Upper Bounds in RL 2

August 6, 2021

Background: Markov Decision Processes

- X - state space, $(X_t)_{t>0}$ - sequence of random states;
- A - action space, $(A_t)_{t>0}$ - sequence of random actions;
- Policy π - distribution on A;

$$\pi(a|x) = \mathrm{P}(A_t = a|X_t = x)$$

- Markov kernel $P^a(x'|x) = \mathrm{P}(X_t = x'|X_{t-1} = x, A_{t-1} = a)$;
- Deterministic reward $r : \mathrm{X} \times \mathrm{A} \to \mathrm{R}$;
- At step t in the state $X_t = x$ the agent performs action $A_t = a \sim \pi(\cdot|x)$, transits to $X_{t+1} = x' \sim P^a(\cdot|x)$ and obtains reward $r_{t+1} = r(x, a)$;

How to measure policy's quality?

Given two policies π_1 and π_2, how to compare their quality?

Definition

Let's fix some policy π. The state value function for *a* state x at time t is

$$V_t^{\pi}(x) = E_{\pi}[\sum_{k=t}^{\infty} \gamma^k r_{k+1} | x_t = x]$$

The ways of estimating V_t^{π}:

- Monte-Carlo: run a series of independent simulations;
- Temporal Difference-based methods (TD);

Aim

- Approximation of optimal solution - finding optimal policies and optimal value functions. How good is the policy we learned from the approximation procedures? How far are we from the optimal solution?
- **To give an answer, upper bounds of optimal state-value function can be evaluated.**

Theory

- For getting the upper bound of optimal Value Function we can use the algorithm based on the next principles.
- Let's $P^a V^*(x) = \int_X V^*(x')P^a(x'|x)dx'$, then

Definition

We call a function V^{up} the *upper solution* for $V^\star$ if it satisfies

$$V^{\mathrm{up}} \geq \max_{a \in A}\{r(x,a) + \gamma P^a V^{\mathrm{up}}\}.$$

Theory

How to construct V^{up} ?

Denote $Y^{x,a} \sim P(\cdot|x,a)$ and $P^a\Phi(x) = 0$ for all $a \in A$.

- Then

$$V^{\mathrm{up}}(x) = E[\max_a \{r(x,a) + \gamma(V^{\mathrm{up}}(Y^{x,a}) - \Phi(Y^{x,a}))\}] \tag{1}$$

We may choose $\Phi_\pi^{x,a}(y) = V^\pi(y) - (P^a V^\pi)(x)$. And due to Banach's fixed point theorem, we can find upper solutions via iterations (UVIP Algorithm):

$$V_{k+1}^{\mathrm{up},\pi}(x) = E[\max_a \{r^a(x) + \gamma(V_k^{\mathrm{up},\pi}(Y^{x,a}) - \Phi_\pi^{x,a}(Y^{x,a}))\}] \tag{2}$$

An important property: $V_{k+1}^{\mathrm{up},\pi}(x) \geq V^*(x)$ for any $x \in X$ and $k \in N$, provided that $V_0^{\mathrm{up},\pi}(x) \geq V^*(x)$ and $\gamma < 1$;

Upper Value Iteration Procedure

For the fixed policy π and its value function we can construct upper solution with the following scheme

$$V_{k+1}^{\mathrm{up}}(x) = \mathsf{E}\left[\max_{a \in \mathsf{A}}\{r^a(x) + \gamma(V_k^{\mathrm{up}}(Y^{x,a}) - V^{\pi}(Y^{x,a}) + (\mathrm{P}_k^a V^{\pi})(Y^{x,a}))\}\right]$$

Upper Value Iteration Procedure

We can introduce several improvements for the basic UVIP algorithm

- Incremental UVIP:

$$V_{k+1}^{\mathrm{up}}(x) = V_k^{\mathrm{up}}(x) + \alpha_k \delta_{k+1}(V_k^{\mathrm{up}}) \mathbb{1}_{\{x = X_k\}},$$

  where

$$\delta_{k+1}(V) = \max_a \{ r^a(X_k) + \gamma(V(Y_{k+1}^a) - V^\pi(Y_{k+1}^a) + (P^a V^\pi(X_k))) \} - V(X_k)$$

