CS 687
Jana Kosecka

Reinforcement Learning

Continuous State MDP's

Value Function approximation

Markov Decision Process - Review

- Formal definition
- 4-tuple (S, A, T, R)
- Set of states S - finite
- Set of actions A - finite $T : S \times A \times S \rightarrow [0,1]$
- Transition model $S \times A \times S \rightarrow R$

  Transition probability for each action, state
- Reward model $S \rightarrow R$

- Goal find optimal value function
- Goal find an optimal policy – find policy which is maximizing the expected reward to go

Value iteration - Review

- Compute the optimal value function first, then the policy
- N states – N Bellman equations, start with initial values, iteratively update until you reach equilibrium

1. Initialize V; $U(s) = 0$
2. For each state x

$$U'(s) = R(s) + \gamma \max_a \sum_{x'} T(s,a,s')U(s')$$

3. If $\left| U'(s) - U(s) \right| > \delta$ then $\delta \leftarrow \left| U(s') - U(s) \right|$
4. until

$$\delta < \varepsilon(1-\gamma)/\gamma$$

- Optimal policy can be obtained before convergence of value iteration

Policy Iteration - Review

- Alternative Algorithm for finding optimal policies
- Takes policy and computes its value
- Iteratively improved policy, until it cannot be further improved

  1. Policy evaluation – calculate the utility of each state under particular policy π_i

  2. Policy improvement – Calculate new MEU policy, using one-step look-ahead based on π_{i+1}

  1. Initialize policy

  2. Evaluate policy get V; For each state do if

$$\max_a \sum_{x'} T(s,u,s')U(s') > \sum_{s'} T(s,\pi(s),s')U(s')$$

$$\pi(s) \leftarrow \arg\max_a \sum_{x'} T(s,u,s')U(s')$$

- Until unchanged

Policy iteration-Review

- For fixed policy – value function can be solved, by solving system of linear equations

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} T(s,a,s')U(s')$$

- No max operation – linear set of equations – unknowns are the values of value function at individual states (11 variables – 11 constraints)

$$U(1,1) = -0.04 + 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1)$$

Value iteration

- Compute the optimal value function first, then the policy
- N states – N Bellman equations, start with initial values, iteratively update until you reach equilibrium
- 1. Initialize V; For each state x

$$U_n(x) \leftarrow R(x) + \gamma \max_a \sum_{x'} T(x,a,x') U_{n-1}(x')$$

Bellman update/backup

$$\left| U_n(x) - U_{n-1}(x) \right| > \delta \qquad \delta < \varepsilon(1-\gamma)/\gamma$$

$$\delta \leftarrow \left| U_n(x) - U_{n-1}(x) \right|$$

- Optimal policy can be obtained before convergence of value iteration

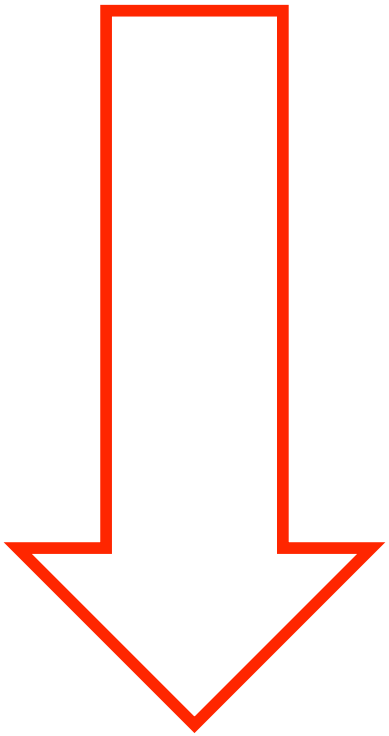
Continuous State MDP's

- Reinforcement learning for robotics
- Continuous State MDP's
- E.g. car control 6-dim space of position and velocities
- Helicopter 12-dim space pose and velocities

- How to find an optimal policy:
- Idea: discretize the state space and use standard algorithm
- Vertices are discrete states
- Reduces actions to a finite set
- Transition function ?

Discretization

- Original MDP (S, A, T, R, γ, H)

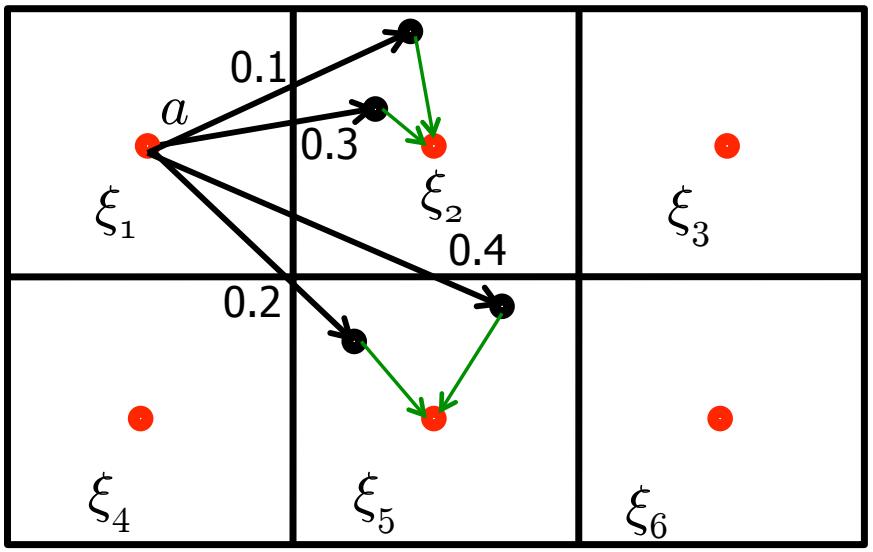


  - Grid the state-space: the vertices are the discrete states.
  - Reduce the action space to a finite set.
    - Sometimes not needed:
      - When Bellman back-up can be computed exactly over the continuous action space
      - When we know only certain controls are part of the optimal policy (e.g., when we know the problem has a "bang-bang" optimal solution)
  - Transition function: see next few slides.

