EECS 496: Sequential Decision Making

Soumya Ray

sray@case.edu

Office: Olin 516

Office hours : T 4-5:30 or by appointment

Note

- Bring calculators for test Thursday

Recap

- Following the greedy policy is not model free because: _____
- To obtain the policy in a model-free way we need the ___ ___ function.
- This is defined as the expected reward obtained when we ________.
- What is the Bellman equation for the Q function?
- What is the relationship between the Q and V functions?
- What is the BOC for the Q function?
- How do we define the optimal policy from the Q function?
- What is the TD error for the Q function?
- Q learning is off policy learning because:______.
- What is the difference between Q-learning and SARSA?
- What are the tradeoffs between model-based and model-free RL?

Dealing with Large State Spaces

- In all the previous methods, we have represented the value functions as tables
- For small MDPs this is OK, but it quickly blows up
  - The state space of PA3 exceeds 10^{10}
- In such cases, we can't represent the value functions/transition functions/action-value functions exactly

Function approximation

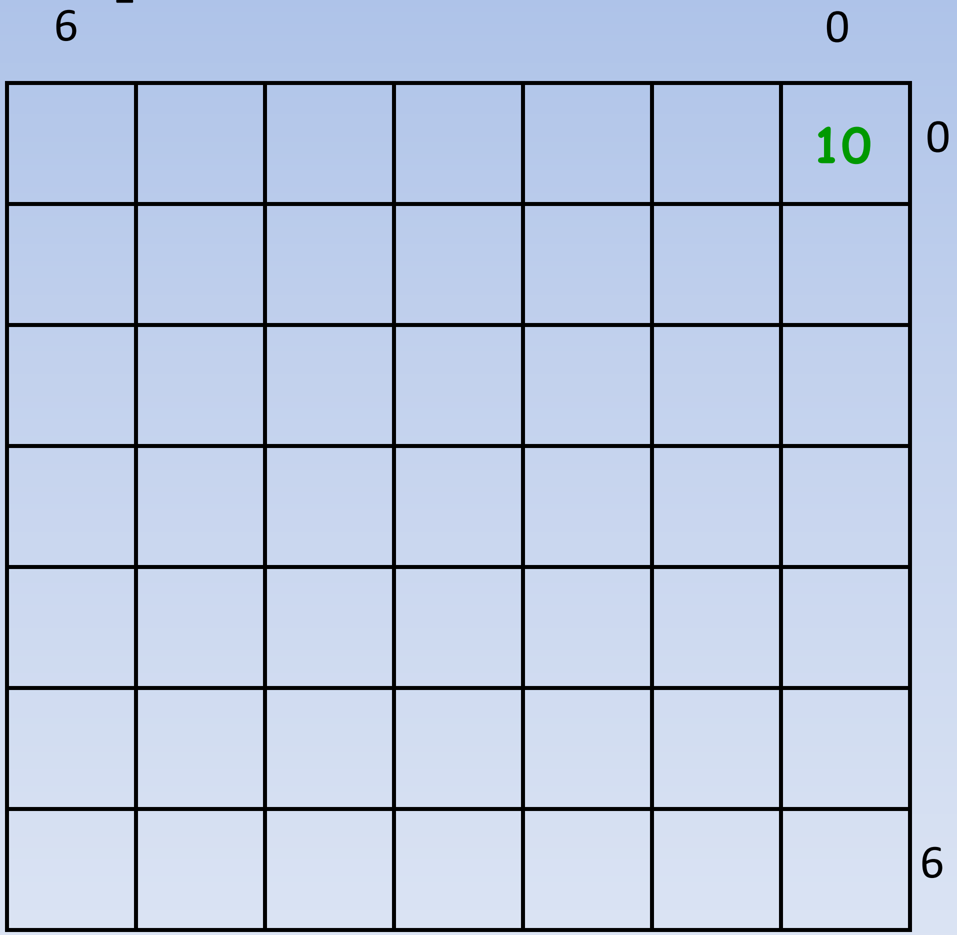
- An alternative is to represent the functions in a parametric form, e.g. using linear functions:

