Introduction to Reinforcement Learning

Kevin Smeyers
Bram Vandendriessche
Katrien Van Meulder

Version, January 25, 2020

Introduction to Reinforcement Learning

Kevin Smeyers, Bram Vandendriessche, Katrien Van Meulder

#wearetothepoint, # AMLD2020

Outline

What is Reinforcement Learning (RL)

- RL is one of **three main learning problems** studied in AI:
- Learning by trial and error and improving over time;
- Natural form of learning, ubiquitous in biology and psychology:

What is Reinforcement Learning?

- **Supervised learning** needs teacher (labeled data!)
- **RL learns on the job:** interact, explore, exploit experience!

What makes RL attractive and interesting?

Natural way to learn complex skills, can we copy this?

RL Example 1: Playing a song by ear

- **Actions and transitions:** Moving from key to key...
- **Immediate reward:** correct note or not;
- **Cumulative reward:** Being able to play whole song
- Relatively easy, one gets immediate valuable feedback...

RL Example 2: Cooking

- **States:**
- **Actions:** e.g. cooking, seasoning, stirring, etc.
- **Transitions:** e.g. from raw to cooked
- **Immediate reward** e.g. smell, look
- **Final reward:** enjoyable dinner
- **Problem:** what's the best sequence of actions (i.e. optimal policy)?

RL Examples: Playing Chess

- **States:** board configurations
- **Actions:** legal moves
- **Value of state/action:** very valuable to determine next move (action),
- **Final reward:** Win(2), lose(0), draw(1)
- **Problem:** what's the optimal action to take in each state? (policy)

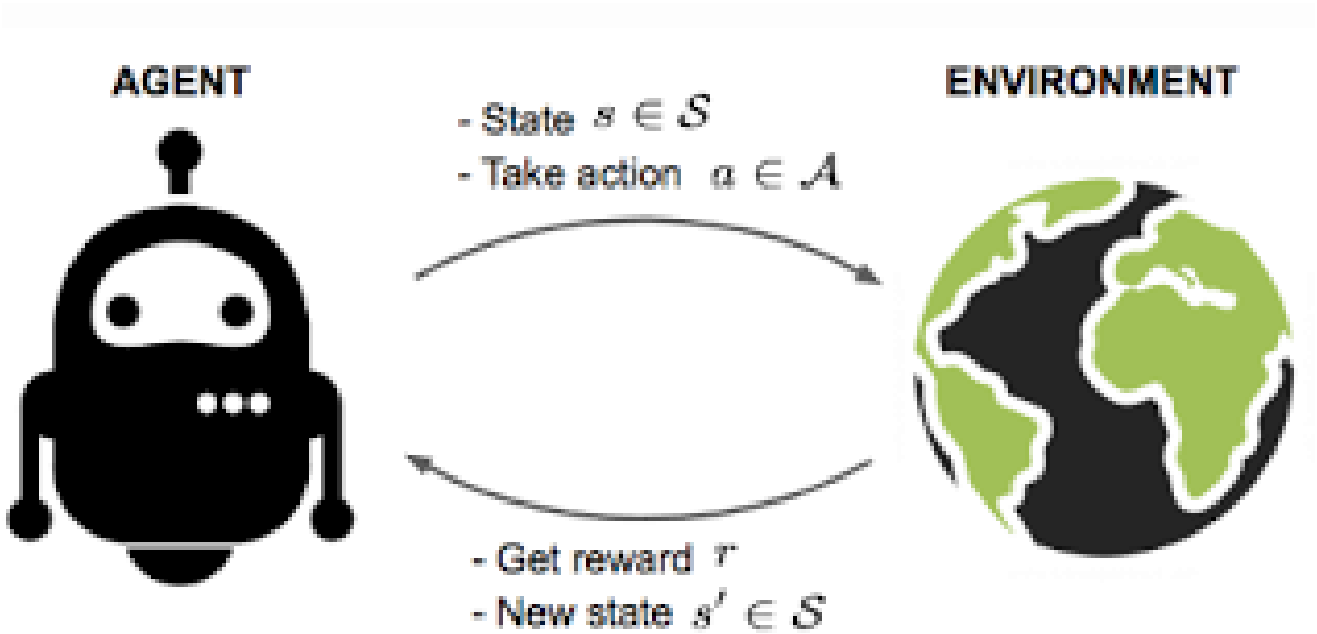
Reinforcement Learning: from **examples** to **abstraction**

- **RL:** Learning from (sequential) interaction:
  take actions to move from state to (better!) state;
- No labeled data, but **feedback** (reward, possibly delayed)
- **Goal:** optimise **end-result** (i.e. long-term/cumulative reward)

RL: Even more abstract version

Environment:

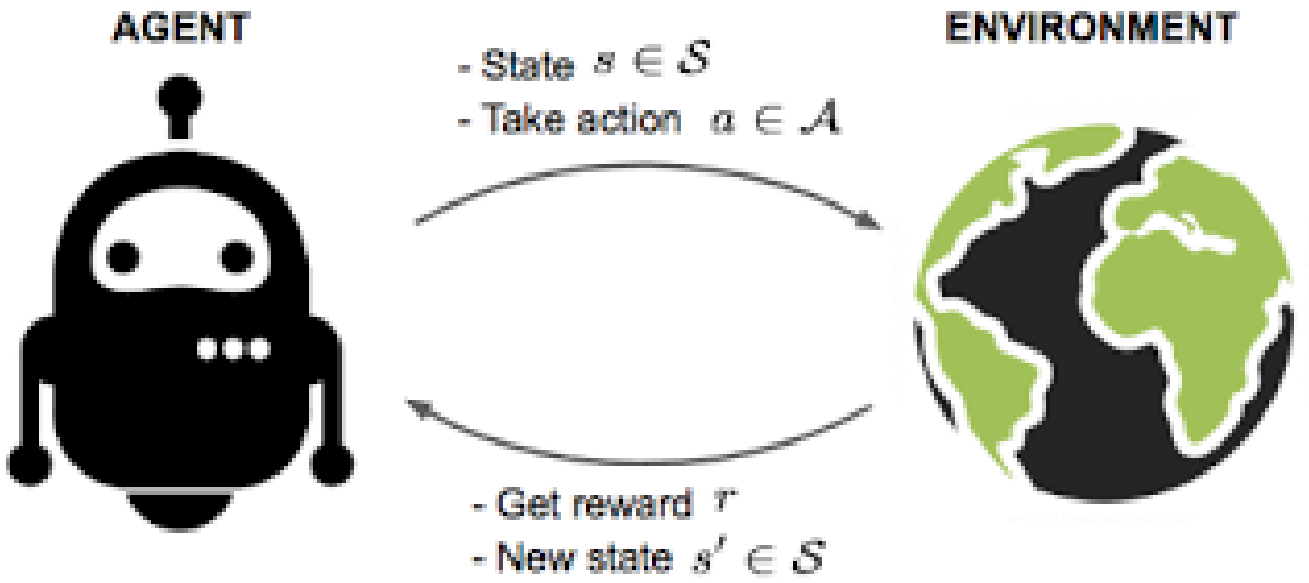
- states(s), actions (a) and transitions: $s \xrightarrow{a} s'$
- (immediate) reward $r(s, a, s')$



RL GOAL: Find optimal policy!

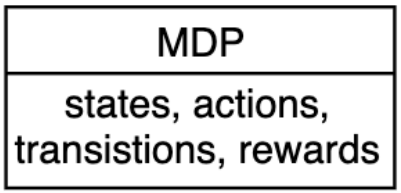
Policy π, tells for each state (s), which action (a) to take:

$$s \xrightarrow{\pi} a$$

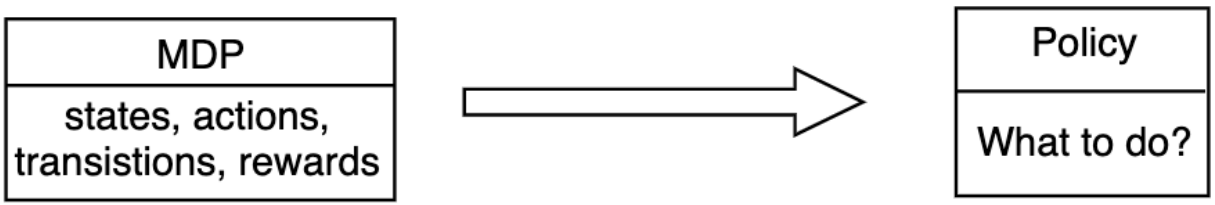


RL Goal: Given environment, determine **optimal policy** π^*:
what **action** *to choose in* **each state** *to achieve* **best FINAL reward***?*

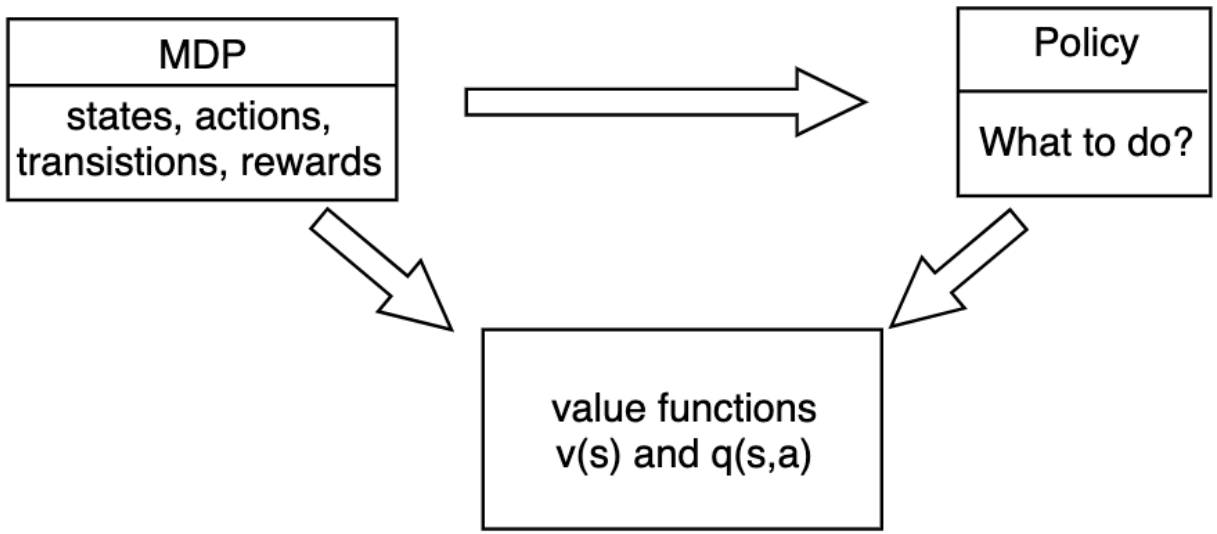
Overview



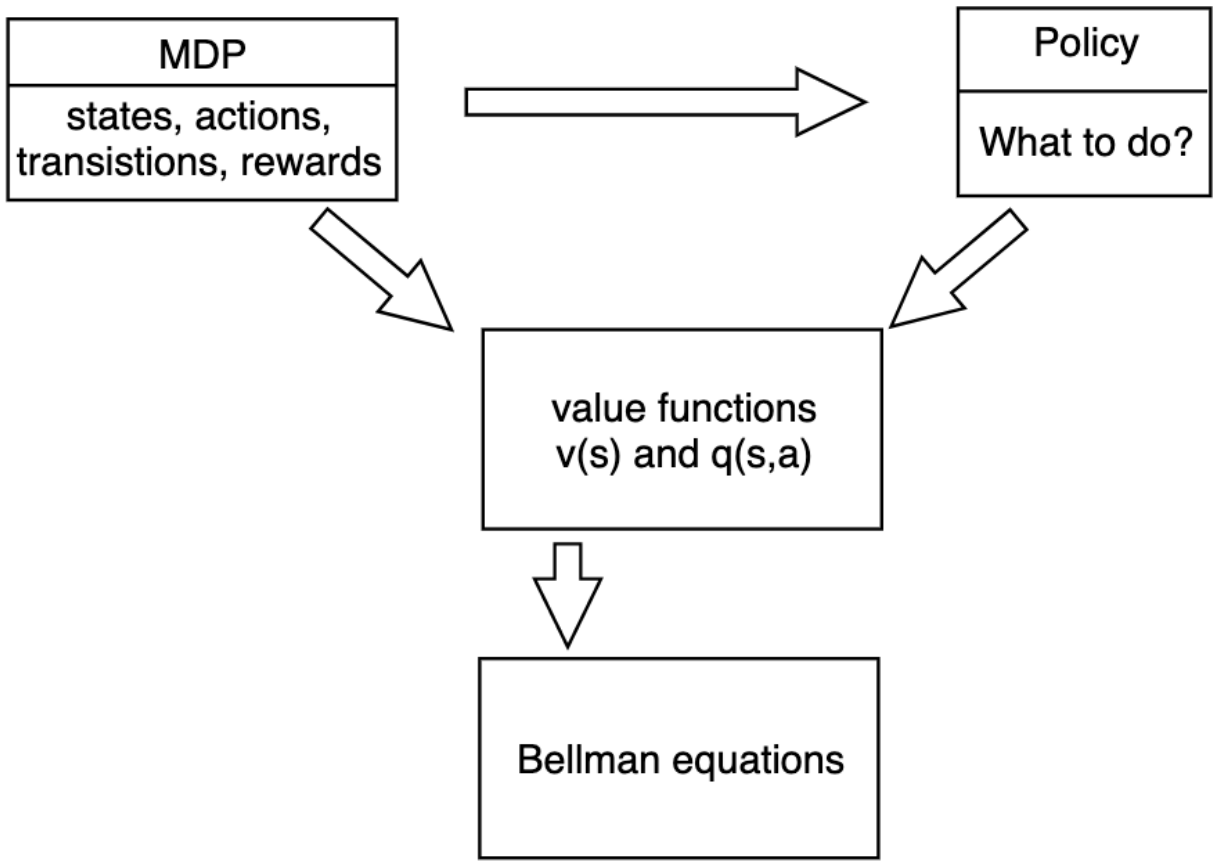
Overview



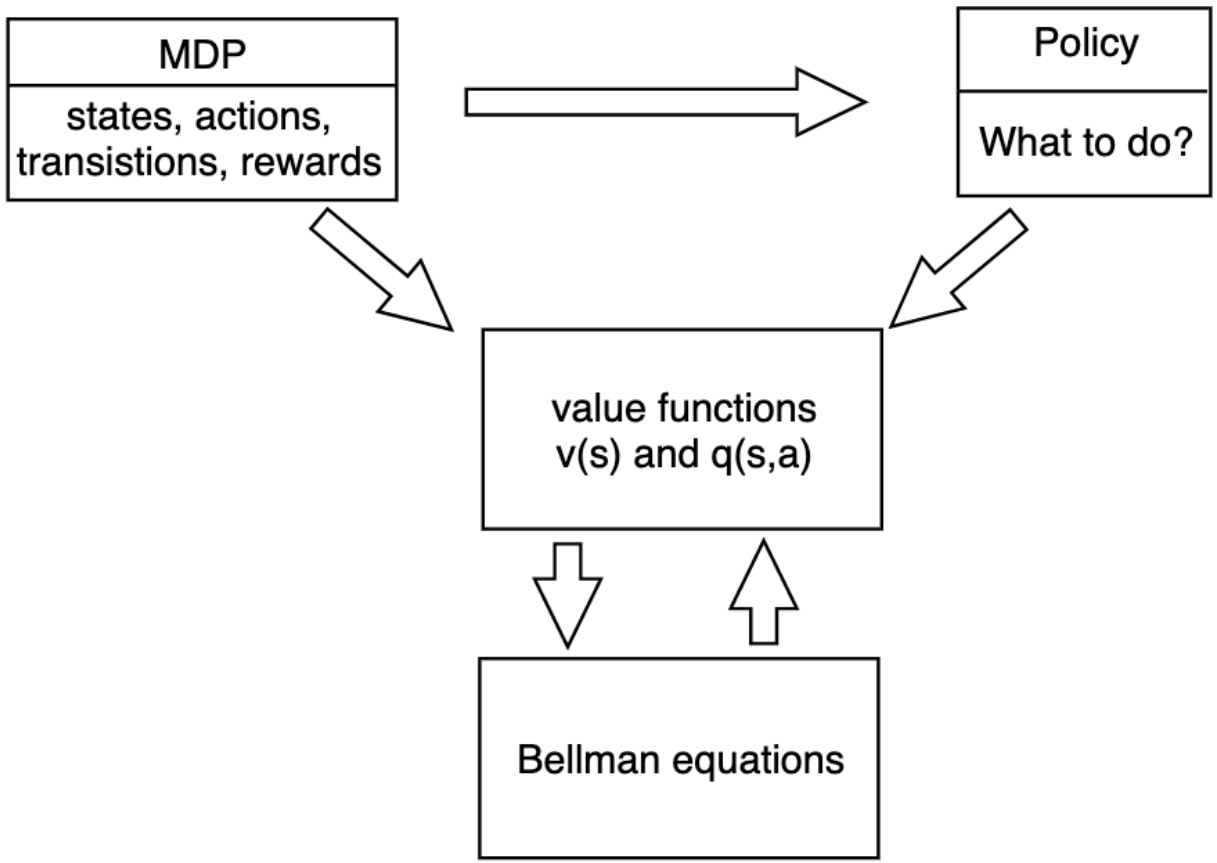
Overview


MDP
states, actions,
transistions, rewards
Policy
What to do?
value functions
v(s) and q(s,a)

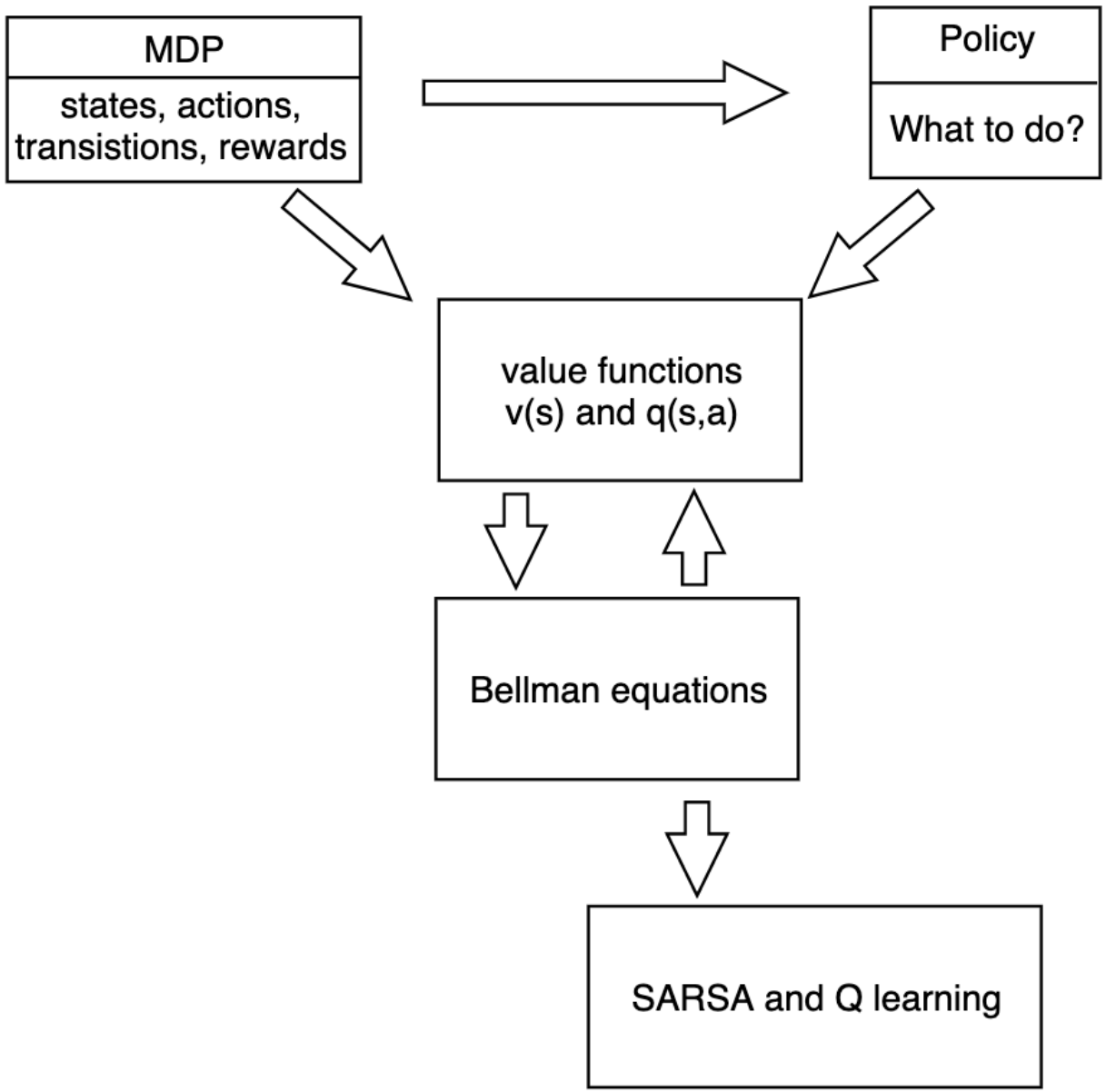
Overview


MDP
states, actions,
transistions, rewards
Policy
What to do?
value functions
v(s) and q(s,a)
Bellman equations

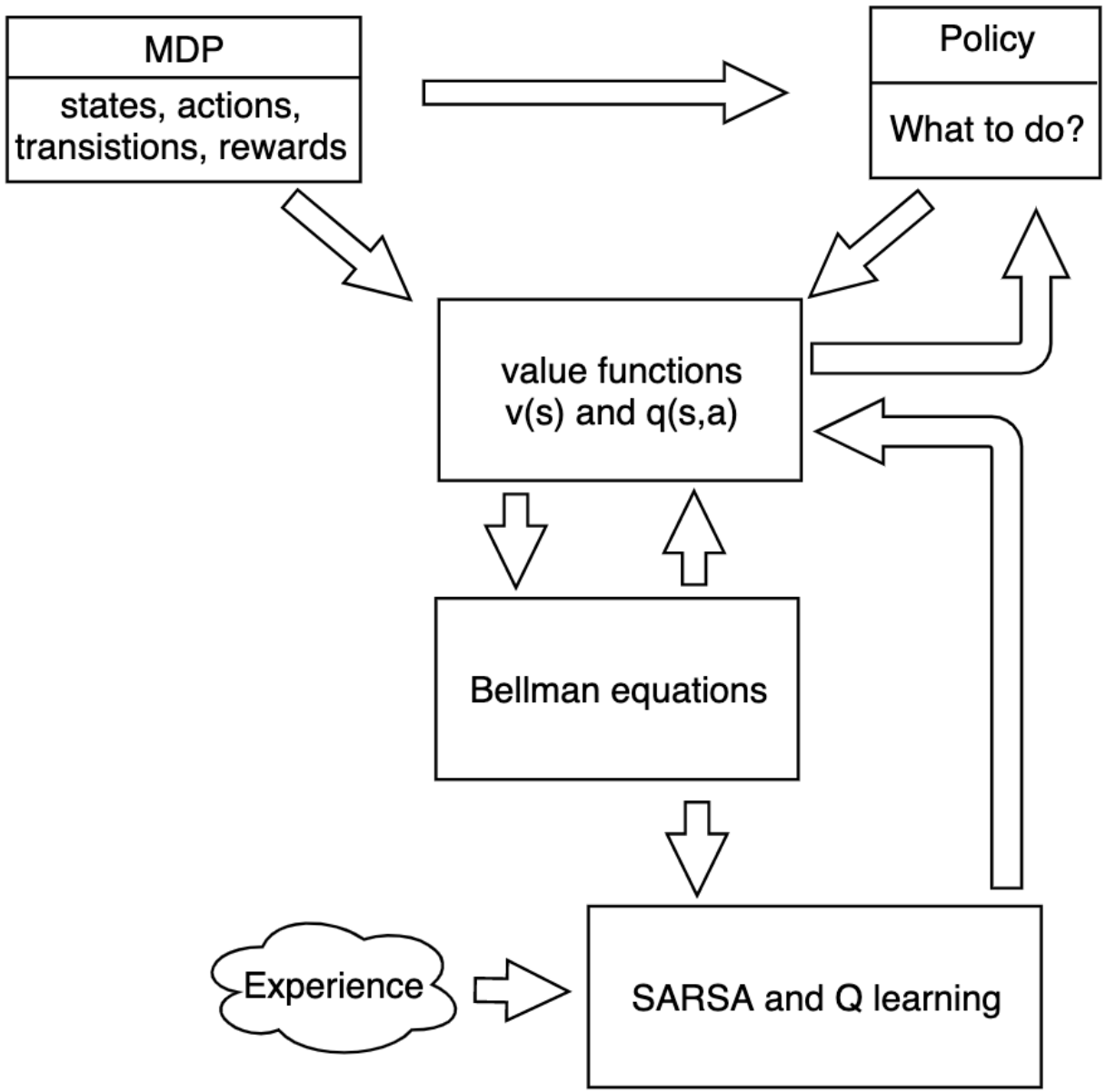
Overview


MDP
states, actions,
transistions, rewards
Policy
What to do?
value functions
v(s) and q(s,a)
Bellman equations

Overview


MDP
states, actions, transistions, rewards
Policy
What to do?
value functions v(s) and q(s,a)
Bellman equations
SARSA and Q learning

