Short introduction to motion planning and control

Karel Zimmermann

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics
Center for Machine Perception
http://cmp.felk.cvut.cz/~zimmerk, zimmerk@fel.cvut.cz

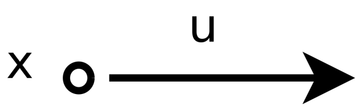
MDP definition

- States: $\mathbf{x} \in X$

x

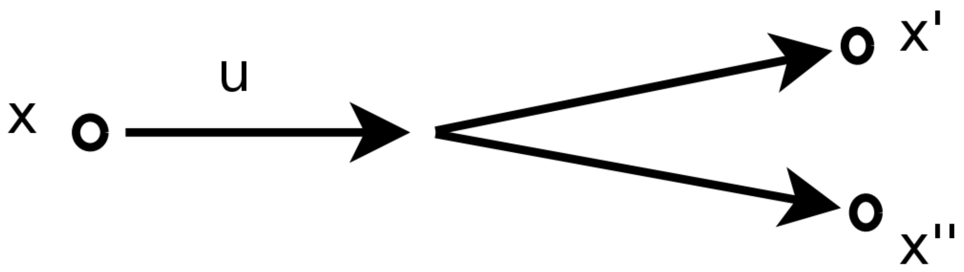
MDP definition

- States: $\mathbf{x} \in X$
- Actions: $\mathbf{u} \in U$



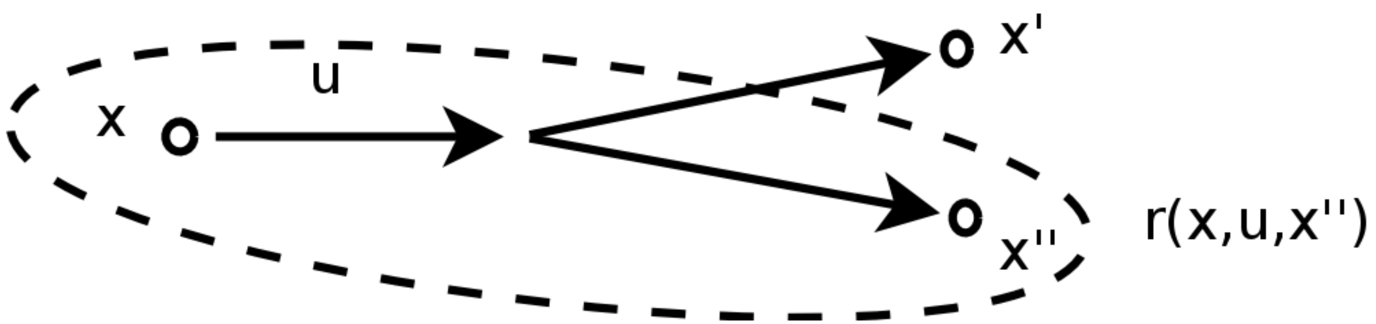
MDP definition

- States: $\mathbf{x} \in X$
- Actions: $\mathbf{u} \in U$
- Transition probability: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u}) : X \times U \times X \to [0; 1]$



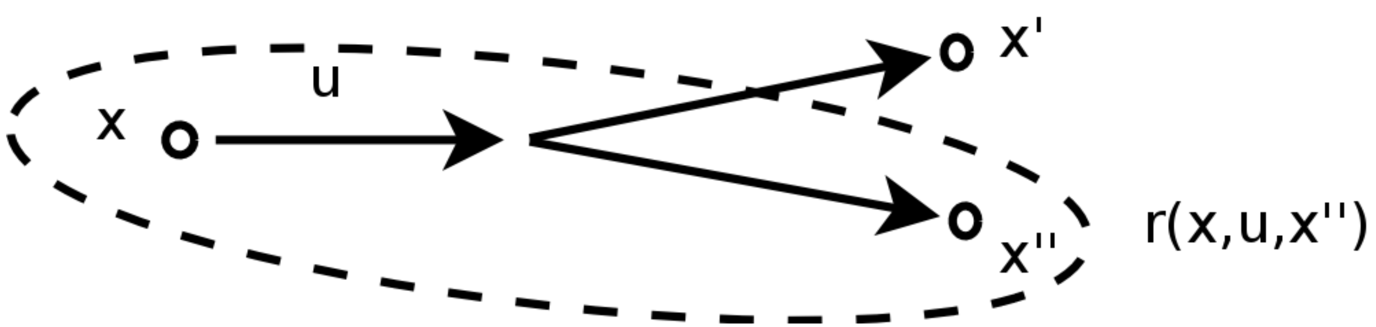
MDP definition

- States: $\mathbf{x} \in X$
- Actions: $\mathbf{u} \in U$
- Transition probability: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u}) : X \times U \times X \to [0;1]$
- Reward: $r(\mathbf{x}, \mathbf{u}, \mathbf{x}') : X \times U \times X \to \mathbb{R}$



MDP definition

- States: $\mathbf{x} \in X$
- Actions: $\mathbf{u} \in U$
- Transition probability: $p(\mathbf{x}'|\mathbf{x}, \mathbf{u}) : X \times U \times X \to [0; 1]$
- Reward: $r(\mathbf{x}, \mathbf{u}, \mathbf{x}') : X \times U \times X \to \mathbb{R}$
- Policy: $\pi_\theta(\mathbf{x}) : X \to U$



MDP definition

- Trajectory is sequence of visited states and performed actions:
  $\tau = (\mathbf{x}_0, \mathbf{u}_0,\ \ \mathbf{x}_1, \mathbf{u}_1,\ \ \mathbf{x}_2, \dots)$

MDP definition

- Trajectory is sequence of visited states and performed actions: $\tau = (\mathbf{x}_0, \mathbf{u}_0, \ \ \mathbf{x}_1, \mathbf{u}_1, \ \ \mathbf{x}_2, \dots)$
- Sum of rewards with limited horizont:

$$r(\tau) = \sum_{i=0}^{H} r(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{i+1})$$

MDP definition

- Trajectory is sequence of visited states and performed actions:
  $\tau = (\mathbf{x}_0, \mathbf{u}_0, \;\; \mathbf{x}_1, \mathbf{u}_1, \;\; \mathbf{x}_2, \dots)$
- Sum of rewards with limited horizont:

$$r(\tau) = \sum_{i=0}^{H} r(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{i+1})$$

- Sum of discounted rewards:

$$r(\tau) = \sum_{i=0}^{\infty} \gamma^i \cdot r(\mathbf{x}_i, \mathbf{u}_i, \mathbf{x}_{i+1})$$

Different platforms

- Many different robots with various complexity of dynamics and dimensionality of state-action space:
  - triple pendulum,
  - two-arm manipulator,
  - mobile platform with auxiliary articulated sub-tracks and lockable differential.

Different problems

- Find the shortest collision-free trajectory in configuration space from start state(s) $\mathbf{x}_s$ to goal state(s) $\mathbf{x}_g$ which respect dynamic constraints of the robot (e.g. Dijkstra, A*, RRT, PRM, LQR-trees or Guided Policy Search)

Different problems

- Find the shortest collision-free trajectory in configuration space from start state(s) $\mathbf{x}_s$ to goal state(s) $\mathbf{x}_g$ which respect dynamic constraints of the robot (e.g. Dijkstra, A*, RRT, PRM, LQR-trees or Guided Policy Search)
- Find the shortest surveillance trajectory which covers all dangerous states (e.g. TSP, RITA)

Different problems

- Find the shortest collision-free trajectory in configuration space from start state(s) $\mathbf{x}_s$ to goal state(s) $\mathbf{x}_g$ which respect dynamic constraints of the robot (e.g. Dijkstra, A*, RRT, PRM, LQR-trees or Guided Policy Search)
- Find the shortest surveillance trajectory which covers all dangerous states (e.g. TSP, RITA)
- Find policy that determines hidden state (active perception, exploration)

Different problems

- Find the shortest collision-free trajectory in configuration space from start state(s) $\mathbf{x}_s$ to goal state(s) $\mathbf{x}_g$ which respect dynamic constraints of the robot (e.g. Dijkstra, A*, RRT, PRM, LQR-trees or Guided Policy Search)
- Find the shortest surveillance trajectory which covers all dangerous states (e.g. TSP, RITA)
- Find policy that determines hidden state (active perception, exploration)

Outline

Planning vs motion control

- Planning requires transition probability, policy is sequence of actions.
- Motion control can use but does not necessarily requires transition probability, it learns policy function.

Lecture plan:

1. Path planning via Rapidly Exploring Random Trees (RRT).
2. Reinforcement learning for robotics (with and without motion model).

