Dynamic programming

Corrado Possieri

Machine and Reinforcement Learning in Control Applications

What is dynamic programming?

Dynamic: the process change with time.

Programming: solving a problem.

- It is a method to solve complex problems.
- The original problem is broken down into sub-problems:
  - solve the sub-problems;
  - combine solutions to solve the original problem.

Requirements for dynamic programming

- DP is a general solution method.
- It requires a special structure
  - optimal substructure
    - the problem can be decomposed into simpler sub-problems;
  - overlapping sub-problems.
    - sub-problems are recurring many times;
    - solutions to sub-problems can be reused.

- MDP satisfy both properties
  - Bellman optimality equation allows to break the problem;
  - value functions reuse solutions.

Planning by dynamic programming

- DP requires knowledge of the function $p(s', r|s, a)$.
- It is used for **planning**.
- With estimates of $p(s', r|s, a)$ it can be used for **learning**.
- It can be used for
  - prediction
    - given π, find v_π;
  - control
    - find π_*.

Policy evaluation

Prediction problem

Given π, estimate v_π.

- Can be addressed by solving the Bellman equation

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}^+, r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma v_\pi(s')\right).$$

- Uniqueness of solution is guaranteed by
  - $\gamma < 1$;
  - eventual termination from all states under π.
- Linear system of $|\mathcal{S}|$ equations in $|\mathcal{S}|$ unknowns.

Iterative policy evaluation

- Iterative application of Bellman expectation backup
  - construct a sequence $v_0, v_1, v_2, \ldots$ of approximations of v_π.
- Synchronous backups
  - initialize v_0 arbitrarily
    - terminal state must have value 0;
  - use Bellman equation as update rule

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}^+, r \in \mathcal{R}} p(s', r|s, a) \left(r + \gamma v_k(s')\right),$$

for all $s \in \mathcal{S}$

- all states are synchronously backed up.

In-place policy evaluation

- Classical policy evaluation stores two value functions
  - $v_k(s)$ values taken at previous time steps;
  - $v_{k+1}(s)$ updated values.
- Alternatively the values can be updated *in-place*
  - each new value immediately overwrites the previous;
- usually convergence is faster.

Iterative policy evaluation

In-place iterative policy evaluation

Input: policy π, threshold θ
Output: estimate of v_π

Initialization

$V(s) \leftarrow \text{random}, \forall s \in \mathcal{S}^+$
$V(\text{terminal}) \leftarrow 0$

Loop

repeat
$\Delta \leftarrow 0$
for each $s \in \mathcal{S}$ **do**
$v \leftarrow V(s)$
$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$
$\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
until $\Delta < \theta$

Convergence of iterative policy evaluation

- v_π is a fixed point of Bellman update.
- The Bellman update can be written in compact form as

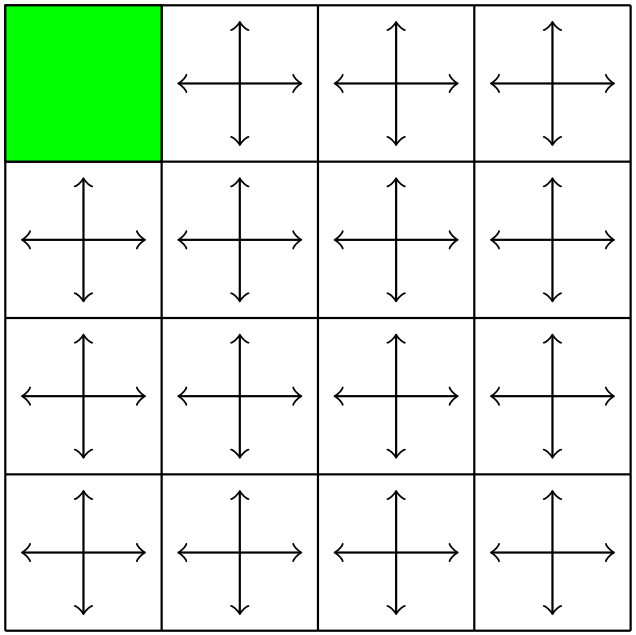
$$T^\pi(v) = R^\pi + \gamma P^\pi v$$

- This operator is a γ-contraction
  - given two policies u and v, one has

$$\begin{aligned}\|T^\pi(v) - T^\pi(u)\|_\infty &= \|R^\pi + \gamma P^\pi v - (R^\pi + \gamma P^\pi u)\|_\infty \\ &\leqslant \gamma \|P^\pi\|_\infty \|u - v\|_\infty \leqslant \gamma \|u - v\|_\infty;\end{aligned}$$

  - by contraction mapping theorem
    - convergence to an unique fixed point;
    - linear convergence rate of γ.