Upper Value Iteration Procedure

We can introduce several improvements for the basic UVIP algorithm

- Incremental UVIP:

$$V_{k+1}^{\mathrm{up}}(x) = V_k^{\mathrm{up}}(x) + \alpha_k \delta_{k+1}(V_k^{\mathrm{up}}) \mathbb{1}_{\{x = X_k\}},$$

  where

$$\delta_{k+1}(V) = \max_a \{ r^a(X_k) + \gamma (V(Y_{k+1}^a) \\ - V^\pi(Y_{k+1}^a) + (P^a V^\pi(X_k)))\} - V(X_k)$$

- Take previous iterations of the UVIP for computing $\Phi_\pi^{x,a}$

Upper Value Iteration Procedure

We can introduce several improvements for the basic UVIP algorithm

- Incremental UVIP:

$$V_{k+1}^{\mathrm{up}}(x) = V_k^{\mathrm{up}}(x) + \alpha_k \delta_{k+1}(V_k^{\mathrm{up}}) \mathbb{1}_{\{x = X_k\}},$$

  where

$$\begin{aligned}\delta_{k+1}(V) = \max_a \{ r^a(X_k) + \gamma (V(Y_{k+1}^a) \\ - V^\pi(Y_{k+1}^a) + (P^a V^\pi(X_k)))\} - V(X_k)\end{aligned}$$

- Take previous iterations of the UVIP for computing $\Phi_\pi^{x,a}$

- Transition kernel approximation

Upper Value Iteration Procedure

We can introduce several improvements for the basic UVIP algorithm

- Incremental UVIP:

$$V_{k+1}^{\mathrm{up}}(x) = V_k^{\mathrm{up}}(x) + \alpha_k \delta_{k+1}(V_k^{\mathrm{up}}) \mathbb{1}_{\{x = X_k\}},$$

  where

$$\delta_{k+1}(V) = \max_a \{ r^a(X_k) + \gamma (V(Y_{k+1}^a) \\ - V^\pi(Y_{k+1}^a) + (P^a V^\pi(X_k))) \} - V(X_k)$$

- Take previous iterations of the UVIP for computing $\Phi_\pi^{x,a}$

- Transition kernel approximation

- Real-Time Upper Solutions

Infinite-horizon case

- Take previous iterations of the UVIP for computing $\Phi_\pi^{x,a}$

Algorithm 1: UVIPv2

Input: $V^\pi, V_0^{\mathrm{up}}, V_1^{\mathrm{up}}, \gamma, \varepsilon$

Result: V^{up}

$k = 1$; **while** $\|V_k^{\mathrm{up}} - V_{k-1}^{\mathrm{up}}\|_{\mathsf{X}} > \varepsilon$ **do**

  for $x \in \mathsf{X}$ **do**

    for $x \in \mathsf{X}, a \in \mathsf{A}$ **do**

      for $y \in \mathsf{X}$ **do**

        $\Phi_\pi^{x,a}(y) = V_{k-1}^{\mathrm{up}}(y) - (\mathrm{P}^a V_{k-1}^{\mathrm{up}})(x)$;

      end

    end

    $V_{k+1}^{\mathrm{up}}(x) = \mathsf{E}[\max_a\{r^a(x) + \gamma(V_k^{\mathrm{up}}(Y^{x,a}) - \Phi_\pi^{x,a}(Y^{x,a}))\}], \quad Y^{x,a} \sim \mathrm{P}^a(\cdot|x)$;

  end

  $k = k + 1$;

end

$V^{\mathrm{up}} = V_k^{\mathrm{up}}$.

