

Non-deterministic Rewards and Actions

Sargur N. Srihari
srihari@cedar.buffalo.edu

Topics in Non-deterministic Rewards and Actions

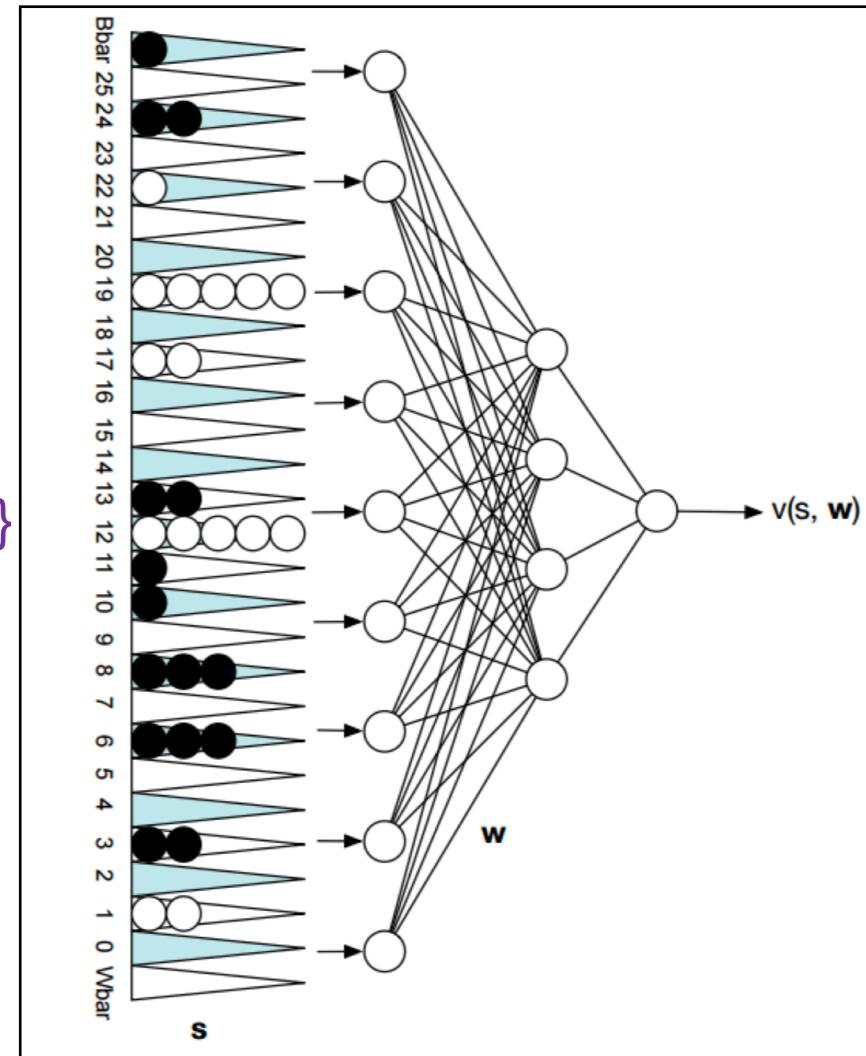
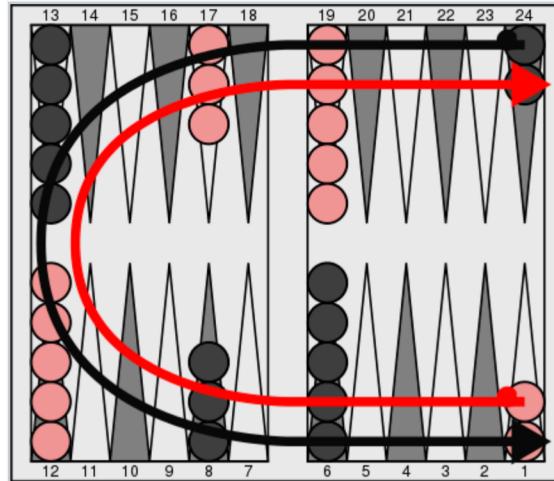
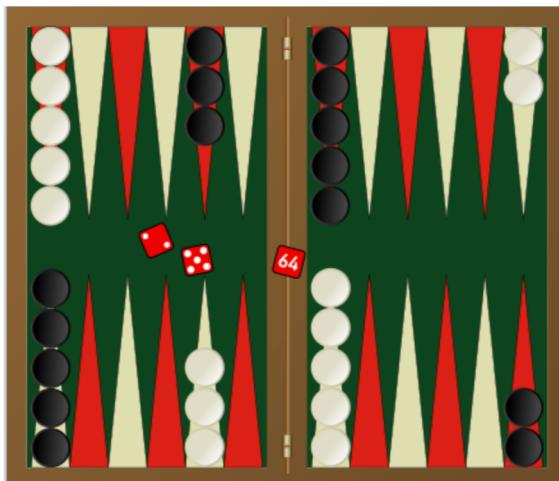
1. Nondeterministic Actions and Rewards
2. Examples of non-determinism
3. Non-deterministic MDP
4. Revised reinforcement training rule