- Discretized MDP $(\bar{S}, \bar{A}, \bar{T}, \bar{R}, \gamma, H)$

Slides P. Abeel, UC Berkeley, CS 287

Discretization: Deterministic Transition onto nearest vertex



Discrete states: { ξ_1 , …, ξ_6 }

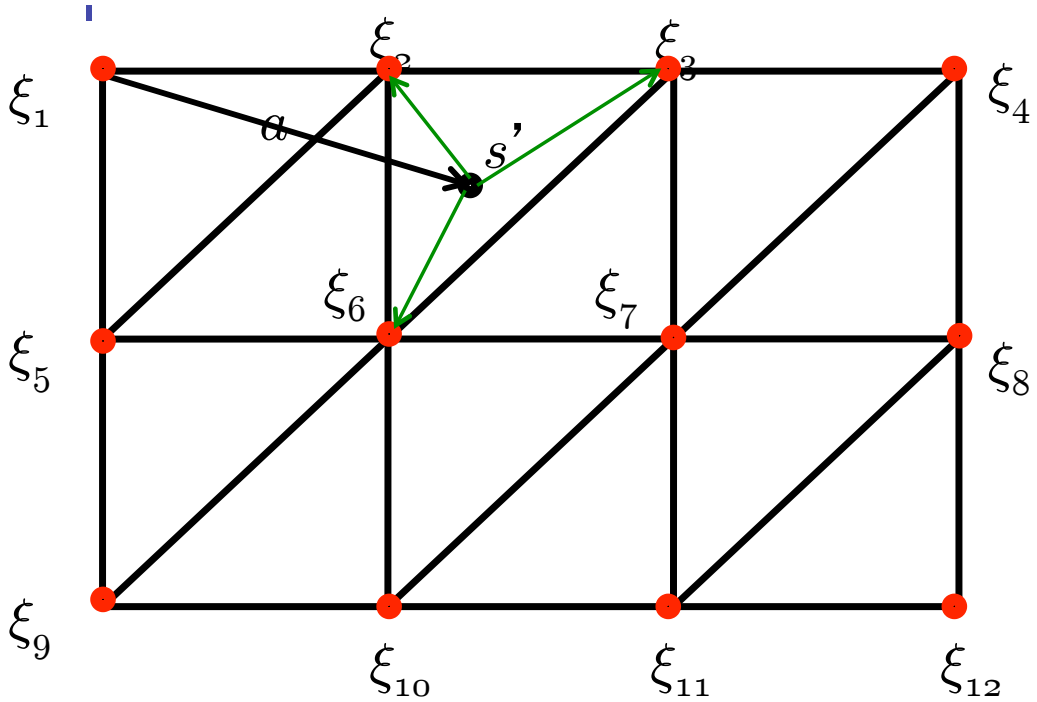
$$P(\xi_2|\xi_1, a) = 0.1 + 0.3 = 0.4;$$
$$P(\xi_5|\xi_1, a) = 0.4 + 0.2 = 0.6$$

Similarly define transition probabilities for all ξ_i

- → Discrete MDP just over the states {ξ_1 , …, ξ_6 }, which we can solve with value iteration
- If a (state, action) pair can results in infinitely many (or very many) different next states: Sample next states from the next-state distribution

Slides P. Abeel, UC Berkeley, CS 287

Discretization: Stochastic Approach



Discrete states: { ξ_1 , …, ξ_{12} }

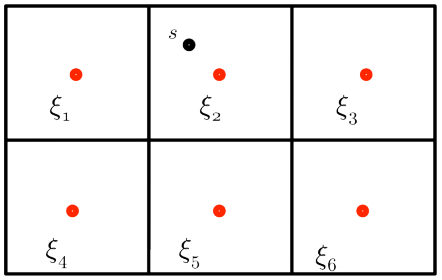
$$P(\xi_2|\xi_1, a) = p_A;$$
$$P(\xi_3|\xi_1, a) = p_B;$$
$$P(\xi_6|\xi_1, a) = p_C;$$
$$\text{s.t. } s' = p_A\xi_2 + p_B\xi_3 + p_C\xi_6$$

- If stochastic: Repeat procedure to account for all possible transitions and weight accordingly
- Need not be triangular, but could use other ways to select neighbors that contribute. "Kuhn triangulation" is particular choice that allows for efficient computation of the weights p_A, p_B, p_C, also in higher dimensions

Slides P. Abeel, UC Berkeley, CS 287

Discretization: How to act 0-step lookahead

- **Nearest Neighbor:** $\pi(s) = \pi(\xi_i) \quad \text{for} \quad \xi_i = \arg \min_{\xi \in \{\xi_1, \ldots, \xi_N\}} \|s - \xi\|$

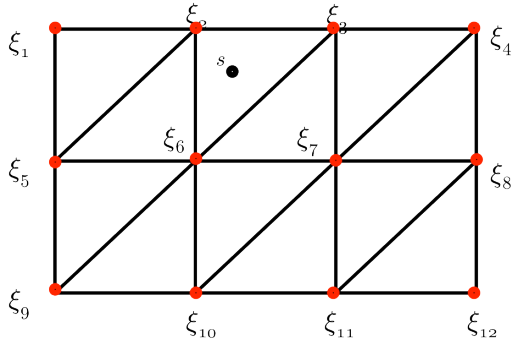


E.g., $\pi(s) = \pi(\xi_2)$

- **(Stochastic) Interpolation:** Find $p_1, \ldots, p_N$ s.t. $s = \sum_{i=1}^{N} p_i \xi_i$

Policy at s: choose $\pi(\xi_i)$ with probability p_i.

If continuous action space, can interpolate actions and choose $\sum_{i=1}^{N} p_i \pi(\xi_i)$



E.g., let p_2, p_3, p_6 be such that
$s = p_2\xi_2 + p_3\xi_3 + p_6\xi_6$
then choose $\pi(\xi_2)$, $\pi(\xi_3)$, $\pi(\xi_6)$
with probabilities p_2, p_3, p_6 respectively.

