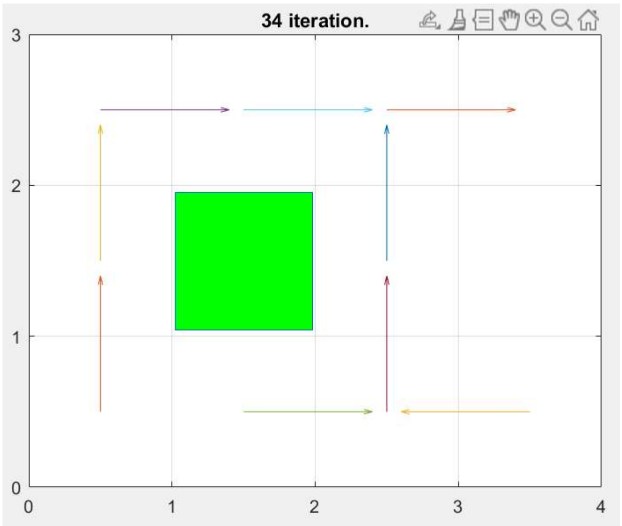
Reinforcement Learning

KUO-SHIH TSENG (曾國師)
DEPARTMENT OF MATHEMATICS
NATIONAL CENTRAL UNIVERSITY, TAIWAN

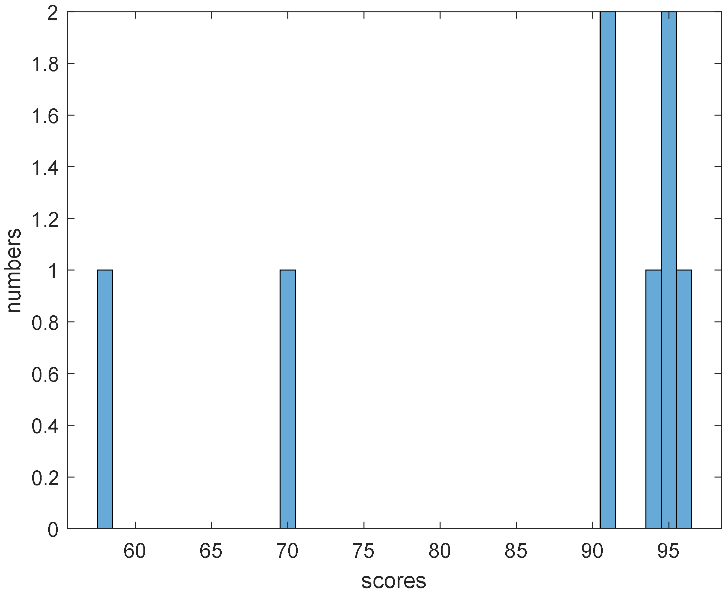
2021/04/28

HW2

- MDP



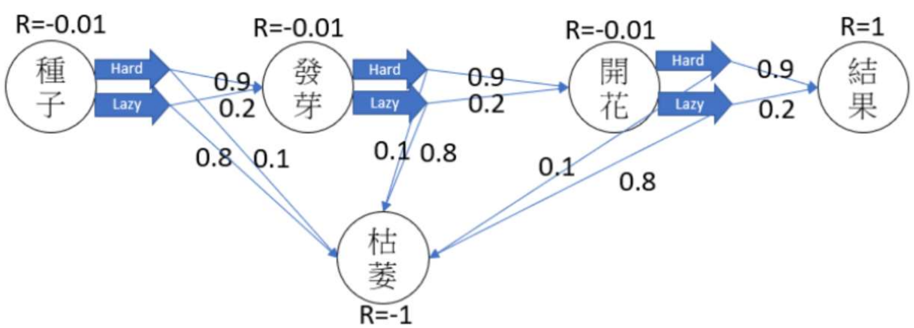
34 iteration.
3
2
1
0
0
1
2
3
4

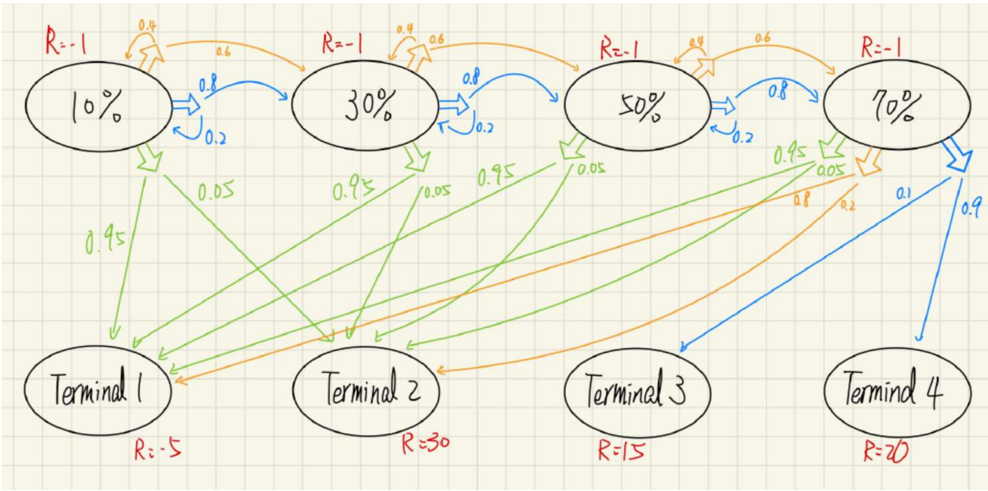


numbers
scores
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
60
65
70
75
80
85
90
95

HW2

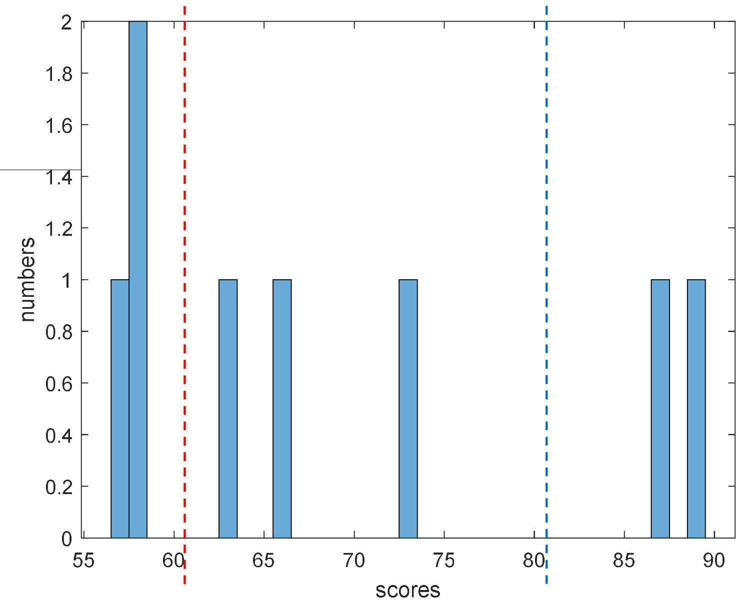
- MDP example





Midterm

1. Perception
   a) Smoothing (20%)
   b) Bayesian search in 2 grids (15%)
   c) Bayesian search in 3 grids (15%)
2. Decision-making
   a) MDP (20%)
   b) Bellman EQ (15%)
   c) MCTS (15%)



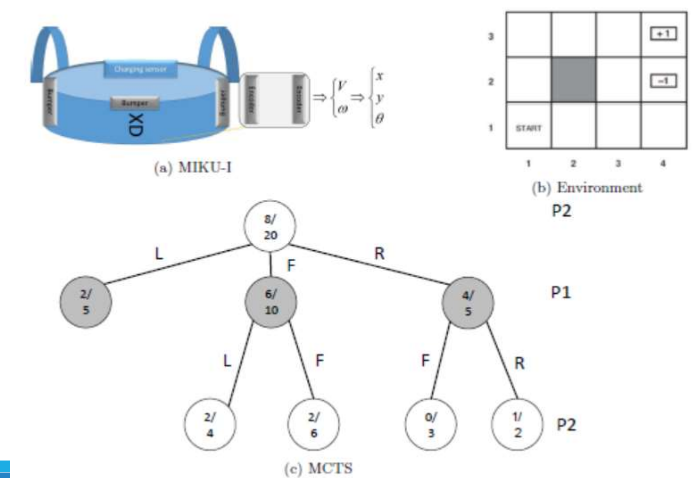


Figure 1: Illustration of the problem.

Course Announcement

- Deadline of final project proposal: **05/12 0am**
- Final Project Proposals should include the following :
  - The goal of learning
  - MDP formulation of your project
    - State, action, transitional probability, rewards, and discount terms.
  - Features of Q function
  - The data flow of your robot
- You can find the sample files of final project on LMS.
- You should cite the prior work (Reference).

Course Announcement

- HW3 was released (Deadline: **5/26 0am**):
- A RL problem (40%)
- A supervised learning problem (40%)
- A RL proof (20%)
- You can start to work on P1 and P3 of HW3 today. Work on it ASAP!