Nondeterministic Rewards and Actions

- We have seen Q learning in deterministic environments
- In the nondeterministic case reward $r(s,a)$ and function $\delta(s,a)$ have probabilistic outcomes
- Examples
 1. In Backgammon, action outcomes are probabilistic
 - Because each move involves a roll of the dice
 2. In robot problems with noisy sensors and effectors, it is appropriate to model actions and rewards as probabilistic

Non-determinism in Backgammon

- Action
 - Move based on 2 dice
- State:
 - 24 board positions, 15 pieces each
 - 9D vector: $\{-1, 0, 3, -2, 0, 2, -3, 0, 1\}$



Example of Non-determinism: Robotics

- Robotics
 - Noisy sensors and effectors
 - Appropriate to model actions and rewards as nondeterministic



Recall Markov Decision Process

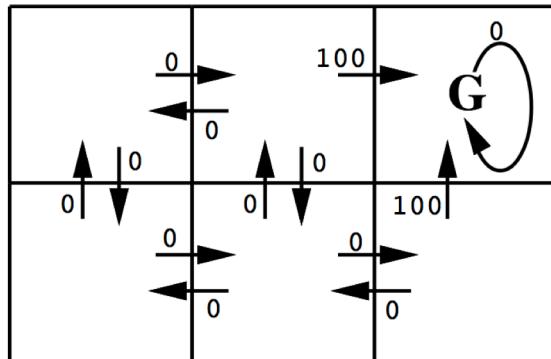
- Agent can perceive a set S of discrete states in its environment
- Has a set A of actions that it can perform
- At each discrete time step t the agent senses the current state s_t , chooses a current action a_t and performs it
- Environment responds by giving it a reward $r_t = r(s_t, a_t)$ and by producing state $s_{t+1} = \delta(s_t, a_t)$
 - The function δ and r are part of the environment and not necessarily known to the agent

Nondeterministic MDP

- Functions $r(s, a)$ and $\delta(s, a)$ can be viewed as
 - First producing a probability distribution over outcomes based on s and a and
 - Then drawing an outcome at random according to this distribution
- Nondeterministic Markov decision process
 - When these probability distributions depend solely on s and a , i.e.,
 - They do not depend on previous states or actions

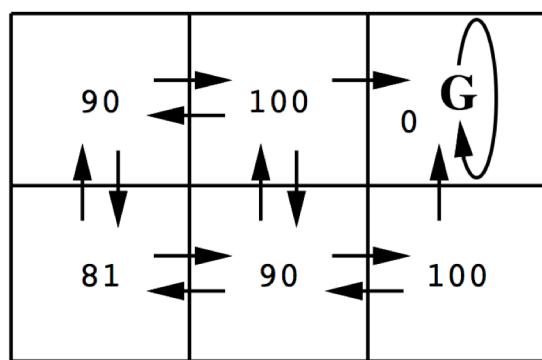
Recall Q Learning

$r(s, a)$
(Immediate Reward)

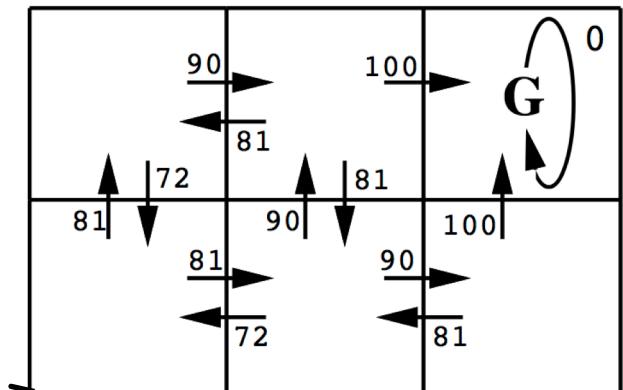


$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

$V^*(s)$
(Maximum Discounted Cumulative Reward)



