Reinforcement Learning

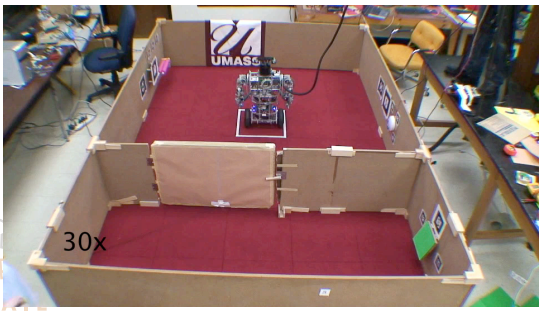
George Konidaris
gdk@csail.mit.edu

November 27th 2012

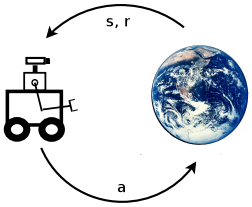
1. INTRODUCTION

1.1 Reinforcement Learning

*Reinforcement Learning is a **machine learning paradigm** concerned with the problem of **learning how to interact** with an environment to **maximize reward**.*



1.2 Formulation



Agent interacts with an environment
At each time t:

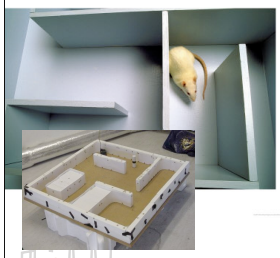
- Receives sensor signal s_t
- Executes action a_t
- *Transition*:
  - new sensor signal s_{t+1}
  - reward r_t

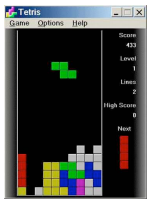
Goal: find policy π that maximizes expected return (sum of discounted future rewards):

$$\max_{\pi} \mathbb{E}\left[R = \sum_{t=0}^{\infty} \gamma^t r_t\right]$$

1.2 Formulation

This formulation is general enough to encompass a wide variety of learned control problems.





1.3 Markov Decision Processes

Common, very useful formalism of the RL problem:

$$< S, A, \gamma, R, T >$$

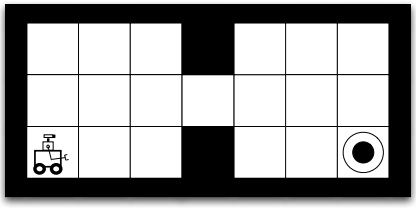
S : set of *states*
A : set of actions
γ : discount factor

R : reward function
$R(s, a, s')$ is the reward received taking action a from state s and transitioning to state s'.

T : transition function
$T(s'|s, a)$ is the probability of transitioning to state s' after taking action a in state s.

1.3 Markov Decision Processes

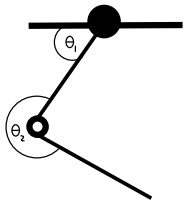
Example:



States: set of grid locations
Actions: up, down, left, right
Transition function: move in direction of action with p=0.9
Reward function: -1 for every step, 1000 for finding the goal

1.3 Markov Decision Processes

Example:



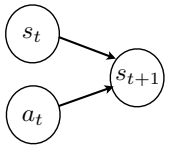
States: $(\theta_1, \dot{\theta_1}, \theta_2, \dot{\theta_2})$ (real-valued vector)
Actions: +1, -1, 0 units of torque added to elbow
Transition function: physics!
Reward function: -1 for every step

1.3 Markov Decision Processes

The Markov Property

$$T(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_1, a_1) = T(s_{t+1}|s_t, a_t)$$

Transitions depend only on the current state and action.
(The agent's history is unimportant.)



1.3 Markov Decision Processes

This allows us to define our target, a **policy**:

$$\pi : S \rightarrow A$$

A policy maps *states* to *actions*.

Given the Markov Property, the **optimal policy** maximizes:

$$\max_{\pi} \forall s, \mathbb{E}\left[R(s) = \sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s\right]$$

1.3 Markov Decision Processes

Reinforcement Learning setting:

- One or both of T, R unknown.
- *Agent does not know how the world works.*
- *When both are known, we have **a planning problem**.*

Exploration vs. Exploitation:

- *Learn more about the environment, or execute best existing policy?*

1.4 Solution Methods

Broadly two families of learning algorithms for RL problems:

- Value Function Methods
- Policy Search Methods

Orthogonal question: *model-based* vs. *model-free*:

- Learn a model, use it to find policy.
- Learn a policy directly.

2. VALUE FUNCTIONS

2.1 Value Functions

Recall that we seek a policy that maximizes:

$$\max_{\pi} \forall s, \mathbb{E}\left[R(s) = \sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s\right]$$

This means that we wish to find a policy that maximizes the **return** from **every state.**

Given a policy, we can estimate of $R(s)$ for every state.

- This is a *value function.*
- It can be used to improve our policy.