Planning

- Open Motion Planning Library: http://wiki.ros.org/ompl

Planning

- Open Motion Planning Library: http://wiki.ros.org/ompl
- The most direct approach to planning is to search: Depth-first search, Breadth-first search, A*, Dijkstra

Planning

- Open Motion Planning Library: http://wiki.ros.org/ompl
- The most direct approach to planning is to search: Depth-first search, Breadth-first search, A*, Dijkstra
- Real robots usually operate in a continuous space.

Planning

- Open Motion Planning Library: http://wiki.ros.org/ompl
- The most direct approach to planning is to search: Depth-first search, Breadth-first search, A*, Dijkstra
- Real robots usually operate in a continuous space.
- Search in continuous high-dimensional space can get stuck in a local minimum, since you can expand infinite number of nodes in a small region.

Planning

- Open Motion Planning Library: http://wiki.ros.org/ompl
- The most direct approach to planning is to search: Depth-first search, Breadth-first search, A*, Dijkstra
- Real robots usually operate in a continuous space.
- Search in continuous high-dimensional space can get stuck in a local minimum, since you can expand infinite number of nodes in a small region.
- RRT [1] efficiently search non-convex, high-dimensional spaces by randomly building a space-filling tree.
- Initial publication [1] has over 1200 citations.

[1] LaValle, Steven M. (October 1998). "Rapidly-exploring random trees: A new tool for path planning". Technical Report (Computer Science Department, Iowa State University) (TR 98-11).

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$
2. Pick a point $\mathbf{x} \in X$ at random.

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$
2. Pick a point $\mathbf{x} \in X$ at random.
3. Check if $\mathbf{x}$ is admissible (e.g. collision-free via direct kinematics)

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$
2. Pick a point $\mathbf{x} \in X$ at random.
3. Check if $\mathbf{x}$ is admissible (e.g. collision-free via direct kinematics)
4. Find the closest node to $\mathbf{x}$ (e.g. KD tree) $\mathbf{y}^* = \arg\min_{\mathbf{y}} \|\mathbf{y} - \mathbf{x}\|$

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$
2. Pick a point $\mathbf{x} \in X$ at random.
3. Check if $\mathbf{x}$ is admissible (e.g. collision-free via direct kinematics)
4. Find the closest node to $\mathbf{x}$ (e.g. KD tree) $\mathbf{y}^* = \arg\min_{\mathbf{y}} \|\mathbf{y} - \mathbf{x}\|$
5. Try connect $\mathbf{y}^*$ with $\mathbf{x}$
   - if possible: add node $\mathbf{x}$ and edge $[\mathbf{y}^*, \mathbf{x}]$ to the search tree.
   - otherwise: throw $\mathbf{x}$ away.

Algorithm of Rapidly Exploring Random Tree

Input: $\mathbf{x}_s$, $\mathbf{x}_g$

1. Initialize search tree by node $\mathbf{x}_s$
2. Pick a point $\mathbf{x} \in X$ at random.
3. Check if $\mathbf{x}$ is admissible (e.g. collision-free via direct kinematics)
4. Find the closest node to $\mathbf{x}$ (e.g. KD tree) $\mathbf{y}^* = \arg\min_{\mathbf{y}} \|\mathbf{y} - \mathbf{x}\|$
5. Try connect $\mathbf{y}^*$ with $\mathbf{x}$
   - if possible: add node $\mathbf{x}$ and edge $[\mathbf{y}^*, \mathbf{x}]$ to the search tree.
   - otherwise: throw $\mathbf{x}$ away.
6. Repeat from 2 until a feasible path is found.

Output: a collision-free path if exist (after infinite number of nodes expanded)

Speed-ups of Rapidly Exploring Random Tree

- Bias to goal (greediness), e.g.
  - with $P = 0.95$ choose $\mathbf{x}$ at random from X,
  - with $P = 0.05$ choose $\mathbf{x} := \mathbf{x}_g$.

Speed-ups of Rapidly Exploring Random Tree

- Bias to goal (greediness), e.g.
  - with $P = 0.95$ choose $\mathbf{x}$ at random from X,
  - with $P = 0.05$ choose $\mathbf{x} := \mathbf{x}_g$.
- Bi-directional search (e.g. RRT-connect)

Speed-ups of Rapidly Exploring Random Tree

- Bias to goal (greediness), e.g.
  - with $P = 0.95$ choose $\mathbf{x}$ at random from X,
  - with $P = 0.05$ choose $\mathbf{x} := \mathbf{x}_g$.
- Bi-directional search (e.g. RRT-connect)
- If optimality is important:

Speed-ups of Rapidly Exploring Random Tree

- Bias to goal (greediness), e.g.
  - with $P = 0.95$ choose $\mathbf{x}$ at random from X,
  - with $P = 0.05$ choose $\mathbf{x} := \mathbf{x}_g$.
- Bi-directional search (e.g. RRT-connect)
- If optimality is important:
  - informed-RRT [3]

Speed-ups of Rapidly Exploring Random Tree

- Bias to goal (greediness), e.g.
  - with $P = 0.95$ choose $\mathbf{x}$ at random from X,
  - with $P = 0.05$ choose $\mathbf{x} := \mathbf{x}_g$.
- Bi-directional search (e.g. RRT-connect)
- If optimality is important:
  - informed-RRT [3]
  - RRT* [2]

[2] Karaman, Sertac; Frazzoli, Emilio, "Incremental Sampling-based Algorithms for Optimal Motion Planning", 2010

[3] J.D. Gammell, S.S. Srinivasa, T.D. Barfoot, "Informed RRT*", IROS, 2014

Properties of Rapidly Exploring Random Tree

- What might be time consuming?

Properties of Rapidly Exploring Random Tree

- What might be time consuming?
  - Nearest neighbor search (grows with the size of the search tree).
  - Collision checker (grow with the number of modeling elements).
  - Complicated dynamics (explicit motion model might not exist).

Properties of Rapidly Exploring Random Tree

- What might be time consuming?
  - Nearest neighbor search (grows with the size of the search tree).
  - Collision checker (grow with the number of modeling elements).
  - Complicated dynamics (explicit motion model might not exist).
- Voronoi bias: the search tree has bias to grow in large open regions.

Properties of Rapidly Exploring Random Tree

- What might be time consuming?
  - Nearest neighbor search (grows with the size of the search tree).
  - Collision checker (grow with the number of modeling elements).
  - Complicated dynamics (explicit motion model might not exist).
- Voronoi bias: the search tree has bias to grow in large open regions.
- Probabilistic completeness: if nodes number reaches infinity than RRT finds a feasible path (if exists) with $P = 1$.

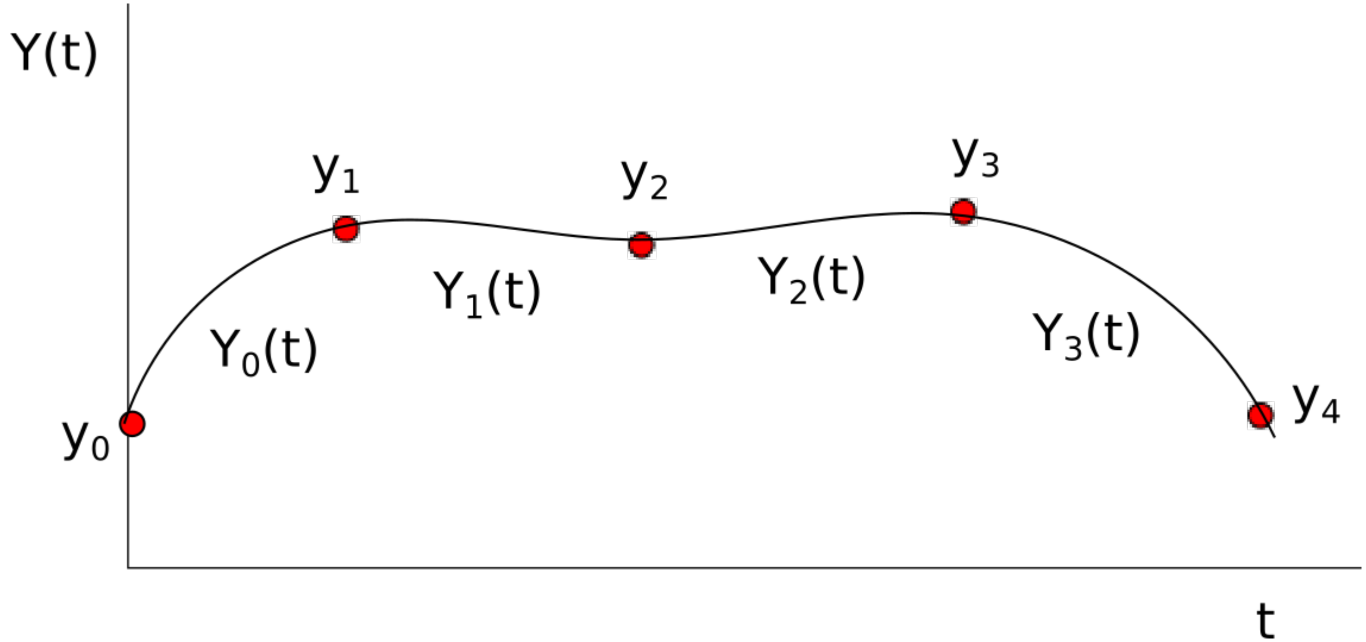
Properties of Rapidly Exploring Random Tree

- What might be time consuming?
  - Nearest neighbor search (grows with the size of the search tree).
  - Collision checker (grow with the number of modeling elements).
  - Complicated dynamics (explicit motion model might not exist).
- Voronoi bias: the search tree has bias to grow in large open regions.
- Probabilistic completeness: if nodes number reaches infinity than RRT finds a feasible path (if exists) with $P = 1$.
- Cost of the best path in the RRT converges almost surely (i.e. with $P = 1$) to a non-optimal value [2].
- Cost of the best path in the RRT* converges almost surely (i.e. with $P = 1$) to the optimal value [2].