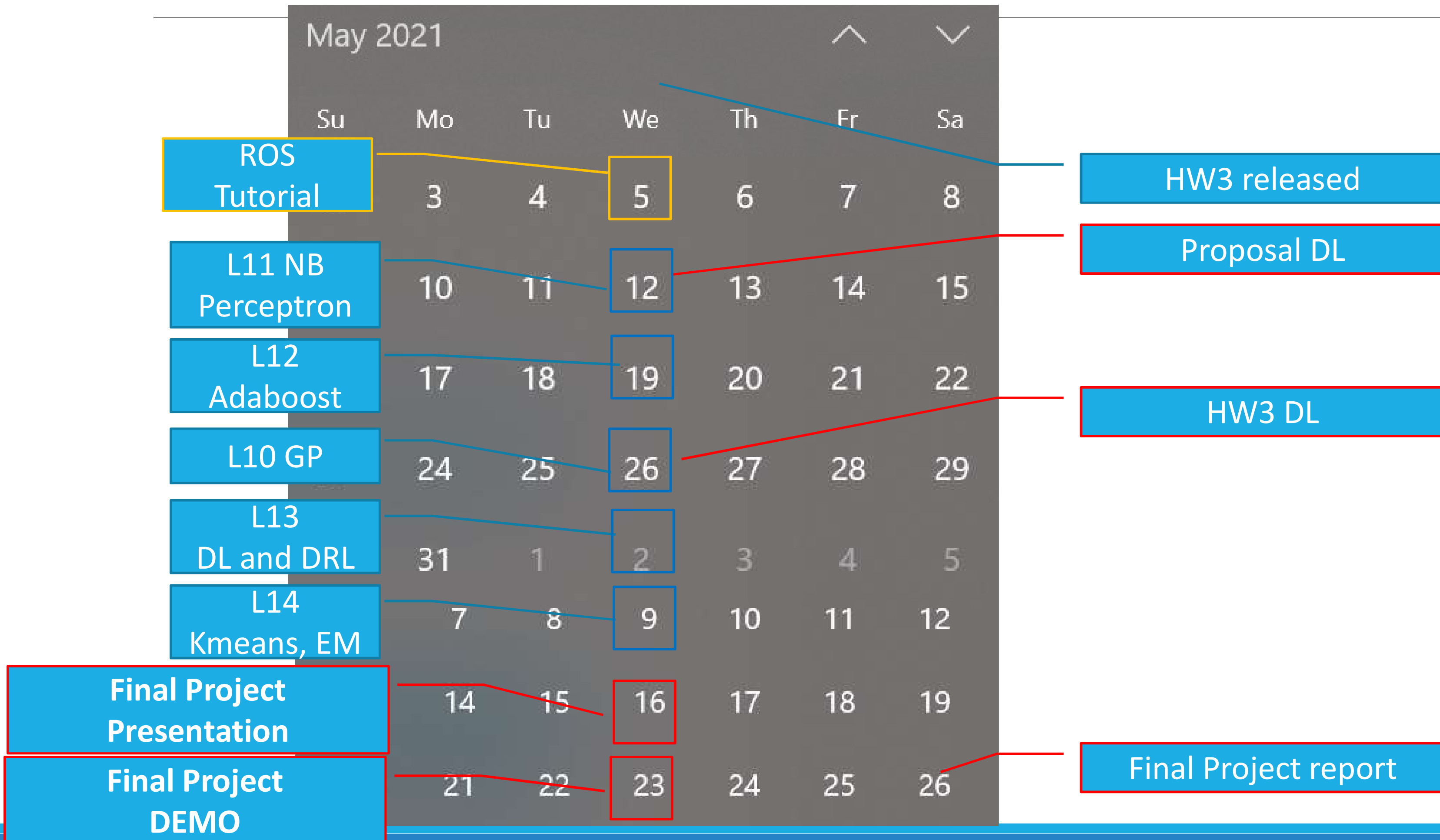
Course Announcement

- The grade for this course will consist of the following components:
  - HW1 10%
  - HW2 10%
  - Midterm Exam 30%
  - HW3 10%
  - Project Proposal (1-2 page) **10%**
  - Project Presentation and Demonstration **10%**
  - Project Report (4-8 pages) **20%**
- NOTICE: I will NOT do curve fitting (e.g., “sqrt(X)*6” for your score) for your scores.

Course Announcement

- To complete your final project, we will have a ROS tutorial for Minibot on 5/5.
  - Take your laptop to class
  - Take your **Minibot** or **bebop** to class with full charged battery.
  - Install virtualbox https://www.virtualbox.org/ before the class
- Download the image file from Minibot wikipage before the class
  - VirtualBox Image (password: hypharos) : https://drive.google.com/open?id=1xTVsPet6WT48Psete6iIkgg-gi1QdOht
- You can use your Minibot or Bebop in this building. You should return your robot to M-213 when you are not using it.

Course Announcement



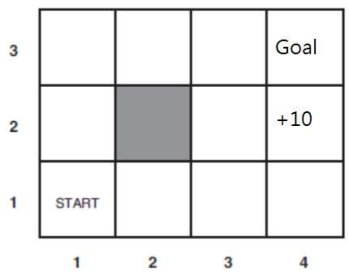
Outline

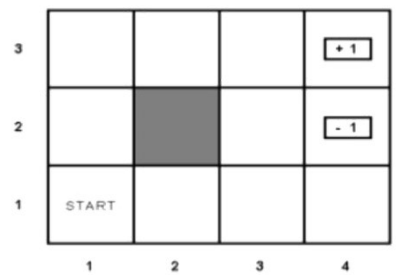
- Need
- Reinforcement learning
- Monte Carlo methods
- TD- learning
- Q-learning
  - Tubular Q-learning
  - function based Q-learning
- Linear regression
- Regularization
- Appendix – More about RL

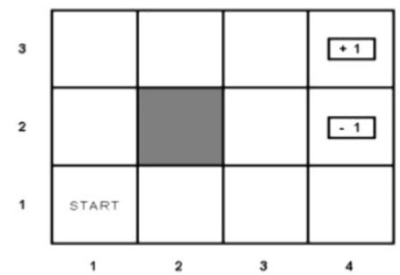
Outline

- LRTA* | MDP (RL) | POMDP

Deterministic action | Probabilistic actions | Probabilistic actions and states







$s, a \rightarrow s'$ | $P(s'|s,a)$ | $P(s'|s,a), P(s)$

L2: Uninformed search
L3: Heuristic search (LRTA*)
L4: Adversarial search
L5: Bayes theorem
L6: Bayes theorem over time

L7: MDP
L9: Reinforcement learning
L10: GP and LWPR
L11: Naïve Bayes and Perceptron
L12: Adaboost
L13: Deep learning and DRL

L8: POMDP

Outline

[Problem solving]

Search problems

Adversarial Search

[Perception and Uncertainty]

Bayes Theorem

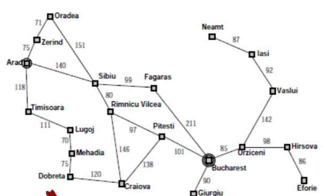
Bayes Filter and Smoothing

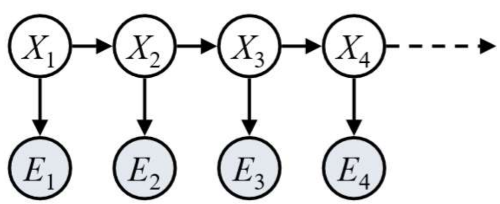
[Learning and Decision-making]

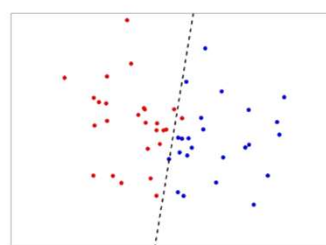
Supervised learning

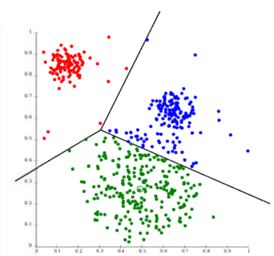
Unsupervised learning

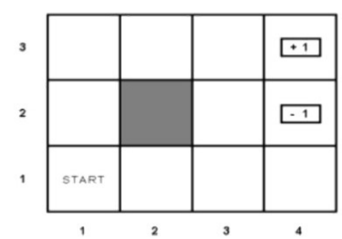
Reinforcement learning











Need

- The disadvantages of iterative approaches for solving MDP are
  - (1) The robot needs to know the environments or states.
  - (2) they need a transition probability model;
  - (3) they need to update each state to converge to a solution.
- For most online learning applications, the robots do not know P and need to update utility immediately.

$[GIVEN]$

$S: state$

$A: action$

$P(s'|s,a): Transition\ probability$

$R(s,a): reward$

$\gamma: discount$

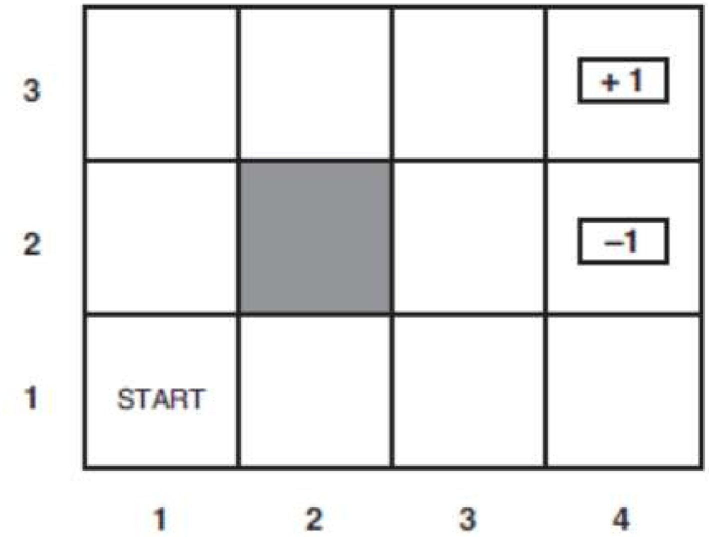
$[Find]$

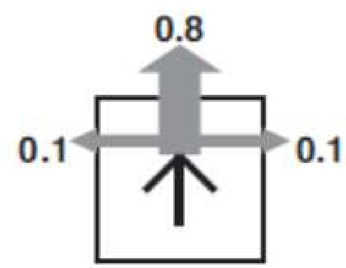
$$\pi^* = \arg\max U^{\pi}(s)$$

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

Need

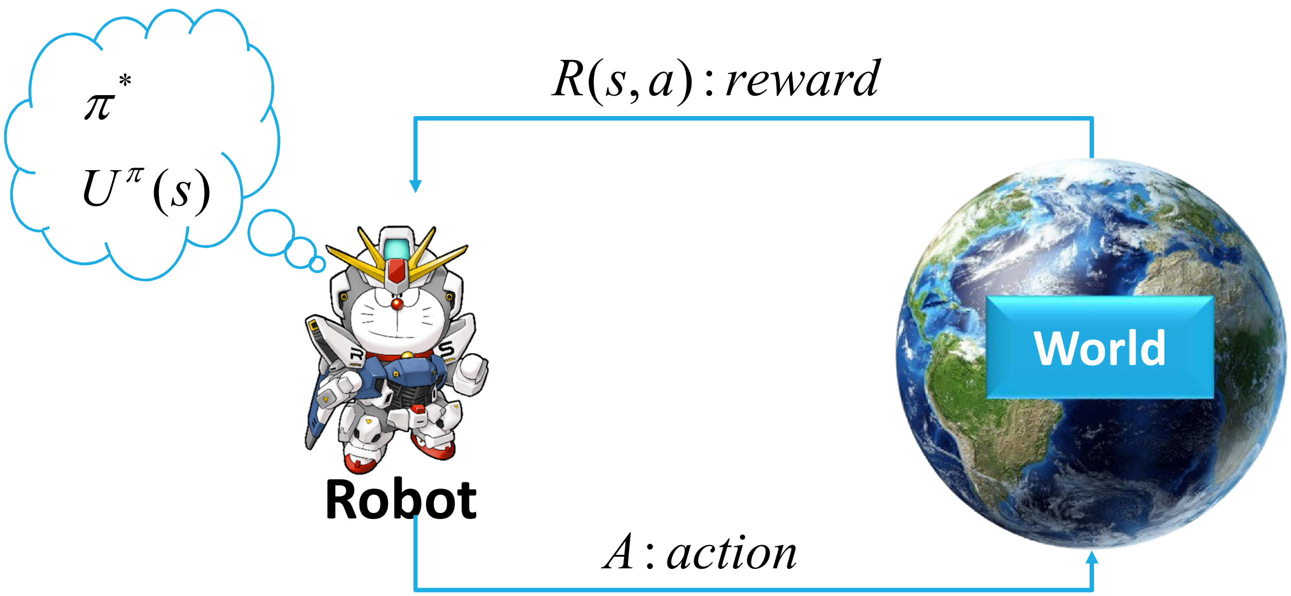
- Let the robot interact with the world and receive reward immediately.
- The robot can try many times to update its utility function. Then, the robot can find optimal actions based on the learned utility function.