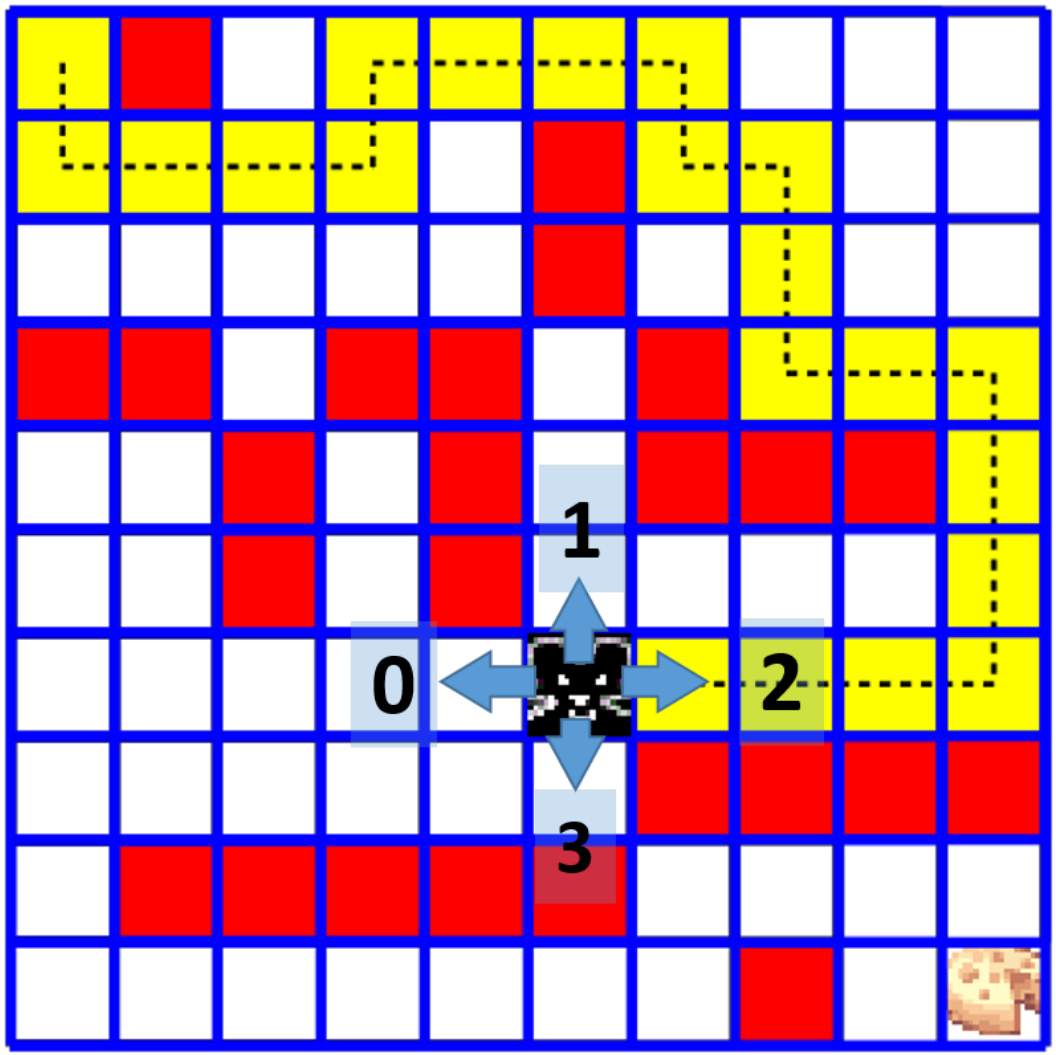
Overview


MDP
states, actions, transistions, rewards
Policy
What to do?
value functions v(s) and q(s,a)
Bellman equations
Experience
SARSA and Q learning

Signature Challenges for RL

- **Evaluative feedback** (reward), not prescriptive;
- **Sequential** (often **delayed**) consequences of actions;
- **Online learning:** improve policies while interacting with environment
  - In *off-line learning:* find optimal strategy **prior** to interaction!
- Need for **trial and error**, combination of exploration and exploitation;

Gridworld Maze as Prototypical RL Example



Outline

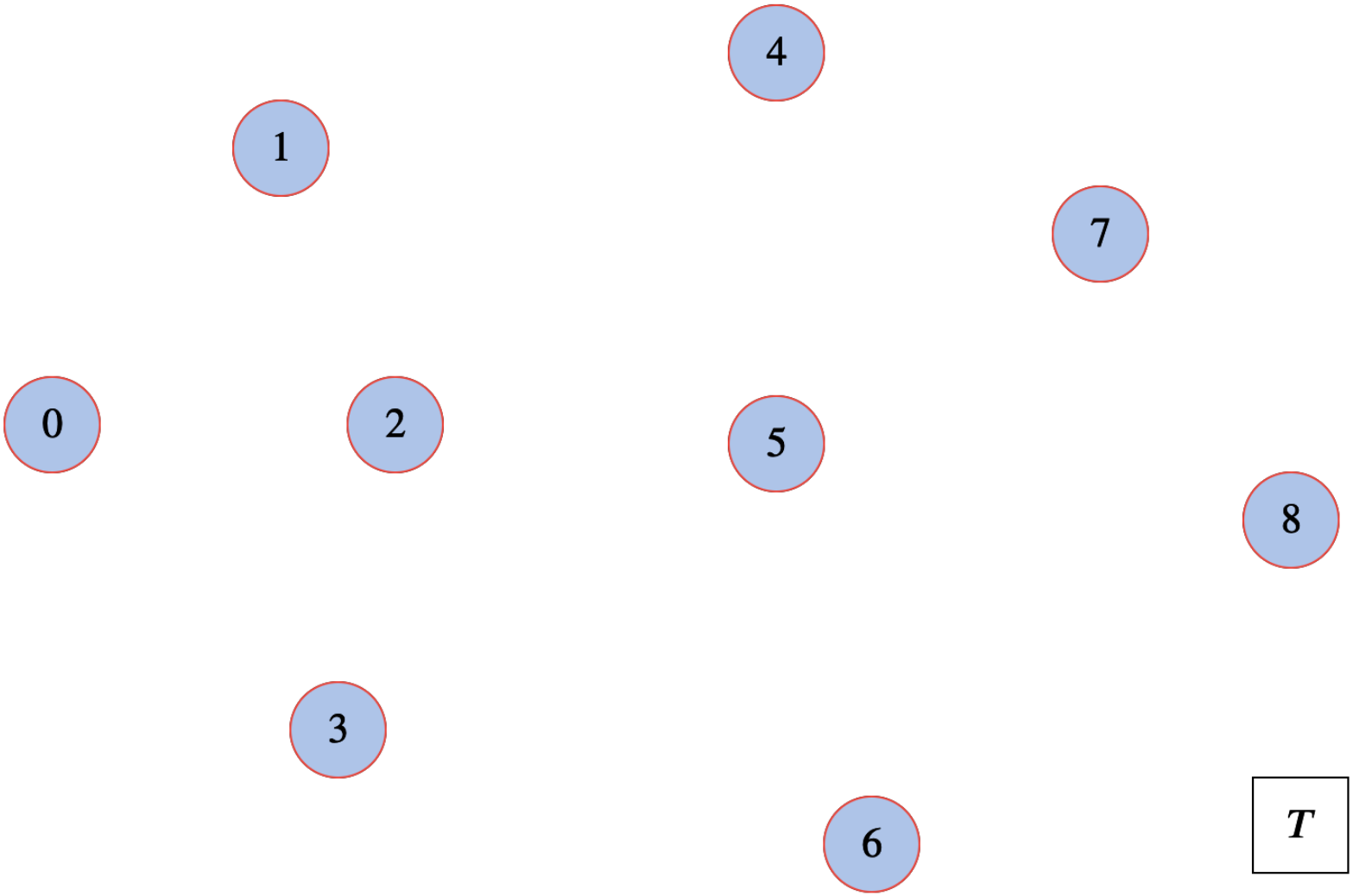
Markov Decision Process (MDP)

Mathematical formalism to capture the **essential characteristics** of the RL problem:

- MDP = states, actions, transitions, rewards, (discount factor)

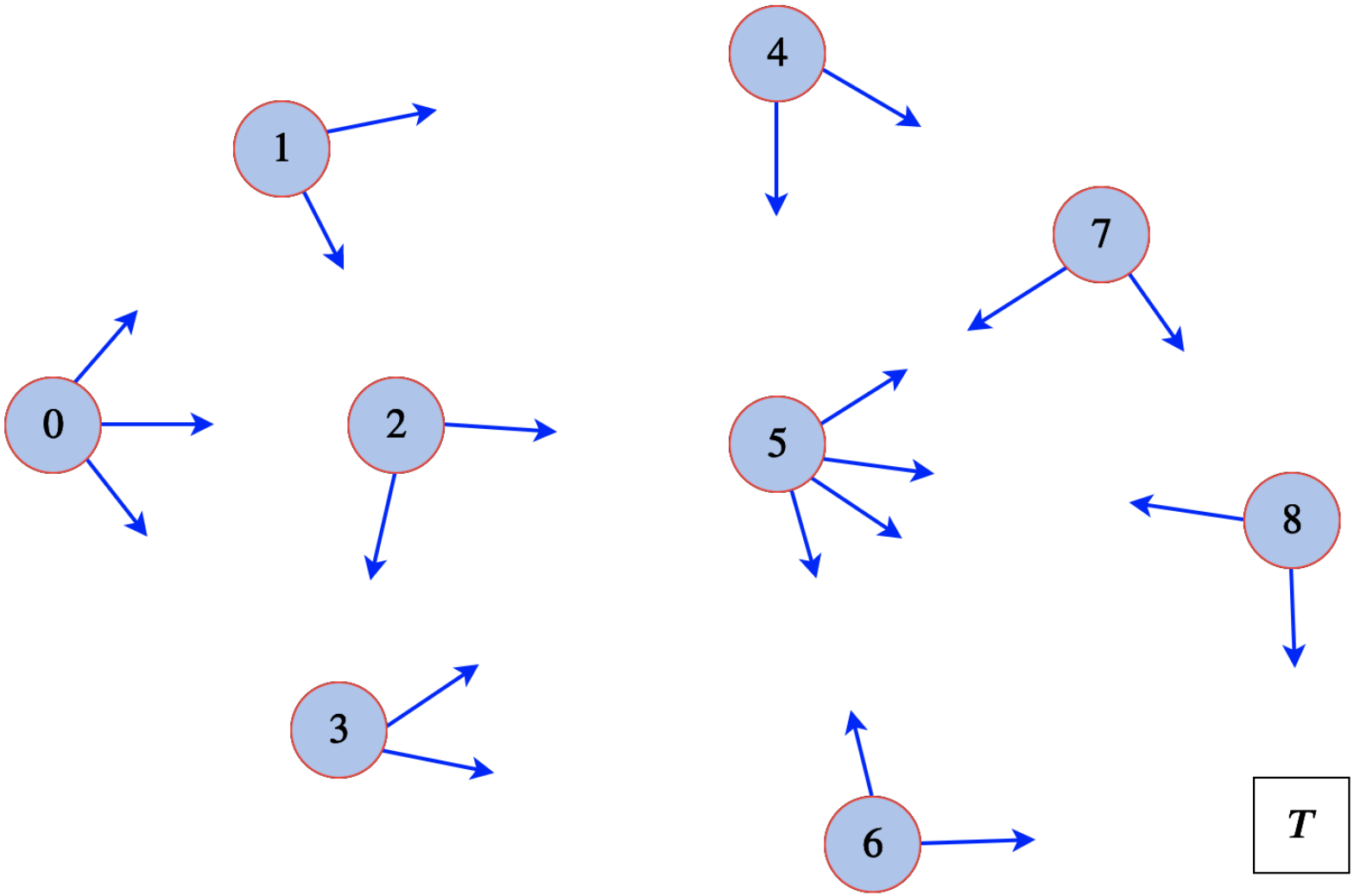
Markov Decision Process (MDP)

States

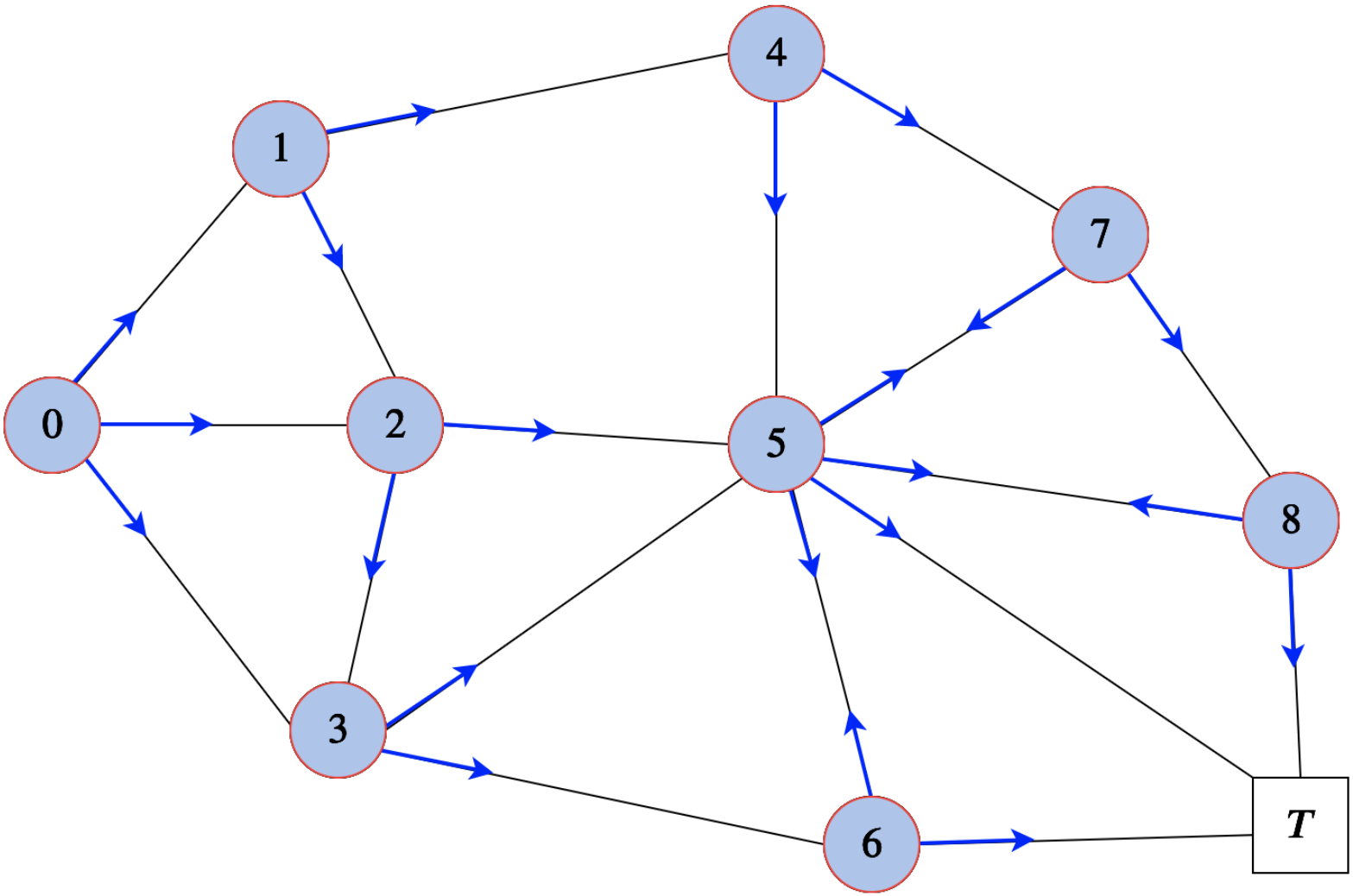


Markov Decision Process (MDP)

States + Actions

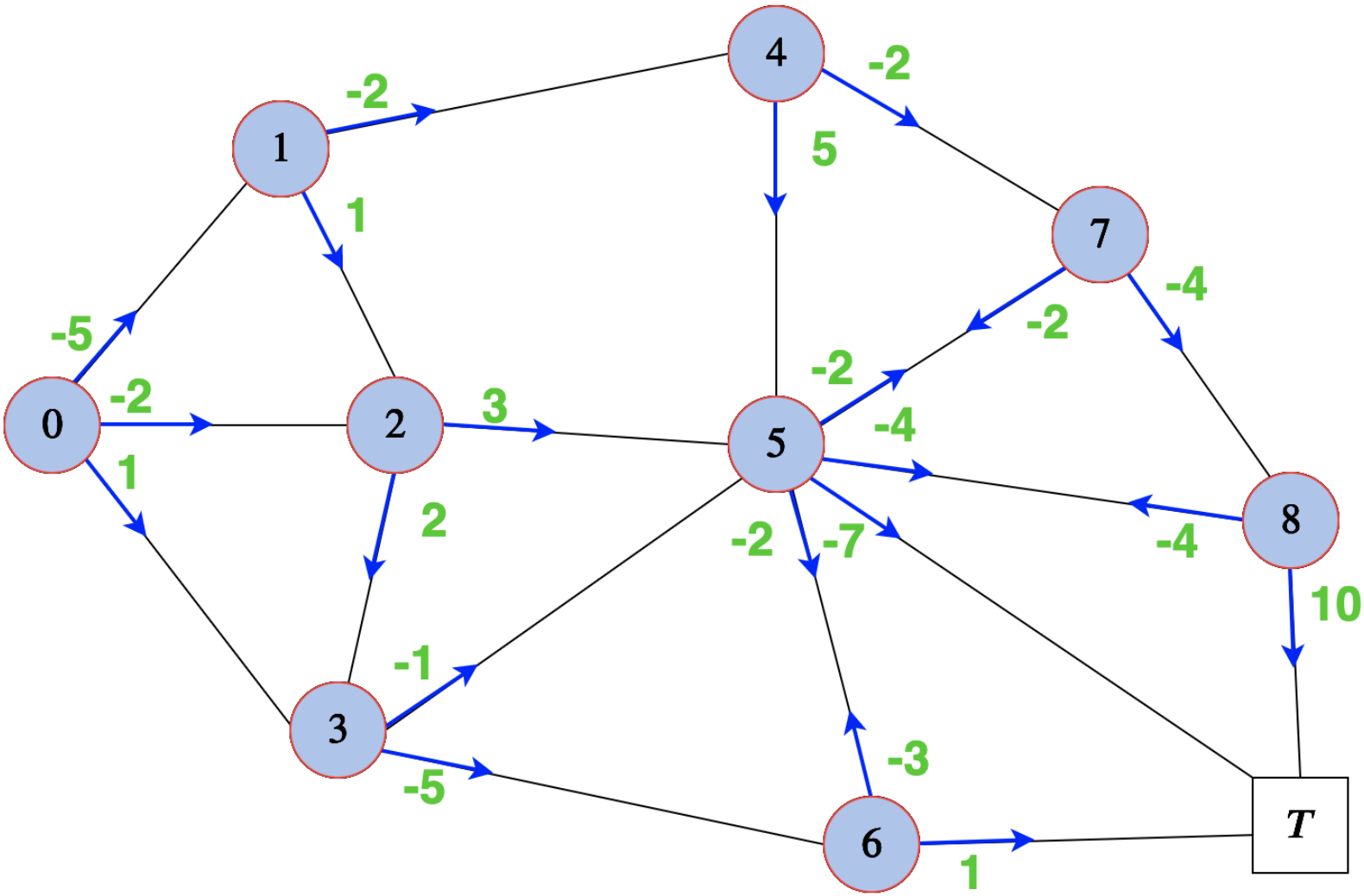


Markov Decision Process (MDP)
States + Actions + Transitions

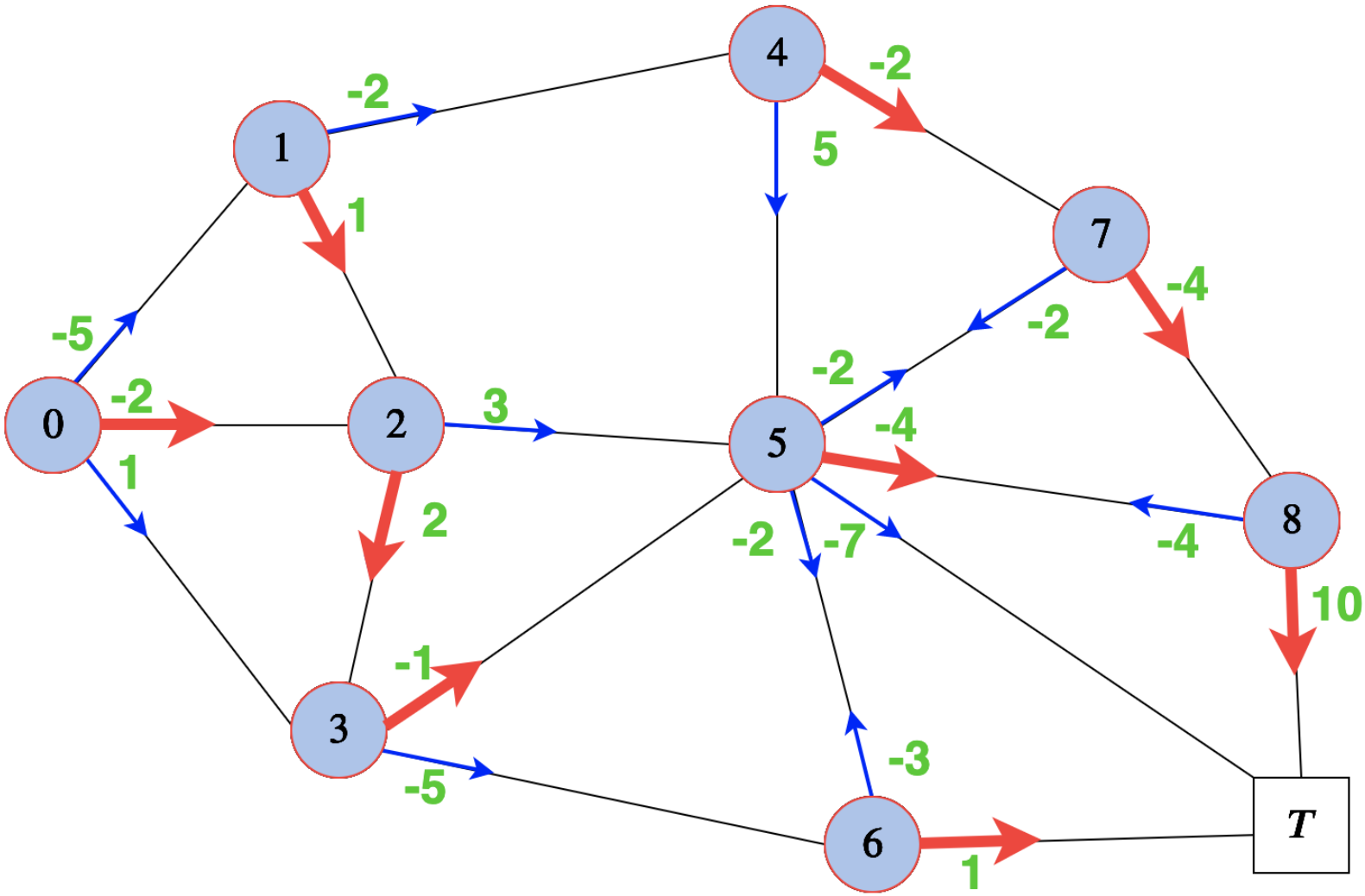


Markov Decision Process (MDP)

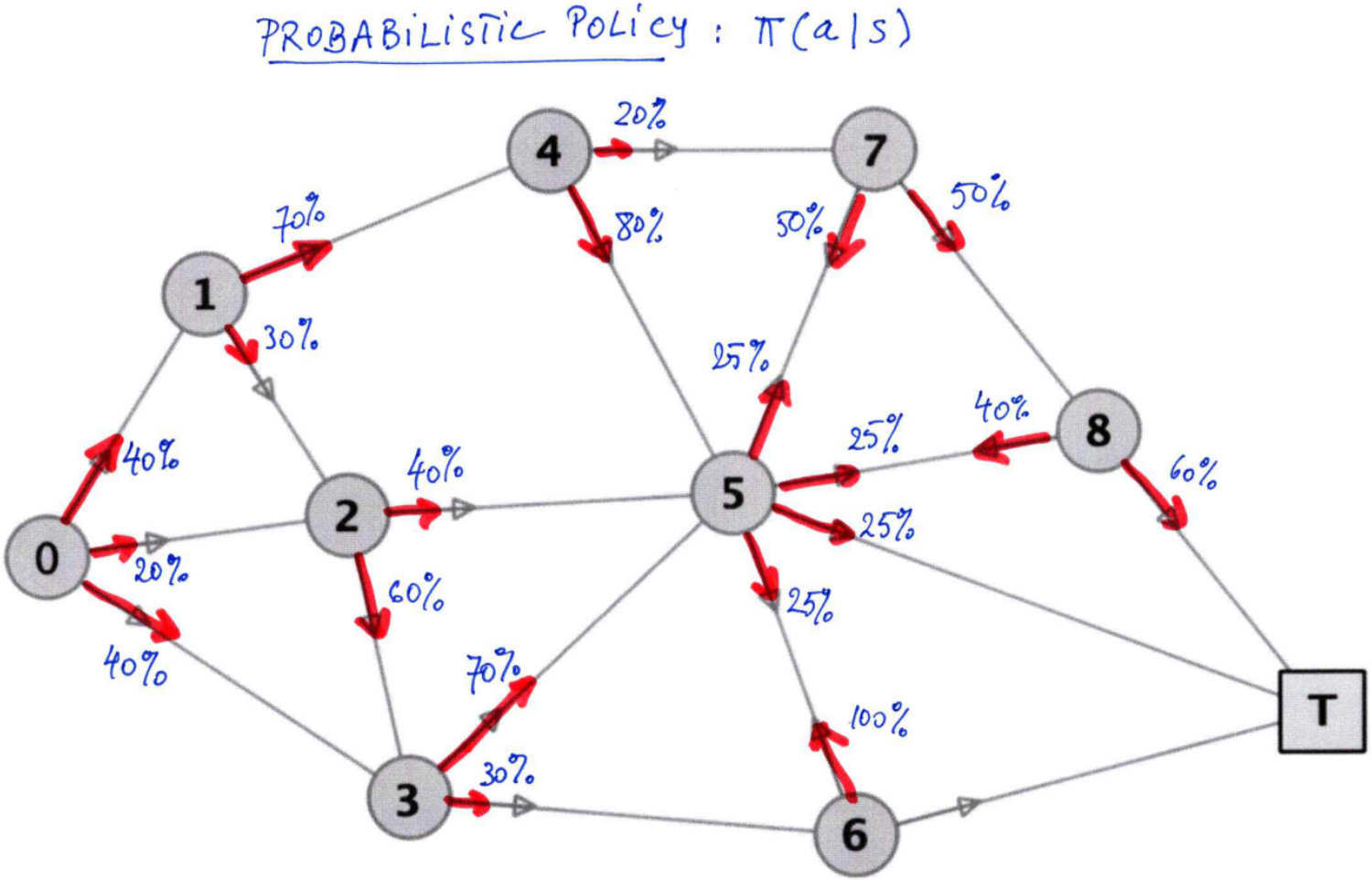
States + Actions + Transitions + Rewards