Quality of Action $Q(s, a)$ values



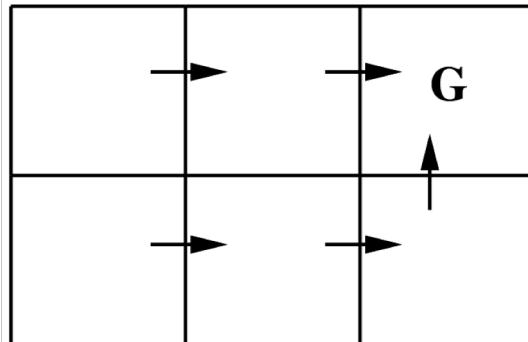
Recurrent Definition

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

$$V^*(s) = \max_{a'} Q(s, a')$$

One Optimal policy



$$\pi^*(s) = \arg \max_{\pi} [r(s, a) + \gamma V^*(\delta(s, a))]$$

$$\pi^*(s) = \arg \max_{\pi} Q(s, a)$$

Extend Q learning to nondeterminism

- Retrace the line of argument for the deterministic case
 - Revise it where needed
- In the deterministic case
 - Cumulative value $V^\pi(s_t)$ achieved by following policy π from initial state s_t is
$$V^\pi(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$
- In the nondeterministic case it is
 - Expected value (over nondeterministic outcomes)
 - of the discounted cumulative reward received by policy

$$V^\pi(s_t) = E \left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \right]$$

Q function for nondeterministic case

- In the deterministic case, Q -function defined as

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

- We generalize again by taking expected value

$$\begin{aligned} Q(s, a) &= E[r(s, a) + \gamma V^*(\delta(s, a))] \\ &= E[r(s, a)] + \gamma E[V^*\delta(s, a)] \\ &= E[r(s, a)] + \gamma \sum_{s'} P(s' | s, a) V^*(s') \end{aligned}$$

- Where $P(s' | s, a)$ is the probability that taking action a in state s will produce next state s'
 - Note: $P(s' | s, a)$ replaces the expected value of the probabilistic δ

MDP with probabilistic state transition

MDP is a 5-tuple (S , A , P_a , R_a , γ)

where

1. S is a finite set of states,

2. A is a finite set of actions

(alternatively, A_s is set of actions from s),

$$\delta(s,a)$$

3. $P_a(s,s') = P(s_{t+1} = s' | s_t = s, a_t = a)$

is probability that action a in state s at time t will lead to state s' at time $t+1$,

[Note: compare to deterministic case

$$\delta(s_t, a_t) = s_{t+1}]$$

4. $R_a(s, s')$ is the immediate reward

(or expected immediate reward)

received after transitioning from state s to state s' , due to action a

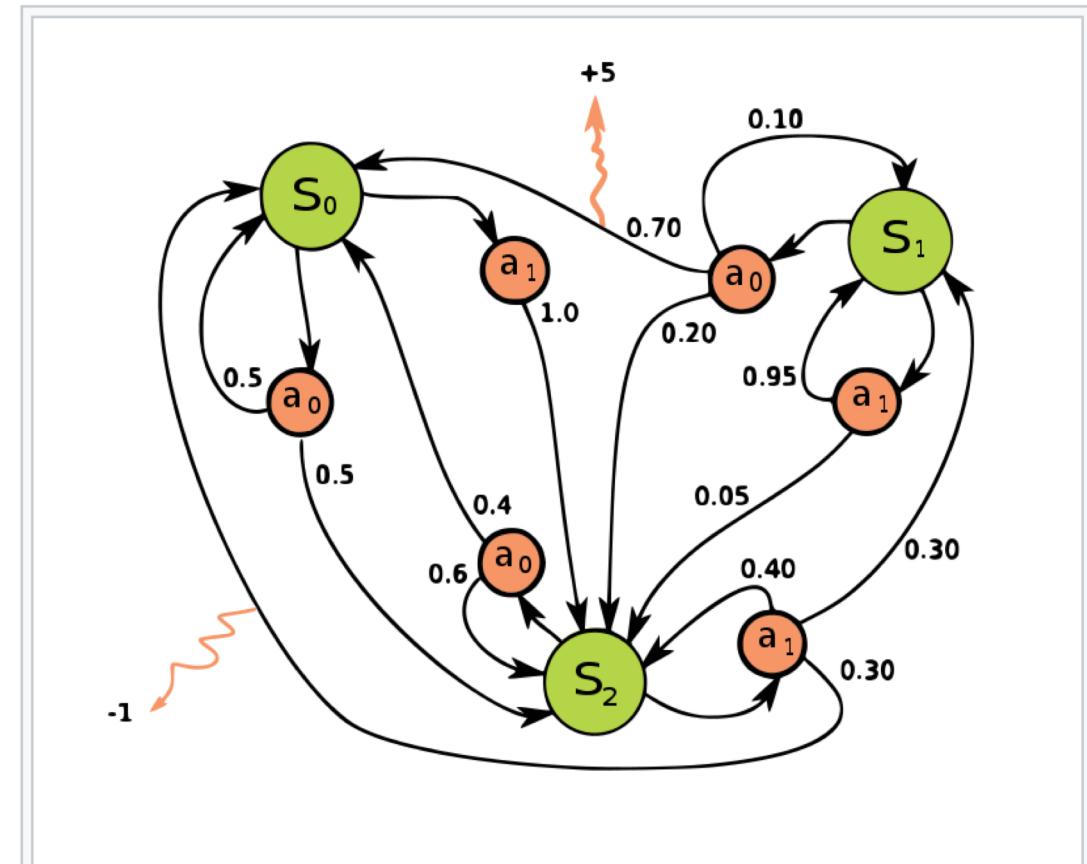
5. $\gamma \in [0,1)$ is the discount factor,

