15-780: Reinforcement Learning

J. Zico Kolter

March 2, 2016

Outline

Challenge of RL

Model-based methods

Model-free methods

Exploration and exploitation

Outline

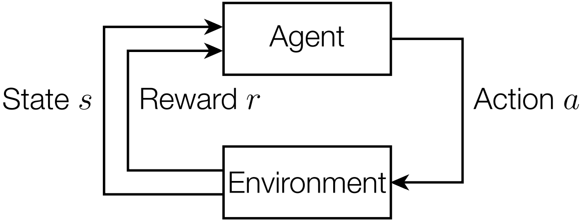
Challenge of RL

Model-based methods

Model-free methods

Exploration and exploitation

Agent interaction with environment



Markov decision processes

Recall a (discounted) Markov decision process is defined by:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R)$$

- $\mathcal{S}$: set of states

- $\mathcal{A}$: set of actions

- $P : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$: transition probability distribution $P(s'|s, a)$

- $R : \mathcal{S} \to \mathbb{R}$: reward function, where $R(s)$ is reward for state s

The RL twist: we don't know P or R, or they are too big to enumerate (only have the ability to act in MDP, observe states and rewards)

Policy $\pi : \mathcal{S} \to \mathcal{A}$ is a mapping from states to actions

We can determine the value of a policy by solving a linear system, or via the iteration (similar to value iteration, but for fixed policy):

$$\hat{V}^{\pi}(s) \leftarrow R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s))\, \hat{V}^{\pi}(s'), \;\; \forall s \in \mathcal{S}$$

We can determine value of optimal policy $V^{\star}$ using value iteration:

$$\hat{V}(s) \leftarrow R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s, a)\, \hat{V}(s'), \;\; \forall s \in \mathcal{S}$$

How can we compute these quantities when P and R are unknown?

Outline

Challenge of RL

Model-based methods

Model-free methods

Exploration and exploitation

Model-based RL

A simple approach: just estimate the MDP from data

Agent acts in the work (according to some policy), observes experience

$$s_1, r_1, a_1, s_2, r_2, a_2, \ldots, s_m, r_m, a_m$$

We form the empircal estimate of the MDP via the counts

$$\hat{P}(s'|s,a) = \frac{\sum_{i=1}^{m-1} \mathbf{1}\{s_i = s, a_i = a, s_{i+1} = s'\}}{\sum_{i=1}^{m-1} \mathbf{1}\{s_i = s, a_i = a\}}$$

$$\hat{R}(s) = \frac{\sum_{i=1}^{m} \mathbf{1}\{s_i = s\} r_i}{\sum_{i=1}^{m} \mathbf{1}\{s_i = s\}}$$

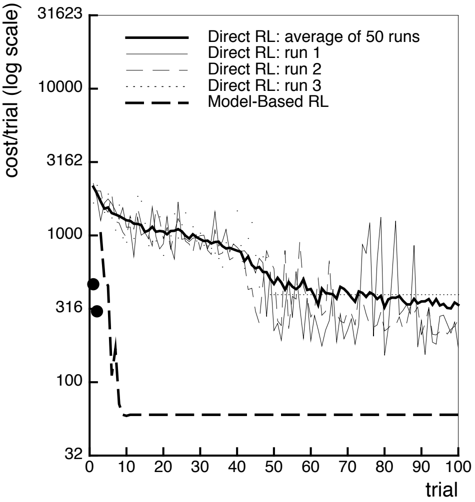
Now solve the MDP $(S, A, \hat{P}, \hat{R})$

Will converge to correct MDP (and hence correct value function / policy) given enough samples of each state

How can we ensure we get the “right” samples? (a challenging problem for all methods we present here, stay tuned)

Advantages (informally): makes “efficient” use of data

Disadvantages: requires we build the the actual MDP models, not much help if state space is too large



(Atkeson and Santamaría, 96)

Outline

Challenge of RL

Model-based methods

Model-free methods

Exploration and exploitation

Model-free RL

Temporal difference methods (TD, SARSA, Q-learning): directly learn value function V^π or $V^\star$ (or a slight generalization of value function, that we will see shortly)

Direct policy search: directly learn optimal policy $\pi^\star$ (covered in a later lecture)

Temporal difference (TD) methods

Let's consider computing the value function for a fixed policy via the iteration

$$\hat{V}^\pi(s) \leftarrow R(s) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,\pi(s))\,\hat{V}^\pi(s'), \;\; \forall s \in \mathcal{S}$$