MDP + Policy (deterministic)

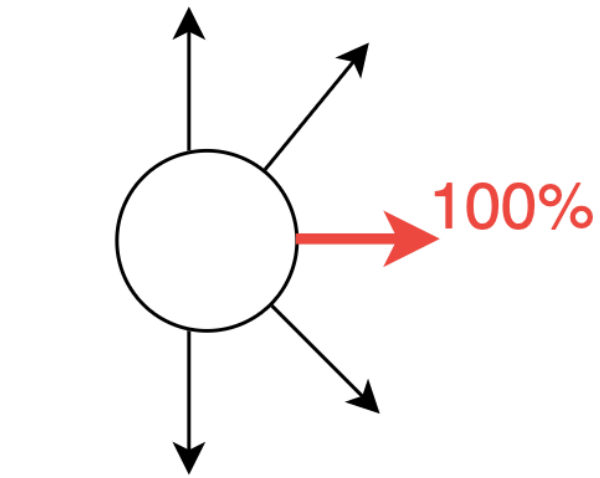


MDP + Policy (Probabilistic/Stochastic)

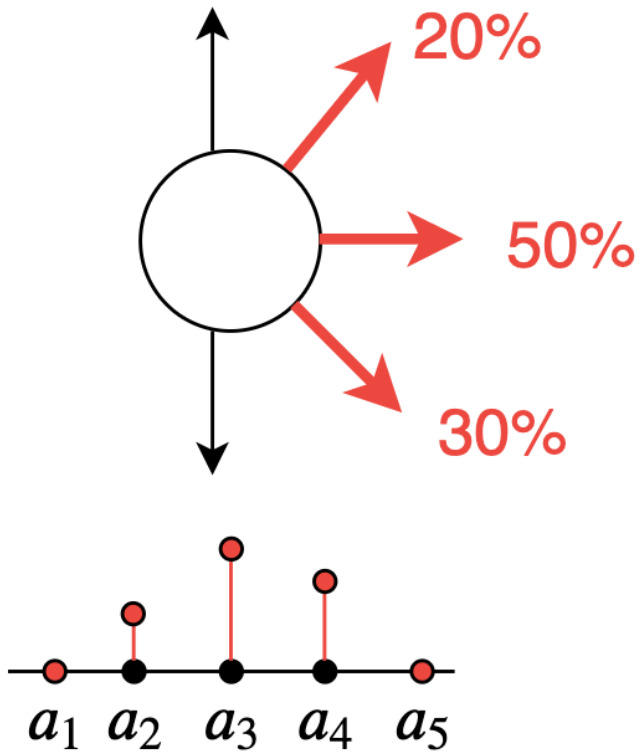


Deterministic vs. Probabilistic Policy

deterministic



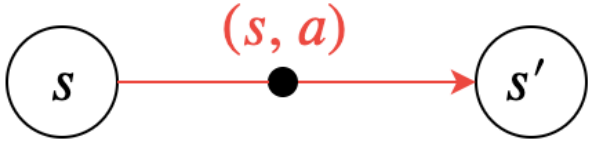
probabilistic



$$s \rightarrow \pi(a \mid s)$$

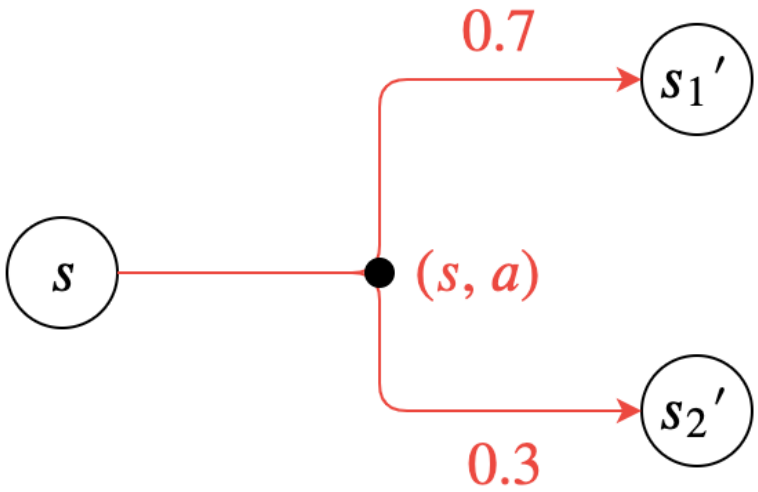
Deterministic vs. Probabilistic Transitions

Deterministic



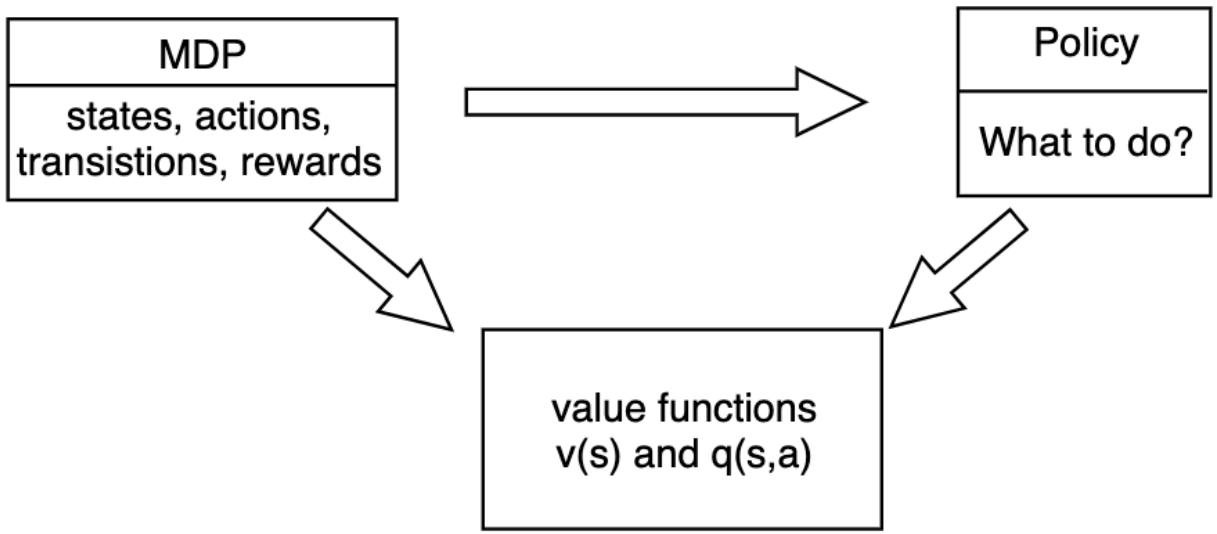
$$p(s' \mid s, a) = 1$$

Probabilistic



$$p(s_1' \mid s, a) = 0.7$$
$$p(s_2' \mid s, a) = 0.3$$

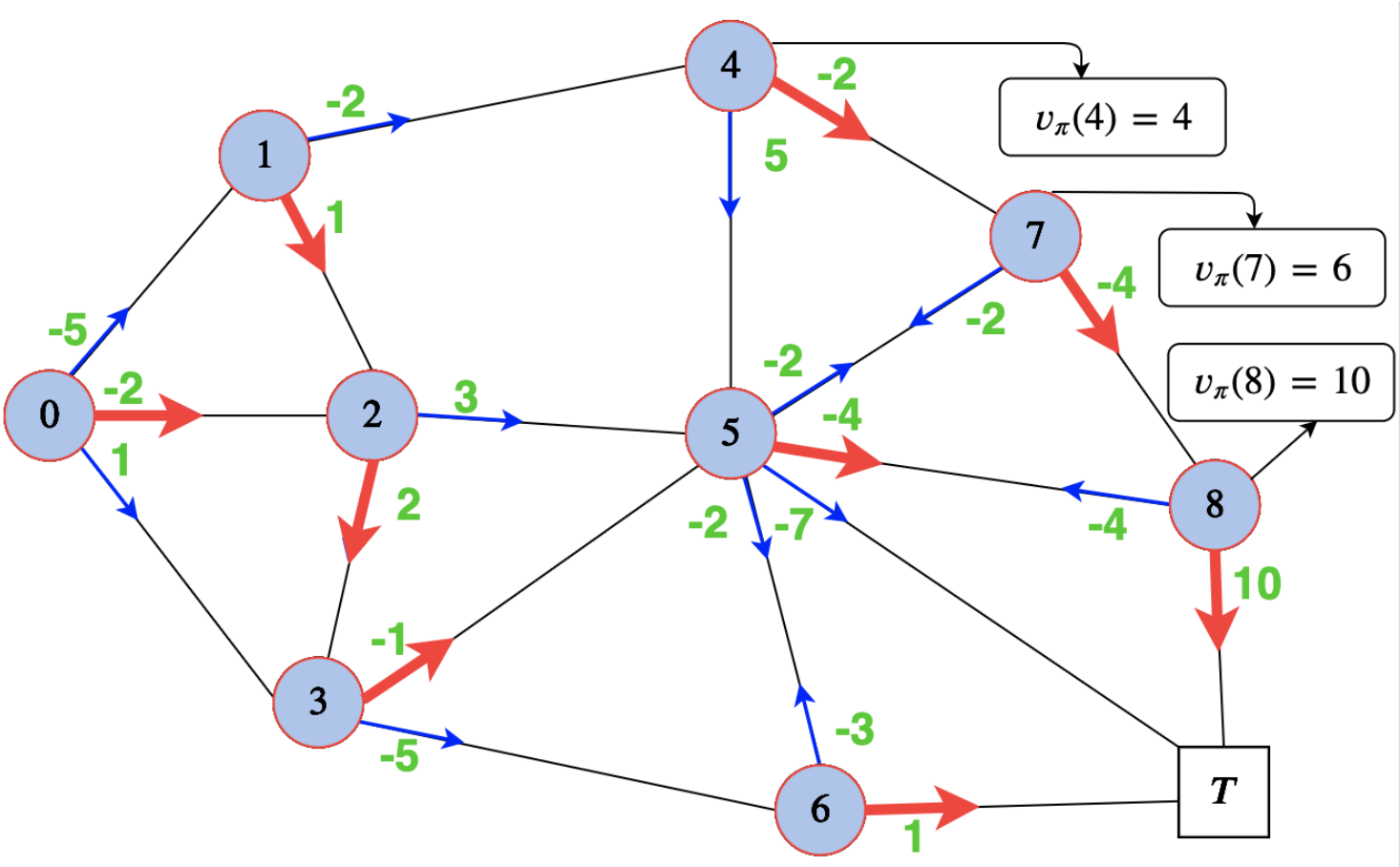
Overview


MDP
states, actions,
transistions, rewards
Policy
What to do?
value functions
v(s) and q(s,a)

MDP + Policy $\longrightarrow$ state value function $v_\pi(s)$

$v_\pi(s)$: value of state s when actions are specified by policy π:

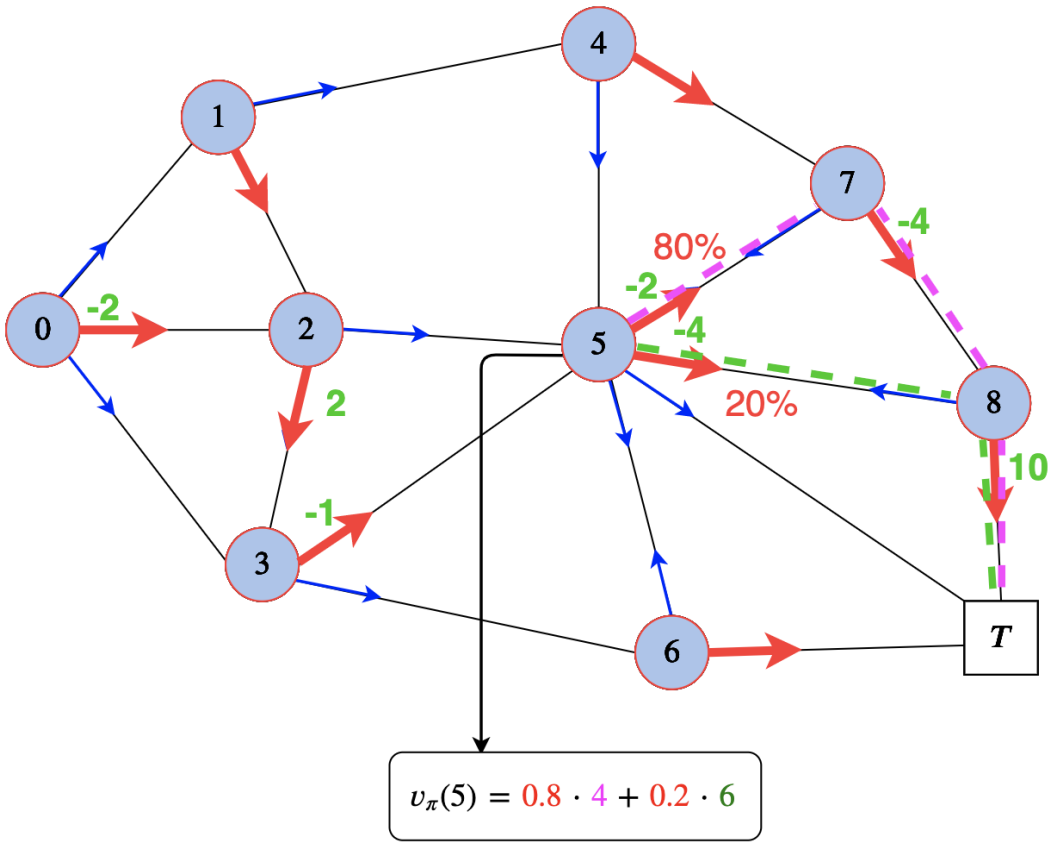
$$v_\pi(s) = \sum_{t=1}^{T} R_t \quad \text{given } s_0 = s$$



MDP + Policy $\longrightarrow$ state value function $v_\pi(s)$

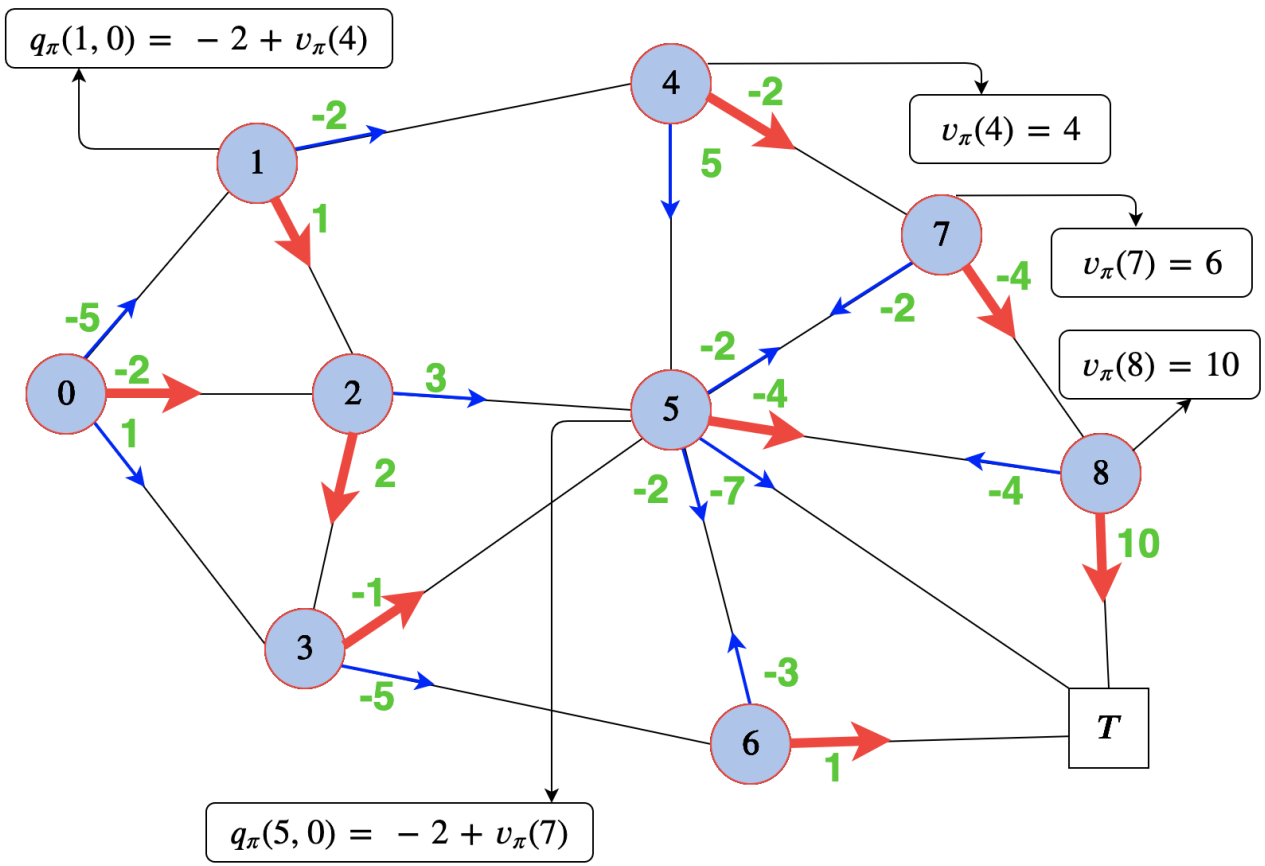
$v_\pi(s)$: expected value of state s when actions are specified by π:

$$v_\pi(s) = E_\pi\left(\sum_{t=1}^{T} R_t \,|\, s_0 = s\right)$$

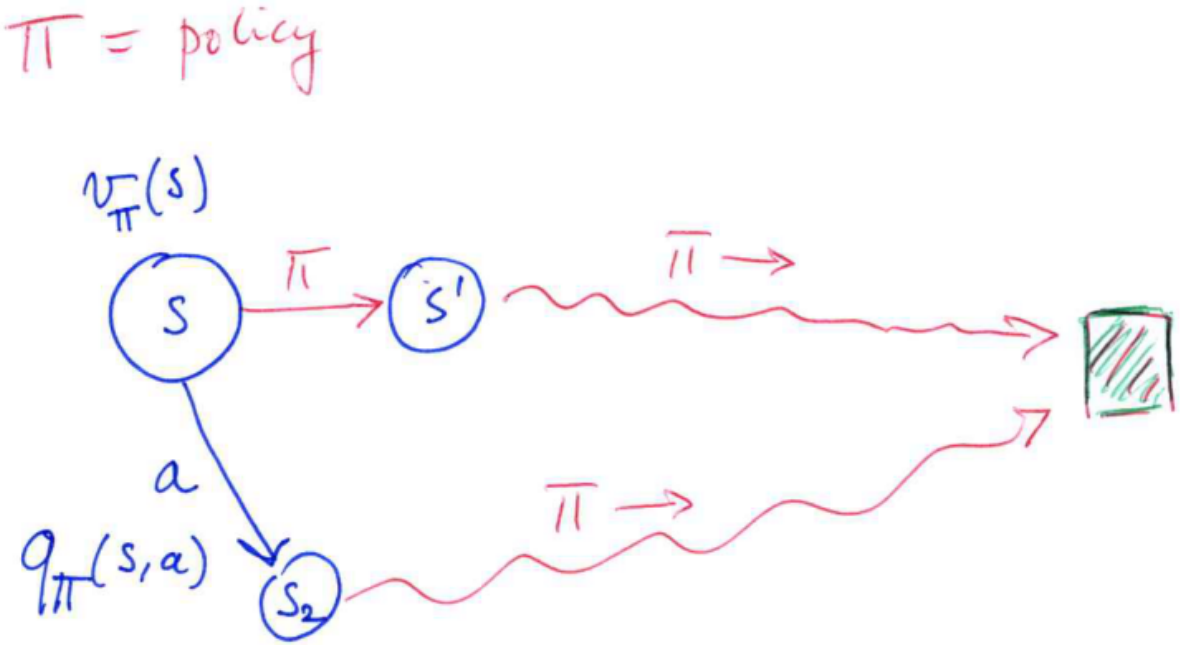


MDP + Policy $\longrightarrow$ state-action value function $q_\pi(s, a)$

$$q_\pi(s, a) = E_\pi \left(\sum_{t=1}^{T} R_t \,|\, s_0 = s,\ a_0 = a \right)$$



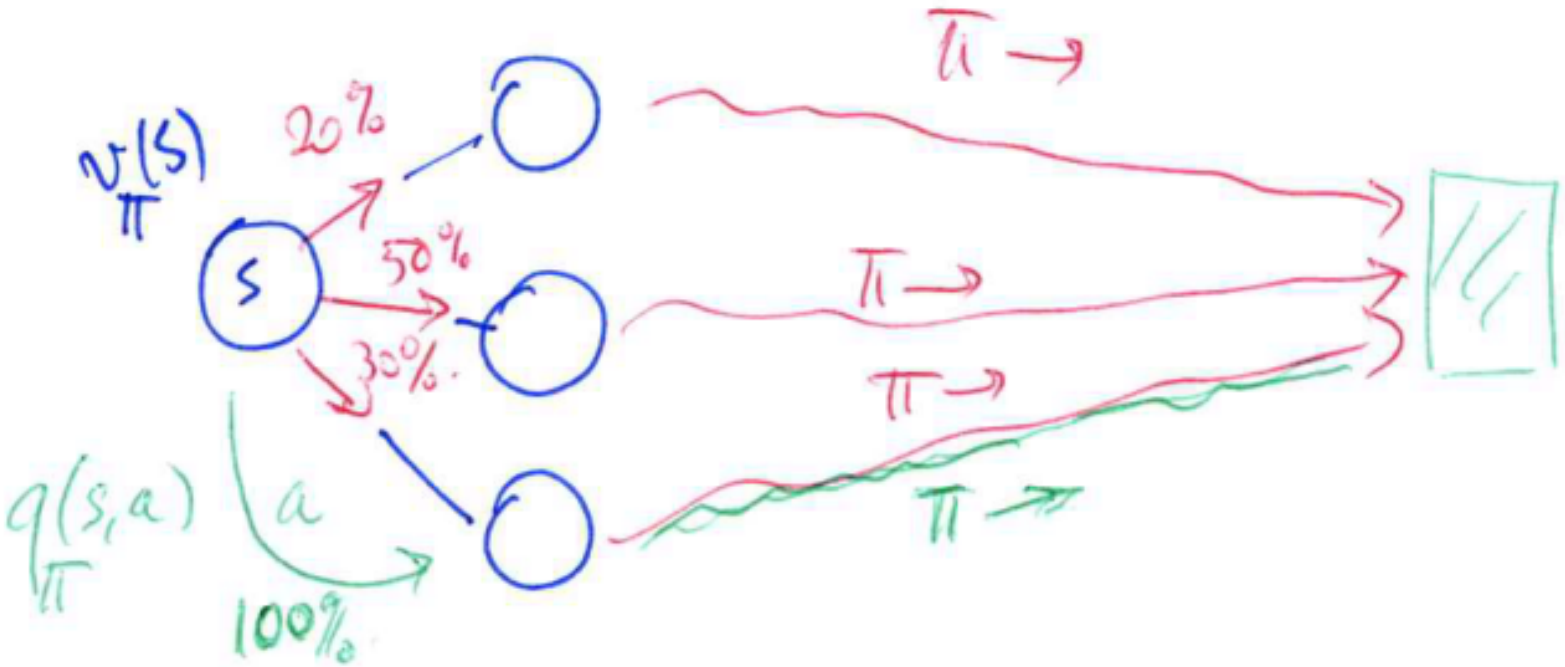
Difference between value functions $v_\pi(s)$ and $q_\pi(s, a)$



- $v_\pi(s)$: allow the policy π to dictate every action along the path;
- $q_\pi(s, a)$: first action a is taken independently of the policy π, from then onward, π dictates the remaining actions taken along the rest of the path.

Difference between value functions $v_\pi(s)$ and $q_\pi(s, a)$

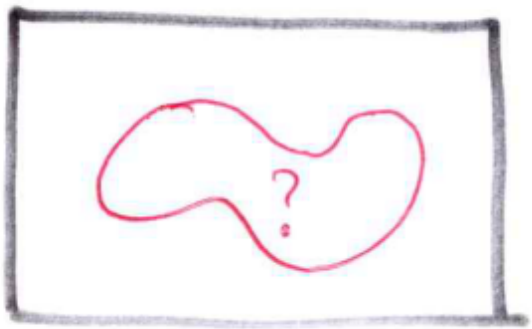
Similar interpretation for stochastic policy.

