Reinforcement Learning

Emma Brunskill
Stanford University

Winter 2019

Midterm Review

Reinforcement Learning Involves

- Optimization
- Delayed consequences
- Generalization
- Exploration

Learning Objectives

- Define the key features of reinforcement learning that distinguishes it from AI and non-interactive machine learning (as assessed by exams).
- Given an application problem (e.g. from computer vision, robotics, etc), decide if it should be formulated as a RL problem; if yes be able to define it formally (in terms of the state space, action space, dynamics and reward model), state what algorithm (from class) is best suited for addressing it and justify your answer (as assessed by the project and exams).
- Implement in code common RL algorithms such as a deep RL algorithm, including imitation learning (as assessed by the homeworks).
- Describe (list and define) multiple criteria for analyzing RL algorithms and evaluate algorithms on these metrics: e.g. regret, sample complexity, computational complexity, empirical performance, convergence, etc (as assessed by homeworks and exams).
- Describe the exploration vs exploitation challenge and compare and contrast at least two approaches for addressing this challenge (in terms of performance, scalability, complexity of implementation, and theoretical guarantees) (as assessed by an assignment and exams).

Learning Objectives

- Define the key features of reinforcement learning that distinguishes it from AI and non-interactive machine learning (as assessed by exams).

- Given an application problem (e.g. from computer vision, robotics, etc), decide if it should be formulated as a RL problem; if yes be able to define it formally (in terms of the state space, action space, dynamics and reward model), state what algorithm (from class) is best suited for addressing it and justify your answer (as assessed by the project and exams).

- Describe (list and define) multiple criteria for analyzing RL algorithms and evaluate algorithms on these metrics: e.g. regret, sample complexity, computational complexity, empirical performance, convergence, etc (as assessed by homeworks and exams).

What We’ve Covered So Far

- Markov decision process planning
- Model free policy evaluation
- Model-free learning to make good decisions
- Value function approximation, focus on model-free methods
- Imitation learning
- Policy search

Reinforcement Learning

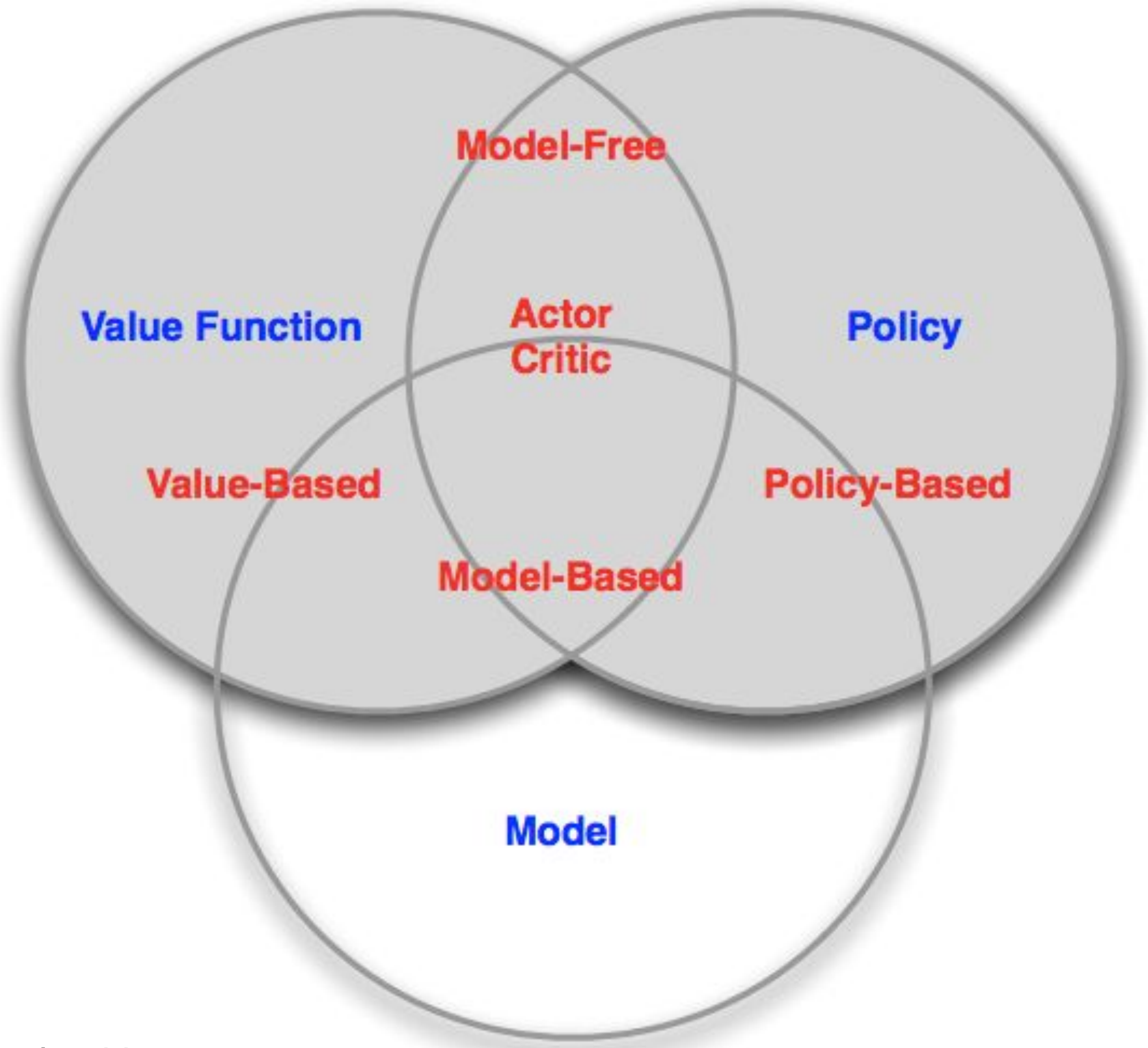


Figure from David Silver’s slides

Reinforcement Learning

model $\longrightarrow$ value $\longrightarrow$ policy

(ordering sufficient but not necessary, e.g. having a model is not required to learn a value)

