Introduction to Reinforcment Learning

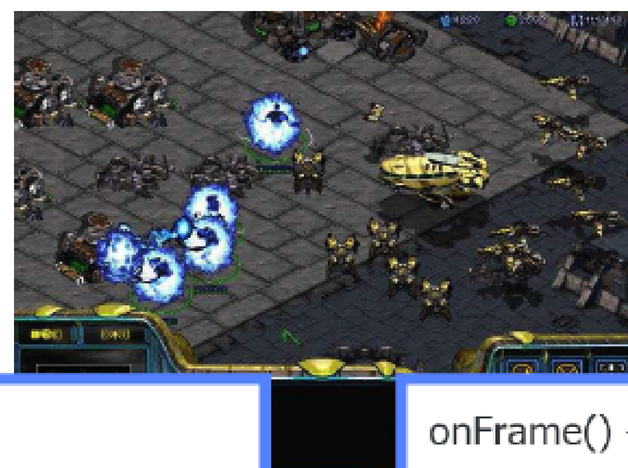
Karel Zimmermann

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`

Some images and codes taken from P.Abbeel, J.Peters, M.Riedmiller, T.Jakab

Motivation examples

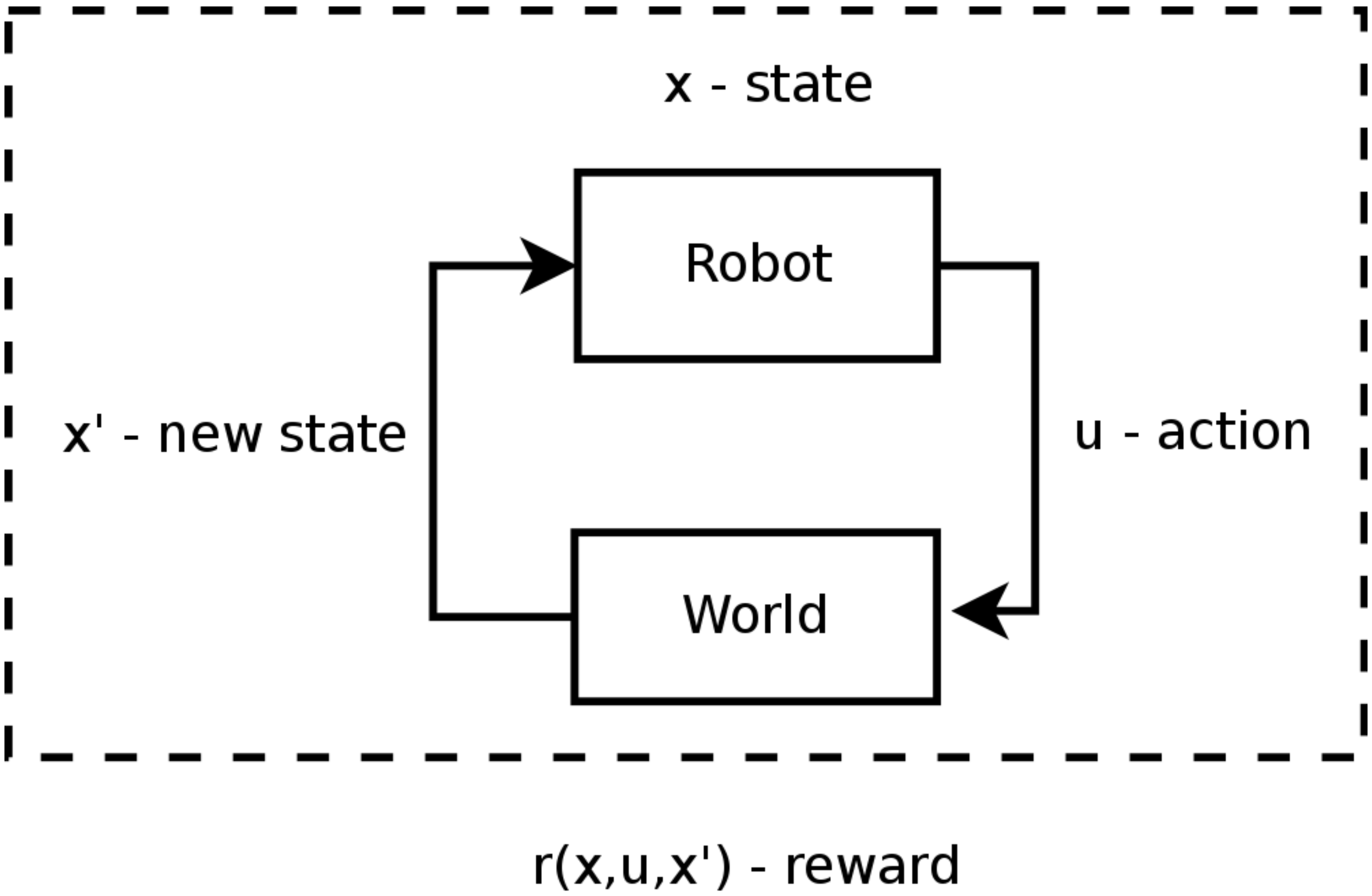
- Learning to control a dynamic process from real world interactions.
- Human teacher is not needed - rewards assigned by environment.



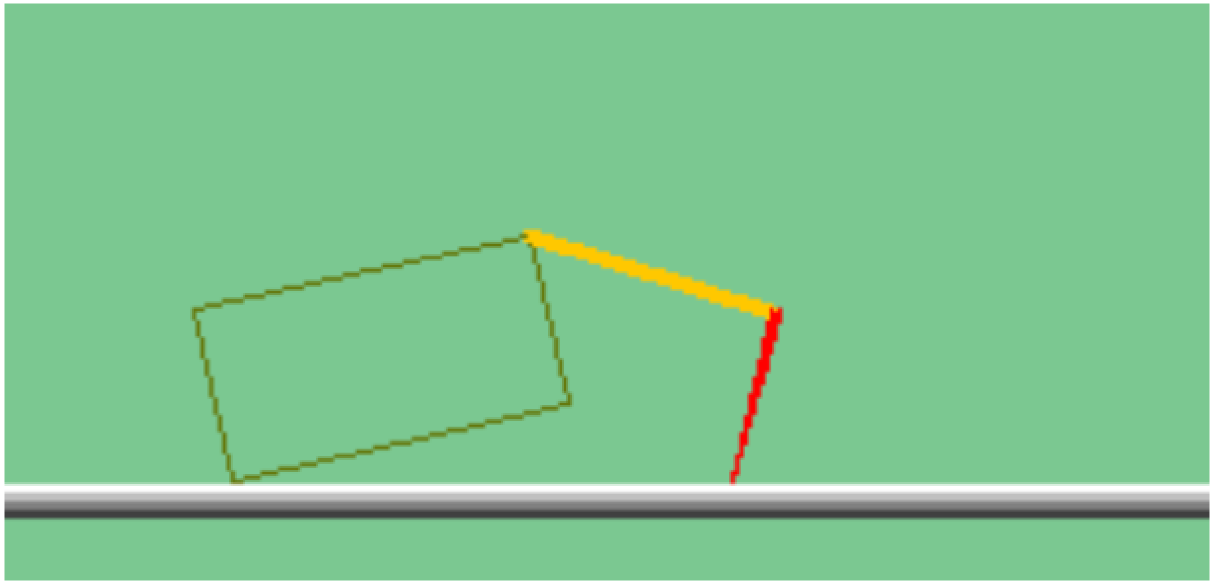
```
onFrame() {
    units = Broodwar->getAllUnits();
    unit->attackUnit(enemyUnit);
}
```

```
onFrame() {
    units = Broodwar->getAllUnits();
    unit->attackUnit(enemyUnit);
}
```

Simplest possible scheme



What are the states, actions and rewards?



- Crawler - show python demo!
- Bouncing ball - show video!
- Ball in the cup - show video!
- Pacman - show python (`01_pacman_states`)

What do we search for?

- Optimal policy (strategy, control) which assigns actions u_i to states x_i.

What do we search for?

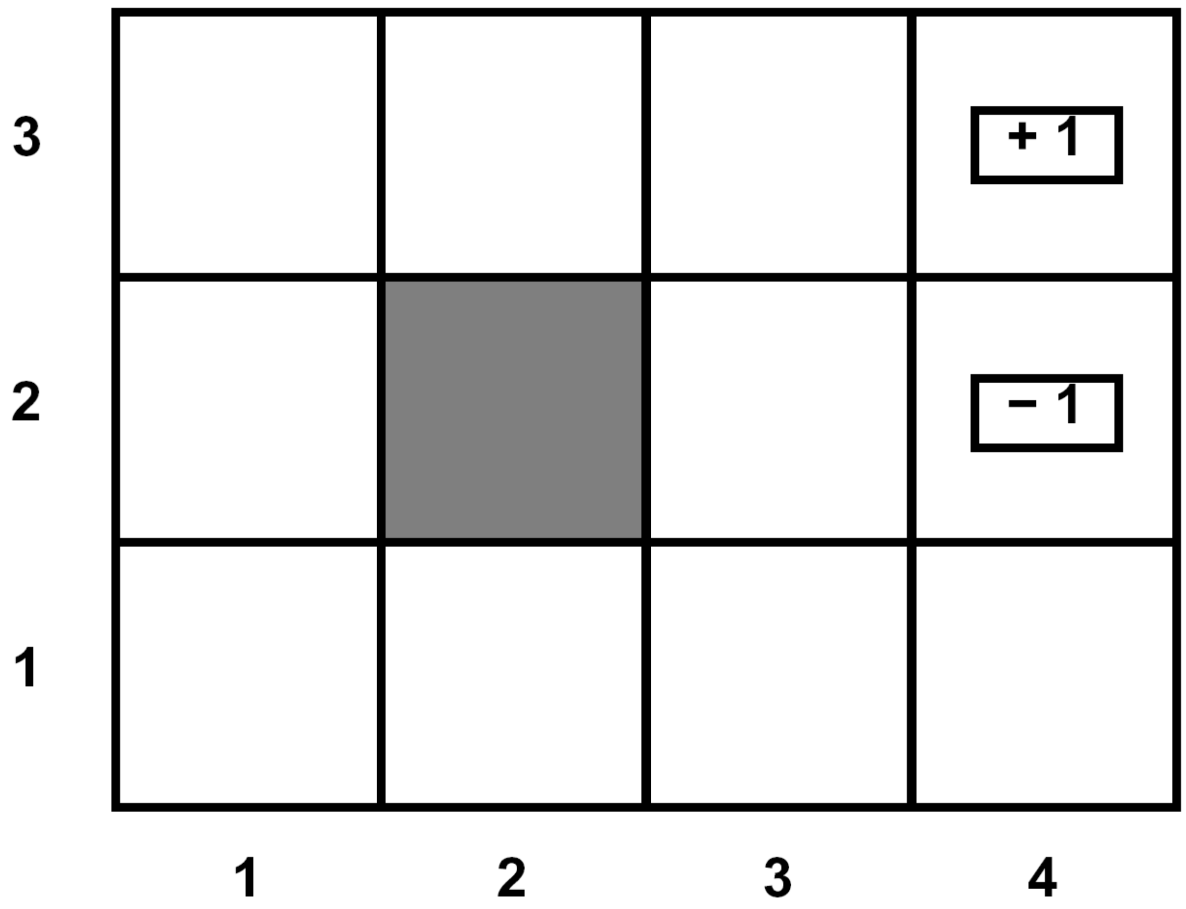
- Optimal policy (strategy, control) which assigns actions u_i to states x_i.
- Optimal $=$ assuring long-term high rewards $\sum_{i=1}^{\infty} r_i$

