Reinforcement Learning

Tabular Methods

Jakub Chojnacki

About the Author

Who Am I

Admin

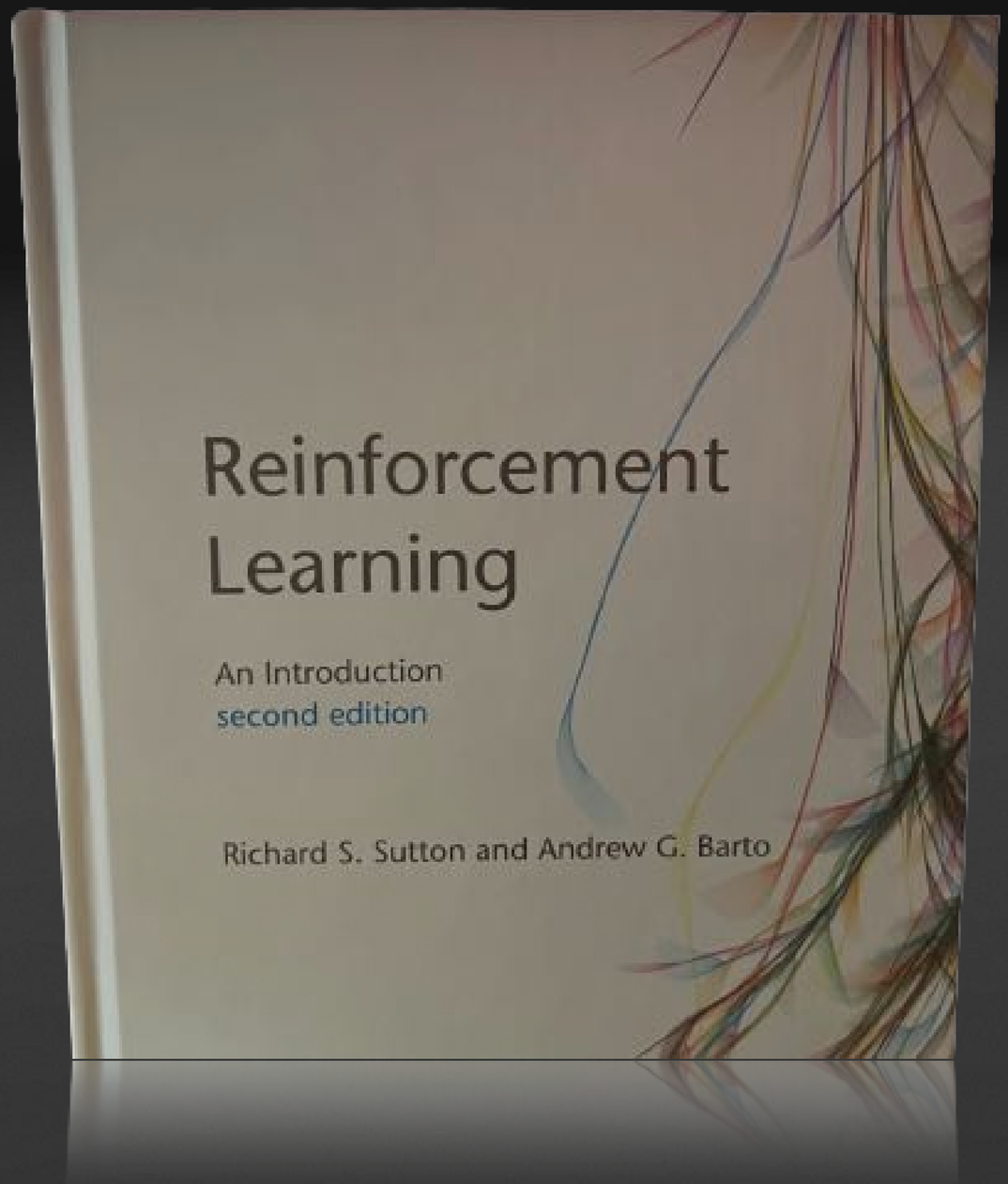
What & When

1. **Introduction** - **[Office: 2.11.2022] & [Online: 3.11.2022] @ 12.00 pm - 1.00 pm**
    1. Deep dive into RL systems
    2. Intuition building
2. **Tabular methods** - **[Office: 9.11.2022] & [Online: 10.11.2022] @ 12.00 pm - 1.00 pm**
    1. Dynamic programming
    2. Monte Carlo methods
    3. Temporal difference methods
3. **Approximate methods** - **[Office: 16.11.2022] & [Online: 17.11.2022] @ 12.00 pm - 1.00 pm**
    1. Intro to Deep Reinforcement Learning
    2. Value function approximation
    3. Policy gradient methods
4. **Modern Deep Reinforcement Learning** - **[Office: 23.11.2022] & [Online: 24.11.2022] @ 12.00 pm - 1.00 pm**
    1. Grokking modern algorithms - { DQN, PPO, DDPG, TD3, SAC, *TQC, *QR-DQN }
    2. Tips and tricks
5. **Coding session** - **[Office: 30.11.2022] & [Online: 1.12.2022] @ 12.00 pm - 1.00 pm**
    1. Learn how to code RL environment using OpenAI API
    2. Choose algorithm, solve the problem and evaluate your agent

Materials

What was used

1. Videos: [David Silver] “Introduction to Reinforcement Learning 2015”
2. Book: [Sutton & Barto] “Reinforcement Learning An Introduction 2nd edition”
3. Book: [Miguel Morales] “Grokking Deep Reinforcement Learning”



Agenda
Topics covered

1. Dynamic Programming
2. Policy Evaluation
3. Generalised Policy Iteration
4. Model Free Prediction Methods
5. On-Policy and Off-Policy
6. Model Free Control Methods

Dynamic Programming

Solve subproblems in order to solve complex problems

Boston Dynamics, Atlas

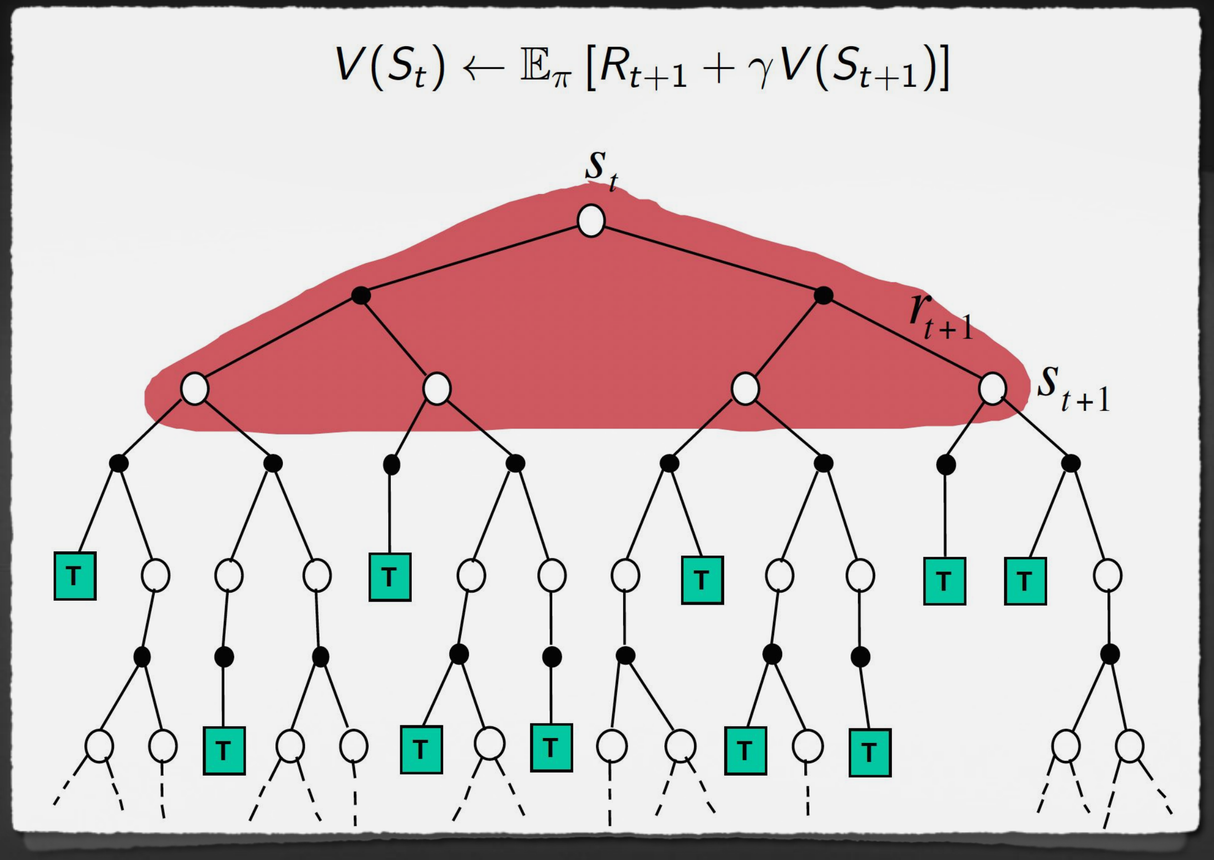
Dynamic Programming

General solution to a problem

1. A method for solving **complex** problems,
2. By breaking them down into **subproblems**
   - Solve the subproblems
   - Combine solutions to subproblems
3. **Markov Decision Process** [MDP] satisfies the properties of Dynamic Programming
   - Optimal solution can be decomposed into subproblems

     Bellman Equation *gives recursive decomposition*
   - Solutions can be cached and reused

     Value function *stores and reuses solutions*



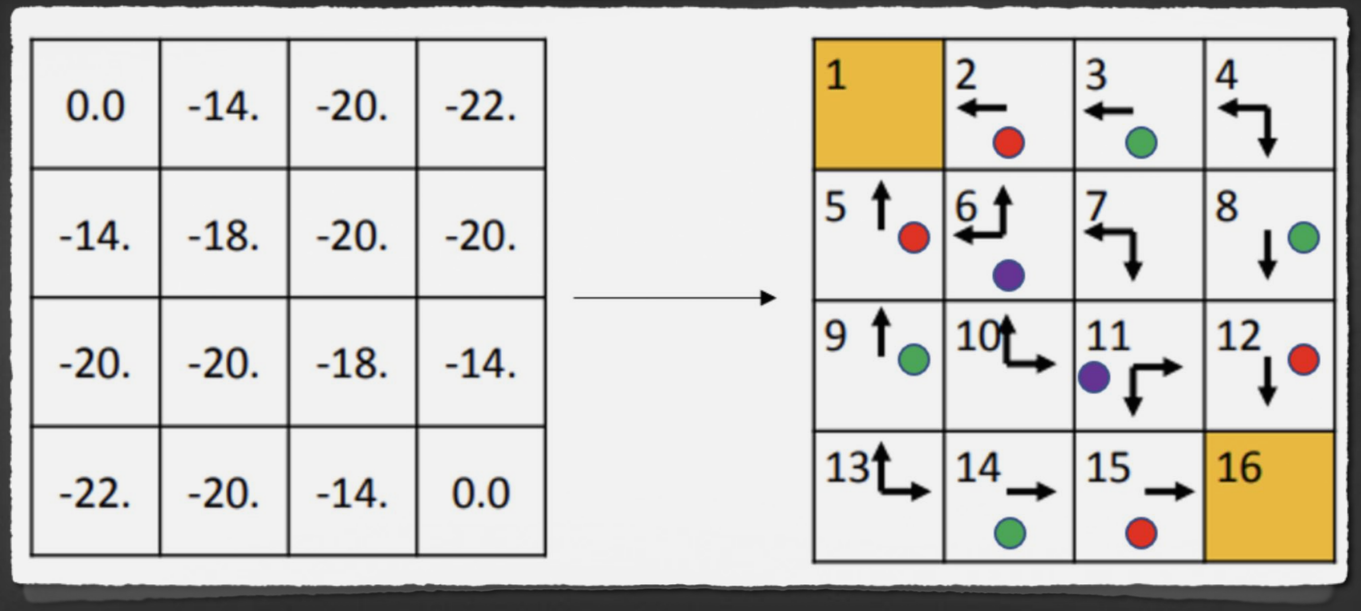
Dynamic Programming Backup Tree

DP Connection to RL

Prediction and Control

1. **Prediction**
   - **Iterative** process of finding the **true value function** for given policy [v_π]
2. **Control**
   - **Iterative** process of **updating** our policy towards **optimality** [π_*]
   - If we know the **optimal value function** [v_*] we can extract optimal policy [π_*]

Dynamic programming assumes full knowledge of an MDP - “cheating”