Slides P. Abeel, UC Berkeley, CS 287

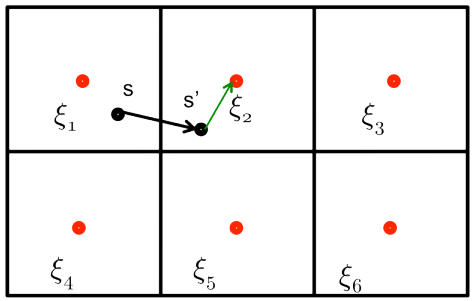
Discretization: 1-step lookahead

- **Use value function found for discrete MDP**

$$\pi(s) = \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \sum_i P(\xi_i; s') V(\xi_i) \right)$$

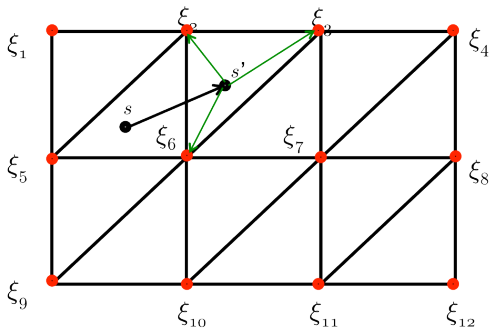
  - **Nearest Neighbor:**

$$P(\xi_i; s') = \begin{cases} 1 & \text{if } \xi_i = \arg\min_{\xi \in \{\xi_1, \ldots, \xi_N\}} \|s - \xi\| \\ 0 & \text{otherwise} \end{cases}$$



  - **(Stochastic) Interpolation:**

$$P(\xi_i; s') \text{ such that } \quad s' = \sum_{i=1}^{N} P(\xi_i; s') \xi_i$$



Slides P. Abeel, UC Berkeley, CS 287

Value Iteration with function approximation

Provides alternative derivation and interpretation of the discretization methods we have covered in this set of slides:

- Start with $V_0^*(s) = 0$ for all s.
- For i=0, 1, … , H-1

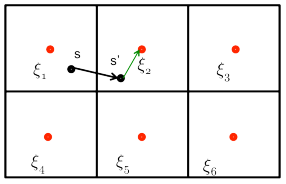
  for all states $s \in \bar{S}$ where $\bar{S}$ is the discrete state set

$$V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \hat{V}_i^*(s') \right]$$

where $$\hat{V}_i^*(s') = \sum_j P(\xi_j; s') V_i^*(\xi_j)$$

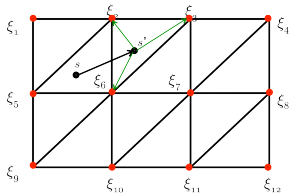
0'th Order Function Approximation

$$P(\xi_i; s') = \begin{cases} 1 & \text{if } \xi_i = \arg\min_{\xi \in \{\xi_1, \ldots, \xi_N\}} \|s - \xi\| \\ 0 & \text{otherwise} \end{cases}$$



1st Order Function Approximation

$$P(\xi_i; s') \text{ such that } \quad s' = \sum_{i=1}^{N} P(\xi_i; s') \xi_i$$



Slides P. Abeel, UC Berkeley, CS 287

Discretization as function approximation

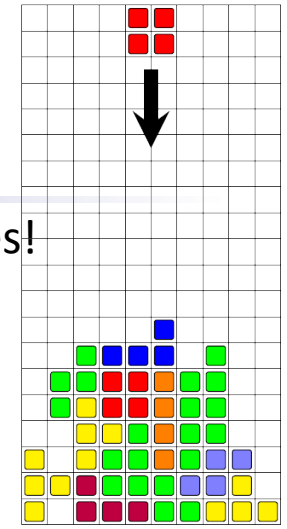
- 0^{th} order Grid based discretization: builds piecewise constant approximation

- 1^{st} order approximation – builds piecewise linear approximation of value function

Continuous State MDP's

- Reinforcement learning for robotics
- Continuous State MDP's
- E.g. car control 6-dim space of position and velocities
- Helicopter 12-dim space pose and velocities
- How to find an optimal policy:
- Idea: discretize the state space and use standard algorithm (curse of dimensionality), approximation of value function (piecewise constant vs linear example)
- Idea: Approximate V directly
- Example: car: continuous state (6-dim for car) actions (2D), helicopter (12-dim state, 4D actions)
- Discretization: impractical for high-dim state spaces

Example Tetris



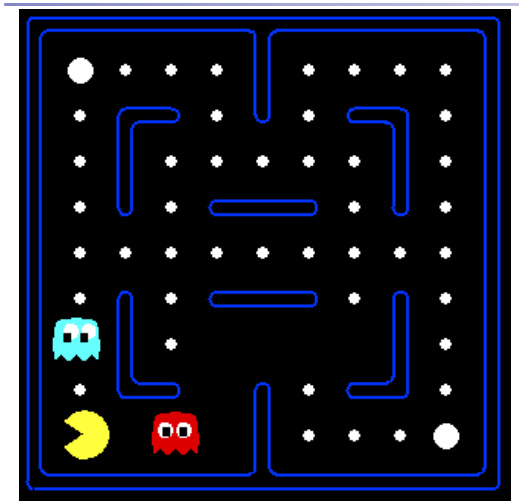
- state: board configuration + shape of the falling piece ~2^{200} stat
  es!
- action: rotation and translation applied to the falling piece
- 22 features aka basis functions ϕ_i
  - Ten basis functions, 0, . . . , 9, mapping the state to the height h[k] of each of the *ten* columns.
  - Nine basis functions, 10, . . . , 18, each mapping the state to the absolute *difference* between heights of successive columns: |h[k+1] − h[k]|, k = 1, . . . , 9.
  - One basis function, 19, that maps state to the maximum column height: $\max_k h[k]$
  - One basis function, 20, that maps state to the number of 'holes' in the board.
  - One basis function, 21, that is equal to 1 in every state.

