Reinforcement Learning III

Dec 03 2008

Large State Spaces

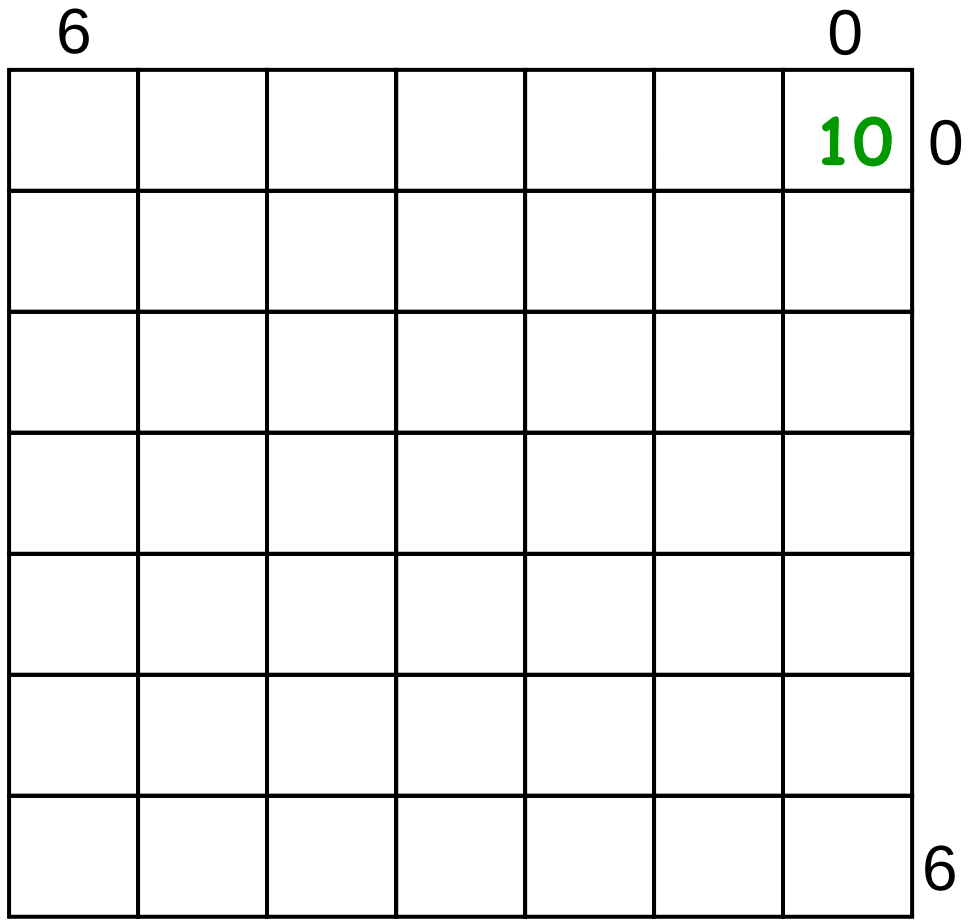
- When a problem has a large state space we can not longer represent the U or Q functions as explicit tables
- Even if we had enough memory
  - Never enough training data!
  - Learning takes too long
- What to do??

Function Approximation

- Never enough training data!
  - Must generalize what is learned from one situation to other **"similar"** new situations
- Idea:
  - Instead of using large table to represent U or Q, use a parameterized function
    - small number of parameters (generally exponentially fewer parameters than the number of states)
  - Learn parameters from experience
  - When we update parameters based on observations in one state, then the U or Q estimate will also change for other similar states
    - facilitates generalization of experience

Example

- Consider grid problem with no obstacles, deterministic actions U/D/L/R (49 states)
- Features for state s=(x,y): f1(s)=x, f2(s)=y (just 2 features)