Properties of Rapidly Exploring Random Tree

- What might be time consuming?
  - Nearest neighbor search (grows with the size of the search tree).
  - Collision checker (grow with the number of modeling elements).
  - Complicated dynamics (explicit motion model might not exist).
- Voronoi bias: the search tree has bias to grow in large open regions.
- Probabilistic completeness: if nodes number reaches infinity than RRT finds a feasible path (if exists) with $P = 1$.
- Cost of the best path in the RRT converges almost surely (i.e. with $P = 1$) to a non-optimal value [2].
- Cost of the best path in the RRT* converges almost surely (i.e. with $P = 1$) to the optimal value [2].
- If accurate dynamic motion model is known ($\dot{\mathbf{x}}$, $\ddot{\mathbf{x}}$ relations are determined), then RRT searches for trajectory in lifted state-space $[\mathbf{x},\ \dot{\mathbf{x}},\ \ddot{\mathbf{x}}]^\top$.

Smooth path interpolation



- If only kinematic model is available, discrete path is searched (i.e. N points $\mathbf{y}_0 \in X, \ldots \mathbf{y}_{N-1} \in X$ in configuration space).
- Trajectory is defined as a smooth interpolation of path
- Example of smooth interpolation is pair-wise cubic function Y, i.e sequence of $N-1$ cubic functions $Y_i(t) : [0, 1] \rightarrow X$ such that:
  - $Y_i(t)$ connects $\mathbf{y}_i$ with $\mathbf{y}_{i+1}$.
  - Y has continuous first and second derivatives.

Planning summary

1. Advantage: It has statistical guarantees of the optimality.
2. Drawback 1: Whenever results of action differs from the model (i.e. everytime), you need to replan the whole trajectory.
3. Drawback 2: Accurate motion model is required.

For some problems, reinforcement learning trades-off optimality (1) for (2)+(3).

What if motion model is unknown

- Can you design a motion control algorithm without motion model?

Problem definition

- We have a robot and we have no idea how to control it.

Problem definition

- We have a robot and we have no idea how to control it.
- Nevertheless, we know what is good and bad state - we have a definition of rewards.

Problem definition

- We have a robot and we have no idea how to control it.
- Nevertheless, we know what is good and bad state - we have a definition of rewards.
- We control it somehow (e.g. with some initial policy) and record the trajectory τ (or several trajectories).

Problem definition

- We have a robot and we have no idea how to control it.
- Nevertheless, we know what is good and bad state - we have a definition of rewards.
- We control it somehow (e.g. with some initial policy) and record the trajectory τ (or several trajectories).
- Given these trajectories, change the policy to increase expected sum of rewards

$$J(\theta) = E\{r(\tau)\}$$

Problem definition

- Denote $p(\tau|\theta)$ probability of trajectory τ occurs when following policy π_θ

Problem definition

- Denote $p(\tau|\theta)$ probability of trajectory τ occurs when following policy π_θ
- Criterion to be maximized is the expected sum of rewards

$$J(\theta) = E\{r(\tau)\} = \int_{\tau \in \mathcal{T}} p(\tau|\theta) r(\tau) \, \mathrm{d}\tau$$

Problem definition

- Denote $p(\tau|\theta)$ probability of trajectory τ occurs when following policy π_θ
- Criterion to be maximized is the expected sum of rewards

$$J(\theta) = E\{r(\tau)\} = \int_{\tau \in \mathcal{T}} p(\tau|\theta) r(\tau)\,\mathrm{d}\tau$$

- We solve the following optimization problem

$$\theta^* = \arg\max_{\theta} J(\theta)$$

Problem solution

- As usually, you can:

Problem solution

- As usually, you can:
  - either solve **primal task** e.g. by following gradient ∇J to maximize $J(\theta)$ directly.
  - primal is often solved in the optimal control community (e.g. LQR),

Problem solution

- As usually, you can:
  - either solve **primal task** e.g. by following gradient ∇J to maximize $J(\theta)$ directly.
  - primal is often solved in the optimal control community (e.g. LQR),
  - or solve **dual task** to find state-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$ which tells expected sum rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
  - optimal policy $\pi^* = \arg\max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u})$
  - dual is often employed by AI community as heuristics for state-space search
  - Q-values could provide metrics for RRT [Tedrake-LQR-trees-2015]

Dual task provides alternative point-of-view (e.g. shadow prices in LP or sparse feature selection for SVM)

Primal task - approximating criterion.

- Use $\pi_\theta(\mathbf{x})$ to get several trajectories τ_i.

Primal task - approximating criterion.

- Use $\pi_\theta(\mathbf{x})$ to get several trajectories τ_i.
- Approximate criterion value in θ as average reward of trajectories $\tau_i \sim p(\tau|\theta)$ generated with policy π_θ

$$J(\theta) = E\{r(\tau)\} = \int_{\tau \in \mathcal{T}} p(\tau|\theta) r(\tau)\,\mathrm{d}\tau \approx \frac{1}{N}\sum_{i=1}^{N} r(\tau_i)$$

Primal task - approximating criterion.

- Use $\pi_\theta(\mathbf{x})$ to get several trajectories τ_i.
- Approximate criterion value in θ as average reward of trajectories $\tau_i \sim p(\tau|\theta)$ generated with policy π_θ

$$J(\theta) = E\{r(\tau)\} = \int\limits_{\tau \in \mathcal{T}} p(\tau|\theta) r(\tau) \, \mathrm{d}\tau \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i)$$

- We can approximate criterion value, what about gradient?

Primal task - approximating gradient

- Can we obtain the gradient by computing also $J(\theta + \Delta\theta)$?

Primal task - approximating gradient

- Can we obtain the gradient by computing also $J(\theta + \Delta\theta)$?
- Of course, but doing it from one sample is quite unstable (especially for high dimensional θ).

Primal task - approximating gradient

- Can we obtain the gradient by computing also $J(\theta + \Delta\theta)$?
- Of course, but doing it from one sample is quite unstable (especially for high dimensional θ).
- Perform several small random perturbations $\Delta\theta_i$ and compute $J(\theta + \Delta\theta_i)$.

Primal task - approximating gradient

- Can we obtain the gradient by computing also $J(\theta + \Delta\theta)$?
- Of course, but doing it from one sample is quite unstable (especially for high dimensional θ).
- Perform several small random perturbations $\Delta\theta_i$ and compute $J(\theta + \Delta\theta_i)$.
- Relation to gradient $\nabla J(\theta)$ is given by the first order Taylor polynom

$$\begin{aligned}
J(\theta + \Delta\theta_i) &= J(\theta) + \nabla J(\theta)^\top \Delta\theta_i \\
\Delta\theta_i^\top \nabla J(\theta) &= J(\theta) - J(\theta + \Delta\theta_i) \\
\underbrace{\begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}}_{\text{matrix } \mathtt{A}} \nabla J(\theta) &= \underbrace{\begin{bmatrix} J(\theta) - J(\theta + \Delta\theta_1)) \\ \vdots \\ J(\theta) - J(\theta + \Delta\theta_n)) \end{bmatrix}}_{\text{vector } \mathbf{b}}
\end{aligned}$$

Primal task - solution

- Gradient is solution of overdetermined set of linear equations:

$$\nabla J(\theta) = \begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}^+ \cdot \begin{bmatrix} J(\theta) - J(\theta + \Delta\theta_1)) \\ \vdots \\ J(\theta) - J(\theta + \Delta\theta_n)) \end{bmatrix}$$

Primal task - solution

- Gradient is solution of overdetermined set of linear equations:

$$\nabla J(\theta) = \begin{bmatrix} \Delta\theta_1^\top \\ \vdots \\ \Delta\theta_n^\top \end{bmatrix}^+ \cdot \begin{bmatrix} J(\theta) - J(\theta + \Delta\theta_1)) \\ \vdots \\ J(\theta) - J(\theta + \Delta\theta_n)) \end{bmatrix}$$