Gridword example

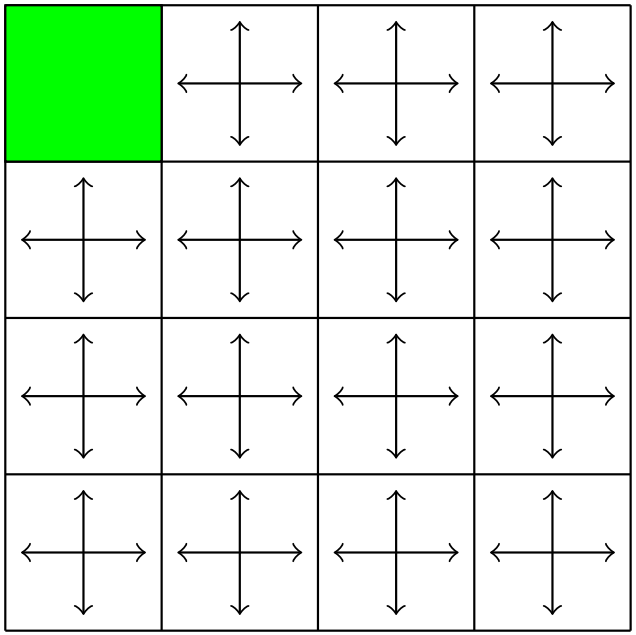


$$R_t = -1$$

- Undiscounted MDP;
- One terminal state;
- Actions leading outside leave state unchanged;
- Reward is -1 until terminal;
- Uniform random policy.

Gridword example

Synchronous update



$$R_t = -1$$

$k = 1$

0	−1.0	−1.0	−1.0
−1.0	−1.0	−1.0	−1.0
−1.0	−1.0	−1.0	−1.0
−1.0	−1.0	−1.0	−1.0

$k = 2$

0	−1.8	−2.0	−2.0
−1.8	−2.0	−2.0	−2.0
−2.0	−2.0	−2.0	−2.0
−2.0	−2.0	−2.0	−2.0

$k = 3$

0	−2.4	−2.9	−3.0
−2.4	−2.9	−3.0	−3.0
−2.9	−3.0	−3.0	−3.0
−3.0	−3.0	−3.0	−3.0

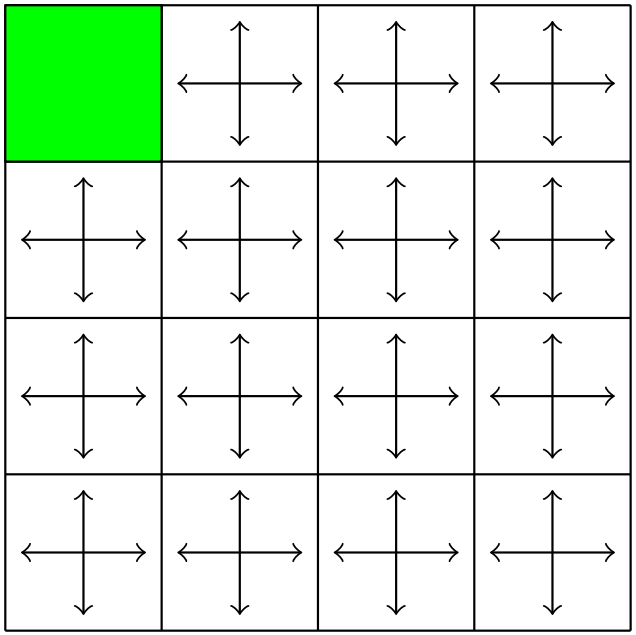
$\vdots$

$k = \infty$

0	−30.	−45.	−52.
−30.	−41.	−50.	−54.
−45.	−50.	−55.	−57.
−52.	−54.	−57.	−59.

Gridword example

In-place update



$$R_t = -1$$

$k = 1$

0	−1.0	−1.2	−1.3
−1.0	−1.5	−1.7	−1.8
−1.2	−1.7	−1.8	−1.9
−1.3	−1.8	−1.9	−1.9

$k = 2$

0	−1.9	−2.5	−2.7
−1.9	−2.8	−3.2	−3.4
−2.5	−3.2	−3.6	−3.7
−2.7	−3.4	−3.7	−3.8

$k = 3$

0	−2.8	−3.8	−4.2
−2.8	−4.0	−4.7	−5.0
−3.8	−4.7	−5.2	−5.4
−4.2	−5.0	−5.4	−5.6

$\vdots$

$k = \infty$

0	−30.	−45.	−52.
−30.	−41.	−50.	−54.
−45.	−50.	−55.	−57.
−52.	−54.	−57.	−59.

How to improve policies

- We know how good is to follow π
  - evaluate v_π.
- What if we take action a rather than $\pi(s)$?
  - the function q_π gives us the corresponding value

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)\left(r + \gamma v_\pi(s')\right).$$