How can we find the optimal policy?

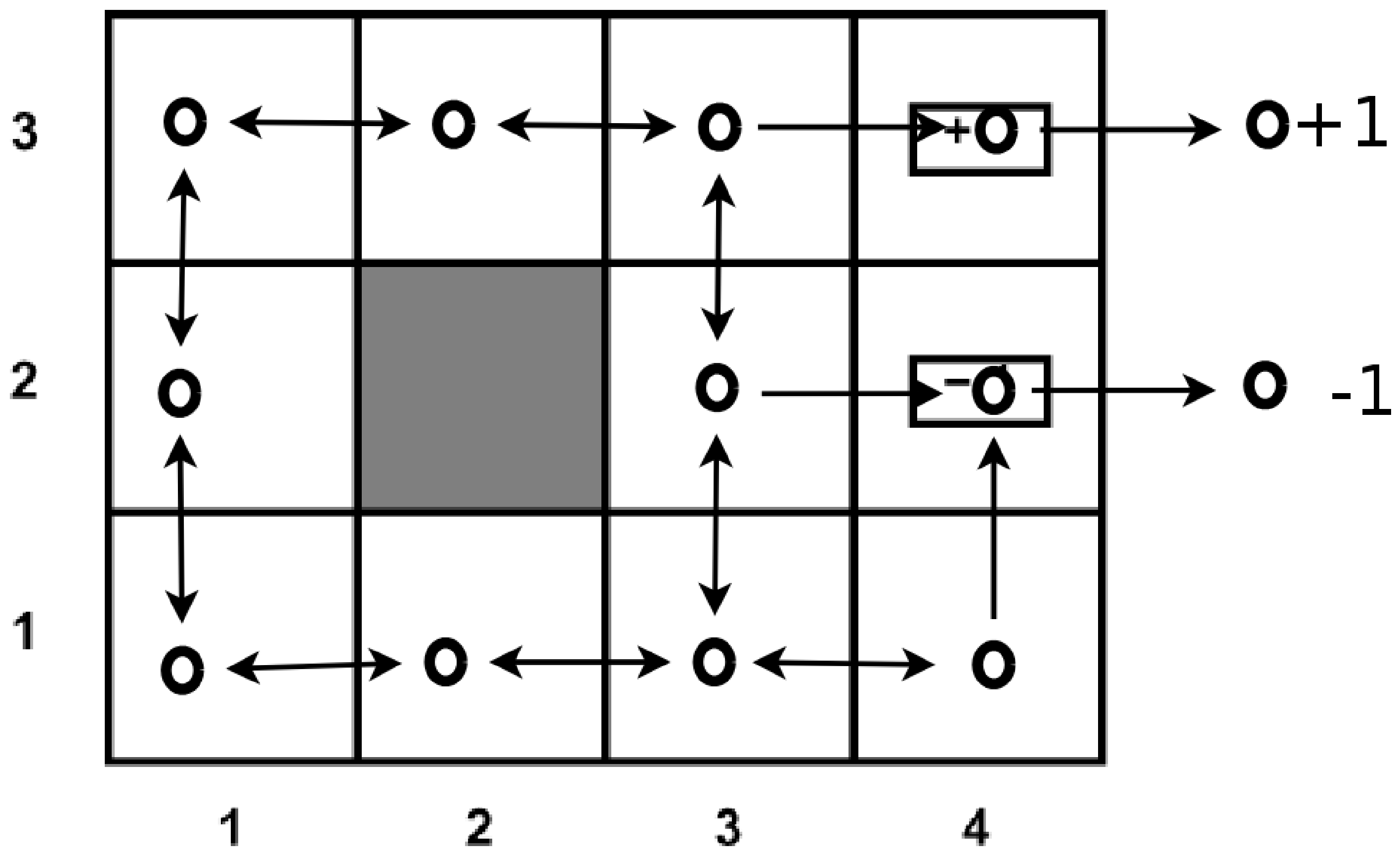
- Depends on the world.

How can we find the optimal policy?

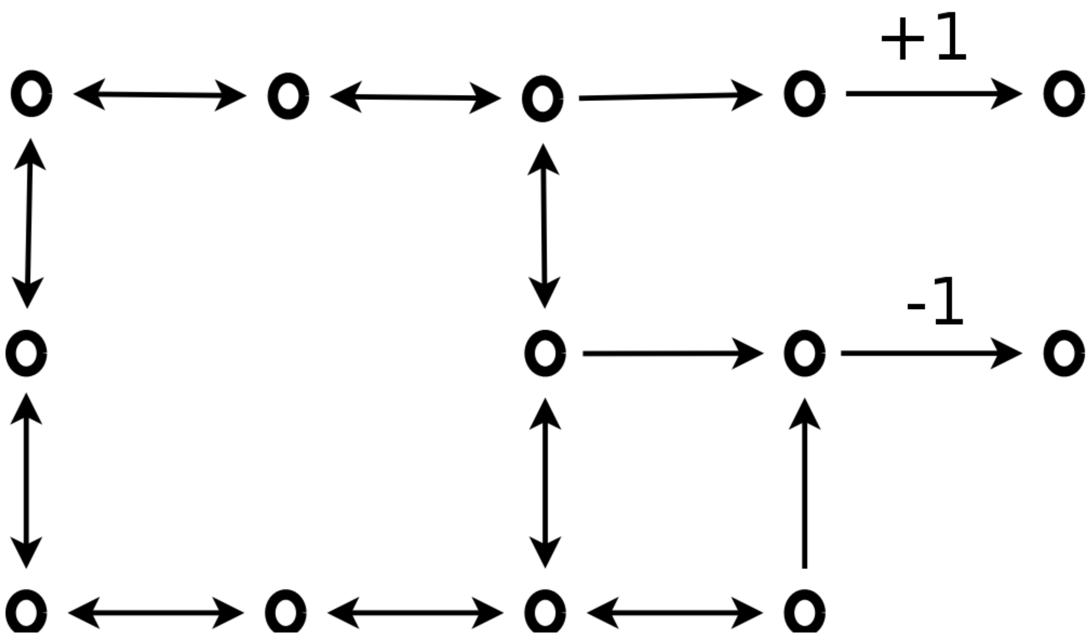
- Depends on the world.
- What about this grid-world?



How can we find the optimal policy?



How can we find the optimal policy?



How can we find the optimal policy?

- Dijkstra yields the optimal policy in some type worlds - usually:
  - deterministic,
  - tiny,
  - static,
  - known in advance

What about more realistic worlds?

- What if transitions are stochastic?

What about more realistic worlds?

- What if transitions are stochastic?

  - expectimax or minmax tree search

- What if the world is dynamically changing (e.g. there is someone else like doc.Ing.Zlo,CSc.)?

What about more realistic worlds?

- What if transitions are stochastic?

  - expectimax or minmax tree search

- What if the world is dynamically changing (e.g. there is someone else like doc.Ing.Zlo,CSc.)?

  - lift the state-space up to higher dimension

What about more realistic worlds?

- What if transitions are stochastic?

  - expectimax or minmax tree search

- What if the world is dynamically changing (e.g. there is someone else like doc.Ing.Zlo,CSc.)?

  - lift the state-space up to higher dimension

- What if the world is unknown in advance?
- What if the robot-world interactions are not explicitly modelable?
- What if the world is huge (continuous and infinite)?

What about more realistic worlds?

- What if transitions are stochastic?

  - expectimax or minmax tree search

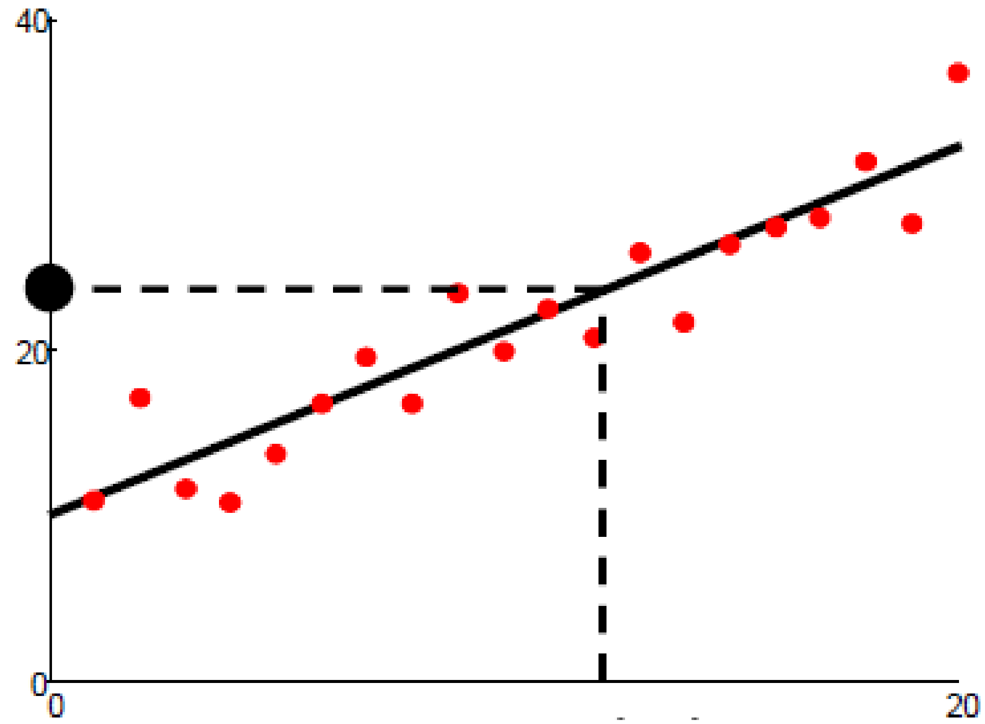
- What if the world is dynamically changing (e.g. there is someone else like doc.Ing.Zlo,CSc.)?