Reinforcement learning

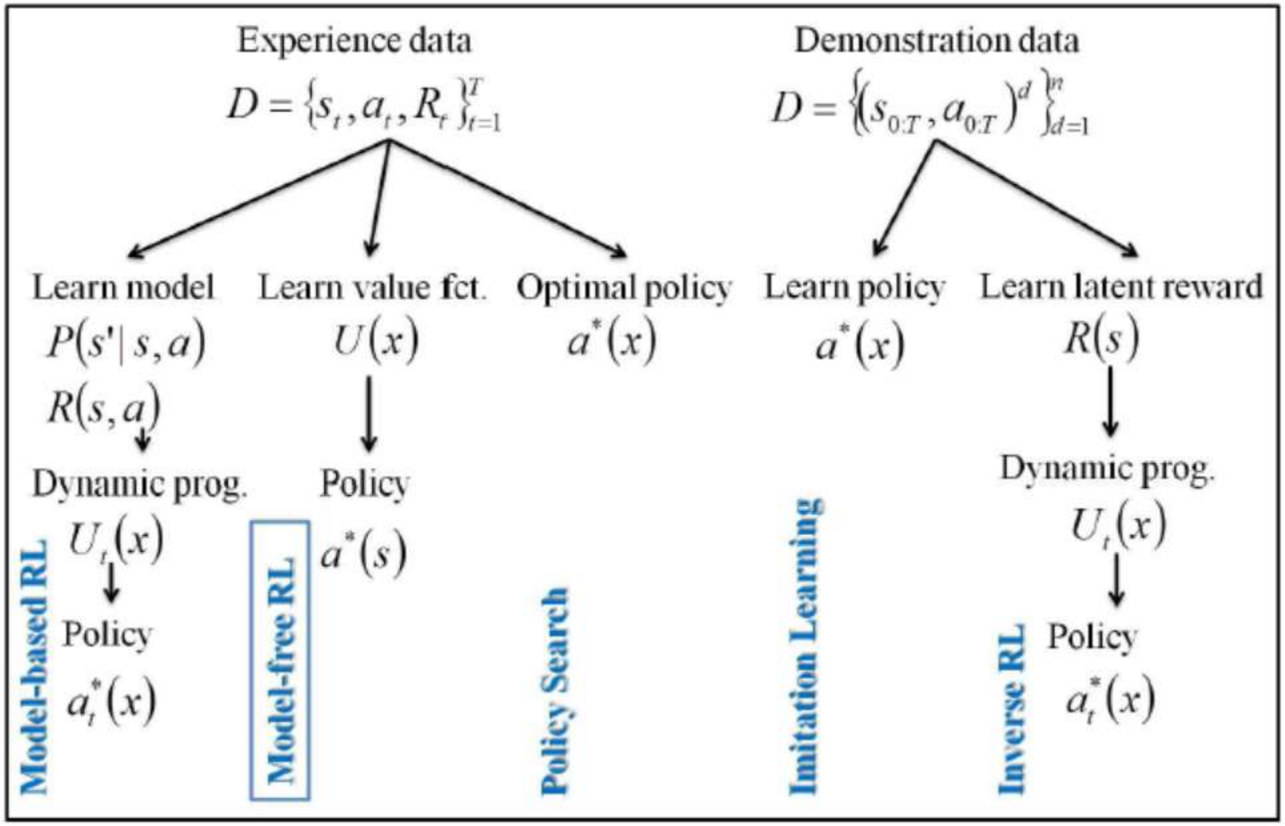
- Let the robot interact with the world and receive the reward immediately. RL is inspired by animals. Animals learn the world via interactions!



Reinforcement learning

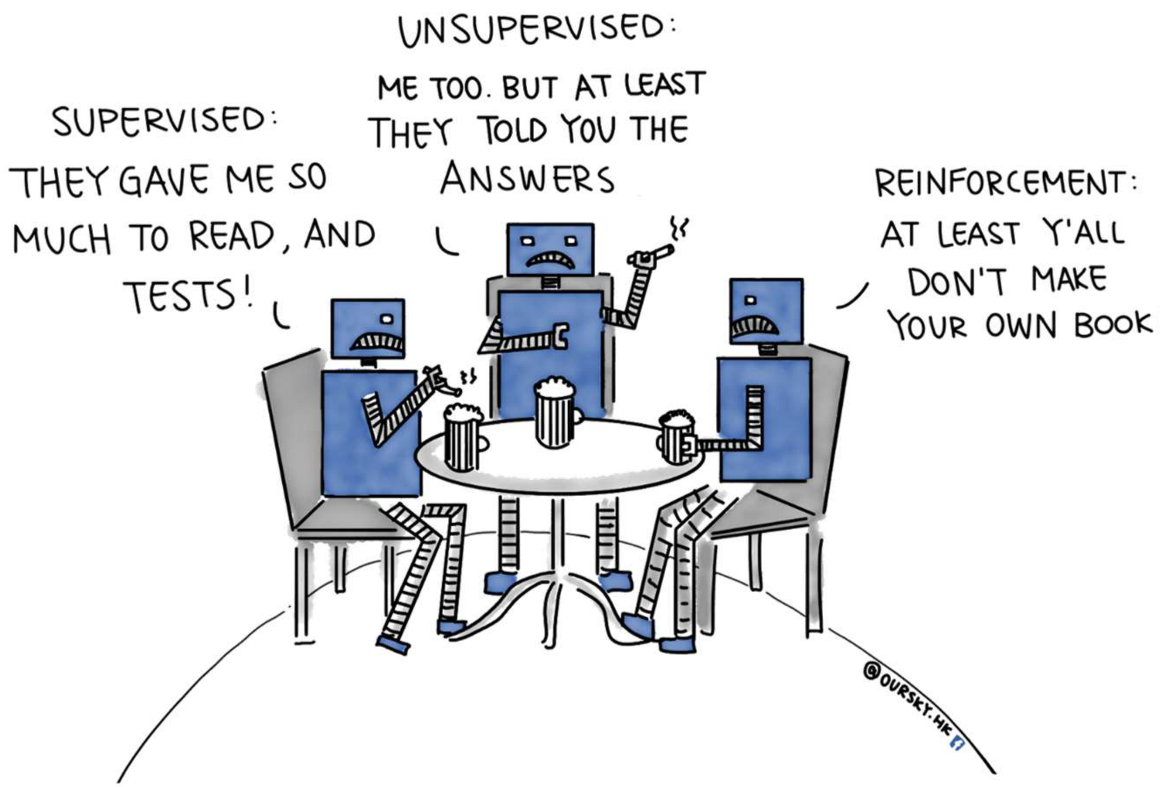
- During the past 10 years, robots are able to do amazing ***control*** tasks (~60 DOFs) using RL.
- In 2015, AlphaGo can beat the best human players.
- In 2015, AI can play 49 Atari games.
- RL+DL is the future of AI?

Reinforcement learning



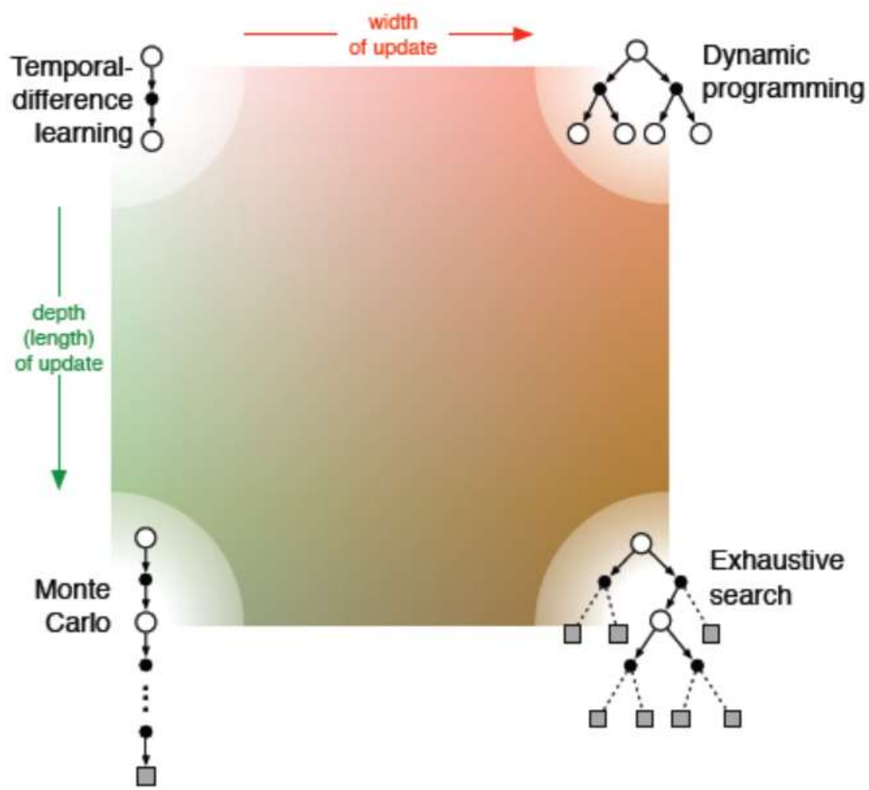
[1] Marc Toussaint. Machine learning and robotics. ICML tutorial, 2011.