Suppose we are in some state s_t, receive reward r_t, take action $a_t = \pi(s_t)$ and end up in state s_{t+1}

We can't update $\hat{V}^\pi$ for all s, but can we update just for s_t?

$$\hat{V}^\pi(s_t) \leftarrow r_t + \gamma \sum_{s' \in \mathcal{S}} P(s'|s_t, a_t)\,\hat{V}^\pi(s')$$

...No, because we still can't compute this sum

But, s_{t+1} is a *sample* from the distribution $P(s'|s_t, a_t)$, so we could perform the update

$$\hat{V}^{\pi}(s_t) \leftarrow r_t + \gamma\,\hat{V}^{\pi}(s_{t+1})$$

Too "harsh" an assignment, assumes that s_{t+1} is the only possible next state; instead "smooth" the update using some $\alpha < 1$

$$\hat{V}^{\pi}(s_t) \leftarrow (1-\alpha)\,\hat{V}^{\pi}(s_t) + \alpha\left(r_t + \gamma\,\hat{V}^{\pi}(s_{t+1})\right)$$

This is the *temporal difference* (TD) algorithm

Temporal difference (TD) algorithm

TD algorithm is essentially stochastic version of policy evaluation iteration

algorithm $\hat{V}^\pi = \text{TD}(\pi, \alpha, \gamma)$
// Estimate value function V^π
initialize $\hat{V}^\pi(s) \leftarrow 0$
repeat
Observe state s and reward r
Take action $a = \pi(s)$, and observe next state s'
$\hat{V}^\pi(s) \leftarrow (1-\alpha)\hat{V}^\pi(s) + \alpha(r + \gamma \hat{V}^\pi(s'))$
return $\hat{V}^\pi$

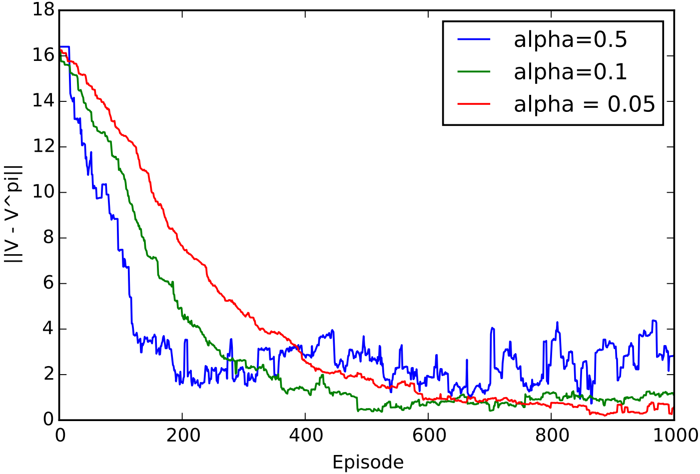
Will converge to $\hat{V}^\pi(s) \rightarrow V^\pi(s)$ (for all s visited frequently enough)

TD Experiments

0	0	0	1
0		0	-100
0	0	0	0

Run TD on gridworld domain for 1000 episodes (each episode consists of 10 steps, sampled according to policy, starting at a random state), initializing with $\hat{V} = R$

TD Progress



Progress of TD methods for different values of α

TD lets us learn the value function of a policy π directly, *without* ever constructing the MDP

But is this really that helpful?

Consider trying to execute greedy policy w.r.t. estimated $\hat{V}^\pi$

$$\pi'(s) = \max_a \sum_{s'} T(s, a, s') \hat{V}^\pi(s')$$

we need a model anyway

SARSA and Q-learning

Q functions (for MDPs in general) are like value functions but defined over state-action pairs

$$Q^\pi(s,a) = R(s) + \sum_{s' \in \mathcal{S}} P(s'|s,a)\, Q^\pi(s', \pi(s'))$$

$$\begin{aligned} Q^\star(s,a) &= R(s) + \sum_{s' \in \mathcal{S}} P(s'|s,a) \max_{a'} Q^\star(s',a') \\ &= R(s) + \sum_{s' \in \mathcal{S}} P(s'|s,a)\, V^\star(s') \end{aligned}$$

I.e., Q function is value of starting is state s, taking action a, and then acting according to π (or optimally, for $Q^\star$)

We can easily construction analogues of value iteration or policy evaluation to construct Q functions directly given an MDP