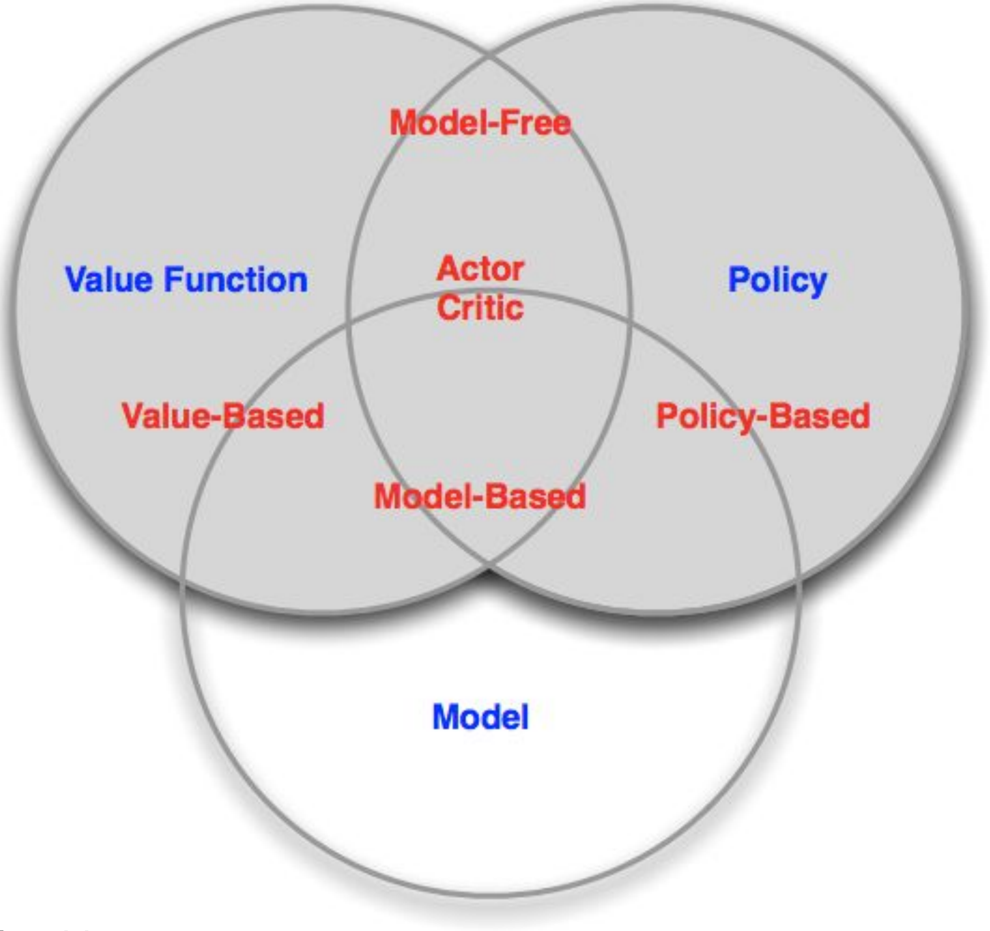
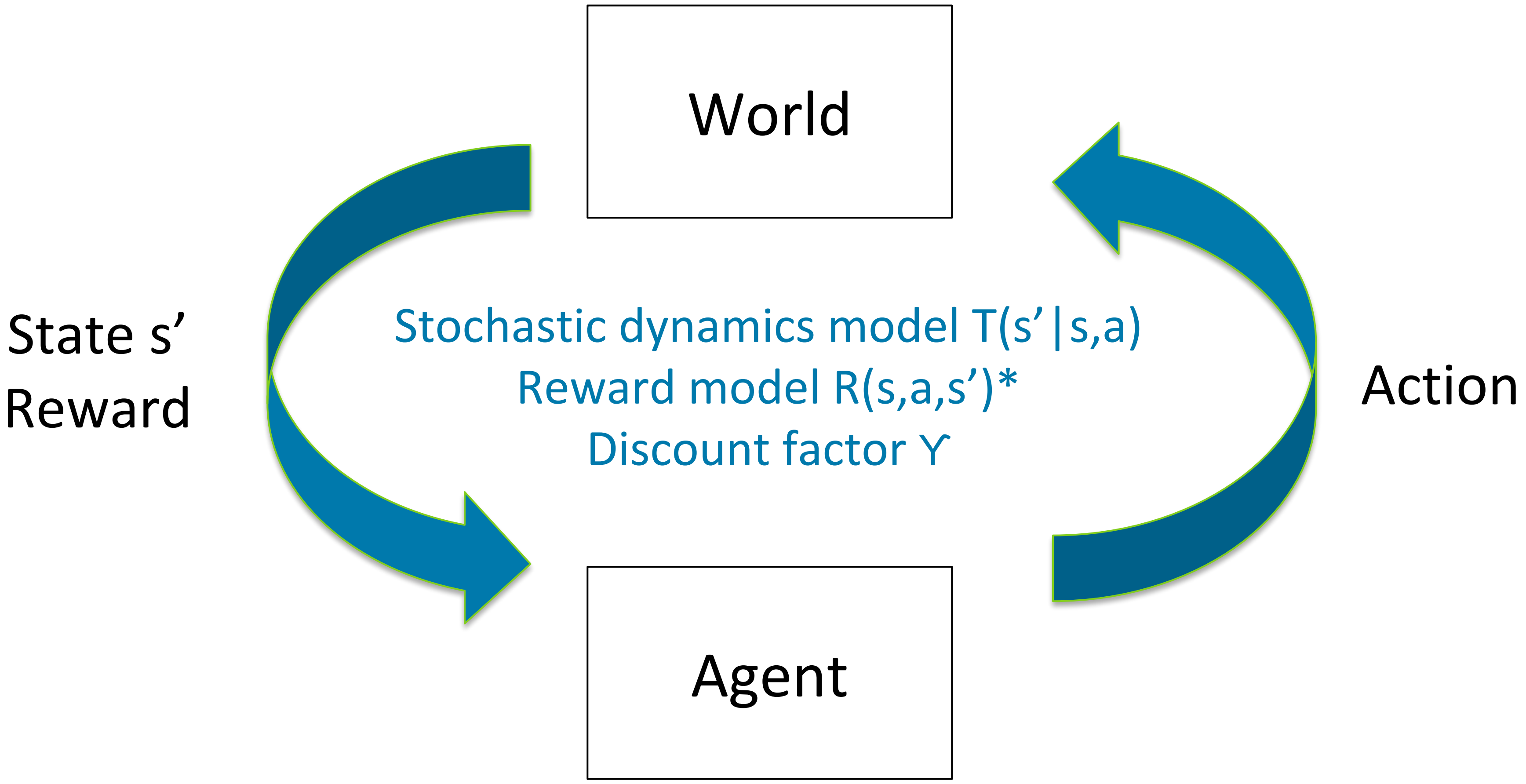