$$\hat{V}_\theta(s) = \sum_{i=0}^{21} \theta_i \phi_i(s) = \theta^\top \phi(s)$$

[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis 1996 (TD); Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]

Slides P. Abeel, UC Berkeley, CS 287

Pacman



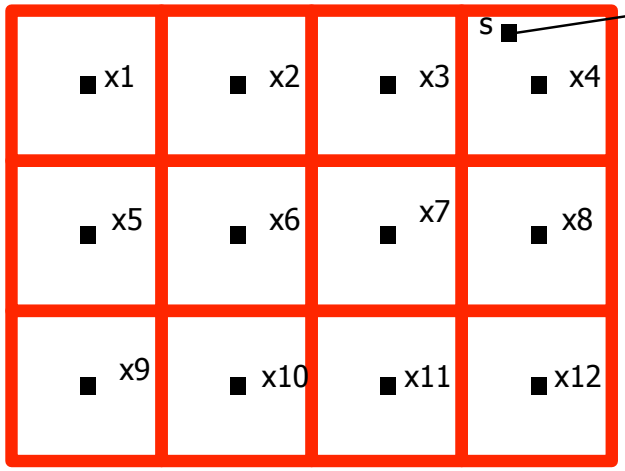
V(s) = θ_0 + θ_1 "distance to closest ghost"
+ θ_2 "distance to closest power pellet"
+ θ_3 "in dead-end"
+ θ_4 "closer to power pellet than ghost is"
+ …

= $\sum_{i=0}^{n} \theta_i \phi_i(s) = \theta^\top \phi(s)$

Slides P. Abeel, UC Berkeley, CS 287

0th order function approximation

- 0'th order approximation (1-nearest neighbor):



$$\hat{V}(s) = \hat{V}(x4) = \theta_4$$

$$\phi(s) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{pmatrix} \qquad \hat{V}(s) = \theta^\top \phi(s)$$

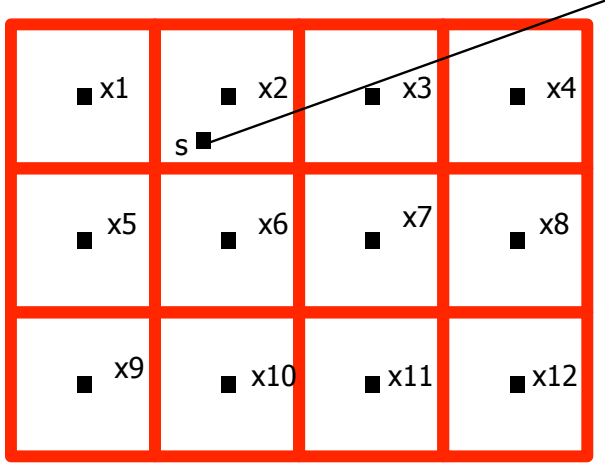
Only store values for x1, x2, …, x12

– call these values $\theta_1, \theta_2, \dots, \theta_{12}$

Assign other states value of nearest "x" state

1st order function approximation

- 1'th order approximation (k-nearest neighbor interpolation):



$$\hat{V}(s) = \phi_1(s)\theta_1 + \phi_2(s)\theta_2 + \phi_5(s)\theta_5 + \phi_6(s)\theta_6$$

$$\phi(s) = \begin{pmatrix} 0.2 \\ 0.6 \\ 0 \\ 0 \\ 0.05 \\ 0.15 \\ 0 \\ \dots \\ 0 \end{pmatrix} \qquad \hat{V}(s) = \theta^\top \phi(s)$$

Only store values for x1, x2, …, x12

– call these values $\theta_1, \theta_2, \ldots, \theta_{12}$

Assign other states interpolated value of nearest 4 "x" states

Function approximation

- Examples:
  - $S = \mathbb{R}, \quad \hat{V}(s) = \theta_1 + \theta_2 s$
  - $S = \mathbb{R}, \quad \hat{V}(s) = \theta_1 + \theta_2 s + \theta_3 s^2$
  - $S = \mathbb{R}, \quad \hat{V}(s) = \sum_{i=0}^{n} \theta_i s^i$
  - $S, \qquad \hat{V}(s) = \log\left(\frac{1}{1 + \exp(\theta^\top \phi(s))}\right)$

Function Approximation

- Main idea:
  - Use approximation $\hat{V}_\theta$ of the true value function V,
    - θ is a free parameter to be chosen from its domain Θ
    - Representation size: $|S|$ → downto: $|\Theta|$

    + : less parameters to estimate

    - : less expressiveness, typically there exist many V for which there is no θ such that $\hat{V}_\theta = V$