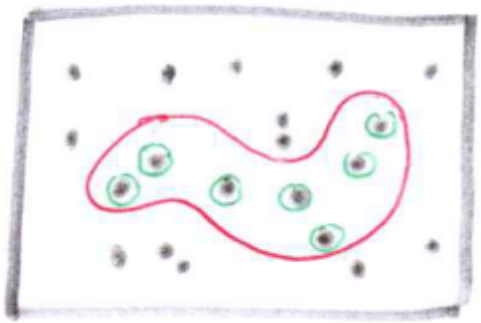


Outline

Coding excursion 1: Monte Carlo (MC) for value estimation

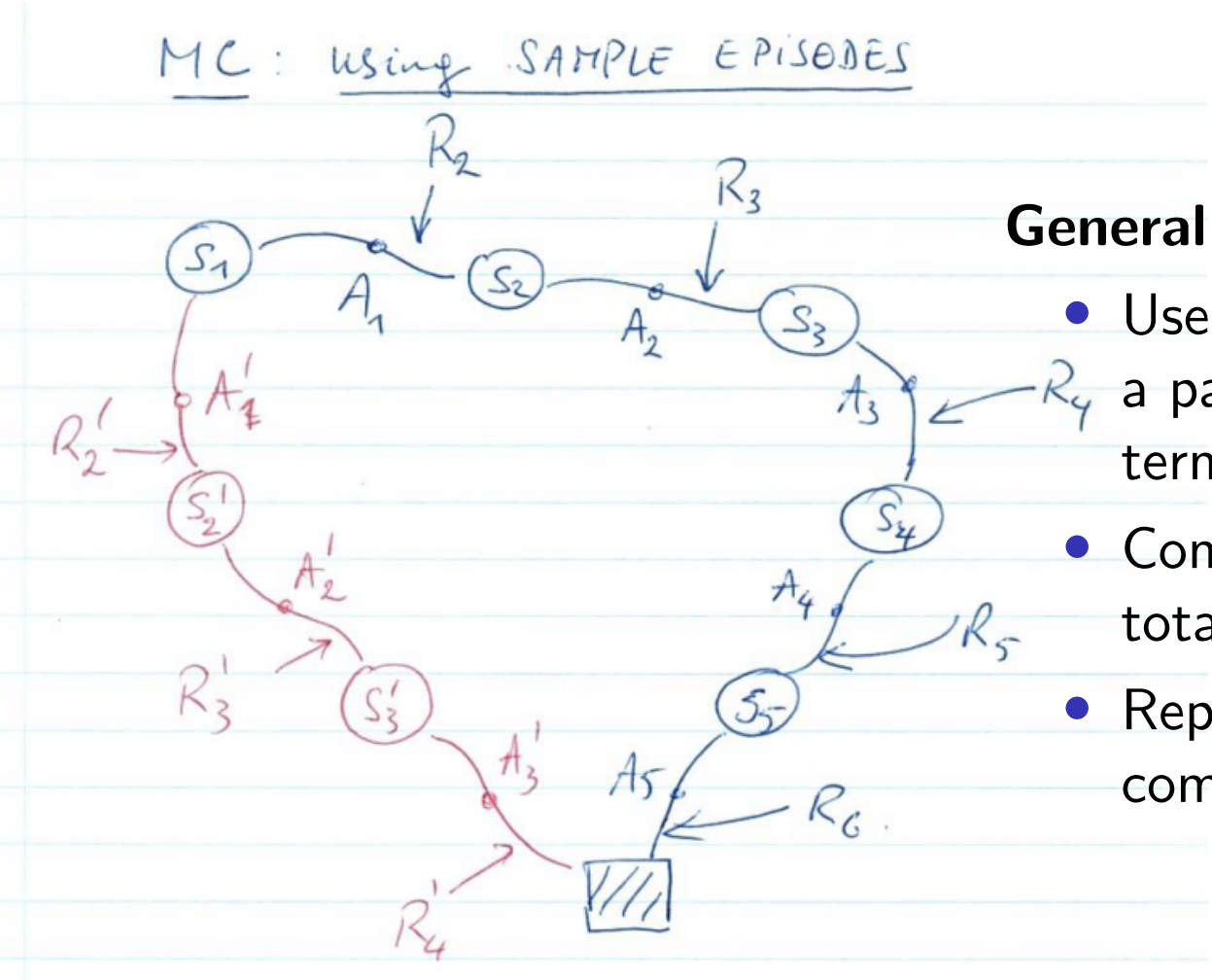
- Monte Carlo (MC): just do it!
- **Computational technique** for finding (approximate) solution to (difficult) problems using **random sampling**
- **Illustration:** Estimating area inside irregular contour:





- MC is costly, sometimes we can do better!

Coding excursion 1: Monte Carlo (MC) for value estimation

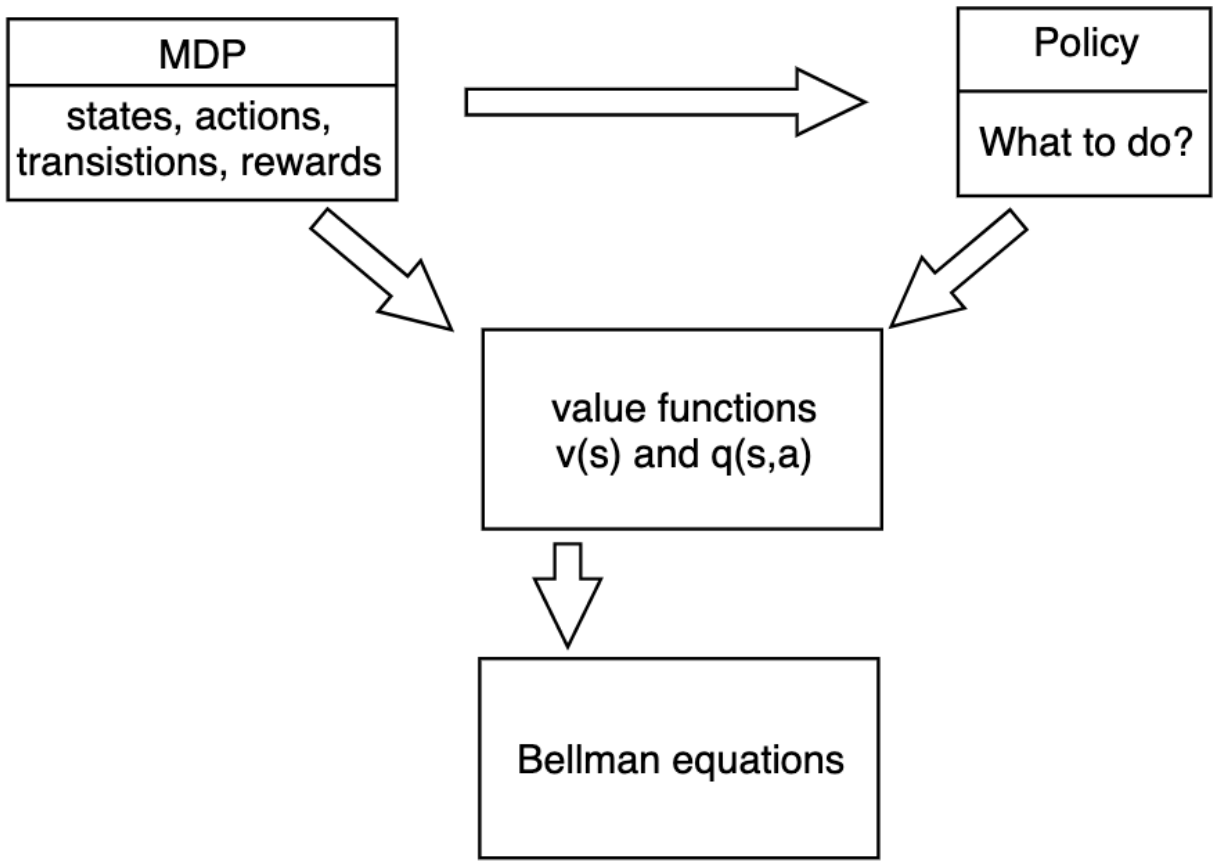


General principle:

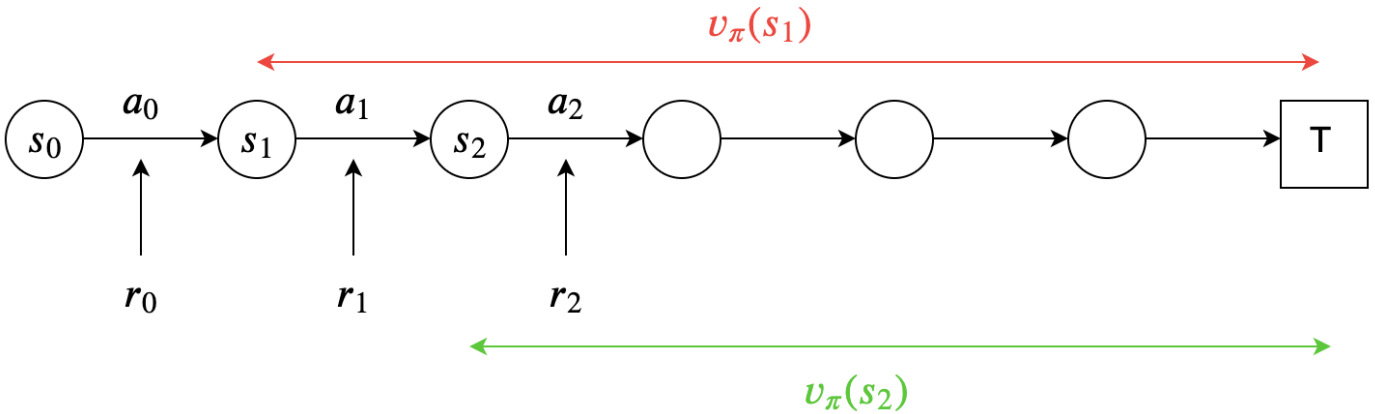
- Use the policy to roll-out a path from initial to terminal state;
- Compute the total/cumulative reward;
- Repeat many times and compute mean

Outline

Overview


MDP
states, actions,
transistions, rewards
Policy
What to do?
value functions
v(s) and q(s,a)
Bellman equations

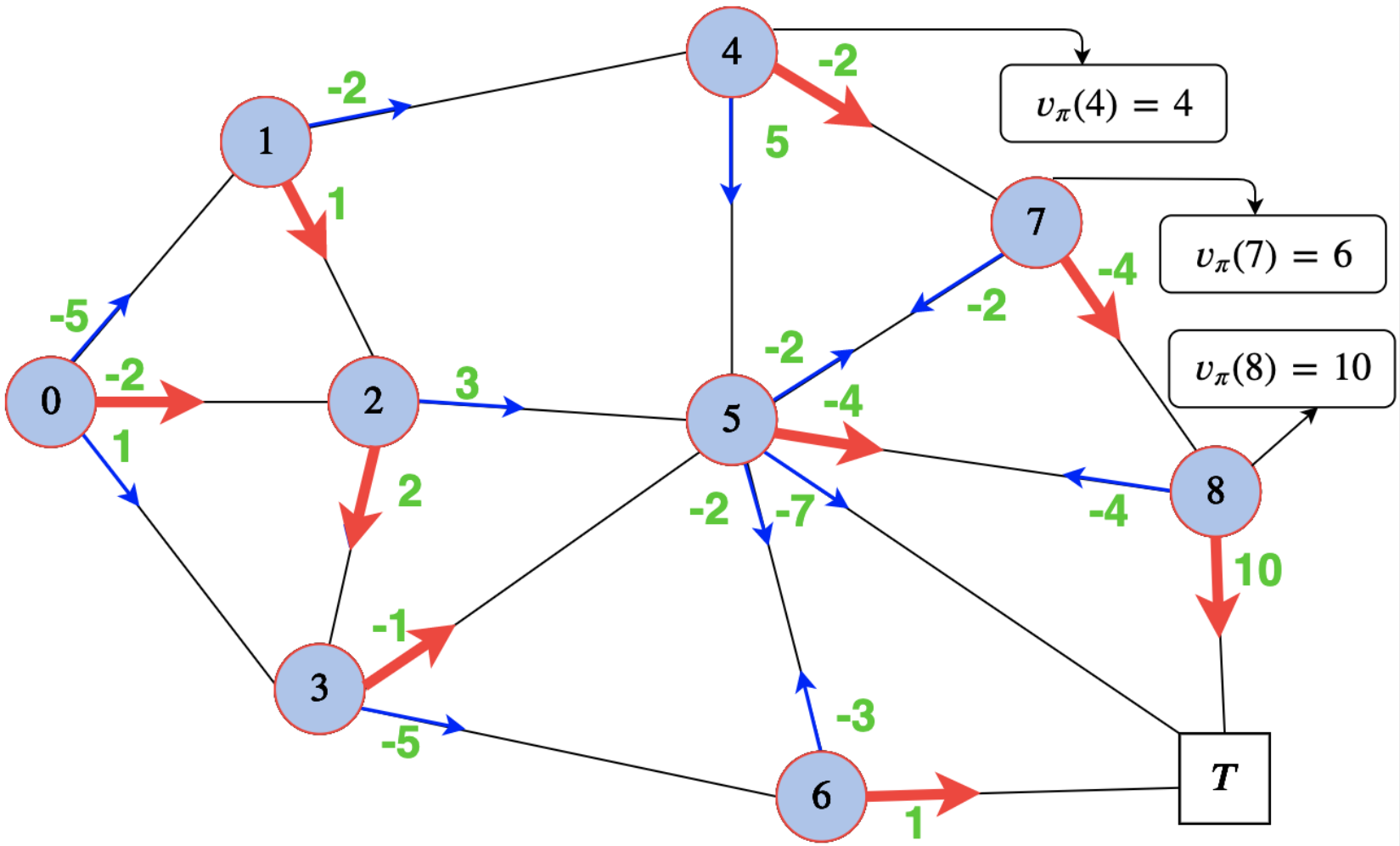
Bellman equation for value function $v_\pi(s)$



For **deterministic policy:** if $s \xrightarrow{a} s'$ then $v_\pi(s) = r(s, a, s') + v_\pi(s')$

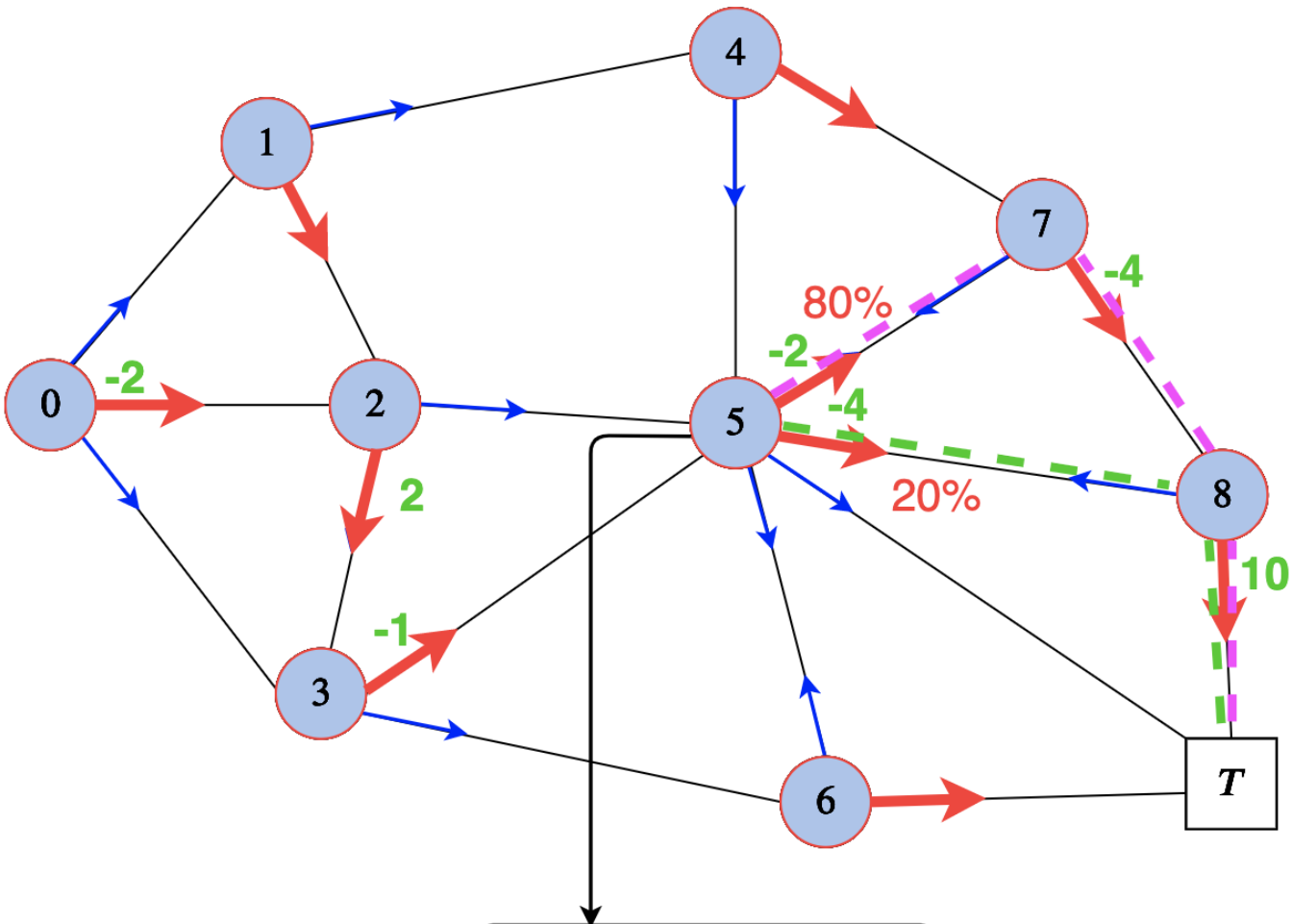
Bellman equation for value function $v_\pi(s)$

For **deterministic policy:** if $s \xrightarrow{a} s'$ then $v_\pi(s) = r(s, a, s') + v_\pi(s')$



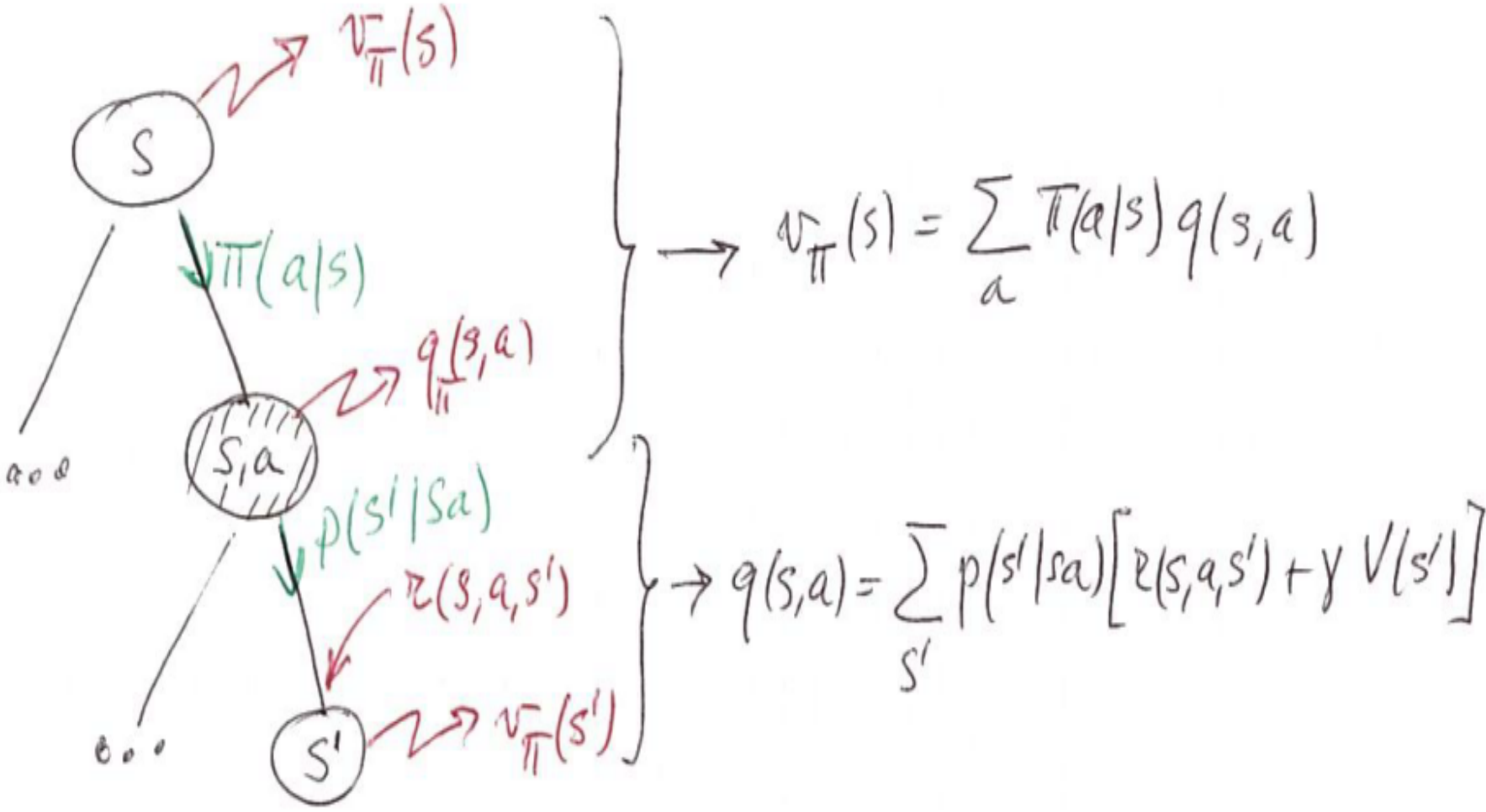
Bellman equation for value function $v_\pi(s)$

For **stochastic policy:**



$$v_\pi(5) = 0.8 \cdot 4 + 0.2 \cdot 6$$

Bellman equations (schematically): Backup Diagram



Bellman equation: Summary

- **Back-up**

$$v_\pi(s) = \sum_a \pi(a \mid s) q(s, a)$$

$$q(s, a) = \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma v(s') \right]$$

- **Combined**

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) \Big[r(s, a, s') + \gamma v_\pi(s') \Big]$$

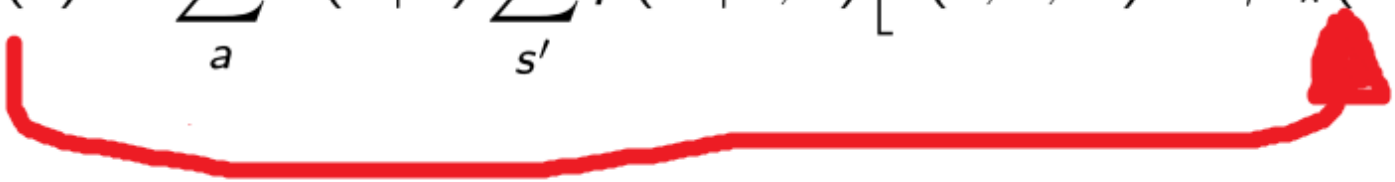
$$q_\pi(s, a) = \sum_{s'} p(s' \mid s, a) \Big[r(s, a, s') + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a') \Big]$$

Outline

Coding excursion 2: Bellman equations and Dynamic Programming

- If model for environment (MDP) is fully known then Bellman equations can be solved for given policy π;
  - **MDP:** states, actions,
  - **MDP:** transitions $p(s' \mid s, a)$, rewards $r(s, a, s')$
  - **Policy:** $\pi(a \mid s)$
- Iterative solution (fixpoint solution)

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) \Big[r(s, a, s') + \gamma v_\pi(s') \Big]$$



Coding excursion 2: Bellman equations and Dynamic Programming

Iterative solution (fixpoint solution)

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} p(s' \mid s, a) \Big[r(s, a, s') + \gamma v_\pi(s') \Big]$$

Pseudo code

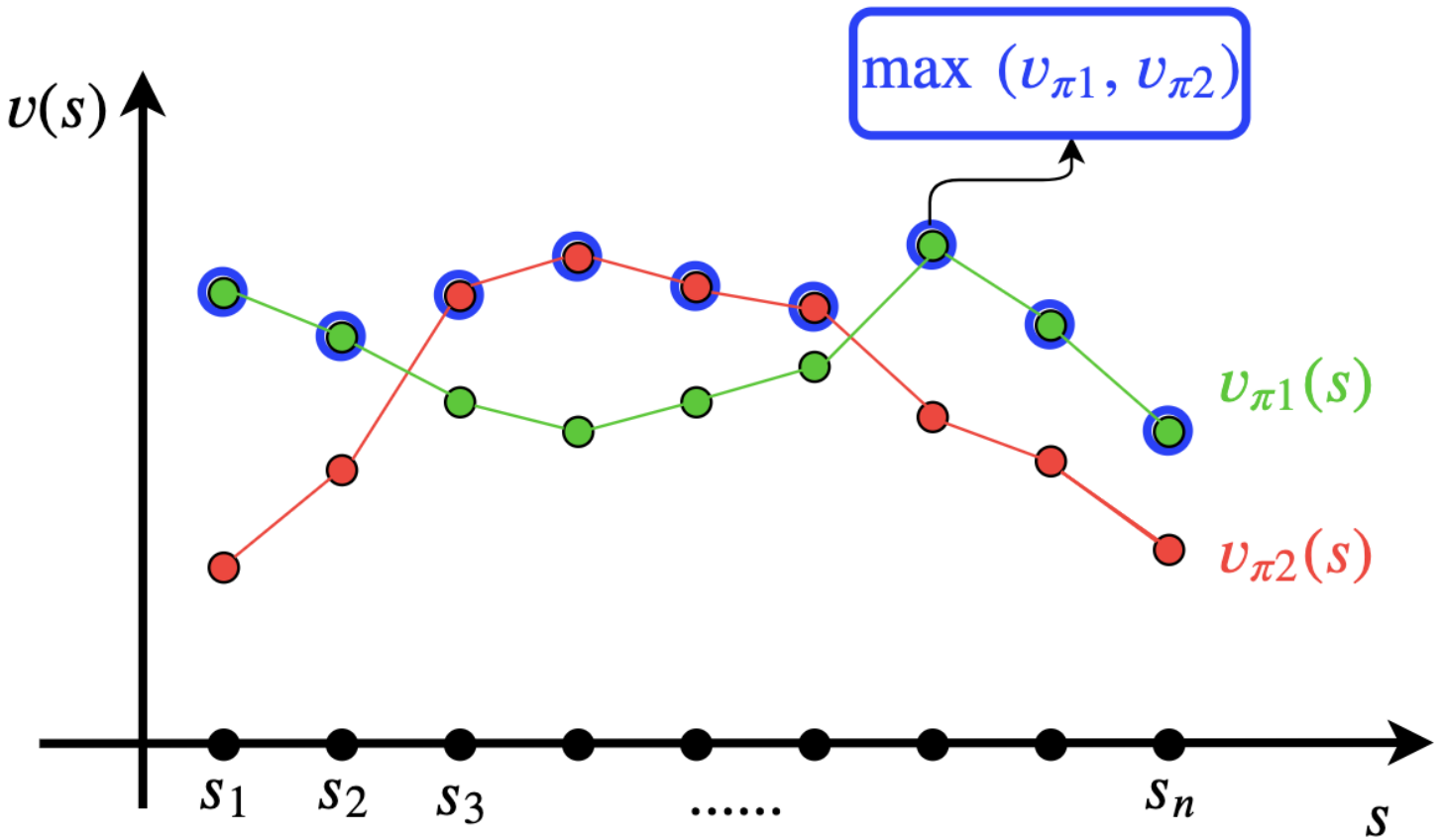
- Initialise $v_\pi(s)$ (zero or randomly) for all states s;
- Repeat until convergence:
  - Pick a random state s
  - Compute right-hand side of equation using policy π and MDP-parameters;
  - Update $v_\pi(s)$