Figure from David Silver's slides

What We’ve Covered So Far

- Markov decision process planning
- Model free policy evaluation
- Model-free learning to make good decisions
- Value function approximation, focus on model-free methods
- Imitation learning
- Policy search

Model: Frequently model as a Markov Decision Process, <S,A,R,T,ϒ>



Policy mapping from state → action

MDPs

- Define a MDP <S,A,R,T,γ>
- Markov property
  - What is this, why is it important
- What are the MDP models / values V / state-action values Q / policy
- What is MDP planning? What is difference from reinforcement learning?
  - Planning = know the reward & dynamics
  - Learning = don’t know reward & dynamics

Bellman Backup Operator

$$V_{k+1}(s) = \max_a \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|a,s) V_k(s') \right]$$

Bellman backup

- Bellman backup is a contraction if discount factor, γ < 1
- Bellman contraction operator: with repeated applications, guaranteed to converge to a single fixed point (the optimal value)

Value vs Policy Iteration

- Value iteration:
  - Compute optimal value if horizon=k
    - Note this can be used to compute optimal policy if horizon = k
  - Increment k
- Policy iteration:
  - Compute infinite horizon value of a policy
  - Use to select another (better) policy
  - Closely related to a very popular method in RL: policy gradient

Policy Iteration (PI)

1. i=0; Initialize $\pi_0(s)$ randomly for all states s
2. Converged = 0;
3. While i == 0 or $|\pi_i - \pi_{i-1}| > 0$
   - i=i+1
   - Policy evaluation: Compute V^π
   - Policy improvement:

$$Q^{\pi_i}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_i}(s')$$

$$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$$

Check Your Understanding

Consider finite state and action MDP and use a lookup table representation, $\gamma < 1$, infinite H:

- Does the initial setting of the value function in value iteration impact the final computed value? Why/why not?

- Do value iteration and policy iteration always yield the same solution?

- Is the number of iterations needed for PI on a tabular MDP with $|A|$ actions and $|S|$ states bounded?

What We've Covered So Far

- Markov decision process planning
- **Model free policy evaluation**
- Model-free learning to make good decisions
- Value function approximation, focus on model-free methods
- Imitation learning
- Policy search

Model-free Passive RL

- Directly estimate Q or V of a policy from data
- The Q function for a particular policy is the **expected discounted sum of future rewards** obtained by following policy starting with (s,a)
- For Markov decision processes,

$$Q^{\pi_i}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_i}(s')$$

Model-free Passive RL

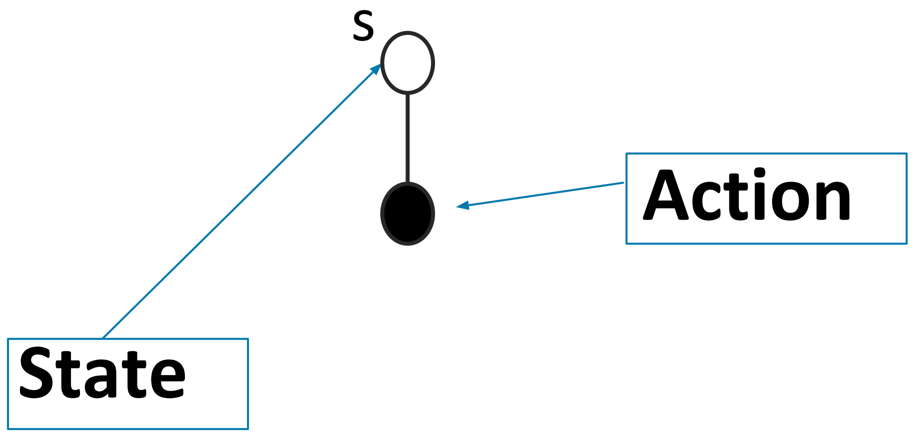
- Directly estimate Q or V of a policy from data
- The Q function for a particular policy is the **expected discounted sum of future rewards** obtained by following policy starting with (s,a)
- For Markov decision processes,

$$Q^{\pi_i}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_i}(s')$$

- Consider episodic domains
  - Act in world for H steps, then reset back to state sampled from starting distribution
- MC: directly average episodic rewards
- TD/Q-learning: use a “target” to bootstrap