Policy improvement theorem

Let π and π' be any pair of deterministic policies such that

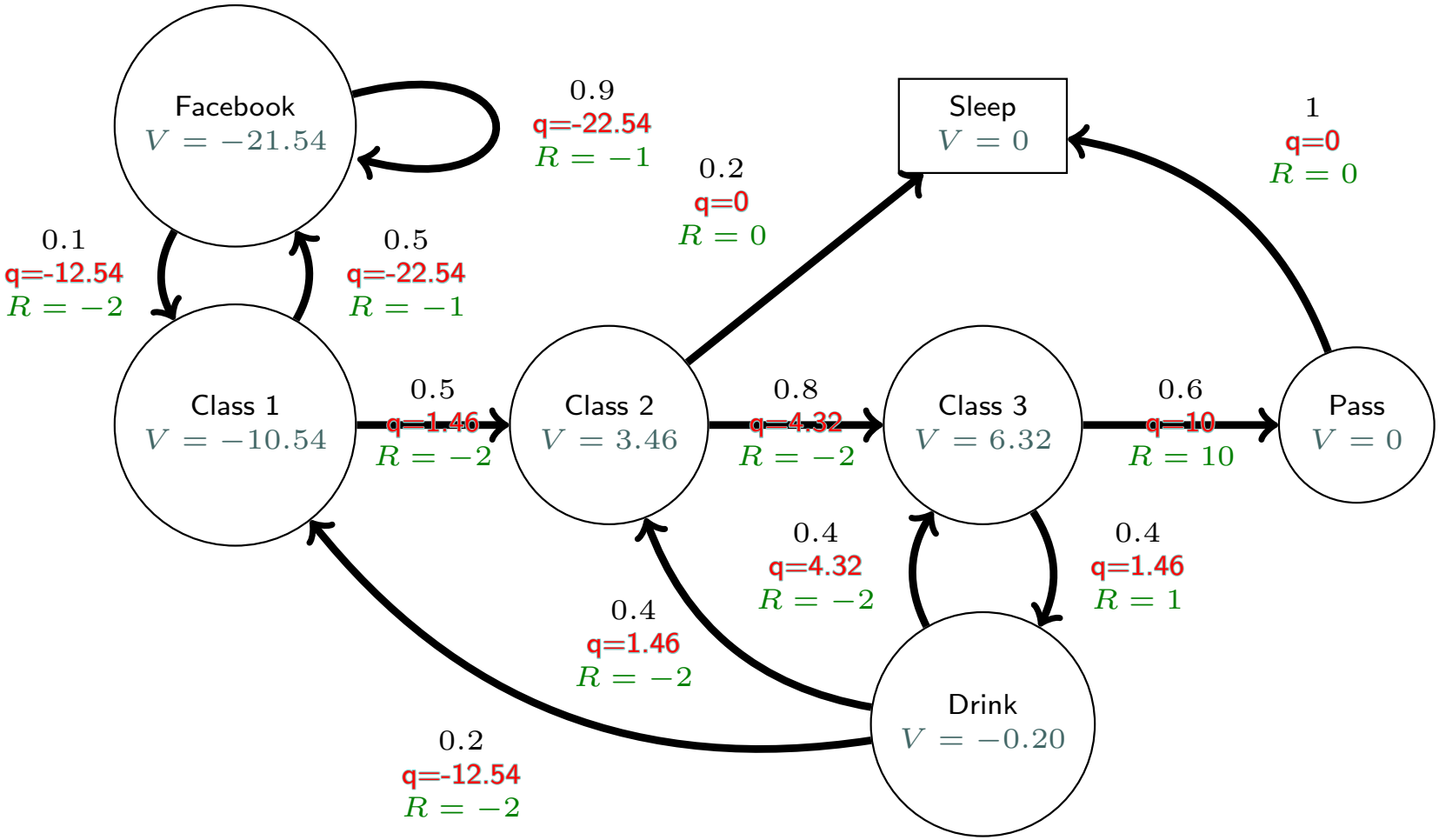
$$q_\pi(s,\pi'(s)) \geqslant v_\pi(s), \quad \forall s \in \mathcal{S}.$$