Garnet + Monte-Carlo

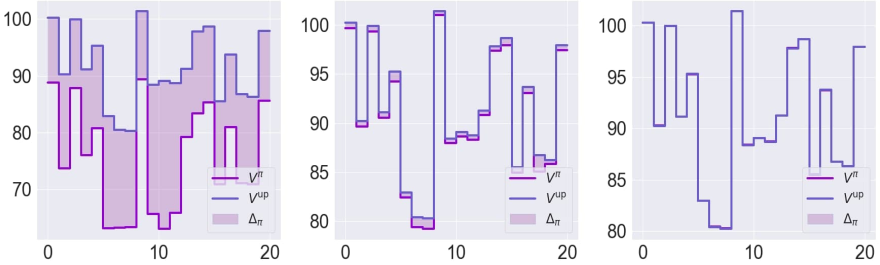


Figure: The difference between V_k^{up} and V^{π}. X-axis represents states of Garnet environment. All of the pairs represents steps of Value Iteration procedure.

Frozen Lake + Monte-Carlo

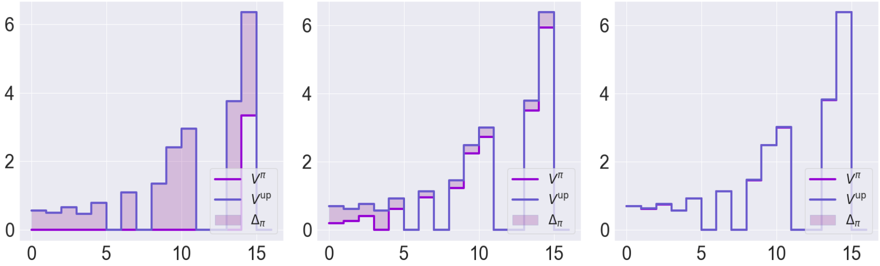
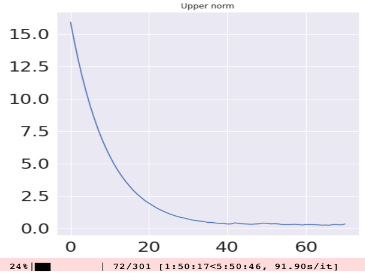


Figure: The difference between V_k^{up} and V^{π}. X-axis represents states of Frozen Lake environment. All of the pairs represents steps of Value Iteration procedure.

Acrobot + Monte-Carlo



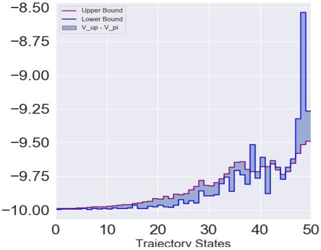


Figure: The difference between $V^{\mathrm{up,k}}$ and V^{π}. X-axis represents states of Acrobot environment. The pictures are computed with DQN policy.

Acrobot + Incremental

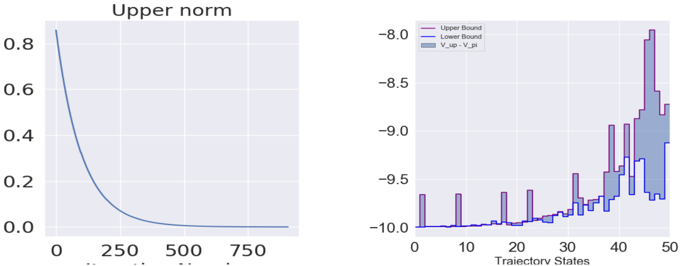


Figure: The difference between $V^{\mathrm{up,k}}$ and V^{π}. X-axis represents states of Acrobot environment. The pictures are computed with DQN policy.

UVIP with empirical P

- Empirically estimate the markov kernel P^a
- Use $\widehat{\mathrm{P}}^a_k(y|x)$ for sampling $Y^{a,X}$

Algorithm 2: (Incremental + Empirical P)

Input: V^π, V^{up}_0, α

Estimate $\widehat{\mathrm{P}}^a_0(\cdot|x)$;

for $k = 1, 2, \dots$ **do**

  Update $\widehat{\mathrm{P}}^a_k(\cdot|x)$;

  for $X \in \mathsf{X}$ **do**

    $\delta_k = \max_{a \in \mathsf{A}}\{r^a(X) + \gamma(V^{\mathrm{up}}_{k-1}(Y^{a,X}) - V^\pi(Y^{a,X}) + (\widehat{\mathrm{P}}^a_k V^\pi)(X))\} - V^{\mathrm{up}}_{k-1}(X)$,