  - lift the state-space up to higher dimension

- What if the world is unknown in advance?
- What if the robot-world interactions are not explicitly modelable?
- What if the world is huge (continuous and infinite)?
- Under such conditions, bare state-space search is not technically possible.
- We would like to learn the optimal policy from real-world examples.

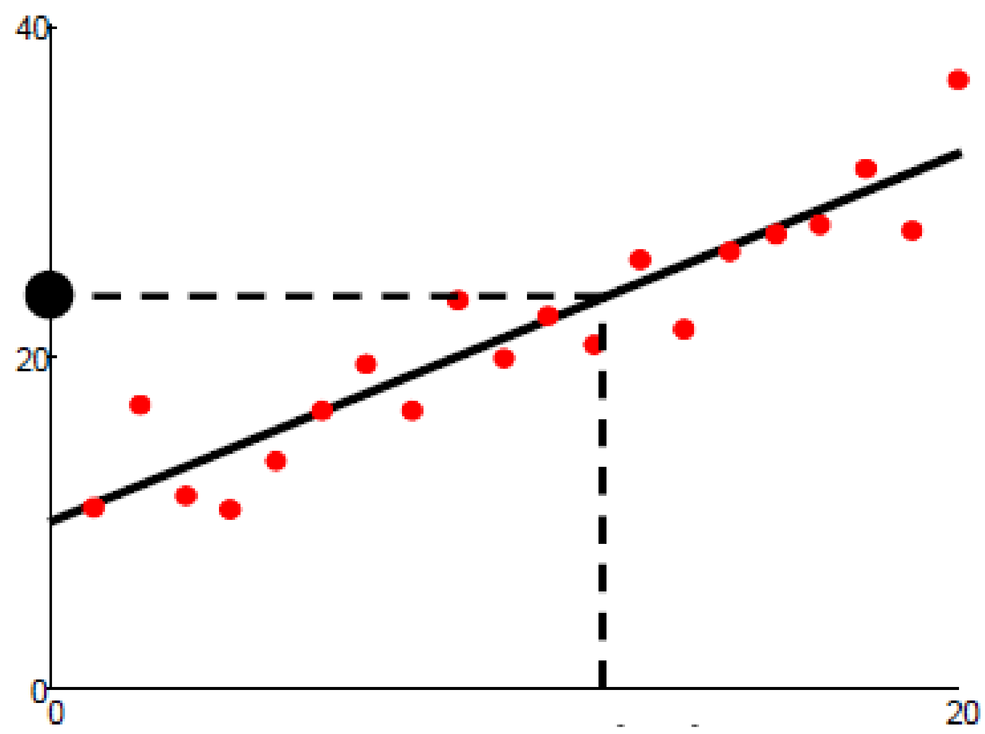
Can machine learning help?

- Why can't we learn $\pi : X \to U$ mapping?



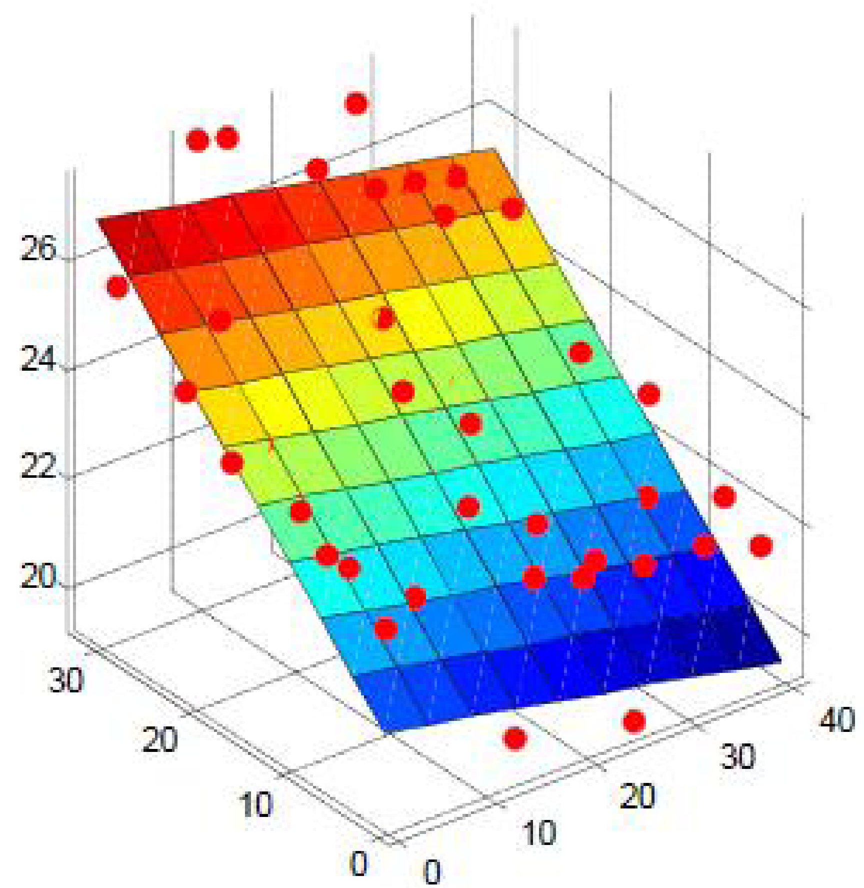
Can machine learning help?

- Why can't we learn mapping $\pi : X \to U$ (policy)?
- Because we do not know the right state-action pairs to train from.
- Nevertheless, there is a way to learn mapping π directly. PRIMAL TASK.



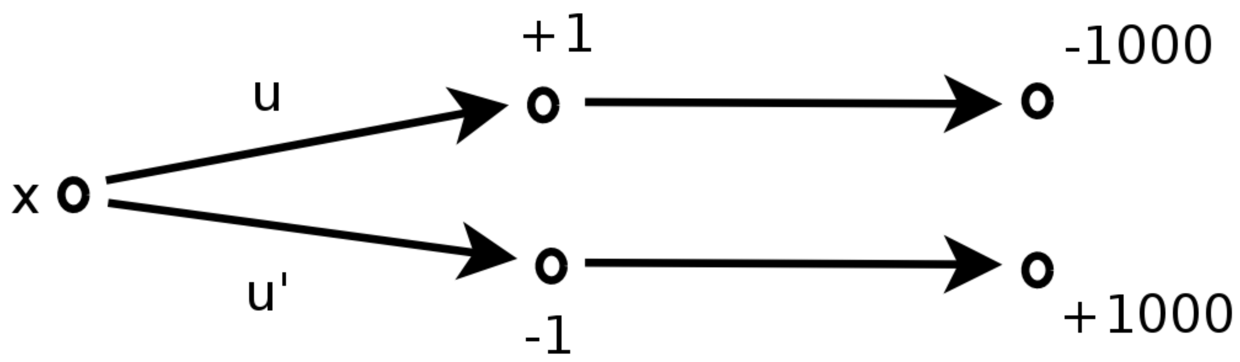
Can machine learning help?

- But we know rewards r coresponding to (x, u) touples.
- What about to learn mapping $Q : X \times U \to \mathbb{R}$ and take action $u^* = \pi(x) = \operatorname{argmax}_u Q(x, u)$?



Can machine learning help?

- But we know rewards r coresponding to (x, u) touples.
- What about to learn mapping $Q : X \times U \to \mathbb{R}$ and take action $u^* = \pi(x) = \operatorname{argmax}_a Q(x, u)$?
- Tooooooo greedy !!!



Can machine learning help?

- But we know rewards r coresponding to (x, u) touples.
- What about to learn mapping $Q : X \times U \to \mathbb{R}$ and take action $u^* = \pi(x) = \operatorname{argmax}_a Q(x, u)$?
- Toooooo greedy !!!
- Nevertheless, also not that bad idea, there is a way to learn mapping Q assigning $\sum_i r_i$ instead of r_1. DUAL TASK.

Lesson outline

- This lecture is about learning the optimal mapping $\pi : X \to U$ from real-world interactions.

Lesson outline

- This lecture is about learning the optimal mapping $\pi : X \to U$ from real-world interactions.
- Focus is on methods without model !!!

Lesson outline

- This lecture is about learning the optimal mapping $\pi : X \rightarrow U$ from real-world interactions.
- Focus is on methods without model !!!
- MDP notation (state, reward, transition probability, etc.).

Lesson outline

- This lecture is about learning the optimal mapping $\pi : X \to U$ from real-world interactions.
- Focus is on methods without model !!!
- MDP notation (state, reward, transition probability, etc.).
- Problem formulation in terms of criterion to be optimized.

Lesson outline

- This lecture is about learning the optimal mapping $\pi : X \to U$ from real-world interactions.
- Focus is on methods without model !!!
- MDP notation (state, reward, transition probability, etc.).
- Problem formulation in terms of criterion to be optimized.
- Primal task (as far as we can get towards natural gradients).