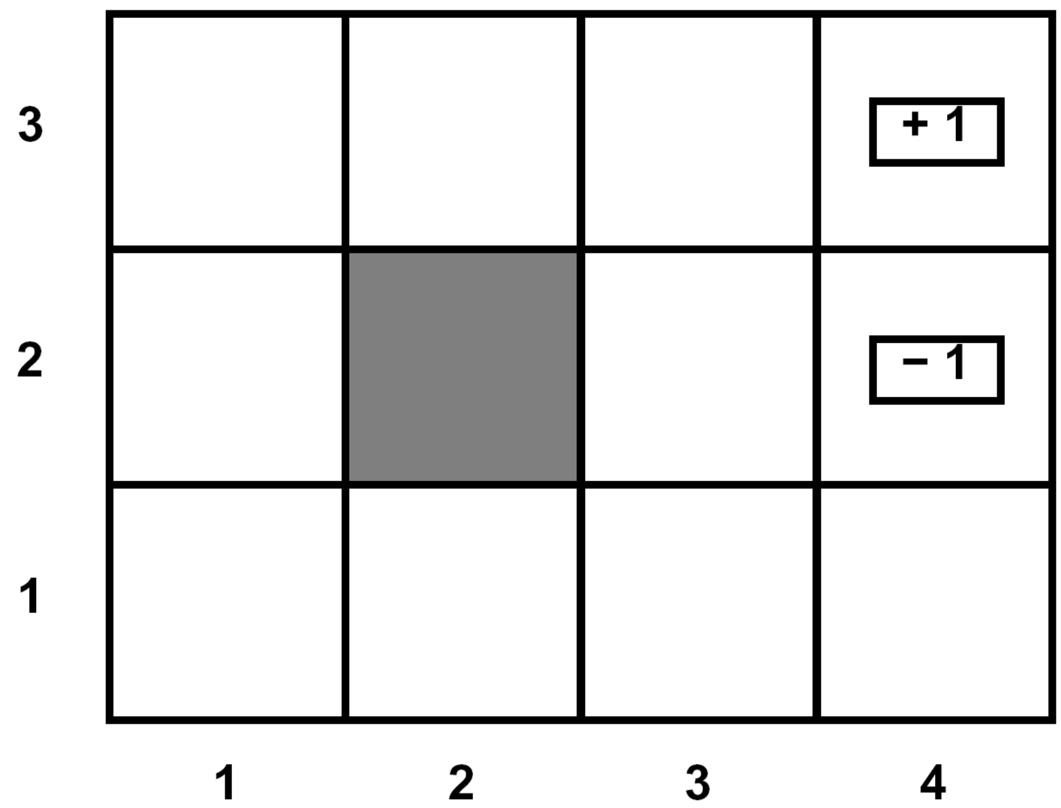
- Algorithm is simple:
  - Randomly initialize θ
  - Use $\pi_\theta(\mathbf{x})$ to get trajectories.
  - Compute $\nabla J(\theta)$ using pseudo-inverse.
  - Update $\theta \quad \leftarrow \quad \theta + \alpha \frac{\nabla J(\theta)}{\|\nabla J(\theta)\|}$

Dual task

- State-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$
- Expected sum of discounted rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.

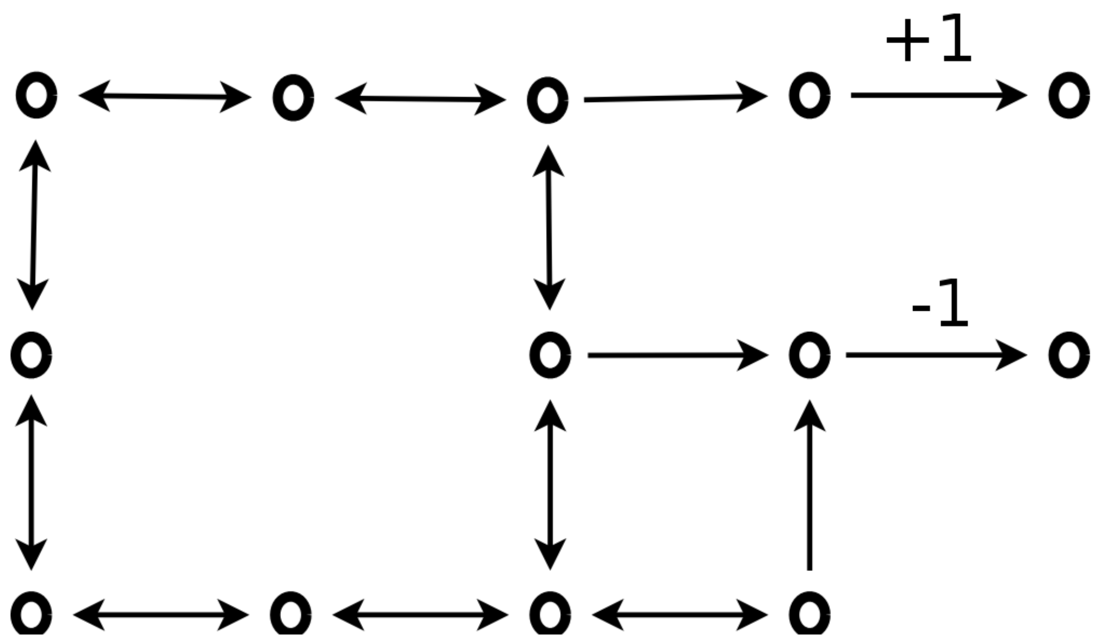
Dual task

- State-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$
- Expected sum of discounted rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
- Let us look at the grid world with stochastic transitions!



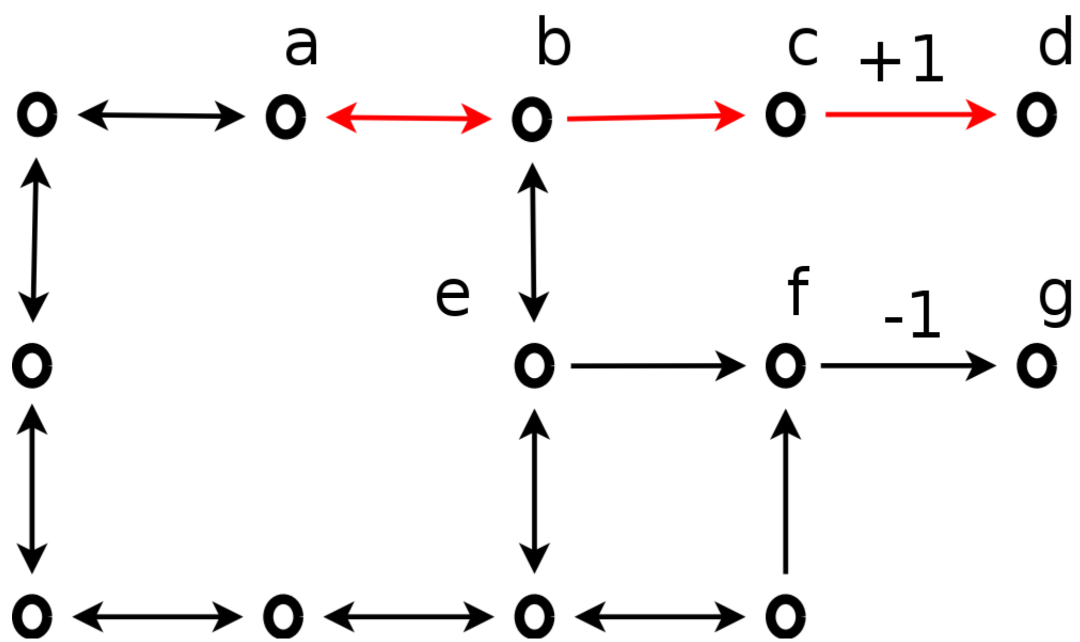
Dual task

- State-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$
- Mean sum of discounted rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
- How can we learn from recorded trajectories and corresponding rewards?