Mapping value function into policy

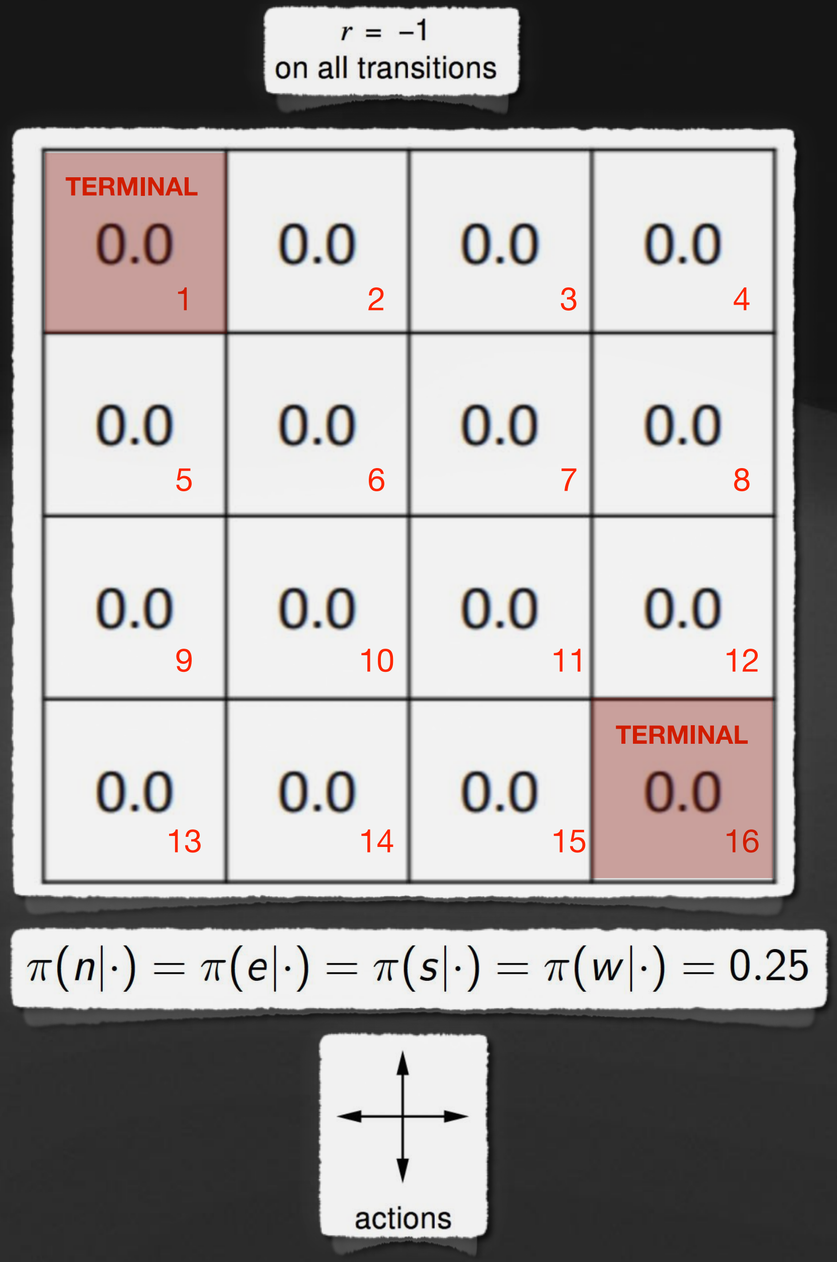
Policy Evaluation

Finding true value function

Boston Dynamics, Atlas

Policy Evaluation
Problem structure

1. Let's look at 4x4 grid world with labelled states from 1 to 16
2. Non terminal states [2, …, 15]
   - We aim to calculate the **true** values for those states
3. Two terminal states [1 ,16]
   - These are our **target** states and will always have a **value of 0**
   - Stepping at **terminal** state **end's** the episode
4. Reward **[-1]** on every **timestep** until reaching the **terminal** sate
5. Agent follows the **uniform random policy**
   - 25% each action no matter the state



Iterative Policy Evaluation

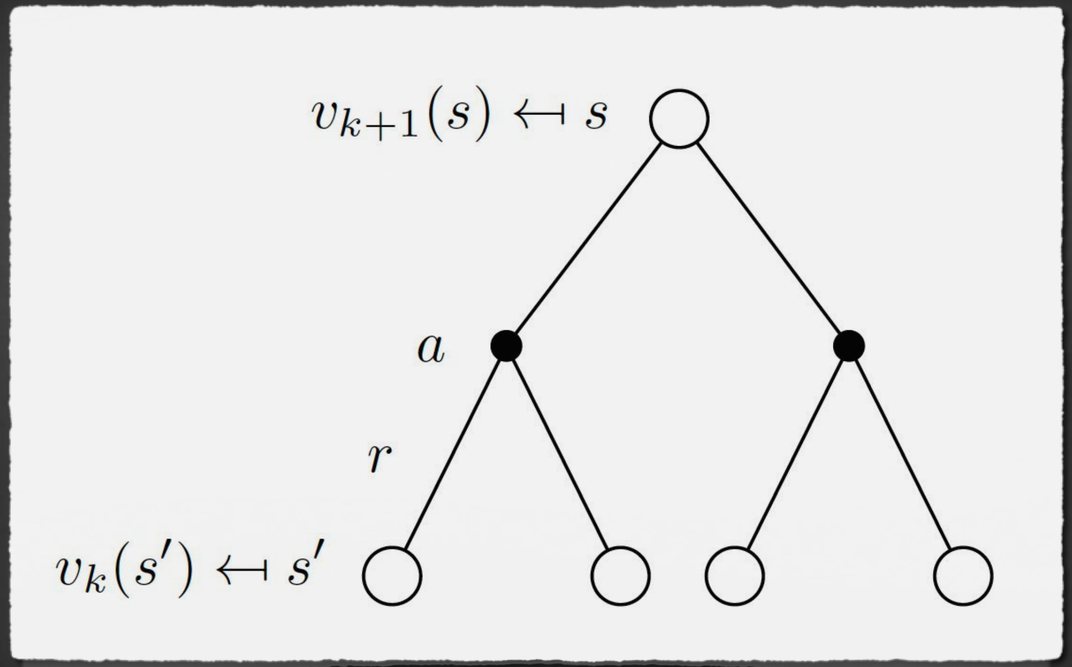
Evaluating a given policy

1. **Problem**
   - **Evaluate** a given policy
2. **Solution**
   - Iterative application of **Bellman Expectation** backup
   - $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$

Synchronous backup solution

- At each iteration $k+1$
- For all states $s \in S$
- Update $v_{k+1}(s)\ from\ v_k(s')$
- Where s' is a successor state of s

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$



Bellman Expectation Equation Backup Tree

Policy Evaluation

First iteration

1. Given random uniform policy
2. Reward of [- 1] per timestep until termination
3. Discount factor $\gamma = 1$

Let's do the **first iteration** and calculate the **value function** for state [6]

Problem

- find value function for state 6

  $v_1(6)$ - how good is to be in state [6] when following random uniform policy **[iteration 1]**

Assumptions

- $\pi(a|6)$ **- 25% [random uniform policy]**

  Probability of taking action [a] in state 6

- $p(s', r|6, a)$ - **100% [deterministic transition function]**

  Probability of **transition** to state [s'] with immediate reward [r] when taking action [a] in state 6

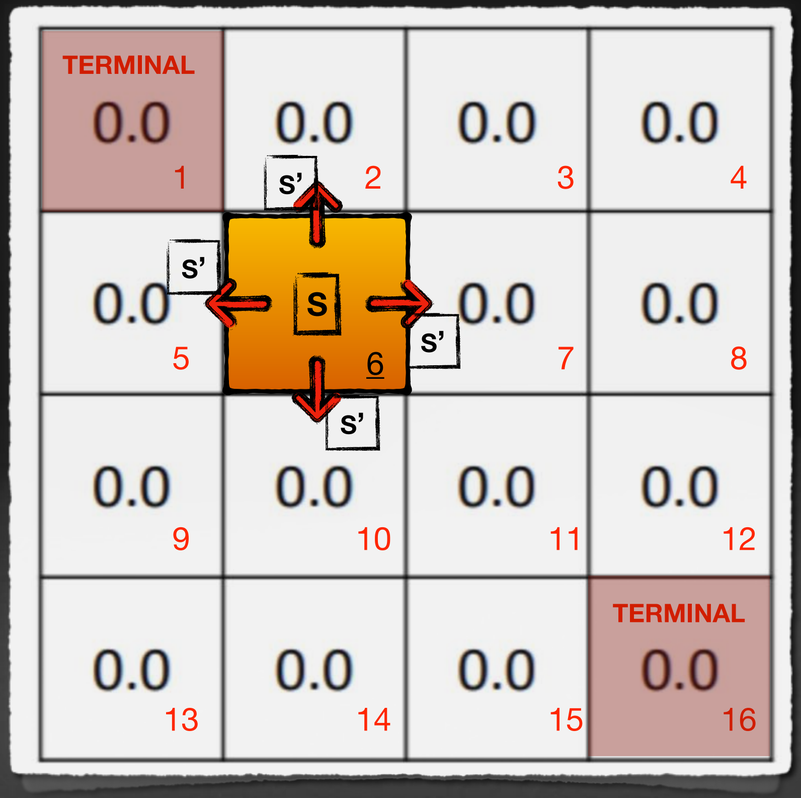
- $v_0(s')$ - **0 [we initialise all successor states with 0]**

  Discounted **value** of successor state calculated in **previous iteration**

  Here **previous iteration** means initialisation

Solution

- **Apply Iterative Bellman Expectation equation until convergence**



$$
\begin{aligned}
v_1(6) &= \sum_{a\in\{u,d,l,r\}} \pi(a|6) \sum_{s',r} p(s',r|6,a)[r+\gamma v_0(s')] \\
&= \sum_{a\in\{u,d,l,r\}} \underbrace{\pi(a|6)}_{=0.25\ \forall a} \sum_{s'} p(s'|6,a)[\underbrace{r}_{=-1} + \gamma \underbrace{v_0(s')}_{=0\ \forall s'}] \\
&= 0.25 * \{-p(2|6,u) - p(10|6,d) - p(5|6,l) - p(7|6,r)\} \\
&= 0.25 * \{-1-1-1-1\} \\
&= -1
\end{aligned}
$$

Iterative Bellman Expectation Equation for v_π

Policy Evaluation

First iteration output

1. After **first iteration** our **estimates** of state value functions **[** $v_1(s)$ **]**
2. has moved towards **TRUE** state value functions **[** $v_\pi(s)$ **]**
3. In order to have a proper estimate of **TRUE** value function we have to continue iteration process
   - $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$
   - Using Iterative Bellman Expectation equation

What we have after [k = 1]

- $v_1(s),\ \forall s \in S$

  Estimate of true value function for all states

What we need

- $v_\pi(s),\ \forall s \in S$

  True value function for all states

Next steps

- **Continue application of Iterative Bellman Expectation equation**
- **Introduce a stopping condition [** ϵ **]**

TERMINAL 0.0 1	-1.0 2	-1.0 3	-1.0 4
-1.0 5	-1.0 6	-1.0 7	-1.0 8
-1.0 9	-1.0 10	-1.0 11	-1.0 12
-1.0 13	-1.0 14	-1.0 15	TERMINAL 0.0 16

State value functions after first iteration

Policy Evaluation

Second iteration

$$v_2(2) = \sum_{a\in\{u,d,l,r\}} \underbrace{\pi(a|2)}_{=0.25\ \forall a} \sum_{s'} p(s'|2,a)[\underbrace{r}_{=-1} + \gamma \underbrace{v_1(s')}_{=\begin{cases}-1, s'\in S\\ 0, s'\in S^+\setminus S\end{cases}}]$$

$$= 0.25 * \{p(2|2,u)[-1-\gamma] + p(6|2,d)[-1-\gamma] + p(1|2,l)[-1-\gamma*0] + p(3|2,r)[-1-\gamma]\}$$

$$\overset{\gamma=1}{=} 0.25 * \{-2-2-1-2\}$$

$$= -1.75$$

Bellman Expectation Equation for state 2

TERMINAL 0.0 (1)	$v_2(2)$ (2)	-1.0 (3)	-1.0 (4)
-1.0 (5)	$v_2(6)$ (6)	-1.0 (7)	-1.0 (8)
-1.0 (9)	-1.0 (10)	-1.0 (11)	-1.0 (12)
-1.0 (13)	-1.0 (14)	-1.0 (15)	TERMINAL 0.0 (16)

$$v_2(6) = \sum_{a\in\{u,d,l,r\}} \underbrace{\pi(a|6)}_{=0.25\ \forall a} \sum_{s'} p(s'|6,a)[\underbrace{r}_{=-1} + \gamma \underbrace{v_1(s')}_{=\begin{cases}-1, s'\in S\\ 0, s'\in S^+\setminus S\end{cases}}]$$

$$= 0.25 * \{p(2|6,u)[-1-\gamma] + p(10|6,d)[-1-\gamma] + p(5|6,l)[-1-\gamma] + p(7|6,r)[-1-\gamma]\}$$

$$\overset{\gamma=1}{=} 0.25 * \{-2-2-2-2\}$$

$$= -2$$

Bellman Expectation Equation for state 6

TERMINAL 0.0 (1)	-1.7 (2)	-2.0 (3)	-2.0 (4)
-1.7 (5)	-2.0 (6)	-2.0 (7)	-2.0 (8)
-2.0 (9)	-2.0 (10)	-2.0 (11)	-1.7 (12)
-2.0 (13)	-2.0 (14)	-1.7 (15)	TERMINAL 0.0 (16)

Output from second iteration

How long should we iterate?

