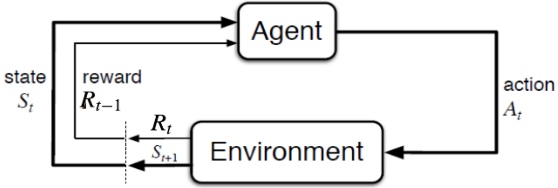
Reinforcement Learning: An Overview

Yi Cui, Fei Feng, Yibo Zeng

Dept. of Mathematics, UCLA

November 13, 2017

Model : Sequential Markov Decision Process



- discrete time steps, t= 1,2,3,...
- $S_t \in \mathcal{S}$, state space
- $A_t \in \mathcal{A}(S_t) \subset \mathcal{A}$, action space
- $R_t(S_t, A_t) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, immediate reward of taking A_t at S_t
- $p(S_{t+1}, R_t | S_t, A_t)$, transition probability

An Optimization Problem

Policy:

$$\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$$

Agent's Goal: choose a policy to maximize a function of reward sequence.

- expected total discounted-reward

$$\begin{aligned}\mathbb{E}[G_1] &= \mathbb{E}[R_1 + \gamma R_2 + \gamma^2 R_3 \cdots] \\ &= \mathbb{E}[\sum_{k=1}^{\infty} \gamma^k R_k]\end{aligned}$$

  where $0 < \gamma < 1$ is a discount rate.

- averaged-reward

An Optimization Problem

State-value function of π

$$V^{\pi}(s) = \mathbf{E}\left[G_t \mid S_t = s; \pi\right]$$

Fix π, V^{π} satisfy **Bellman equation (self-consistency condition)**:

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^{\pi}(s')],\ \forall s \in \mathcal{S}$$

Action-value function of π

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbf{E}\left[G_t \mid S_t = s, A_t = a; \pi\right] \\ &= \mathbf{E}\left[R_t + \gamma G_{t+1} \mid S_t = s, A_t = a; \pi\right] \\ &= \mathbf{E}\left[R_t + \gamma V^{\pi}(S_{t+1}) \mid S_t = s, A_t = a; \pi\right] \end{aligned}$$

Optimal Policy and Optimal Value

Build a partial ordering over policies

$$\pi \geq \pi' \iff V^{\pi}(s) \geq V^{\pi'}(s),\ \forall s \in \mathcal{S}$$

Find an **optimal policy** π^* such that

$$V^*(s) := V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s),\ \forall s \in \mathcal{S}.$$

The **Bellman Optimality Equations** for V^*, Q^* are

$$V^*(s) = \max_{a} \mathbf{E}[R_t + \gamma V^*(S_{t+1}) | S_t = s, A_t = a] \tag{1}$$

$$Q^*(s,a) = \mathbf{E}[R_t + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a]. \tag{2}$$

Note that

$$\pi^*(s) = \arg\max_{a} Q^*(s,a), \quad V^*(s) = \max_{a} Q^*(s,a),\ \forall s \in \mathcal{S}.$$

General Approach for π^*

- If the model $p(s', r|s, a)$ is available $\rightarrow$ dynamic programming.
- If no model $\rightarrow$ reinforcement learning(RL) algorithms
  1. model-based method: learn model then derive optimal policy.
  2. **model-free** method: learn optimal policy without learning model.

For model-free method,

- Value-based: evaluate Q^*, derive a greedy policy
  1. Tabular Implementation: **Sarsa**[1], **Q-learning**[2]
  2. Function Approximation: **Deep Q-learning**
- Policy-based: Search over policy space for better value: **Actor-critic(DDPG)**

On-policy v.s. Off policy

- **Learning Policy**:
  a policy that maps experience into a current choice
  (experience contains: states visited, actions chosen, rewards received, etc.)

- **Update Rule:**
  How the algorithm uses experience to change its estimate of the optimal value function.

