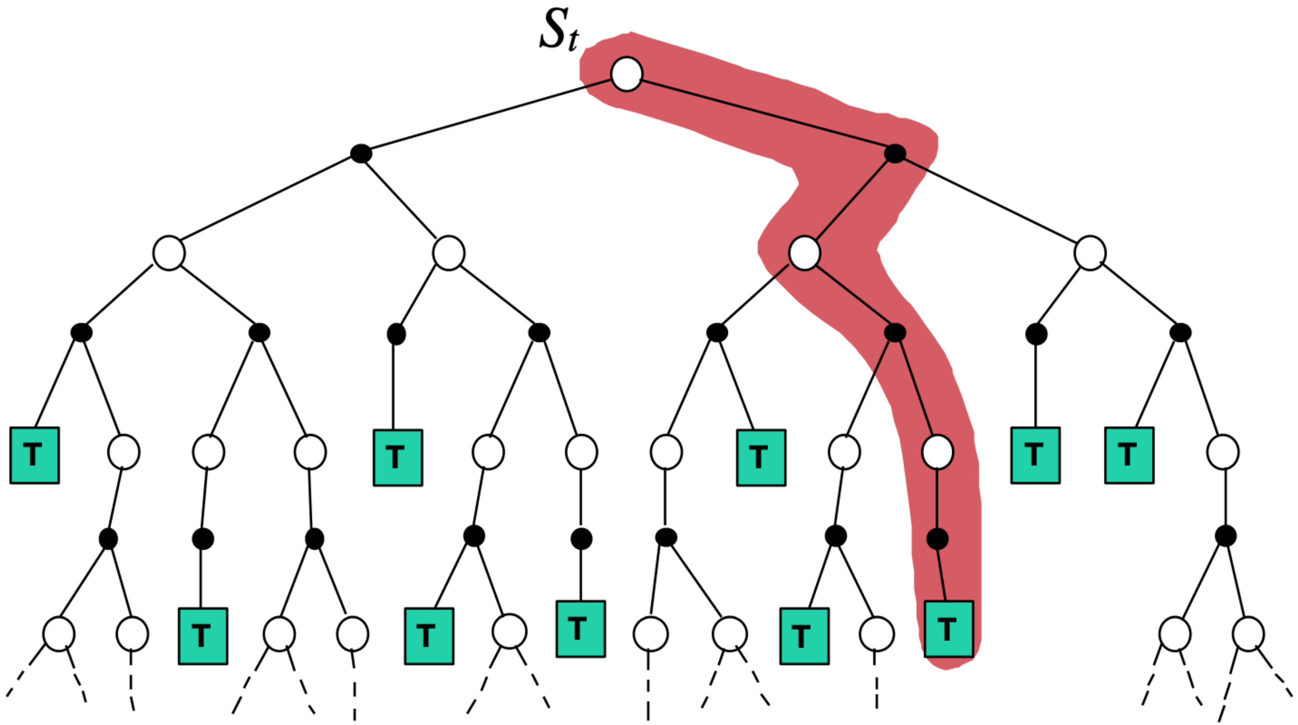
Value Iteration

$$U(s) \leftarrow R(s) + \gamma \max_{a} \sum_{s'} P(s' \mid s, a) U(s')$$



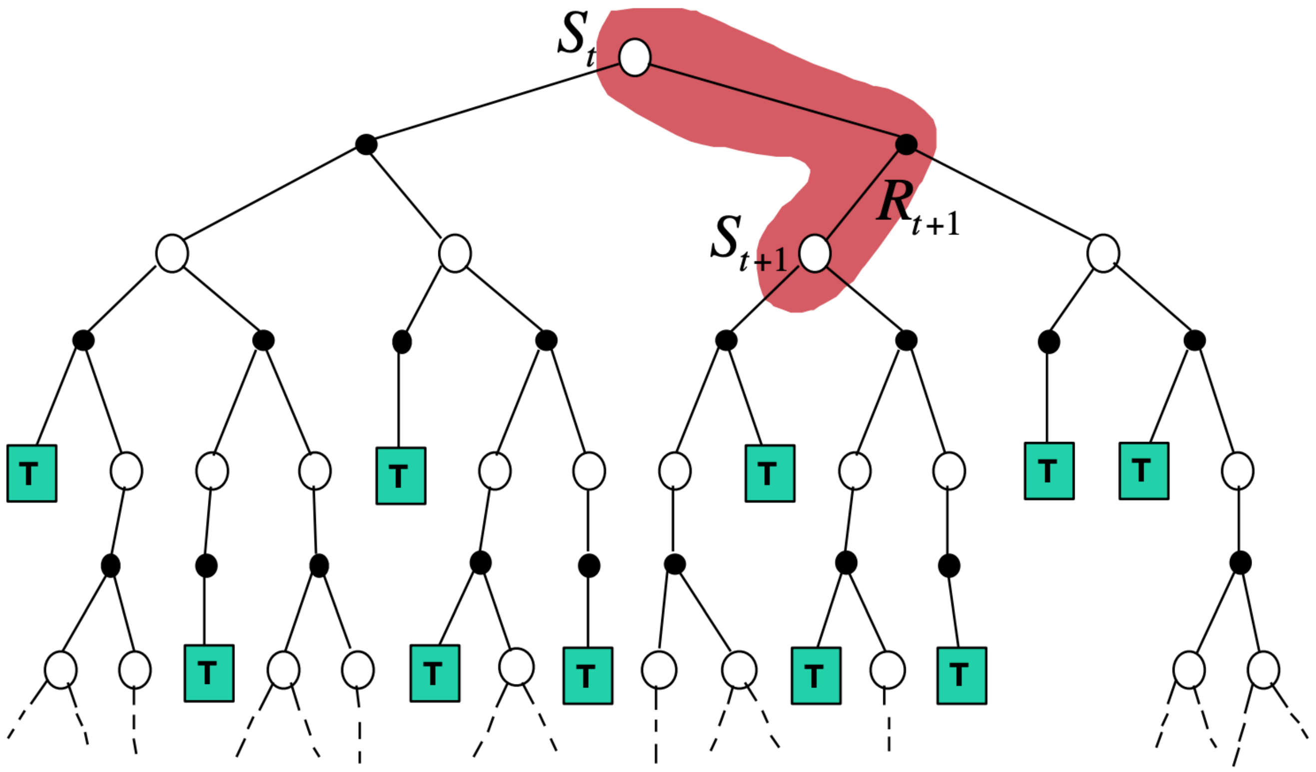
Simple Monte Carlo

$$U(s) \leftarrow U(s) + \alpha(U_{\text{episode}}(s) - U(s))$$



Simple Temporal-Difference

$$U(s) \leftarrow U(s) + \alpha[R + \gamma U(s') - U(s)]$$



TD example: Driving home

Consider driving home:

- each day you drive home
- your goal is to try and predict how long it will take at particular stages
- when you leave office you note the time, day, & other relevant info
    - Consider the policy evaluation or prediction task

TD example: Driving home

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

TD example: Driving home

Turn to RL problem

- Rewards: Time elapsed between states
- Utility: Expected time to get home from a state
- $\gamma = 1, \alpha = 1$

			V(s)	V(office)
State	*Elapsed Time (minutes)*	R	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

TD example: Driving home

State	Elapsed Time (minutes)	R	V(s) Predicted Time to Go	V(office) Predicted Total Time
leaving office, friday at 6	0	5	30	30
reach car, raining	5	15	35	40
exiting highway	20	10	15	35
2ndary road, behind truck	30	10	10	40
entering home street	40	3	3	43
arrive home	43		0	43

Update $U(\text{office})$:

$$U(s_t) \leftarrow U(s_t) + \alpha(R(s_{t+1}) + \gamma U(s_{t+1}) - U(s_t))$$

$$U(\text{office}) \leftarrow U(\text{office}) + \alpha(R(\text{car}) + \gamma U(\text{car}) - U(\text{office}))$$