Outline

Optimal value functions

The optimal value functions are defined by the **pointwise maximum over policies**:

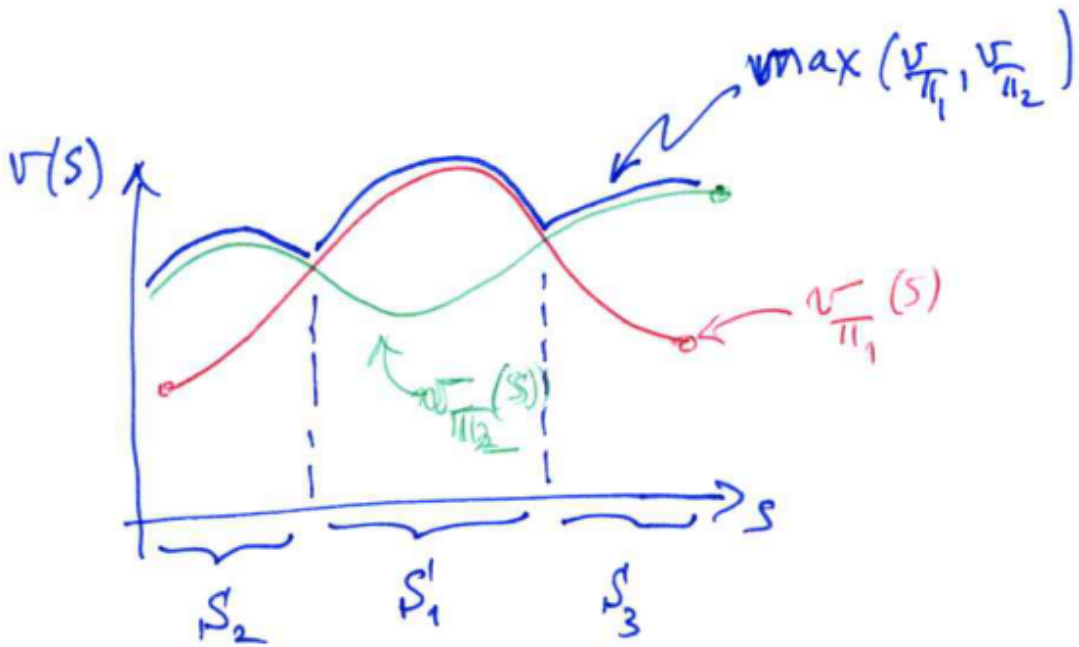
$$\forall s, a: \quad v^*(s) := \max_\pi v_\pi(s) \qquad \text{and} \quad q^*(s,a) := \max_\pi q(s,a)$$



Optimal policy π^*

There exists an **optimal policy** π^* such that its value functions corresponds to the optimal value functions:

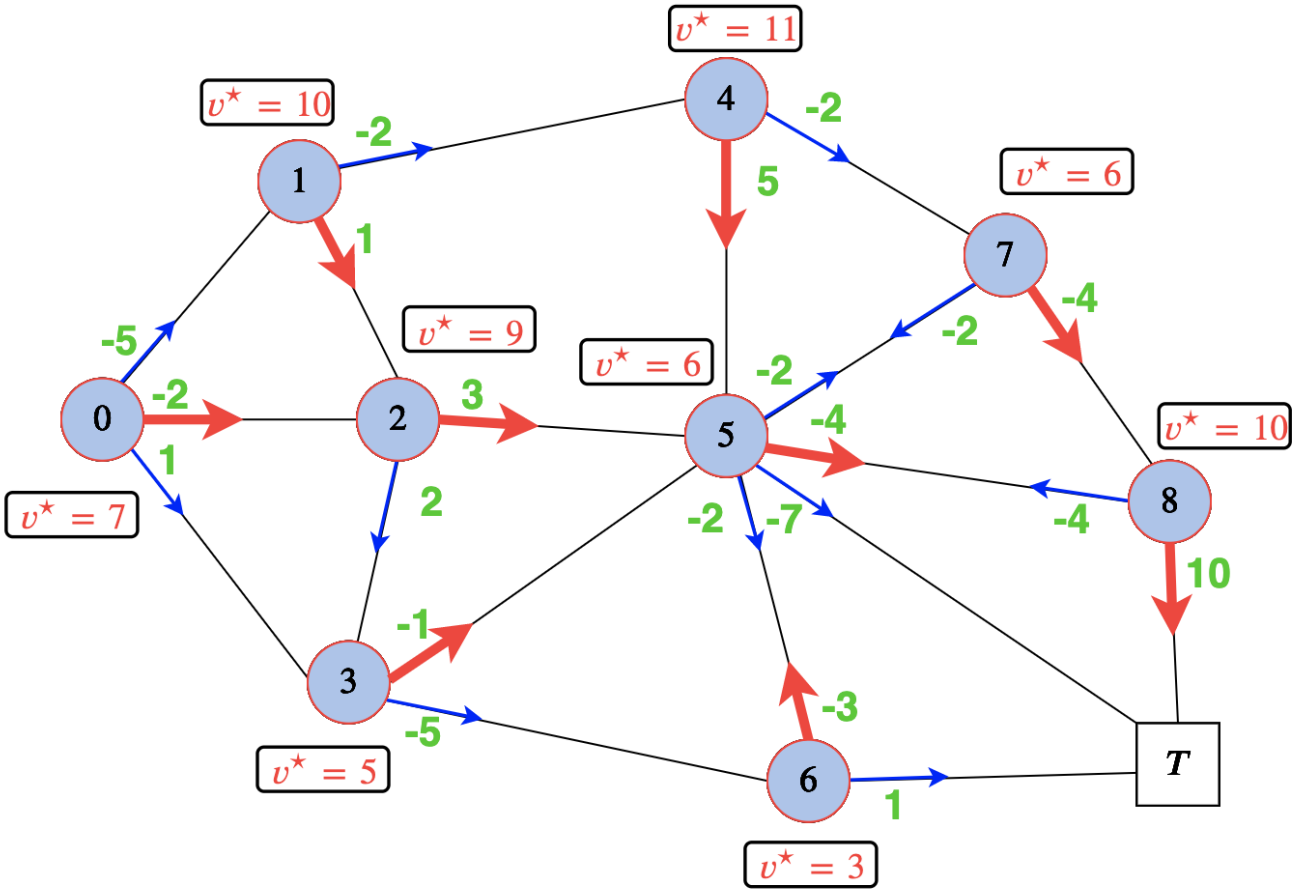
$$v_{\pi^*}(s) = v^*(s) \qquad \text{and} \qquad q_{\pi^*}(s,a) = q^*(s,a)$$



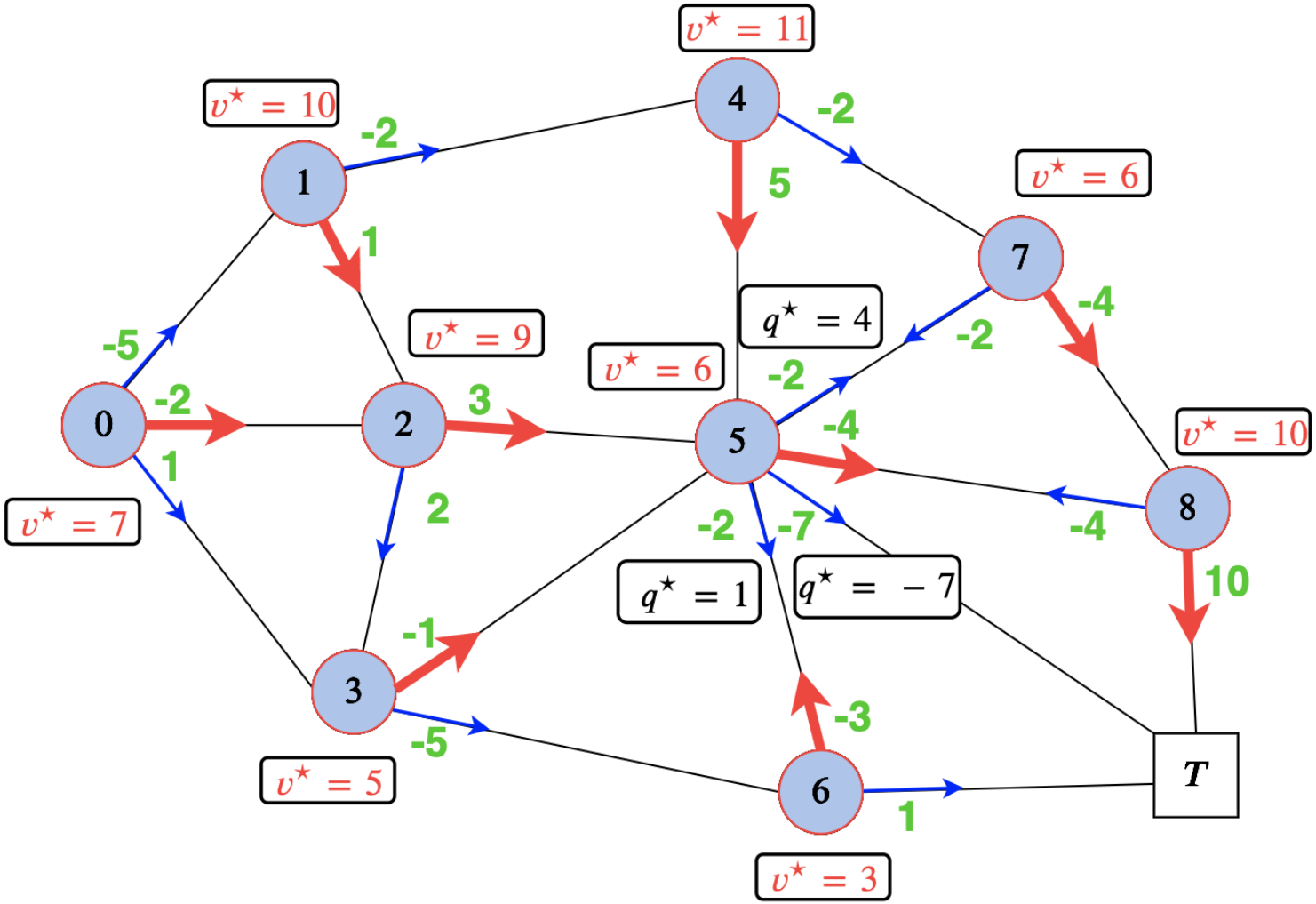
$$\pi^*(s) = \begin{cases} \pi_1(s) & \text{if } s \in S_2 \\ \pi_2(s) & \text{if } s \in S_1 \cup S_3 \end{cases}$$

Optimal policy

An optimal policy π^* assign optimal values v^* to all states

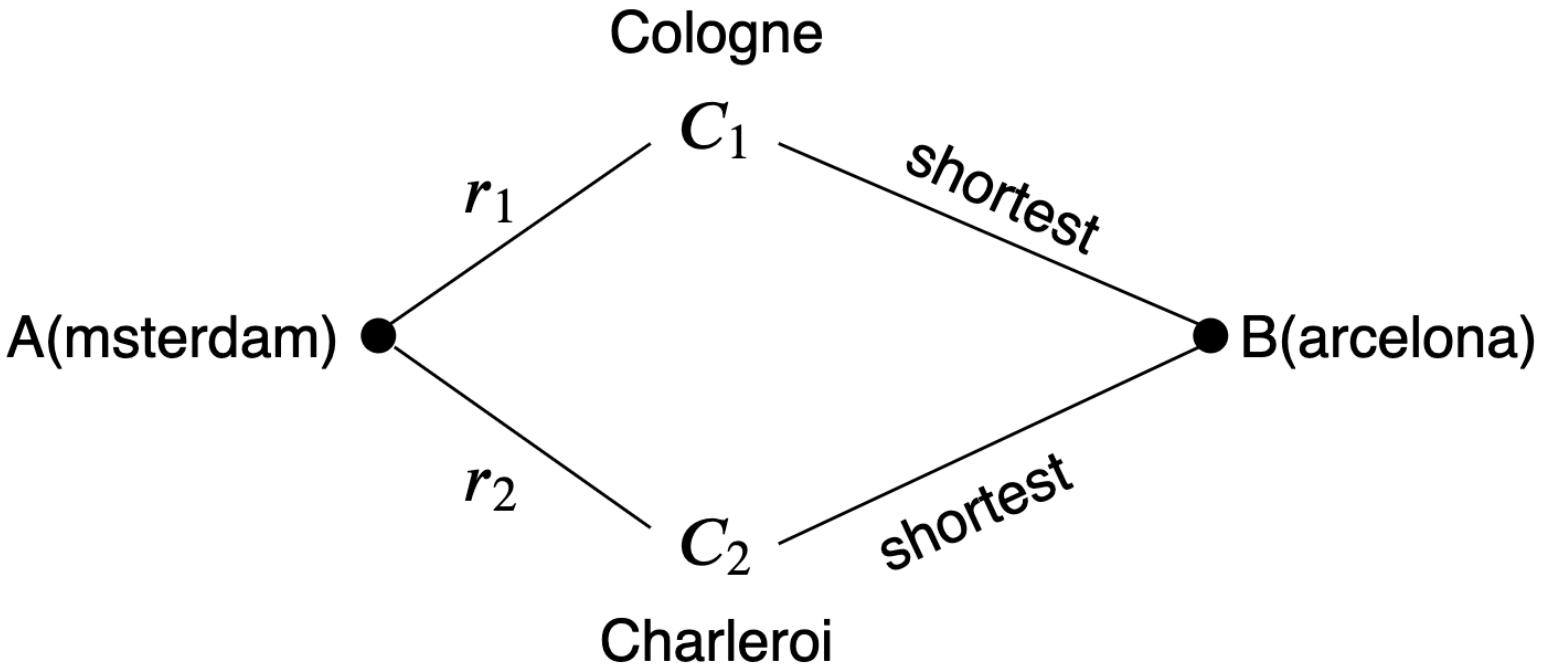


Bellman Optimality conditions



$$v^*(s) = \max_a q^*(s, a)$$

Bellman Optimality equation: Travel Distance Analogy



$$d^*(A, B) = \min_{C_i} \{r_i + d^*(C_i, B)\}$$

Bellman optimality conditions (for deterministic transitions)

- Travel Analogy: Shortest distance paths:

$$d^*(A, B) = \min_{C_i} \{r_i + d^*(C_i, B)\}$$

- Deterministic transitions: $s \xrightarrow{a} s'$

$$v^*(s) = \max_a \left\{ r(s, a, s') + v^*(s') \right\}$$

$$q^*(s, a) = r(s, a, s') + v^*(s') = r(s, a, s') + \max_{a'} q^*(s', a')$$

Bellman optimality equations (General form)

- **Optimal state value function**

$$v^*(s) = \max_a q^*(s, a)$$

- **Optimal state-action value function**

$$q^*(s, a) = \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma v^*(s') \right]$$

- **Combined**

$$v^*(s) = \max_{a \in A} \sum_{s'} p(s' \mid s, a) \Big[r(s, a, s') + \gamma v^*(s') \Big]$$

$$q^*(s, a) = \sum_{s'} p(s' \mid s, a) \Big[r(s, a, s') + \gamma \max_{a' \in A} q^*(s', a') \Big]$$

Backup Diagram for Bellman Optimality Equations

Optimize over actions!

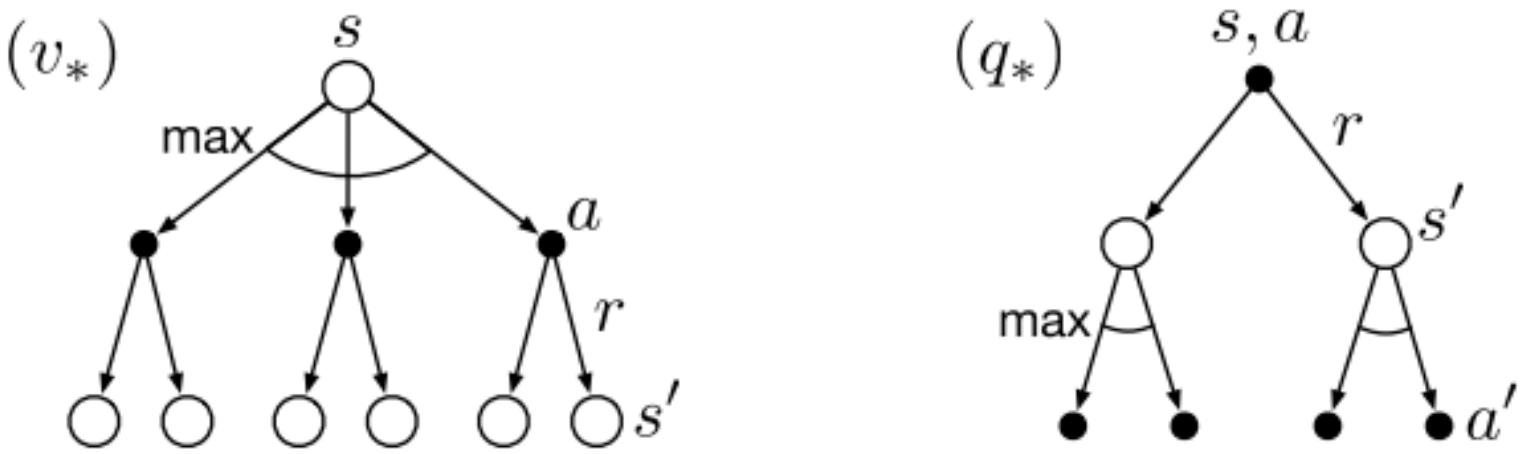


Figure 3.5: Backup diagrams for v_* and q_*

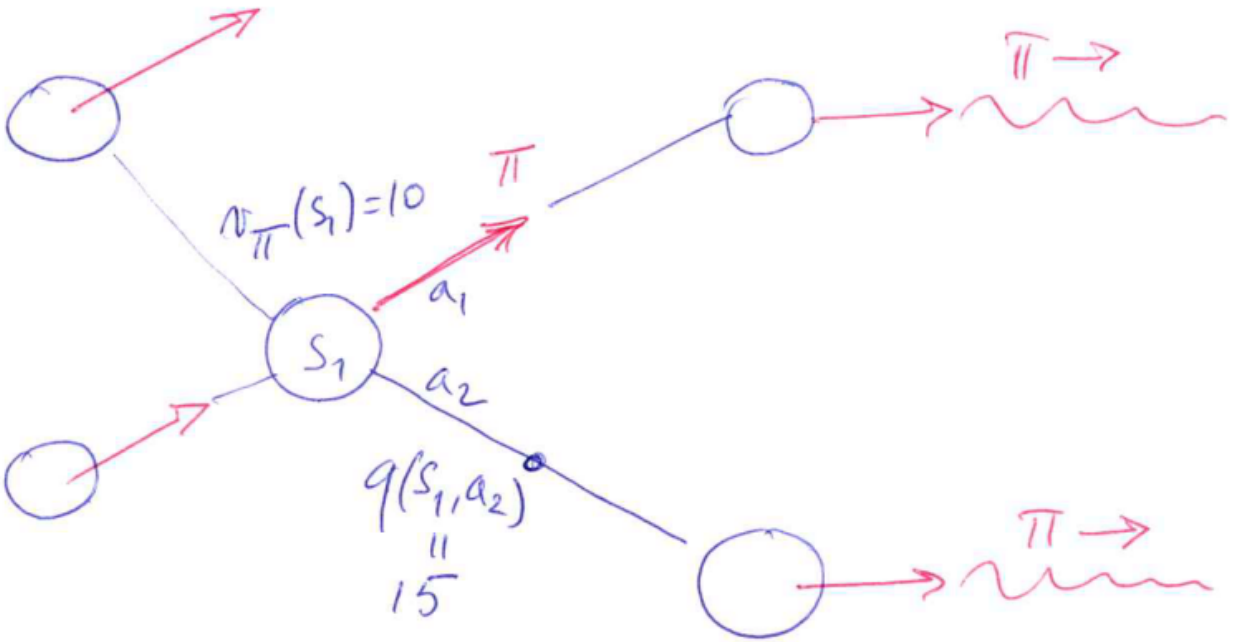
Bellman Optimality Conditions

- v^* and q^* satisfy a set of (non-linear) equations analoguous to Bellman equations for v_π and q_π;
- these non-linear equations do **not refer** explicitly to the optimal policy;
- As a consequence we can find the optimal policy by first solving these equations to find $v^*(s)$ (and $q^*(s,a)$) and then use **greedification** to determine the corresponding optimal policy π^*:

$$\forall s: \quad a_{opt} = \arg\max_a q^*(s,a)$$

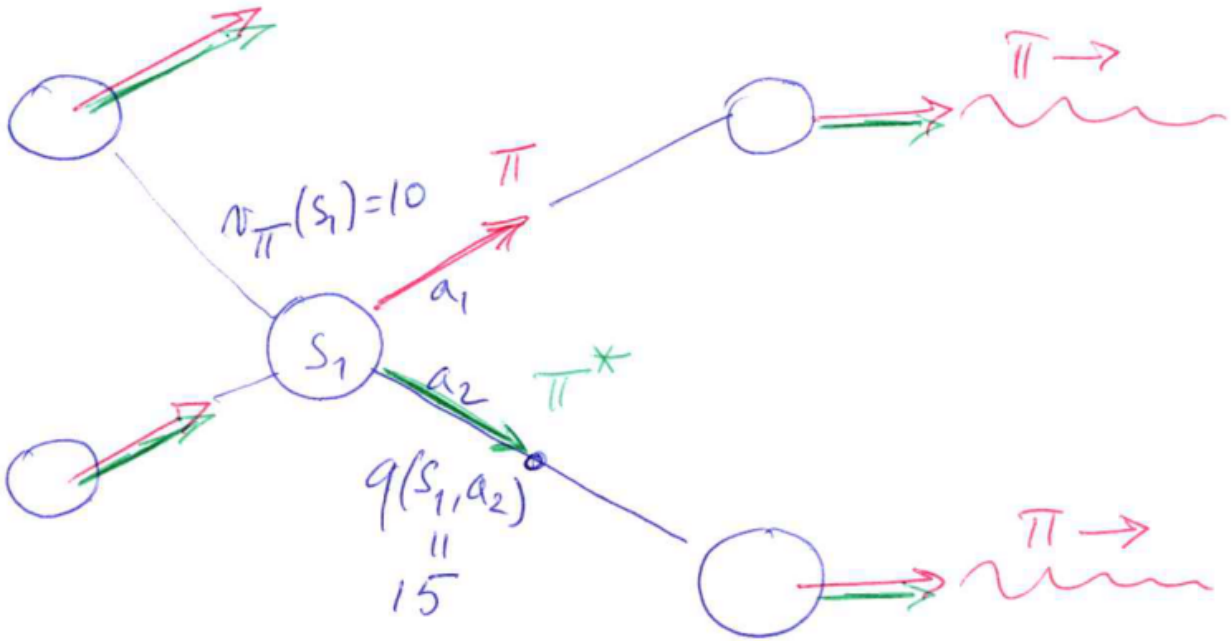
Outline

Incremental policy improvement



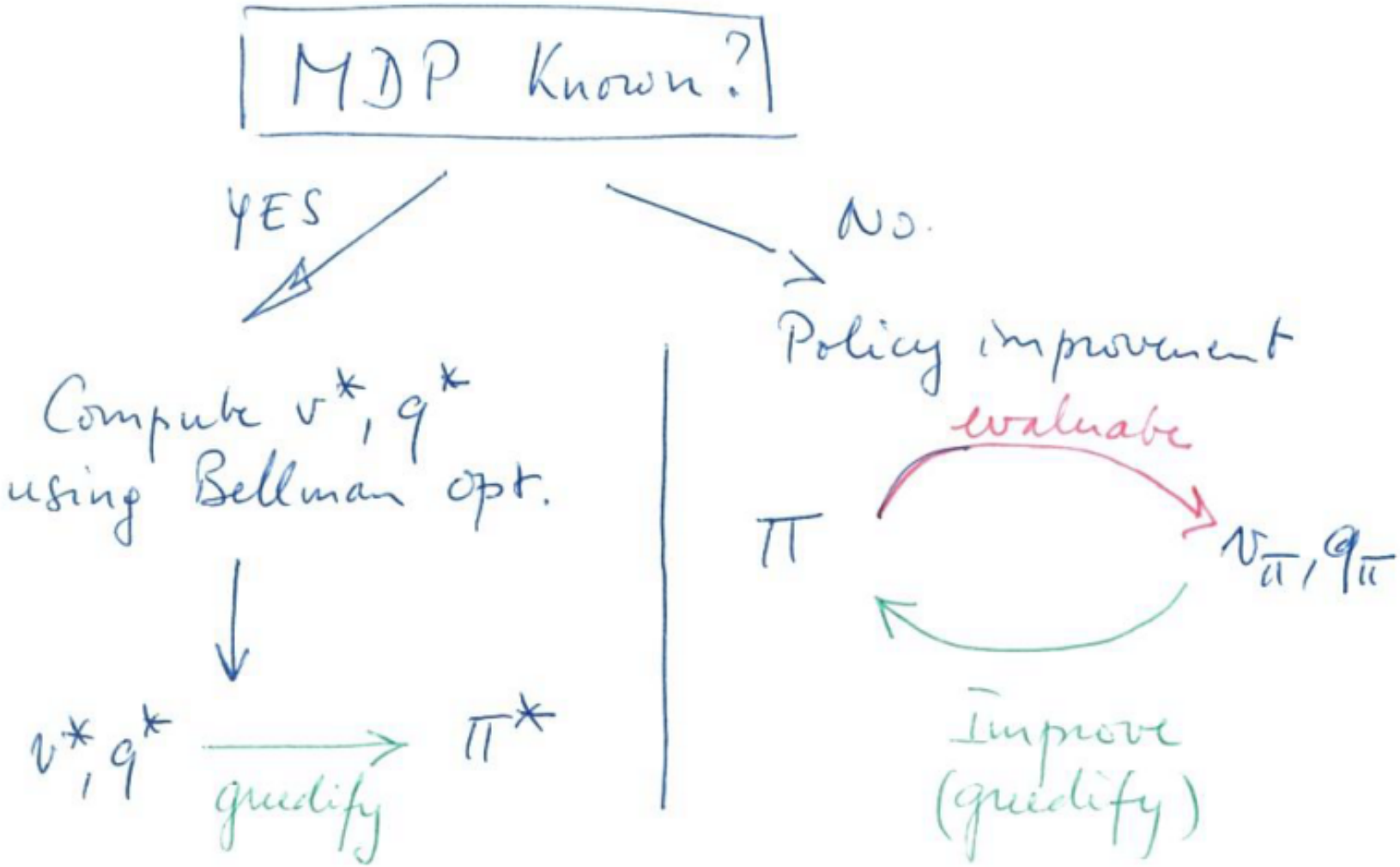
Room for improvement: In state s_1 we have $v_\pi(s_1) < q_\pi(s_1, a_2)$, hence switch from action a_1 to a_2.