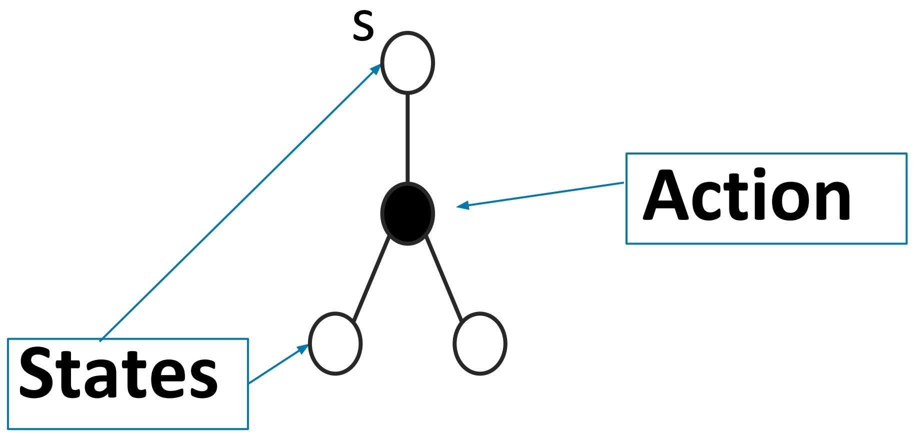
Dynamic Programming Policy Evaluation

$$V^{\pi}(s) \leftarrow \mathbb{E}_{\pi}[r_t + \gamma V_{i-1} | s_t = s]$$



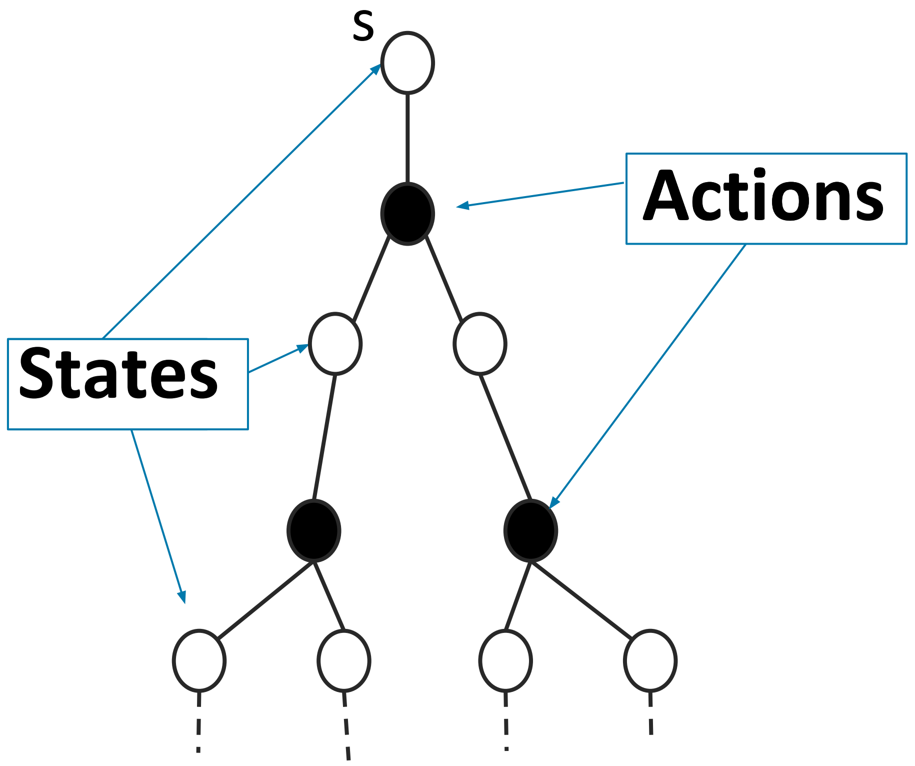
Dynamic Programming Policy Evaluation

$$V^{\pi}(s) \leftarrow \mathbb{E}_{\pi}[r_t + \gamma V_{i-1} | s_t = s]$$



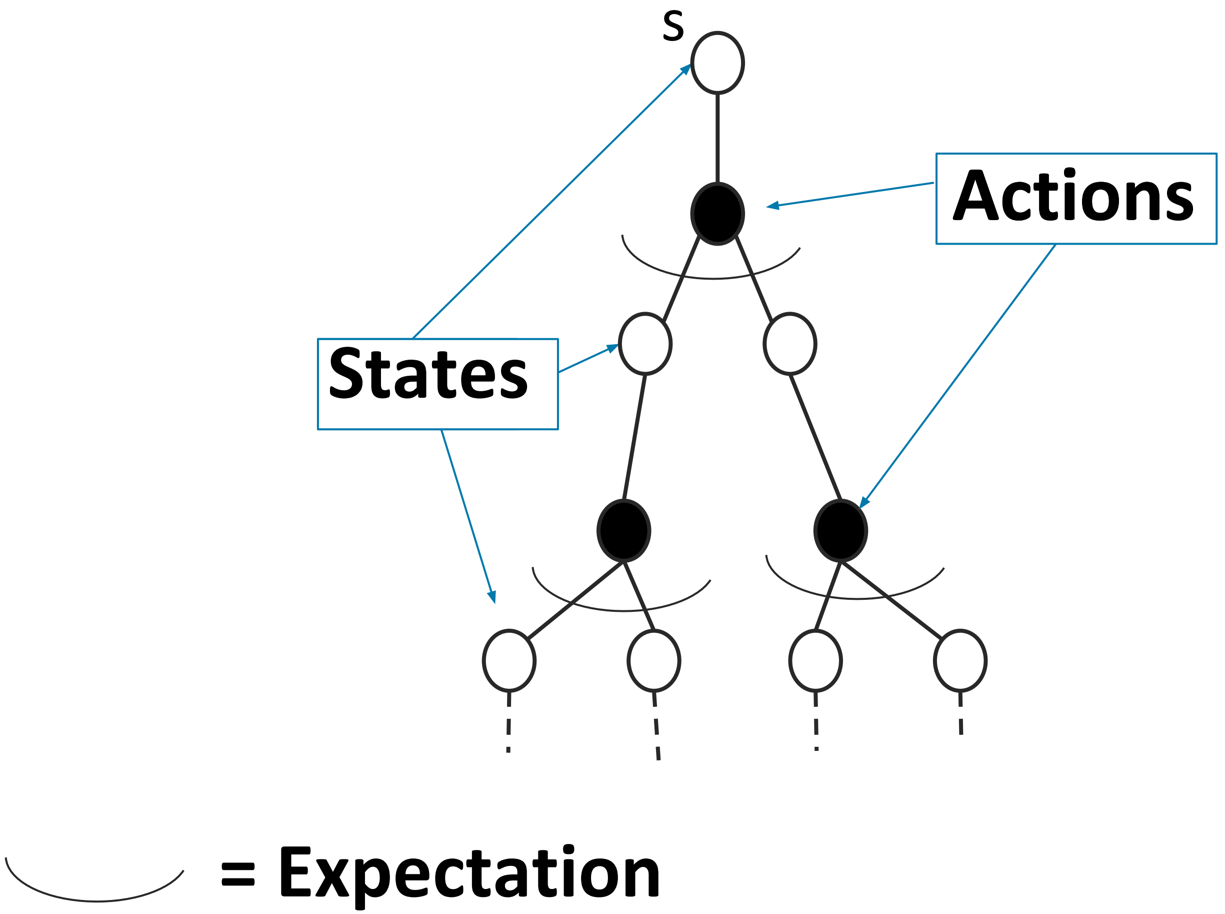
Dynamic Programming Policy Evaluation

$$V^{\pi}(s) \leftarrow \mathbb{E}_{\pi}[r_t + \gamma V_{i-1} | s_t = s]$$



Dynamic Programming Policy Evaluation

$$V^{\pi}(s) \leftarrow \mathbb{E}_{\pi}[r_t + \gamma V_{i-1} | s_t = s]$$

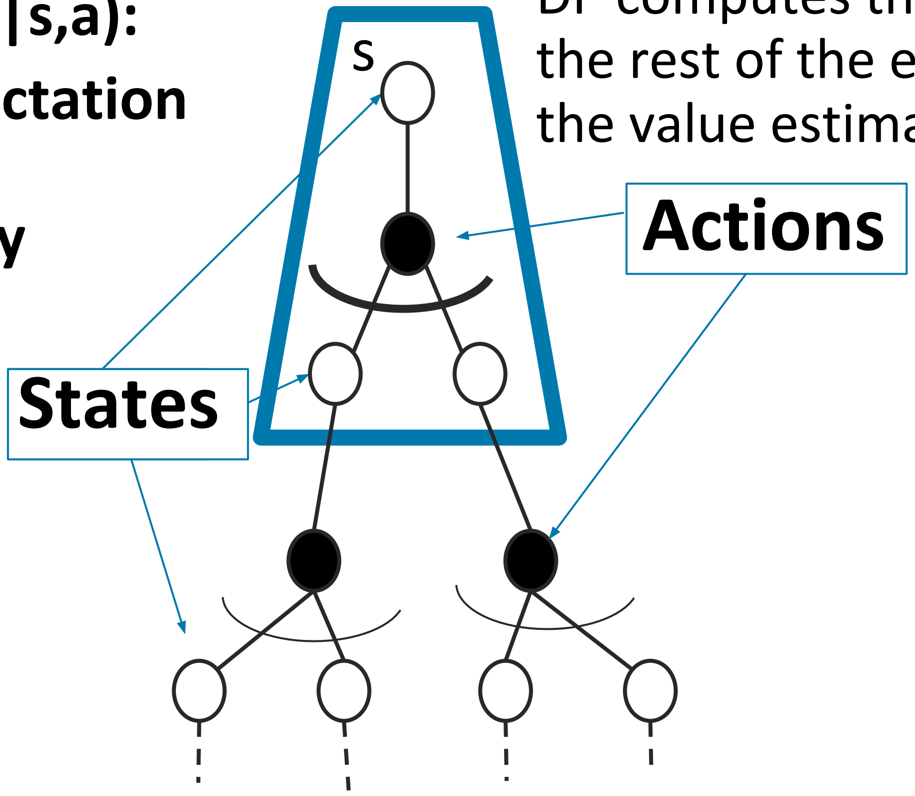


Dynamic Programming Policy Evaluation

$$V^{\pi}(s) \leftarrow \mathbb{E}_{\pi}[r_t + \gamma V_{i-1} | s_t = s]$$