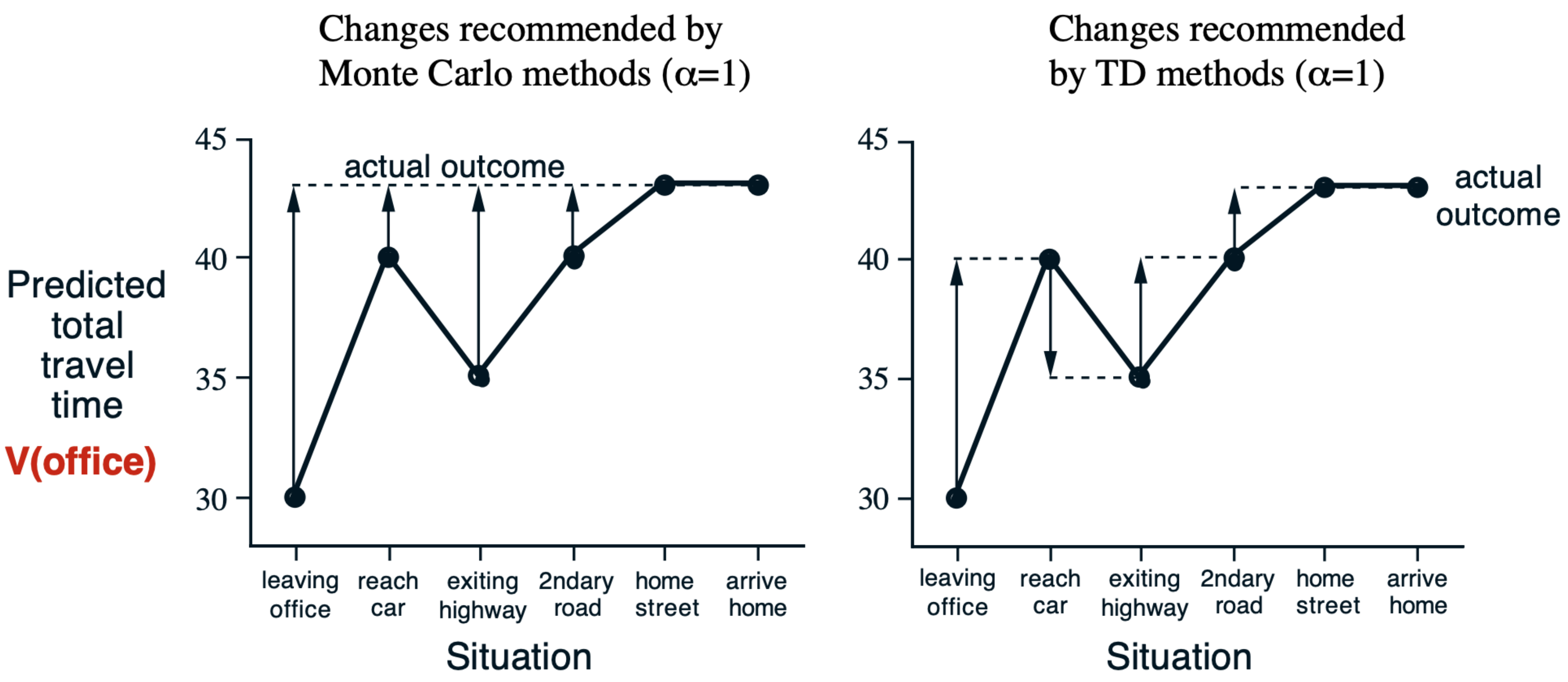
new $U(\text{office}) = 40$, $\Delta = +10$

Update $U(\text{car})$:

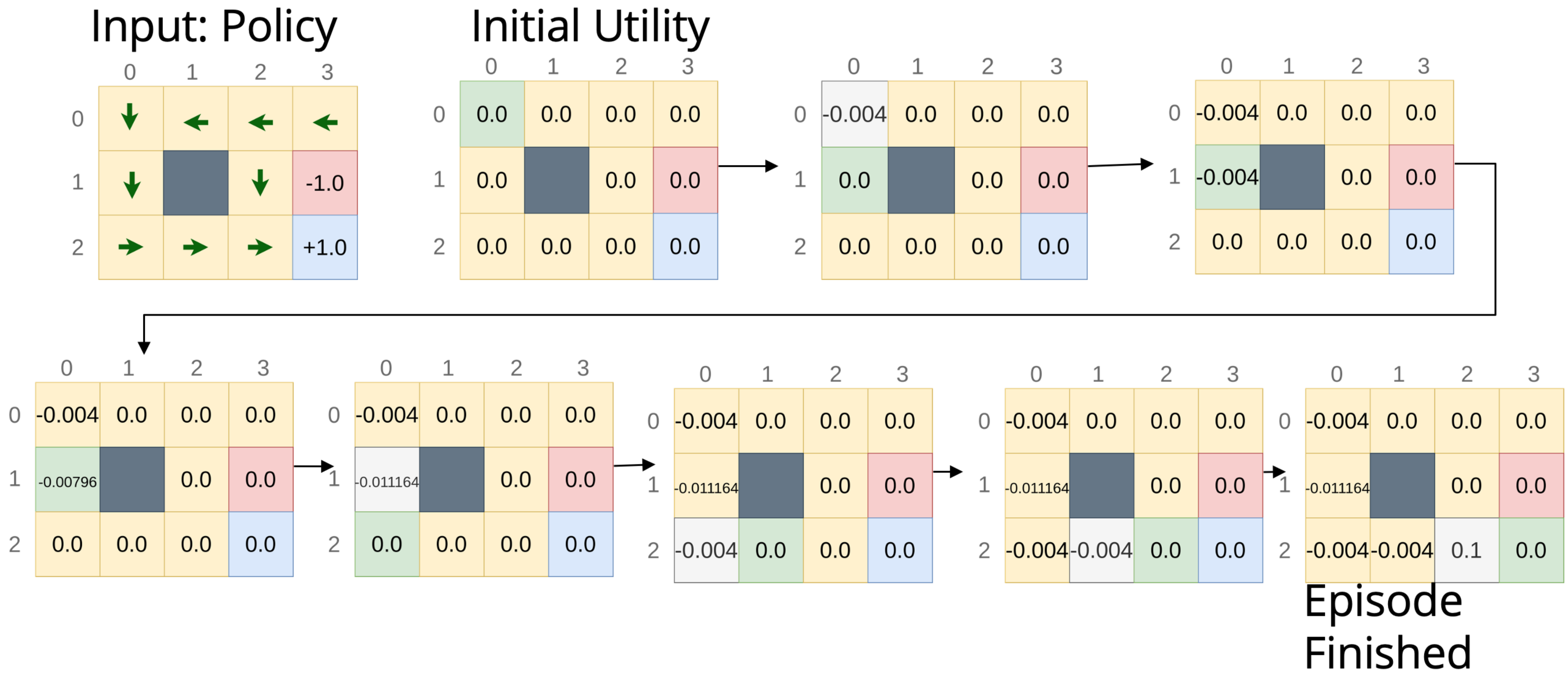
$$U(\text{car}) \leftarrow U(\text{car}) + \alpha(R(\text{highway}) + \gamma U(\text{highway}) - U(\text{car}))$$