Policy improvement for deterministic policy



$$\pi \longrightarrow \pi^* \qquad \text{where} \qquad \pi^*(s) = \begin{cases} a_2 & \text{if} \quad s = s_1 \\ \pi(s) & \text{if} \quad s \neq s_1 \end{cases}$$

Finding the optimal policy π^*



Outline

Coding Excursion 3: Finding an optimal policy

Bellman optimality equations: (General form)

- **Optimal state value function**

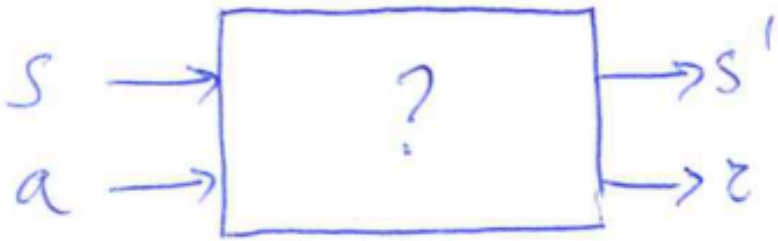
$$v^*(s) = \max_a q^*(s, a)$$

- **Optimal state-action value function**

$$q^*(s, a) = \sum_{s'} p(s' \mid s, a) \left[r(s, a, s') + \gamma v^*(s') \right]$$

Outline

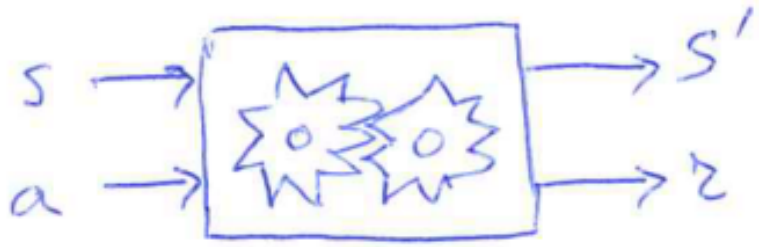
Model-free vs. model-based



Model-free

Black box

LEARNING



Model-based.

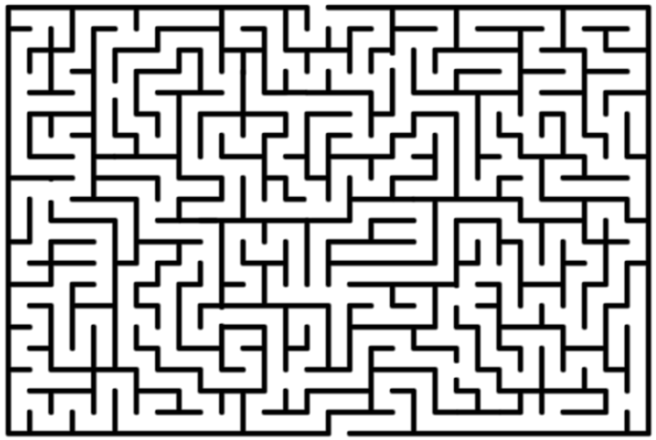
White box.

PLANNING

(Dyn. progr)

Planning (model-based) versus Learning (model-free)

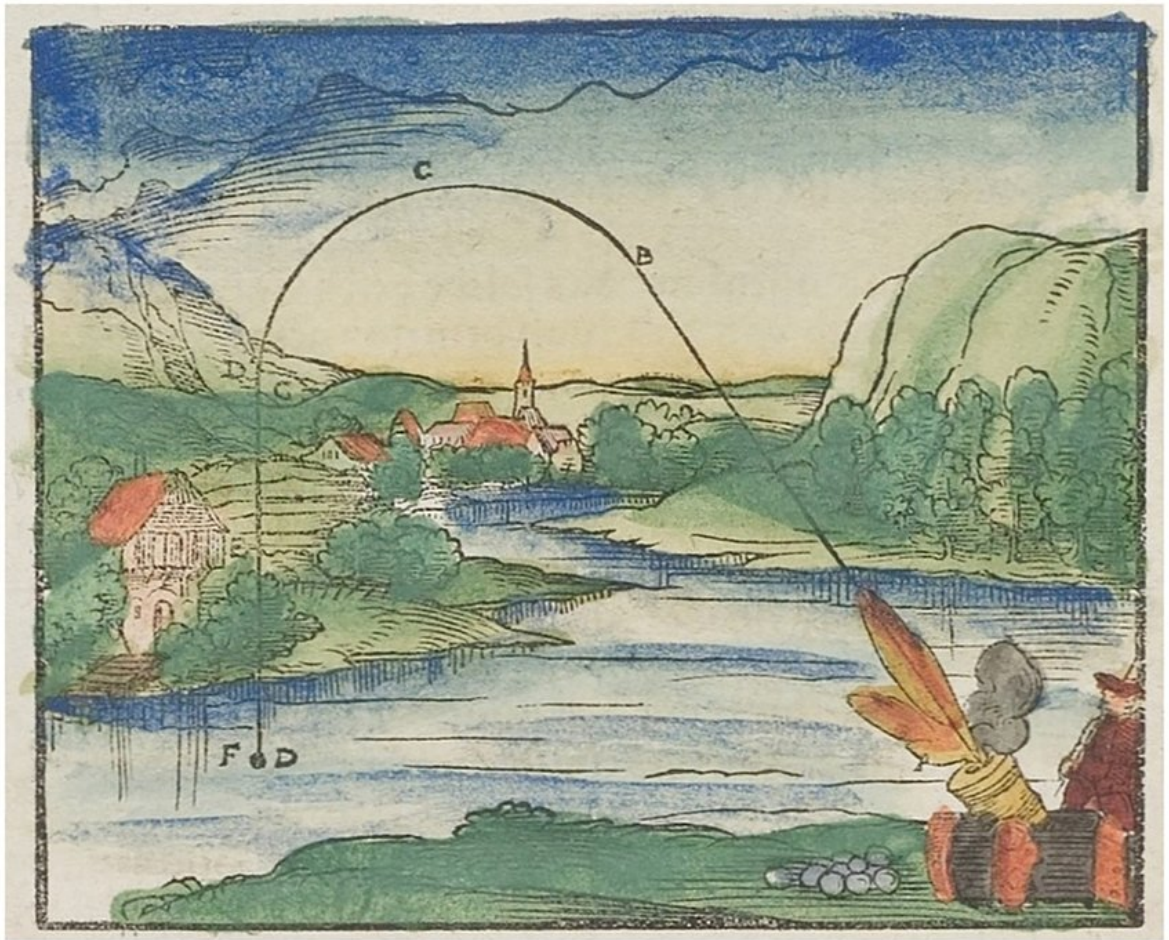
Model-based



Model-free

Model-free versus model-based

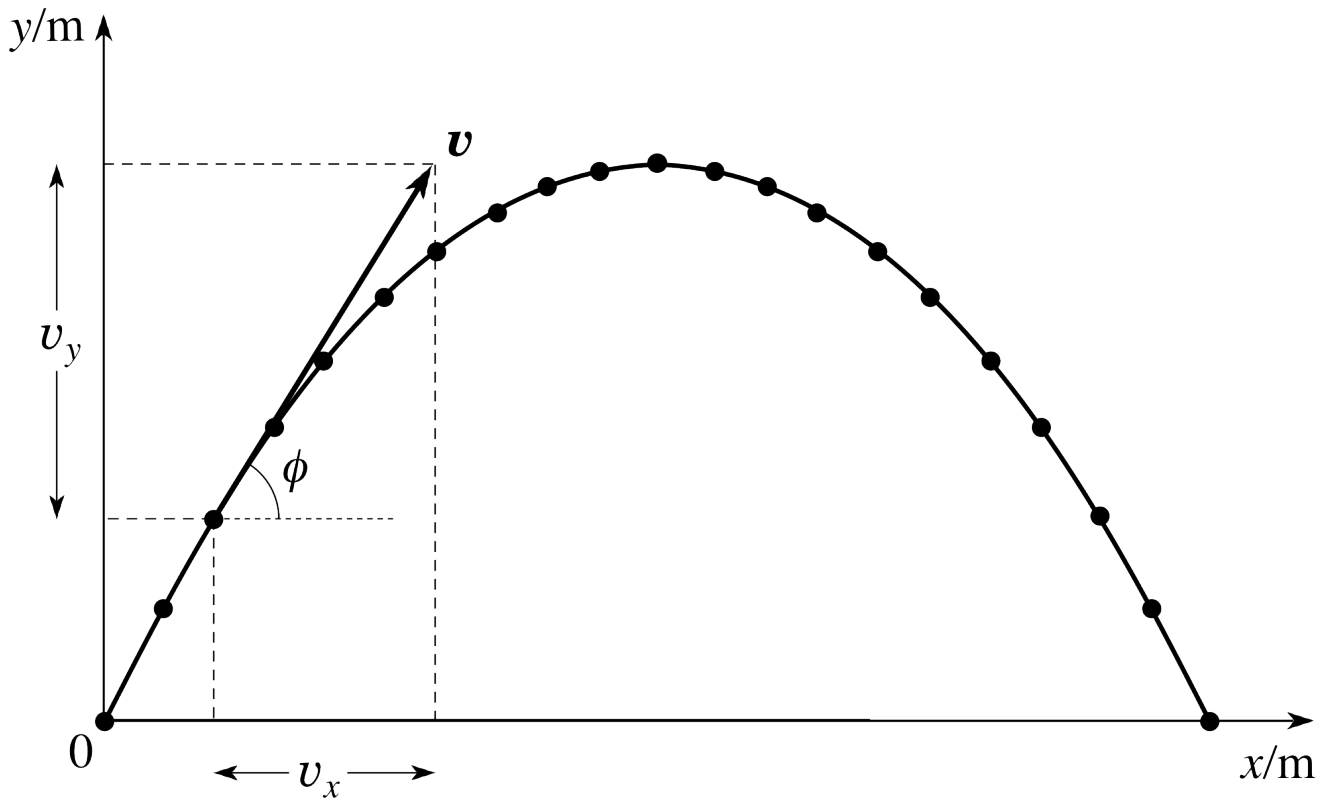
Model-free: Ballistics in the Middle Ages: Trial and Error



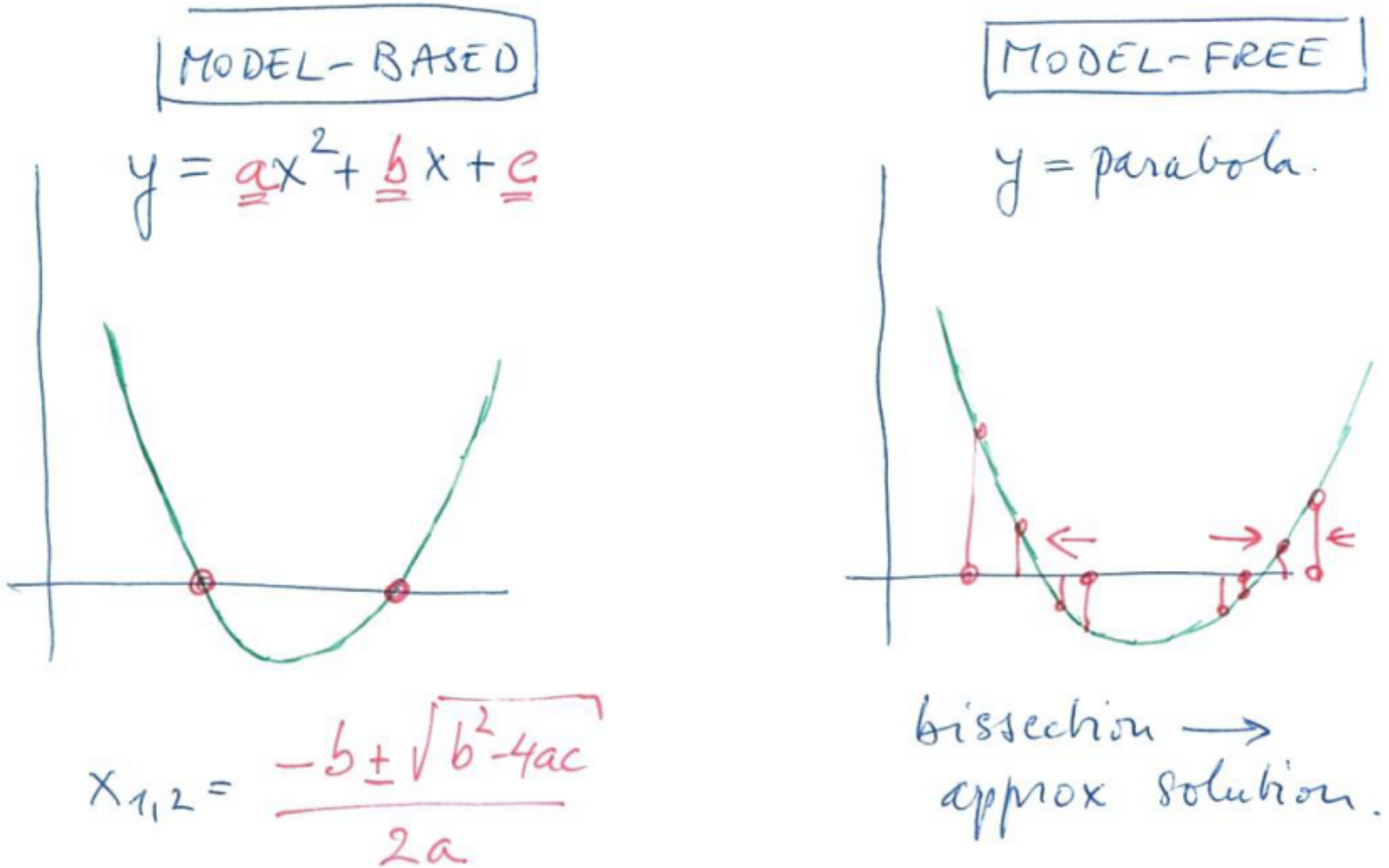
Model-free versus model-based

Model-based:

- Newton's laws for ballistics;
- Friction, Coriolis acceleration, etc.



Model-based vs. Model-free
Analogy: Computing zeros for quadratic equation



Model-based vs model-free

- **Model-based:** the MDP $= (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ is completely specified;
  - Solve the Bellman equations!

- **Model-free:** only **direct experience**, i.e. sample paths (states, actions and rewards) are given. Put differently, only experience-based information is given!
  - Random search but Bellman equations allow to propagate values!

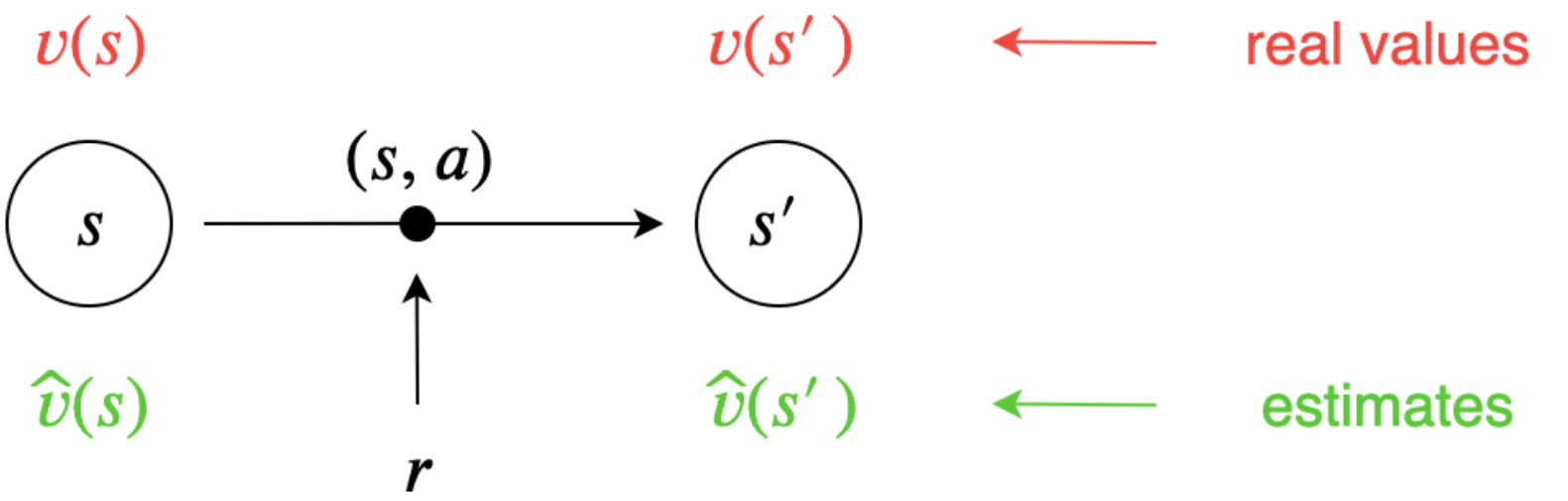
Outline

Solving model-free RL problems

- The agent has **no prior knowledge** about states, actions, rewards, transitions!
- By **acting in the world**, the agent gains experience.
  An experience can be expressed as a 4-tuple: (s, a, r, s')
- Over time the agent collects a list of experiences that he uses to find value functions (and corresponding policy) . . . HOW?
- **Key observation:** Bellman equations link values in neighbouring states along paths!

Temporal differencing (TD): Use Bellman eqs to propagate values!

Temporal-differencing: Exploiting Bellman eqs.



time t

$\hat{v}(s)$

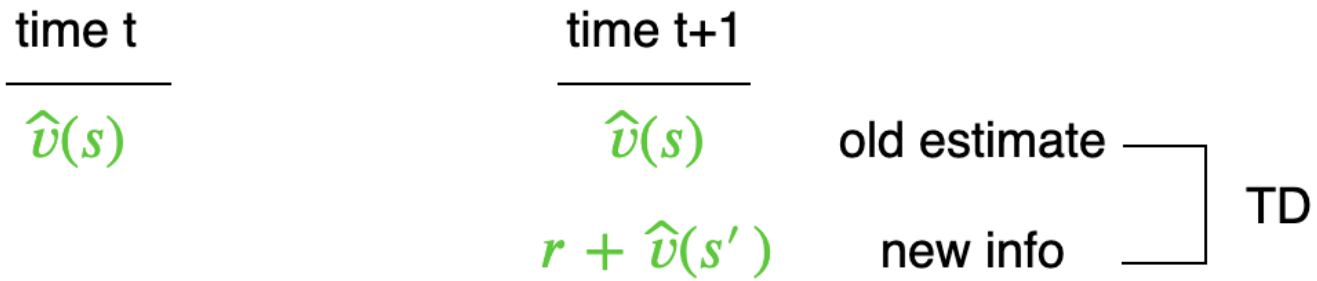
time t+1

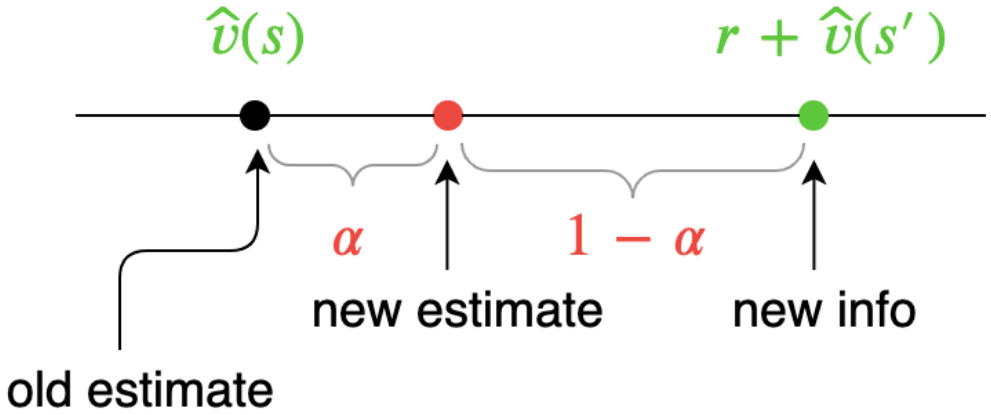
$\hat{v}(s)$ old estimate

$r + \hat{v}(s')$ new info

TD

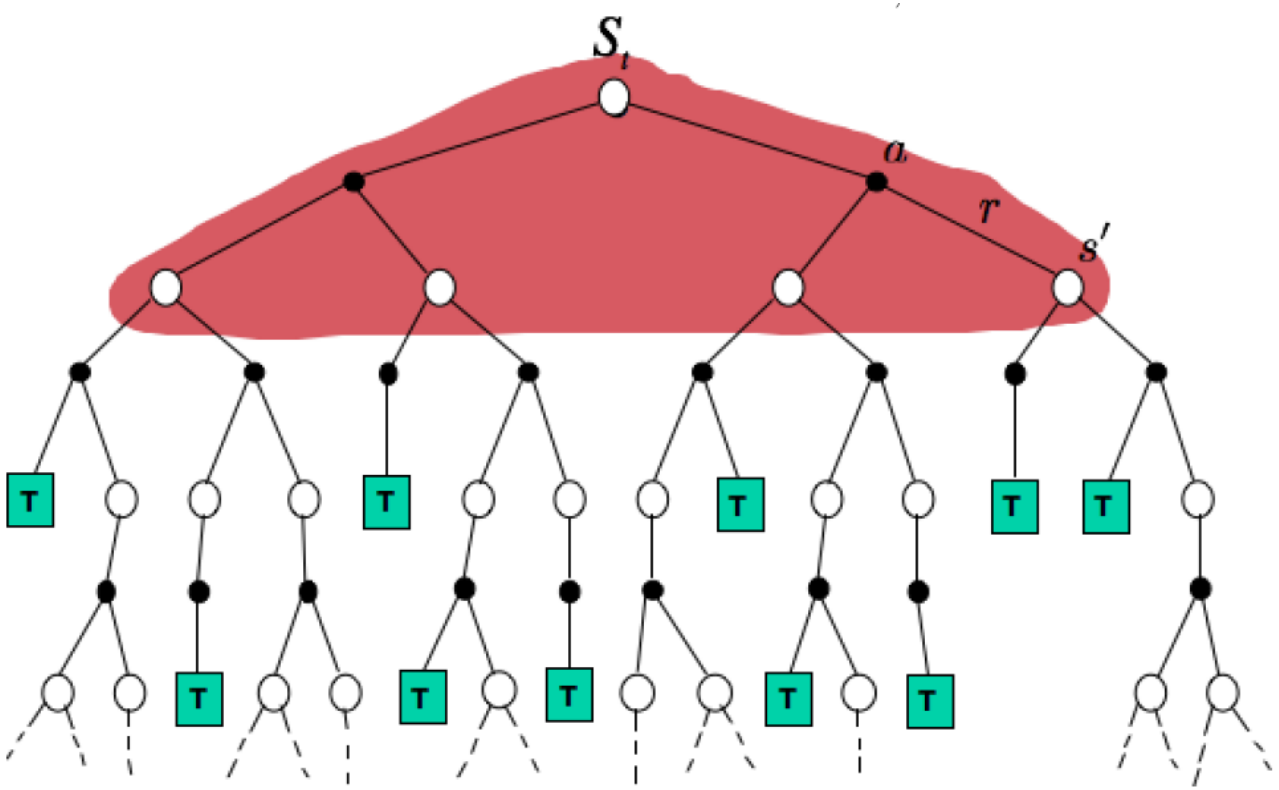
Temporal-differencing: Exploiting Bellman eqs.





$$\hat{v}_{\text{new}}(s) = (1 - \alpha)\hat{v}_{old}(s) + \alpha\left(r + \hat{v}(s')\right)$$