Then, it holds that

$$v_{\pi'}(s) \geqslant v_\pi(s), \quad \forall s \in \mathcal{S}.$$

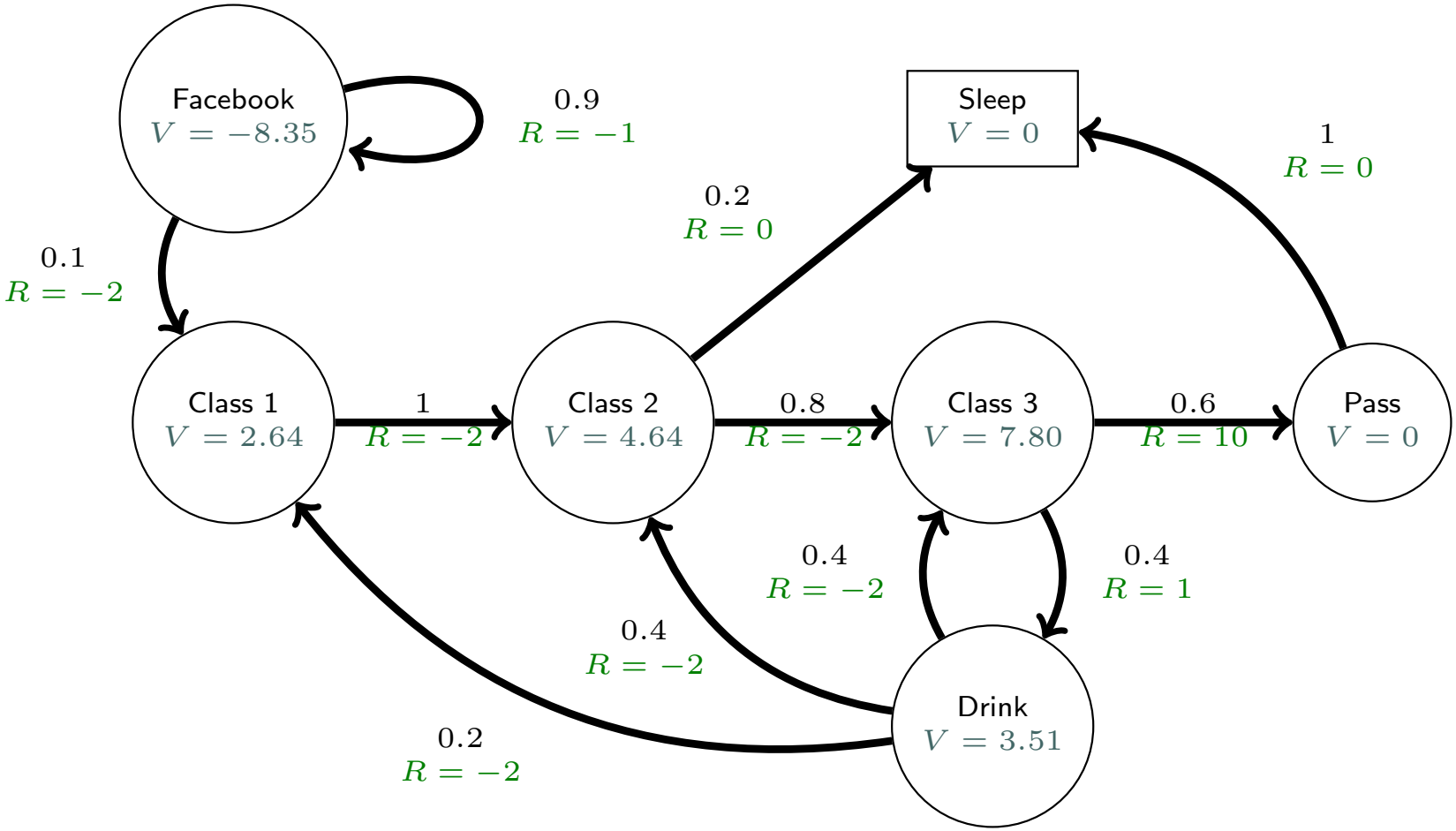
Application to the student Markov chain

$\gamma = 1$



Improved policy in the student Markov chain

$\gamma = 1$



Policy improvement theorem

$$
\begin{aligned}
v_\pi(s) &\leqslant q_\pi(s, \pi'(s)) \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&\leqslant \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s] \\
&\leqslant \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) | S_t = s] \\
&\vdots \\
&\leqslant \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots | S_t = s] \\
&= v_{\pi'}(s)
\end{aligned}
$$

Improved policy selection

- $q_\pi(s, \pi'(s)) > v_\pi(s) \implies v_{\pi'}(s) > v_\pi(s)$ for some $s \in \mathcal{S}$.
- If $q_\pi(s, a) > v_\pi(s)$
  - let π' be a policy identical to π except that

$$\pi'(s) = a \neq \pi(s);$$

  - $\pi \leftarrow \pi'$.
- The improved policy π' is *greedy* with respect to $q_\pi(s, a)$

$$\begin{aligned}\pi'(s) &= \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \sum_{s', r} p(s', r|s, a) \left(r + \gamma v_\pi(s')\right).\end{aligned}$$