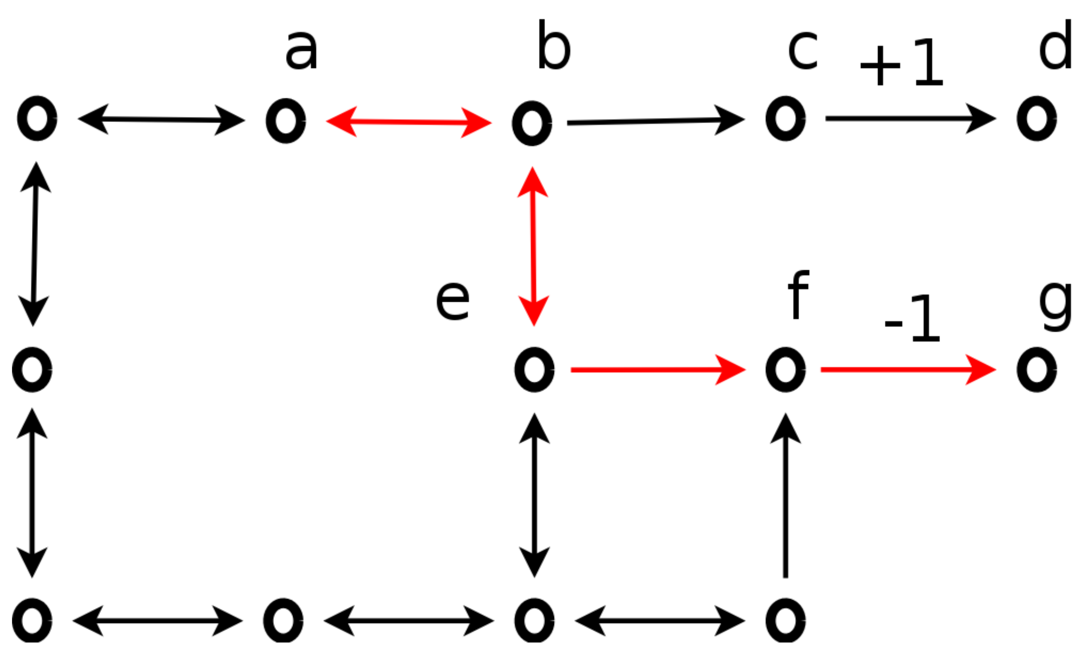
Dual task - naive learning example

- State-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$
- Mean sum of discounted rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
- How can we learn from recorded trajectories and corresponding rewards?
- $\tau_1 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{R}, \mathrm{c}, \mathrm{R}, \mathrm{d}), \qquad r(\tau_1) = 1$



Dual task - naive learning example

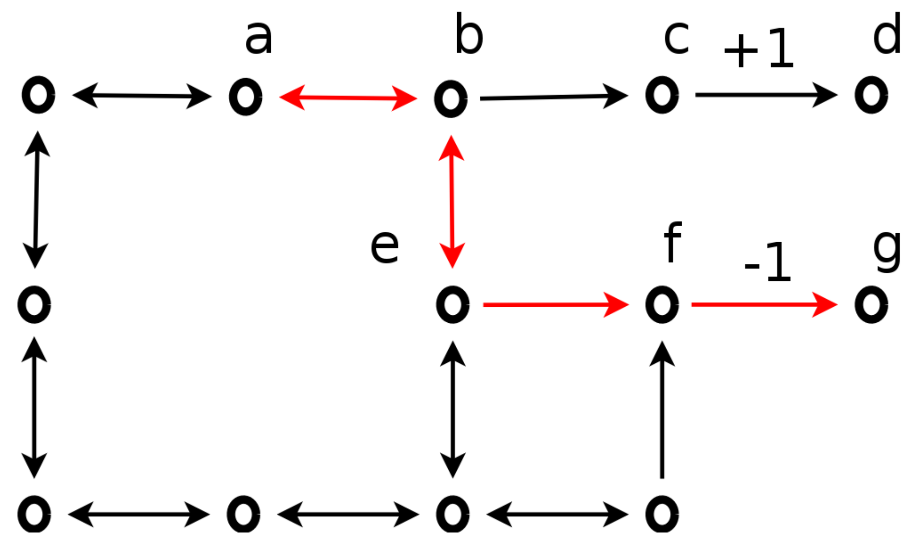
- State-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$
- Mean sum of discounted rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
- How can we learn from recorded trajectories and corresponding rewards?
- $\tau_1 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{R}, \mathrm{c}, \mathrm{R}, \mathrm{d}), \qquad r(\tau_1) = 1$
- $\tau_2 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{D}, \mathrm{e}, \mathrm{R}, \mathrm{f}, \mathrm{R}, \mathrm{g}), \quad r(\tau_2) = -1$



Dual task - naive learning example

- State-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$
- Mean sum of discounted rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
- How can we learn from recorded trajectories and corresponding rewards?
- $\tau_1 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{R}, \mathrm{c}, \mathrm{R}, \mathrm{d}), \qquad r(\tau_1) = 1$
- $\tau_2 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{D}, \mathrm{e}, \mathrm{R}, \mathrm{f}, \mathrm{R}, \mathrm{g}), \quad r(\tau_2) = -1$

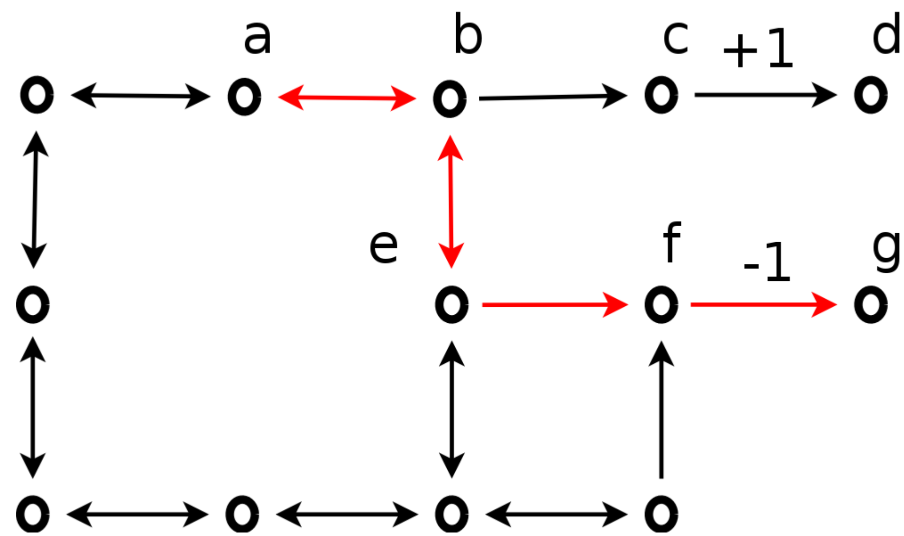
Q	R - right	D - down
a		
b		
c		
e	?	



Dual task - naive learning example

- State-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$
- Mean sum of discounted rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
- How can we learn from recorded trajectories and corresponding rewards?
- $\tau_1 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{R}, \mathrm{c}, \mathrm{R}, \mathrm{d}), \qquad r(\tau_1) = 1$
- $\tau_2 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{D}, \mathrm{e}, \mathrm{R}, \mathrm{f}, \mathrm{R}, \mathrm{g}), \quad r(\tau_2) = -1$

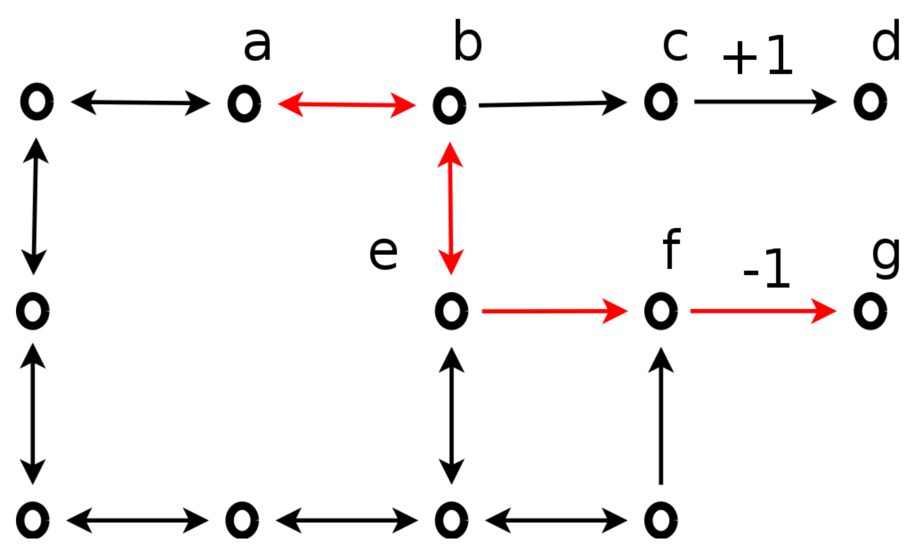
Q	R - right	D - down
a		
b		
c		
e	-1	



Dual task - naive learning example

- State-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$
- Mean sum of discounted rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
- How can we learn from recorded trajectories and corresponding rewards?
- $\tau_1 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{R}, \mathrm{c}, \mathrm{R}, \mathrm{d}), \qquad r(\tau_1) = 1$
- $\tau_2 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{D}, \mathrm{e}, \mathrm{R}, \mathrm{f}, \mathrm{R}, \mathrm{g}), \quad r(\tau_2) = -1$
- What is wrong? Why I learned nothing about policy for a?

