EECS 391: Introduction to AI

Soumya Ray

http://vorlon.case.edu/~sray/eecs391_sp12/index.html

Email: sray@case.edu

Office: Olin 516

Office hours: Tues 2:30-4:00 or by appointment

Temporal Difference Learning

- Start with an arbitrary value function V_0
- *For each observed (s,a,s')*, do

$$V_{i+1}^{\pi}(s) \leftarrow V_i^{\pi}(s) + \alpha\left[R(s,a) + \gamma V_i^{\pi}(s') - V_i^{\pi}(s)\right]$$

- Until $\left|V_{i+1}^{\pi}(s) - V_i^{\pi}(s)\right|$ is very small

Learning rate

Temporal difference error

Notice: No explicit $T(s,a,s')$; $R(s,a)$ does not need to be stored

Q-Learning

- The model-free counterpart of active ADP, based on TD-learning
- We define the action-value function, or Q function, $Q^{\pi}(s,a)$
  - The expected future reward for starting at s, taking action a, and following policy π thereafter

$$V^{\pi}(s) = E\left(\sum_t \gamma^t R(s_t, \pi(s_t)) \,|\, s_0 = s\right)$$

$$Q^{\pi}(s,a) = E\left(\sum_t \gamma^t R(s_t, \pi(s_t)) \,|\, s_0 = s, a_0 = a\right)$$

The *Q(s,a)* function

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

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

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

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

$$\pi(s) = \arg\max_{a} Q(s, a)$$

"Greedy" policy

$$V^{\pi}(s) = \max_{a} Q(s, a)$$

Q-Learning

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

Temporal difference error for the Q-function

$$Q_{i+1}(s,a) \leftarrow Q_i(s,a) +$$

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

- Until $\left|Q_{i+1}(s,a) - Q_i(s,a)\right|$ is small enough

Comparison

Have T and R?	Task?	Model-based/free?	Method
Yes	Evaluate Policy	N/A	Solve Bellman Equations
Yes	Find Optimal Policy	N/A	Value/Policy Iteration
No	Evaluate Policy	Model-based	Adaptive DP
No	Evaluate Policy	Model-free	TD-learning
No	Find Optimal Policy	Model-based	"Active" Adaptive DP
No	Find Optimal Policy	Model-free	Q-learning

Model-based and model-free RL

- Given an RL problem, which should we use?
- Model-free methods are more flexible and storage efficient, but slower
- In some RL problems, a reasonable model can be quickly acquired; here model-based methods work well
- Interestingly, no difference in asymptotic rate of convergence between the two

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 (e.g. the MDP in PA5 has >1e60 Q-values)
- 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:

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

- Each $f_i(s,a)$ is a *feature* of a state and action
  - For example, "does this action in this state move me closer to my target"?

Consequences

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

Q-learning with function approximation

- Suppose $Q_w(s,a) = \sum_i w_i f_i(s,a)$
- How does Q-learning work now?
  - We 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":

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

- and our current value, $Q_w(s,a) = \sum_i w_i f_i(s,a)$

Q-learning with function approximation

- Define the "TD-loss" function:

$$L(w) = \frac{1}{2}\left(Q_t(s,a) - Q_w(s,a)\right)^2$$

$$= \frac{1}{2}\left(R(s,a) + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a)\right)^2$$

Q-learning with function approximation

- Then:

$$L(w) = \frac{1}{2}\left(R(s,a) + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a) \right)^2$$

$$\frac{\partial L}{\partial w_i} = -\left(R(s,a) + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a) \right) \frac{\partial Q}{\partial w_i}$$

$$= -\left(R(s,a) + \gamma \max_{a'} Q_w(s',a') - Q_w(s,a) \right) f_i(s,a)$$

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

Example: Tactical Battles in SimpleRTS

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

- **Actions**: $Attack(F,E)$
  - Causes friendly unit F to attack enemy E

- **Policy**: represented via Q-function $Q(s,Attack(F,E))$
  - Each decision cycle loop through each friendly unit F and select enemy E to attack that maximizes $Q(s,Attack(F,E))$

- $Q(s,Attack(F,E))$ generalizes over any friendly and enemy units F and E
  - Linear function approximation with Q-learning

Example: Tactical Battles in SimpleRTS

$$\hat{Q}_w(s,a) = w_1 + w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

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

- **Example Features**:
  - # of other friendly units that are currently attacking E
  - Health of friendly unit F
  - Health of enemy unit E
  - Is E the enemy unit that F is currently attacking?
  - …
- Features are well defined for any number of agents

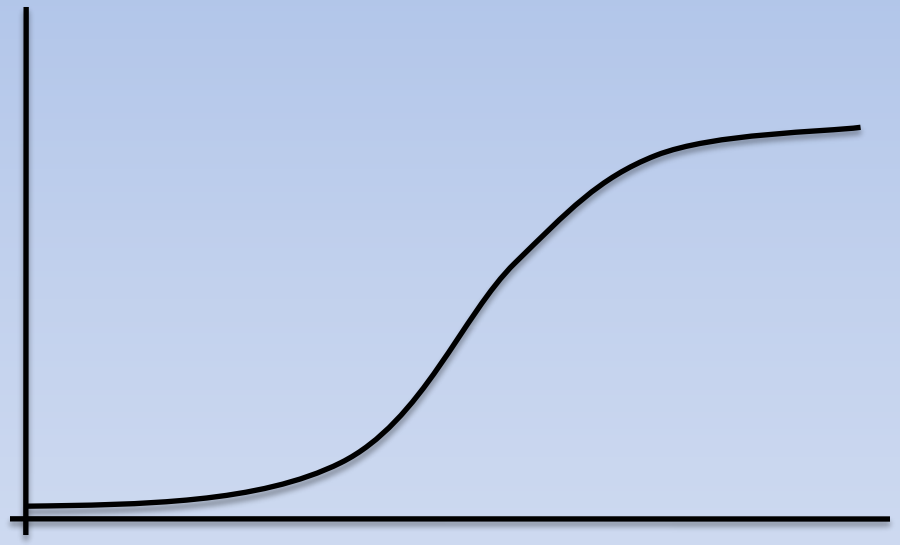
Notes about features

- Features must be computable from **current** primitive state and action alone, or from finite history
  - But can be arbitrarily complex otherwise
    - E.g. Shortest path length to closest enemy

Evaluating RL Algorithms

- Suppose we want to measure the performance of an RL algorithm, say Q-learning, on a problem
- As the algorithm learns, every so often we'll "freeze" the Q-function
- Then for a while, we'll just execute the greedy policy: $\pi(s) = \arg\max_a Q(s,a)$ and track the cumulative rewards

Learning Curves



Because of exploration and stochastic effects, real RL learning curves are much noisier than this, as you'll see in PA5.

Episodic Problems

- In some RL problems, has a “terminal” or “absorbing” state
  - These are called “episodic” problems
  - Most games are like this
  - In these cases, we often put “Number of episodes” on the x-axis of learning curves

Applications of RL

- Reinforcement learning methods have been successfully applied to many complex problems
- In the early 1990s, Gerry Tesauro wrote a program to play backgammon using neural networks to represent Q-functions
- The algorithm was trained by playing against itself for 300,000 games
- The resulting network played as well as the best players at that time

http://www.research.ibm.com/massive/tdl.html

I believe this is true evidence that you have accomplished what you intended to achieve… you have built a smart machine which learns from experience pretty much the same way humans do.

–Kit Woolsey to Gerry Tesauro

Other applications

- Robot control
- Elevator control
- Helicopter control
- Job-shop scheduling
- Network routing

Connections to human learning

- RL in the brain