Lesson outline

- This lecture is about learning the optimal mapping $\pi : X \to U$ from real-world interactions.
- Focus is on methods without model !!!
- MDP notation (state, reward, transition probability, etc.).
- Problem formulation in terms of criterion to be optimized.
- Primal task (as far as we can get towards natural gradients).
- Dual task (without proving that it is the dual task)

Lesson outline

- This lecture is about learning the optimal mapping $\pi : X \to U$ from real-world interactions.
- Focus is on methods without model !!!
- MDP notation (state, reward, transition probability, etc.).
- Problem formulation in terms of criterion to be optimized.
- Primal task (as far as we can get towards natural gradients).
- Dual task (without proving that it is the dual task)
- Curse of dimensionality (states represented by features).

Lesson outline

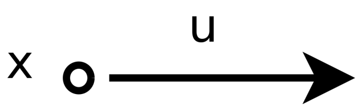
- This lecture is about learning the optimal mapping $\pi : X \to U$ from real-world interactions.
- Focus is on methods without model !!!
- MDP notation (state, reward, transition probability, etc.).
- Problem formulation in terms of criterion to be optimized.
- Primal task (as far as we can get towards natural gradients).
- Dual task (without proving that it is the dual task)
- Curse of dimensionality (states represented by features).
- Other related problems (imitation learning, exploration)

MDP definition

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

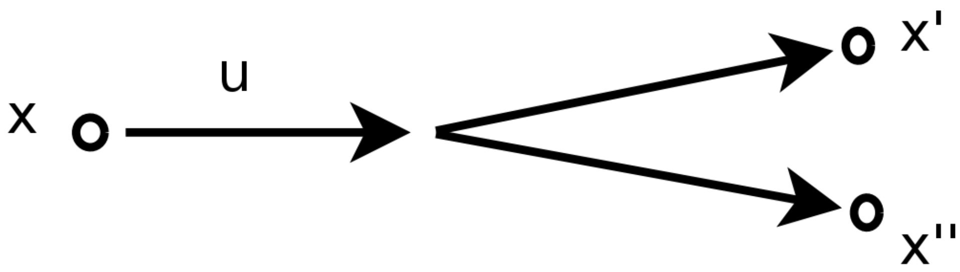
MDP definition

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



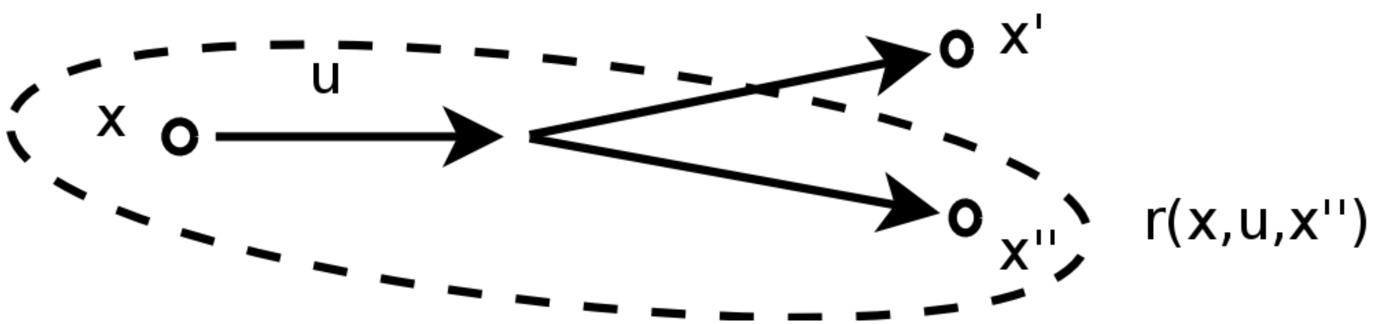
MDP definition

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



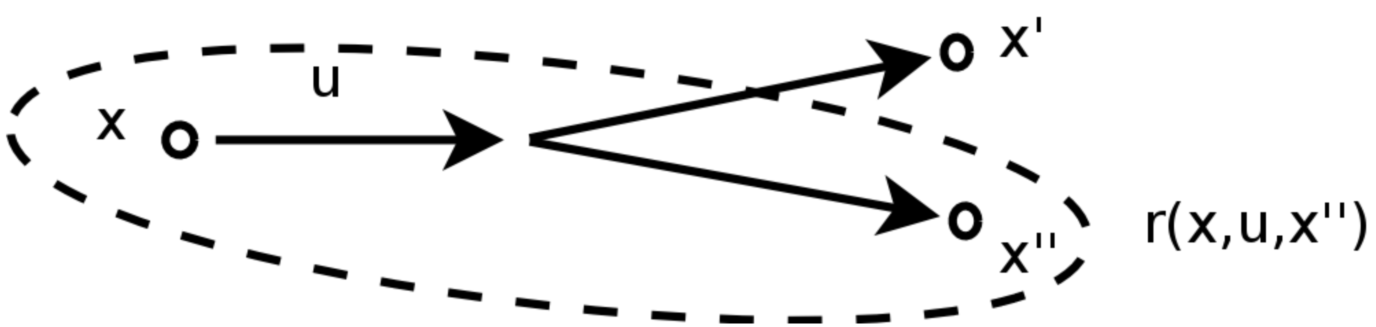
MDP definition

- 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: $\pi(\mathbf{x}) : X \to U$ (at least for now, but better to use probability)



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:

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

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 mean sum of rewards

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

Problem definition

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

Problem definition

- Denote $p(\tau|\pi)$ probability of trajectory τ occurs when following policy π
- Criterion to me maximized is the mean sum of rewards

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

Problem definition

- Denote $p(\tau|\pi)$ probability of trajectory τ occurs when following policy π
- Criterion to me maximized is the mean sum of rewards

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

- We solve the following optimization problem

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

Problem solution

- As usually, you can:

Problem solution

- As usually, you can:
  - either solve **primal task** e.g. by following gradient ∇J to maximize $J(\pi)$ 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 $J(\pi)$ directly.
  - primal is often solved in the optimal control community (e.g. LQR),
  - or solve **dual task** by searching for dual variable Q via lagrange multipliers and follow policy $\pi^* = \arg\max_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u})$
  - dual is often solved by AI community (e.g. state-space search for games)

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

Primal task

- How do we solve the following optimization problem

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

Primal task

- How do we solve the following optimization problem

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

- Let us choose policy $\pi(\theta) = \theta^\top \mathbf{x}$ parameterized by coeffitients θ.

Primal task

- How do we solve the following optimization problem

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

- Let us choose policy $\pi(\theta) = \theta^\top \mathbf{x}$ parameterized by coeffitients θ.
- then optimization problem reduces to

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

- How can we compute $J(\theta) = E\{r(\tau)\}$ fro a given θ?

Primal task - approximating criterion.

- Use $\pi(\theta)$ to get several trajectories τ_i.

Primal task - approximating criterion.

- Use $\pi(\theta)$ to get several trajectories τ_i.
- Approximate criterion value in θ as average reward

$$J(\theta) = E\{r(\tau)\} \approx \frac{1}{N}\sum_{i=1}^{N} r(\tau_i)$$

Primal task - approximating criterion.

- Use $\pi(\theta)$ to get several trajectories τ_i.
- Approximate criterion value in θ as average reward

$$J(\theta) = E\{r(\tau)\} \approx \frac{1}{N}\sum_{i=1}^{N} r(\tau_i)$$

- 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 initilize θ
  - Use $\pi(\theta)$ to get trajectories.
  - Compute $\nabla J(\theta)$ using pseudo-inverse.
  - Update $\theta \quad \leftarrow \quad \theta + \alpha \frac{\nabla J(\theta)}{\|\nabla J(\theta)\|}$

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 initilize θ
  - Use $\pi(\theta)$ to get trajectories.
  - Compute $\nabla J(\theta)$ using pseudo-inverse.
  - Update $\theta \quad \leftarrow \quad \theta + \alpha \frac{\nabla J(\theta)}{\|\nabla J(\theta)\|}$
- Show example in MATLAB - `go_toy_finite_difference.m`.

Primal task - pros and cons

- No model identification needed.

Primal task - pros and cons

- No model identification needed.
- Converges to local minima - good initialization needed.

Primal task - pros and cons

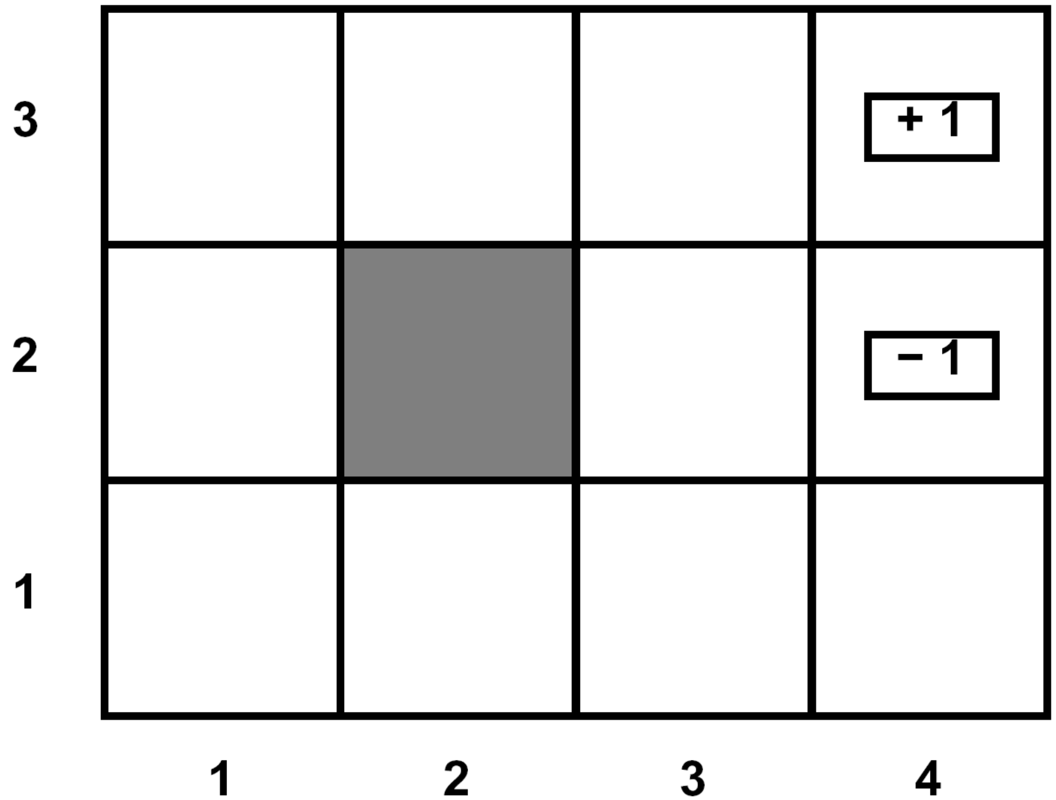
- No model identification needed.
- Converges to local minima - good initialization needed.
- There are better gradient approximations - natural gradient methods [Kober-IJRR-2013].