Policy improvement

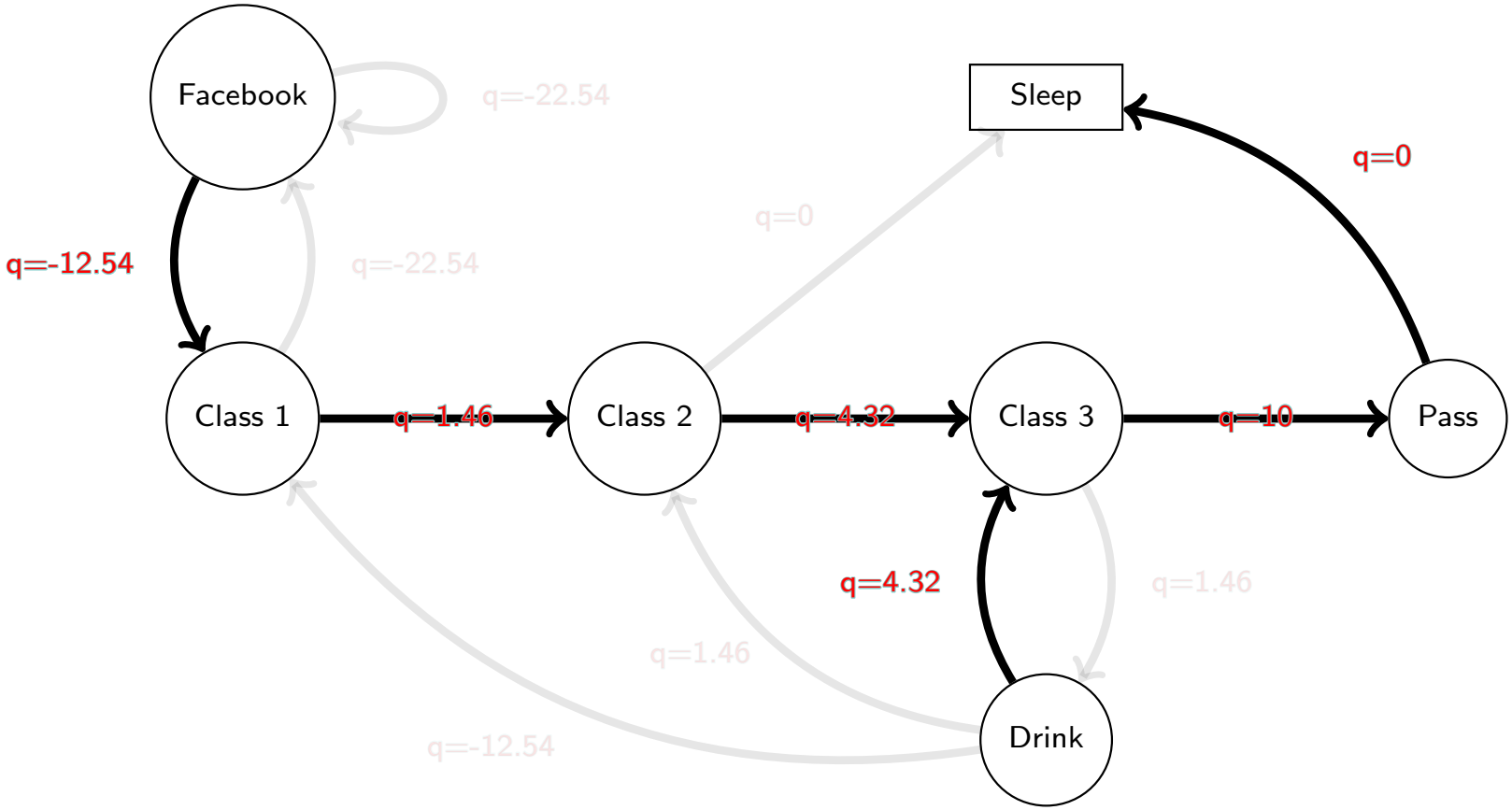
- π' meets the conditions of the policy improvement theorem
  - $v_{\pi'}(s) \geqslant v_\pi(s), \quad \forall s \in \mathcal{S}.$
- What if $v_\pi = v_{\pi'}$?
  - for all states $s \in \mathcal{S}$, it holds that

$$v_{\pi'}(s) = \max_a \sum_{s',r} p(s', r|s, a) \left(r + \gamma v_{\pi'}(s')\right);$$

  - $v_{\pi'}$ satisfies the Bellman optimality condition
    - $v_{\pi'} = v_*.$
- All these properties hold also in the stochastic case
  - as long as all sub-maximal actions are given zero probability.

Improvement in the student Markov chain

$\gamma = 1$



Policy iteration

- Alternating evaluation and improvement we obtain a sequence of monotonically improving value functions and policies

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} v_{\pi_2} \xrightarrow{\mathrm{I}} \pi_3 \xrightarrow{\mathrm{E}} v_{\pi_3} \xrightarrow{\mathrm{I}} \dots$$

- There are finitely many deterministic policies for MDPs
  - policy iteration converges in finite iterations.
- The convergence is often surprisingly fast.

Policy iteration algorithm

Policy iteration

Output: π_*

Initialization

$V(s) \leftarrow \text{random}, \forall s \in \mathcal{S}^+,$
$V(\text{terminal}) \leftarrow 0$
$\pi(s) \leftarrow \text{random}, \forall s \in \mathcal{S}$