new $U(\text{car}) = 30$, $\Delta = -5$

TD example: Driving home

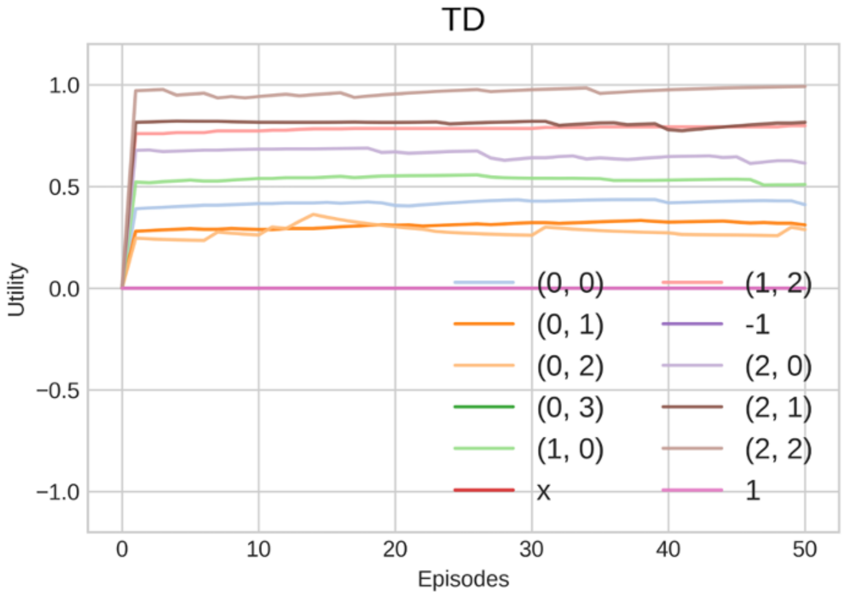


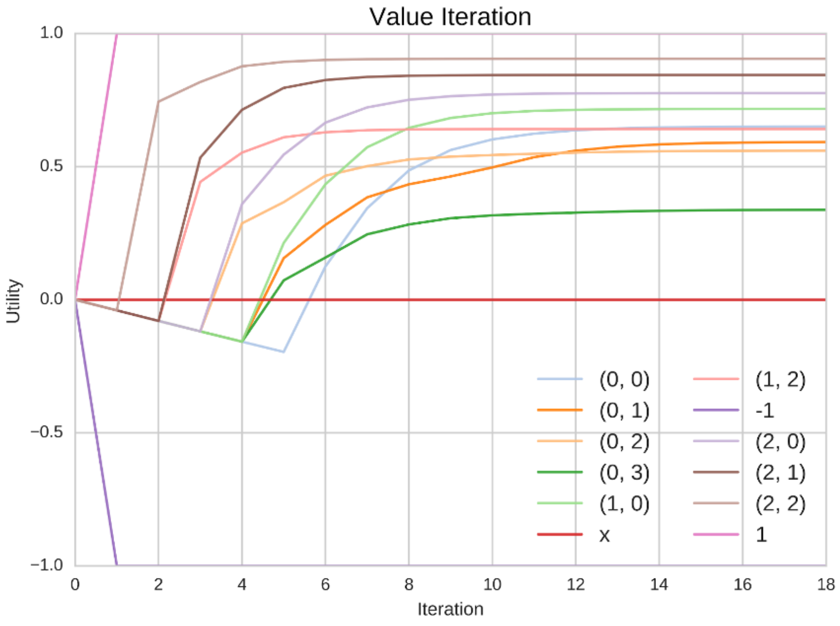
TD:Episode 0 (α=0.1)



$$U(s) \leftarrow U(s) + \alpha[R + \gamma U(s') - U(s)]$$

TD more episodes





REINFORCEjs

Fork me on GitHub

- [About]
- [GridWorld: DP]
- [GridWorld: TD]
- [PuckWorld: DQN]
- [WaterWorld: DQN]

Temporal Difference Learning Gridworld Demo

```
// agent parameter spec to play with (this gets eval()'d on Agent reset)
var spec = {}
spec.update = 'qlearn'; // 'qlearn' or 'sarsa'
spec.gamma = 0.9; // discount factor, [0, 1)
spec.epsilon = 0.2; // initial epsilon for epsilon-greedy policy, [0, 1)
spec.alpha = 0.1; // value function learning rate
spec.lambda = 0; // eligibility trace decay, [0,1). 0 = no eligibility traces
spec.replacing_traces = true; // use replacing or accumulating traces
spec.planN = 50; // number of planning steps per iteration. 0 = no planning

spec.smooth_policy_update = true; // non-standard, updates policy smoothly to follow max_a Q
spec.beta = 0.1; // learning rate for smooth policy update
```

Advantages of TD learning

- TD methods do not require a model of the environment, only experience
- TD methods can be fully **incremental**
  - Make updates before knowing the final outcome
  - Requires less memory
  - Requires less peak computation

- You can **learn without the final outcome**, from incomplete sequences

Active Reinforcement Learning

An Active RL agent can have two (different) policies:

- policy $\rightarrow$ Used to generate actions
  ($\longleftrightarrow$ Interact with environment to gather sample data)
- Learning policy $\rightarrow$ Target action policy to learn
  (the "good"/optimal policy the agent eventually aims to discover through interaction)

If Behavior policy = Learning policy $\rightarrow$ On–policy learning

If Behavior policy $\neq$ Learning policy $\rightarrow$ Off–policy learning

Active Reinforcement Learning

- **On–policy learning**: the behavior policy used to generate samples is the same as the target policy → Agent learns the value of the policy being used, including exploration actions.
  - The used policy is usually "soft" and non–deterministic, to ensure there is always exploration.

- **Off–policy learning**: the behavior and the target policy are different. The target policy is learned regardless (independently) of the actions chosen for exploring the environment. The agent follows a policy but learns the value of a different policy.