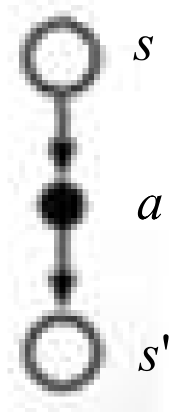
Reinforcement learning



From: Facebook OURSKY

Reinforcement learning

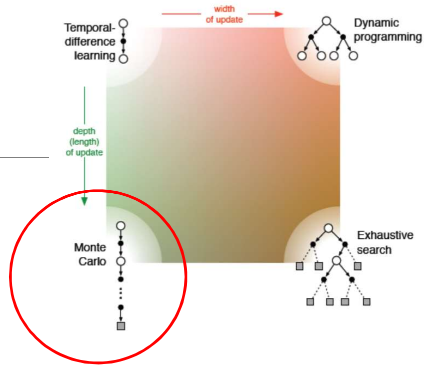




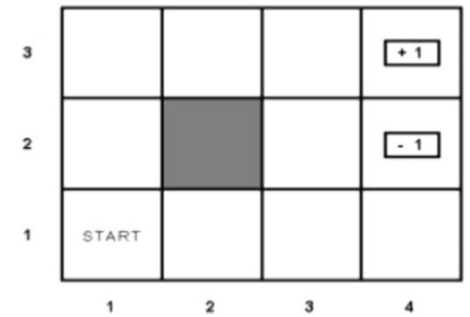
Richard S. Sutton and Andrew G. Barto, "Reinforcement Learning: An Introduction," MIT Press, 2nd edition, 2018.

Monte Carlo methods

- Monte Carlo approaches can be applied to compute the utility function.
- For example, there are some trial samples.



$$(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$$
$$(1,1)_{-.04} \rightsquigarrow (1,2)_{-.04} \rightsquigarrow (1,3)_{-.04} \rightsquigarrow (2,3)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (3,3)_{-.04} \rightsquigarrow (4,3)_{+1}$$
$$(1,1)_{-.04} \rightsquigarrow (2,1)_{-.04} \rightsquigarrow (3,1)_{-.04} \rightsquigarrow (3,2)_{-.04} \rightsquigarrow (4,2)_{-1} \ .$$



- The utility function can be solved by Bellman EQ and these samples:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(S_t)\right]$$

or

$$U(s) = R(s) + \gamma \max_{a}\left[\sum_{s'} U(s')P(s' \mid s, a)\right]$$

TD-learning

- Temporal-difference (TD) learning is proposed based on dynamic programming (DP) and Monte Carlo concepts.
- TD updates the utility function based on the difference of utility after each action. Hence, it needs a lot of samples (Monte Carlo) and updates parts of utility (DP) without reaching terminals.
- The updated equation of the utility function is as follows:

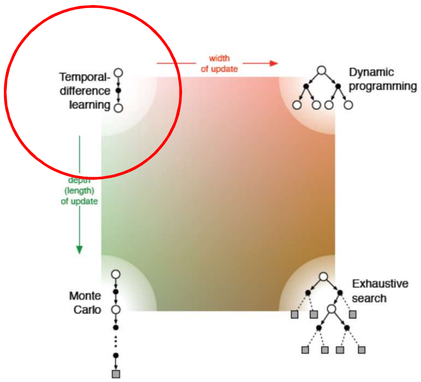
$$U(s) \leftarrow U(s) + \alpha\big(R(s) + \gamma U(s') - U(s)\big)$$

$U(s)$: utility function

$R(s)$: reward

α : learning rate

- TD is a **model free** approach.



TD-learning

- Given an action (a), reward (R) and next state (s'), the robot updates U(s).

$$s = (3,1)$$

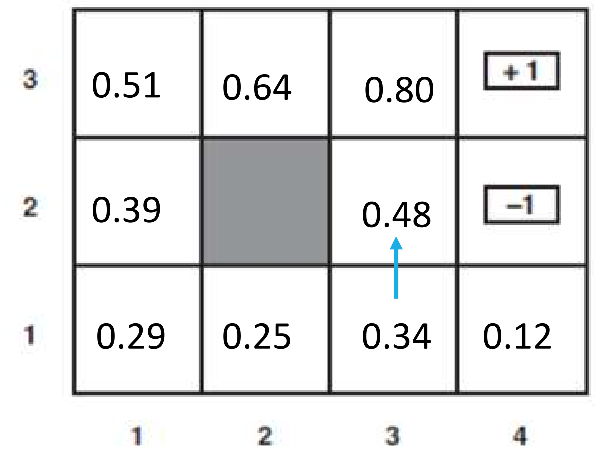
$$s' = (3,2)$$

$$\gamma = 1, \alpha = 0.1$$

$$U(s) \leftarrow U(s) + \alpha\left(R(s) + \gamma U(s') - U(s)\right)$$

$$U(s) \leftarrow 0.34 + 0.1\left(-0.04 + 0.48 - 0.34\right)$$

$$U(s) = 0.35$$

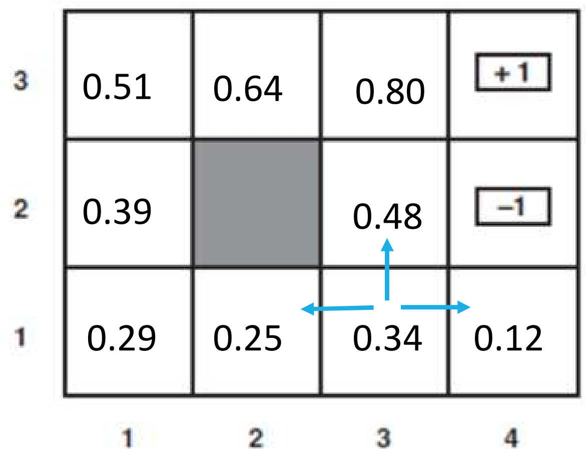


When $\left(R(s) + \gamma U(s') - U(s)\right) = 0$,

$U(s)$ will not be updated.

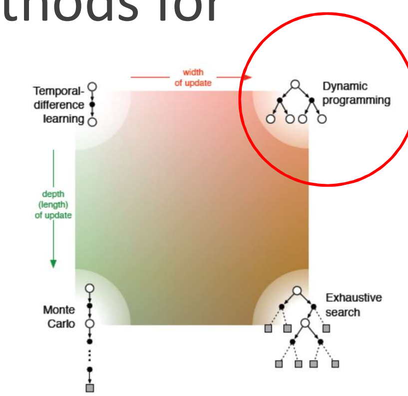
TD-learning

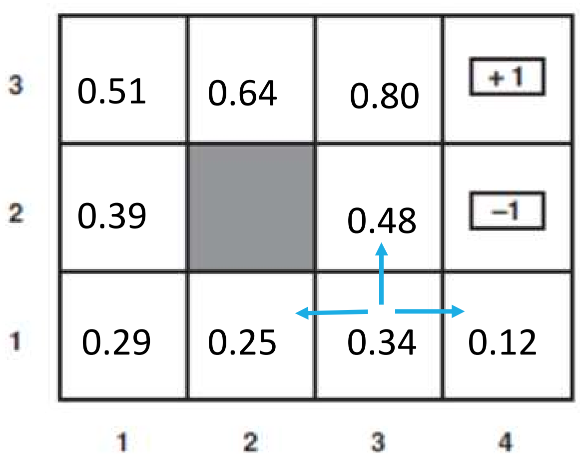
- A disadvantage of TD is that it needs a lot of samples since the robot does not know **what the next action is** while collecting data.
- To improve TD, the robot needs a function with **actions** and **states** variables. *U(s)* → *Q(s,a)*



Q-learning

- In Q-learning, the robot learns an **action-utility** function Q(s,a).
- With the learned Q function, the robot knows what the next best action is. Since Q-learning does not need transition probability and action section models, it is one of the *model-free* methods for reinforcement learning.





$$s = (3,1)$$

$$\begin{cases} Q(s, a = left) \\ Q(s, a = up) \\ Q(s, a = right) \end{cases}$$

Pick up the action with the maximal Q value

$$U(s) = \max_a Q(s,a)$$

Q-learning

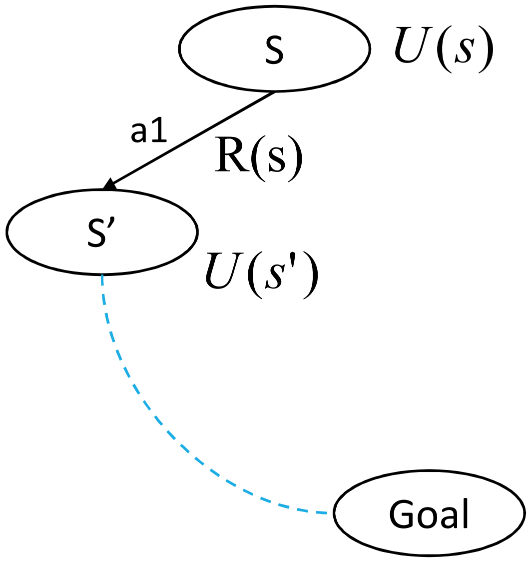
- Bellman equation: Assuming the agent chooses the optimal action, The utility of a state is <u>the immediate reward for that state</u> + <u>the expected discounted utility of the next state</u>.

$$U(s) = \underbrace{R(s)}_{\text{immediate reward}} + \underbrace{\gamma \max_a \left[\sum_{s'} U(s') P(s' | s, a) \right]}_{\text{expected discounted utility}}$$

- Bellman equation of Q-learning

$$Q(s,a) = \underbrace{R(s)}_{\text{immediate reward}} + \underbrace{\gamma \left[\sum_{s'} \max_{a'} Q(s',a') P(s' | s, a) \right]}_{\text{expected discounted utility}}$$

$$\mathrm{Q}(s,a) = R(s) + \gamma \max_{a'} Q(s',a')$$



Q-learning

- The TD learning can be extended to Q learning.

$$U(s) \leftarrow U(s) + \alpha\left(R(s) + \gamma U(s') - U(s)\right)$$

TD learning

$$\because U(s) = \max_{a} Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

Q learning

$$or$$

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(R(s) + \gamma \max_{a'} Q(s',a')\right)$$

$Q(s,a)$: action – utility function (Q function)

$R(s)$: reward

α : learning rate

Q-learning

- Q-learning

$[GIVEN]$

$S: state$

$A: action$

$R(s,a): reward$

$\gamma: discount$

$\alpha:$ learning rate

$[Find]$

$Q(s,a):$ action – utility function (Q function)

$U(s) = \max_a Q(s,a)$

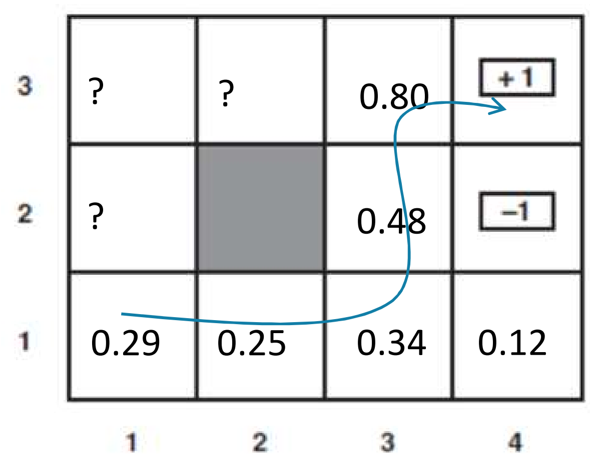
$\pi^* = \arg\max U^\pi(s)$

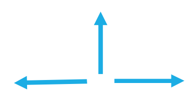
$U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

Q-learning

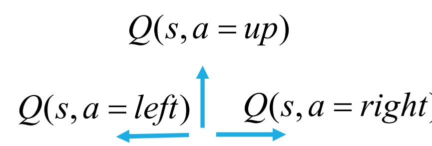
- Trade-off between exploration and exploitation of the Q function.
- If the robot always select the action with the maximal utility, it could follow the same path without exploring other paths.
- ε-greedy was proposed to solve this issue. The robot will action randomly with ε probability and select best action with (1- ε) probability.