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

- Each $f_i(s,a)$ is a *feature* of a state and action pair
  - It is expected that these features are somehow correlated with/influencing/causing the value of the state/action pair

Example

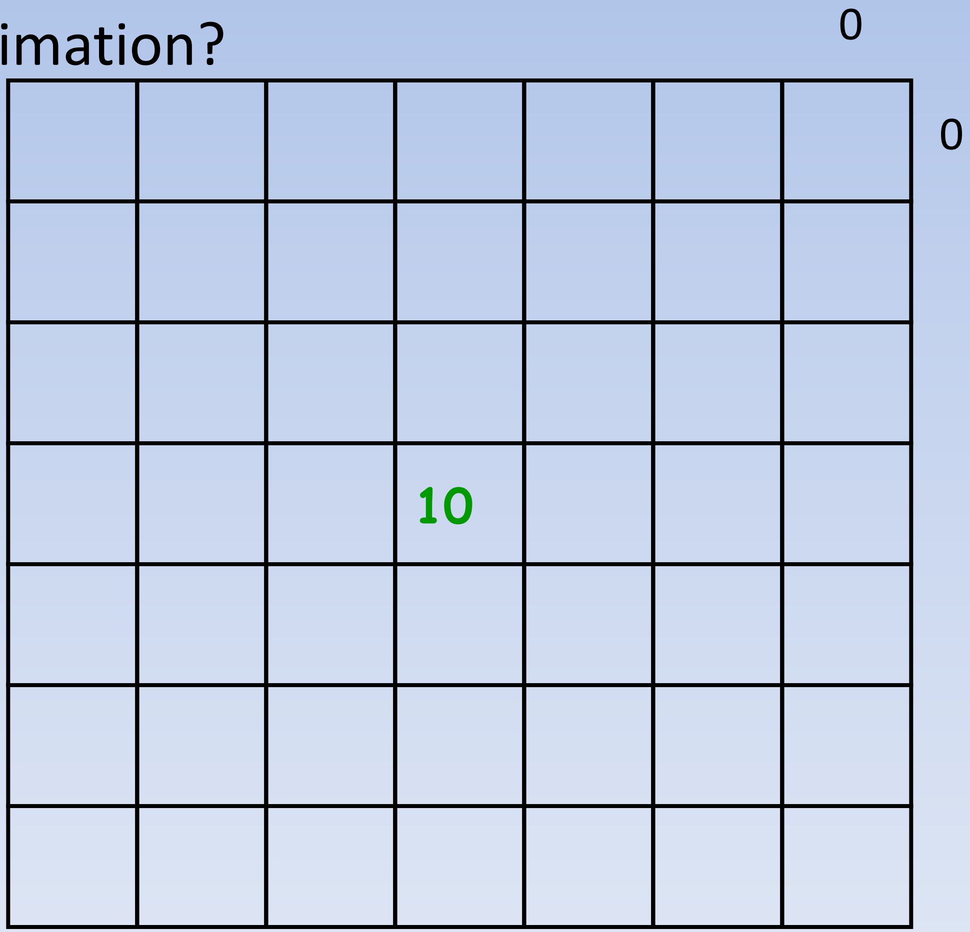
- Consider grid problem with no obstacles, deterministic actions U/D/L/R (49 states), γ=1, −1 reward for each step
- Features for state s=(x,y): f_1(s)=x, f_2(s)=y (just 2 features)
- $V(s) = \theta_0 + \theta_1\, x + \theta_2\, y$
- Is there a good linear approximation?
  - Yes.
  - θ_0 =10, θ_1 = -1, θ_2 = -1
  - (note upper right is origin)