Q–function

- So far we used the utility function

$$U^{\pi}(s) = E\left(R(s) + \gamma R(s_1 + \gamma^2 R(s_2) + \cdots + \gamma^n R(s_n)\right)$$

- Policy is fixed here
- In the active learning setting we need actions incorporated
- Now we introduce Q–function (action–value function)

$$Q^{\pi}(s, a) = E\left(R(s) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots + \gamma^n R(s_n) | a\right)$$

- Gives the utility given a state and the action on it

On-policy TD: SARSA

```
|Initialize:
|   Q(s,a)← arbitrarily and Q(terminal state,.) =0
|Repeat (for each episode):
|   Initialize S
|   Choose A from S using policy derived from Q
|   Repeat (for each step of episode):
|       Take action A, observe R, S'
|       Choose A' from S' using policy derived from Q
|       Q(S,A) ← Q(S,A) + α [R + γ Q(S',A') –Q(S,A)]
|       S ← S'; A ← A'
|   until S is terminal
```

$(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ Quintuple used in the update rule gives rise to the name SARSA

On–policy TD: SARSA

Choosing the policy from the Q–function:

- greedy

$$\pi(s) \leftarrow \mathrm{argmax_a Q(s, a)}$$

- ε–greedy

$$\pi(s) = \begin{cases} \mathrm{argmax_a Q(s, a)} & \text{if } \sigma > \epsilon; \\ a \sim A(s) & \text{if } \sigma \leq \epsilon; \end{cases}$$

σ random number $[0, 1]$

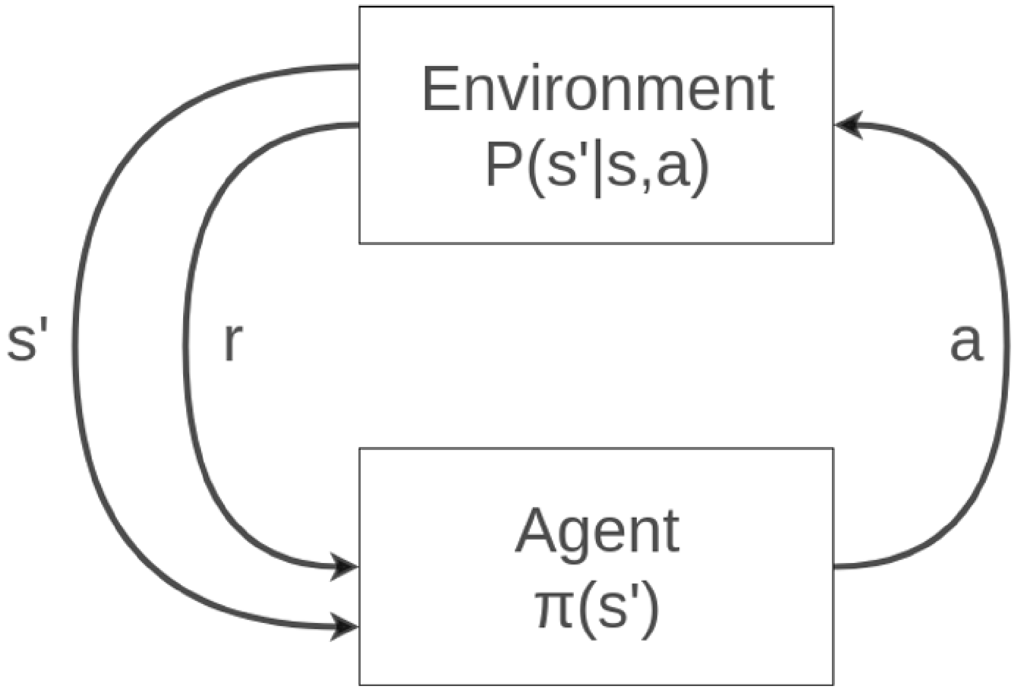
- **Exploration – Exploitation trade–off**