Functuin approximation – supervised learning

- Given:
  - set of examples

$$(s^{(1)}, V(s^{(1)}), (s^{(2)}, V(s^{(2)}), \ldots, (s^{(m)}, V(s^{(m)})$$

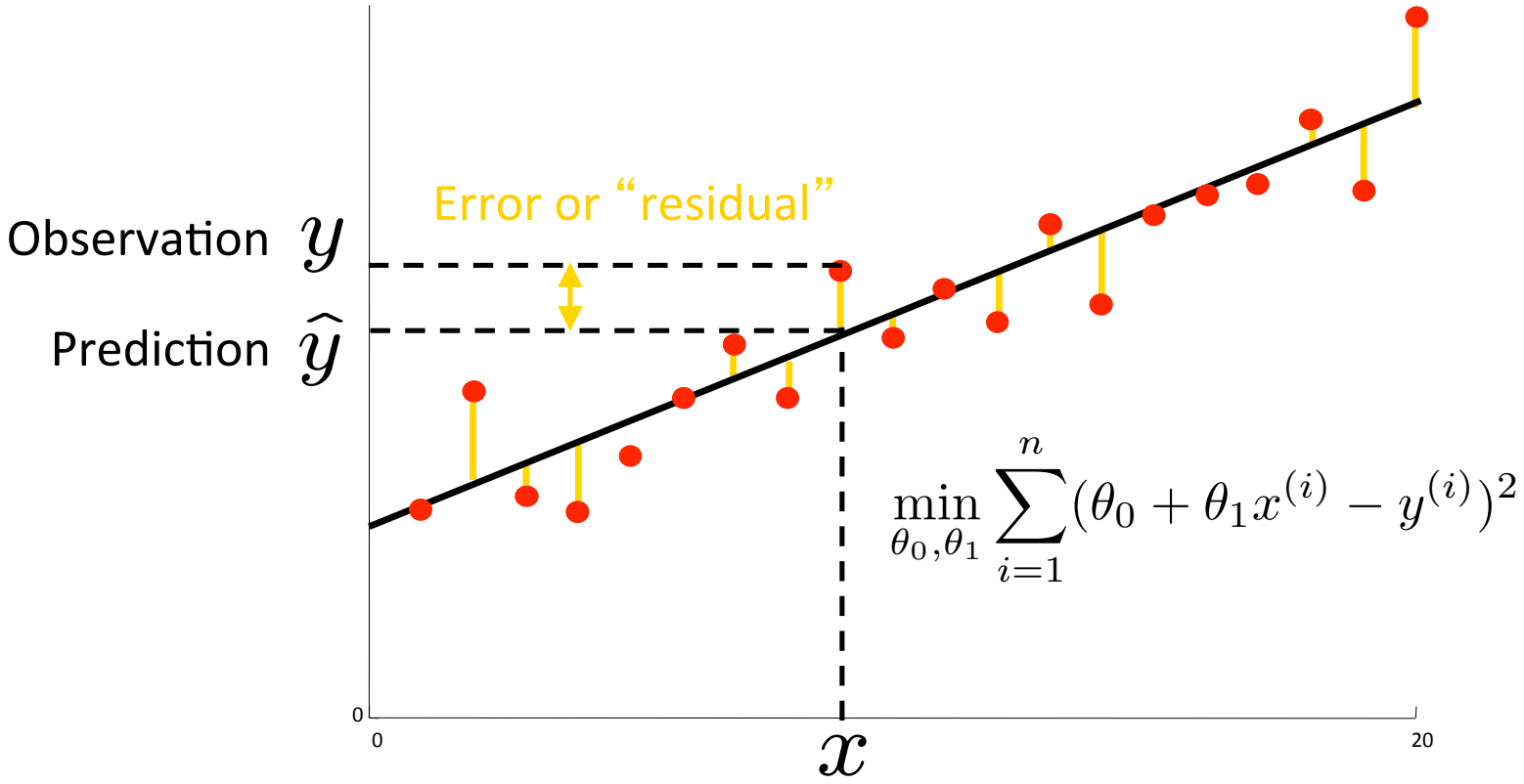
- Asked for:
  - "best" $\hat{V}_\theta$

- Representative approach: find θ through least squares:

$$\min_{\theta \in \Theta} \sum_{i=1}^{m} (\hat{V}_\theta(s^{(i)}) - V(s^{(i)}))^2$$

Supervised Learning Example

- Linear regression



- To avoid overfitting: reduce number of features used

- Practical approach: leave-out validation
  - Perform fitting for different choices of feature sets using just 70% of the data
  - Pick feature set that led to highest quality of fit on the remaining 30% of data

Value Iteration

- Pick some $S' \subseteq S$ (typically $|S'| << |S|$)
- Initialize by choosing some setting for $\theta^{(0)}$
- Iterate for i = 0, 1, 2, …, H:
  - Step 1: Bellman back-ups

$$\forall s \in S' : \quad \bar{V}_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \hat{V}_{\theta^{(i)}}(s') \right]$$

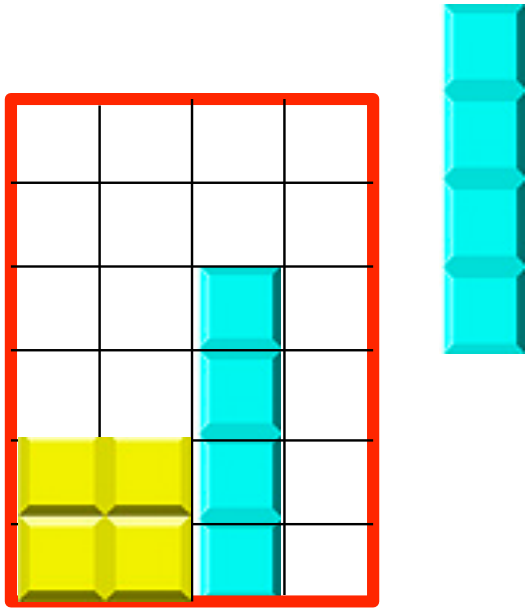
  - Step 2: Supervised learning

    find $\theta^{(i+1)}$ as the solution of:

$$\min_\theta \sum_{s \in S'} \left(\hat{V}_{\theta^{(i+1)}}(s) - \bar{V}_{i+1}(s) \right)^2$$