repeat

Policy evaluation

repeat

for each $s \in \mathcal{S}$ **do**

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$

until convergence of $V(s)$

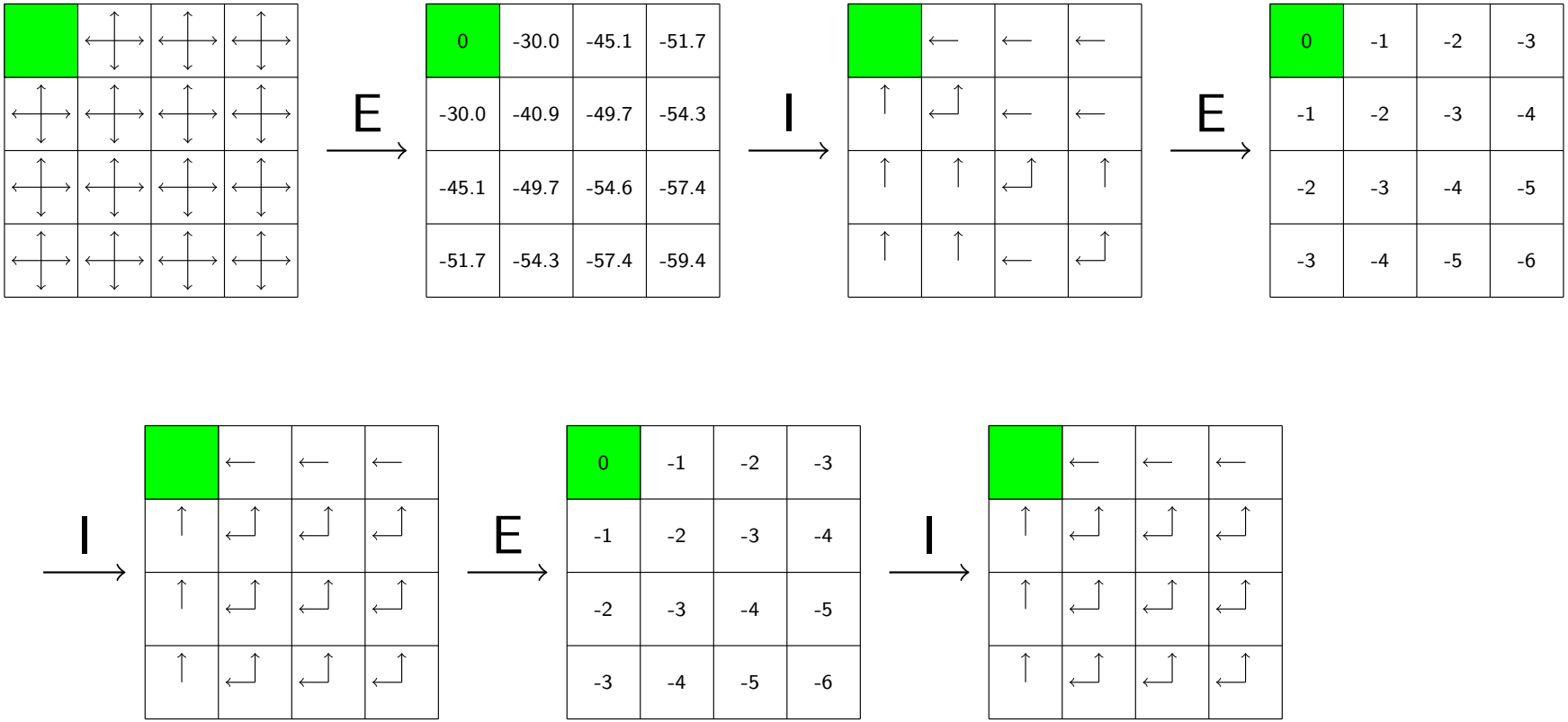
Policy improvement

for each $s \in \mathcal{S}$ **do**

$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$

until $\pi(s)$ is policy stable **return** $\pi(s)$

Policy iteration on the gridword example



Drawbacks of policy iteration

- Policy iteration requires policy evaluation
  - requires multiple sweeps through the state;
  - converges only in the limit.
- Can we truncate updates
  - if we stop after a single sweep, we obtain **value iteration**.

Optimality principle

An optimal policy π_* is constituted by two components

- an optimal first action;
- optimal policy from successor state.

Bellman's principle of optimality

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

If we know $v_*(s')$, we can construct

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a) \left(r + \gamma v_*(s')\right).$$

Value iteration

- Iterative application of Bellman expectation backup
  - construct a sequence $v_0, v_1, v_2, \ldots$ of approximations of v_*.
  - initialize v_0 arbitrarily
    - terminal state must have value 0;
  - use Bellman equation as update rule

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a) \left(r + \gamma v_k(s')\right),$$

    for all $s \in \mathcal{S}$.

- No explicit policy.
- It combines
  - a sweep of policy evaluation;
  - a sweep of policy improvement.

Convergence of value iteration

- The Bellman optimality update can be written as

$$T^*(v) = \max_a \left\{ R^a + \gamma P^a v \right\}.$$

- v_* is a fixed point of T^*.
- the optimal policy π_* is such that

$$T^*(v) = R^{\pi_*} + \gamma P^{\pi_*} v;$$

  - the operator T^* is a γ-contraction;
  - by contraction mapping theorem
    - convergence to an unique fixed point;
    - linear convergence rate of γ.