Exploration:
Random actions

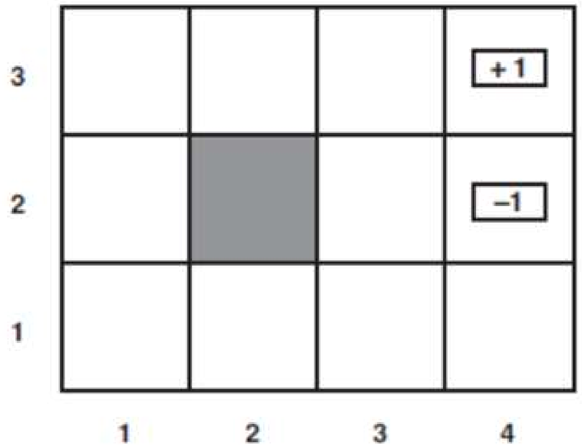
or



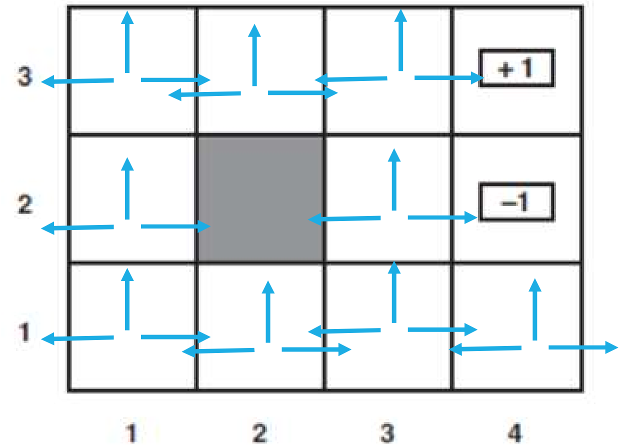
Exploitation:
Pick up the best action based on Q

Tubular Q-learning

- Although Q(s,a) can provide the utility of each action at each state, it means there are more values to learn!



U(s): There are 9 values to learn.

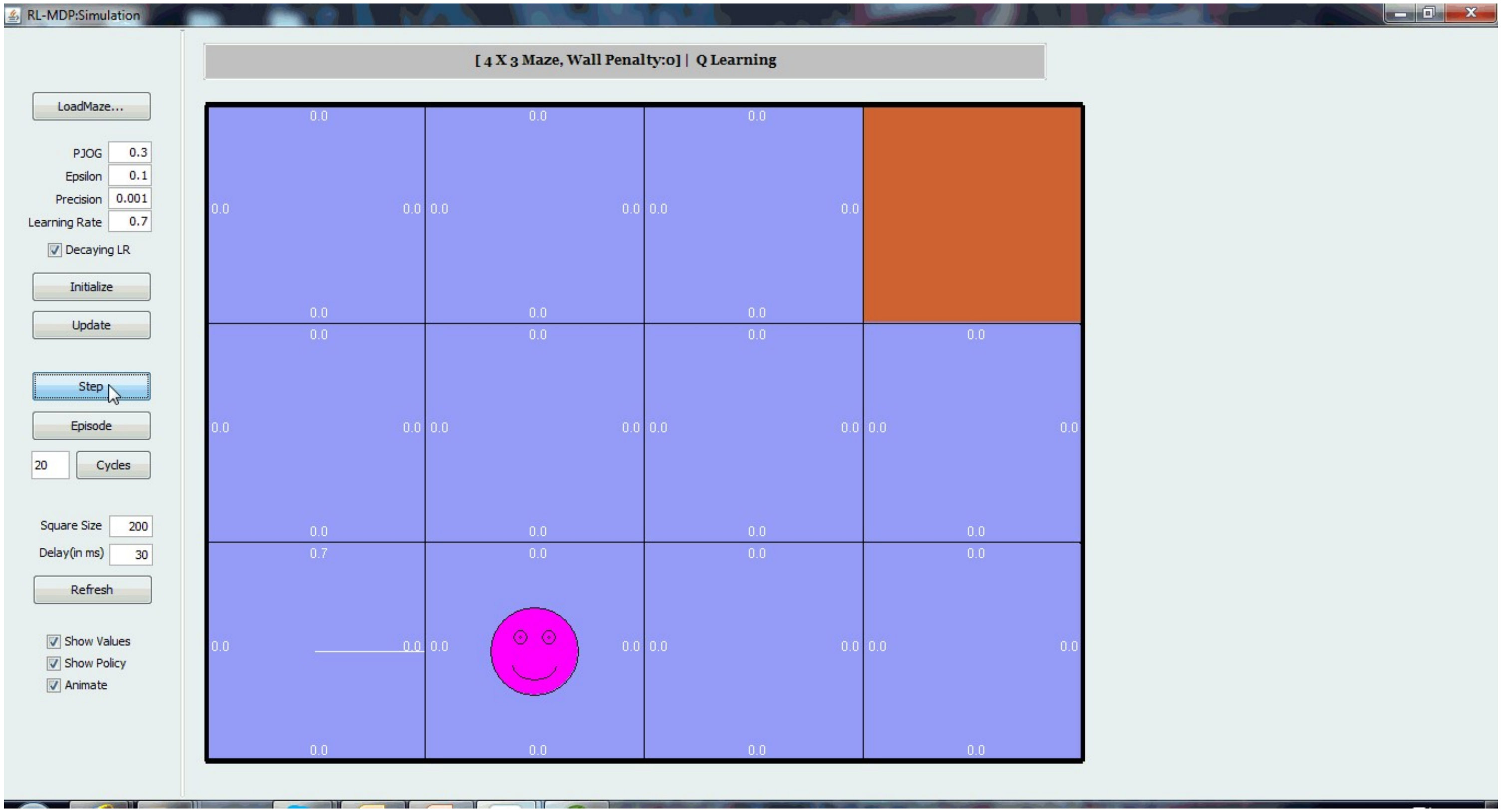


Q(s,a): There are 9*3 values to learn.

$$\text{There are } |S| \cdot |A| \text{ values in Q table}$$

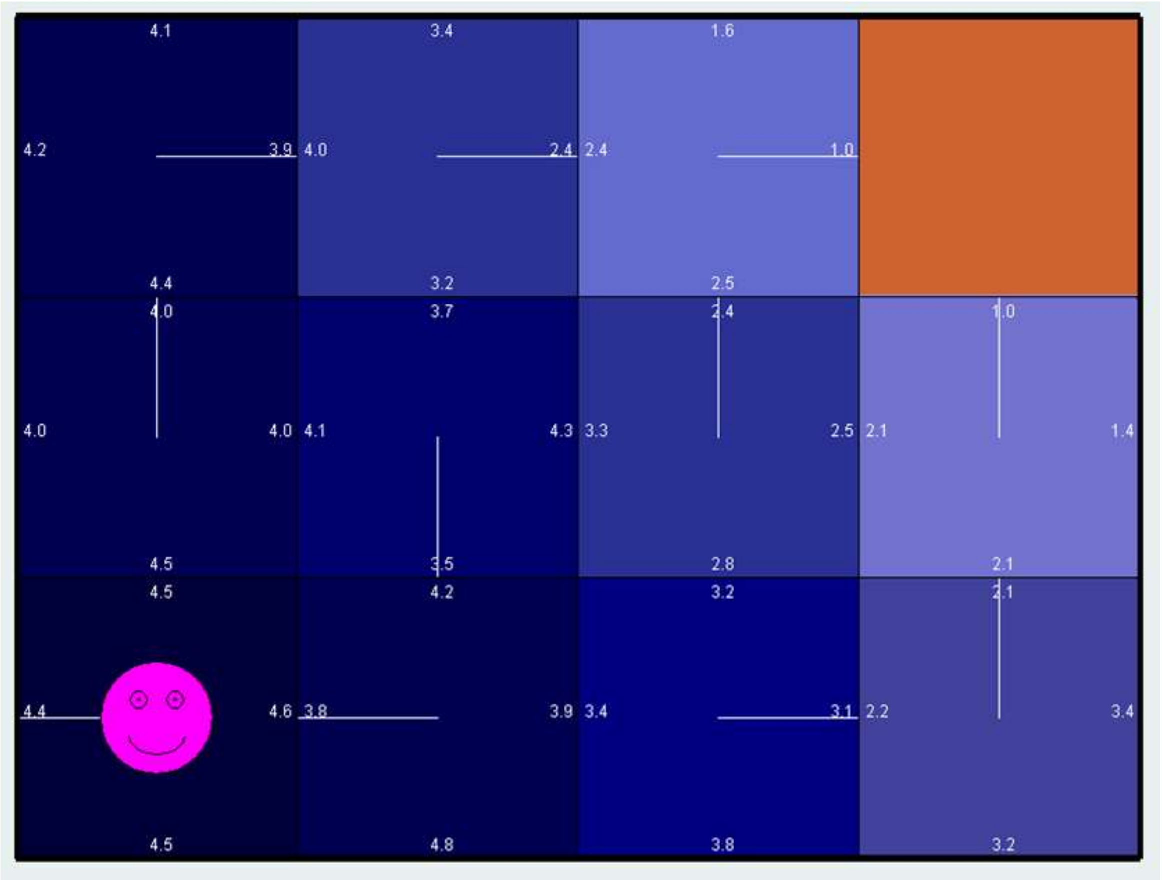
$$|S| : \#\text{ of states, } |A| : \#\text{ of actions}$$

Tubular Q-learning



http://www.cs.cmu.edu/~awm/rlsim/

Tubular Q-learning



This is similar to HW3 P1.