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 $\mathbf{x}$.

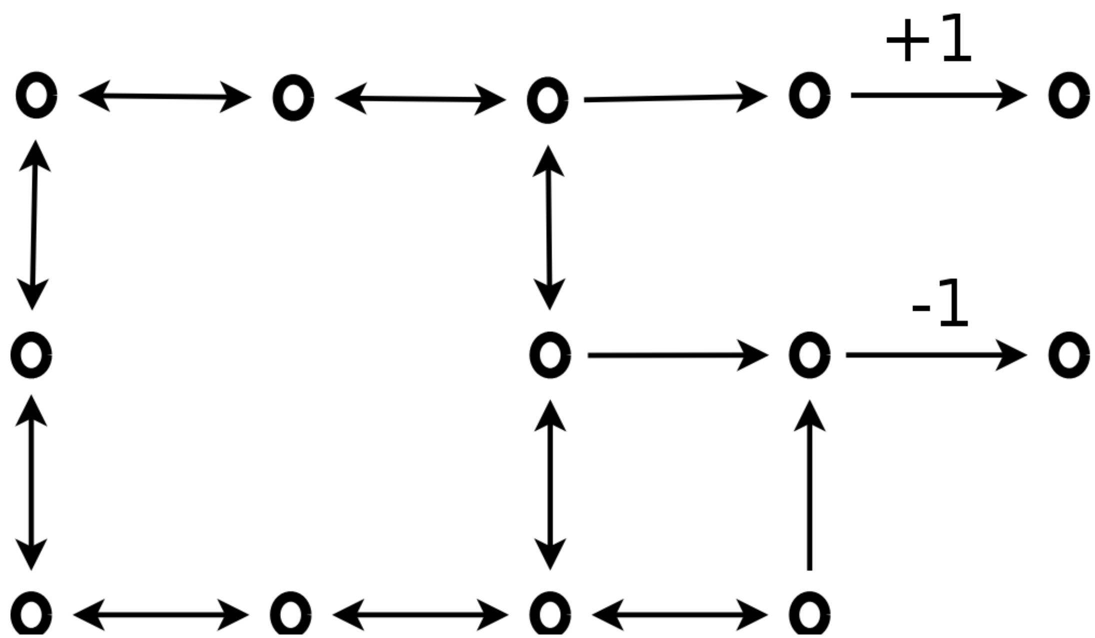
Dual task

- 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 $\mathbf{x}$.
- Let us look at the grid world with stochastic transitions!



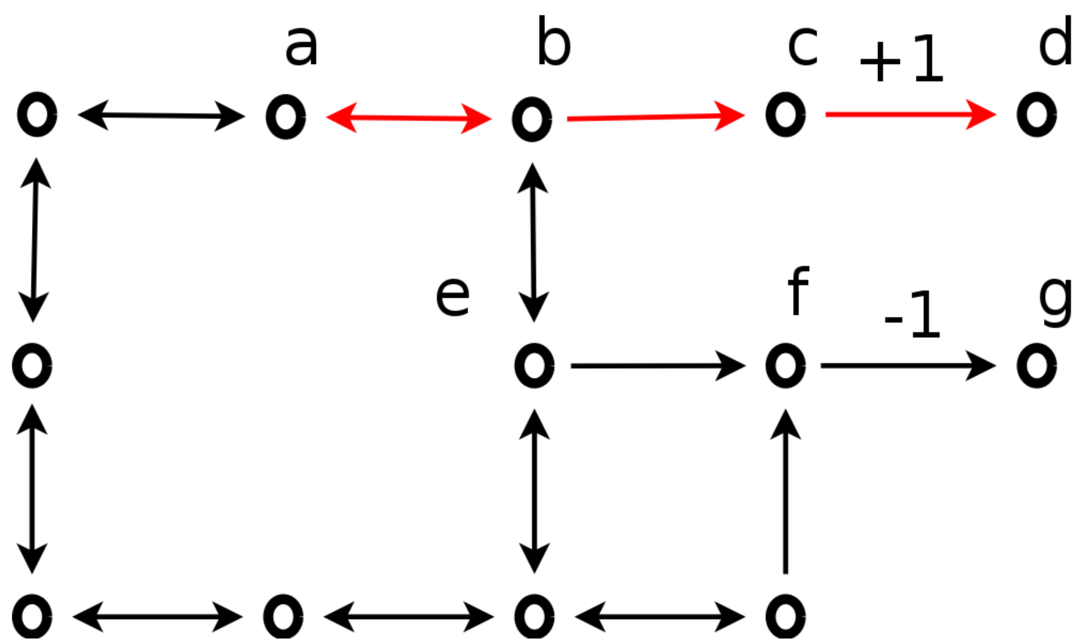
Dual task

- 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?



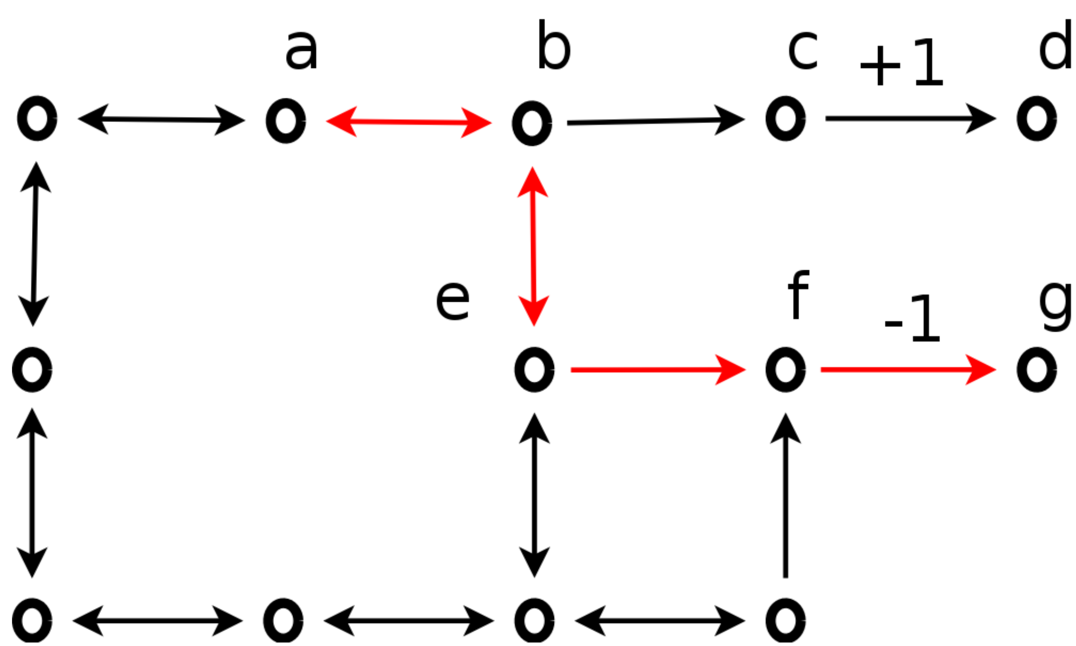
Dual task - naive learning example

- 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

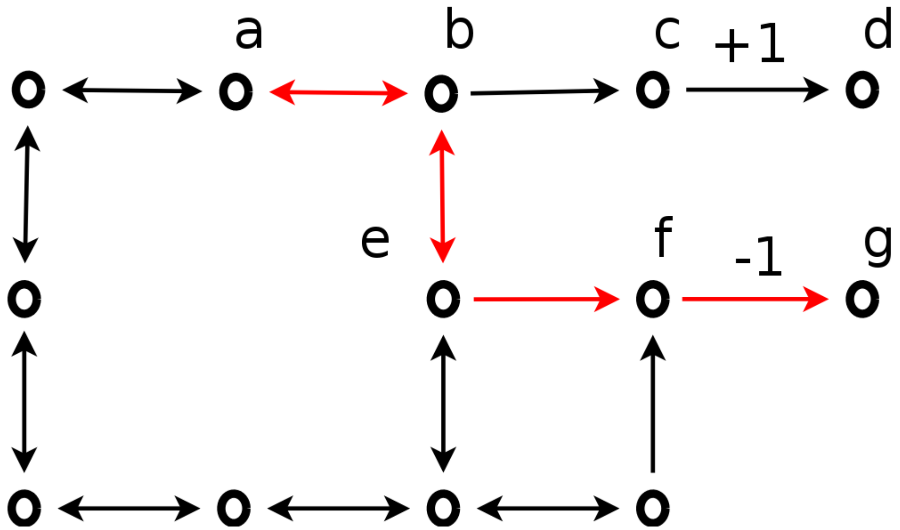
- State-action function $Q(\mathbf{x}, \mathbf{u}) : X \times U \rightarrow \mathbb{R}$
- Mean sum of discounted rewards when choosing action $\mathbf{u}$ from state $\mathbf{x}$.
- How can we learn from recorded trajectories and corresponding rewards?
- $\tau_1 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b},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



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?
- $\tau_1 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{R}, \mathrm{c}, \mathrm{R}, \mathrm{d}), \qquad r(\tau_1) = 1$
- $\tau_2 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{D}, \mathrm{e}, \mathrm{R}, \mathrm{f}, \mathrm{R}, \mathrm{g}), \ \ r(\tau_2) = -1$