Off–policy TD: Q–learning

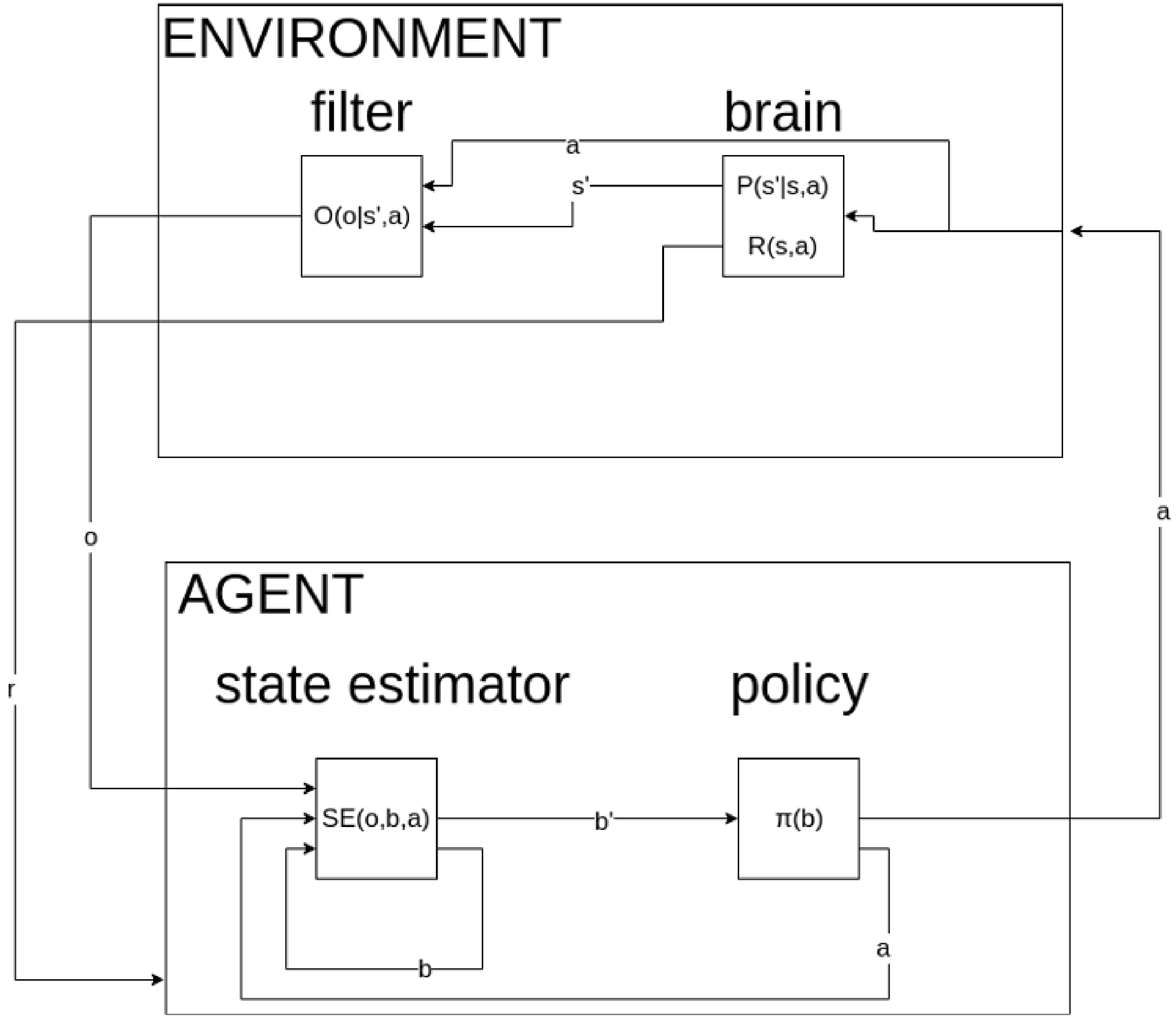
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R(s_{t+1}) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

```
|Initialize:
|   Q(s,a)← arbitrarily and Q(terminal state,.) =0
|Repeat (for each episode):
|   Initialize S
|   Repeat (for each step of episode):
|       Choose A from S using policy derived from Q (e.g., ε–greedy)
|       Take action A, observe R, S’
|       Choose A’ from S’ using policy derived from Q
|       Q(S,A) ← Q(S,A) + α [R + γ max_a Q(S’,a) –Q(S,A)]
|       S ← S’
|   until S is terminal
```