Value iteration algorithm

Value iteration

Input: threshold θ

Output: v_*, π_*

$V(s) \leftarrow$ random, $\forall s \in \mathcal{S}^+$,

$V(\text{terminal}) \leftarrow 0$

repeat

$\Delta \leftarrow 0$

for each $s \in \mathcal{S}$ **do**

$v \leftarrow V(s)$

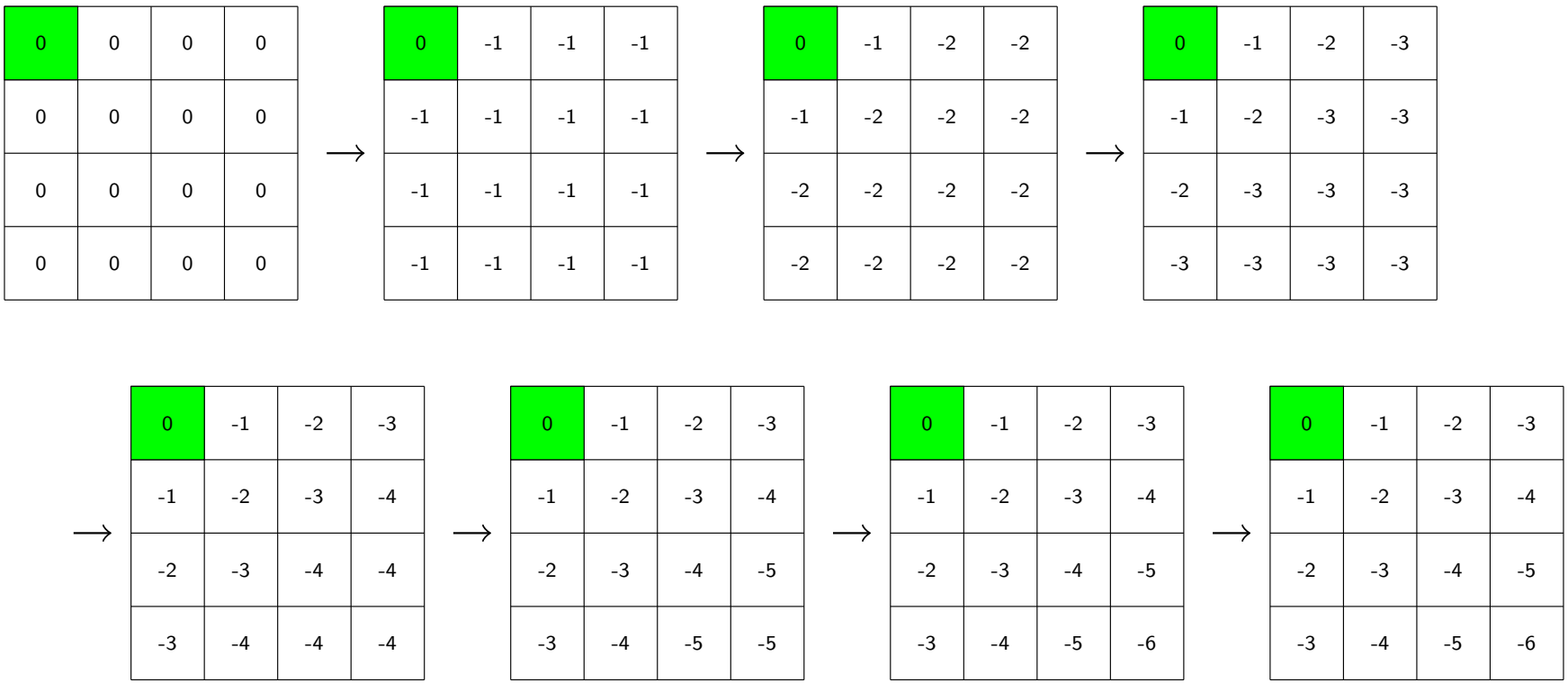
$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$

$\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$

until $\Delta < \theta$

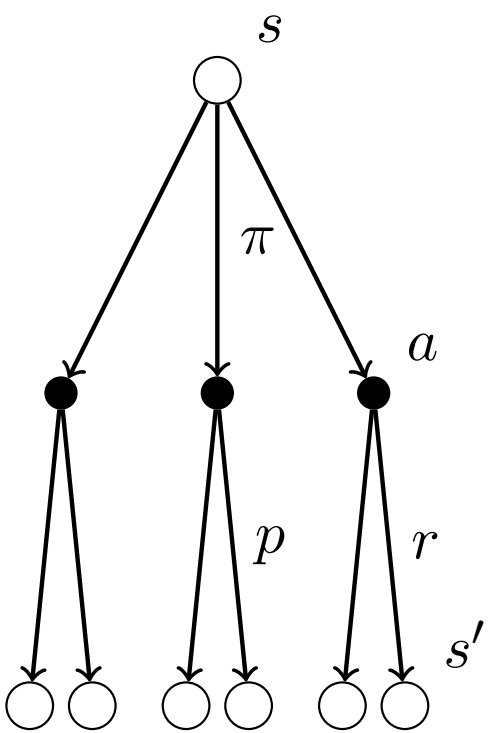
return V, $\pi_*(s) = \arg\max_a \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$

