

Reinforcement learning: TD(λ) and Q-learning

Hal Daumé III

CS5350: Machine Learning

22 April 2008

1. Review of solving MDPs
2. Solving MDPs with unknowns
 - 2.1 Supervised learning
 - 2.2 Certainty-equivalent learning
 - 2.3 Temporal-difference learning
 - 2.4 Q-learning
3. Related models
4. Wrap-up

Markov system *with actions*

Model the world as **states** and **transitions**:

1. States = S_1, S_2, \dots, S_N
2. Rewards = r_1, r_2, \dots, r_N
3. A set of actions a_1, a_2, \dots, a_M
4. Transition matrix = \mathcal{P} with
 $\mathcal{P}_{ij}^k = p(\text{next} = S_j \mid \text{this} = S_i \text{ and I take action } k)$
5. Discount factor γ

Markov process:

1. Begin in state i
2. Received reward r_i
3. Choose action k
4. Randomly move to state j with probability \mathcal{P}_{ij}^k
5. Go to (1) but discount remaining rewards by γ

Value iteration for MDPs

Recall that for non-decision processes, we had:

$$J_i^t = r_i + \gamma \sum_j \mathcal{P}_{ij} J_j^{t-1}$$

For processes with actions, we have:

$$J_i^t = r_i + \max_k \gamma \sum_j \mathcal{P}_{ij}^k J_j^{t-1}$$

- ▶ Above is known as “Bellman’s Equation”
- ▶ J values can be computed exactly as before
- ▶ Also known as ... **Dynamic Programming**

MDPs with Unknowns

General problem: we know neither \mathcal{P} nor r !

For simplicity, we back up again to the **no-action** case!

All we observe is a **trajectory**:

$$S_1, r_1, S_2, r_2, S_3, r_3, \dots, S_T, r_T$$

For example, we may observe:

$$S_1, r = 0, S_2, r = 0, S_3, r = 4, S_2, r = 0, S_4, r = 0, S_5, r = 0$$

From this, we wish to estimate:

$$J_i^* = \text{Expected reward starting in state } i$$

How can we do this?

Estimating J^* by counting...

Observe:

$$S_1, r = 0, S_2, r = 0, S_3, r = 4, S_2, r = 0, S_4, r = 0, S_5, r = 0$$

(assume $\gamma = 1/2$)

	State	Times	LTDR	Mean(LTDR)
We compute:	S_1	1	1	1
	S_2	2,4	2,0	1
	S_3	3	4	4
	S_4	5	0	0
	S_5	6	0	0

Can use these as estimates of J^*

Implementing counting...

Need to store three counts for each state:

- ▶ **Count**, the number of times we've visited this state
- ▶ **Sum**, the sum of (discounted) rewards through this state
- ▶ **Elig**, the *eligibility* of this state

We observe reward r in state i and update:

$$\begin{aligned} \mathbf{Elig}_j &\leftarrow \gamma \mathbf{Elig}_j \quad (\forall j) \\ \mathbf{Elig}_i &\leftarrow \mathbf{Elig}_i + 1 \\ \mathbf{Sum}_j &\leftarrow \mathbf{Sum}_j + r \mathbf{Elig}_j \quad (\forall j) \\ \mathbf{Count}_i &\leftarrow \mathbf{Count}_i + 1 \end{aligned}$$

And then compute:

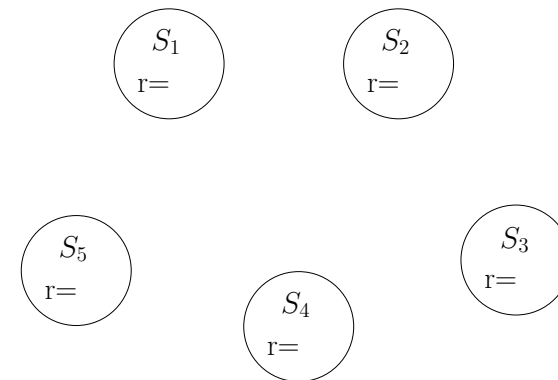
$$J_j^* = \mathbf{Sum}_j / \mathbf{Count}_j$$

Requires $\mathcal{O}(N)$ memory and $\mathcal{O}(N)$ time to update

Alternative: Certainty-equivalent learning

Observe:

$$S_1, r = 0, S_2, r = 0, S_3, r = 4, S_2, r = 0, S_4, r = 0, S_5, r = 0$$



Now, how to estimate J values?