Tubular Q-learning

- Disadvantage of tubular Q-learning
  1. Storage space: |S|X|A|
  2. Exploration time
  3. Convergence time
- If the state space and action space is larger, the Q function is difficult to converge.
- To solve these issues, function-based Q learning was proposed. The Q function can be approximated by functions.

Function-based Q learning

- To improve tubular Q learning, function approximation was adopted.
- There are two kinds of approximations
- Parametric approximation:
  - The parametric approximators are mappings from a parameter space to the space Q functions. For example, polynomial basis (feature) is a popular approach.
- Nonparametric approximation:
  - The non parametric approximators are derived from the available data. For example, Gaussian processes (GP) and neural network (NN) are popular after 2005.

Function-based Q learning

- Parametric approximation:
- The Q function is approximated by some features.

$$Q(s,a) = \sum_i w_i f_i(s,a)$$

w : weighting vector

f : features

- How to find w? → gradient descent
- How to find f? → hand crafting

Function-based Q learning

- There are two ways to compute W:
- Offline approach: least square (pseudo inverse)

$$\min\left[Q(s,a)-Q(s,a)\right]^2=\min\left[(Y-WF)^T(Y-WF)\right]$$

$$W=\left(F^T F\right)^{-1} F^T Y$$

- Online approach: gradient descent (online least-squares)

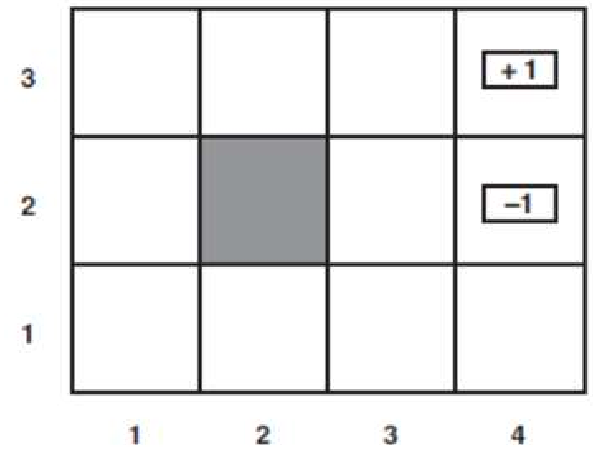
$$E=\frac{1}{2}\left[(Y-WF)^T(Y-WF)\right] \Rightarrow w_i \leftarrow w_i-\alpha\frac{\partial E}{\partial w_i}$$

$$w_i \leftarrow w_i+\alpha\left[R(s)+\gamma\max_{a'}Q(s',a')-Q(s,a)\right]\frac{\partial Q(s,a)}{\partial w_i}$$

Function-based Q learning

- Online approach: gradient descent (online least-squares)

$$w_i \leftarrow w_i + \alpha\left[R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]\frac{\partial Q(s,a)}{\partial w_i}$$



$EX:$

$$Q(s,a) = w_0 + w_1 x + w_2 y$$

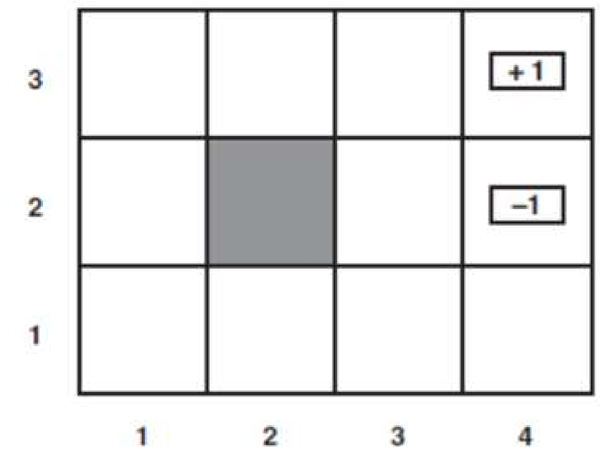
$$w_0 \leftarrow w_0 + \alpha\left[R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

$$w_1 \leftarrow w_1 + \alpha\left[R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]x$$

$$w_2 \leftarrow w_2 + \alpha\left[R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]y$$

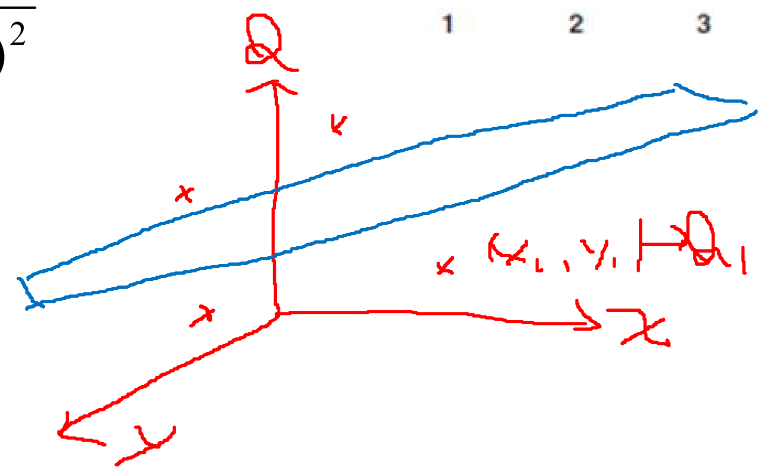
Function-based Q learning

- How to find good features?
  - The distance between the robot and the goal (4,3)
  - Polynomial terms of x and y



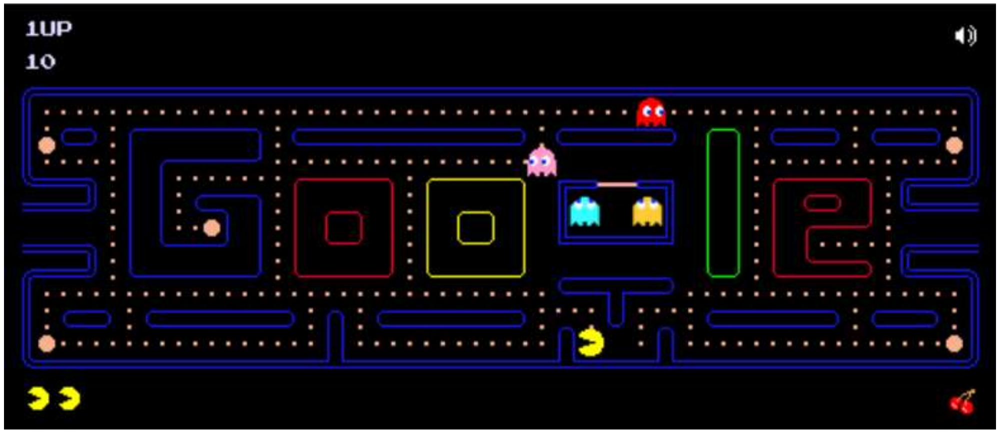
$$Q(s,a) = \sum_i w_i f_i(s,a)$$

$$= w_0 + w_1 x + w_2 y + w_3 \sqrt{(x - x_g)^2 + (y - y_g)^2}$$



Function-based Q learning

- How to find good features for Pac-Man?
  - Distance to closest ghost
  - Distance to closest dot
  - Number of ghosts
  - 1 / (dist to dot)^2
  - …etc.



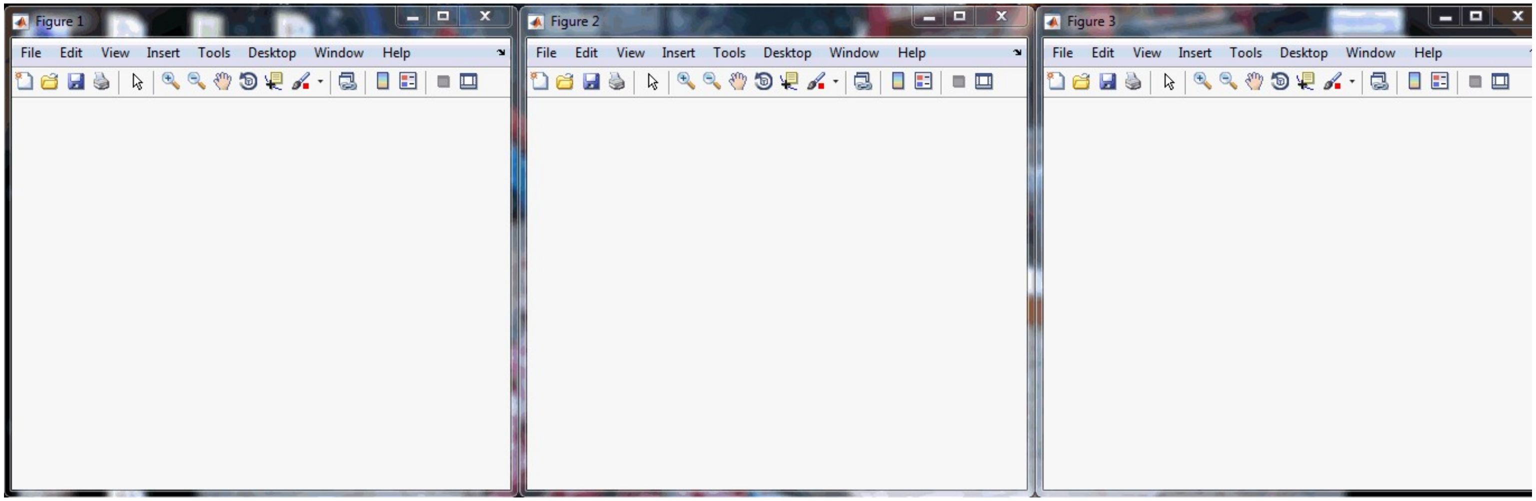
$$Q(s,a) = w_0 + w_1\sqrt{(x-x_g)^2 + (y-y_g)^2}$$
$$+ w_2\sqrt{(x-x_d)^2 + (y-y_d)^2}$$
$$+ w_3\left(1/(x-x_d)^2 + (y-y_d)^2\right)$$

Function-based Q learning

- Q-learning using 5 features

$$Q(s,a) = \sum w_i f_i$$

- Basis functions
  - Constant
  - Distance
  - Delta_ theta
  - UPPER
  - Turn



Function-based Q learning

- Function-based Q learning works but humans need to find features (hand crafting).
- Finding good features is the key to function-based Q learning.
- As aforementioned examples, we need domain knowledge to find good features. This processing is called feature engineering.
- The features worked well for A problem could not work for B problem.
- Hence, researchers also adopted nonparametric approximation (GP and NN) for Q learning.