Sarsa: On-policy TD Control

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

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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)   Learning Policy
  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)   Update Rule
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-learning: Off-policy TD Control

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

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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

Learning Policy

Update Rule

The Drawbacks of Tabular Methods

	a1	a2	a3	a4
s1	Q(1,1)	Q(1,2)	Q(1,3)	Q(1,4)
s2	Q(2,1)	Q(2,2)	Q(2,3)	Q(2,4)
s3	Q(3,1)	Q(3,2)	Q(3,3)	Q(3,4)
s4	Q(4,1)	Q(4,2)	Q(4,3)	Q(4,4)

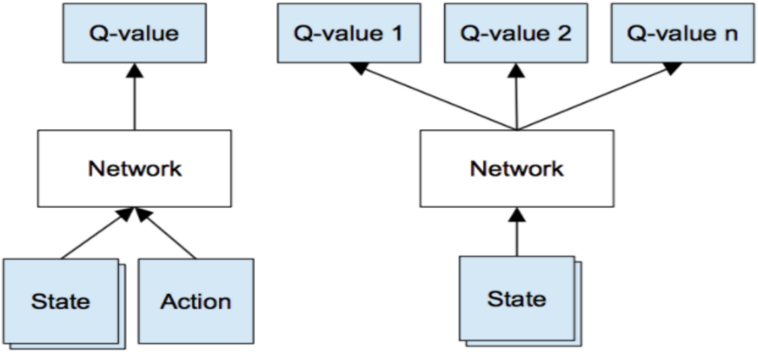
- Huge memory demand, non-scalable;
- Impossible to list all states in more complicated scenarios.

Solution: Use functions to approximate Q.

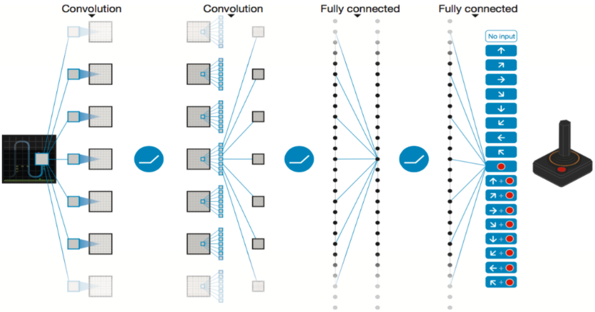
Deep Q-learning: Neural Network + Q-learning

Idea: represent Q-function with a neural network, that takes the state and action as input and outputs the corresponding Q-value.

~~Update Q table each episode~~ $\rightarrow$ Update parameters of deep Q-network(DQN).



Deep Q-learning



- Four continuous 84×84 images.
- Two convolution layers.
- Two denses.
- Output:vectors including every action's Q value.

Deep Q-learning Algorithm

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 action selection
    otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$
    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 Experience Replay
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$ target Network
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ 更新Q-network权重
    Every C steps reset $\hat{Q} = Q$ 每隔C步更新target network
  End For
End For

However,

DQN can only handle **discrete** and **low-dimension action spaces**.

Recall:

- Learning Policy:

$$\arg\max_{a} Q(\phi(s_t), a; \theta)$$

- Update Rule:

$$y_j := r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta^-)$$

Discretize the action space?

- **not scalable**

  actions increases exponentially
  $3^7 = 2187$

- **not fine-tuning**

  naive discretization removes crucial information:
  the structure of $\mathcal{A}$, the optimal action

Policy-Based Methods

Deterministic Policy:

$$\mu_\theta : \mathcal{S} \to \mathcal{A}$$

where $\theta \in \mathbb{R}^n$ is a vector of parameters.

Agent's goal: obtain an optimal policy which maximizes the expected discounted return from the start state, denoted by the performance objective

$$J(\mu_\theta) := \mathbf{E}[G_1 | \mu_\theta] \tag{3}$$

$$J(\mu_\theta) := \mathbf{E}[G_1|\mu_\theta]$$

In a more specific way,

$$\begin{aligned} J(\mu_\theta) =& \int_{\mathcal{S}} \rho^{\mu}(s) r(s, \mu_\theta) ds \\ =& \mathbf{E}_{s \sim \rho^{\mu}}[r(s, \mu_\theta(s))], \end{aligned} \tag{4}$$

where ρ^{μ}(s') is the (improper) discounted state distribution:

$$\rho^{\pi}(s') := \int_{\mathcal{S}} \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p(s \to s', t, \pi) ds \tag{5}$$

Deterministic Policy Ascent

Theorem 1

Deterministic Policy Gradient Theorem (Silver et al., 2014)

$$\begin{aligned}\nabla_\theta J(\mu_\theta) =& \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta Q^\mu(s, \mu_\theta(s)) ds \\ =& \mathbf{E}_{s \sim \rho^\mu}[\nabla_\theta Q^\mu(s, \mu_\theta(s))] \\ =& \mathbf{E}_{s \sim \rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}]\end{aligned} \tag{6}$$

Question:

How to estimate the action-value function $Q^\pi(s, a)$?