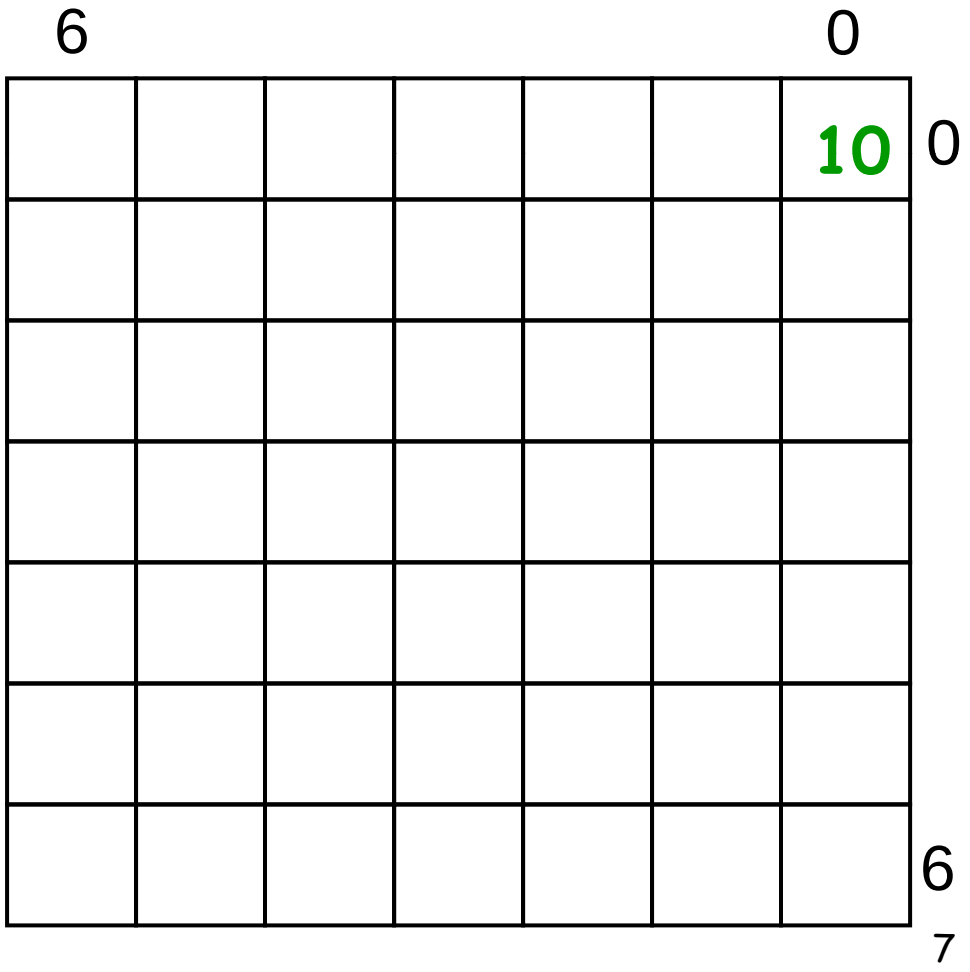
Linear Function Approximation

- Define a set of state features $f_1(s)$, …, $f_n(s)$
  - The features are used as our representation of states
  - States with similar feature values will be treated similarly
- A common approximation is to represent V*(s)* as a weighted sum of the features (i.e. a linear approximation)

$$U_\theta(s) = \theta_0 + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s)$$

Example

- Consider grid problem with no obstacles, deterministic actions U/D/L/R (49 states)
- Features for state s=(x,y): f1(s)=x, f2(s)=y (just 2 features)
- $U(s) = \theta_0 + \theta_1 x + \theta_2 y$
- Is there a good linear approximation?
  - Yes.
  - θ_0 =10, θ_1 = -1, θ_2 = -1
  - (note upper right is origin)
- U(s) = 10 - x - y
  subtracts Manhattan dist. from goal reward
- Instead of storing a table of 49 entries, we now only need to store 3 parameters