which represents the difference in

importance between future rewards and

present rewards.

$$S=\{S_0, S_1, S_2\}, A=\{a_0, a_1\}, r_{a0}(S_1, S_0)=+5, r_{a1}(S_2, S_0)=-1$$



Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows).

Recursive expression for Q

- In the deterministic case

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

- In the nondeterministic case we can express Q recursively as

$$Q(s, a) = E[r(s, a)] + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a')$$

- To summarize
 - We have redefined $Q(s, a)$ as the expected value of the quantity in the deterministic case

Non-deterministic training rule

- Q -learning algorithm for deterministic case is

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a')$$

- We need a new training rule to accommodate nondeterministic environment functions r and δ
 - Deterministic rule fails to converge in this setting
 - $r(s, a)$ produces different rewards each time the transition (s, a) is repeated. In this case training rule repeatedly alters $\hat{Q}(s, a)$ even if we initialize it to correct Q function
 - Difficulty is overcome by modifying training rule
 - so it takes a decaying weighted average of the current \hat{Q} value and the revised estimate

Revised training rule

- Writing \hat{Q}_n to denote the agent's estimate on the n^{th} iteration, the following training rule assures convergence

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n)\hat{Q}_{n-1}(s, a) + \alpha_n \left[r + \gamma \max_{a'} \hat{Q}_{n-1}(s', a') \right]$$

– Where

action a in state s will produce next state s'

$$\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)}$$

- s and a are the state and action updated during the n^{th} iteration
- $\text{visits}_n(s, a)$ is the total no of times state action pair has been visited upto and including the n^{th} iteration

Key idea of revised rule

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n) \hat{Q}_{n-1}(s, a) + \alpha_n \left[r + \gamma \max_{a'} \hat{Q}_{n-1}(s', a') \right]$$

- Revisions to \hat{Q} are made more gradually
 - If we set α_n to 1 then we would have exactly the training rule for the deterministic case

$$\hat{Q}_n(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}_{n-1}(s', a')$$
 - With smaller α_n this term is now averaged in with the current $\hat{Q}(s, a)$ to produce the updated value
- α_n as defined by

$$\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)}$$
 - decreases as n increases, so that updates become smaller as training progresses

Q-Learning with probabilistic $\delta(s, a)$

Q-learning: Learn function $Q : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$

Require:

States $\mathcal{X} = \{1, \dots, n_x\}$

Actions $\mathcal{A} = \{1, \dots, n_a\}, \quad A : \mathcal{X} \Rightarrow \mathcal{A}$

Reward function $R : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$

Black-box (probabilistic) transition function $T : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{X}$

Learning rate $\alpha \in [0, 1]$, typically $\alpha = 0.1$

Discounting factor $\gamma \in [0, 1]$

procedure QLEARNING($\mathcal{X}, A, R, T, \alpha, \gamma$)

 Initialize $Q : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$ arbitrarily

while Q is not converged **do**

 Start in state $s \in \mathcal{X}$

while s is not terminal **do**

 Calculate π according to Q and exploration strategy (e.g. $\pi(x) \leftarrow \arg \max_a Q(x, a)$)

$a \leftarrow \pi(s)$

$r \leftarrow R(s, a)$

 ▷ Receive the reward

$s' \leftarrow T(s, a)$

 ▷ Receive the new state

$Q(s', a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a'))$

return $\overset{s}{\underset{s'}{\overleftarrow{Q}}}$

Convergence of Q learning for nondeterministic MDP

- Consider a Q -learning agent in a nondeterministic MDP with bounded rewards $\forall s, a, |r(s, a)| \leq c$
- The Q -learning agent uses the training rule

$$\hat{Q}_n(s, a) \leftarrow (1 - \alpha_n)\hat{Q}_{n-1}(s, a) + \alpha_n \left[r + \gamma \max_{a'} \hat{Q}_{n-1}(s', a') \right]$$