Actor-Critic

- **Actor:**
  adjust the stochastic policy $\mu_\theta(s)$ by stochastic gradient ascent.

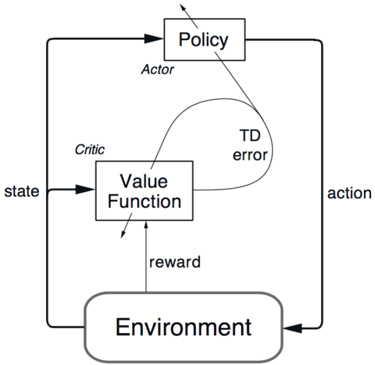
- **Critic:** inspired by the success of DQN
  estimate the action-value function

$$Q^\pi(s,a) \approx Q^{\mu_\theta}(s,a) \tag{7}$$

DNN? Experience Replay? Target Network?

Off-Policy Deterministic Policy Gradient(Silver et.al, 2014)

$$\nabla_{\theta^\mu}\mu \approx \mathbf{E}_{\mu'}[\nabla_{\theta^\mu}Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t|\theta^\mu)}] \tag{8}$$
$$= \mathbf{E}_{\mu'}[\nabla_a Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t)}\nabla_{\theta_\mu}\mu(s|\theta^\mu)|_{s=s_t}] \tag{9}$$



Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**

Learning Policy

    Select action $\boxed{a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t}$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    $\boxed{\text{Set } y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})}$

Update Rule for $Q(s, a; \theta^Q)$

    Update critic by minimizing the loss: $\boxed{L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2)}$
    Update the actor policy using the sampled gradient:

$$\boxed{\nabla_{\theta^\mu} \mu|_{s_i} \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}}$$

Update Rule for μ_{θ^μ}

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

Experience Replay

    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q)^2)$
    Update the actor policy using the sampled gradient:

$$\nabla_{\theta^\mu} \mu|_{s_i} \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

Target Network

  end for
end for

Asynchronous One-Step Q-learning

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

```
// Assume global shared θ, θ⁻, and counter T = 0.
Initialize thread step counter t ← 0
Initialize target network weights θ⁻ ← θ
Initialize network gradients dθ ← 0
Get initial state s
repeat
    Take action a with ε-greedy policy based on Q(s, a; θ)
    Receive new state s' and reward r
    y = { r                              for terminal s'
        { r + γ max_a' Q(s', a'; θ⁻)     for non-terminal s'
    Accumulate gradients wrt θ: dθ ← dθ + ∂(y − Q(s, a; θ))² / ∂θ
    s = s'
    T ← T + 1 and t ← t + 1
    if T mod I_target == 0 then
        Update the target network θ⁻ ← θ
    end if
    if t mod I_AsyncUpdate == 0 or s is terminal then
        Perform asynchronous update of θ using dθ.
        Clear gradients dθ ← 0.
    end if
until T > T_max
```

Asynchronous Advantage Actor-Critic(A3C)