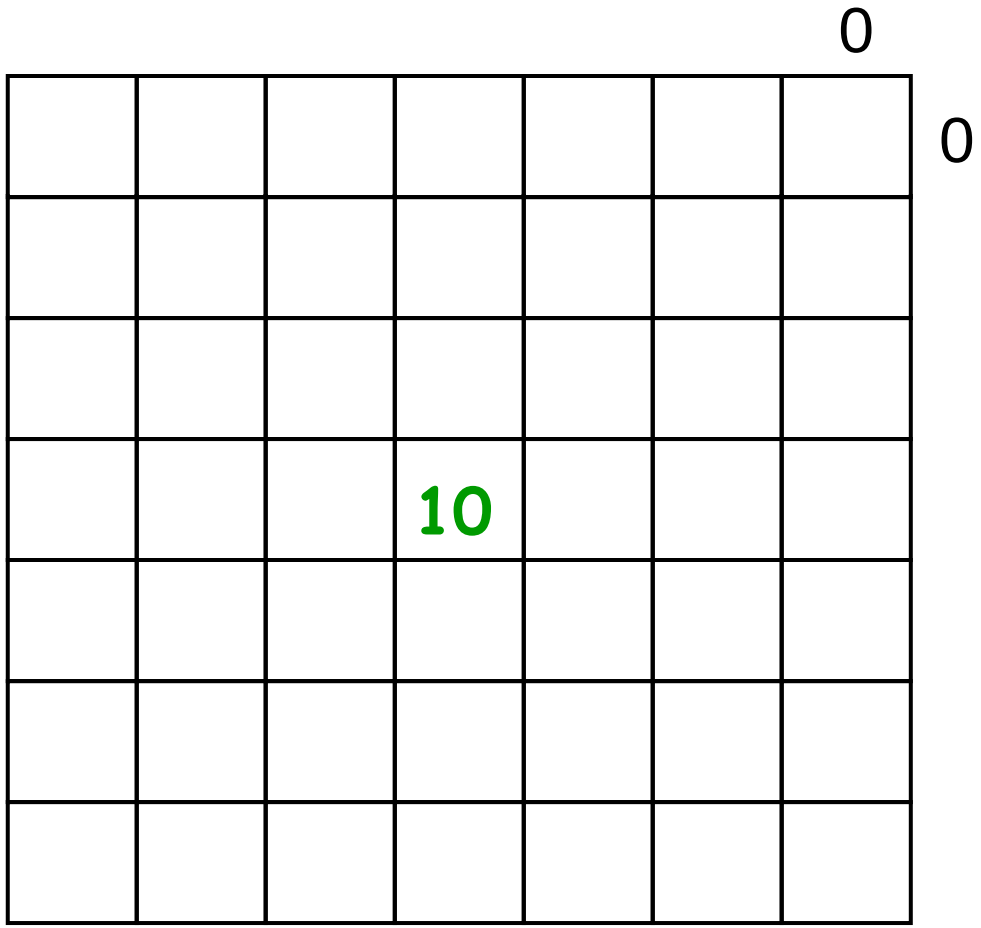


Function approximation accuracy

- The approximation accuracy is fundamentally limited by the information provided by the features
- Can we always define features that allow for a perfect linear approximation?
  - Yes. Assign each state an indicator feature. (I.e. i'th feature is 1 iff i'th state is present and θ_i represents value of i'th state)
  - Of course this requires far too many features and gives no generalization.