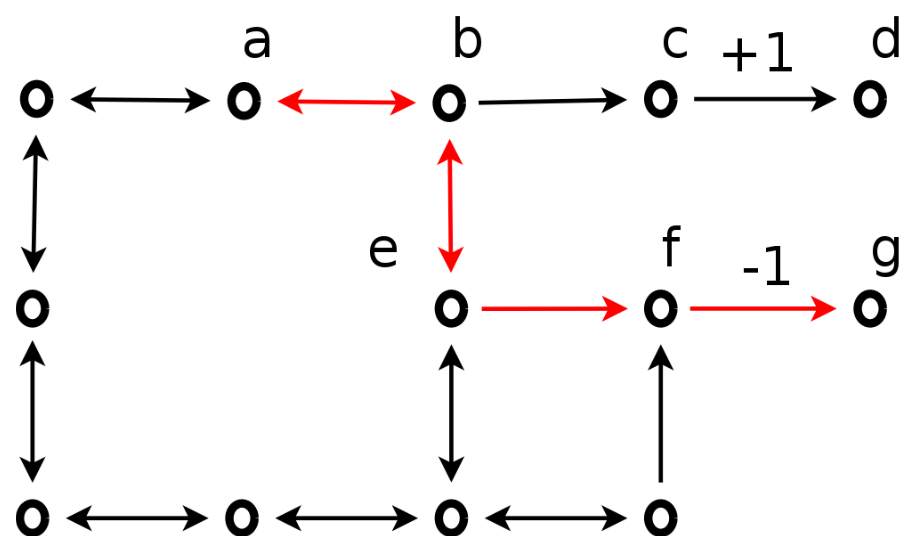
Q	R - right	D - down
a	**0**	
b	1	-1
c	1	
e	-1	



Dual task - naive learning example

- State-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$
- Mean sum of discounted rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
- How can we learn from recorded trajectories and corresponding rewards?
- $\tau_1 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{R}, \mathrm{c}, \mathrm{R}, \mathrm{d}), \qquad r(\tau_1) = 1$
- $\tau_2 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{D}, \mathrm{e}, \mathrm{R}, \mathrm{f}, \mathrm{R}, \mathrm{g}), \ \ r(\tau_2) = -1$
- I know that I can behave better from b, can I use it?

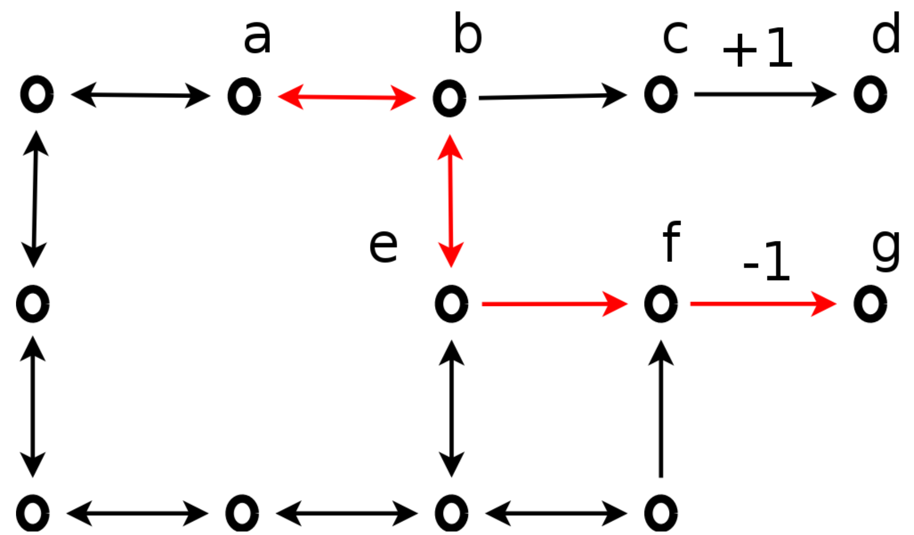
Q	R - right	D - down
a	**0**	
b	1	-1
c	1	
e	-1	



Dual task - naive learning example

- State-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \to \mathbb{R}$
- Mean sum of discounted rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
- How can we learn from recorded trajectories and corresponding rewards?
- $\tau_1 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{R}, \mathrm{c}, \mathrm{R}, \mathrm{d}), \qquad r(\tau_1) = 1$
- $\tau_2 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{D}, \mathrm{e}, \mathrm{R}, \mathrm{f}, \mathrm{R}, \mathrm{g}), \quad r(\tau_2) = -1$
- I know that I can behave better from b, can I use it?
- Recursively: $Q(\mathrm{a}, \mathrm{R}) = \mathrm{average}(\mathrm{reward\_for\_a} + \mathrm{best\_rewards\_from\_b})$

Q	R - right	D - down
a	**1**	
b	1	-1
c	1	
e	-1	



recursive definition of Q

- Define $Q(\mathbf{x}, \mathbf{u})$ recursively:
  - If model is unknown

$$Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$$

recursive definition of Q

- Define $Q(\mathbf{x}, \mathbf{u})$ recursively:
  - If model is unknown

$$Q(\mathbf{x}, \mathbf{u}) = r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}')$$

  - If a stochastic model is known

$$Q(\mathbf{x}, \mathbf{u}) = \sum_{\mathbf{x}'} p(\mathbf{x}'|\mathbf{u}, \mathbf{x}) \Big[r(\mathbf{x}, \mathbf{u}, \mathbf{x}') + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}', \mathbf{u}') \Big]$$

(Bellman equation)

Q-learning

- Initialize $Q(\mathbf{x}, \mathbf{u}) = 0 \quad \forall_{\mathbf{x},\mathbf{u}}$

Q-learning

- Initialize $Q(\mathbf{x}, \mathbf{u}) = 0 \quad \forall_{\mathbf{x},\mathbf{u}}$
- Drive the robot and record trajectories like that:

$$(\mathbf{x}_0, \mathbf{u}_0, \mathbf{x'}_0, r_0), \quad (\mathbf{x}_1 = \mathbf{x'}_0, \mathbf{u}_1, \mathbf{x'}_1, r_1), \quad \ldots$$

Q-learning

- Initialize $Q(\mathbf{x}, \mathbf{u}) = 0 \quad \forall_{\mathbf{x},\mathbf{u}}$
- Drive the robot and record trajectories like that:

  $(\mathbf{x}_0, \mathbf{u}_0, \mathbf{x'}_0, r_0), \quad (\mathbf{x}_1 = \mathbf{x'}_0, \mathbf{u}_1, \mathbf{x'}_1, r_1), \quad \ldots$
- For $\mathbf{x} \in X, \ \mathbf{u} \in U$

$$Q(\mathbf{x}, \mathbf{u}) = \frac{1}{n} \sum_{i \in \{\mathbf{x}_i = \mathbf{x}, \mathbf{u}_i = \mathbf{u}\}} r_i + \gamma \max_{\mathbf{u}'} Q(\mathbf{x'}_i, \mathbf{u}')$$

- End

Q-learning

- Initialize $Q(\mathbf{x},\mathbf{u}) = 0 \quad \forall_{\mathbf{x},\mathbf{u}}$
- Drive the robot and record sequences:

$(\mathbf{x}_0, \mathbf{u}_0, \mathbf{x'}_0, r_0), \quad (\mathbf{x}_1 = \mathbf{x'}_0, \mathbf{u}_1, \mathbf{x'}_1, r_1), \quad \dots$

——— Iterate until convergence ———

- For $\mathbf{x} \in X, \;\; \mathbf{u} \in U$

$$Q(\mathbf{x},\mathbf{u}) = \frac{1}{n} \sum_{i \in \{\mathbf{x}_i = \mathbf{x}, \mathbf{u}_i = \mathbf{u}\}} r_i + \gamma \max_{\mathbf{u'}} Q(\mathbf{x'}_i, \mathbf{u'})$$

- End

———(fixed point algorithm for system of lin. eq.)———

Q-learning

- Initialize $Q(\mathbf{x},\mathbf{u}) = 0 \quad \forall_{\mathbf{x},\mathbf{u}}$

Iterate until good policy found

- Drive the robot and record sequences:

$$(\mathbf{x}_0, \mathbf{u}_0, \mathbf{x'}_0, r_0), \quad (\mathbf{x}_1 = \mathbf{x'}_0, \mathbf{u}_1, \mathbf{x'}_1, r_1), \quad \ldots$$