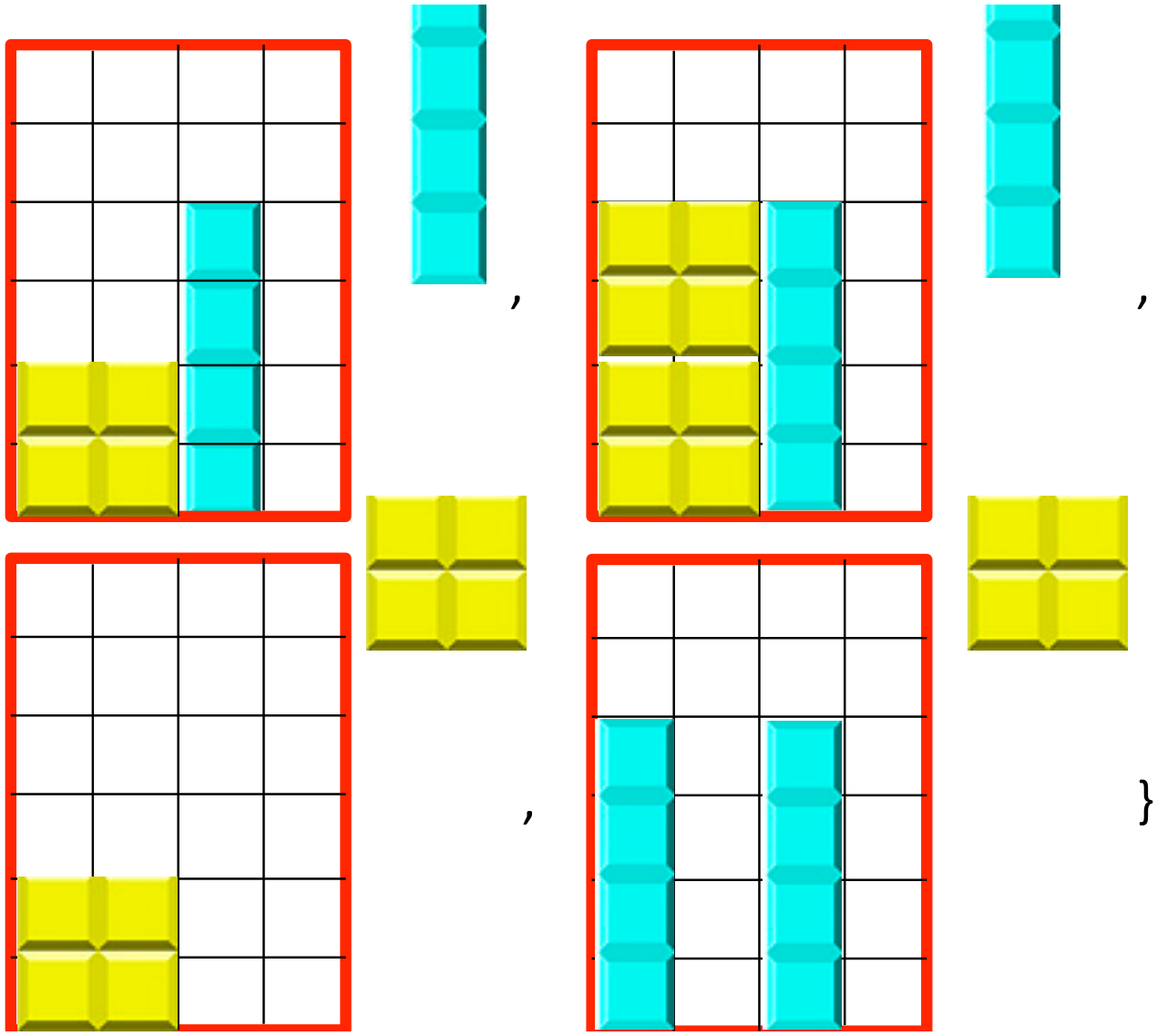
Mini Tetris example

- Mini-tetris: two types of blocks, can only choose translation (not rotation)
  - Example state:

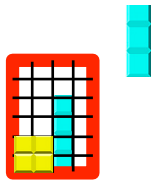
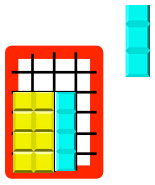
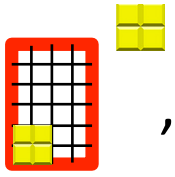
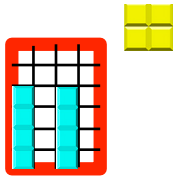


  - Reward = 1 for placing a block
  - Sink state / Game over is reached when block is placed such that part of it extends above the red rectangle
  - If you have a complete row, it gets cleared