Know model P(s'|s,a): reward and expectation over next states computed exactly

DP computes this, bootstrapping the rest of the expected return by the value estimate V_{i-1}



= Expectation

- Bootstrapping: Update for V uses an estimate

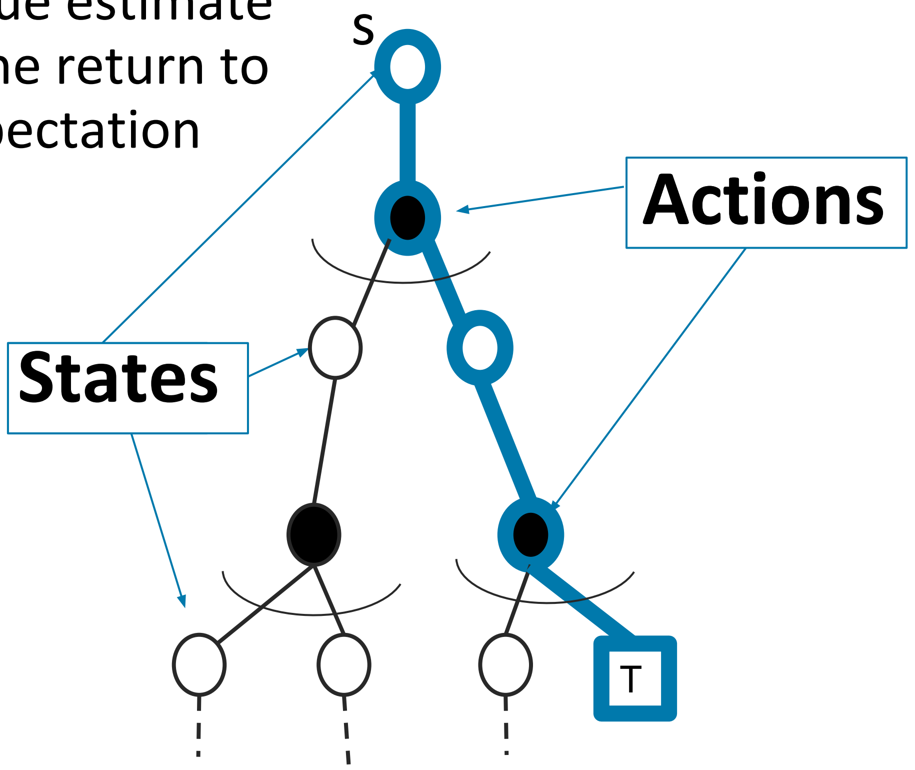
MC Policy Evaluation

$$V^{\pi}(s) = V^{\pi}(s) + \alpha(G_{it} - V^{\pi}(s))$$

MC Policy Evaluation

$$V^\pi(s) = V^\pi(s) + \alpha(G_{it} - V^\pi(s))$$

MC updates the value estimate using a **sample** of the return to approximate an expectation



= Expectation

= Terminal state

Temporal Difference Policy Evaluation

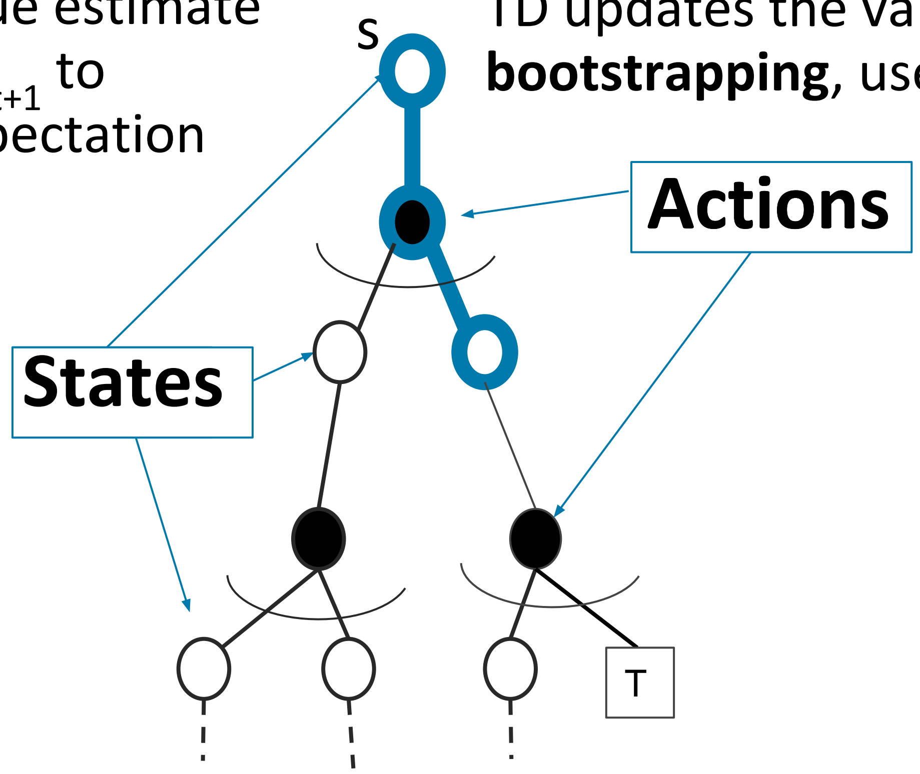
$$V^{\pi}(s_t) = V^{\pi}(s_t) + \alpha\left([r_t + \gamma V^{\pi}(s_{t+1})] - V^{\pi}(s_t)\right)$$