Agent – Environment in MDPs



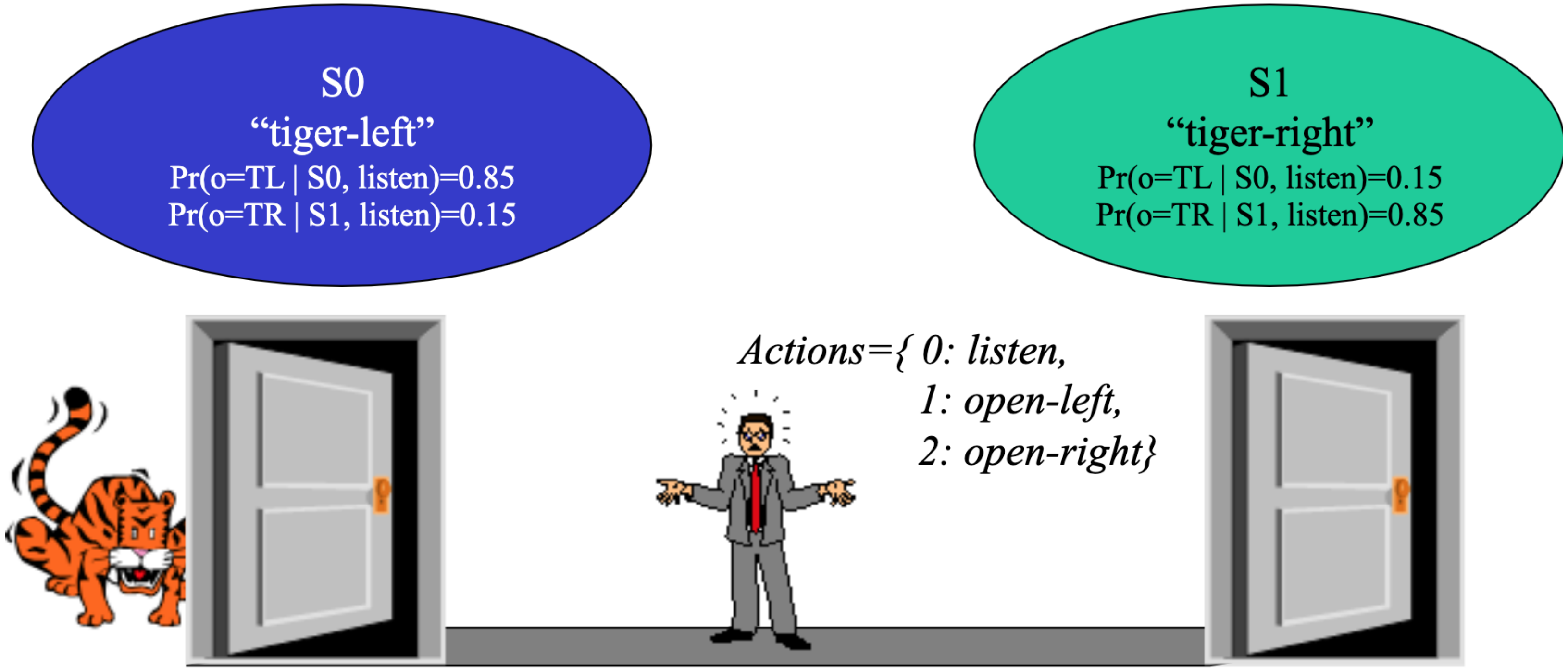
Agent – Environment in POMDPs



Compare with MDPs

- Observations
- State estimator: Generating belief states

Classic POMDP example



Reward Function

- *Penalty for wrong opening: -100*
- *Reward for correct opening: +10*
- *Cost for listening action: -1*

Observations

- *to hear the tiger on the left (TL)*
- *to hear the tiger on the right(TR)*

Source: Hui Huang U. Illinois

References

- https://mpatacchiola.github.io/blog/2016/12/09/dissecting-reinforcement-learning.html
- Artificial Intelligence: A Modern Approach, Russell and Norvig
- Reinforcement Learning: An Introduction, Sutton and Barto
- https://github.com/cmarasinou/WALLE-RL
  - Wall-E grid-world implementation
- Demos: https://cs.stanford.edu/people/karpathy/reinforcejs/