Mini tetris

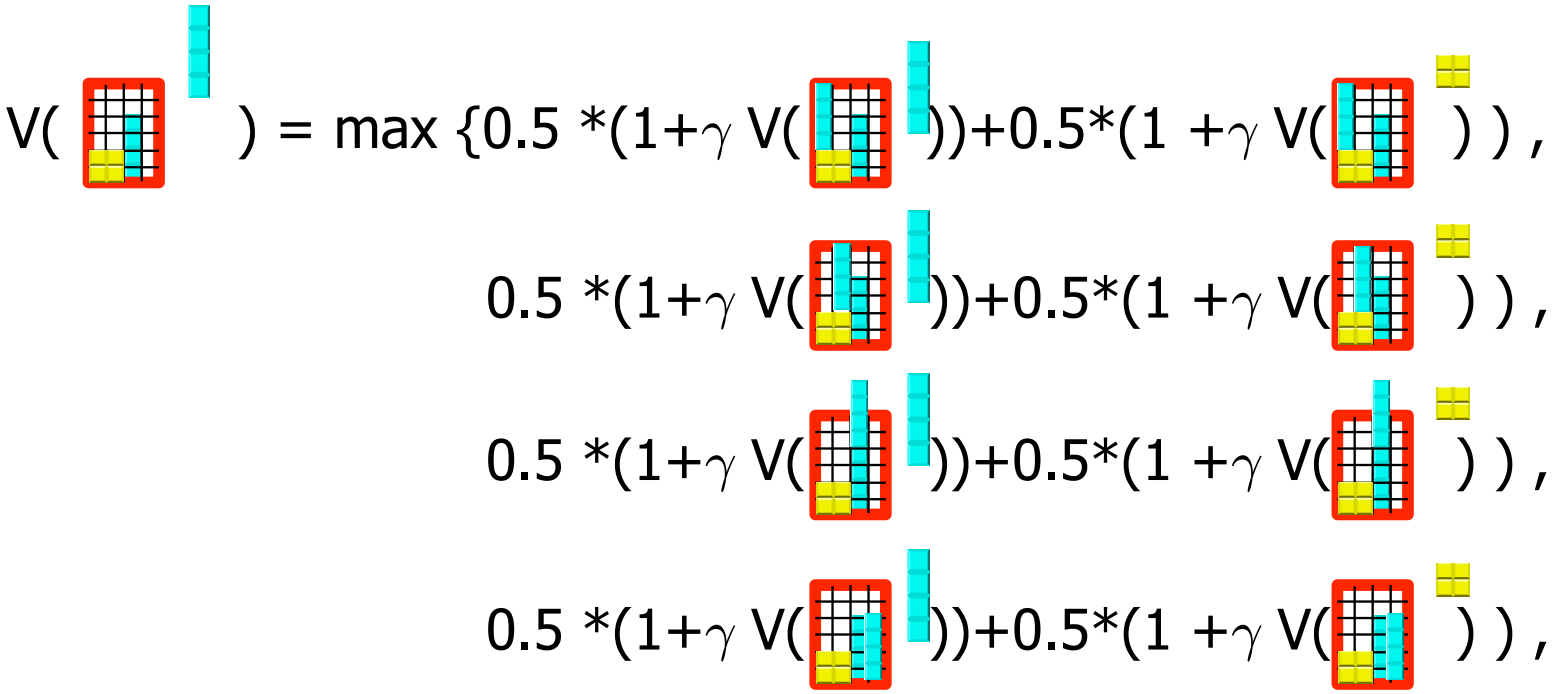
$S' = \{$  $\}$

Mini tetris

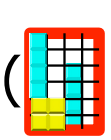
S' = {  ,  ,  ,  }

- 10 features aka basis functions $\acute{A}_i$
  - Four basis functions, 0*, . . . , 3, mapping the state to the height h[k] of each of the four* columns.
  - Three basis functions, 4*, . . . , 6, each mapping the state to the absolute difference* between heights of successive columns: $|h[k+1] - h[k]|$, $k = 1, \ldots, 3$.
  - One basis function, 7, that maps state to the maximum column height: $\max_k h[k]$
  - One basis function, 8, that maps state to the number of 'holes' in the board.
  - One basis function, 9, that is equal to 1 in every state.
- Init $\theta^{(0)}$ = (-1, -1, -1, -1, -2, -2, -2, -3, -2, 10)

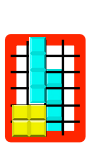
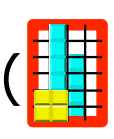
- Bellman back-ups for the states in S':



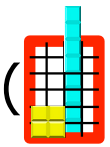
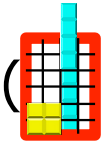
- Bellman back-ups for the states in S':

V()=max {0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

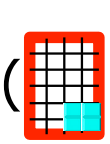
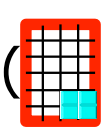
(6,2,4,0, 4, 2, 4, 6, 0, 1) (6,2,4,0, 4, 2, 4, 6, 0, 1)

0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

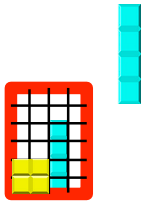
(2,6,4,0, 4, 2, 4, 6, 0, 1) (2,6,4,0, 4, 2, 4, 6, 0, 1)

0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

(sink-state, V=0) (sink-state, V=0)

0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$())}

(0,0,2,2, 0,2,0, 2, 0, 1) (0,0,2,2, 0,2,0, 2, 0, 1)

V()=max {0.5 *(1+γ -30)+0.5*(1 +γ -30),

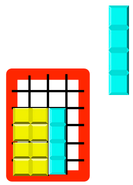
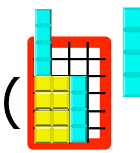
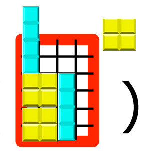
0.5 *(1+γ -30)+0.5*(1 +γ -30),

0.5 *(1+γ 0)+0.5*(1 +γ 0),

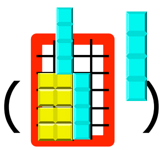
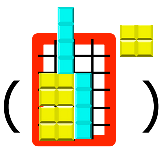
0.5 *(1+γ 6)+0.5*(1 +γ 6),

= 6.4 (for γ = 0.9)

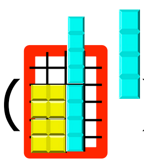
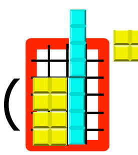
- Bellman back-ups for the second state in S':

V()=max {0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

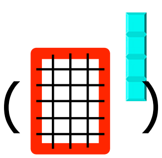
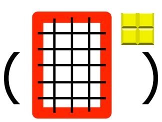
(sink-state, V=0) (sink-state, V=0)

0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

(sink-state, V=0) (sink-state, V=0)

0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

(sink-state, V=0) (sink-state, V=0)

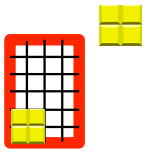
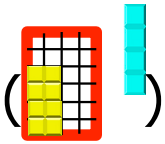
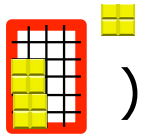
0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$())}

(0,0,0,0, 0,0,0, 0, 0, 1) -> V = 20

(0,0,0,0, 0,0,0, 0, 0, 1) -> V = 20

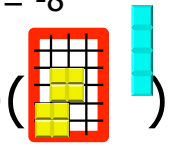
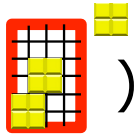
= 19

- Bellman back-ups for the third state in S':

V()=max {0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

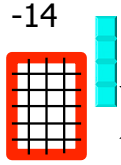
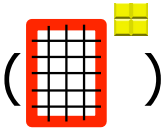
(4,4,0,0, 0,4,0, 4, 0, 1)
-> V = -8

(4,4,0,0, 0,4,0, 4, 0, 1)
-> V = -8

0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

(2,4,4,0, 2,0,4, 4, 0, 1)
-> V = -14

(2,4,4,0, 2,0,4, 4, 0, 1)
-> V = -14

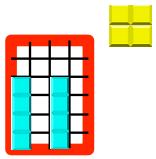
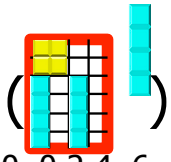
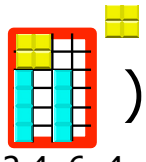
0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