Value iteration on the gridword example

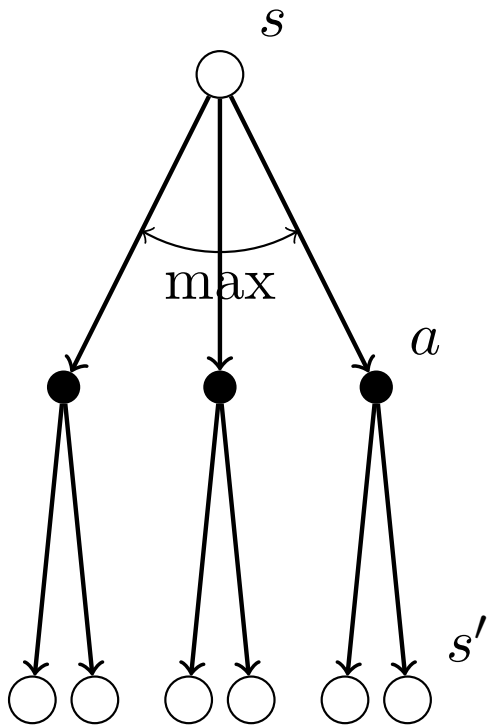


Backup diagrams of PI and VI

Policy evaluation



Value iteration



Asynchronous dynamic programming

- DP involves operations over the entire state set.
- If the state set is large, avoid systematic sweeps
  - backup the state with the largest remaining error

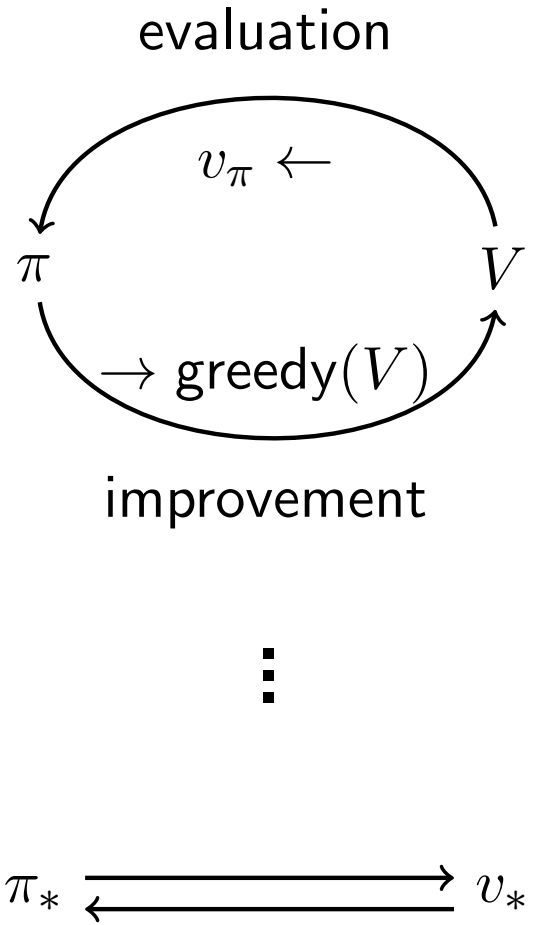
$$\left| \max_a \sum_{s',r} p(s', r|S_t, a) \left(r + \gamma V(S')\right) - v(s) \right|$$

    - requires knowledge of inverse dynamics (predecessor states);
    - implemented efficiently by maintaining a priority queue;
  - use agent's experience to guide the selection of states
    - observe S_t, A_t, R_{t+1};
    - update $V(S_t) \leftarrow \max_a \sum_{s',r} p(s', r|S_t, a) \left(r + \gamma V(s')\right)$;
  - converges if it continues to update the values of all states.

Improvement and evaluation

Improvement: making the policy greedy with respect to the value function makes the value function incorrect for the changed policy.

Evaluation: making the value function consistent with the policy typically causes that policy no longer to be greedy.



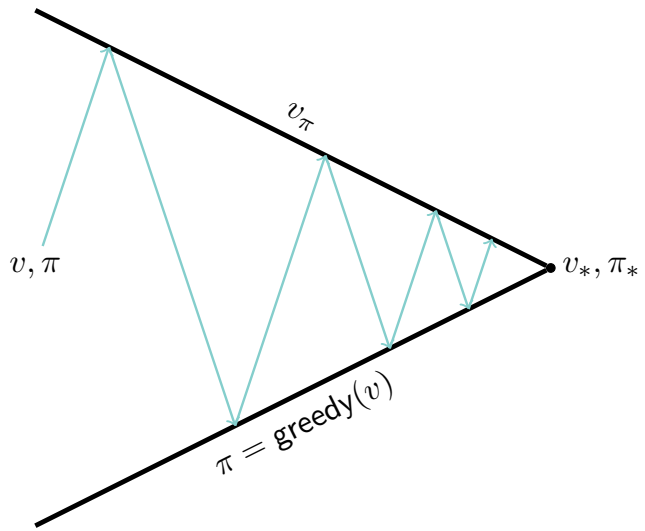
Generalized policy evaluation

Policy evaluation: estimate v_π

- any method to evaluate policies.

Policy improvement: find $\pi' \geqslant \pi$

- any method to improve policies.



Efficiency of dynamic programming

Prediction	Iterative policy evaluation
Control	Policy iteration
	Value iteration

- Complexity $O(mn^2)$ per iteration, for m actions and n states.
- Using q_π leads to complexity $O(m^2n^2)$ per iteration.
- Faster than any direct search in policy space.
- **Curse of dimensionality**: the number of states often grows exponentially with the number of state variables.
- With large state spaces, asynchronous DP is often preferred.