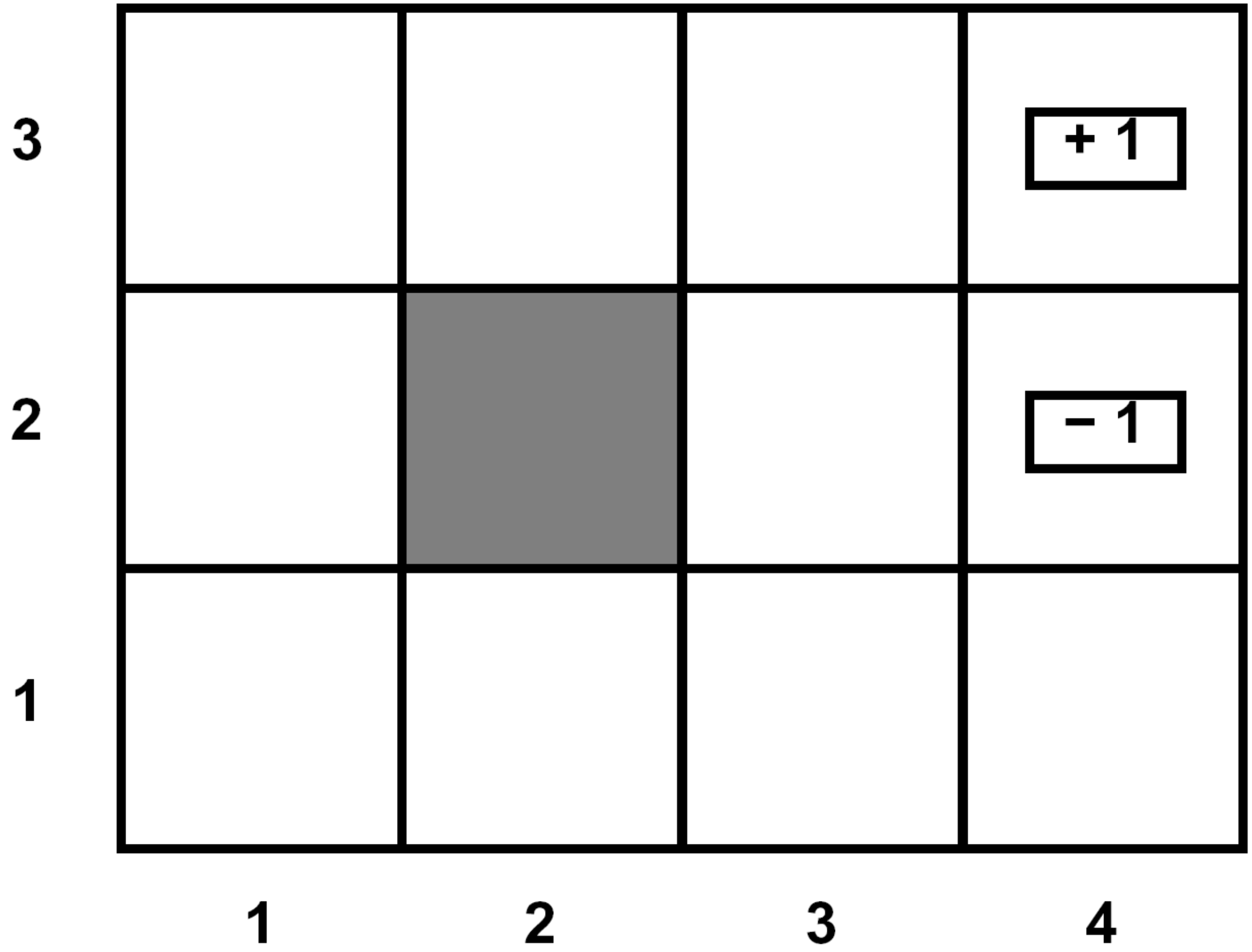
- For $\mathbf{x} \in X,\ \mathbf{u} \in U$

$$Q(\mathbf{x},\mathbf{u}) = \frac{1}{n} \sum_{i \in \{\mathbf{x}_i = \mathbf{x}, \mathbf{u}_i = \mathbf{u}\}} r_i + \gamma \max_{\mathbf{u}'} Q(\mathbf{x'}_i, \mathbf{u}')$$

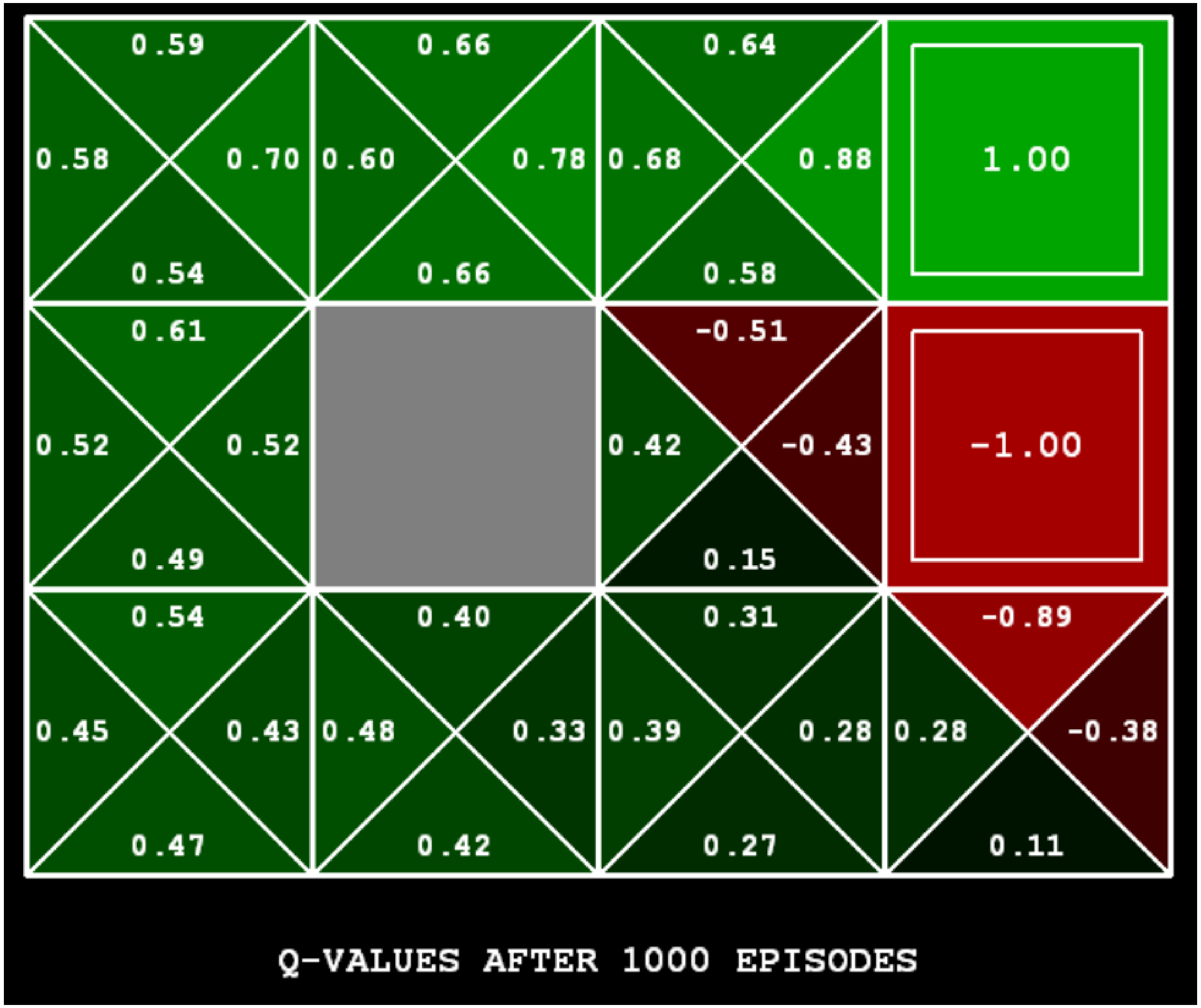
- End

State-value function example I - grid-world

- Q-learning for stochastic grid-world.



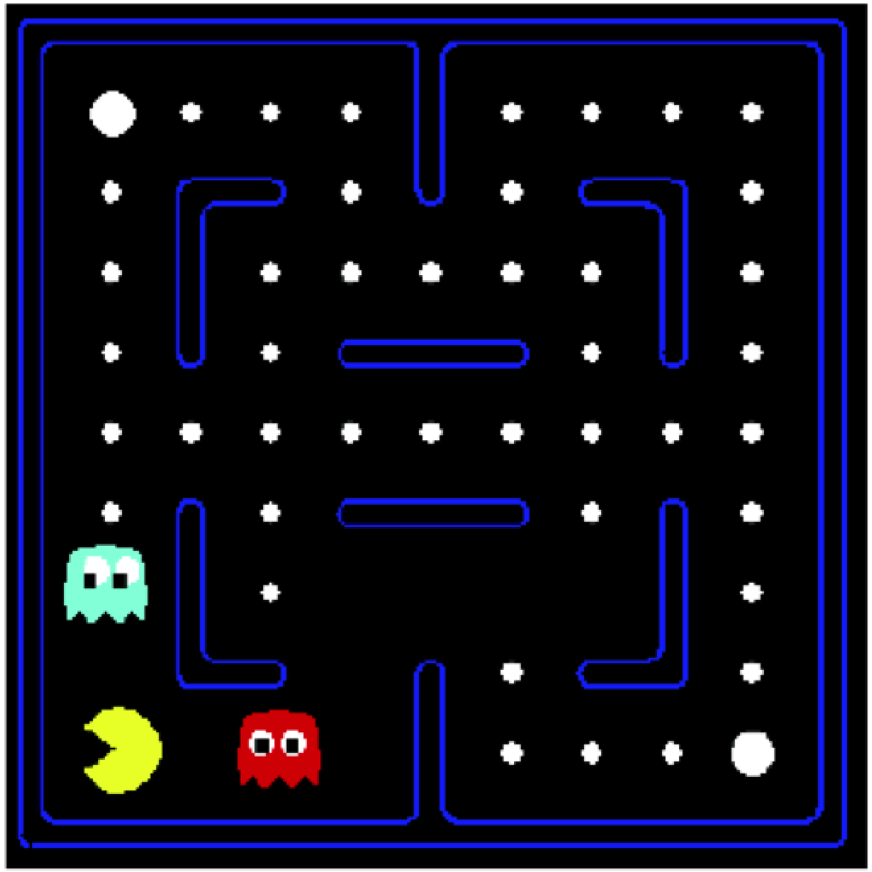
State-value function example I - grid-world



Where is the catch?