Q function leads to new TD-like methods

As with TD, observe state s, reward r, take action a (but not necessarily $a = \pi(s)$), observe next state s'

SARSA: estimate $Q^\pi(s, a)$

$$\hat{Q}^\pi(s, a) \leftarrow (1 - \alpha)\hat{Q}^\pi(s, a) + \alpha \left(r + \gamma \hat{Q}^\pi(s', \pi(s')) \right)$$

Q-learning: estimate $Q^\star(s, a)$

$$\hat{Q}^\star(s, a) \leftarrow (1 - \alpha)\hat{Q}^\star(s, a) + \alpha \left(r + \gamma \max_{a'} \hat{Q}^\star(s', a') \right)$$

Again, these algorithms converge to true Q^π, $Q^\star$ if all state-action pairs seen frequently enough

The advantage of this approach is that we can now select actions *without* a model of MDP

SARSA, greedy policy w.r.t. $Q^\pi(s, a)$

$$\pi'(s) = \max_a \hat{Q}^\pi(s, a)$$

Q-learning, optimal policy

$$\pi^\star(s) = \max_a \hat{Q}^\star(s, a)$$

So with Q-learning, for instance, we can learn optimal policy without model of MDP

Q-Learning Experiments

0	0	0	1
0		0	-100
0	0	0	0

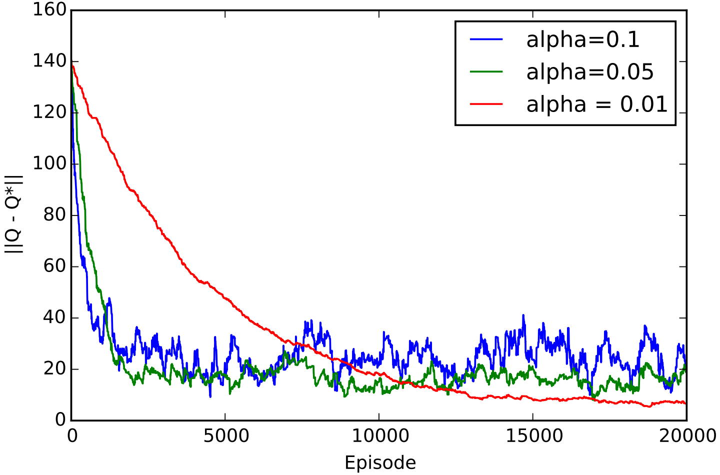
Run Q-Learning on gridworld domain for 20000 episodes (each episode consists of 10 steps), initializing with $\hat{Q}(s, a) = R(s)$

Policy: act according to current optimal policy

$$\pi^\star(s) = \max_a \hat{Q}^\star(s, a)$$

with probability 0.9, act randomly with probaility 0.1 (called an Epsilon-greedy strategy, more on this shortly)

Q-Learning Progress



Progress of Q-learning methods for different values of α

Function approximation

Something is amiss here: we justified model-free RL approaches to avoid learning MDP, but we still need to keep track of value for each state

A major advantage to model-free RL methods is that we can use *function approximation* to represent value function compactly

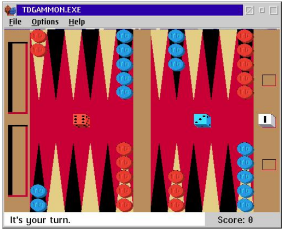
Without going into derivations, let $\hat{V}^\pi(s) = f_\theta(s)$ denote function approximator parameterized by θ, TD update is

$$\theta \leftarrow \theta + \alpha(r + \gamma f_\theta(s') - f_\theta(s))\nabla_\theta f_\theta(s)$$

where $\nabla_\theta f_\theta(s)$ denotes a *gradient* (vector of derivatives) of f with respect to the parameters (much more on this next class)

Similar updates for SARSA, Q-learning

TD Gammon

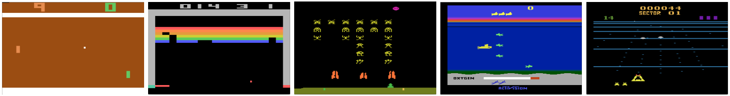


Developed by Gerald Tesauro at IBM Watson in 1992

Used TD w/ neural network as function approximator (known model, but much too large to solve as MDP)

Achieved expert-level play, many world experts changed strategies based upon what AI found

Q-learning for Atari games



Initial paper by Volodymyr Mnih et al., 2013 at DeepMind, more recent paper in Nature, 2015

Q-learning with a deep neural network to learn to play games directly from pixel inputs

DeepMind acquired by Google in Jan 2014

Outline

Challenge of RL

Model-based methods

Model-free methods

Exploration and exploitation

Exploration/exploitation problem

All the methods discussed so far had some condition like "assuming we visit each state enough", or "taking actions according to some policy"

A fundamental question: if we don't know the system dynamics, should we take exploratory actions that will give us more information, or exploit current knowledge to perform as best we can?