Changed Reward: Bad linear approximation

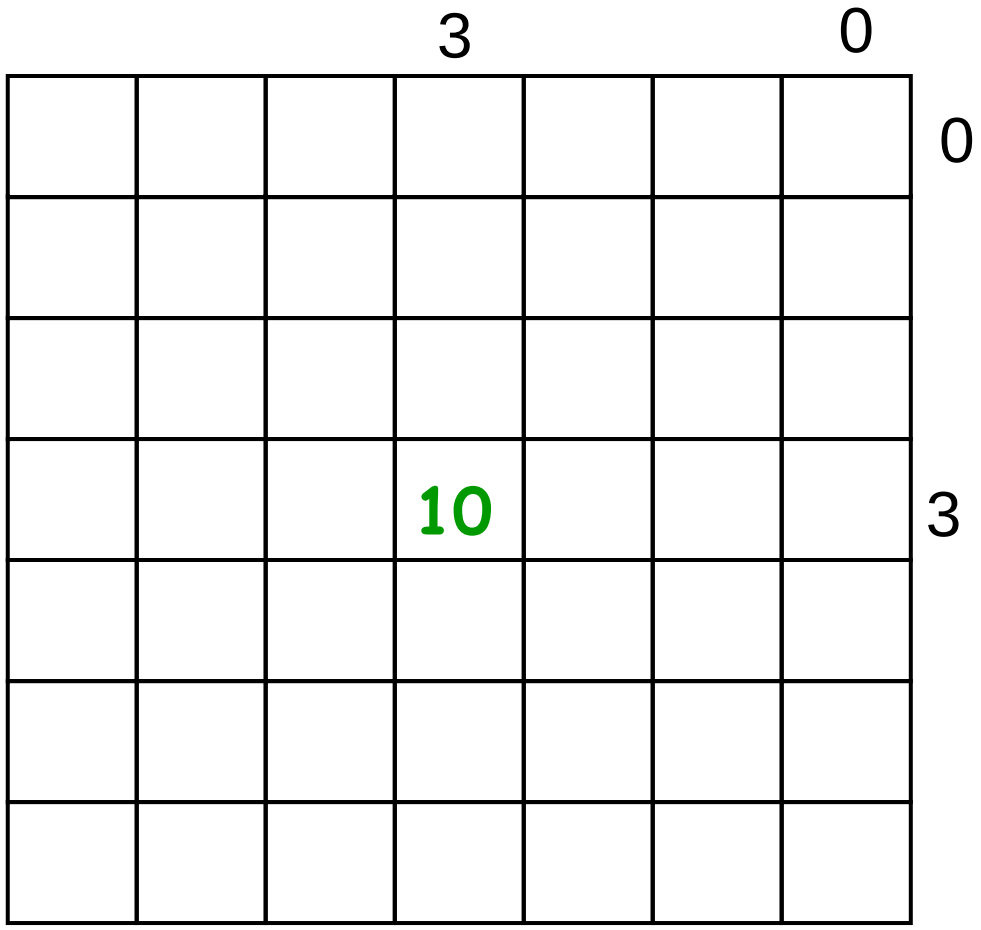
- $U(s) = \theta_0 + \theta_1 x + \theta_2 y$
- Is there a good linear approximation?
  - No.



But What If…

- $U(s) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 z$

- Include new feature z
  - z= |3-x| + |3-y|
  - z is dist. to goal location
- Does this allow a good linear approx?
  - θ_0 =10, $\theta_1 = \theta_2 = 0$, $\theta_0 = -1$



Linear Function Approximation

- Define a set of features f1(s), …, fn(s)
  - The features are used as our representation of states
  - States with similar feature values will be treated similarly
  - More complex functions require more complex features

$$U_\theta(s) = \theta_0 + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s)$$

- Our goal is to learn good parameter values (i.e. feature weights) that approximate the value function well
  - How can we do this?
  - Use TD-based RL and somehow update parameters based on each experience.

TD-based RL for Linear Approximators

1. Start with initial parameter values
2. Take action according to an explore/exploit policy (should converge to greedy policy, i.e. GLIE)
3. Update estimated model
4. Perform TD update for each parameter

$$\theta_i \leftarrow ?$$

5. Goto 2

What is a "TD update" for a parameter?

