RL with function approximation

CS434
Large state space

- Exiting methods require storing $U(S)$ or $Q(a, S)$.
- When a problem has a large state space we can no longer represent the $U$ or $Q$ functions as explicit tables.
- Even if we had enough memory, it is still not practically feasible, because:
  - We never have enough training data.
  - Learning takes too long.
- What to do?
Better use of training data

Given limited training experience, we must generalize what is learned from one situation to other “similar” new situations

Basic idea:
- Instead of using large table to represent $U$ or $Q$, use a parameterized function (e.g., a linear function of some state features)
  - The number of parameters should be small compared to number of states (generally exponentially fewer parameters)
- Learn parameters from given experience
- When we update the parameters based on observations in one state, then our $V$ or $Q$ estimate will also change for other similar states
  - i.e. the parameterization facilitates generalization of experience
Linear Function Approximation

• Define a set of state features $f_1(s), ..., f_n(s)$
  – The features are used to represent the states
  – States with similar feature values will be deemed similar

• A common approximation is to represent $U(s)$ as a weighted sum of the features (i.e. a linear function approximation of $U$)

$$\hat{U}_\theta(s) = \theta_0 + \theta_1 f_1(s) + \theta_2 f_2(s) + ... + \theta_n f_n(s)$$
Linear function approximation

- How good is the approximation accuracy fundamentally depends on the information provided by the features
- Can we always define features that allow for a perfect linear approximation?
  - In theory, yes.
  - We can simply assign each state an indicator feature. (i.e. $i$’th feature is 1 iff $i$’th state is present and $\theta_i$ represents value of $i$’th state)
  - Of course this requires far too many parameters and gives no generalization.
- We will look at a simple example to demonstrate how using different features can significantly change how well a function approximation works
Example

• Consider a simple grid problem with 49 states
• S(0,0) has reward 10, other states all zeros
• Deterministic actions U/D/L/R
• Lets consider just 2 features for state s=(x,y): f1(s)=x, f2(s)=y
• U(s) = \theta_0 + \theta_1 x + \theta_2 y
• Is there a good linear approximation?
  – Yes.
  – \theta_0 = 10, \theta_1 = -1, \theta_2 = -1
  – (note upper right is origin)
• U(s) = 10 - x - y
  subtracts Manhattan dist. from goal reward
• Instead of a table of 49 entries, we store 3 parameters
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 \( z \):
  - Distance to goal location
  - i.e., \( z = |3-x| + |3-y| \)
- Does this allow a good linear approximation?
  - \( \theta_0 = 10 \)
  - \( \theta_1 = \theta_2 = 0 \)
  - \( \theta_3 = -1 \)
Learning the linear approximator

• Our goal is to learn good parameter values i.e. feature weights) that approximate the value function $U(S)$ well

• This needs to be incorporated into the overall reinforcement learning framework
  – 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. Observe outcome, 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. Go to 2

What should the target value be?
Use the TD prediction based on the next state \( s' \)

\[ U(s) = R(s) + \gamma \hat{U}_\theta(s') \]
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. Observe outcome, 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

- Step 2 requires a model to select greedy action
- For applications such as Backgammon it is easy to get a simulation-based model
- For others it is difficult to get a good model
- But we can do the same thing for model-free Q-learning
Q-learning with Linear Approximators

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

Features are a function of states and actions.

1. Start with initial parameter values
2. Take action \( a \) according to an explore/exploit policy (should converge to greedy policy, i.e. GLIE) transitioning from \( s \) to \( s' \)
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
Example: Tactical Battles in Wargus

- Wargus is a real-time strategy (RTS) game
  - Tactical battles are a key aspect of the game

- **RL Task:** learn a policy to control $n$ friendly agents in a battle against $m$ enemy agents
  - Policy should be applicable to tasks with different sets and numbers of agents

5 vs. 5

10 vs. 10
Example: Tactical Battles in Wargus

- **States**: contain information about locations, health, and current activity of all friendly and enemy agents

- **Actions**: Attack(F,E)
  - causes friendly agent F to attack enemy E

- **Policy**: represented via Q-function Q(s,Attack(F,E))
  - Each decision cycle loop through each friendly agent F and select enemy E to attack that maximizes Q(s,Attack(F,E))
Example: Tactical Battles in Wargus

\[ \hat{Q}_\theta(s, a) = \theta_1 + \theta_1 f_1(s, a) + \theta_2 f_2(s, a) + ... + \theta_n f_n(s, a) \]

• **Example Features:**
  – # of other friendly agents that are currently attacking E
  – Health of friendly agent F
  – Health of enemy agent E
  – Difference in health values
  – Walking distance between F and E
  – 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 E?
  – ...

• Features are well defined for any number of agents
Example: Tactical Battles in Wargus
• Linear Q-learning in 5 vs. 5 battle
Worlds Best Backgammon Player

- Neural network with 80 hidden units (nonlinear function approximator)
- Used TD-updates for 300,000 games against self
- Is one of the top (2 or 3) players in the world!