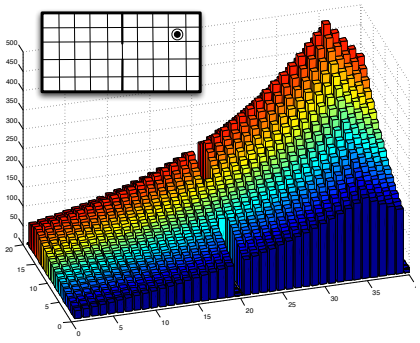
2.1 Value Functions

We define a value function as follows:

$$V_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, s_0 = s\right]$$

This is the value of state s **under policy** π.

2.1 Value Functions



2.1 Value Functions

Similarly, we define a state-action value function as follows:

$$Q_\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, s_0 = s, a_0 = a\right]$$

This is the value of executing a in state s, under policy π.

Note that:

$$Q_\pi(s, \pi(s)) = V_\pi(s)$$

Generally:

- To learn a policy, we need Q.
- We can use V if we have T.
- Theory is similar, V is clearer to write.

2.2 Policy Iteration

Recall that we seek the policy that maximizes $V_\pi(s), \forall s$.

Therefore we know that, for the optimal policy π^*:

$$V_{\pi^*}(s) \geq V_\pi(s), \forall \pi, s$$

$$Q_{\pi^*}(s,a) \geq Q_\pi(s,a), \forall \pi, s, a$$

This means that any change to π that increase Q anywhere obtains a better policy.

2.2 Policy Iteration

This leads to a general policy improvement framework:

1. Start with a policy π
2. Learn Q_π
3. Improve π
   a. $\pi(s) = \max_a Q(s,a), \forall s$

Repeat

This is known as **policy iteration**.
It is guaranteed to converge to the optimal policy.

Steps 2 and 3 can be interleaved as rapidly as you like.
Usually, perform 3a *every time step*.

2.3 Value Function Learning

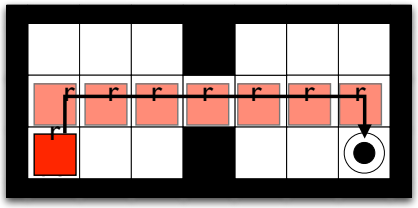
Learning proceeds by gathering samples of $Q(s,a)$.

Methods differ by:

- How you get the samples.
- How you use them to update Q.

2.4 Learning: Monte Carlo

Simplest thing you can do: sample $R(s)$.



Do this repeatedly, average values:

$$Q(s,a) = \frac{R_1(s) + R_2(s) + ... + R_n(s)}{n}$$

2.4 Learning: Monte Carlo

This is rather slow.

- Must wait until the goal has been reached to update.
- If there is no goal (continuing task), you must decide when to cut off.
- High variance.

2.5 Learning: Temporal Difference

Where can we get more (immediate) samples?

Idea: *there is an important relationship between temporally successive states.*

$$R(s_t) = r_t + \gamma R(s_{t+1})$$

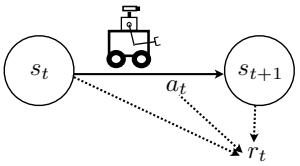
Hence, ideally and in expectation:

$$r_t + \gamma V(s_{t+1}) - V(s_t) = 0$$

V *is correct if this holds in expectation for all states.*
When it does not, it is known as a *temporal difference error.*

2.5 Learning: Temporal Difference

What does this look like?



$$V(s_t) \leftarrow r_t + \gamma V(s_{t+1})$$
$$Q(s_t, a_t) \leftarrow r_t + \gamma Q(s_{t+1}, a_{t+1})$$

2.5 Learning: Temporal Difference

Sarsa: very simple algorithm

1. Initialize Q(s, a)
2. For *n* episodes
   - observe transition (s, a, r, s', a')
   - compute TD error $\delta = r + \gamma Q(s', a') - Q(s, a)$
   - update Q: $Q(s, a) = Q(s, a) + \alpha\delta$
   - select and execute action based on Q

2.5 Learning: Temporal Difference

In Sarsa, we use a sample transition:

$$(s, a, r, s', a')$$

This results in a *sample backup*.

If we had T, we could replace this sample with the full expectation:

$$\delta = \mathbb{E}_{\pi,T}\left[r + \gamma Q(s', a')\right] - Q(s, a)$$

This is known as a *full backup*.
Resulting algorithm: **dynamic programming.**

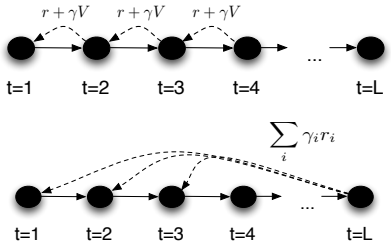
2.5 Learning: Temporal Difference

Dynamic programming finds an optimal policy in time polynomial in $|S|$ and $|A|$.

(Bear in mind that there are $|A|^{|S|}$ possible policies.)

2.6 Learning: Complex Backups

TD and MC two extremes of obtaining samples of Q:



2.6 Learning: Complex Backups

We can generalize this to the idea of an n-step rollout:

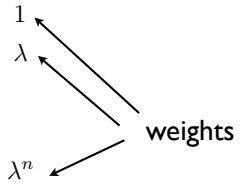
$$R_{s_t}^{(n)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n})$$

Each tells us something about the value function.
- We can combine *all* n-step rollouts.
- This is known as a *complex backup*.