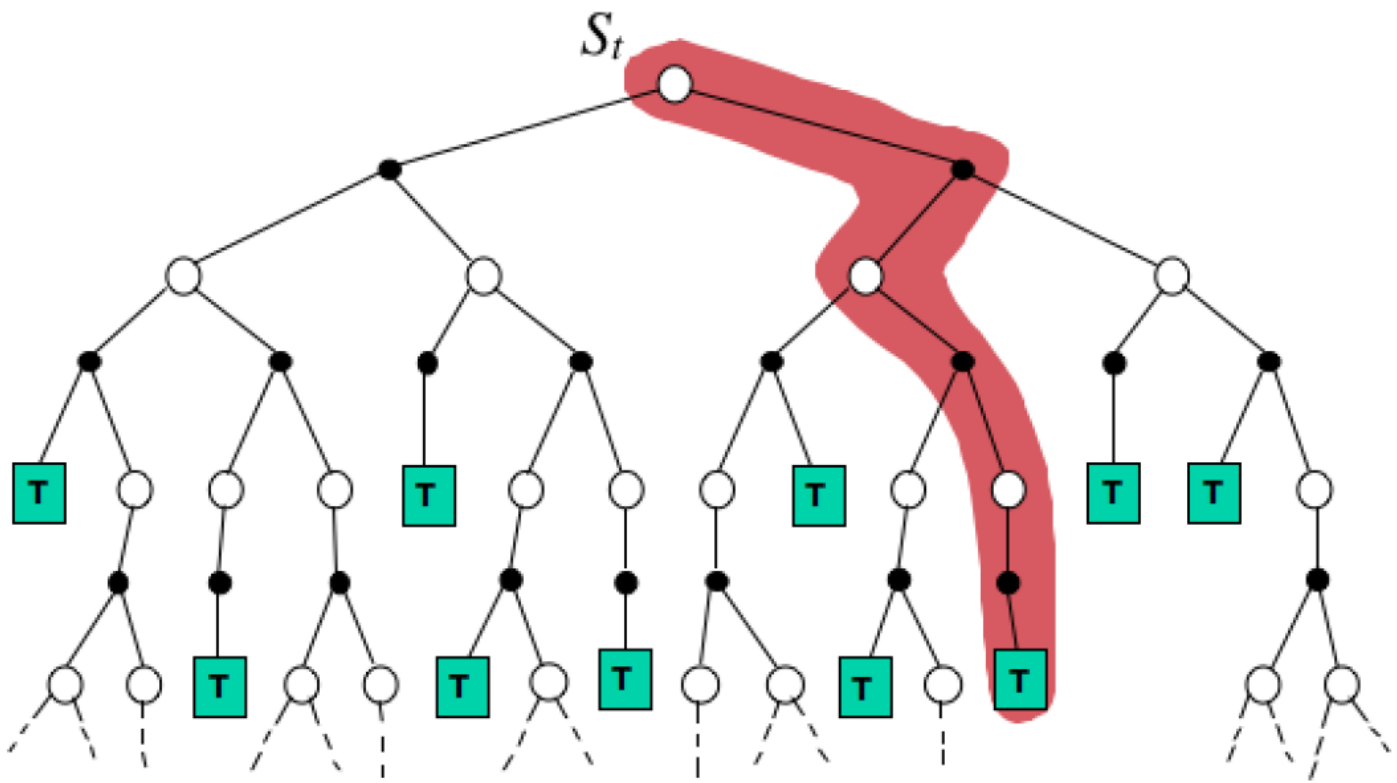
Dynamic Programming (DP): Backup diagram



Monte-Carlo (MC): Backup diagram

Update rule: $V(S_t) \leftarrow (1-\alpha)V(S_t) + \alpha G_t$

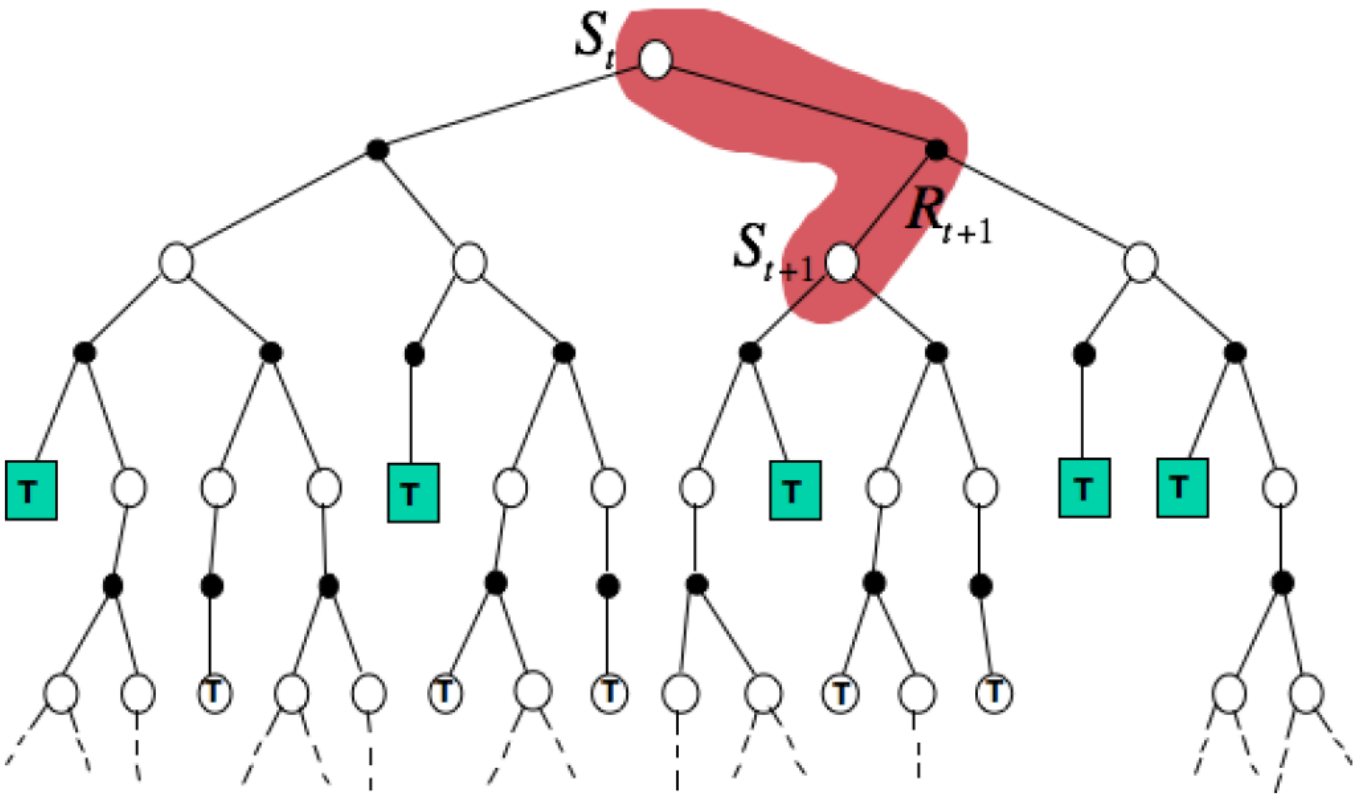
$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$



Temporal Difference (TD): Backup diagram

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

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



Caveats when addressing model-free RL problems

1. **We need to compute $q(s, a)$ rather than $v(s)$**
   - To **improve a policy**, we need to know for each state s which action a results in the highest $q(s, a)$:

     construct $\pi : s \mapsto a_{max}$ where $q(s, a_{max}) = \max_{a'} q(s, a')$

     - **Model-based:** $q(s, a)$ can be computed from $v(s)$:

       $$q(s, a) = r(s, a, s') + v(s')$$

     - **Model-free:** $q(s, a)$ needs to be estimated explicitly!
2. **Keep exploring!**
   - Balance exploration versus exploitation
   - E.g. ϵ-greedy, soft-max, etc.

Applying TD to model-free RL problems

Recall: Model-free, hence focus on value function $q(s, a)$

1. **SARSA**: Evaluating a given policy π
   - Bellman: $q_\pi(s, a) = r(s, a, s') + \sum_{a'} \pi(a' \mid s') q_\pi(s', a')$
   - SARSA update rule (sample-based):

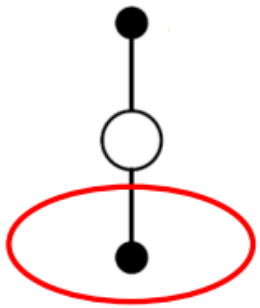
$$q(s, a) \leftarrow (1 - \alpha) q_\pi(s, a) + \alpha(r(s, a, s') + q(s', a'))$$

2. **Q-learning:** Estimating the **optimal** value function $q^*(s, a)$
   - Bellman: $q^*(s, a) = r(s, a, s') + \max_{a'} q^*(s', a')$
   - Q-learning update rule (sample-based):

$$q^*(s_t, a_t) \leftarrow (1 - \alpha) q^*(s_t, a_t) + \alpha(r(s, a, s') + \max_{a'} q^*(s', a'))$$

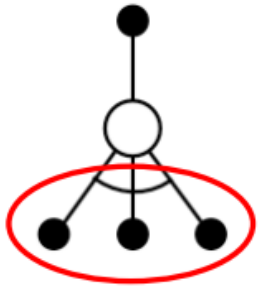
SARSA vs. Q-learning: On-Policy vs. Off-Policy

Sarsa:



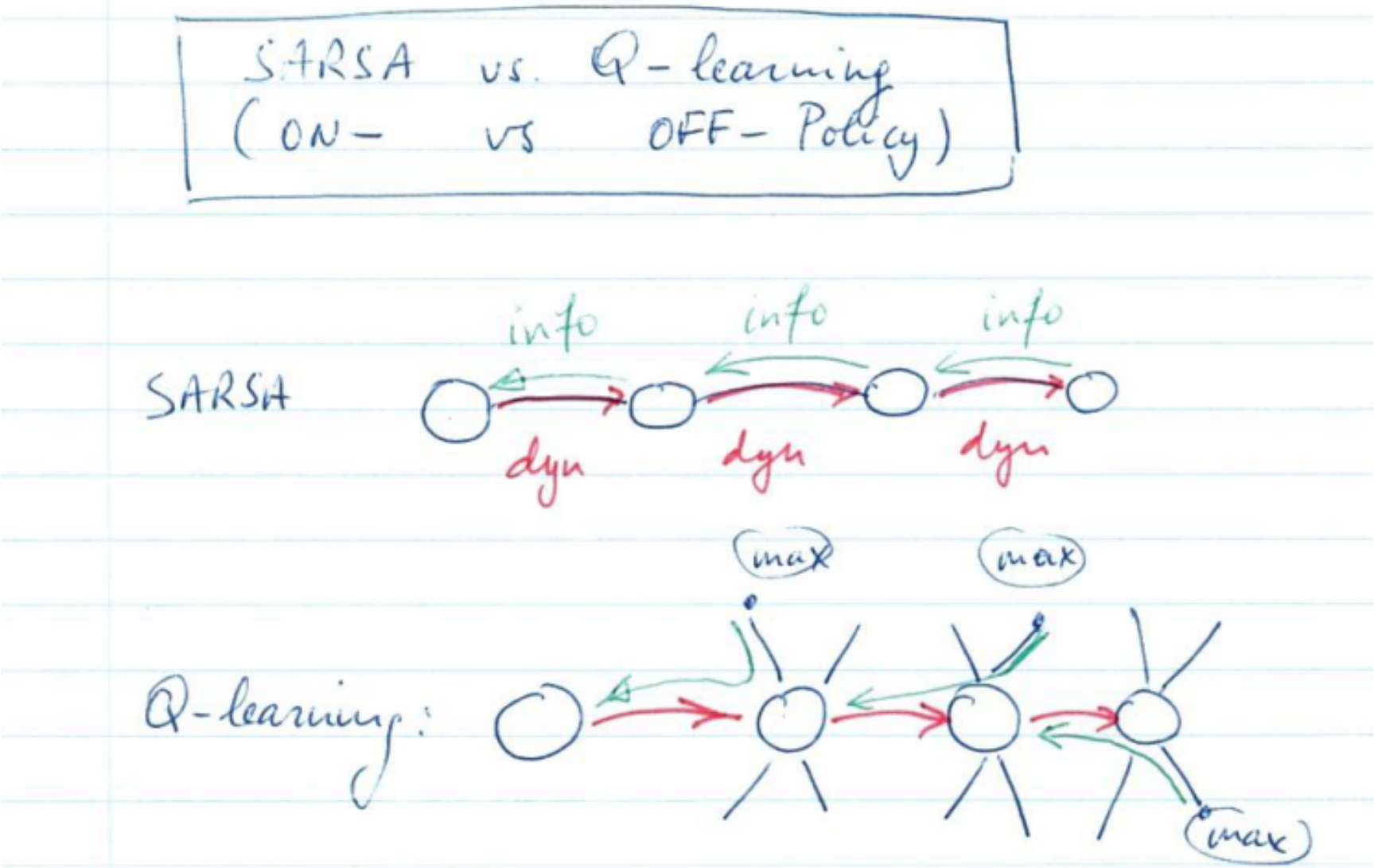
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Q-learning:



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

SARSA (on-policy) vs. Q-Learning (off-policy)



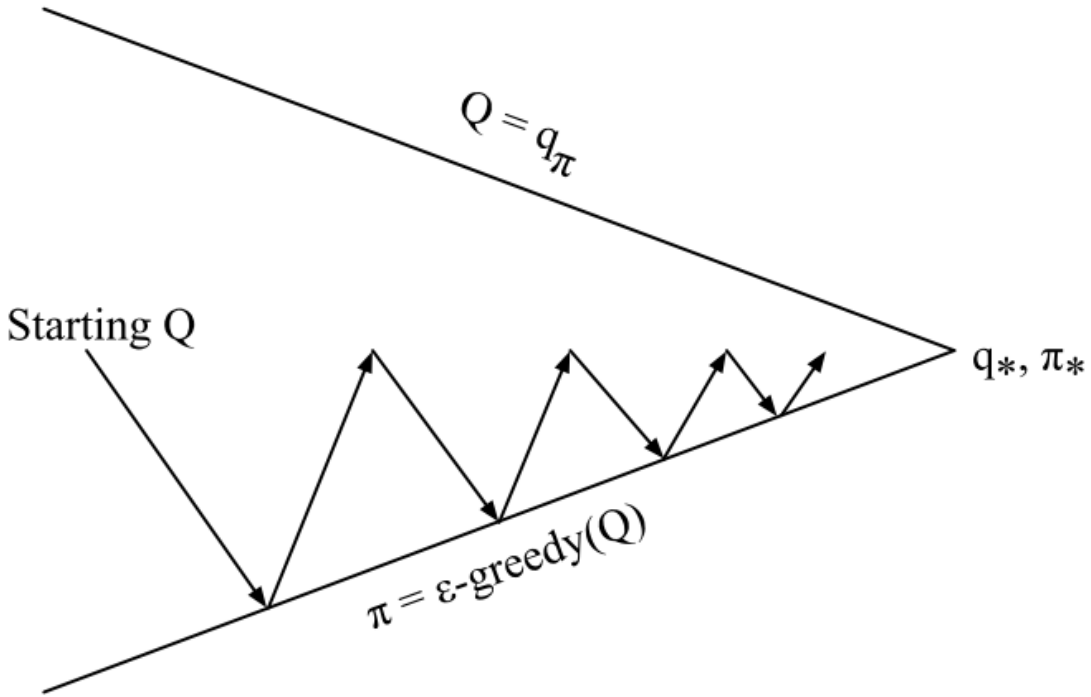
SARSA: Pseudo-code for model-free control

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

SARSA model-free control: Schematically

SARSA only evaluates the value function q_π for a given policy π, then use policy improvement to find a better policy. Repeat!



Every time-step:

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

Policy improvement ϵ-greedy policy improvement

Q-learning: Pseudo-code for off-policy TD control

Make TD off-policy: bootstrap with best action, not actual action:

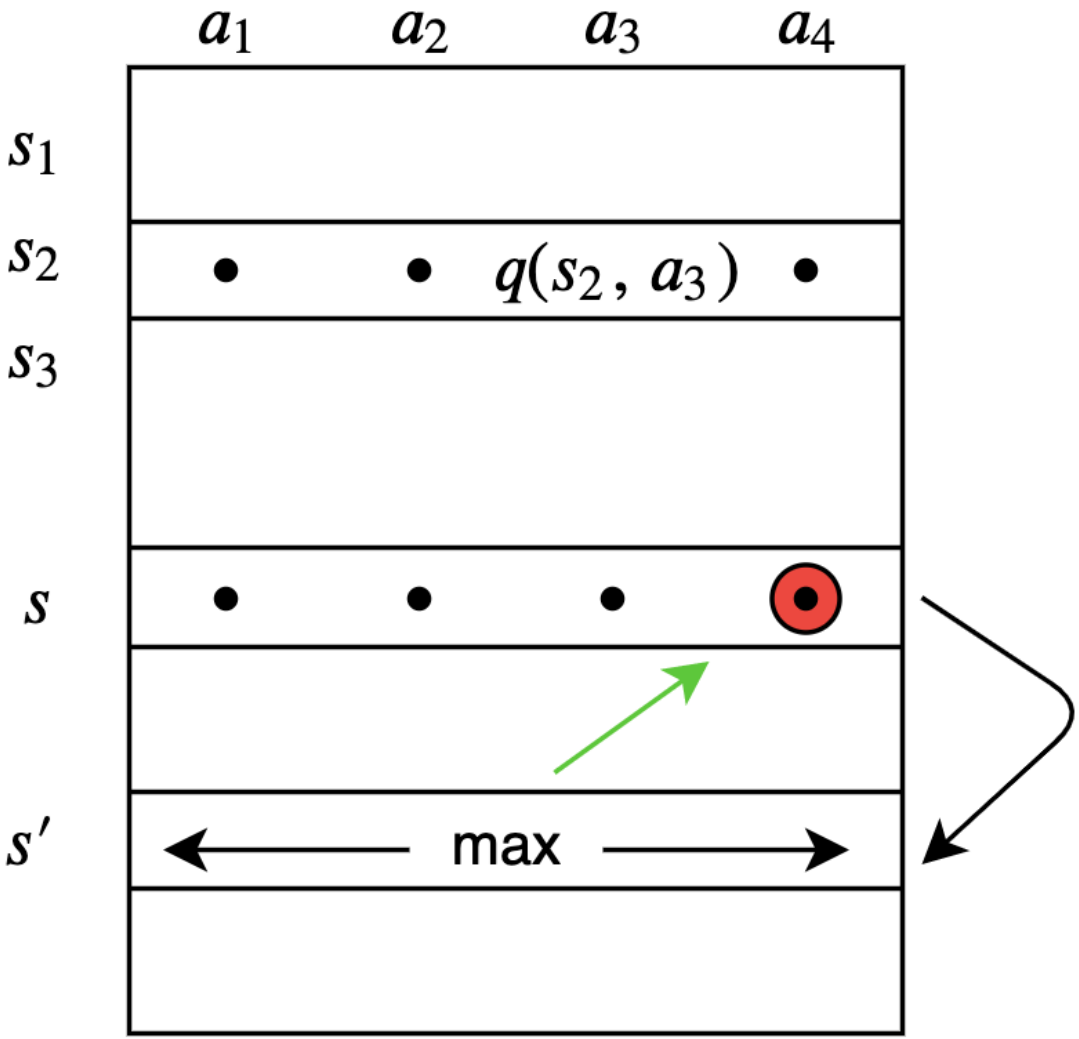
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state},$
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'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Outline

Coding excursion 4: Implement SARSA and Q-Learning



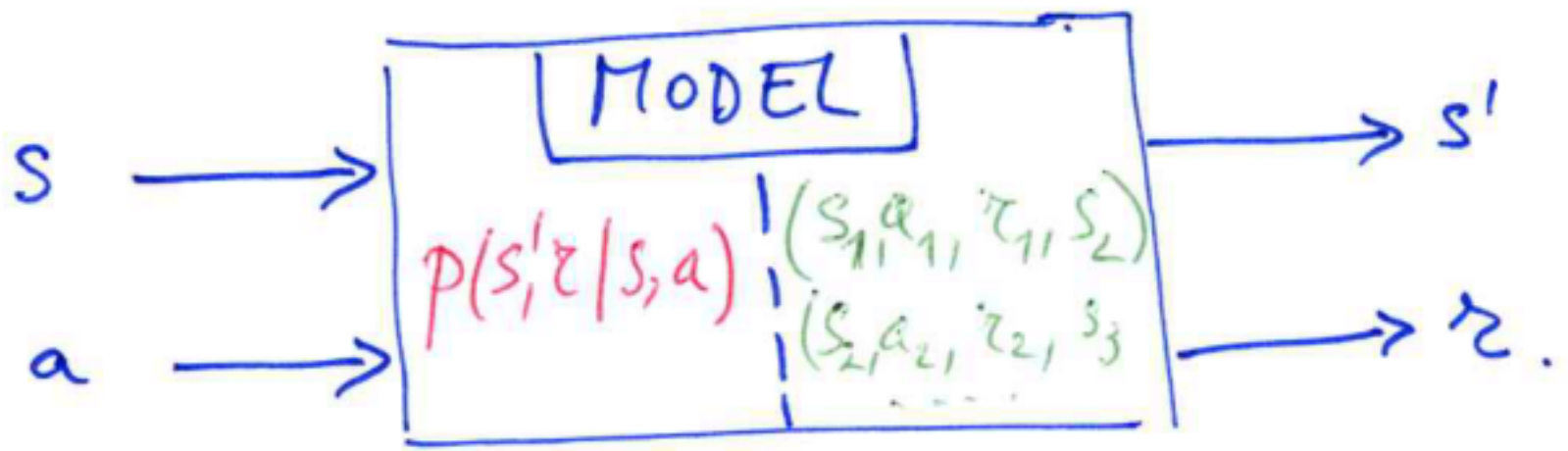
Outline

Dyna-Q; Integrating planning and learning

Model: tells agent what will happen next ...

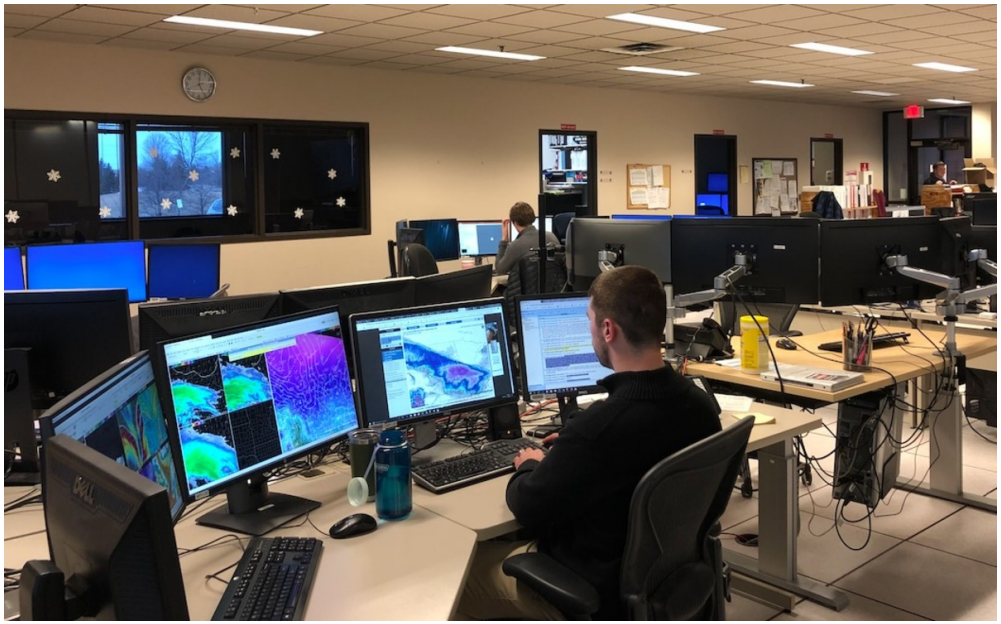
- **Model-based**: planning
- **Model-free**: learning

Distributional vs. Sample-based Model



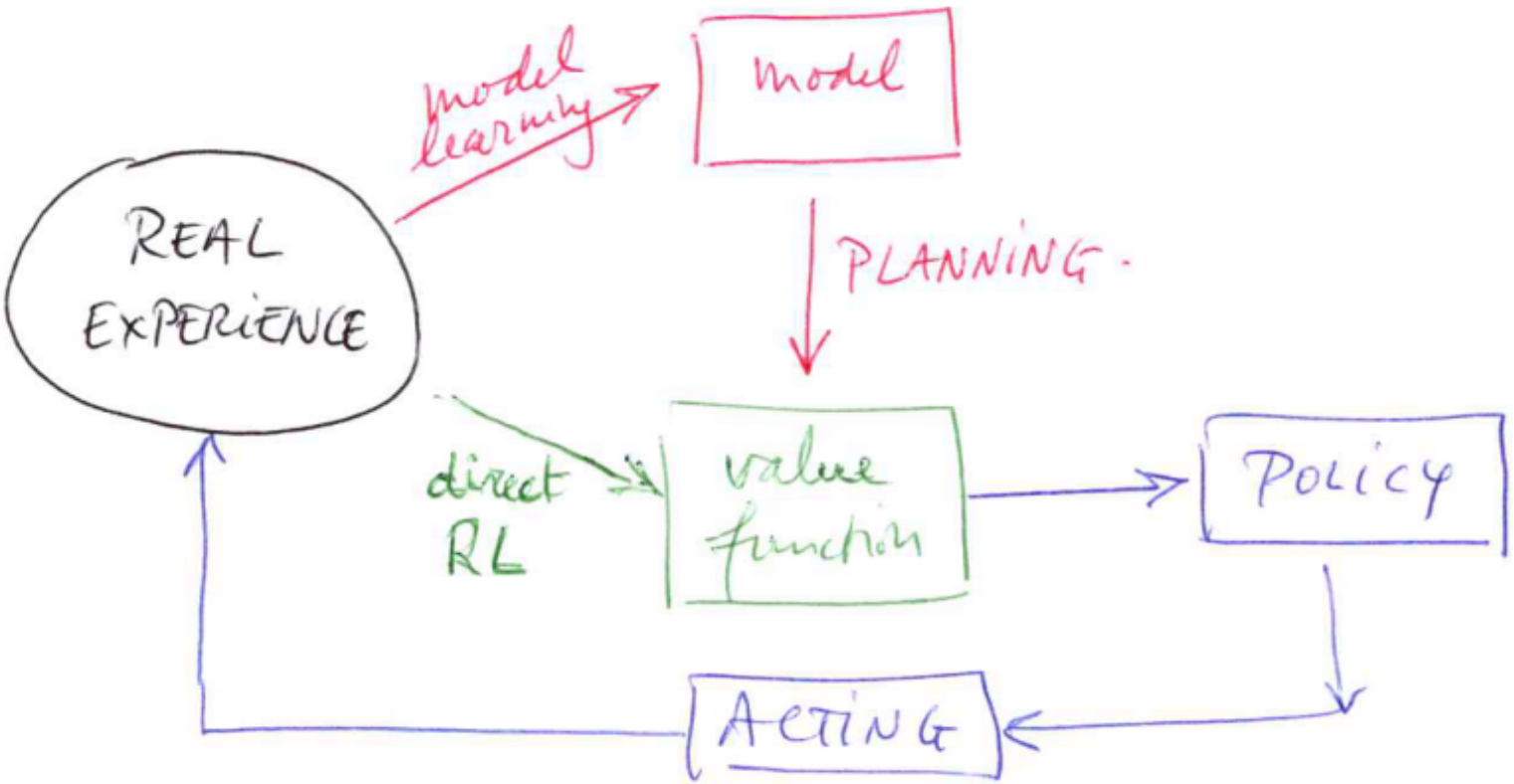
Model-based learning

- Until now:Model = fully specified MDP!
- More generally: anything that helps the agent to plan:
- More accurate models are more effective (myths vs. science):
  - Folklore and weather saying:
    wet and windy May fills the barn with corn and hay.
  - Meteorological models running on supercomputer