- V(s) = 10 - x - y
  subtracts Manhattan dist.
  from goal reward

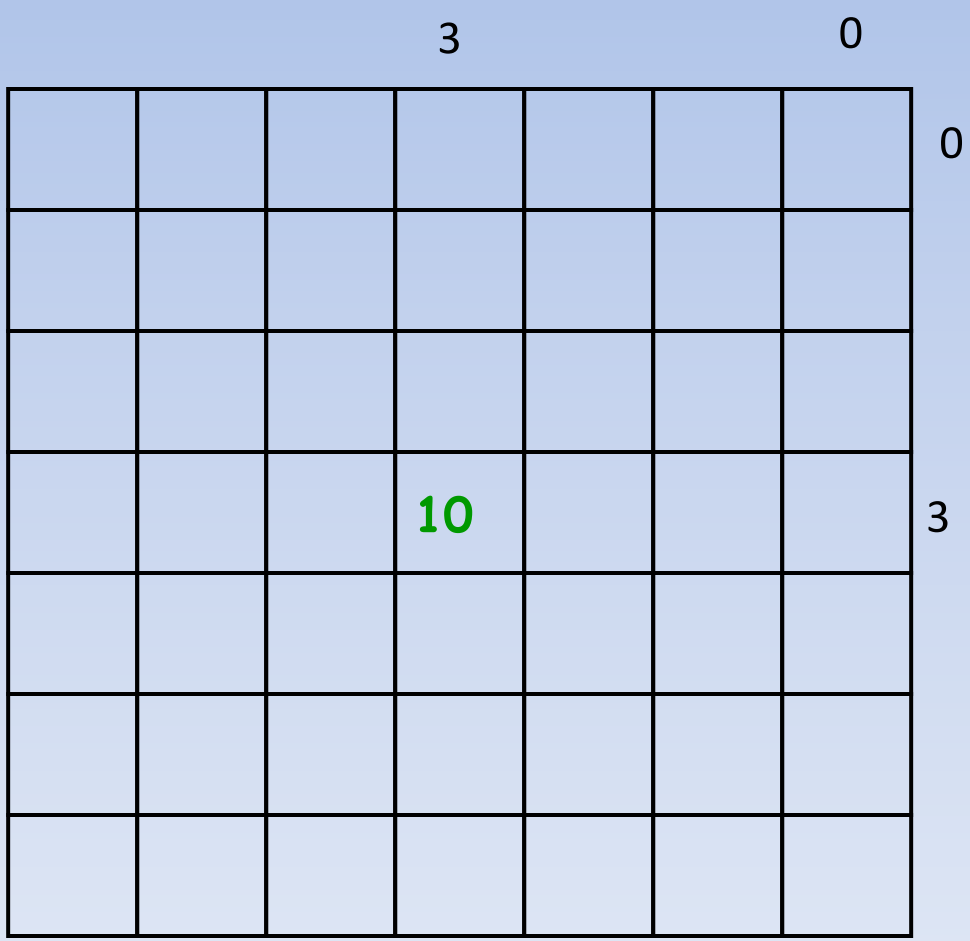
Example 2

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



Example 3

- $V(s) = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 z$
- Include new feature z
  - $z = |3-x| + |3-y|$
  - z is distance to goal location
- Does this allow a good linear approx?
  - $\theta_0 = 10$, $\theta_1 = \theta_2 = 0$,
    $\theta_0 = -1$



Consequences

+ Function approximation results in huge storage savings
+ Because states/actions are represented via features, provides *generalization*
  + Connection to Machine Learning
- Leads to *aliasing* (POMDPs)
- Convergence guarantees may be lost
- Need careful feature design for effective learning

Note

- Can we always define features that allow for a perfect linear representation of the value function?
  - Yes. Assign each state an indicator feature. (i.e. i^{th} feature is 1 iff environment is in i^{th} state , θ_i represents value of i^{th} state)
  - Should we do this?