Let’s set stopping condition

Policy Evaluation
Stopping condition

1. Close **approximation** of **true** value function is sufficient to satisfy **Bellman Expectation** equation for MDP
2. We do not want to waste recourse on **infinite iteration processes**
3. Therefore we can set the iteration stopping condition

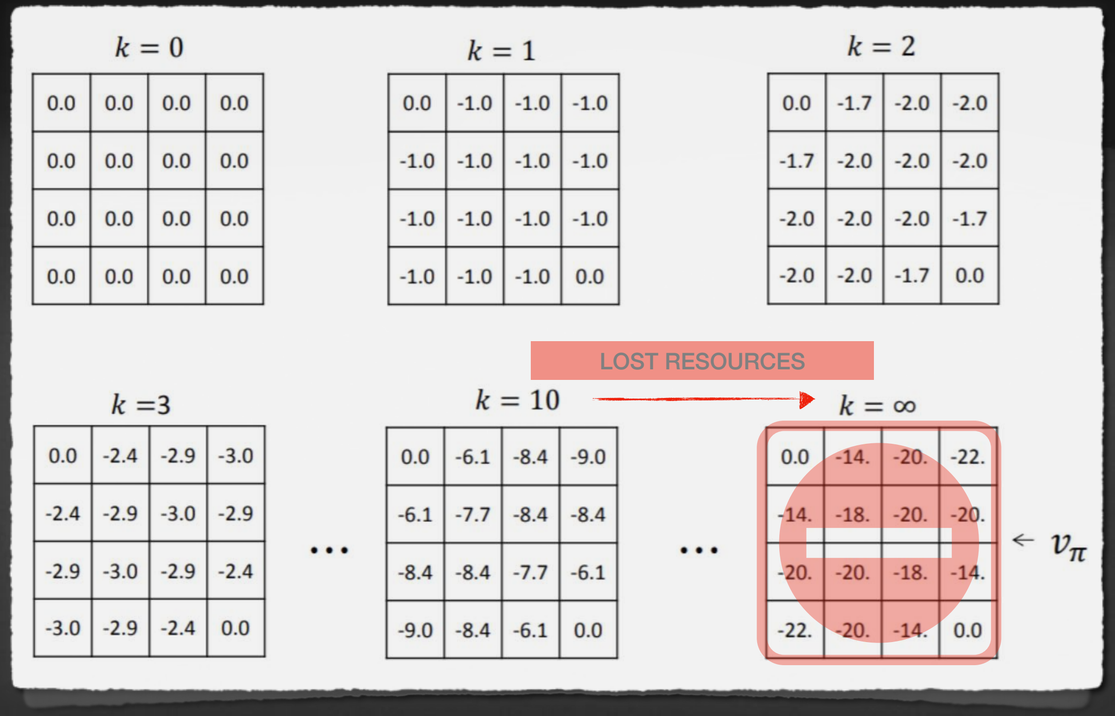
Let ϵ be some small value **[ϵ = 0.01]**

- If value **difference** between **current** iteration and **previous** iteration is **smaller then or equals** to **epsilon** for all states in the state space
- **Then stop the iteration process**

$$v_{k+1}(s) - v_k(s) \leq \epsilon,\ \forall s \in S$$

- We've found the **approximation** to **true value function**

$$v_{k+1}(s) \approx v_\pi(s), \forall s \in S$$



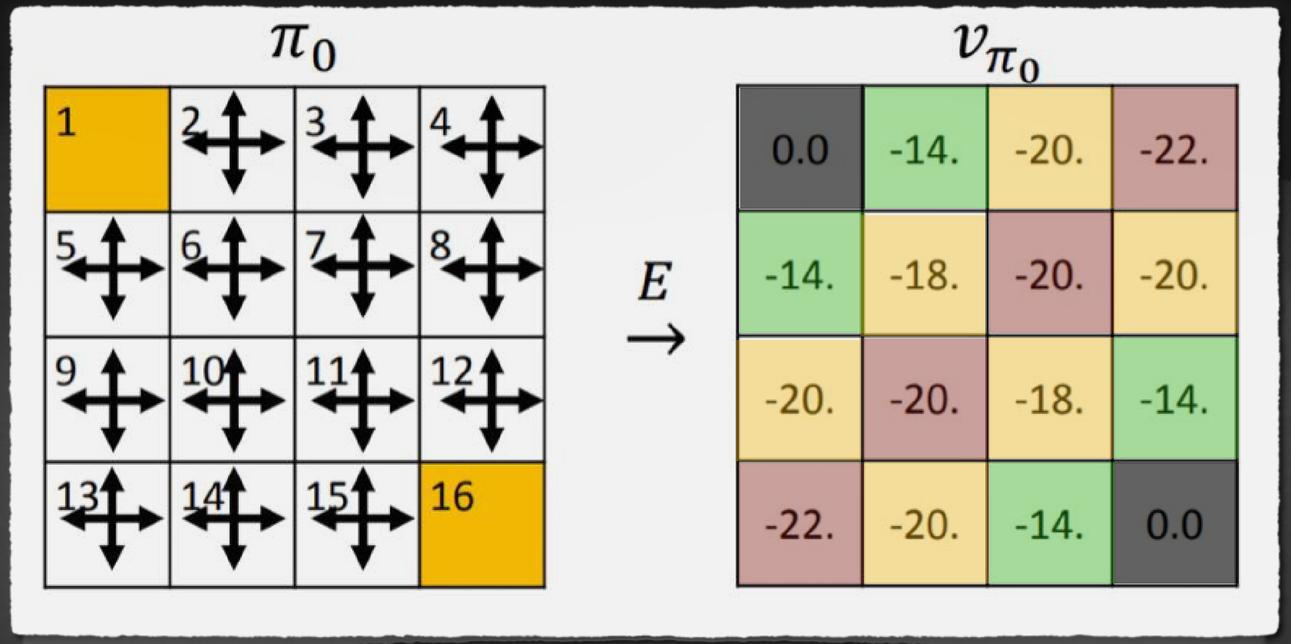
Iterative Policy Evaluation process

Once we have $v_\pi(s)$

We know how good is our policy w . r . t every state within the state space

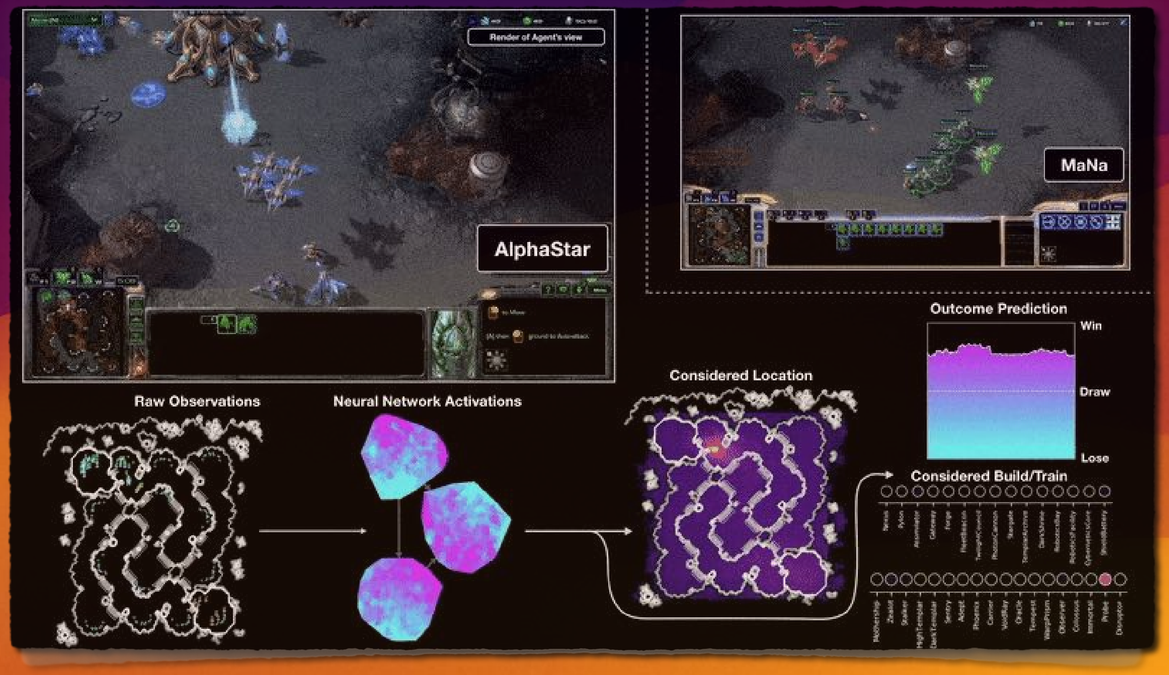
Policy Evaluation

Random Uniform Policy Evaluation Output



Generalised Policy Iteration

Improving our policy towards optimality



DeepMind, AlphaStar

The Issue

Random Uniform Policy

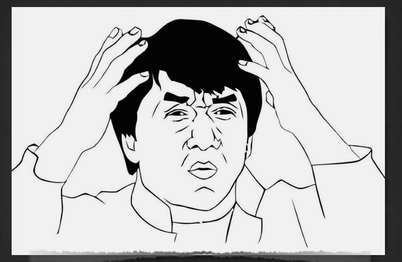
1. Uniform random policy **does not** seem like an **optimal** solution to our problem

2. For instance there is no point to consider other actions in state [15] besides **EAST** [e]

   - $\pi(e \mid 15) = 1$
   - $\pi(s \mid 15) = 0$
   - $\pi(n \mid 15) = 0$
   - $\pi(w \mid 15) = 0$

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

0.0 1	-14. 2	-20. 3	-22. 4
-14. 5	-18. 6	-20. 7	-20. 8
-20. 9	-20. 10	-18. 11	-14. 12
-22. 13	-20. 14	-14. 15	0.0 16



MAKES NO SENSE

Policy Improvement

How to improve policy

1. Given a policy [π]
2. and its evaluated **true value function** [$v_\pi(s)$]

Problem

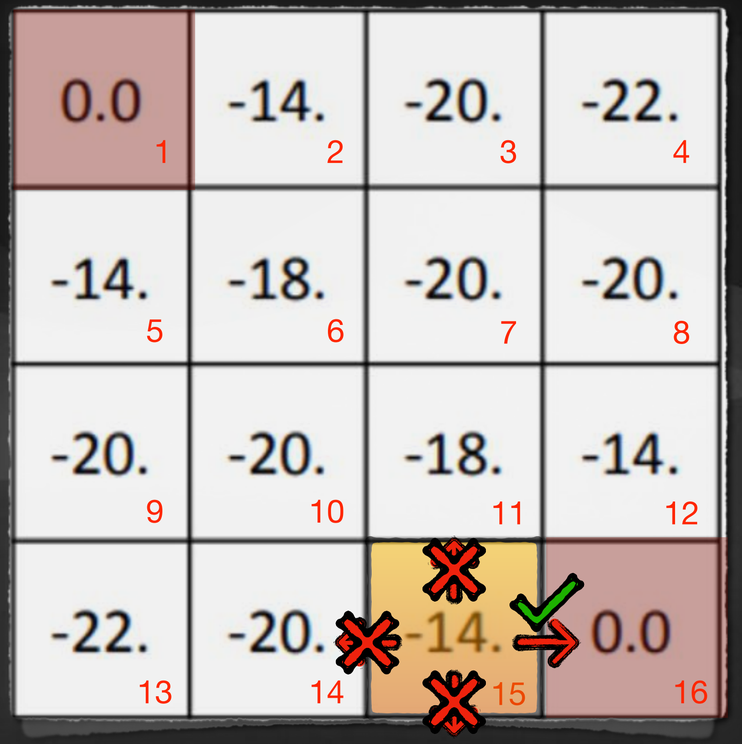
- Find the new better policy [π']

Solution

- improve the policy by acting **greedily** with respect to $v_\pi(s)$ for **every state** in the state space

$$\pi' = greedy(v_\pi)$$

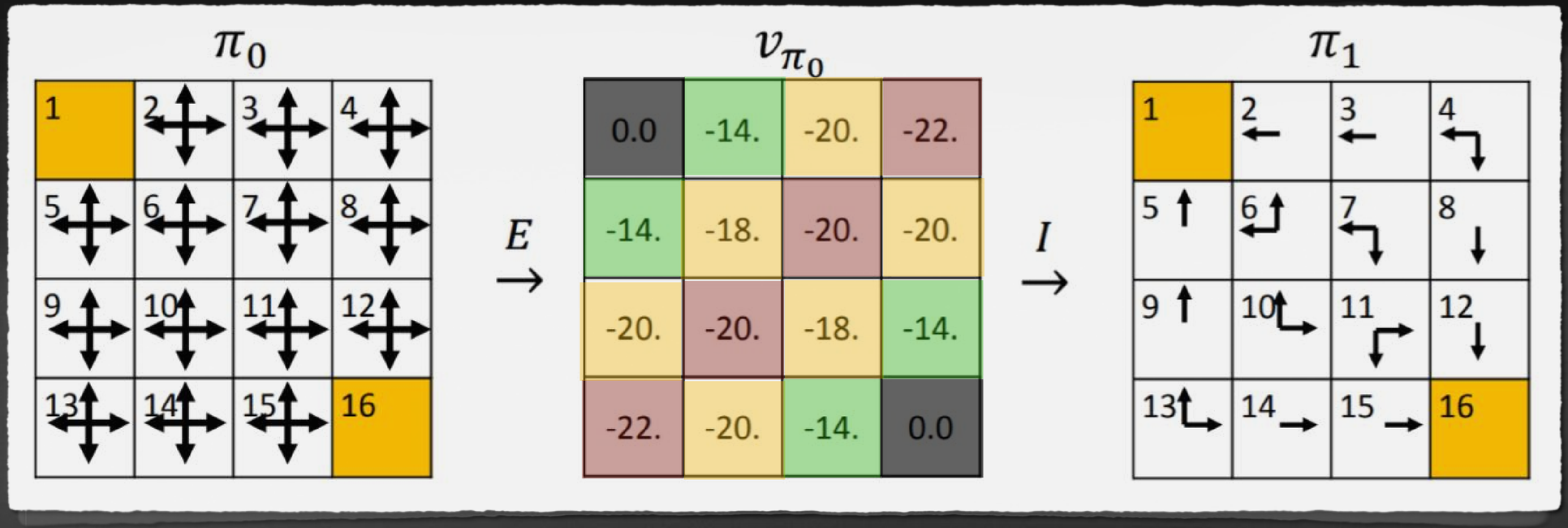
$$\pi'(s) = \arg\max_{a \in A}[R_s^a + P_{ss}^a v_\pi(s')]$$



New improved policy for state [15]

Improved Policy

New better policy



Policy Improvement Side Effect

Deprecated true value function

1. Since our new policy is produced [π']
2. And the **probabilities** of **actions** has changed in at least single state
   - $\pi(a \mid s) \neq \pi'(a \mid s), \exists s \in S,$
3. The **old true value function** is out dated
   - $v_{\pi}(s) \neq v_{\pi'}(s), \exists s \in S$
4. The **policy evaluation** step has to be **repeated** for the new policy [π']