    $Y^{a,X} \sim \widehat{\mathrm{P}}^a_k(\cdot|x)$

    $V^{\mathrm{up}}_k(X) = V^{\mathrm{up}}_{k-1}(X) + \alpha\delta_k$

  end

end

Garnet

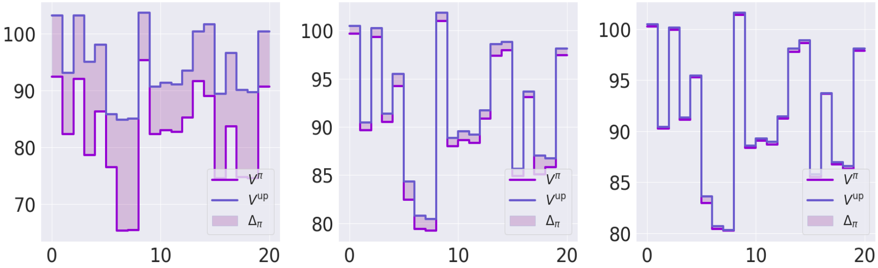


Figure: The difference between $V^{\mathrm{up},\mathrm{k}}$ and V^π. X-axis represents states of Garnet environment. All of the pairs represents steps of Value Iteration procedure.

Garnet

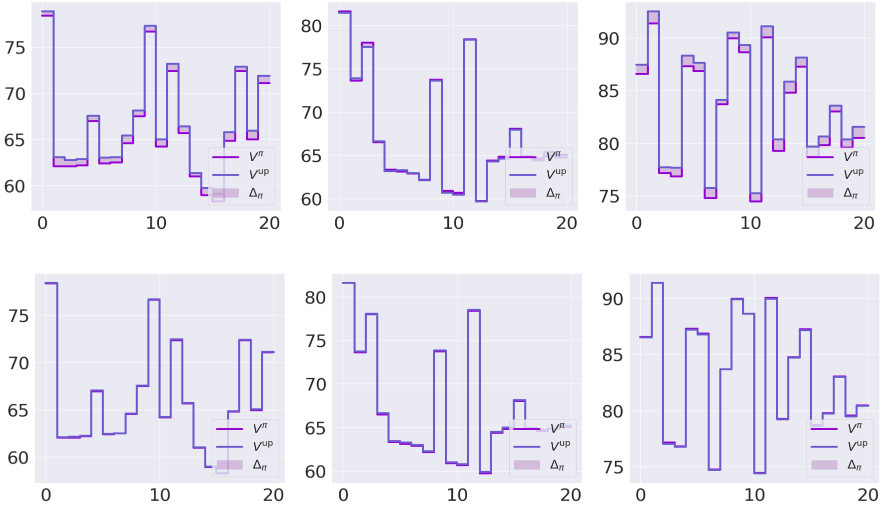


Figure: First row – sampling from $\widehat{\mathrm{P}}^a_k(y|x)$, second row – from $\mathrm{P}^a(y|x)$

Frozen Lake

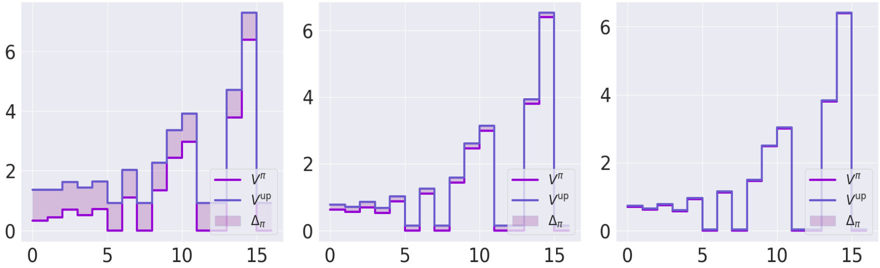


Figure: The difference between $V^{\mathrm{up,k}}$ and V^π. X-axis represents states of Frozen Lake environment. All of the pairs represents steps of Value Iteration procedure.