Example: Tactical Battles in SEPIA

- **<u>States</u>**: contain information about the locations, health, and current activity of all friendly and enemy unit

- **<u>Actions</u>**: *Attack(E)*
  - – causes friendly unit to attack enemy E

- **<u>Policy</u>**: represented via exp($Q_{\theta,F}(s,Attack(E))$)
  - – At each event point loop through each friendly unit F and select enemy E to attack that maximizes $Q_{\theta,F}(s,Attack(E))$

- $Q_{\theta,F}(s,Attack(E))$ generalizes over any enemy agents E
  - – Each friendly unit maintains its own policy

- **RL Task**: learn a policy to control n friendly units in a battle against m enemy units

Example: Tactical Battles in SEPIA

$$Q_{\theta,F}(s,a) = w_{1,F} + w_{1,F}\, f_1(s,a) + w_{2,F}\, f_2(s,a) + \ldots + w_{n,F}\, f_n(s,a)$$

- Set of features
  $\{f_1(s,Attack(E)), \ldots., f_n(s,Attack(E))\}$

- **<u>Example Features</u>**:
  - # of other friendly agents that are currently attacking E
  - Health of enemy agent E relative to F
  - Is E the enemy agent that F is currently attacking?
  - Is F the closest friendly agent to E?
  - Is E the closest enemy agent to F?
  - …
- Features are well defined for any number of enemies
- Could have different features for different types of units, e.g. melee and ranged units

Q-Learning

- Start with an arbitrary Q function Q_0
- Follow greedy policy π with GLIE exploration
- For each observed (s,a,s'), do

$$Q_{i+1}(s,a) \leftarrow Q_i(s,a) + \alpha\left[R(s,a) + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a)\right]$$

- Until convergence

Q-Learning with Linear Function Approximation

- Suppose $Q_{\mathbf{w}}(s,a)=\sum_i w_i f_i(s,a)$
- How does Q-learning work now?
  - We'll need to update the w_i's to change the Q function
- Suppose we observe a transition (s, a, s')
- For this transition, we have a "target Q-value":

$$Q_t(s,a)=\left(R(s,a)+\gamma \max_{a'} Q_{\mathbf{w}}(s',a')\right)$$

- and our current value, $Q_{\mathbf{w}}(s,a)=\sum_i w_i f_i(s,a)$

Q-Learning with Function Approximation

- Define the TD-loss function:

$$E(\mathbf{w}) = \frac{1}{2}\left(Q_t(s,a) - Q_{\mathbf{w}}(s,a)\right)^2$$

Target Q value

Current Estimate

- Then if we minimize this loss function w.r.t. $\mathbf{w}$, we will find the $\mathbf{w}$ we need

Q-Learning with Linear Function Approximation

$$E(\mathbf{w}) = \frac{1}{2}\left(Q_t(s,a) - Q_{\mathbf{w}}(s,a)\right)^2$$

$$\frac{\partial E}{\partial w_i} = \left(Q_t(s,a) - Q_{\mathbf{w}}(s,a)\right)\left(-\frac{\partial Q_{\mathbf{w}}}{\partial w_i}\right)$$

$$\frac{\partial Q_{\mathbf{w}}}{\partial w_i} = \frac{\partial\left(\sum_i w_i f_i(s,a)\right)}{\partial w_i} = f_i(s,a)$$

$$= \left(Q_t(s,a) - Q_{\mathbf{w}}(s,a)\right)\left(-f_i(s,a)\right)$$

$$w_i \leftarrow w_i - \alpha \frac{\partial E}{\partial w_i}$$

Q-Learning with Linear Function Approximation

- Start with an arbitrary $\mathbf{w}$. This defines an initial Q_0.
- Follow greedy policy π with GLIE exploration
- For each observed (s,a,s'), do

$$Q_t(s,a) = R(s,a) + \gamma \max_{a'} Q_{\mathbf{w}}(s',a')$$

$$w_i \leftarrow w_i + \alpha \left(Q_t(s,a) - Q_{\mathbf{w}}(s,a) \right) f_i(s,a)$$

- ~~Until convergence~~

How to check??

Checking convergence

- For large state spaces, no explicit convergence test is possible
- Usually, two possibilities (or combination)
  - Run algorithm for large number of iterations, then stop
  - Stop when the value of the best greedy policy has been stable for a while