Value function $v_{\pi'}$:

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

Value function v_{π}:

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

New vs old true value function

Policy Iteration Algorithm

Evaluate the true value function and then improve

Policy Iteration
Pushing towards optimality

1. Given policy evaluation algorithm to estimate v_π
2. And policy improvement algorithm to generate $\pi' \geq \pi$

Problem

- Find the optimal policy [π_*]

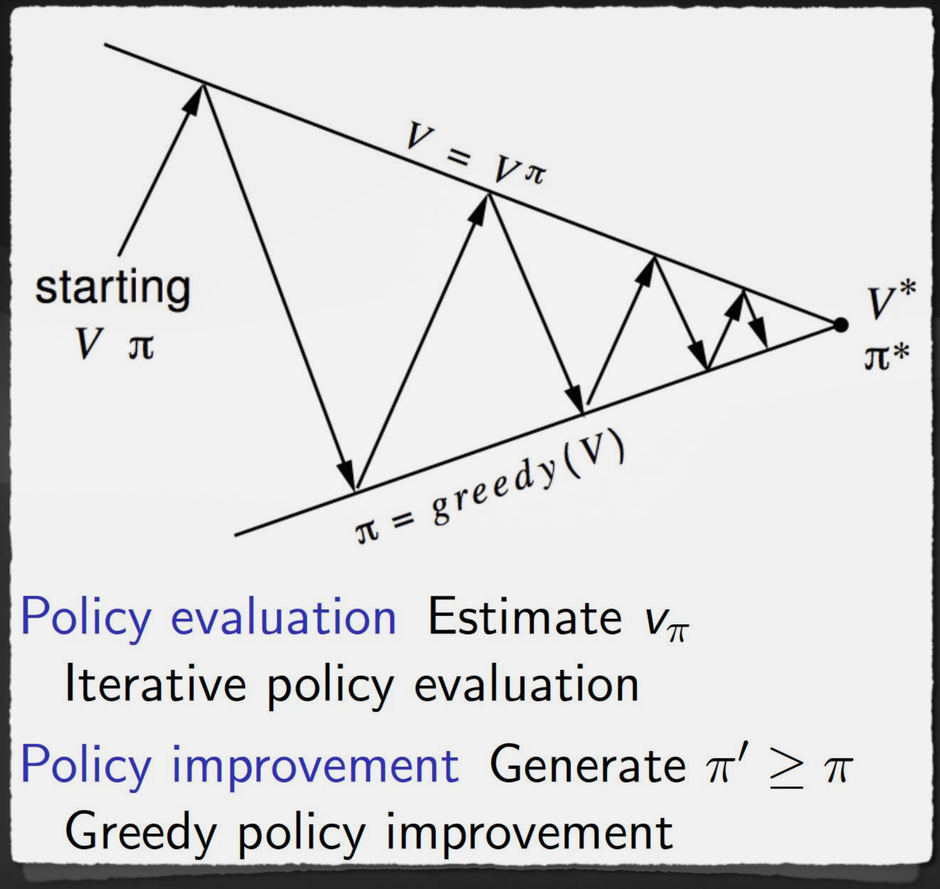
Solution

- Apply **policy evaluation** and **policy improvement** algorithms
- until the **produced policy** true value function values does not change compared to values of previous iteration

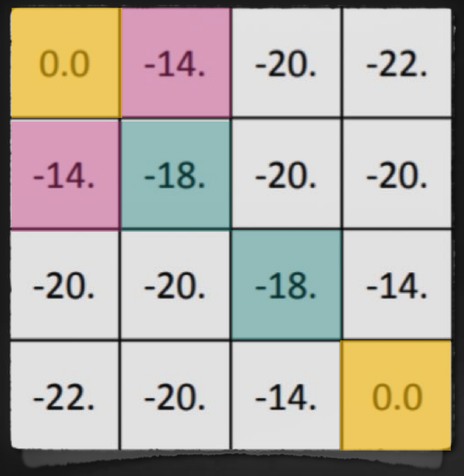
$$v_{\pi'}(s) \tilde{=} v_\pi(s), \forall s \in S$$

Then

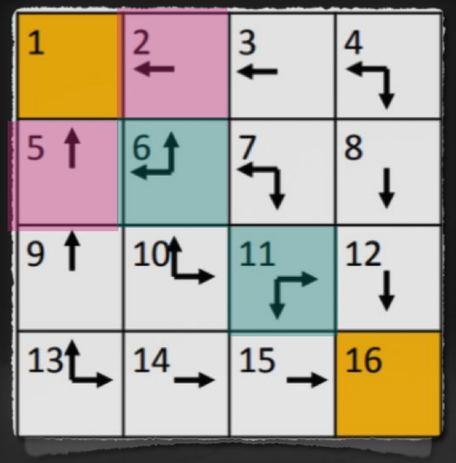
$$\pi', \pi = \pi_*$$



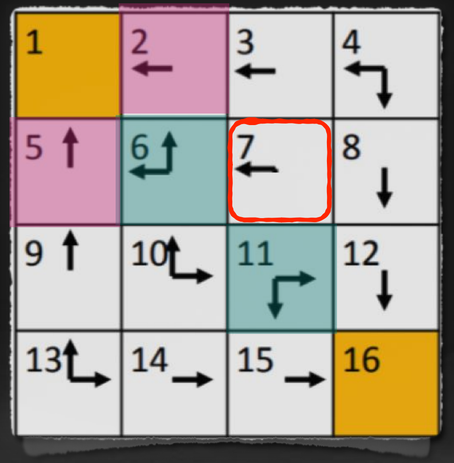
Policy Iteration Algorithm



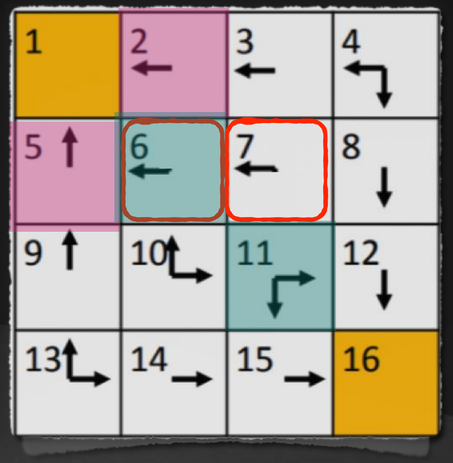
Optimal Value Function v_*



Optimal Policy π_{*_1}



Optimal Policy π_{*_2}



Optimal Policy π_{*_3}

We can have many optimal policies [π_*]

But only one optimal value function [v_*]

Value Iteration Algorithm

Do not evaluate true value function

Value Iteration

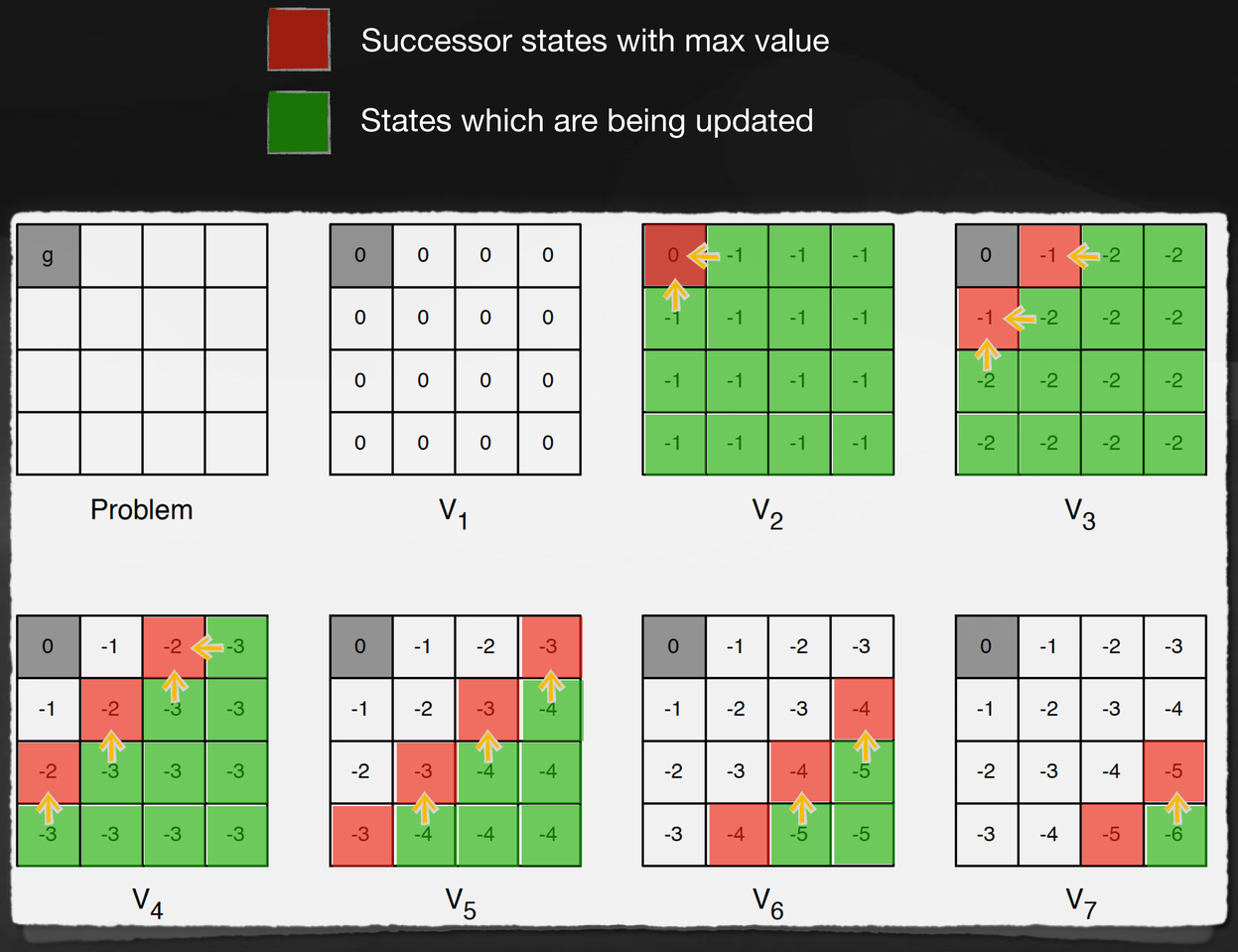
Extreme case of policy iteration

1. Unlike policy iteration there is **no explicit policy**
2. If we know the solution to subproblems $v_*(s')$
3. Then the solution $v_*(s)$ can be found by **one-step** lookahead
4. Using **Bellman Optimality Equation** instead of Bellman Expectation Equation

   - $v_*(s) \leftarrow \max_{a \in A} [R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')]$

   - $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_*$

5. **Intuition - start with the final rewards and work backwards**



Value Iteration Algorithm

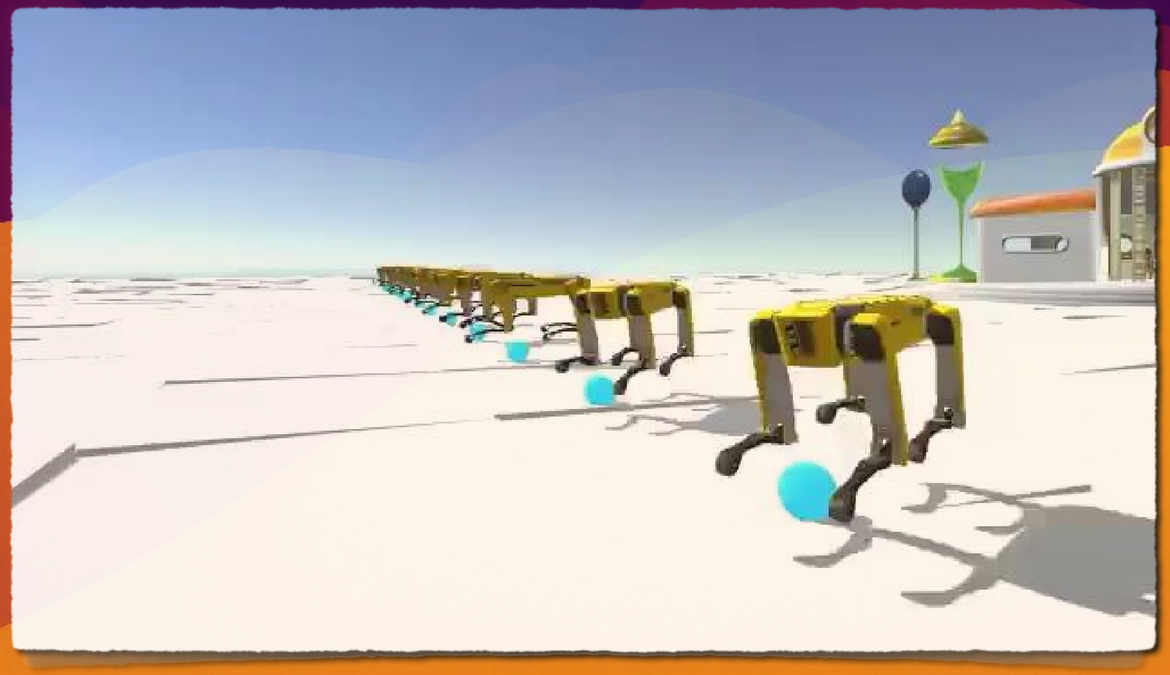
Policy Iteration and Value Iteration are dynamic programming methods,
therefore require full knowledge of an MDP

How can we get rid of that limitation ?