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```
// Assume global shared parameter vectors θ and θ_v and global shared counter T = 0
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter t ← 1
repeat
    Reset gradients: dθ ← 0 and dθ_v ← 0.
    Synchronize thread-specific parameters θ' = θ and θ'_v = θ_v
    t_start = t
    Get state s_t
    repeat
        Perform a_t according to policy π(a_t|s_t; θ')
        Receive reward r_t and new state s_{t+1}
        t ← t + 1
        T ← T + 1
    until terminal s_t or t − t_start == t_max
    R = { 0             for terminal s_t
        { V(s_t, θ'_v)  for non-terminal s_t// Bootstrap from last state
    for i ∈ {t − 1, . . . , t_start} do
        R ← r_i + γR
        Accumulate gradients wrt θ': dθ ← dθ + ∇_θ' log π(a_i|s_i; θ')(R − V(s_i; θ'_v))
        Accumulate gradients wrt θ'_v: dθ_v ← dθ_v + ∂(R − V(s_i; θ'_v))^2/∂θ'_v
    end for
    Perform asynchronous update of θ using dθ and of θ_v using dθ_v.
until T > T_max
```

MengDi Wang: Primal-Dual π

Infinite-horizon Average-reward MDP

- Maximize value function

$$\bar{v}^{\pi} \equiv \bar{v}^{\pi}(i) = \lim_{N\to\infty} \mathbf{E}^{\pi}\left[\frac{1}{N}\sum_{t=1}^{N} r_{i_t i_{t+1}}(a_t) \,\middle|\, i_1 = i\right], \; i \in \mathcal{S},$$

$$h_i^* = \lim_{N\to\infty} \mathbf{E}^{\pi^*}\left[\sum_{t=1}^{N} r_{i_t i_{t+1}}(a_t) - N\bar{v}^* \,\middle|\, i_1 = i\right], \qquad \forall\, i \in \mathcal{S}.$$

  -

- Bellman equation

$$\bar{v}^* + h_i^* = \max_{a\in\mathcal{A}}\left\{\sum_{j\in\mathcal{S}} p_{ij}(a)h_j^* + \sum_{j\in\mathcal{S}} p_{ij}(a)r_{ij}(a)\right\}, \qquad \forall\, i \in \mathcal{S},$$

Linear Programming

- Primal

$$\begin{aligned} &\text{minimize}_{\bar{v},\mathbf{h}} \quad \bar{v} \\ &\text{subject to} \quad \bar{v}\cdot\mathbf{1} + (I - P_a)\,\mathrm{h} - \mathbf{r}_a \geq 0, \qquad \forall\, a \in \mathcal{A}, \end{aligned}$$

- Dual

$$\begin{aligned} \text{maximize} \quad & \sum_{a\in\mathcal{A}} \mu_a^\top \mathbf{r_a} \\ \text{subject to} \quad & \sum_{a\in\mathcal{A}} \left(I - P_a^\top\right)\mu_a = 0, \\ & \sum_{a\in\mathcal{A}}\sum_{i\in\mathcal{S}} \mu_{a,i} = 1, \quad \mu_a \geq 0, \ \ \forall\, a\in\mathcal{A}. \end{aligned}$$

- Minimax formulation

$$\min_{\mathbf{h}\in\mathcal{H}} \max_{\mu\in\mathcal{U}} \sum_{a\in\mathcal{A}} \mu_a^\top \left((P_a - I)\mathbf{h} + \mathbf{r}_a\right).$$

$$\mathcal{H} = \left\{\mathbf{h} \in \Re^{|\mathcal{S}|} \;\middle|\; \|\mathbf{h}\|_\infty \leq 2t^*_{mix}\right\},$$

$$\mathcal{U} = \left\{\mu = (\mu_a)_{a\in\mathcal{A}} \;\middle|\; \mathbf{1}^\top\mu = 1, \mu \geq 0, \sum_{a\in\mathcal{A}} \mu_a \geq \frac{1}{\sqrt{\tau}|\mathcal{S}|}\mathbf{1}\right\}.$$

Dual-ascent-primal-descent

$$\mu^{t+1} = \mathrm{argmin}_{\mu \in \mathcal{U}.} D_{KL}(\mu || \mu^t \cdot \exp(\Delta^{t+1})),$$
$$\mathbf{h}^{t+1} = \mathbf{Proj}_{\mathcal{H}} \left[\mathbf{h}^t + \mathbf{d}^{t+1} \right],$$