Finite-horizon case

Algorithm 3: Real-Time Upper Solutions(RTUS)

Input: $V^{\pi}, \forall x \in \mathrm{X}, \forall t \in [H], V_t^{\mathrm{up},0}, V_t^{\mathrm{up},1}$

for $k = 1, 2, \ldots$ **do**

Initialize X_1^k

for $t = 1, \ldots, H$ **do**

$A_t^k \in \arg\max_{a \in \mathrm{A}} \{ r^a(X_t^k) + \mathrm{P}^a V_{t+1}^{\mathrm{up,k-1}}(X_t^k) \}$

Act with A_t^k and observe X_{t+1}^k

$\delta_t^k(x) =$

$(\max_{a \in \mathrm{A}} \{ r^a(x) + V_{t+1}^{\mathrm{up,k-1}}(Y_{t+1}^{a,k}) - V_{t+1}^{\mathrm{up,k-2}}(Y_{t+1}^{a,k}) +$

$(\mathrm{P}^a V_{t+1}^{\mathrm{up,k-2}})(x) \} - V_t^{\mathrm{up,k-1}}(x)) \cdot \mathbb{I}\left\{ x = X_t^k \right\},$

$Y_{t+1}^{a,k} \sim \mathrm{P}^a(\cdot | X_t^k)$

$V_t^{\mathrm{up,k}}(x) = V_t^{\mathrm{up,k-1}}(x) + \alpha_t \delta_t^k(x)$

end

end

Garnet

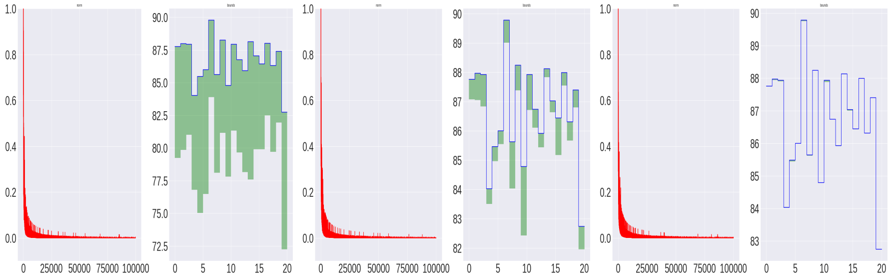
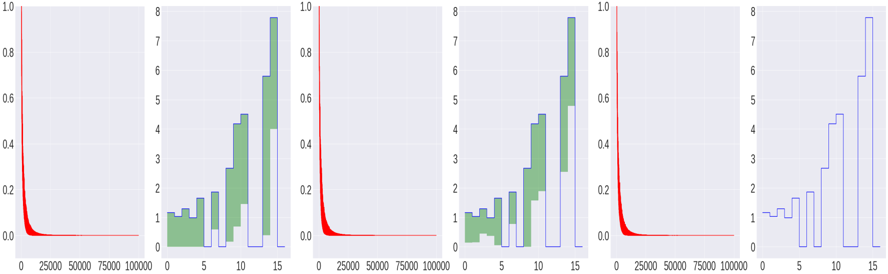


Figure: The difference between $V_0^{\mathrm{up,k}}$ and V_0^{π}. X-axis represents states of Garnet environment. The first pair of pictures is computed with Q-learning policy on 1000 iterations, the second on 5000 iterations and the third group with RTDP algorithm.

Frozen Lake



Kernel-Based Reinforcement Learning

- The transition functions of KBRL's model, $\hat{P}^a : \hat{S} \times \hat{S}x \mapsto [0,1]$, are given by:

$$\hat{P}^a(\hat{s}_i^b|s) = \begin{Bmatrix} \kappa_\tau^a(s, s_i^b), a = b \\ 0, \mathit{else} \end{Bmatrix}$$

where $\kappa_\tau^a(s, s_i^b)$ is normalized kernel function.