Model Free Methods

Model Free Prediction Methods

Evaluating policy directly from episodes of experience

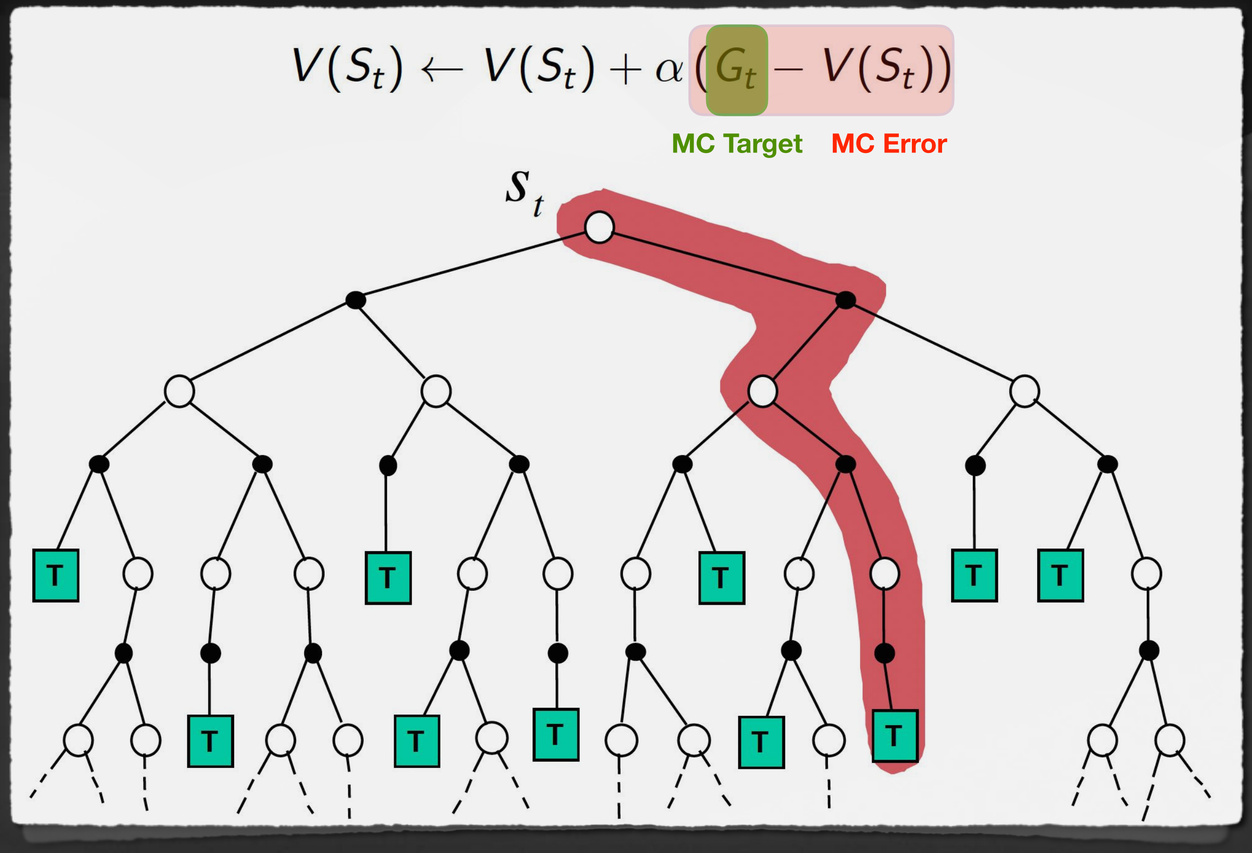


Boston Dynamics, Quadrupedal Robot Sim Training

Monte Carlo Learning

Model-Free Episodic Prediction

1. Monte Carlo **does not** need access to an MDP
   - MC is model-free
   - We sample our environment **[MDP]** in order to get **trajectories** [τ]
2. Monte Carlo is a **prediction** method
   - **Policy Evaluation** mechanism
3. Monte Carlo methods learn **directly** from **episodes** of **experience**
   - $estimate \leftarrow estimate \ + \ \alpha[\ actual\ return - estimate\ ]$
   - $estimate \leftarrow estimate \ + \ \alpha[\ MC\ Error\ ]$
4. Monte Carlo learns from **complete** episodes
   - **No bootstrapping**
   - can only be applied to **episodic problems**
   - All episodes must **terminate**
5. Monte Carlo uses **actual return** **[** G_t **]** as a target
   - It is the simplest possible idea
   - **Update value toward actual return [** G_t **]**

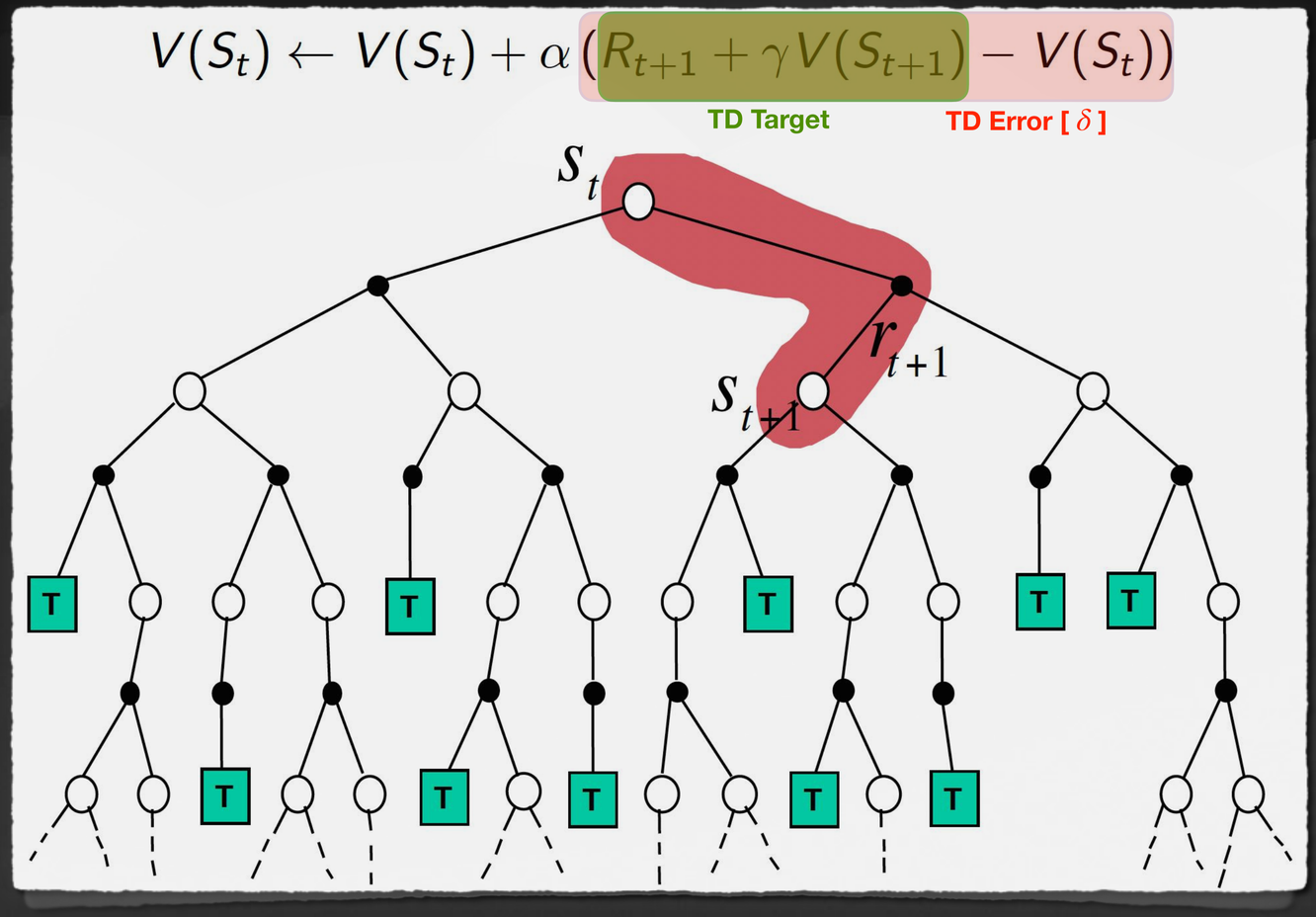


MC backup tree

Temporal Difference Learning

Model-Free Continuous Prediction

1. Temporal Difference methods **does not** need access to an MDP
   - TD is model-free
   - We sample our environment **[MDP]** in order to get **trajectories** [τ]
2. Temporal Difference is a **prediction** method
   - **Policy Evaluation** mechanism
3. Temporal Difference methods learn **directly** from **episodes** of **experience**
   - $estimate \leftarrow estimate \ + \ \alpha[\ estimated \ return - estimate \ ]$
   - $estimate \leftarrow estimate \ + \ \alpha[\ TD \ Error \ ]$
   - **One step TD is referred to as TD [0]**
4. Temporal Difference learns from **incomplete** episodes
   - **Uses bootstrapping**
   - Updates a guess towards a guess
5. Temporal Difference uses **estimated return** as a target
   - **Update value toward estimated return**



TD [0] backup tree

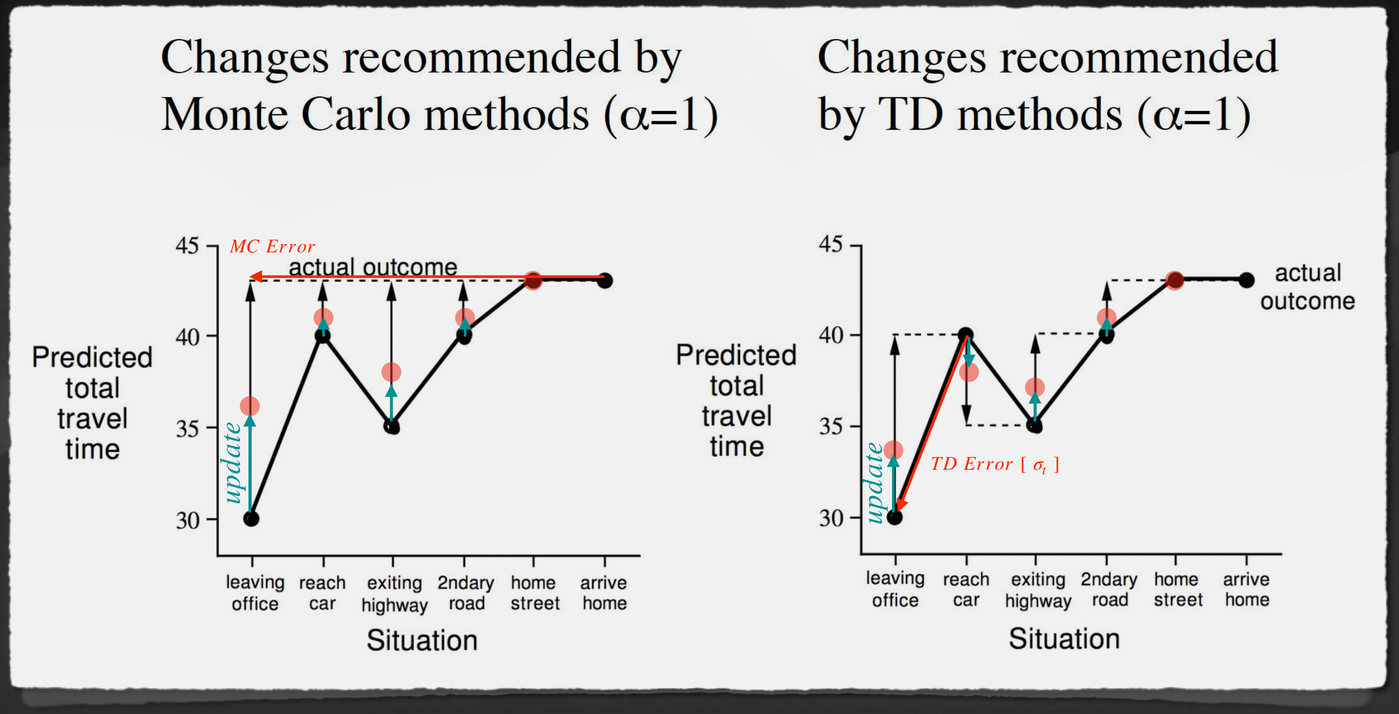
Monte Carlo vs Temporal Difference

Which one is better and how to choose ?