Aside: Gradient Descent for Squared Error

- Suppose that we have a sequence of states and target values for each state $\langle s_1, u(s_1)\rangle, \langle s_2, u(s_2)\rangle, \ldots$
  - E.g. produced by the TD-based RL loop
- Our goal is to minimize the sum of squared errors between our estimated function and each target value:

$$E_j = \frac{1}{2}\left(\hat{U}_\theta(s_j) - u(s_j)\right)^2$$

squared error of example j

our estimated value for j'th state

target value for j'th state

- After seeing j'th state **gradient descent rule** tells us to update all parameters by:

$$\theta_i \leftarrow \theta_i - \alpha \frac{\partial E_j}{\partial \theta_i}, \qquad \frac{\partial E_j}{\partial \theta_i} = \frac{\partial E_j}{\partial \hat{U}_\theta(s_j)} \frac{\partial \hat{U}_\theta(s_j)}{\partial \theta_i}$$

learning rate

Aside: continued

$$\theta_i \leftarrow \theta_i + \alpha \frac{\partial E_j}{\partial \theta_i} = \theta_i + \alpha \underbrace{\left(u(s_j) - \hat{U}_\theta(s_j)\right)}_{\frac{\partial E_j}{\partial \hat{U}_\theta(s_j)}} \frac{\partial \hat{U}_\theta(s_j)}{\partial \theta_i}$$

depends on form of approximator

- For a linear approximation function:

$$\hat{U}_\theta(s) = \theta_1 + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s)$$

$$\frac{\partial \hat{U}_\theta(s_j)}{\partial \theta_i} = f_i(s_j)$$

- Thus the update becomes: $\theta_i \leftarrow \theta_i + \alpha\left(u(s_j) - \hat{U}_\theta(s_j)\right) f_i(s_j)$
- For linear functions this update is guaranteed to converge to best approximation for suitable learning rate schedule

TD-based RL for Linear Approximators

1. Start with initial parameter values
2. Take action according to an explore/exploit policy (should converge to greedy policy, i.e. GLIE) Transition from s to s'
3. Update estimated model
4. Perform TD update for each parameter

$$\theta_i \leftarrow \theta_i + \alpha\left(u(s) - \hat{U}_\theta(s)\right) f_i(s)$$

5. Goto 2

What should we use for "target value" v(s)?

- Use the TD prediction based on the next state s'

$$u(s) = R(s) + \gamma \hat{U}_\theta(s')$$

this is the same as previous TD method only with approximation

TD-based RL for Linear Approximators

1. Start with initial parameter values
2. Take action according to an explore/exploit policy (should converge to greedy policy, i.e. GLIE)
3. Update estimated model
4. Perform TD update for each parameter

$$\theta_i \leftarrow \theta_i + \alpha\left(R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)\right) f_i(s)$$

5. Goto 2

- Note that step 2 still requires T to select action
- To avoid this we can do the same thing for model-free Q-learning

Q-learning with Linear Approximators

$$\hat{Q}_\theta(s,a) = \theta_0 + \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$$

Features are a function of states and actions.

1. Start with initial parameter values
2. Take action according to an explore/exploit policy (should converge to greedy policy, i.e. GLIE)
3. Perform TD update for each parameter

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

4. Goto 2

• For both Q and U, these algorithms converge to the closest linear approximation to optimal Q or U.

Summary of RL

- MDP
  - Definition of an MDP (T, R, S)
  - Solving MDP for optimal policy: Value iteration, policy iteration
- RL
  - Difference between RL and MDP
  - Different methods for Passive RL: DUE, ADP, TD
  - Different method for Active RL: ADP, Q-Learning with TD learning
  - Function approximation for large state/action space

Learning objectives

1) Students are able to apply supervised learning algorithms to prediction problems and evaluate the results.

2) Students are able to apply unsupervised learning algorithms to data analysis problems and evaluate results.

3) Students are able to apply reinforcement learning algorithms to control problem and evaluate results.

4) Students are able to take a description of a new problem and decide what kind of problem (supervised, unsupervised, or reinforcement) it is.