Certainty-equivalent updates

Need to store for each state:

- ▶ Count of each transition
- ▶ Sum of rewards

Takes $\mathcal{O}(N^2)$ memory.

Updates take $\mathcal{O}(1)$ time, but re-evaluation of J^* is $\mathcal{O}(N)$ in the best case ($\mathcal{O}(N^3)$ in the worst):

$$J_i^* = \tilde{r}_i + \gamma \sum_j \tilde{\mathcal{P}}_{ij} J_j^*$$

How can we improve efficiency?

Do a single “backup” of J :

$$J_i^* \leftarrow \tilde{r}_i + \gamma \sum_j \tilde{\mathcal{P}}_{ij} J_j^*$$

Status

Algorithm	Space	Update Cost	Data Efficiency
Supervised learning	$\mathcal{O}(\# \text{ states})$	$\mathcal{O}(\# \text{ states})$	Okay
Certainty-equivalent	$\mathcal{O}(\# \text{ arcs})$	$\mathcal{O}(\# \text{ arcs})$	Great
Backup C-E	$\mathcal{O}(\# \text{ arcs})$	$\mathcal{O}(1)$	Bad
Prioritized C-E	$\mathcal{O}(\# \text{ arcs})$	$\mathcal{O}(1)$	Good

Goal: green in all columns!

Temporal-difference learning

Suppose we observe: $S_i \rightarrow S_j$ with reward r

Update J_i^* to be closer to expected future rewards:

$$J_i^* \leftarrow (1 - \alpha) J_i^* + \alpha [\text{exp. future rewards}]$$

$$\leftarrow (1 - \alpha) J_i^* + \alpha [r_i + \gamma J_j^*]$$

$\alpha = \text{learning rate}$

As always, will (usually) converge to the true values.

Define: $\sigma^2 = \mathbb{V}[\text{reward}] = \mathbb{E}[(r - J^*)^2]$

Then:

$$\lim_{t \rightarrow \infty} \mathbb{E}[(J_t - J^*)^2] = \frac{\alpha \sigma^2}{2 - \alpha}$$

▶ All states visited infinitely often

▶ $\sum_t \alpha_t = \infty$

▶ $\sum_t \alpha_t^2 < \infty$

▶ Eg: $\alpha_t = 1/t$

Temporal-difference(λ) learning

Key problem with TD is that it only does one “backup.”

Variant TD(λ) adds eligibility traces, ala supervised learning:

- ▶ TD(0) is standard TD learning (no eligibility)
- ▶ TD(1) uses complete eligibility (\approx supervised learning)
- ▶ TD(λ) discounts eligibility by a factor of λ
Too-small values are not updated at all
 $\lambda = 0.7$ tends to work well in practice
- ▶ Optimality holds for all $0 \leq \lambda \leq 1$

Updated Status

Algorithm	Space	Update Cost	Data Efficiency
Supervised learning	$\mathcal{O}(\# \text{ states})$	$\mathcal{O}(\# \text{ states})$	Okay
Certainty-equivalent	$\mathcal{O}(\# \text{ arcs})$	$\mathcal{O}(\# \text{ arcs})$	Great
Backup C-E	$\mathcal{O}(\# \text{ arcs})$	$\mathcal{O}(1)$	Bad
Prioritized C-E	$\mathcal{O}(\# \text{ arcs})$	$\mathcal{O}(1)$	Good
TD(0)	$\mathcal{O}(\# \text{ states})$	$\mathcal{O}(1)$	Bad
TD(λ)	$\mathcal{O}(\# \text{ states})$	$\mathcal{O}(-\frac{1}{\log(\gamma\lambda)})$	Okay

But remember—we still haven't talked about actions!!!