MC vs TD

Offline vs Online update

1. Learning **before** knowing the final outcome
    - TD is **online**
        Update estimates every step or every n-steps
    - MC is **offline**
        Wait until end of episode to update estimates
2. Learning **without** knowing 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
3. Bias and Variance
    - MC has **high variance**, **zero bias**
    - TD has **low variance**, **some bias**
4. **Convergence**
    - MC converges to solution with **minimum mean-squared error**
        Best fit to the observed returns
    - TD converges to the **maximum likelihood Markov model**
        Solution to the MDP that best fits the data

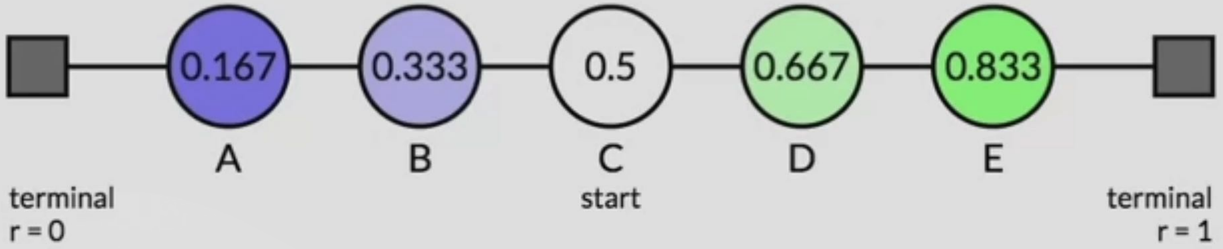


MC vs TD updates

MC vs TD [0]

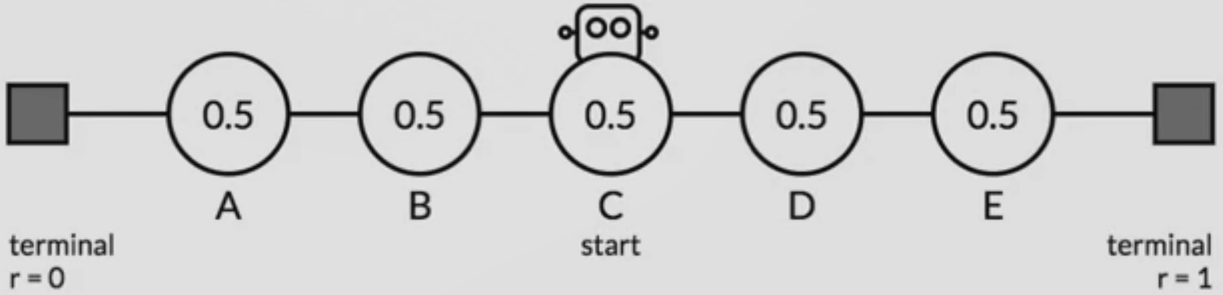
Let’s see it in action

Target / Exact Values

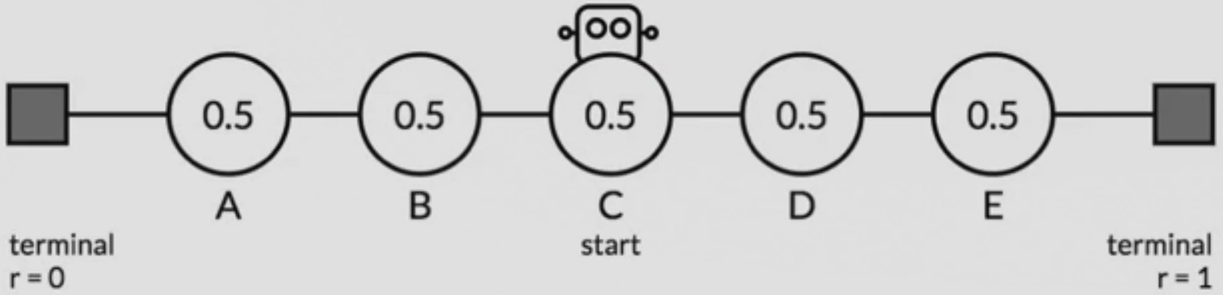


Updates using TD Learning

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



Updates using Monte Carlo



MC vs TD [0]

Which one to choose

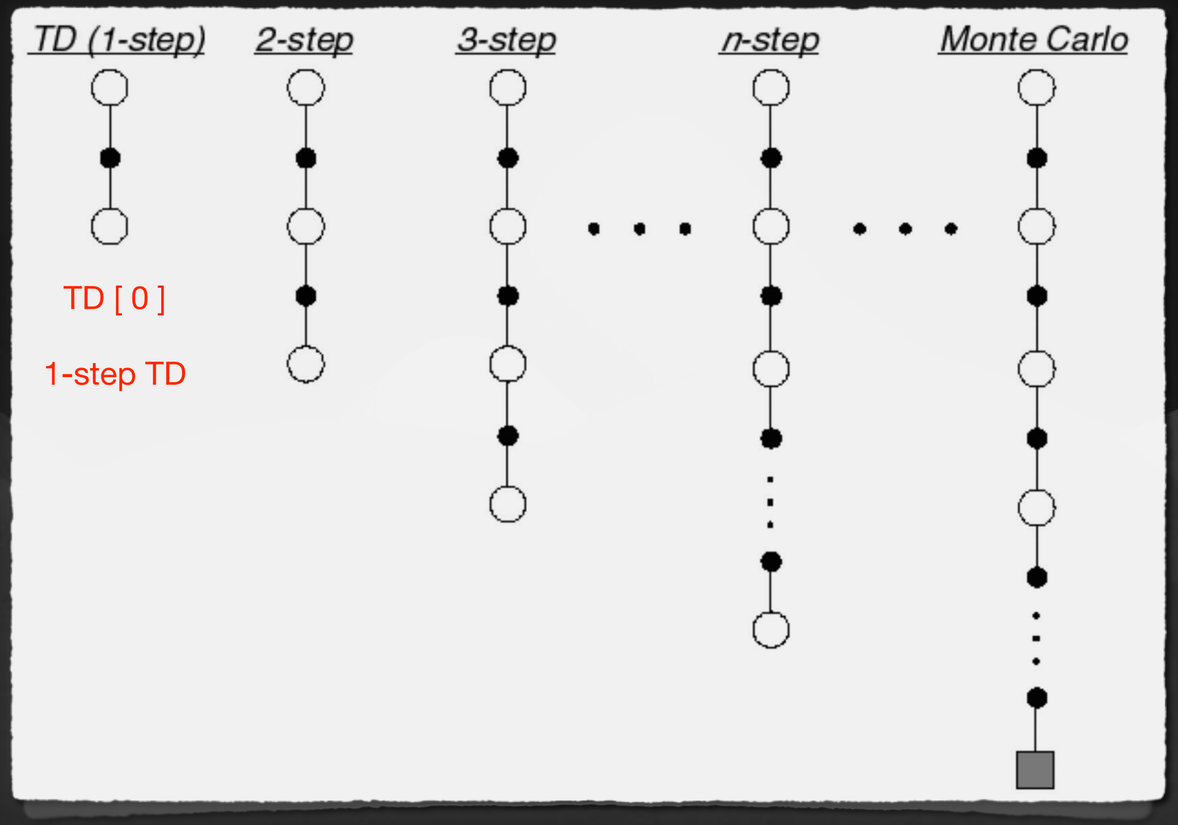
“ If I am to choose between one evil and another,
I’d rather not choose at all “

~ Geralt of Rivia, The Witcher

n-step TD

TD is not only about one step

1. **TD [0]** is designed to update our estimates after **one step [1-step]**
2. **But it is not a hard restriction**
3. Therefore can we define the **general n-step** temporal difference learning update
   - $V(S_t) \leftarrow V(S_t) + \alpha[G_t^{(n)} - V(S_t)]$
4. If we set our n-step return to **[∞]** we end up with **Monte Carlo**

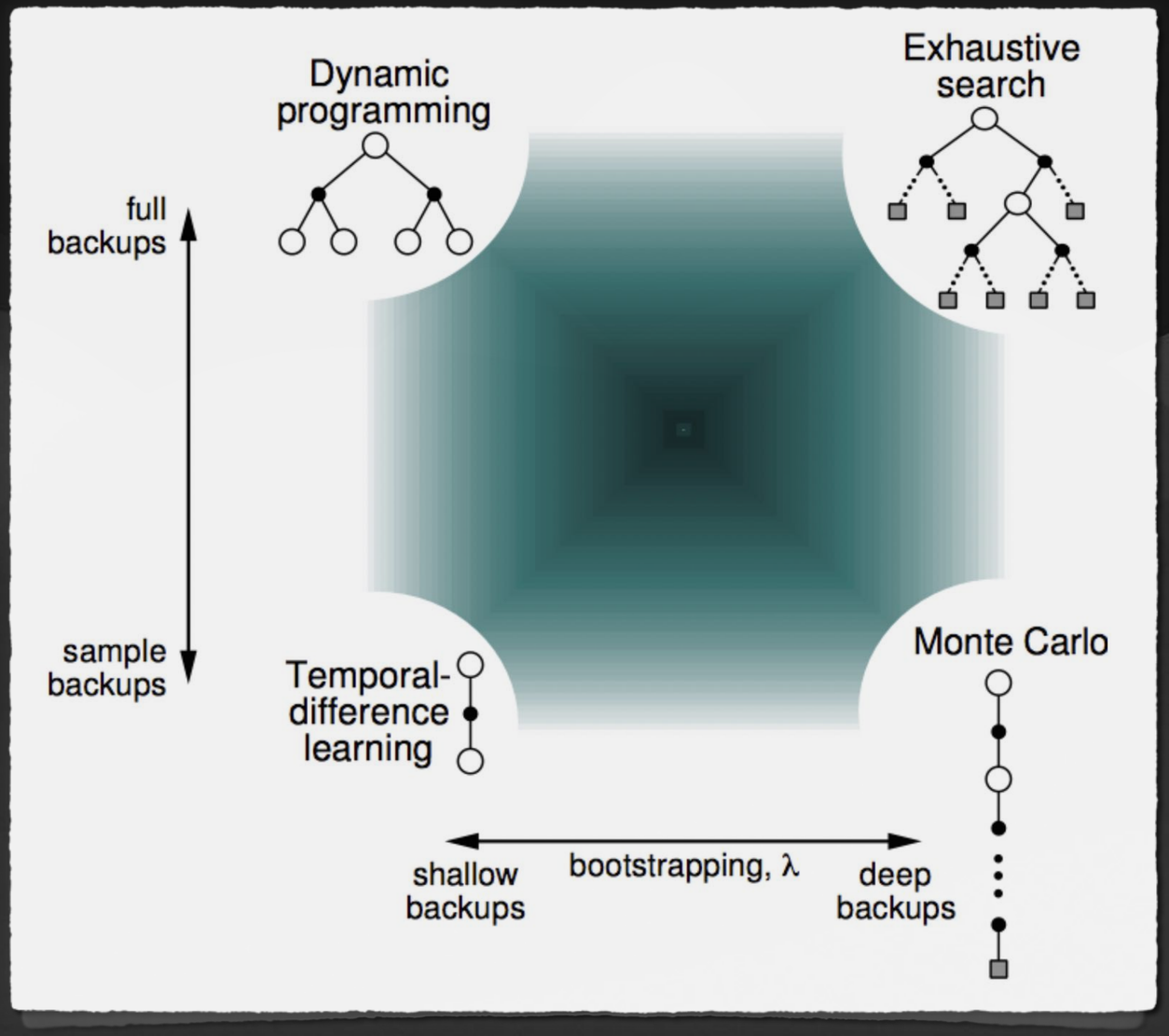


$$
\begin{aligned}
n = 1 \quad & (TD) \quad && G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\
n = 2 \quad & && G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\
\vdots \quad & && \vdots \\
n = \infty \quad & (MC) \quad && G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T
\end{aligned}
$$

General return equation

Unified View of RL

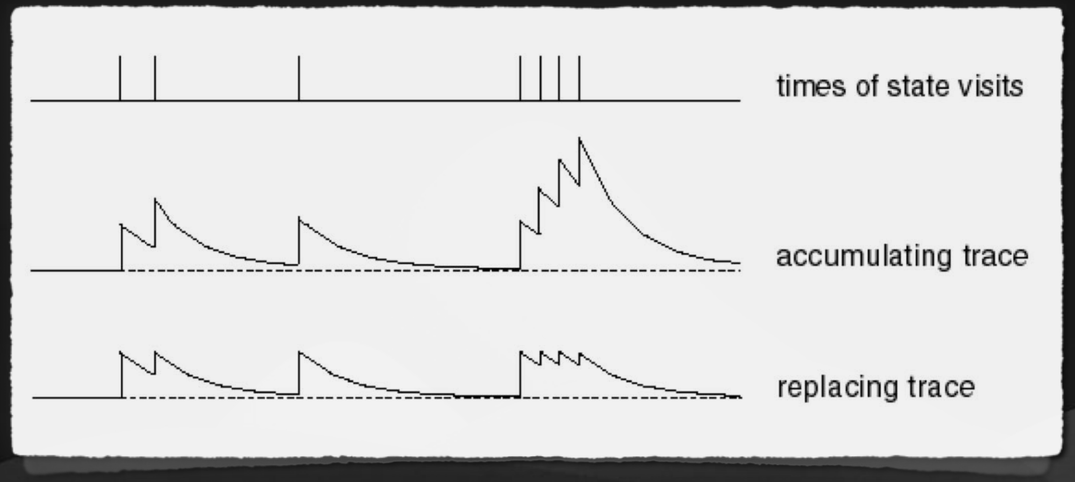
Tabular Methods State Space



Eligibility Traces

Frequency and Recency heuristics

1. **Frequency** heuristic
   - Assign credit to most **frequent** states
2. **Recency** heuristic
   - Assign credit to most **recent** states
3. **Eligibility traces combine both heuristics**
4. Uses **trace decay parameter** [λ]
   - Trace **d**ecay factor over time
   - $\lambda \in [0; 1]$
5. **Accumulating trace**
   - Every time state is visited **accumulate** eligibility coefficient
6. **Replacing trace**
   - Clip eligibility coefficient to a maximum value of one