Atari games

Algorithm 2: Approximate UVIP

Input: Sample $(\mathsf{x}_1, \ldots, \mathsf{x}_N)$; V^π, $\widetilde{V}_0^{\mathrm{up}}$, M_1, M_2, γ, ε

Result: $\widehat{V}^{\mathrm{up}}$

Generate $r^a(X_i), Y_j^{\mathsf{x}_i,a} \sim \mathrm{P}^a(\cdot|\mathsf{x}_i)$ for all $i \in [N]$, $j \in [M_1 + M_2]$, $a \in \mathsf{A}$;

$k = 1$; **while** $\|\widetilde{V}_k^{\mathrm{up}} - \widetilde{V}_{k-1}^{\mathrm{up}}\|_{\mathsf{X}_N} > \varepsilon$ **do**

  for $a \in \mathsf{A}$ **do**

    for $i \in [N]$ **do**

      for $j \in [M_1 + M_2]$ **do**

        $\widehat{V}_k^{\mathrm{up}}(Y_j^{\mathsf{x}_i,a}) = I[\widetilde{V}_k^{\mathrm{up}}](Y_j^{\mathsf{x}_i,a})$ with $I[\cdot](\cdot)$ defined in (6);

      end

      $\overline{V}^{(i,a)} = M_1^{-1} \sum_{j=1}^{M_1} V^\pi(Y_j^{\mathsf{x}_i,a})$;

    end

  end

  for $i \in [N]$ **do**

    $\widetilde{V}_{k+1}^{\mathrm{up}}(\mathsf{x}_i) = M_2^{-1} \sum_{j=M_1+1}^{M_1+M_2} \max_{a \in \mathcal{A}} \{ r^a(\mathsf{x}_i) + \gamma \big(\widehat{V}_k^{\mathrm{up}}(Y_j^{\mathsf{x}_i,a}) - V^\pi(Y_j^{\mathsf{x}_i,a}) + \overline{V}^{(i,a)} \big) \}$;

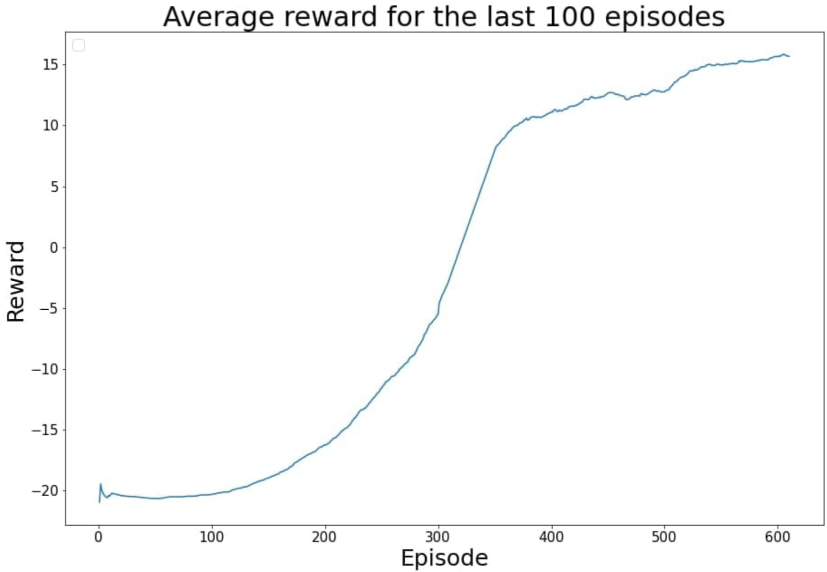
  end

  $k = k + 1$;

end

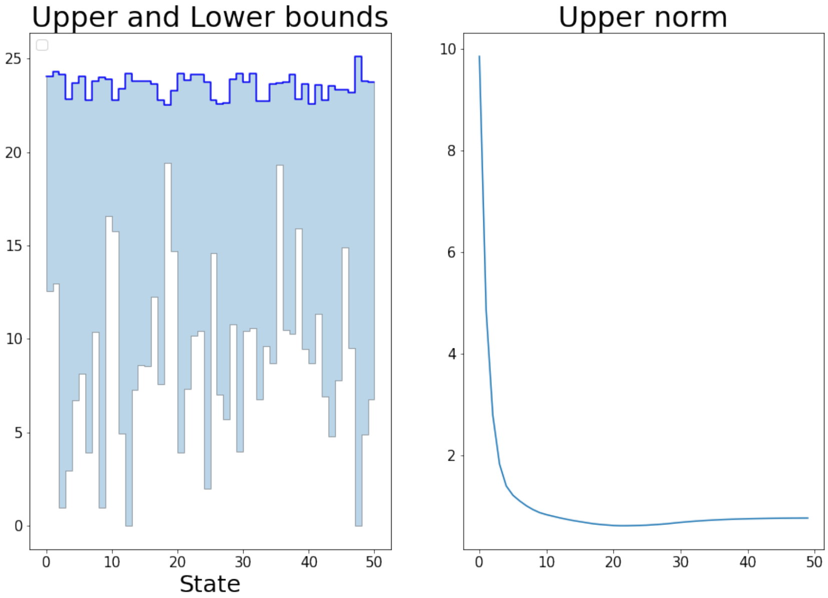
$\widehat{V}^{\mathrm{up}} = \widehat{V}_k^{\mathrm{up}}$.