Temporal Difference Policy Evaluation

$$V^\pi(s_t) = V^\pi(s_t) + \alpha([r_t + \gamma V^\pi(s_{t+1})] - V^\pi(s_t))$$

TD updates the value estimate using a **sample** of s_{t+1} to approximate an expectation

TD updates the value estimate by **bootstrapping**, uses estimate of $V(s_{t+1})$



= Expectation

$\boxed{\text{T}}$ **= Terminal state**

Check Your Understanding?

(Answer Yes/No/NA to Each Algorithm for Each Part)

- Usable when no models of current domain
  - DP: MC: TD:
- Handles continuing (non-episodic) domains
  - DP: MC: TD:
- Handles Non-Markovian domains
  - DP: MC: TD:
- Converges to true value of policy in limit of updates*
  - DP: MC: TD:
- Unbiased estimate of value
  - DP: MC: TD:

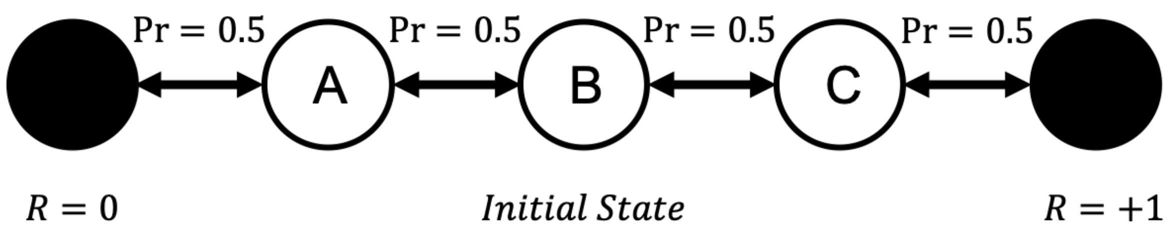
* For tabular representations of value function.

Some Important Properties to Evaluate Policy Evaluation Algorithms

- Usable when no models of current domain
  - DP: No MC: Yes TD: Yes
- Handles continuing (non-episodic) domains
  - DP: Yes MC: No TD: Yes
- Handles Non-Markovian domains
  - DP: No MC: Yes TD: No
- Converges to true value in limit*
  - DP: Yes MC: Yes TD: Yes
- Unbiased estimate of value
  - DP: NA MC: Yes TD: No

* For tabular representations of value function. More on this in later lectures

Random Walk

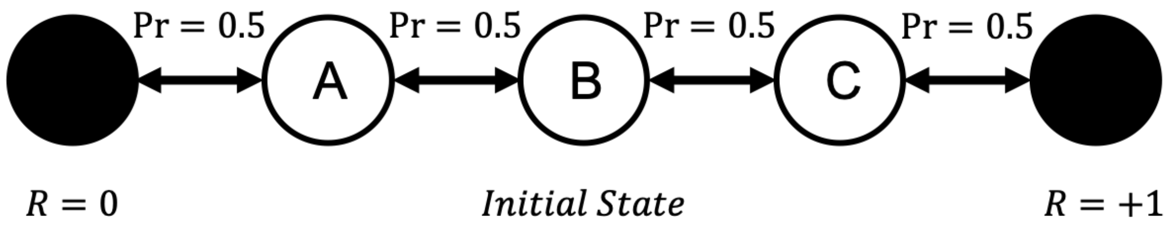


All states have zero reward, except the rightmost that has reward +1.
Black states are terminal.
Random walk with equal probability to each side.
Each episodes starts at state B and discount factor = 1

1.What is the true value of each state?

Consider the trajectory B, C, B, C, Terminal (+1)
2.What is the first visit MC estimate of V(B)?
3.What is the TD learning updates given the data in this order: (C, Terminal, +1), (B, C, 0), (C, B, 0)? with learning rate "a".
4.How about if we reverse the order of data? with learning rate "a".

Random Walk



1.What is the true value of each state?

Episodic, with 1 reward at right, value of each state is equal to the probability that a random walk terminates at the right side, So

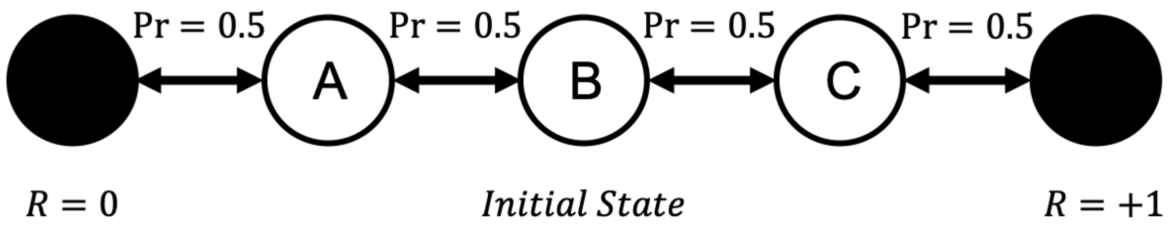
V(A) = 1/4, V(B) = 2/4, V(c) = 3/4

Consider the trajectory B, C, B, C, Terminal (+1)

2.What is MC first visit estimate of V(B)?