Function-based Q learning

- Q learning (Tabular v.s. function approximation)

$$1.\ \text{run } a = \max_{a} Q(s,a)$$

$$2.\ \text{get reward } \mathrm{R(s,a)}$$

$$3.\ Q(s,a) \leftarrow Q(s,a) + \alpha\left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

☹ Storage space
☹ Exploration time
☹ Convergence time

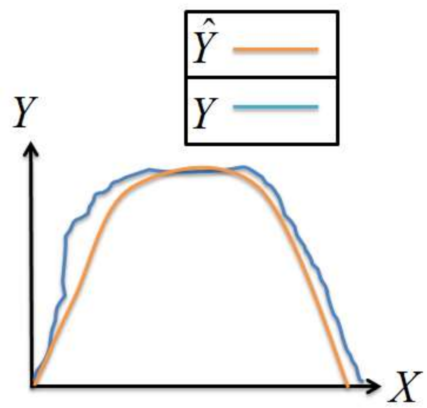
$$Q(s,a) = \sum w_i f_i$$

$$1.\ \text{run } a = \max_{a} Q(s,a)$$

$$2.\ \text{get reward } \mathrm{R(s,a)}$$

$$3.\ w_i \leftarrow w_i + \alpha\left[R(\mathrm{s,a}) + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] f_i$$

☺ Storage space ↓
☺ Exploration time ↓
☺ Convergence time ↓
☺ **ML techniques**
☹ **Find features?**



Function-based Q learning

- Function-based Q learning works but humans need to find features (hand crafting).
- Issues of RL:
  - Credit assignment issues
  - Exploration and exploitation

Linear regression

- The principles of the theory are derived, as are those of rational mechanics, from a very small number of primary facts.

  – Joseph Fourier, 1878

$$y = w_0 \bullet 1 + w_1 \bullet \cos(\omega_1 x) + w_2 \bullet \sin(\omega_2 x)$$

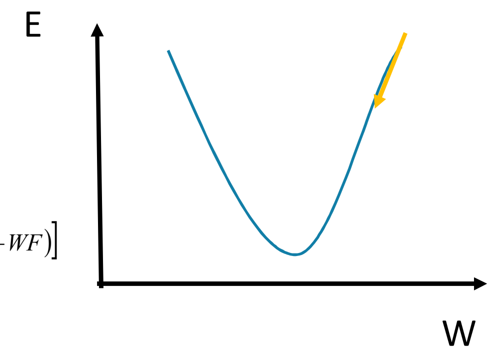
$$y = w_0 \bullet 1 + w_1 \bullet x + w_2 \bullet x^2 + \ldots\ldots$$

$$y = w_0 \bullet 1 + w_1 \bullet f_1(x) + w_2 \bullet f_2(x) + \ldots\ldots$$

- These functions are called “basis.” Linear regression is to find the weighting vector.

Linear regression

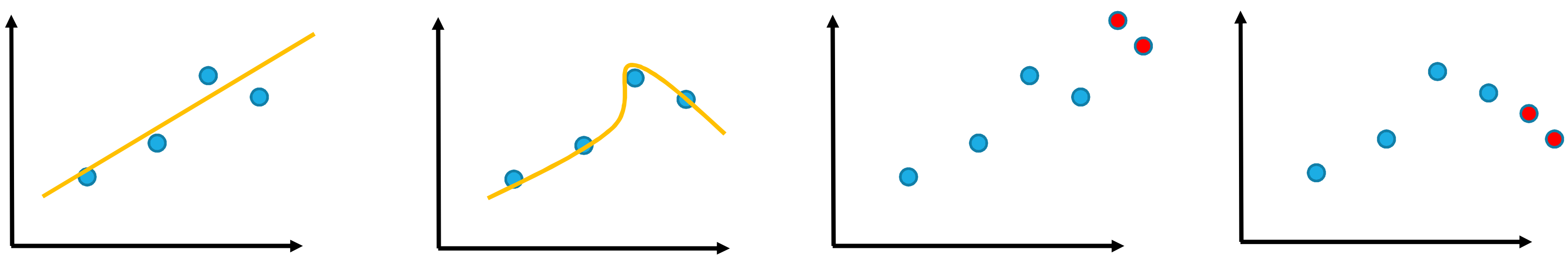
- A popular approach is to minimize the least square:



E= $$(y-y)^T(y-y)=(y-WF)^T(y-WF)$$

$$\min\left[Q(s,a)-Q(s,a)\right]^2=\min\left[(Y-WF)^T(Y-WF)\right]$$
$$W=\left(F^TF\right)^{-1}F^TY$$

- Offline approach: least square (pseudo inverse)
- Online approach: gradient descent (online least-squares)
- Another issue is overfitting. The regression try to fit the training data.



Regularization

- Regularization is a way to avoid overfitting issues.

$$\cos t(h) = Loss(h) + \lambda Complexity(h)$$

$$Complexity(h_w) = L_q(\mathbf{w}) = \sum_i |w_i|^q$$

- For example,

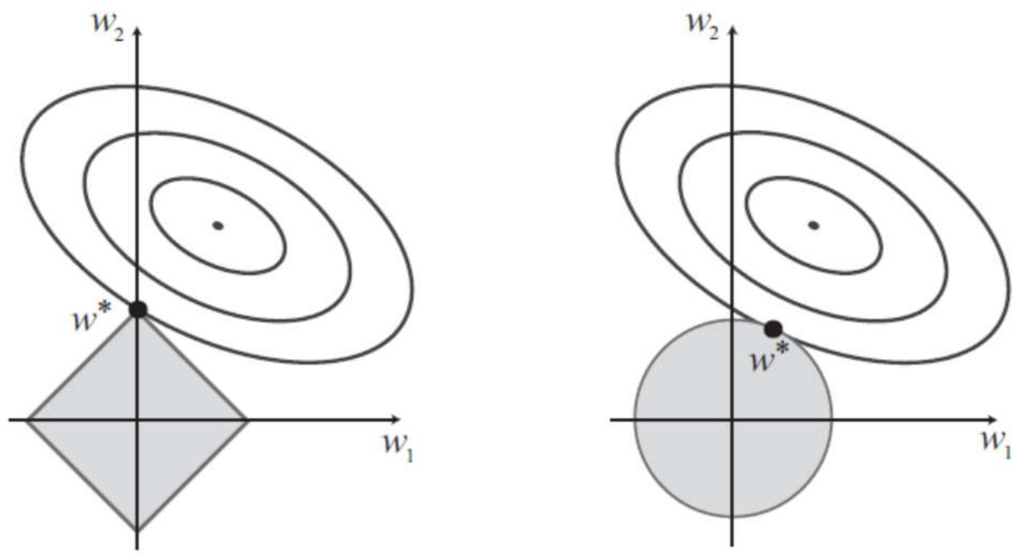
$$\min\left[(Y - WX)^2 + \lambda |W|^2\right] \ldots \text{L2 norm}$$

$$\min\left[(Y - WX)^2 + \lambda |W|\right] \ldots \text{L1 norm}$$

- These two minimization problems can be solved by gradient descent approaches.

Regularization

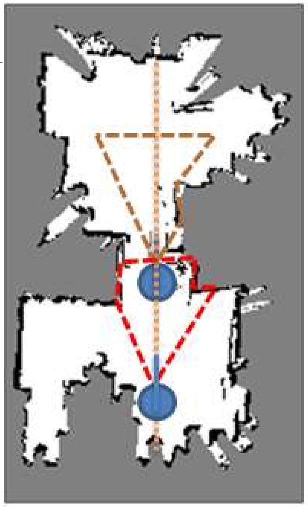
- L1 norm and L2 norm.
- L1 norm has the sparse property! L1 norm is applied to ***sparse learning,*** which is to learn the weighting and **enforce** elements of weighting vector are 0. (Feature selection!)



Regularization

- Coverage function approximation
- Given: (Xi,Yi), i=1~m, Xi={xi,yi}
- Find: W

robot1 robot2



X : Robot position, Y : Coverage

$$Y_{(m,1)} = X_{(m,n)} W_{(n,1)}$$

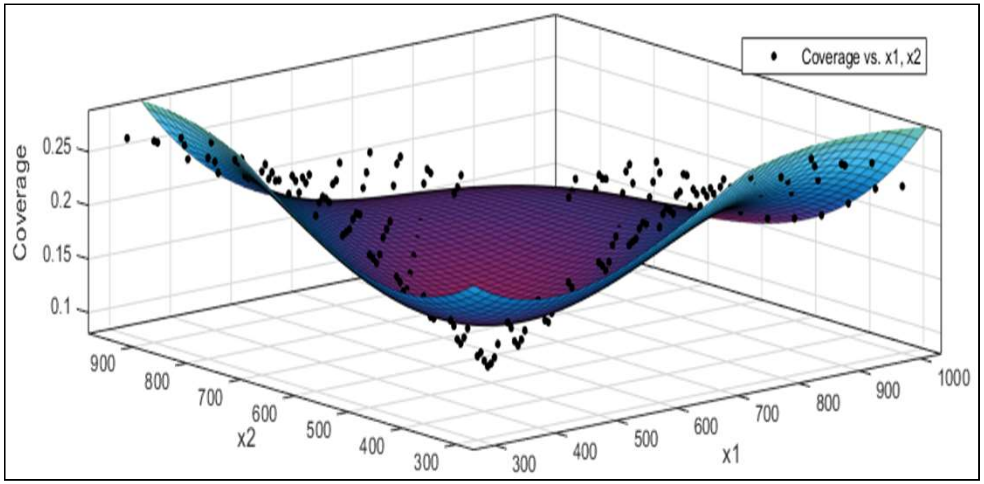
$$Y = w_0 1 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 xy + w_5 xy^2 + w_6 x^2 y + w_7 y + w_8 y^2 + w_9 y^3$$

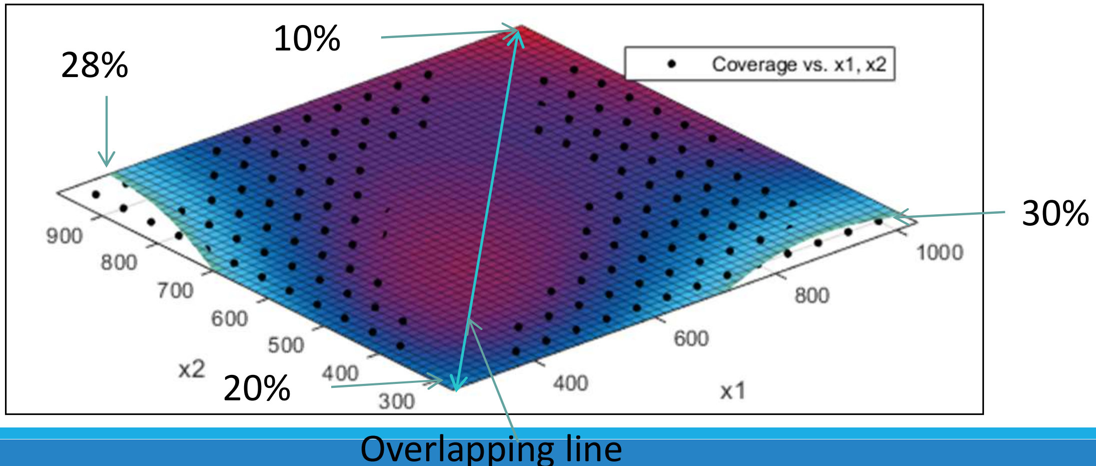
$$W = (X^T X)^{-1} X^T Y$$ **1. Pseudo inverse → overfitting?**