Atari games



Average reward for the last 100 episodes
Reward
Episode
15
10
5
0
−5
−10
−15
−20
0
100
200
300
400
500
600

Atari games

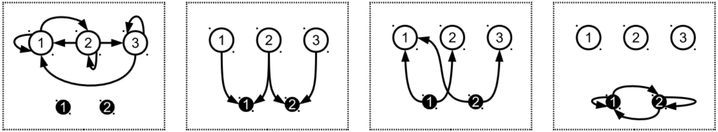

Upper and Lower bounds
25
20
15
10
5
0
0
10
20
30
40
50
State
Upper norm
10
8
6
4
2
0
10
20
30
40
50

Future work

- Merge modifications and run on more complicated environments(for example, Atari games)

Thank you for your attention!

Stochastic-Factorization



$$\mathbf{P} = \begin{bmatrix} \times & \times & 0 \\ \times & \times & \times \\ \times & 0 & \times \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} \times & 0 \\ \times & \times \\ 0 & \times \end{bmatrix} \quad \mathbf{K} = \begin{bmatrix} \times & \times & 0 \\ \times & 0 & \times \end{bmatrix} \quad \bar{\mathbf{P}} = \begin{bmatrix} \times & \times \\ \times & \times \end{bmatrix}$$

Figure: Reducing the dimension of a transition model from n = 3 states to m = 2 artificial states. Original states s_i are represented as big white circles; small black circles depict artificial states $\bar{s}_h$. The symbol '$\times$' is used to represent nonzero elements.

Kernel-Based Stochastic Factorization

Algorithm 1 Batch KBSF

Input: $S^a = \{(s_k^a, r_k^a, \hat{s}_k^a) | k = 1, 2, ..., n_a\}$ for all $a \in A$ ▷ Sample transitions
$\bar{S} = \{\bar{s}_1, \bar{s}_2, ..., \bar{s}_m\}$ ▷ Set of representative states
Output: $\tilde{\mathbf{v}} \approx \hat{\mathbf{v}}^*$
for each $a \in A$ **do**
  Compute matrix $\dot{\mathbf{D}}^a$: $\dot{d}_{ij}^a = \bar{\kappa}_{\bar{\tau}}(\hat{s}_i^a, \bar{s}_j)$
  Compute matrix $\dot{\mathbf{K}}^a$: $\dot{k}_{ij}^a = \kappa_\tau^a(\bar{s}_i, s_j^a)$
  Compute vector $\bar{\mathbf{r}}^a$: $\bar{r}_i^a = \sum_j \dot{k}_{ij}^a r_j^a$
  Compute matrix $\bar{\mathbf{P}}^a = \dot{\mathbf{K}}^a \dot{\mathbf{D}}^a$
Solve $\bar{M} \equiv (\bar{S}, A, \bar{\mathbf{P}}^a, \bar{\mathbf{r}}^a, \gamma)$ ▷ *i.e.*, compute $\bar{\mathbf{Q}}^*$
Return $\tilde{\mathbf{v}} = \Gamma \mathbf{D} \bar{\mathbf{Q}}^*$, where $\mathbf{D}^\intercal = \left[(\dot{\mathbf{D}}^1)^\intercal, (\dot{\mathbf{D}}^2)^\intercal, ...(\dot{\mathbf{D}}^{|A|})^\intercal \right]$