- Initialize table $\hat{Q}(s, a)$ to arbitrary finite values and use a discount factor $0 \leq \gamma < 1$
- Let $n(i, s, a)$ be the iteration corresponding to the i^{th} time that action a is applied to state s
- If each (s, a) is visited infinitely often, $0 \leq \gamma < 1$, and

$$\sum_{i=1}^{\infty} \alpha_{n(i, s, a)} = \infty, \quad \sum_{i=1}^{\infty} [\alpha_{n(i, s, a)}]^2 = \infty$$

- then $\forall s, a, \hat{Q}_n(s, a) \rightarrow Q(s, a)$ as $n \rightarrow \infty$ with probability 1

Non-deterministic MDP



Industrial Robotics

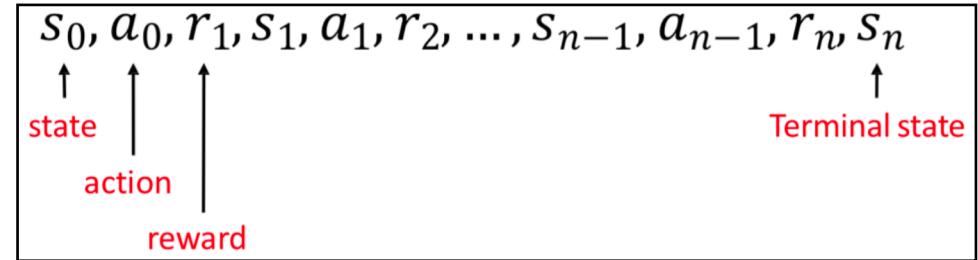
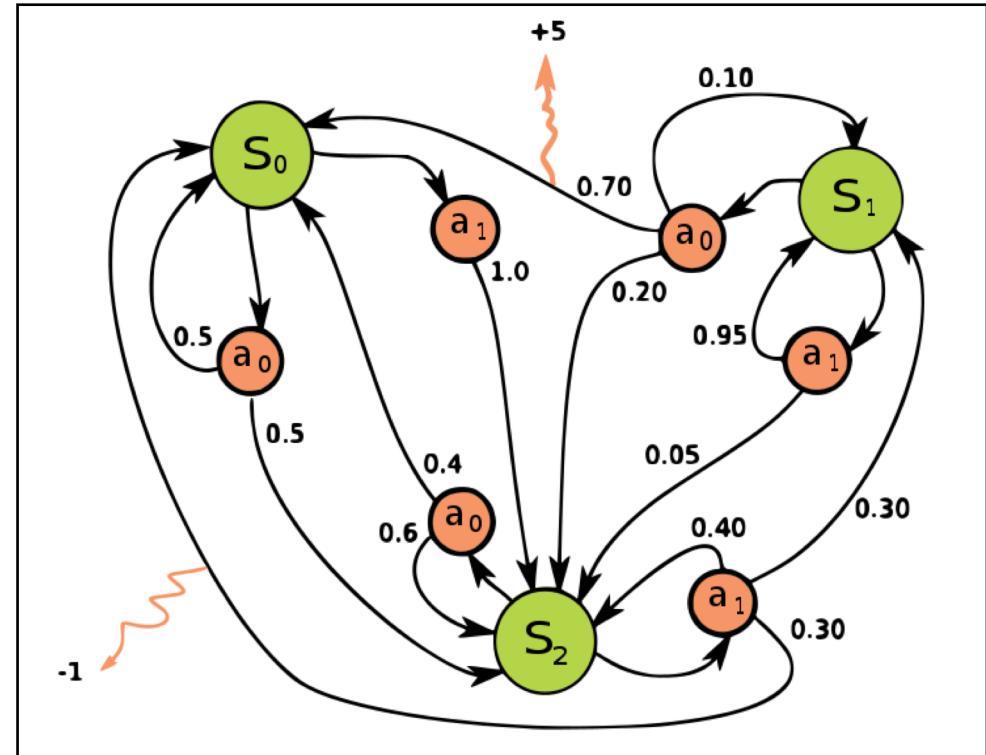
Task: Bin Packing

Goal G:

Pick a box and put it into a container.

State s:

Raw Pixels of the world.



Need for large number of iterations

- While Q-learning and related reinforcement learning algorithms can be proven to converge under certain conditions, in practice systems that use Q-learning often require thousands of iterations to converge
- TD-Gammon trained for 1.5 million backgammon games, each of which contained tens of state-action transitions