Example: a model based procedure that does *not* work

1. Use all past experience to build models $\hat{T}$ and $\hat{R}$ of MDP

2. Find optimal policy for $(S, A, \hat{T}, \hat{R})$ using e.g. value iteration, act according to this policy

Issue is that bad initial estimates in the first few cases can drive policy into sub-optimal region, and never explore further

Key idea: instead of acting according to greedy policy, act according to a policy that will *explore* state-action pairs until we get a "good" estimate of the value function

- **Epsilon-greedy policy**

$$\pi(s) = \begin{cases} \max_a \hat{Q}(s,a) & \text{with probability } 1-\epsilon \\ \text{random action} & \text{otherwise} \end{cases}$$

- **Boltzmann policy**

$$P(a|s) = \frac{\exp\{\tau Q(s,a)\}}{\sum_{a' \in \mathcal{A}} \exp\{\tau Q(s,a')\}}$$

where $\tau \geq 0$ is some parameter ($\tau = 0$: random, $\tau = \infty$: greedy)

Want to decrease ϵ, increase τ as we see more examples, e.g. $\epsilon = 1/\sqrt{n(s)}$ where $n(s)$ is number of times we have visited state s

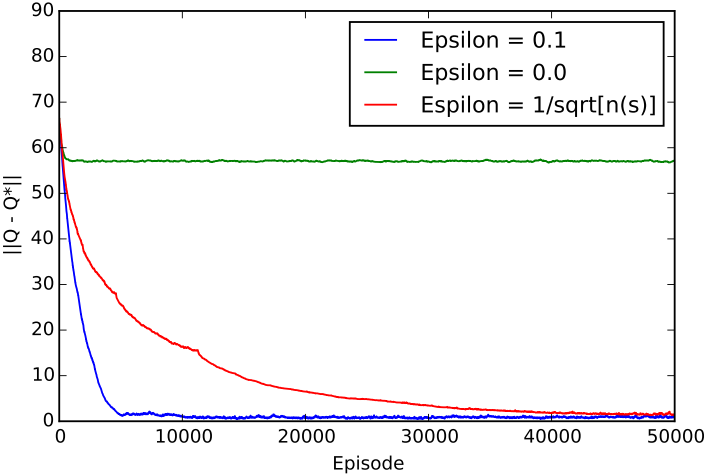
Exploration Experiments

0	0	0	1
0		0	-100
0	0	0	0

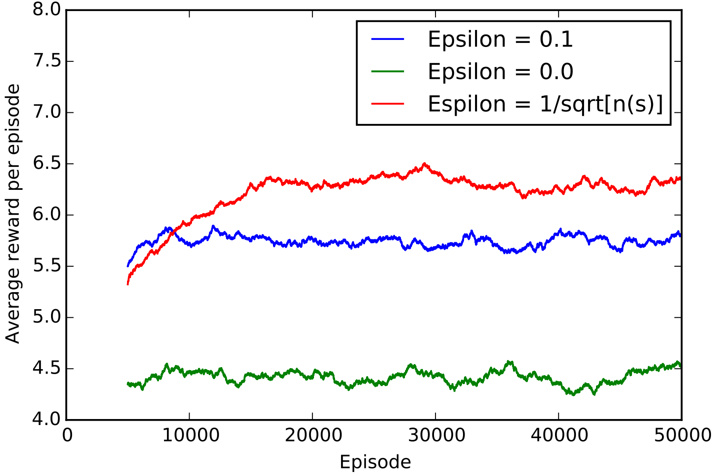
Grid world *but with random uniform* $[0, 1]$ *rewards instead of rewards above*

Initialize Q function with $\hat{Q}(s, a) = 0$

Run with $\alpha = 0.05$, $\epsilon = 0.1$, $\epsilon = 0$ (greedy), $\epsilon = 1/\sqrt{n(s)}$



Error in value function approximation for different settings of ϵ



Average reward (sliding average over past 5000 episodes) for different strategies