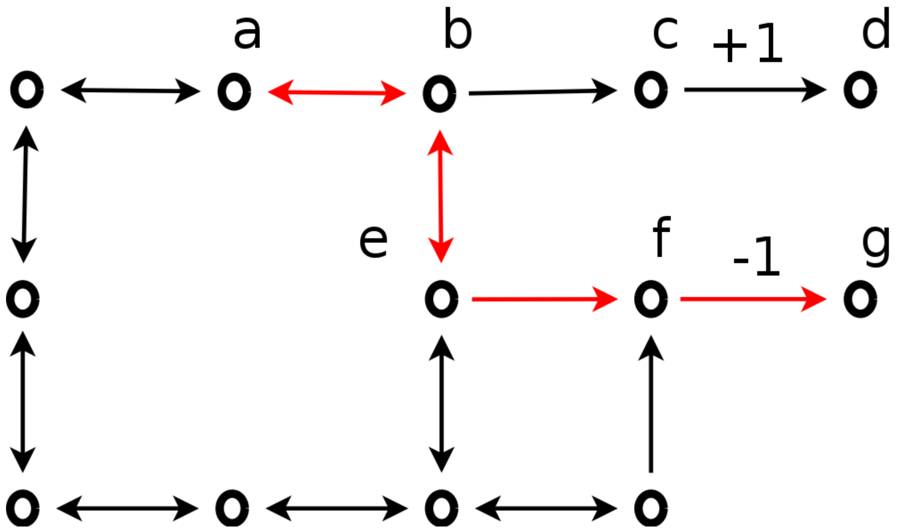
Q	R - right	D - down
a		
b		
c		
e	?	



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?
- $\tau_1 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{R}, \mathrm{c}, \mathrm{R}, \mathrm{d}), \qquad r(\tau_1) = 1$
- $\tau_2 : \ (\mathrm{a}, \mathrm{R}, \mathrm{b}, \mathrm{D}, \mathrm{e}, \mathrm{R}, \mathrm{f}, \mathrm{R}, \mathrm{g}), \quad r(\tau_2) = -1$

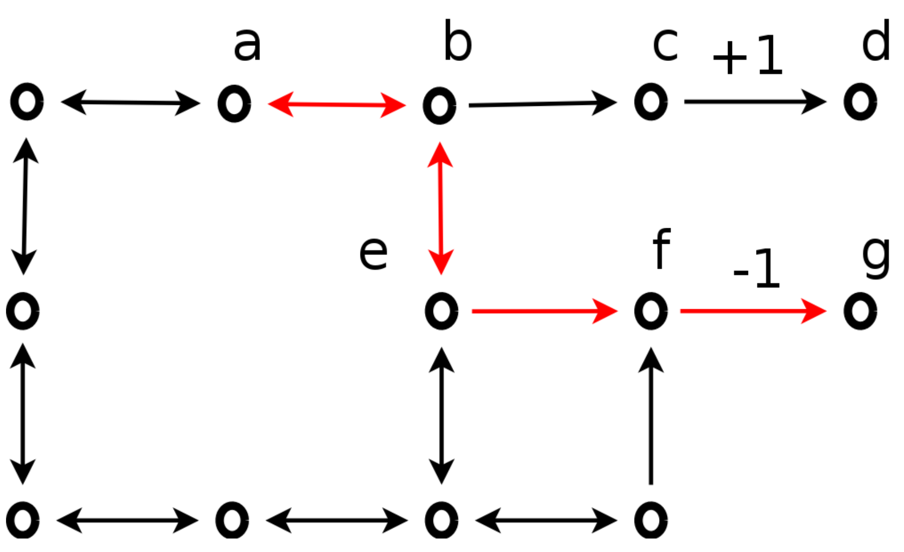
Q	R - right	D - down
a		
b		
c		
e	-1	



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?

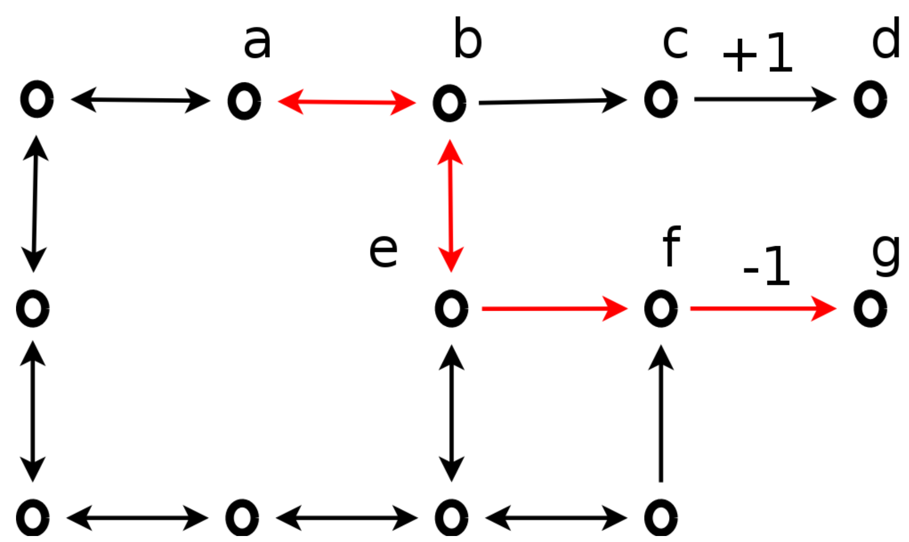
Q	R - right	D - down
a	**0**	
b	1	-1
c	1	
e	-1	



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?

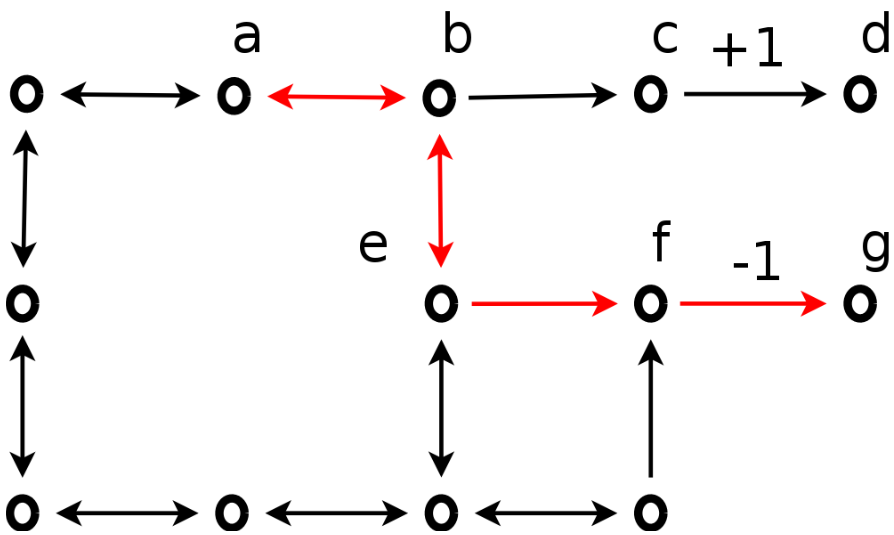
Q	R - right	D - down
a	**0**	
b	1	-1
c	1	
e	-1	



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 transition deterministic

$$p(\mathbf{x}'|\mathbf{u}, \mathbf{x}) = 1 \quad \Rightarrow \quad \mathbf{x} \rightarrow \mathbf{x}'$$

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

recursive definition of Q

- Define $Q(\mathbf{x},\mathbf{u})$ recursively:
  - If transition deterministic

$$p(\mathbf{x}'|\mathbf{u},\mathbf{x}) = 1 \quad \Rightarrow \quad \mathbf{x} \rightarrow \mathbf{x}'$$

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

  - If transition stochastic

$$p(\mathbf{x}'|\mathbf{u},\mathbf{x}) < 1 \quad \Rightarrow \quad \mathbf{x} \rightarrow ?$$

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

(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:

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

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:

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

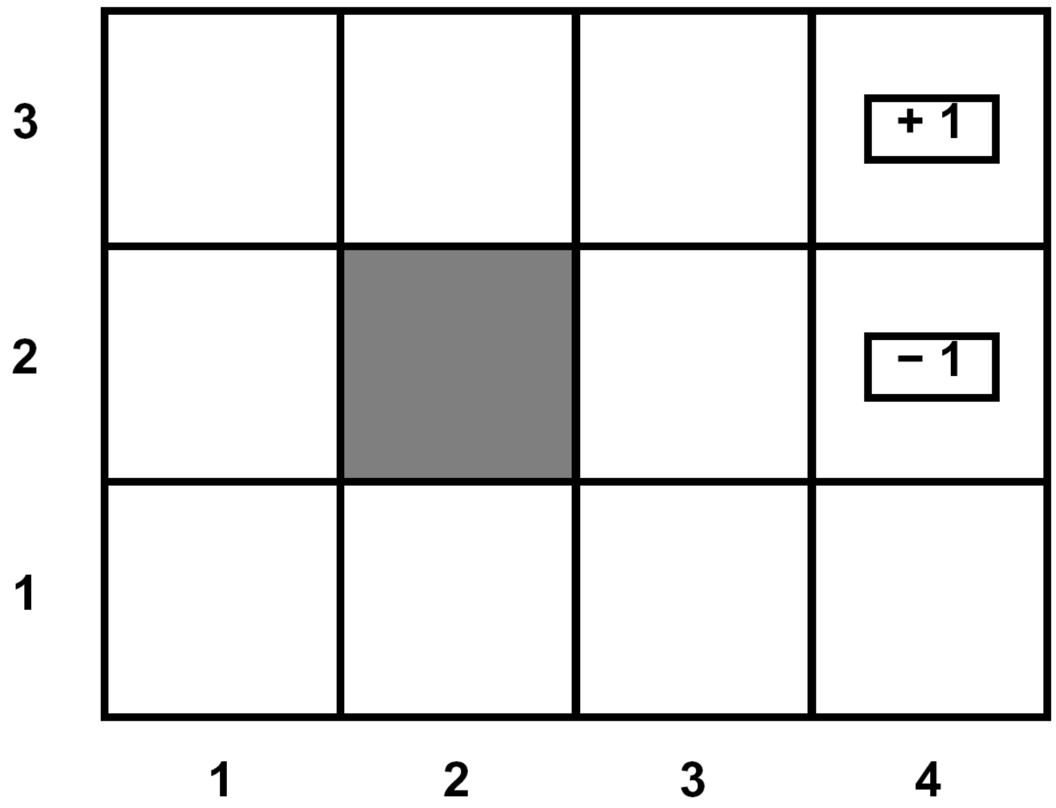
- For $\mathbf{x} \in X, \ \mathbf{u} \in U$