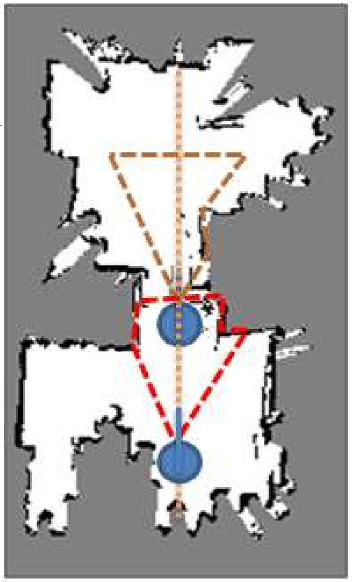
or

$$\mathbf{w}_{k+1} = \tau_{\lambda t}\left(\mathbf{w}_k - 2t\mathbf{X}^T(\mathbf{X}\mathbf{w}_k - \mathbf{Y})\right)$$ **2. ISTA (L1 norm)→ feature selection**
Iterative Shrinkage-Thresholding algorithm

Regularization

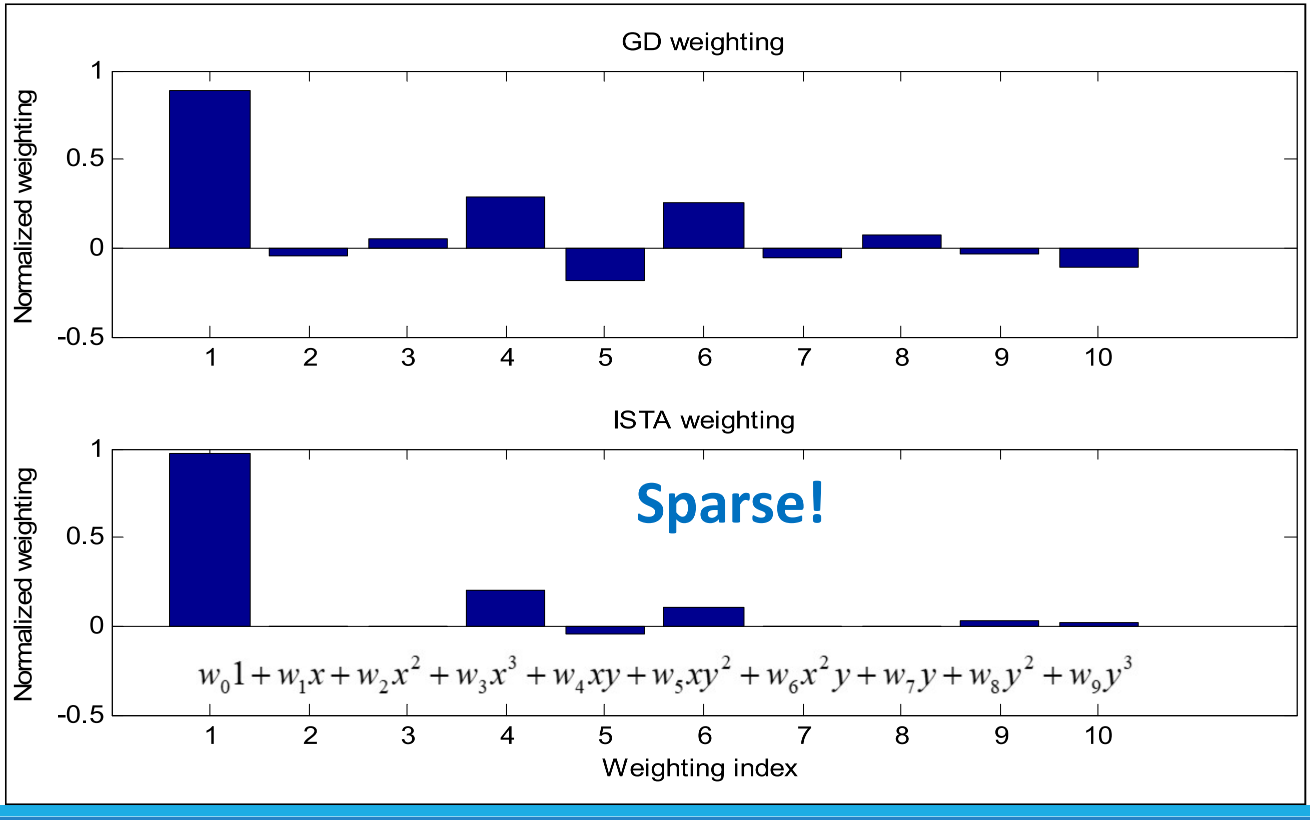






Regularization

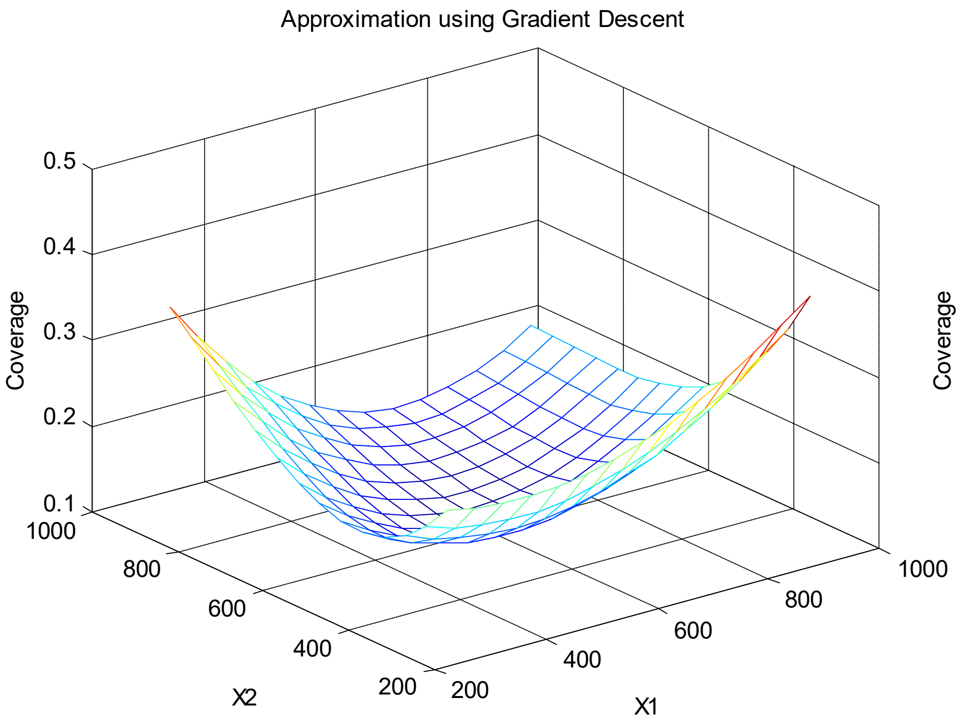
- Results:

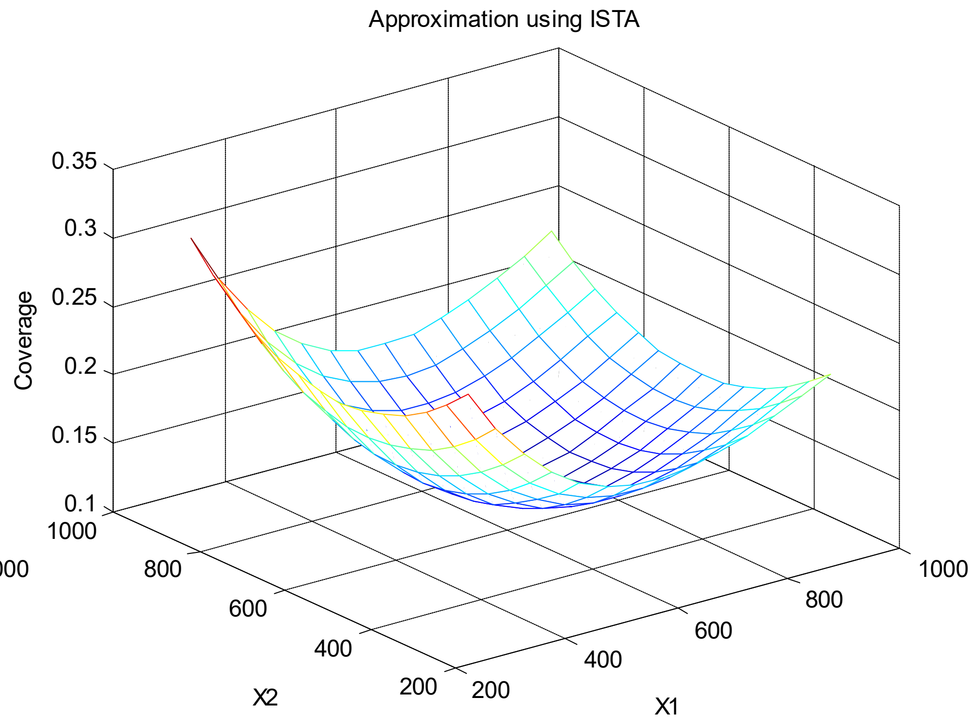


$$\lambda = 0.0005$$

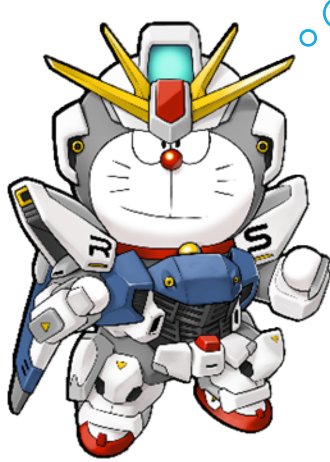
Regularization

- Results:





Q&A



Appendix – More about RL

- On-policy (sarsa) v.s. off-policy
- Policy gradient
- Actor-critic

Appendix

- "You don't learn to walk by following rules. You learn by doing, and by falling over."
  — Richard Branson, Entrepreneur