2.6 Learning: Complex Backups

Weighted sum:

$$R^{(1)} = r_0 + \gamma V(s_1)$$
$$R^{(2)} = r_0 + \gamma r_1 + \gamma^2 V(s_2)$$
$$\vdots$$
$$R^{(n)} = \sum_{i=0}^{n-1} \gamma^i r_i + \gamma^n V(s_n)$$



Estimator:

$$R_{s_t}^{\lambda} = (1-\lambda)\sum_{n=0}^{\infty} \lambda^n R_{s_t}^{(n+1)}$$

2.6 Learning: Complex Backups

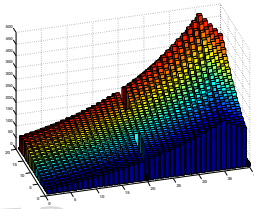
This is called the λ-return.

- At λ=0 we get TD, at λ=1 we get MC.
- Intermediate values of λ usually best.
- TD(λ) family of algorithms

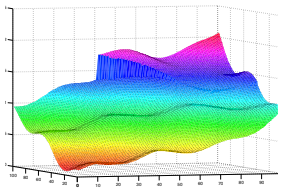
2.7 Value Function Approximation

What if the states are real-valued?

- Cannot use table to represent Q.
- States may never repeat: must *generalize*.



vs



2.7 Value Function Approximation

How do we represent general function of state variables?

Many choices:

- Most popular is *linear value function approximation*.
- Use set of basis functions $\phi_1, \ldots, \phi_m$
- Define linear function of them:

$$\bar{V}(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$

Learning task is to find vector of weights **w** to best approximate V.

2.7 Value Function Approximation

One choice of basis functions:

- Just use state variables directly: $[1, x, y]$

Another:

- Polynomials in state variables.
- E.g., $[1, x, y, xy, x^2, y^2, xy^2, x^2yx^2y^2]$
- This is like a Taylor expansion.

Another:

- Fourier terms on state variables.
- E.g., $[1, cos(\pi x), cos(\pi y), \cos(\pi[x+y])]$
- This is like a Fourier Series expansion.

2.7 Value Function Approximation

Algorithms generalize to linear case:

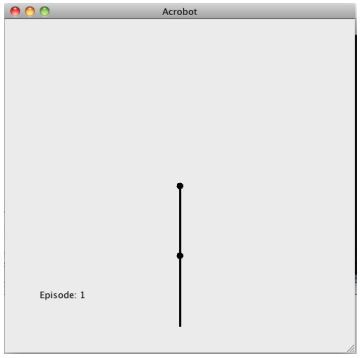
- Tabular case is linear case with indicator basis functions.

Two broad families of methods:

1. Incremental, online methods (e.g., Sarsa)
   - Process each sample as it comes in, discard
   - cf., stochastic gradient descent
   - Slow (samples) but fast (time, memory)

2. Batch methods (e.g., LSTD)
   - Store sufficient statistics, perform batch least-squares
   - cf., linear least-squares
   - Fast (samples) but slow (time, memory)

2.7 Value Function Approximation



2.7 Value Function Approximation

TD-Gammon: Tesauro (circa 1992-1995)

- At or near best human level
- Learn to play Backgammon through self-play
- 1.5 million games
- Neural network function approximator
- TD(λ)

Changed the way the best human players played.

3. POLICY SEARCH (briefly)

3.1 Policy Search

Sometimes policies are simpler than value functions:

- Parametrized program $\pi(s, a|\theta)$

Sometimes we wish to search in space of restricted policies.

In such cases it makes sense to search directly in *policy-space* rather than trying to learn a value function.

3.2 Policy Gradient

Can apply *any* generic optimization method for θ.

One particular approach: policy gradient.

- Compute and ascend $\partial R/\partial\theta$
- This is the gradient of return w.r.t policy parameters

Policy gradient theorem:

$$\frac{\partial R}{\partial \theta} = \sum_s d^\pi(s) \sum_a \frac{\partial \pi(s,a)}{\partial \theta}(Q^\pi(s,a) - b(s))$$

Therefore, one way is to learn Q and then ascend gradient.
Q need only be defined using basis functions computed from θ.

3.2 Policy Gradient

The majority of successful robot applications use policy search / policy gradient methods.

3.2 Policy Gradient

Example:
Kohl and Stone, ICRA 2004.

3.2 Policy Gradient

Learning Dynamic Arm Motions for Postural Recovery

Scott Kuindersma, Rod Grupen, Andy Barto
University of Massachusetts Amherst

Humanoids 2011
Bled, Slovenia

4. SUMMARY

4.1 Reinforcement Learning

Machine Learning for control.

Very active area of current research, applications in:

- Robotics
- Operations Research
- Computer Games
- Theoretical Neuroscience

AI

- The primary function of the brain is control.

Way, *way* lots of work remains.