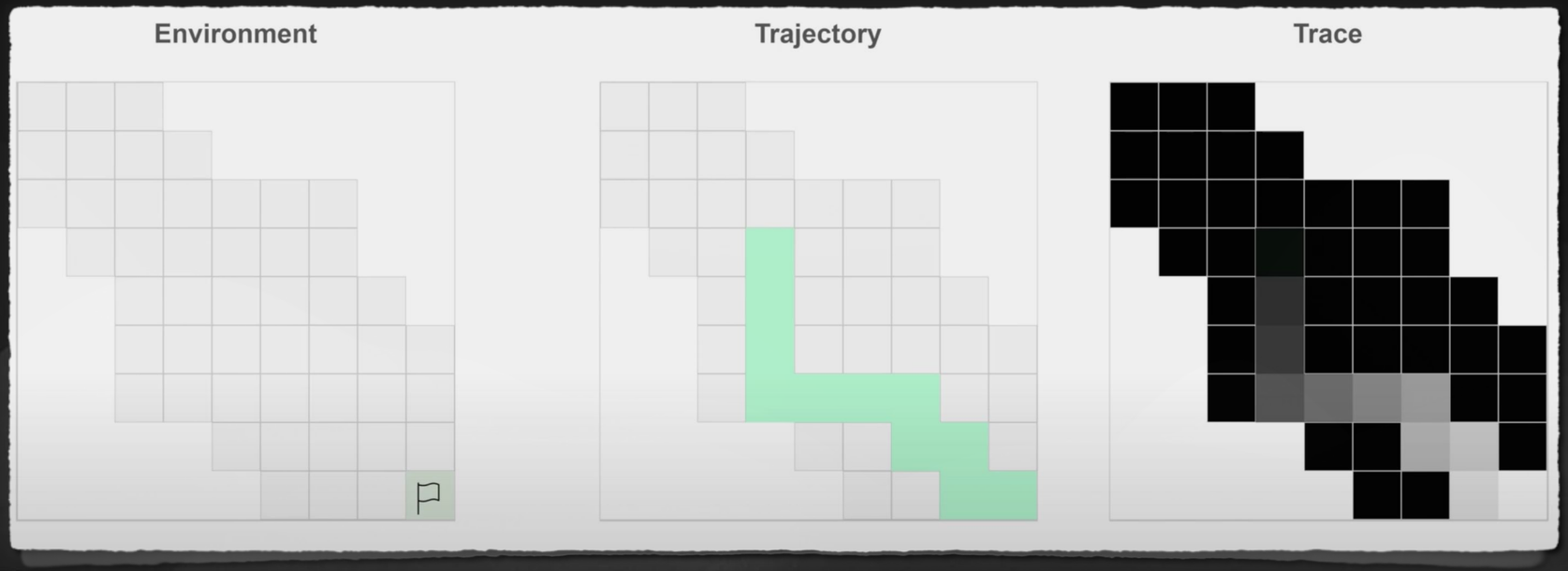
Eligibility coefficient throughout time

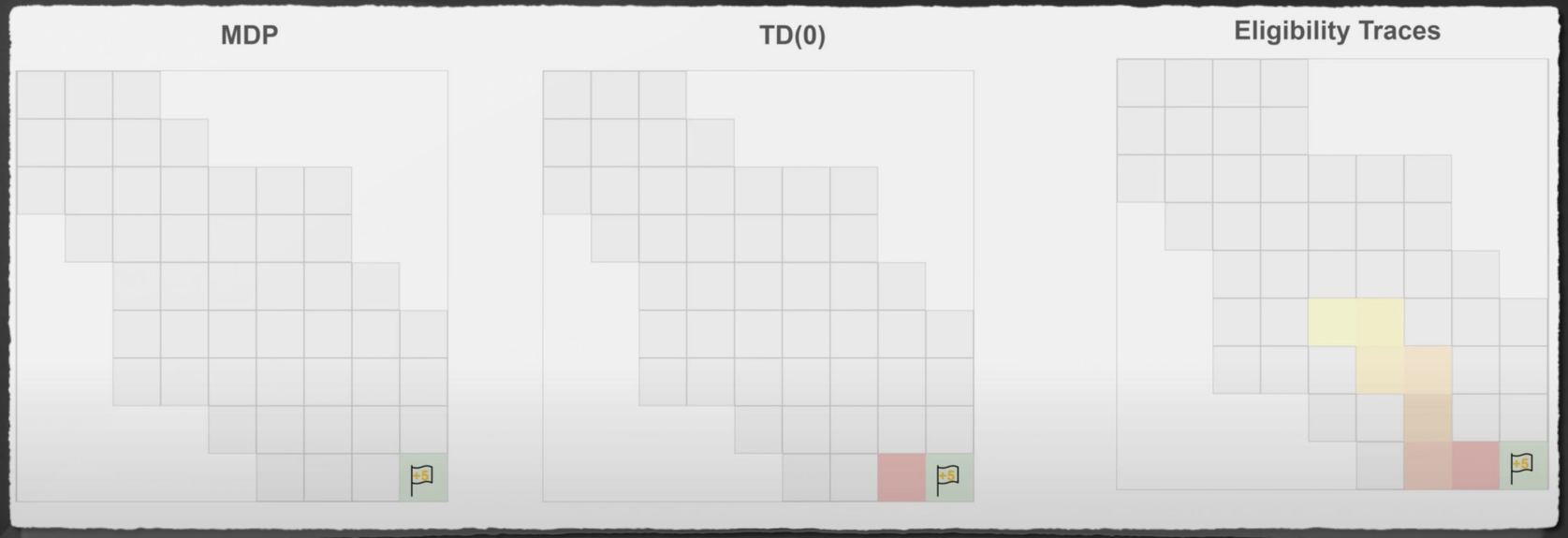
- **Accumulating Trace**

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t, \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$

- **Replacing Trace**

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t, \\ 1 & \text{if } s = s_t \end{cases}$$


Environment
Trajectory
Trace


MDP
TD(0)
Eligibility Traces

TD [λ]
The Power of Eligibility Traces

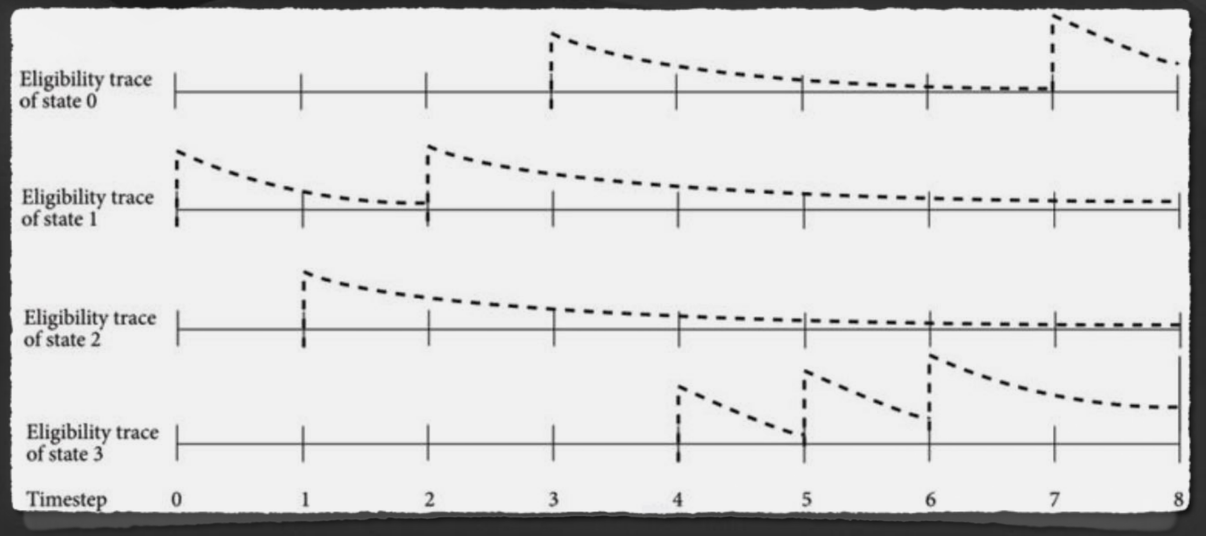
1. Consider combination of **Temporal Difference Learning** and **Eligibility Traces**
   - Given TD Error

     $$\delta_t = R_{t+1} + \gamma V_{t+1}(s) - V_t(s)$$

   - Update Value $V_t(s)$ for every state
   - In **proportion** to **TD Error** [δ_t] and **eligibility trace** $E_t(s)$

     $$\begin{cases} V_t(s) \leftarrow V_t(s) + \alpha\delta_t E_t(s) \\ E_0(s) = 0 \\ E_t(s) = \gamma\lambda E_{t-1}(s) + 1(S_t = s) \end{cases}$$

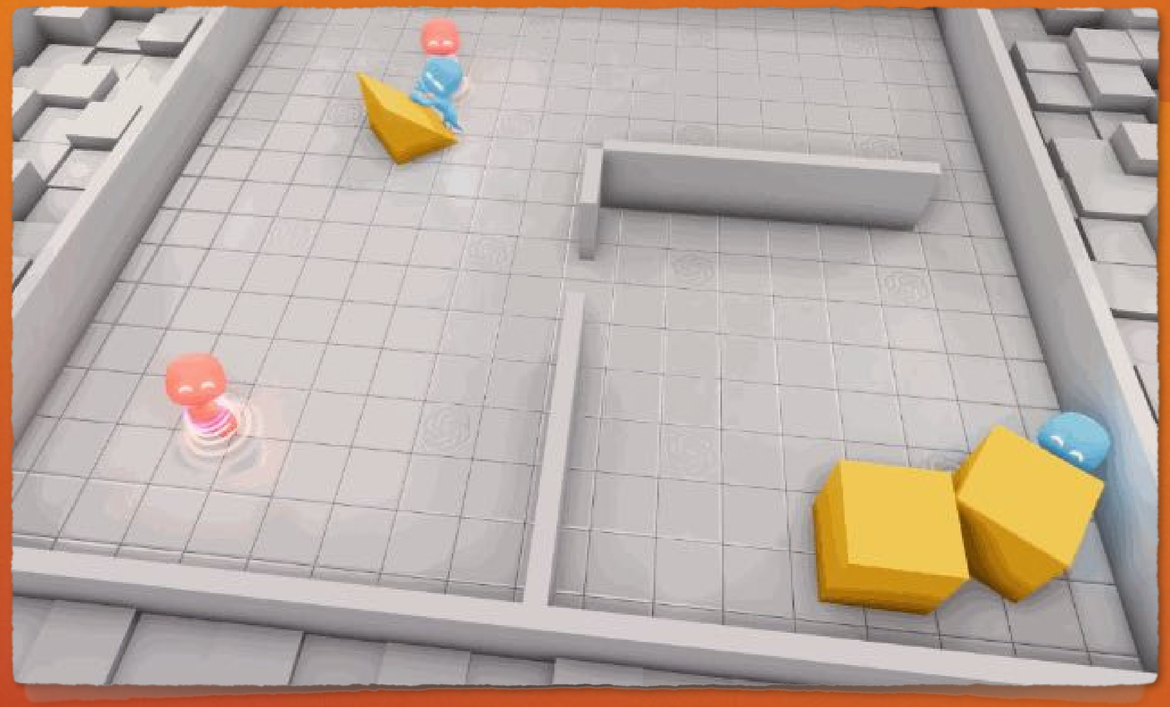
2. When eligibility decay parameter [λ] is set to **0**
   - **Do not use eligibility traces at all**
   - **Credit is given to current state**
   - This is exactly equivalent to **TD [0]**
3. When eligibility decay parameter [λ] is set to **1**
   - **Never decay eligibility trace vector**
   - **Credit is deferred until end of episode**
   - Over the course of an episode, total update is the same as total update for **Monte Carlo**



Eligibility trace vector

On-Policy vs Off-Policy

Learning from self mistakes or other’s mistakes



OpenAI, Multi-Agent Competition

On-Policy
Learn on the job

1. **Prefer own experience instead of others experience**

2. Learn about **policy [π]** from experience sampled from **policy [π]**

   - $\pi \leftarrow target\ policy$

     Policy we want to learn about

   - $\pi \leftarrow behaviour\ policy$

     Policy used to generate trajectories

3. **Intuition**: Use only experience generated by **latest policy**

Agent learning how to handstand from its own experience

Off-Policy

Look over someone's shoulder

1. **Learn from demonstrations of others**
2. Learn about **policy [π]** from experience sampled from **policy [μ]**
   - $\pi \leftarrow target\ policy$

     Policy we want to learn about
   - $\mu \leftarrow behaviour\ policy$

     Policy used to generate trajectories
3. **Intuition**: Experience from **older and out-dated policies** can be reused

Agent learning how to handstand from other's experience

Importance Sampling

Look over someone's shoulder

1. In order to use **behaviour's policy** [μ] experiences to estimate values for **target policy** [π]
2. We have to satisfy the Bellman Expectation criteria, namely
   - **<u>Every action under target policy</u>** has to be taken at least occasionally under behaviour policy [assumption of convergence]
   - Estimate expected values under **target policy** given **<u>samples from behaviour policy</u>**
3. Importance sampling **weights returns** according to the relative **probability** of the trajectories occurring under the target policy [π] and behaviour policy [μ]
4. **<u>Off-Policy MC Methods are useless since the probability distribution over trajectories are almost never correlated</u>**

$$\begin{aligned}\Pr\{A_t, S_{t+1}, A_{t+1}, \dots, S_T \mid S_t, A_{t:T-1} \sim \pi\} \\ = \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \\ = \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k),\end{aligned}$$