MC(V(B)) = +1

Random Walk



3.What is the TD learning updates given the data in this order: (C, Terminal, +1), (B, C, 0), (C, B, 0)? How about if we reverse the order of data?

$$V(C) = 0 + \alpha(1 - 0) = \alpha$$

$$V(B) = 0 + \alpha(0 + \gamma * \alpha - 0) = \gamma\alpha^2$$

$$V(C) = \alpha + \alpha(0 + \gamma * \gamma\alpha^2 - \alpha) = \alpha + \gamma^2\alpha^3 - \alpha^2$$

Reverse order:

$$V(C) = 0$$

$$V(B) = 0$$

$$V(C) = 0 + \alpha(1 - 0) = \alpha$$

Some Important Properties to Evaluate Model-free Policy Evaluation Algorithms

- Bias/variance characteristics
- Data efficiency
- Computational efficiency

What We’ve Covered So Far

- Markov decision process planning
- Model free policy evaluation
- **Model-free learning to make good decisions**
- Value function approximation, focus on model-free methods
- Imitation learning
- Policy search

Q-Learning

- Update Q(s,a) every time experience (s,a,s’,r)
    - Create new target / sample estimate

$$\begin{aligned} Q_{samp}(s,a) &= r + \gamma V(s') \\ &= r + \gamma \max_{a'} Q(s',a') \end{aligned}$$

- Update estimate of Q(s,a)

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha Q_{samp}(s,a)$$

Q-Learning Properties

- If acting randomly*, Q-learning converges Q*
  - Optimal Q values
  - Finds optimal policy
- Off-policy learning
  - Can act in one way
  - But learning values of another π (the optimal one!)

*Again, under mild reachability assumptions

Check Your Understanding: T/F (or T/F and under what conditions)

- In an MDP with finite state- and action spaces using a lookup table, Q-learning with a e-greedy policy converges to the optimal policy in the limit of infinite data.

- Monte-Carlo estimation cannot be used in MDPs with large state-spaces.

What We’ve Covered So Far

- Markov decision process planning
- Model free policy evaluation
- Model-free learning to make good decisions
- **Value function approximation, focus on model-free methods**
- Imitation learning
- Policy search

Monte Carlo vs TD Learning: Convergence in On Policy Case

- Evaluating value of a single policy

$$MSVE(w) = \sum_{s \in S} d(s) \left(V^{\pi}(s) - \tilde{V}^{\pi}(s, w) \right)^2$$

- where
  - d(s) is generally the on-policy π stationary distrib
  - ~V(s,w) is the value function approximation

Convergence given infinite amount of data?

Tsitsiklis and Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. 1997

Monte Carlo Convergence: Linear VFA

- Evaluating value of a single policy

$$MSVE(w) = \sum_{s \in S} d(s) \left(V^{\pi}(s) - \tilde{V}^{\pi}(s,w) \right)^2$$

- where
  - d(s) is generally the on-policy π stationary distrib
  - ~V(s,w) is the value function approximation
- Linear VFA: $V(s) = \sum_i w_i f_i(s)$
- **Monte Carlo converges to min MSE possible!**

$$MSVE(w_{MC}) = \min_w \sum_{s \in S} d(s) \left(V^{\pi}(s) - \tilde{V}^{\pi}(s,w) \right)^2$$

Tsitsiklis and Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. 1997

TD Learning Convergence: Linear VFA

- Evaluating value of a single policy

$$MSVE(w) = \sum_{s \in S} d(s) \left(V^{\pi}(s) - \tilde{V}^{\pi}(s, w) \right)^2$$

- where
  - d(s) is generally the on-policy π stationary distrib
  - ~V(s,w) is the value function approximation
- Linear VFA: $V(s) = \sum_i w_i f_i(s)$
- **TD converges to constant factor of best MSE**

$$MSVE(\boldsymbol{w}_{TD}) \leq \frac{1}{1-\gamma} \min_{\boldsymbol{w}} \sum_{s \in S} d(s)(V^{\pi}(s) - \hat{V}^{\pi}(s; \boldsymbol{w}))^2$$

Tsitsiklis and Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. 1997

Off Policy Learning

Q-learning with function approximation can diverge (not converge even given infinite data)

Deep Learning & Model Free Q learning

$$\mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\left[\left(\underbrace{r+\gamma\max_{a'}Q(s',a';w_i^-)}_{\text{Q-learning target}}-\underbrace{Q(s,a;w_i)}_{\text{Q-network}}\right)^2\right]$$

- Running stochastic gradient descent
- Now use a deep network to approximate Q

Challenges

- Challenge of using function approximation
  - Local updates (s,a,r,s’) highly correlated
  - “Target” (approximation to true value of s’) can change quickly and lead to instabilities

DQN: Q-learning with DL

- Experience replay of mix of prior (s_i,a_i,r_i,s_{i+1}) tuples to update Q(w)
- Fix target Q (w–) for number of steps, then update
- Optimize MSE between current Q and Q target

$$\mathbb{E}_{s,a,r,s'\sim\mathcal{D}_i}\left[\left(\underbrace{r+\gamma\max_{a'}Q(s',a';w_i^-)}_{\text{Q-learning target}}-\underbrace{Q(s,a;w_i)}_{\text{Q-network}}\right)^2\right]$$

- Use stochastic gradient descent

Deep RL

- Experience replay is hugely helpful
- Target stabilization is also helpful
- No guarantees on convergence (yet)
- Some other influential ideas
  - Double Q (two separate networks, each act as a "target" for each other)
  - Dueling: separate value and advantage
  - Many advances in deep RL build on prior ideas for RL with look up table representations

Check Your Understanding: T/F (or T/F and under what conditions)

- In finite state spaces with features that can represent the true value function, TD learning with value function approximation always finds the true value function of the policy given sufficient data.

What We’ve Covered So Far

- Markov decision process planning
- Model free policy evaluation
- Model-free learning to make good decisions
- Value function approximation, focus on model-free methods
- **Imitation learning**
- **Policy search**

$\longrightarrow$ **These will only be tested at a lighter level, since hw 3 will be posted after the midterm**

Imitation Learning

- Behavioral cloning
  - Definition
  - What can go wrong

Imitation Learning

- Inverse reinforcement learning
  - Formulation
  - How many reward models are compatible with a demonstration of the state-action sequence assuming that sequence comes from the optimal policy?