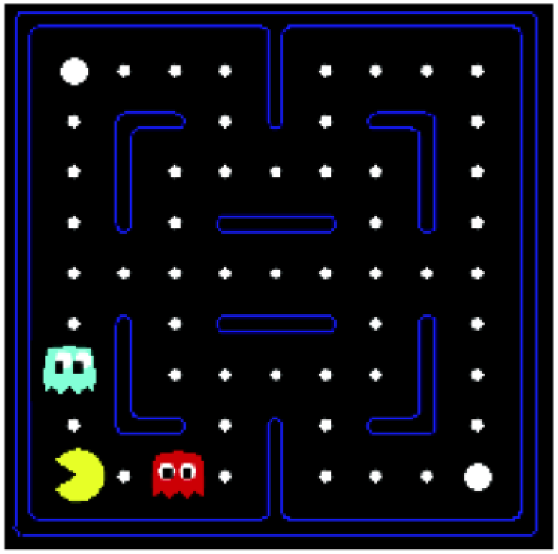
Where is the catch?

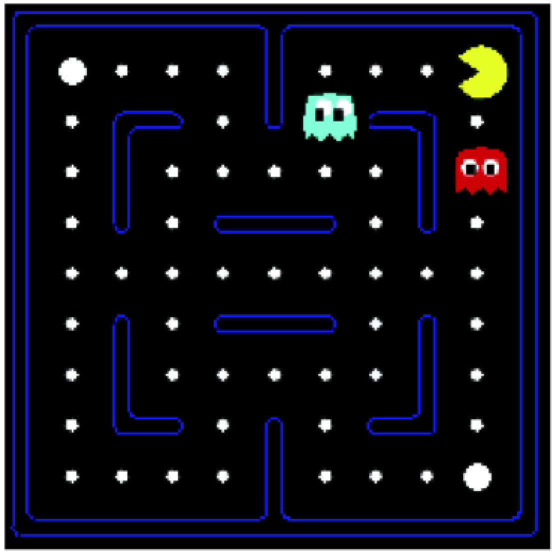
- Curse of dimensionality - considered state space for pacman.

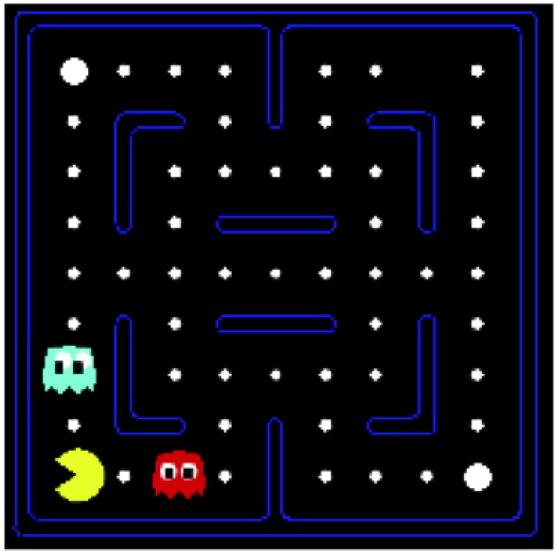


Where is the catch?

- Curse of dimensionality - are these states the same? Do we want it?



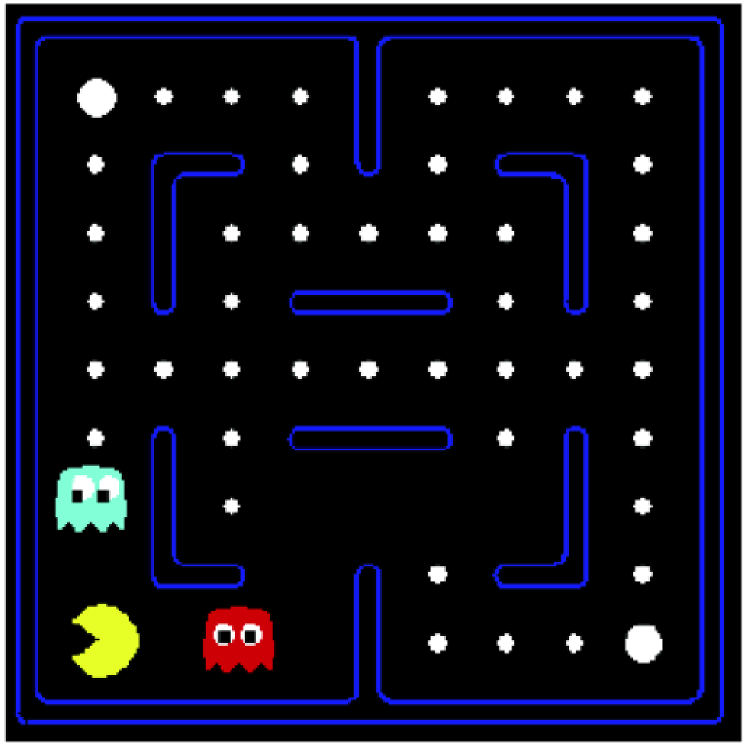




Where is the catch?

- Curse of dimensionality - we need to replace high-dimensional states $\mathbf{x}$ and control $\mathbf{u}$ by low-dimensional features $\Phi(\mathbf{x}, \mathbf{u})$.

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist to dot)^2
    - Is Pacman in a tunnel? (0/1)
    - ...... etc.
    - Is it the exact state on this slide?
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Where is the catch?

- Curse of dimensionality - Q-learning

Iterate until convergence

- For $\mathbf{x} \in X, \; \mathbf{u} \in U$

$$Q(\mathbf{x},\mathbf{u}) = \frac{1}{n} \sum_{i \in \{\mathbf{x}_i = \mathbf{x}, \mathbf{u}_i = \mathbf{u}\}} r_i + \gamma \max_{\mathbf{u}'} Q(\mathbf{x}'_i, \mathbf{u}')$$

- End

Where is the catch?

- Curse of dimensionality - approximate Q-learning

Iterate until convergence

- For all $\mathbf{x}_i,\ \mathbf{u}_i$

$$y_i = r_i + \gamma \max_{\mathbf{u}'} [\theta^\top \Phi(\mathbf{x}'_i, \mathbf{u}'))]$$

- End
- Fit Q-function to approximate mapping between $\Phi(\mathbf{x}_i, \mathbf{u}_i)$ and y_i

$$\theta \leftarrow \arg\min_{\theta} \|\theta^\top \Phi(\mathbf{x}_i, \mathbf{u}_i) - y_i\|$$

Where is the catch?

- Curse of dimensionality

Where is the catch?

- Curse of dimensionality
- Reward tuning

Where is the catch?

- Curse of dimensionality
- Reward tuning
- Exploration vs exploitation
  - ϵ-greedy exploration
  - or exploration extension $Q(\Phi(\mathbf{x},\mathbf{u})) + \frac{k}{N(\Phi)}$

Where is the catch?

- Curse of dimensionality
- Reward tuning
- Exploration vs exploitation
  - ϵ-greedy exploration
  - or exploration extension $Q(\Phi(\mathbf{x}, \mathbf{u})) + \frac{k}{N(\Phi)}$
- Safe exploration, cooperative tasks, hierarchical reinforcment learning.

Conclusions

- Primal Dual task
  - convergence issues
  - do we need to know sum of rewards?
- Do not forget features!
- What you can do?

What you can do?

- Work with us on:
  - real Search&Rescue platform
  - better IRO tasks
- TORCS - Racing **and demolishon derby** simulator competition.
  http://en.wikipedia.org/wiki/TORCS
- Starcraft competition
  http://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/