Probability of the subsequent trajectory given policy π

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}.$$

Importance sampling ratio

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}.$$

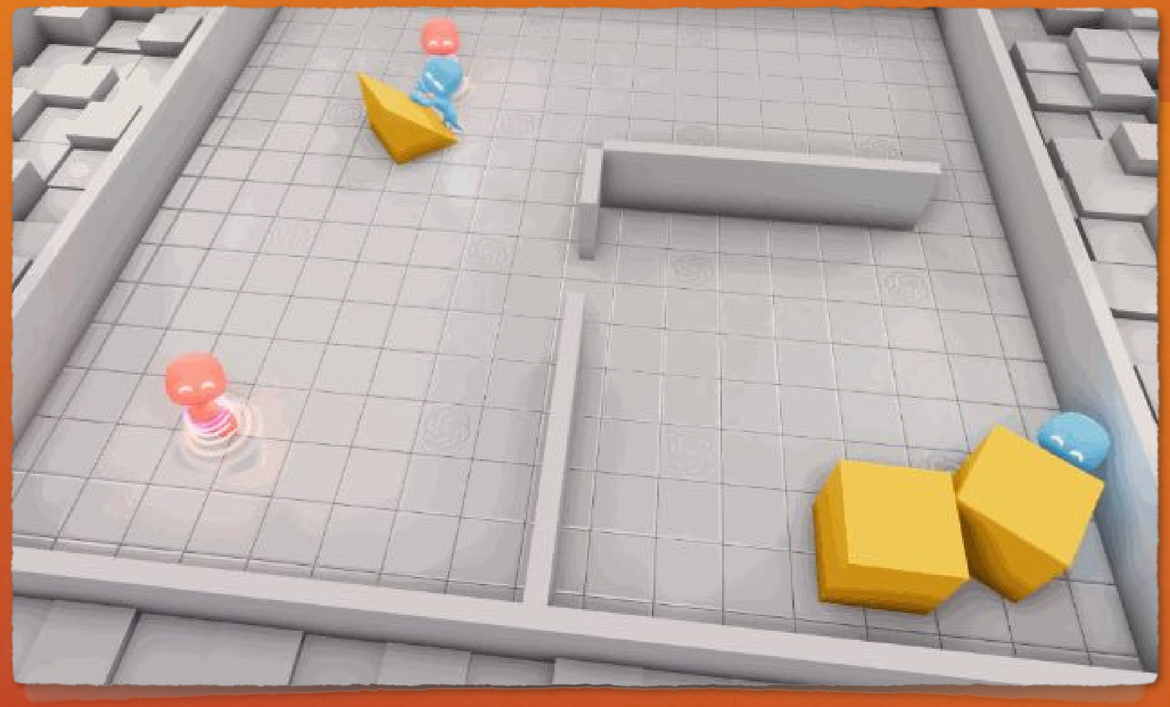
Ordinary importance sampling

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}},$$

Weighted importance sampling

Model Free Control Methods

Using experience to improve behaviour



OpenAI, Multi-Agent Competition

Model Free Control

GPI without MDP

1. Real world problems **<u>can not be directly</u>** modelled as an **MDP**
2. For most of those problems either
   - MDP is unknown, but experience can be sampled
     - **Robot walking**
     - **Protein Folding**
   - MDP is known, but is too big to use, except samples
     - **Game of Go**
     - **Aeroplane Logistics**
3. **<u>Model free control</u>** methods can solve those problems using only **sampled experience**

QRDQN solving Pacman

Model Free Control

Generalised Policy Iteration

1. Greedy policy improvement over **state value function** [$v(s)$] **requires model of MDP**

   - $\pi'(s) = \arg\max_{a \in A} [R_s^a + P_{ss'}^a v(s')]$ ✗

     Since we do not have access to an MDP we do not know the transition dynamics matrix [$P_{ss'}^a$]

   - Therefore we **can not** use it in **policy improvement** step

2. On the other hand the greedy policy improvement over **action value function** [$Q(s,a)$] **is model free**

   - $\pi'(s) = \arg\max_{a \in A} Q(s,a)$ ✓

     We just pick the action that maximises the Q value for current state

     We do not need the model to roll anything forward as opposed to V function

3. In order to ensure **continual exploration** we will consider **[$\epsilon - greedy$] policy improvement**

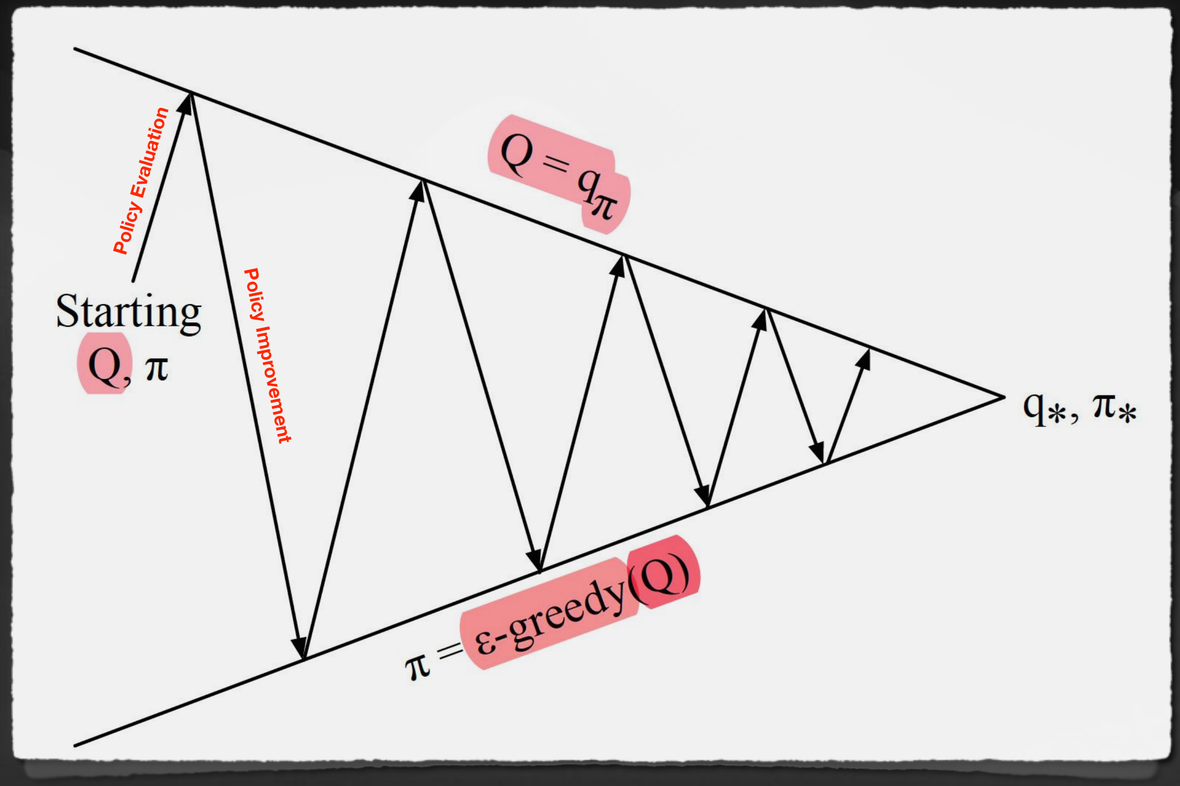
   - With probability $1 - \epsilon$ choose the **greedy** action

   - With probability ϵ choose an action at **random**

QRDQN solving Pacman

Model Free Control

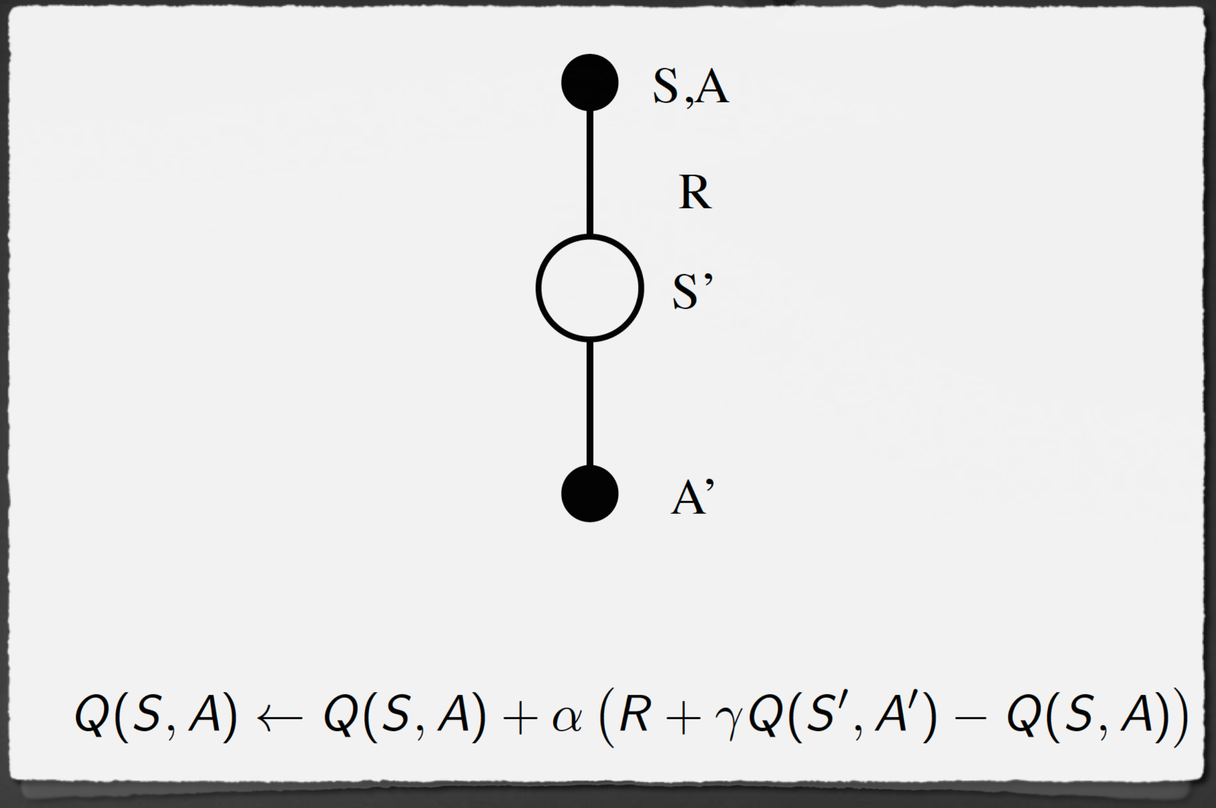
Generalised Policy Iteration with Action-Value function



SARSA

On-Policy Model Free TD Control

1. SARSA is built with respect to **Bellman Expectation Equation** for **action value function** [$q_\pi(s, a)$]
   - It takes the advantage of sampled backup from **Q-Policy Iteration**
2. TD [0] as the policy **evaluation step**
   - Update Q values after one step
3. $\epsilon - greedy$ **policy improvement** step
4. Since SARSA is On-Policy method it is a common practice to use eligibility traces instead of bare 1-step TD
   - $SARSA(\lambda)$



Action value function update in SARSA

Q-Learning

Off-Policy Model Free TD Control

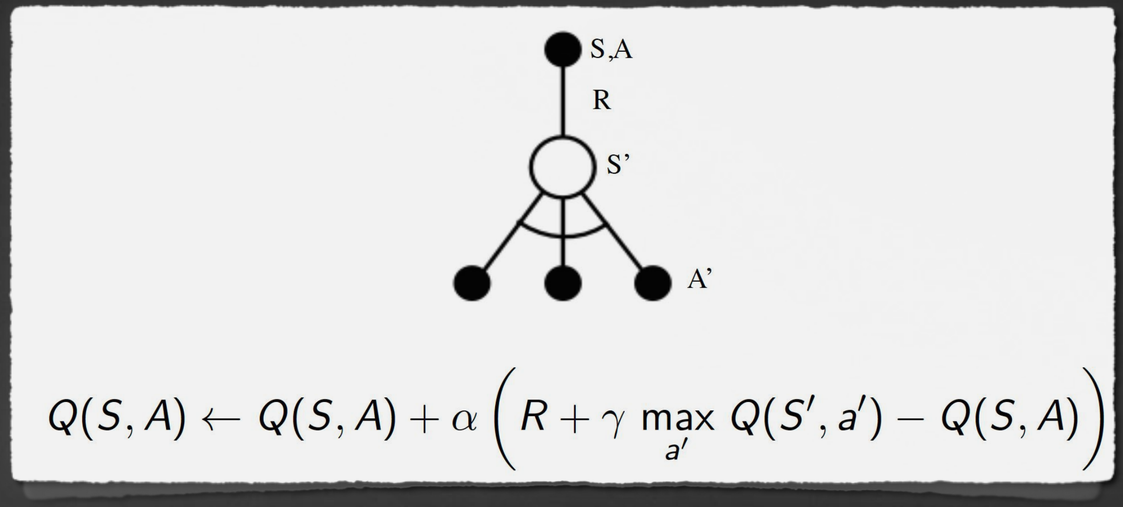
1. Q-Learning is built with respect to **Bellman Optimality Equation** for **action value function** [$q_*(s, a)$]
   - It takes the advantage of sampled backup from **Q-Value Iteration**
2. TD [0] as the policy evaluation step
   - Update Q values after one step
3. **No** importance sampling is required
   - Next action is chosen using **behaviour policy**

     $$A_{t+1} \sim \mu(\cdot \mid S_t)$$

   - But we consider **alternative successor action** $w \,.\, r \,.\, t$ **target policy**

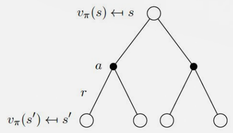
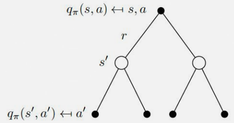
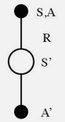
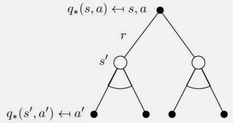
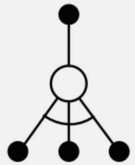
     $$A' \sim \pi(\cdot \mid S_t)$$

4. Therefore we can allow **both behaviour and target** policies to **improve**
   - Behaviour policy [μ] is $\epsilon - greedy\, w \,.\, r \,.\, t$ action value function
   - Target policy is $greedy\, w \,.\, r \,.\, t$ action value function



Action value function update in Q-Learning

Relationship between DP and TD

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $q_\pi(s,a)$	 Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s,a)$	 Q-Value Iteration	 Q-Learning

Bellman Equations

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$	$V(S) \overset{\alpha}{\leftarrow} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma Q(S',A') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma Q(S',A')$
Q-Value Iteration	Q-Learning
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \overset{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

where $x \overset{\alpha}{\leftarrow} y \equiv x \leftarrow x + \alpha(y - x)$

Generalised Policy Iteration

Thank You

Appendix