Dyna-Q; Integrating planning and learning

Real experience can be used in two ways:



DYNA-Q: Algorithm

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \varepsilon$-greedy(S,Q)

(c) Execute action A; observe resultant reward, R, and state, S'

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

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

(f) Repeat n times:

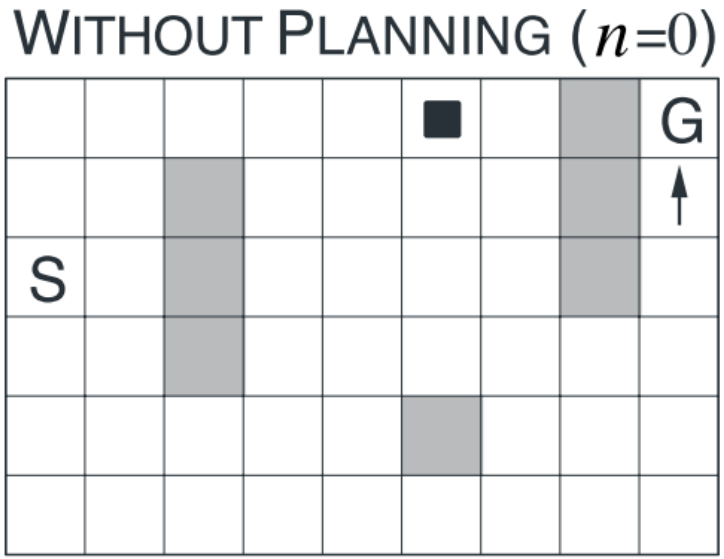
$S \leftarrow$ random previously observed state

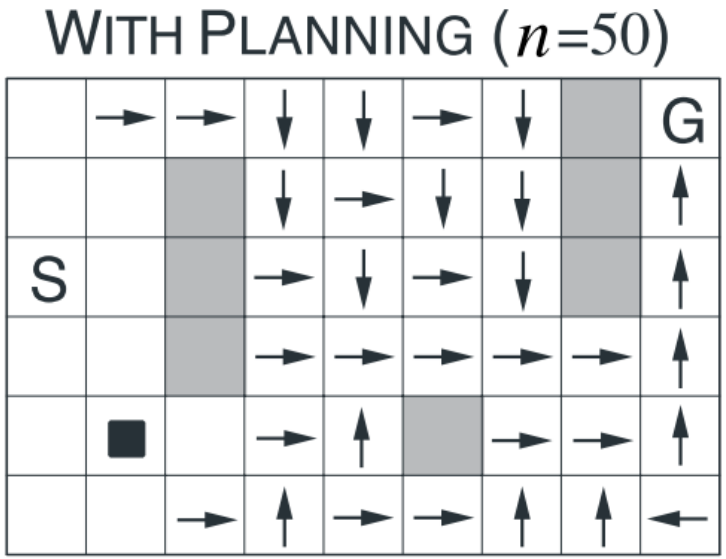
$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$

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

DYNA-Q: Maze example

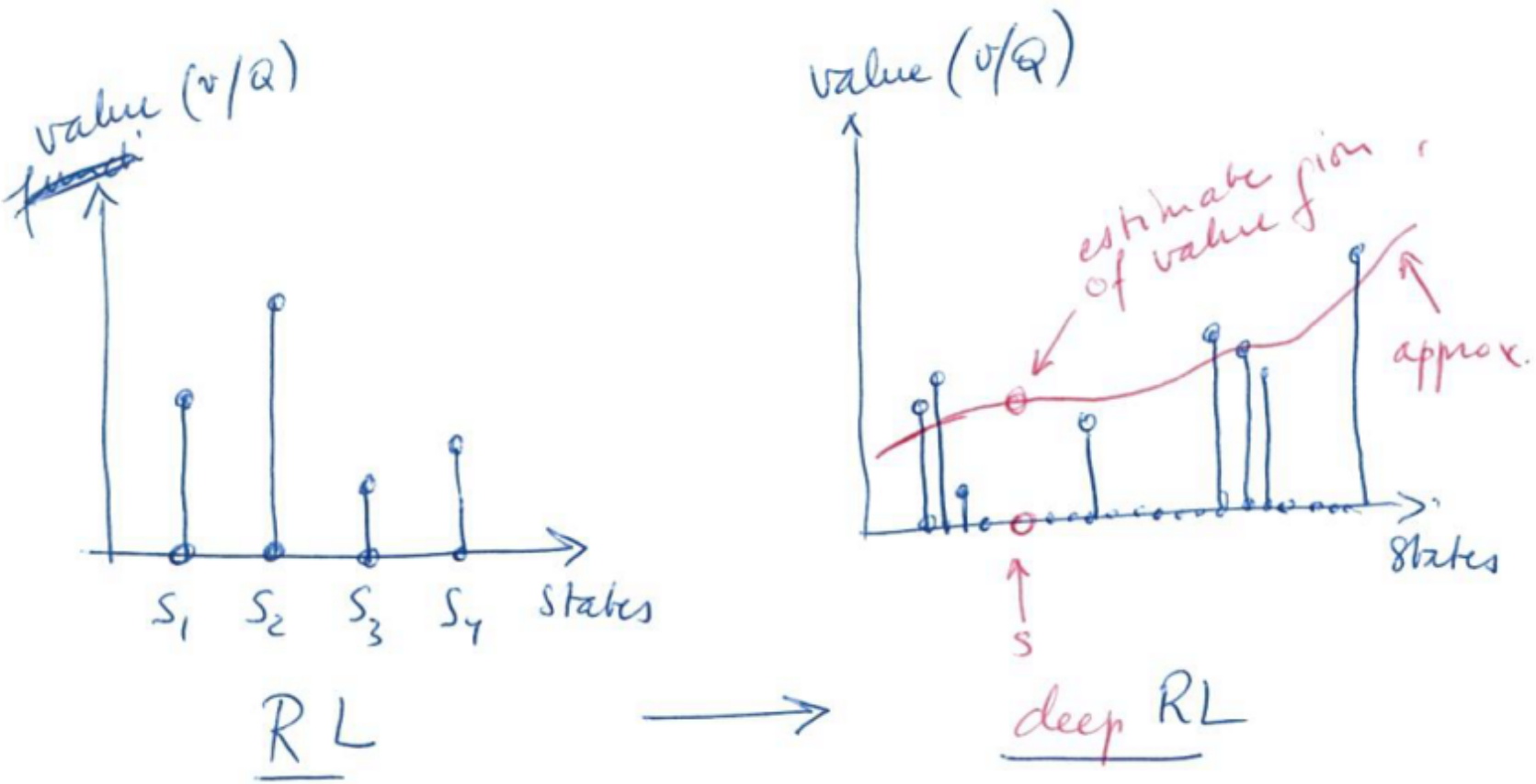




Outline

Deep RL

- What if number of states or actions is **huge**!
- Use function approximation (i.e. generalisation) to estimate value functions in unseen states;



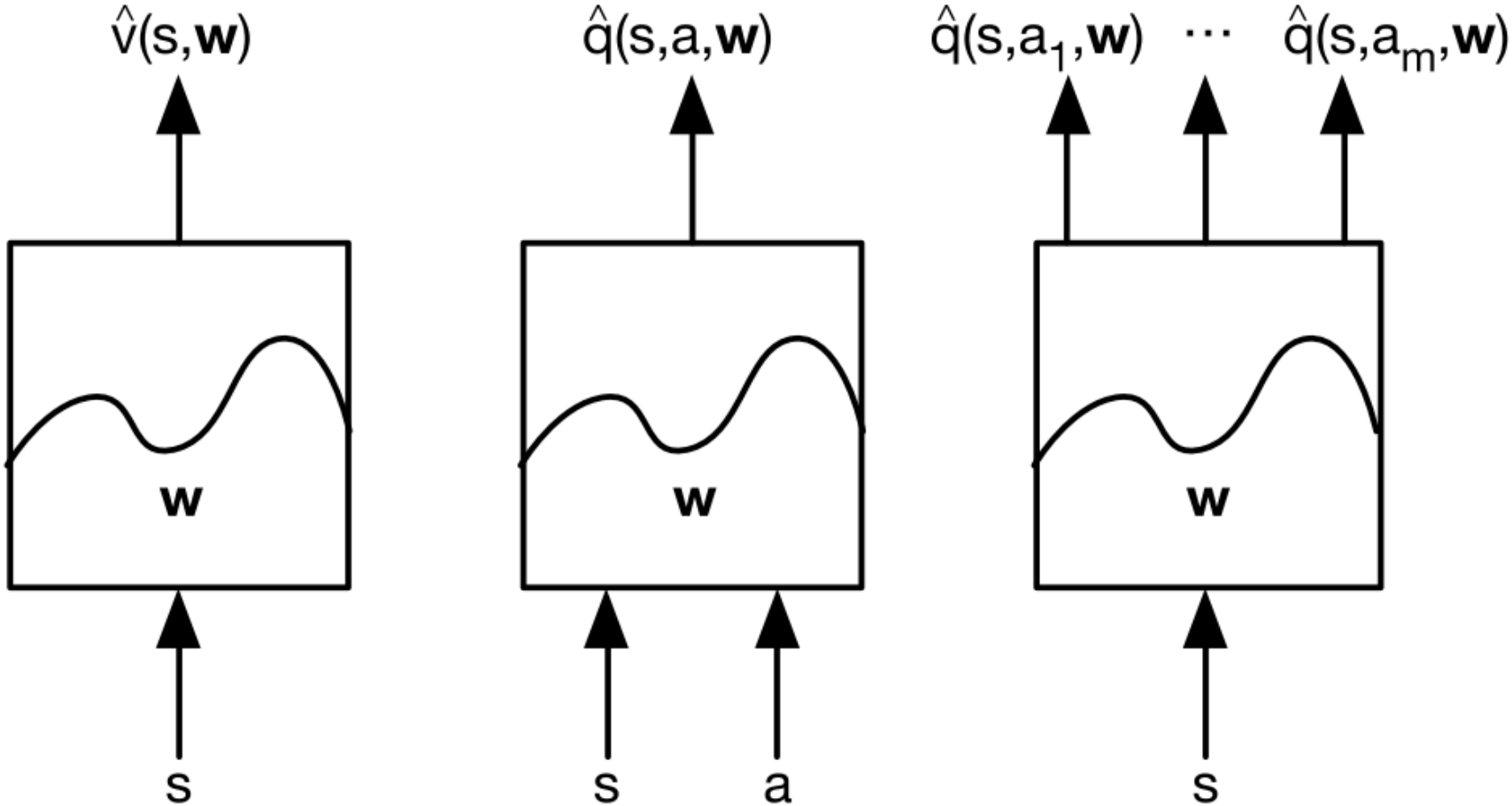
Value Function Approximation for Large RL Problems

- So far we have represented value function by a *lookup table*
  - Every state s has an entry $V(s)$
  - Or every state-action pair s, a has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with *function approximation*

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

  - *Generalise* from seen states to unseen states
  - *Update* parameter $\mathbf{w}$ using MC or TD learning

Types of Value Function Approximation



Which function approximation?

Plenty of choice:

- Linear combinations of features (basis functions)
- Neural networks (ANNs): universal function approximator;
- Miscellaneous techniques: e.g. decision or regression trees,
- etc. ...

Gradient-based solution methods:

use differentiable approximators

Stochastic Gradient Descent

- Goal: find parameter vector $\mathbf{w}$ minimising mean-squared error between approximate value fn $\hat{v}(s, \mathbf{w})$ and true value fn $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

- Gradient descent finds a local minimum

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

Incremental Prediction Algorithms

- Have assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a *target* for $v_\pi(s)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

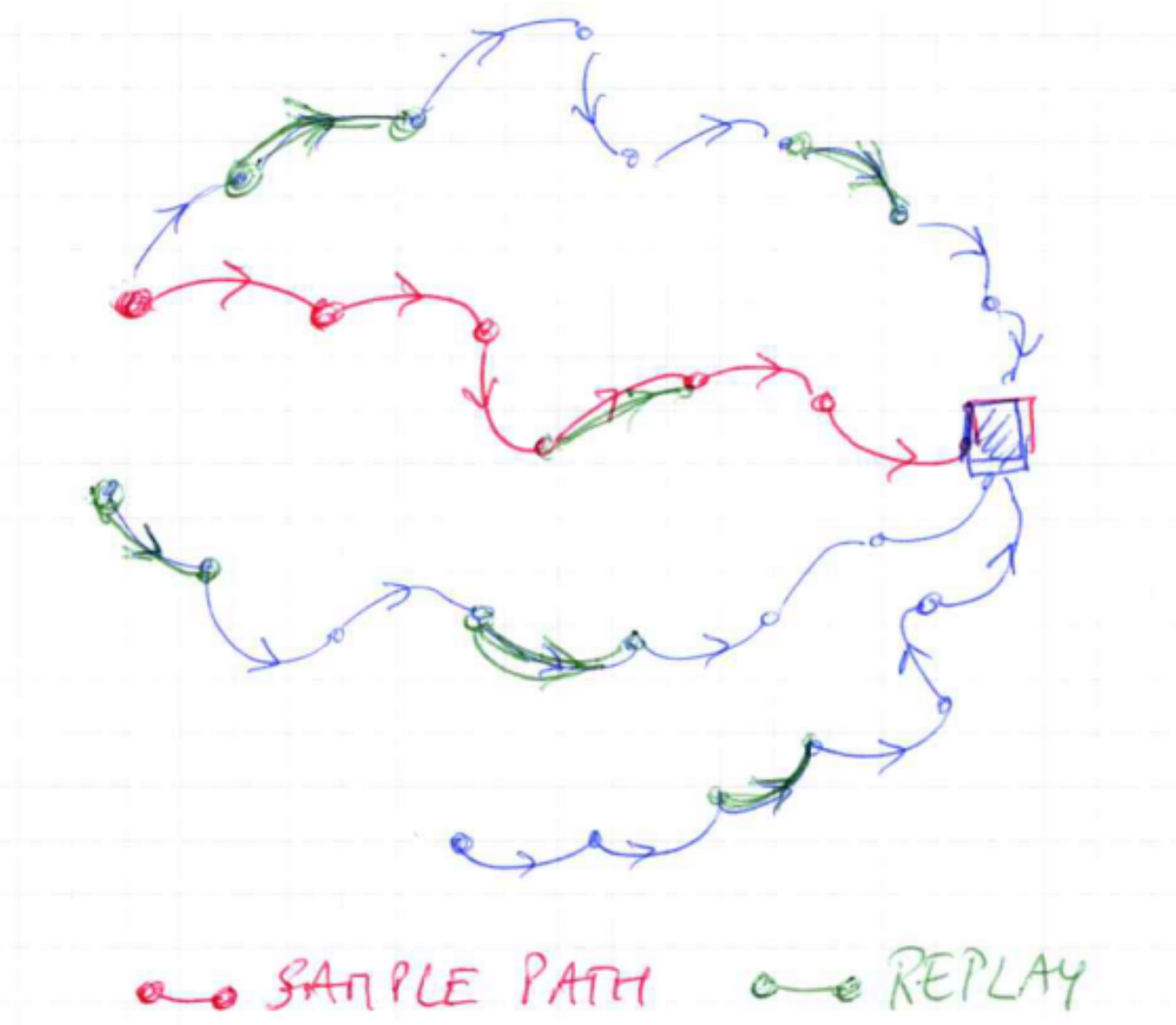
  - For TD(0), the target is the TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

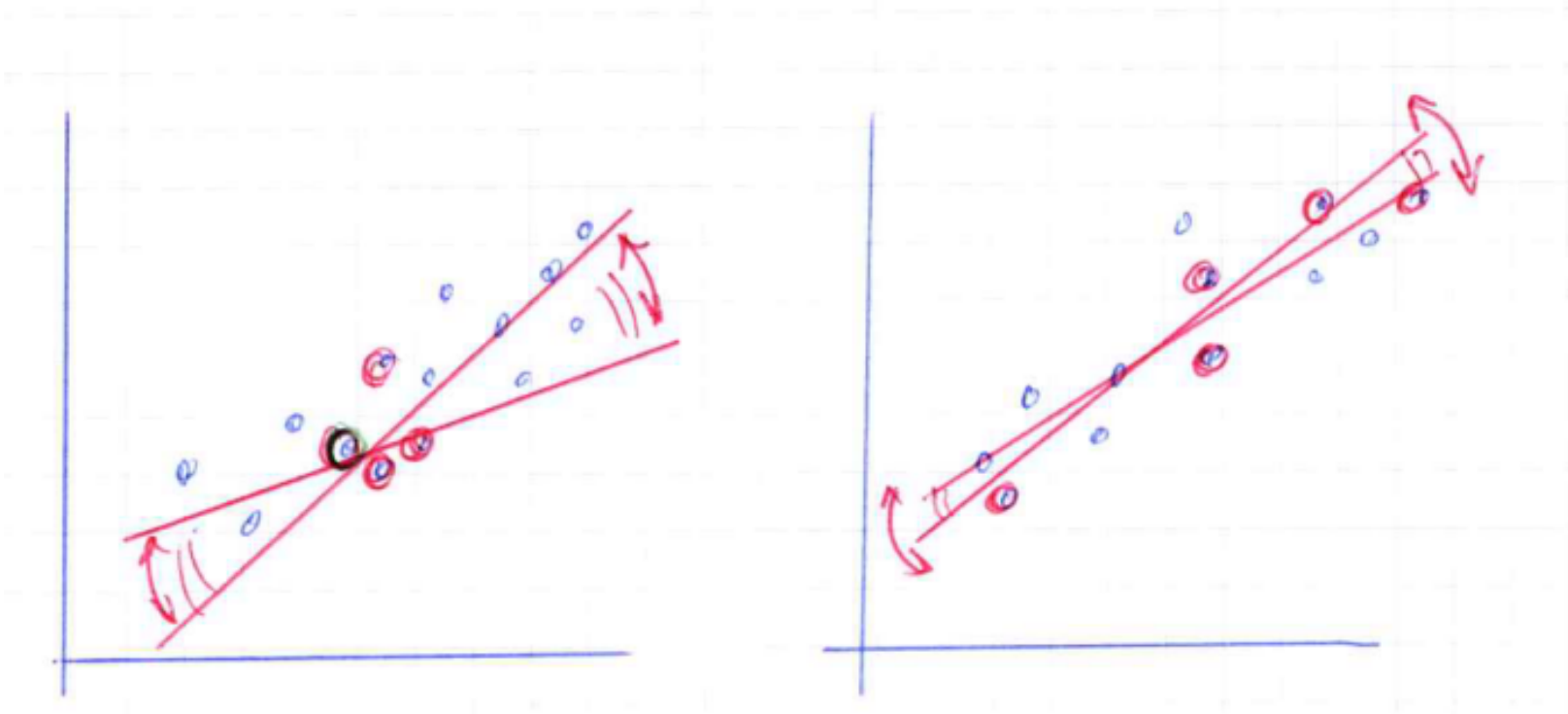
Batch methods and Experience Replay

- Standard Q-learning: Data are NOT used optimally:
  - sample paths are not re-used;
  - values along sample paths are strongly correlated (unstable!)!
- Batch methods seek to find best-fitting solution
- given all of the agent's experience (experience replay)

Sample path versus experience replay



Correlated vs. uncorrelated



DQN: Experience replay in Deep Q-Networks

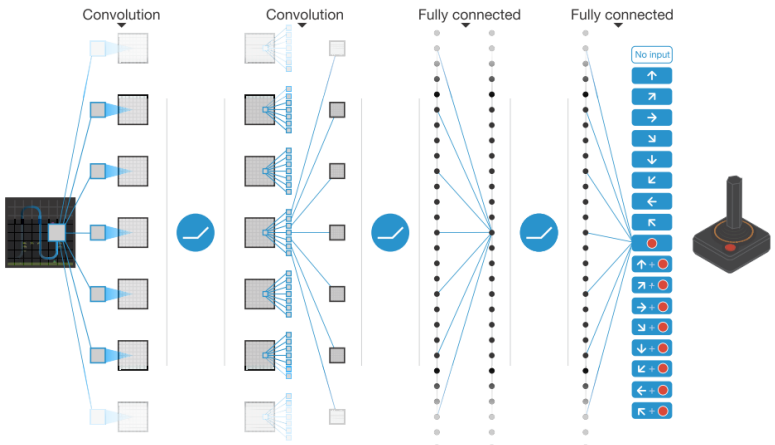
- DQN uses experience replay and fixed Q-targets
- Store transition (s_t,a_t,r_{t+1},s_{t+1}) in replay memory D
- Sample random mini-batch of transitions (s,a,r,s′) from D
- Compute Q-learning targets w.r.t. old, fixed parameters w−

- Optimize MSE between Q-network and Q-learning targets

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

DQN: Deep Q-Network, *(Mnih et al, Nature 2015)*

- Single RL architecture achieved (super)human performance on suite of classic ATARI games;



- **Key features and improvements:**
  - Function approximation based on deep NN;
  - Experience replay (data re-use and de-correlation)
  - Use of target network for stabilisation

DQN: Deep Q-Network, *(Mnih et al, Nature 2015)*

- **Loss function**

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

- **Gradient**

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right].$$

DQN Pseudo-code

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

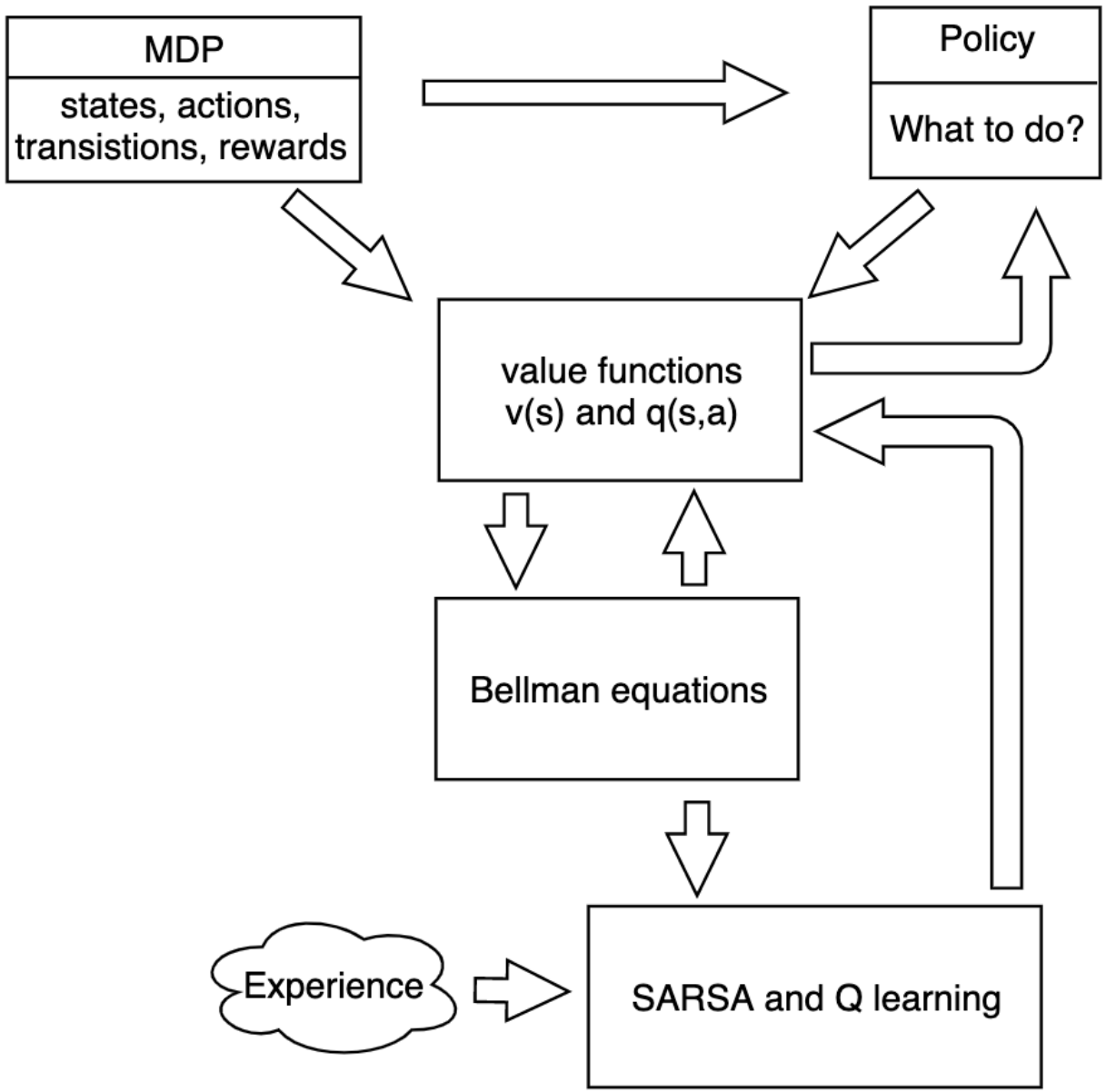
Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

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 $\hat{Q} = Q$

Outline

Summary


MDP
states, actions, transistions, rewards
Policy
What to do?
value functions v(s) and q(s,a)
Bellman equations
Experience
SARSA and Q learning

Summary

- RL: learning from sequential interaction
- RL formalism: Markov Decision Process (MDP):
  MDP = states + actions + rewards + transitions ($+\delta$)
- Bellman equations for value functions (internal consistency)
- Model-based (planning) versus model-free (learning)
- Bellman optimality condition
- Gists of Recent Deep RL Algorithms
- etc