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

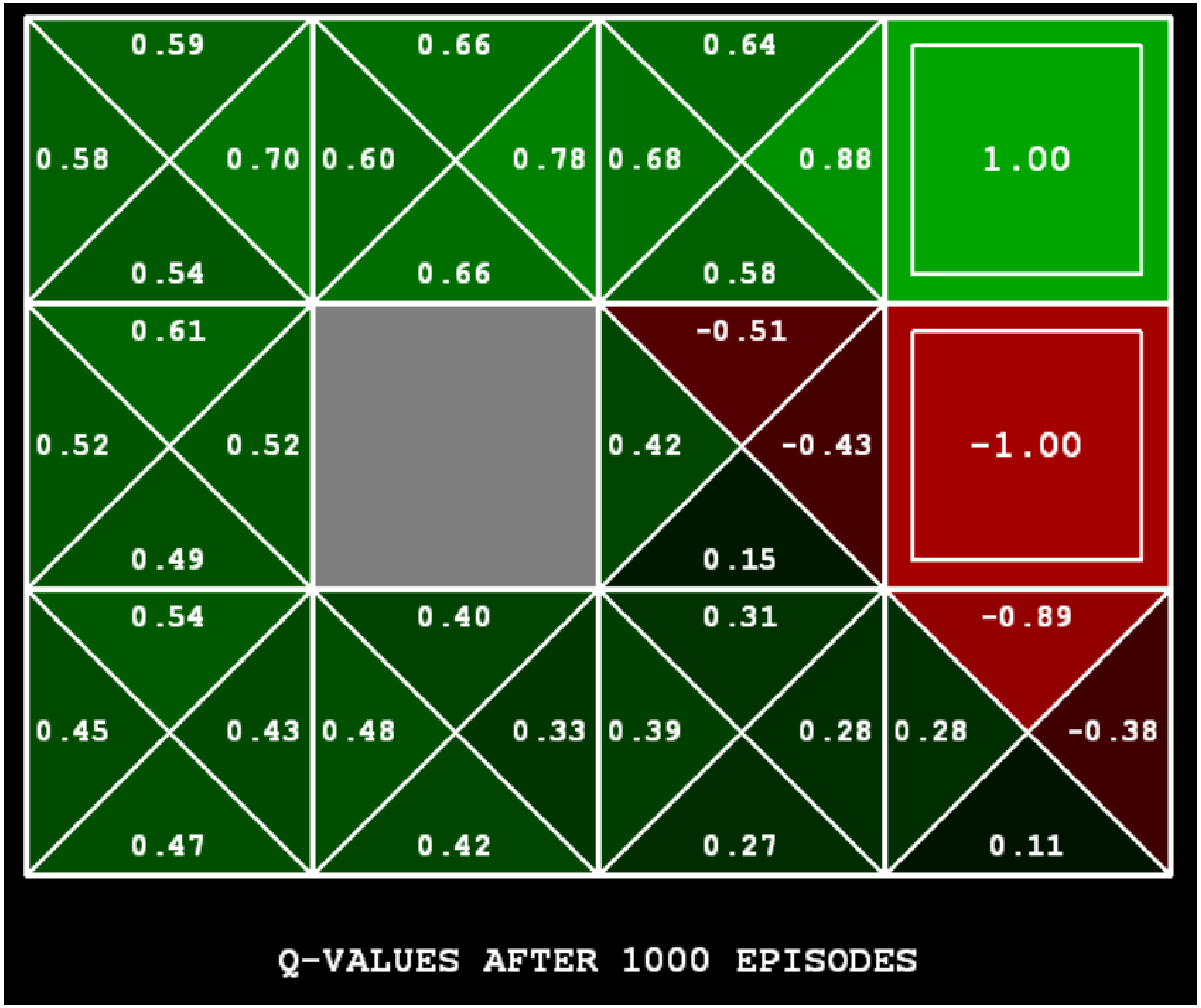
- End

State-value function example I - grid-world

- Show python demo `00_grid_world` and `01_grid_world_noise`

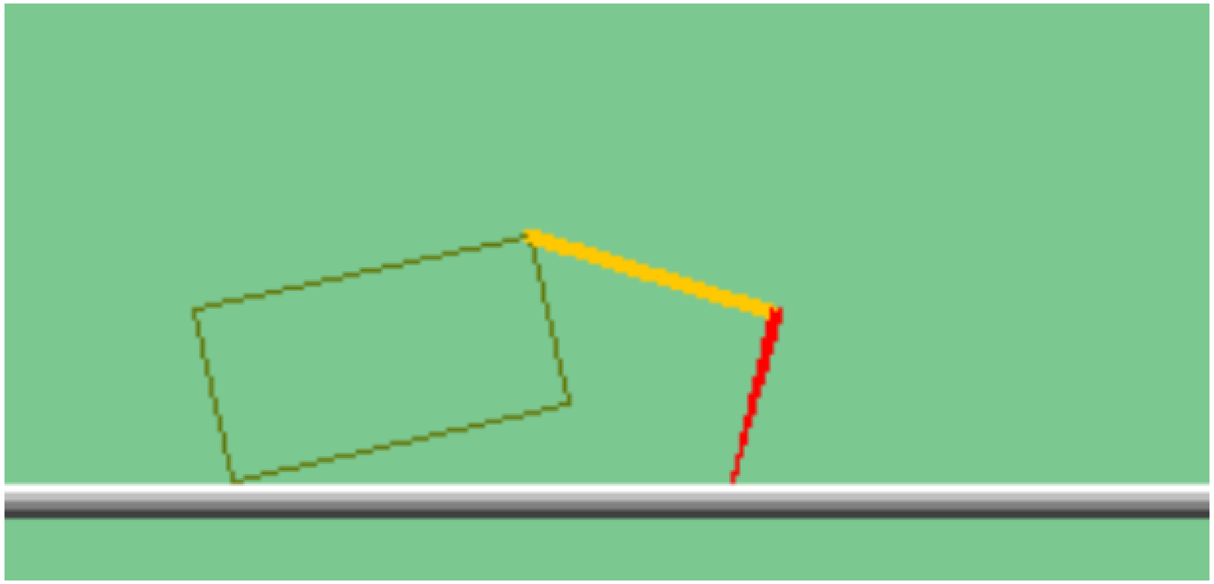


State-value function example I - grid-world



State-value function example II - crawler

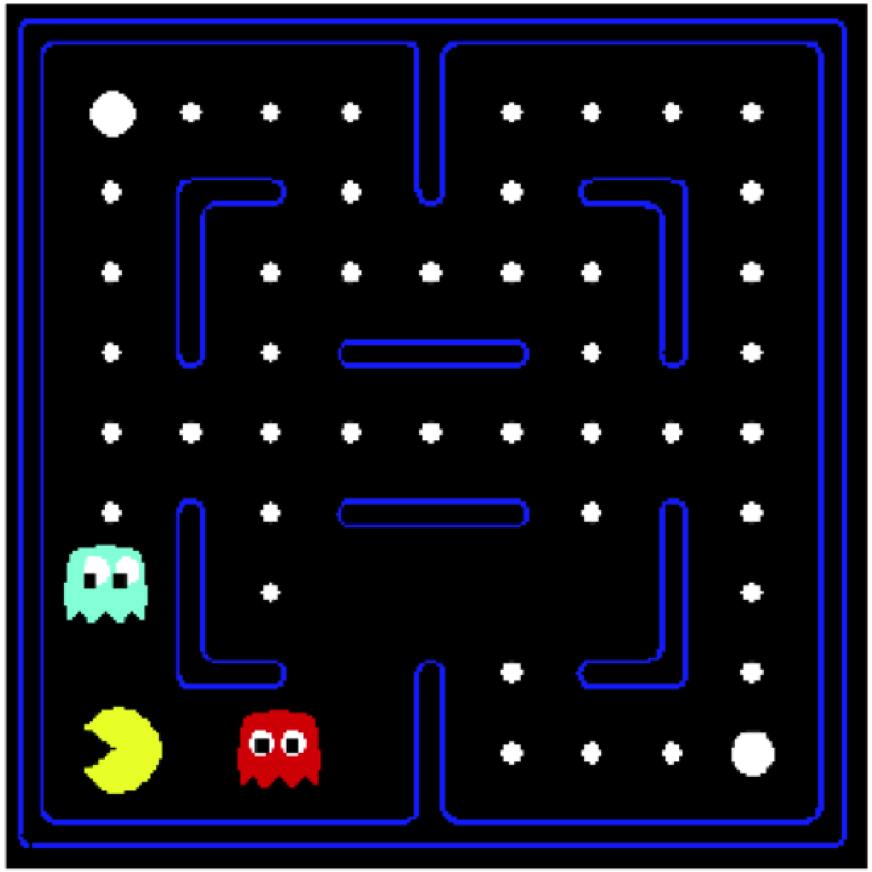
- Show python demo - `02_crawler`
- What are rewards?
- What is U, X and Q dimensionality?



Where is the catch?

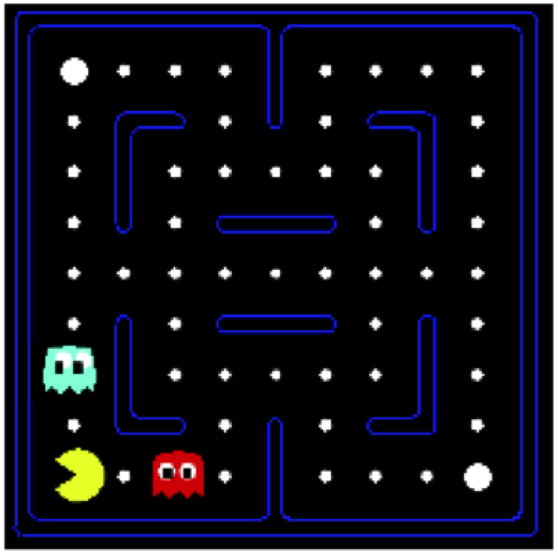
Where is the catch?