(0,0,0,0, 0,0,0, 0, 0, 1)
-> V = 20

(0,0,0,0, 0,0,0, 0, 0, 1)
-> V = 20

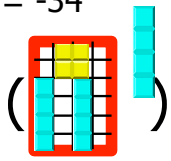
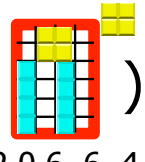
= 19

- Bellman back-ups for the fourth state in S':

V()=max {0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

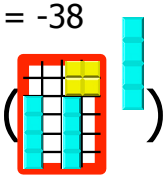
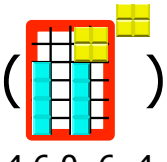
(6,6,4,0, 0,2,4, 6, 4, 1)
-> V = -34

(6,6,4,0, 0,2,4, 6, 4, 1)
-> V = -34

0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

(4,6,6,0, 2,0,6, 6, 4, 1)
-> V = -38

(4,6.6,0, 2,0,6, 6, 4, 1)
-> V = -38

0.5 *(1+$\gamma\,\theta^{\top}\phi$())+0.5*(1 +$\gamma\,\theta^{\top}\phi$()) ,

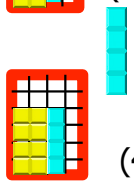
(4,0,6,6, 4,6,0, 6, 4, 1)
-> V = -42

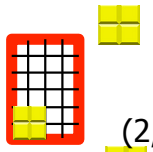
(4,0,6,6, 4,6,0, 6, 4, 1)
-> V = -42

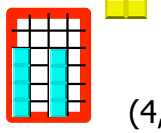
= -29.6

- After running the Bellman back-ups for all 4 states in S' we have:

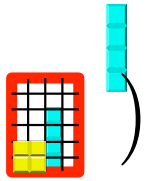
V()= 6.4
(2,2,4,0, 0,2,4, 4, 0, 1)

V()= 19
(4,4,4,0, 0,0,4, 4, 0, 1)

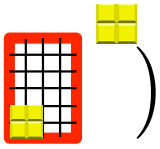
V()= 19
(2,2,0,0, 0,2,0, 2, 0, 1)

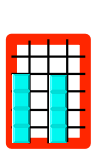
V()= -29.6
(4,0,4,0, 4,4,4, 4, 0, 1)

- We now run supervised learning on these 4 examples to find a new θ:

$$\min_{\theta}(6.4 - \theta^{\top}\phi(\ \ ))^2$$ 

$$+(19 - \theta^{\top}\phi(\ \ ))^2$$ 

$$+(19 - \theta^{\top}\phi(\ \ ))^2$$ 

$$+((-29.6) - \theta^{\top}\phi(\ \ ))^2$$ 

→ Running least squares gives new θ

$$\theta^{(1)} = (0.195, 6.24, -2.11, 0, -6.05, 0.13, -2.11, 2.13, 0, 1.59)$$

Learning a model for MDP

- Before state transtion probabilities and rewards known
- These are usually not given
- We can have a simulator and observed a set of trails
- Estimate T(s,a,s') as number of times we took actio a in state a we got to state s'/number of times we too action a in state s

Continuous State MDP

- To obtain a model – learn one
- Given a simulator – execute some random policy
- Record actions and states – learn a model of dynamics

$$s_{t+1} = As_t + Ba_t$$

- For linear model find such A and B to fit best the observed sequences, Get a deterministic model
- Stochastic model

$$s_{t+1} = As_t + Ba_t + \varepsilon_t$$

- Or you can use locally weighted linear regression (to learn a non-linear model)

Appoximate Value Function

- E.g. linear combination of features (some functions of state)
- Approximate value function as

$$V(s) = \Theta^T \varphi(s)$$

- Now how to adopt value iteration ?
- Idea – repeatedly fit the values of parameters of value function Θ

Fitted Value Iteration

- Sample set of states at random $s^{(1)}, s^{(2)}, \ldots, s^{(m)}$
- Initialize Θ

1. For each state

for each action { % sample set of k next states

given the model, compute the estimate of the V (rhs of Bellman)

$$q(a) = \frac{1}{k} \sum_{j=1}^{k} [R(s^i) + \gamma V(s'_j)]$$

} set

$$y^i = \max_a q(a)$$

$$V(s^i) \approx y^i$$

In the original value iteration

$$\Theta^i = \arg\min_\Theta \frac{1}{2} \sum_{i=1}^{n} (\Theta^T \varphi(s) - y^i)^2$$

} % Find the value of parameters as close as possible to the simulated values

Fitted Value Iteration

3. Repeat {

For $i = 1, \ldots, m$ {

For each action $a \in A$ {

Sample $s'_1, \ldots, s'_k \sim P_{s^{(i)}a}$ (using a model of the MDP).

Set $q(a) = \frac{1}{k} \sum_{j=1}^{k} R(s^{(i)}) + \gamma V(s'_j)$

// Hence, $q(a)$ is an estimate of $R(s^{(i)}) + \gamma \mathrm{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$.

}

Set $y^{(i)} = \max_a q(a)$.

// Hence, $y^{(i)}$ is an estimate of $R(s^{(i)}) + \gamma \max_a \mathrm{E}_{s' \sim P_{s^{(i)}a}}[V(s')]$.

}

// In the original value iteration algorithm (over discrete states)

// we updated the value function according to $V(s^{(i)}) := y^{(i)}$.

// In this algorithm, we want $V(s^{(i)}) \approx y^{(i)}$, which we'll achieve

// using supervised learning (linear regression).

Set $\theta := \arg\min_\theta \frac{1}{2} \sum_{i=1}^{m} \left(\theta^T \phi(s^{(i)}) - y^{(i)}\right)^2$

Fitted Value Iteration

- Converge to optimal value function
- Issues: how to choose the features, how to choose the policy
- You cannot pre-compute the policy for each state
- Only when you are in some state, select the policy

Variations of MDP's

- Finite horizon MDP's
- Action – State rewards
- Non-stationary MDP's

- LQR - Continuous state space, action space special form of reward function

Reinforcement

- Stanford Helicopter Project

- Learn complex maneuvers given some sample trajectories
- Standford Helicopter