$$\Delta^{t+1} \mid \mathcal{F}_t = \beta \cdot \frac{h_j^t - h_i^t + r_{ij}(a) - M}{\mu_{i,a}^t} \mathbf{e}_{i,a}, \qquad \text{with probability } \mu_{i,a}^t,$$
$$\mathbf{d}^{t+1} \mid \mathcal{F}_t = \alpha \cdot (\mathbf{e}_i - \mathbf{e}_j), \qquad \text{with probability } \mu_{i,a}^t p_{i,j}(a),$$

$$\mathbf{E}\left[\Delta_a^{t+1} \mid \mathcal{F}_t\right] = \beta \left((P_a - I)\mathbf{h}^t + \mathbf{r}_a - M \cdot \mathbf{1} \right), \qquad a \in \mathcal{A},$$

$$\mathbf{E}\left[\mathbf{d}^{t+1} \mid \mathcal{F}_t\right] = \alpha \sum_{a \in \mathcal{A}} \mu_a^\top (I - P_a).$$

Simplify for implementation

$$\xi_i^t = \sum_{a \in \mathcal{A}} \mu_{i,a}^t, \qquad \pi_{i,a}^t = \frac{\mu_{i,a}^t}{\xi_i^t}, \qquad \mu_{i,a}^t = \xi_i^t \pi_{i,a}^t \qquad \forall\, i \in \mathcal{S}, a \in \mathcal{A}.$$

Algorithm

Algorithm 1 Primal-Dual π Learning

Input: Precision level $\epsilon > 0$, $\mathcal{S}$, $\mathcal{A}$, t^*_{mix}, τ, $\mathcal{SO}$
Set $\mathrm{h} = 0 \in \Re^{|\mathcal{S}|}$, $\xi = \frac{1}{|\mathcal{S}|}\mathbf{1} \in \Re^{|\mathcal{S}|}$, $\pi_i = \frac{1}{|\mathcal{A}|}\mathbf{1} \in \Re^{|\mathcal{A}|}$ for all $i \in \mathcal{S}$
Set $T = \tau^2(t^*_{mix})^2|\mathcal{S}||\mathcal{A}|$
Set $\beta = \frac{1}{t^*_{mix}}\sqrt{\frac{\log(|\mathcal{S}||\mathcal{A}|)}{2|\mathcal{S}||\mathcal{A}|T}}, \alpha = |\mathcal{S}|t^*_{mix}\sqrt{\frac{\log(|\mathcal{S}||\mathcal{A}|)}{2|\mathcal{A}|T}}, M = 4t^*_{mix} + 1$
for $t = 1, 2, 3, \ldots, T$ **do**
  Sample (i, a) with probability $\xi_i \pi_{i,a}$
  Sample j with probability $p_{ij}(a)$ from $\mathcal{SO}$
  $\Delta \leftarrow \beta \cdot \frac{h^t_j - h^t_i + r_{ij}(a) - M}{\xi^t_{i,a}\pi^t_{i,a}}$
  $h_i \leftarrow \min\{h_i + \alpha, 2t^*_{mix}\}, h_j \leftarrow \max\{h_j - \alpha, -2t^*_{mix}\}$,
  $\xi_i \leftarrow \xi_i + \pi_{i,a}\left(\exp\{\Delta\} - 1\right), \xi \leftarrow \mathrm{argmin}_{\xi}\left\{D_{KL}(\hat{\xi}\|\xi) \mid \mathbf{1}^\top \hat{\xi} = 1, \hat{\xi} \geq 0, \hat{\xi} \geq \frac{1}{\sqrt{\tau}|\mathcal{S}|}\mathbf{1}\right\}$
  $\pi_{i,a} \leftarrow \pi_{i,a} \cdot \exp\{\Delta\}, \pi_i \leftarrow \pi_i / \|\pi_i\|_1$
  $\pi^{t+1} \leftarrow \pi$
  $t \leftarrow t + 1$
end for
Ouput: $\hat{\pi} = \frac{1}{T}\sum_{t=1}^{T} \pi^t$

Convergence

Theorem 1 (Finite-Iteration Duality Gap). *Let $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathrm{r})$ be an arbitrary MDP tuple satisfying Assumptions 1, 2. Then the sequence of iterates generated by Algorithm 1 satisfies*

$$\frac{1}{T}\sum_{t=1}^{T} \mathrm{E}\left[\sum_{a\in\mathcal{A}} (\mathrm{h}^* - P_a\mathrm{h}^* - \mathrm{r}_a)^\top \mu_a^t\right] + \bar{v}^* \leq \tilde{\mathcal{O}}\left(t_{mix}^*\sqrt{\frac{|\mathcal{S}||\mathcal{A}|}{T}}\right),$$

where $\mu_{i,a}^t = \xi_i^t \pi_{i,a}^t$ for $i \in \mathcal{S}, a \in \mathcal{A}$, $t = 1, \ldots, T$.

References:

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards." PhD thesis. King's College, Cambridge, 1989.