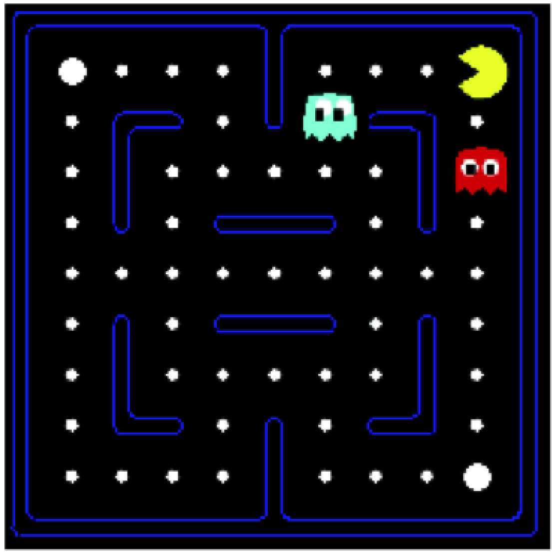
- Curse of dimensionality - considered state space for pacman.
- Show python demo `03_pacman_small_states` and `04_pacman_small_states_long_training`

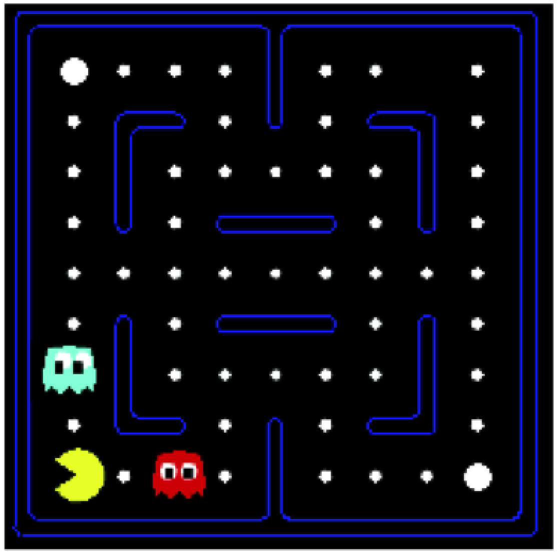


Where is the catch?

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



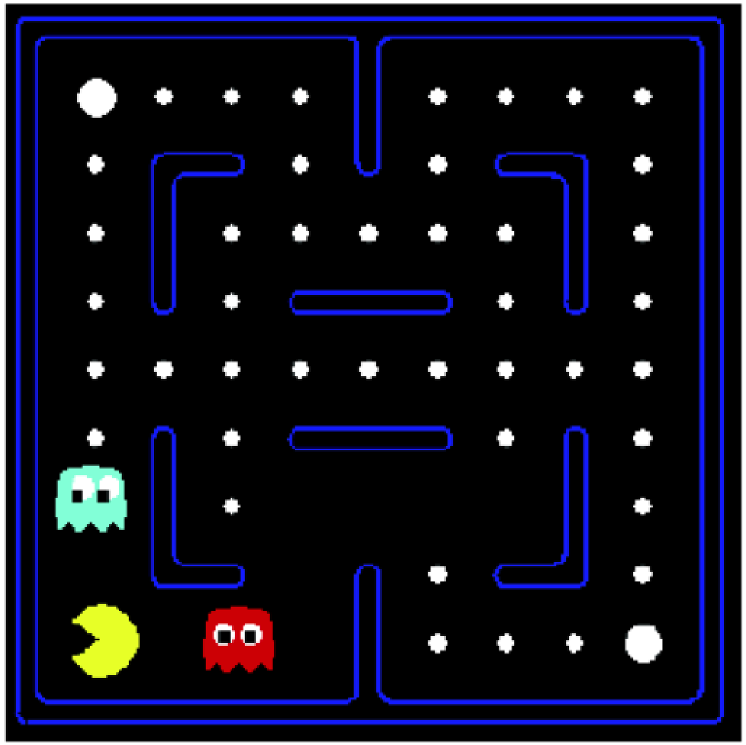




Where is the catch?

- Curse of dimensionality - we need to replace high-dimensional states $\mathbf{x}$ and control $\mathbf{u}$ by low-dimensional features $\Phi(\mathbf{x}, \mathbf{u})$.
- Show python demo `05_pacman_small_features` and `06_pacman_large_features`

- 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 - 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

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

- Inaccurate Q function - do we really need it?

Where is the catch?

- Curse of dimensionality
- Reward tuning (reasons: reward improvement, initialization, imitation learning).

Where is the catch?

- Curse of dimensionality
- Reward tuning (reasons: reward improvement, initialization, imitation learning).
  - Define reward as $R(\mathbf{x}, \mathbf{u}, \mathbf{x}'|\mathbf{w}) = \mathbf{w}^\top \cdot \Phi(\mathbf{x}, \mathbf{u}, \mathbf{x}')$

Where is the catch?

- Curse of dimensionality
- Reward tuning (reasons: reward improvement, initialization, imitation learning).
  - Define reward as $R(\mathbf{x}, \mathbf{u}, \mathbf{x}'|\mathbf{w}) = \mathbf{w}^\top \cdot \Phi(\mathbf{x}, \mathbf{u}, \mathbf{x}')$
  - Learn policy wrt some weights

Where is the catch?

- Curse of dimensionality
- Reward tuning (reasons: reward improvement, initialization, imitation learning).
  - Define reward as $R(\mathbf{x}, \mathbf{u}, \mathbf{x}' | \mathbf{w}) = \mathbf{w}^\top \cdot \Phi(\mathbf{x}, \mathbf{u}, \mathbf{x}')$
  - Learn policy wrt some weights
  - Use policy $\Rightarrow$ trajectory τ_p with reward $R(\tau_p | \mathbf{w}) = \sum_{\tau_p} R(\mathbf{x}, \mathbf{u}, \mathbf{x}' | \mathbf{w})$

Where is the catch?

- Curse of dimensionality
- Reward tuning (reasons: reward improvement, initialization, imitation learning).
  - Define reward as $R(\mathbf{x}, \mathbf{u}, \mathbf{x}' | \mathbf{w}) = \mathbf{w}^\top \cdot \Phi(\mathbf{x}, \mathbf{u}, \mathbf{x}')$
  - Learn policy wrt some weights
  - Use policy $\Rightarrow$ trajectory τ_p with reward $R(\tau_p | \mathbf{w}) = \sum_{\tau_p} R(\mathbf{x}, \mathbf{u}, \mathbf{x}' | \mathbf{w})$
  - Use expert $\Rightarrow$ trajectory τ_e with reward $R(\tau_e | \mathbf{w}) = \sum_{\tau_e} R(\mathbf{x}, \mathbf{u}, \mathbf{x}' | \mathbf{w})$

Where is the catch?

- Curse of dimensionality
- Reward tuning (reasons: reward improvement, initialization, imitation learning).
  - Define reward as $R(\mathbf{x},\mathbf{u},\mathbf{x}'|\mathbf{w}) = \mathbf{w}^\top \cdot \Phi(\mathbf{x},\mathbf{u},\mathbf{x}')$
  - Learn policy wrt some weights
  - Use policy $\Rightarrow$ trajectory τ_p with reward $R(\tau_p|\mathbf{w}) = \sum_{\tau_p} R(\mathbf{x},\mathbf{u},\mathbf{x}'|\mathbf{w})$
  - Use expert $\Rightarrow$ trajectory τ_e with reward $R(\tau_e|\mathbf{w}) = \sum_{\tau_e} R(\mathbf{x},\mathbf{u},\mathbf{x}'|\mathbf{w})$
  - Find weights making expert better:

$$\mathbf{w}^* = \arg\max_{w} R(\tau_e|\mathbf{w}) - R(\tau_e|\mathbf{w})$$

Where is the catch?

- Curse of dimensionality
- Reward tuning (reasons: reward improvement, initialization, imitation learning).
  - Define reward as $R(\mathbf{x},\mathbf{u},\mathbf{x}'|\mathbf{w}) = \mathbf{w}^\top \cdot \Phi(\mathbf{x},\mathbf{u},\mathbf{x}')$
  - Learn policy wrt some weights
  - Use policy $\Rightarrow$ trajectory τ_p with reward $R(\tau_p|\mathbf{w}) = \sum_{\tau_p} R(\mathbf{x},\mathbf{u},\mathbf{x}'|\mathbf{w})$
  - Use expert $\Rightarrow$ trajectory τ_e with reward $R(\tau_e|\mathbf{w}) = \sum_{\tau_e} R(\mathbf{x},\mathbf{u},\mathbf{x}'|\mathbf{w})$
  - Find weights making expert better:

$$\mathbf{w}^* = \arg\max_{w} R(\tau_e|\mathbf{w}) - R(\tau_e|\mathbf{w})$$

  - Iterate.

Where is the catch?

- Curse of dimensionality
- Reward tuning (reasons: reward improvement, initialization, imitation learning).
- Exploration vs exploitation (show demo `02_crawler`).
  - 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 (reasons: reward improvement, initialization, imitation learning).
- Exploration vs exploitation (show demo `02_crawler`).
  - ϵ-greedy exploration
  - or exploration extension $Q(\Phi(\mathbf{x}, \mathbf{u})) + \frac{k}{N(\Phi)}$
- Simulator/model (inaccuracy problem but it can decrease real-world interactions).
- 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?

- Pacman (show roomba pacman !!!) http://inst.eecs.berkeley.edu/~cs188/pacman/html/navigation.html?page=p3/p3_introduction
- Work with us on:
  - Nifti robot - show adaptive traversability demo!
  - better IRO tasks - can doc.Ing.Zlo,CSc. be captured via reinforcment learning?
- Starcraft competition http://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/
- TORCS - Racing **and demolishon derby** simulator competition. http://en.wikipedia.org/wiki/TORCS