Policy Search

- Why are stochastic parameterized policies useful?
- Are policy gradient methods the only form of policy search? If not, are they the best type?
- Does the likelihood ratio policy gradient require us to know the dynamics model?
- Give 2 ideas that are used to reduce the variance of the default likelihood ratio policy gradient estimator

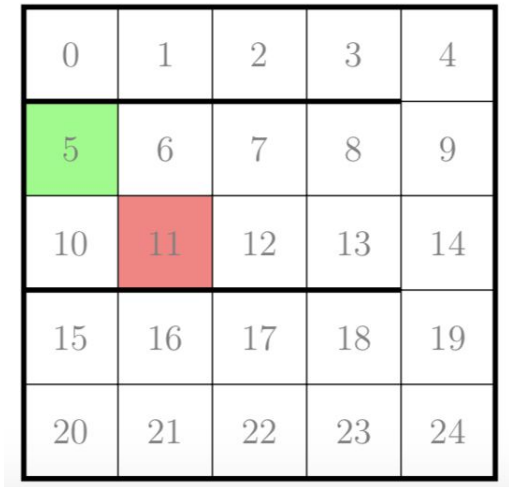
Midterm review

- Markov decision process planning
- Model free policy evaluation
- Model-free learning to make good decisions
- Value function approximation, focus on model-free methods
- Imitation learning
- Policy search

Midterm review

- To study: go through lecture notes, do all of assignments 1 and 2.
- Can also look through session materials for extra examples
- Do the practice midterm/s
  - Ignore parts on topics we have not covered in this iteration of CS234
  - Of two prior iterations, last year's is most similar to this year
- Reach out to us on piazza or office hours with any questions
- You can bring a 1 sided 1 page of notes.
- Good luck!

Extra Practice: Value Iteration



4 actions: Up, Down, Left, Right
Deterministic and all actions succeed (unless hitting the wall)
Taking any action from the green target square (#5) earns a reward of +5 and ends the episode.

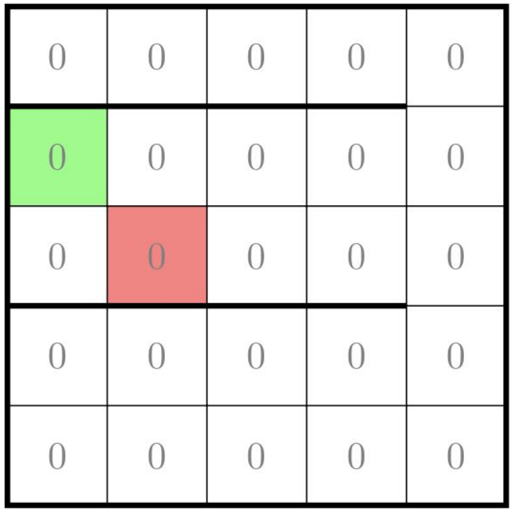
Taking any action from the red square (#11) earns a reward of -5 and ends the episode.

Otherwise each (s,a) pair has reward -1

What is the value of each state, after 1st and 2nd iteration of value iteration?, all values are initialized to 0.
What is the optimal value function?
What is the resulting optimal policy?

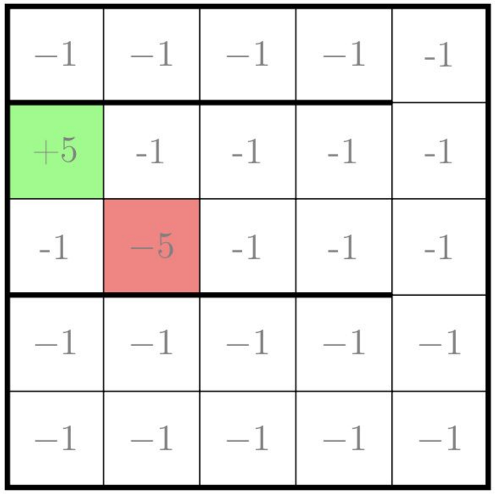
Extra Practice: Value Iteration

Step 0



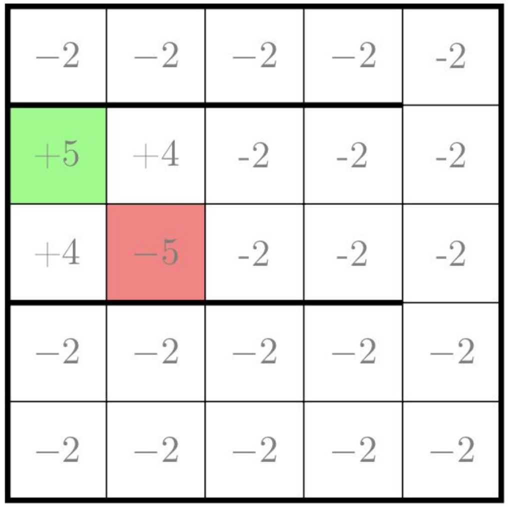
0	0	0	0	0
🟩 0	0	0	0	0
0	🟥 0	0	0	0
0	0	0	0	0
0	0	0	0	0

Step 1

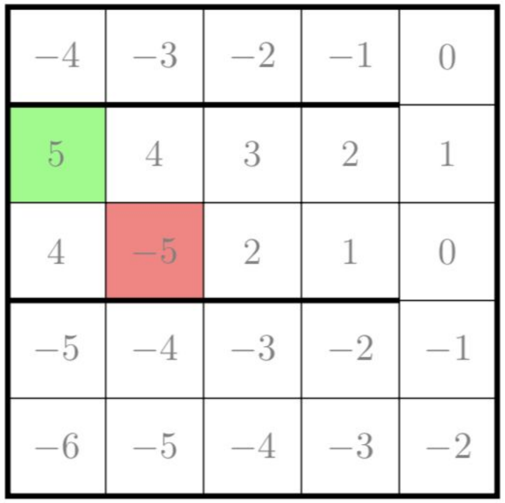


−1	−1	−1	−1	-1
🟩 +5	-1	-1	-1	-1
-1	🟥 −5	-1	-1	-1
−1	−1	−1	−1	−1
−1	−1	−1	−1	−1

Step 2



−2	−2	−2	−2	-2
🟩 +5	+4	-2	-2	-2
+4	🟥 −5	-2	-2	-2
−2	−2	−2	−2	−2
−2	−2	−2	−2	−2



−4	−3	−2	−1	0
🟩 5	4	3	2	1
4	🟥 −5	2	1	0
−5	−4	−3	−2	−1
−6	−5	−4	−3	−2

Optimal Value Function

Optimal Policy: Shortest path to the green state.