The credit assignment problem

Observation sequence:

In state	43	, reward =	0	, action =	2
	39		0		4
	22		0		1
	21		0		3
	13		0		1
	21		0		2
	26		100		

What did I do that made live so good???

Model-based policy learning in MDPs

Algorithm	Space	Update Cost	Data Efficiency
Certainty-equivalent	$\mathcal{O}(\# \text{ arcs})$	$\mathcal{O}(\# \text{ arcs})$	Great
Backup C-E	$\mathcal{O}(\# \text{ arcs})$	$\mathcal{O}(1)$	Bad
Prioritized C-E	$\mathcal{O}(\# \text{ arcs})$	$\mathcal{O}(1)$	Good

- ▶ All of these just worry about estimating the model
- ▶ Once estimated, just use MDP value iteration (rather than Markov system value iteration)

What's missing?

Two ideas for choosing actions:

- ▶ $a = \arg \max_a r_i + \gamma \sum_j \tilde{P}_{ij}^a \tilde{J}_j^*$
- ▶ $a = \text{random}$

Q-learning: Model-free reinforcement learning

Define:

$Q_{i,a}^*$ = Expected LTDR for action a from state i
if I am subsequently optimal

What does this remind us of?

Can you define J^* in terms of Q^* ?

Can you define Q^* in terms of J^* ?

Q-learning update

Note that:

$Q_{i,a}^*$ = Expected LTDR for action a from state i
if I am subsequently optimal

$$r_i + \sum_j \mathcal{P}_{ij} \max_{a'} Q_{j,a'}^*$$

In Q-learning, we store/learn Q^* rather than J^* .

Recall TD-learning:

$$\begin{aligned} J_i^* &\leftarrow (1 - \alpha)J_i^* + \alpha [\text{exp. future rewards}] \\ &\leftarrow (1 - \alpha)J_i^* + \alpha [r_i + \gamma J_j^*] \end{aligned}$$

What might we have for Q^* ?

$$Q_{i,a}^* \leftarrow (1 - \alpha)Q_{i,a}^* + \alpha \left[r_i + \gamma \max_{a'} Q_{j,a'}^* \right]$$

Q-learning: Practical matters

Good things:

- ▶ Q^* values are not biased by any exploration policy
- ▶ Avoids **credit assignment** problem
- ▶ Will (eventually) converge to optimal policy
- ▶ Actually used in practice

Bad things:

- ▶ Can be (very) slow to converge
- ▶ Not good to always be greedy nor random
- ▶ Boltzmann exploration: $prob(a) \propto \exp[-Q_{i,a}^*]$
- ▶ Or just initialize with high values

Modern reinforcement learning

Mostly focuses on large/continuous state spaces:

[Andrew Ng](#) – works on helicopters; Pegasus algorithm

[Drew Bagnell](#) – works on path planning/control; PSDP

[Sham Kakade/Michael Kearns](#) – reinforcement learning theory; E^3

[John Langford](#) – reductions for reinforcement learning; CPI

[Stefan Schaal](#) – imitation learning, humanoid robotics; NPG

[Sebastian Thrun](#) – simultaneous localization and mapping; DARPA

More work on “classical” reinforcement learning:

[Rich Sutton](#) – work on POMDPs with many observations

[Satinder Singh](#) – predictive state representations

[Andy Barto](#) – hierarchical reinforcement learning

Class wrap-up

Summary of Tasks

Task	Input	Output	Data
Binary classification	$x \in \mathcal{X}$	$y \in \{0, 1\}$	$(x_n, y_n)_{n=1}^N$
Multiclass classification	$x \in \mathcal{X}$	$y \in \{1, 2, \dots, M\}$	$(x_n, y_n)_{n=1}^N$
Regression	$x \in \mathcal{X}$	$y \in \mathbb{R}$	$(x_n, y_n)_{n=1}^N$
Component clustering	$x \in \mathcal{X}$	$y \in \{1, 2, \dots, M\}$	$(x_n)_{n=1}^N$
Hierarchical clustering	$x \in \mathcal{X}$	a <i>tree</i>	$(x_n)_{n=1}^N$
Manifold learning	$x \in \mathbb{R}^D$	$y \in \mathbb{R}^d, d \ll D$	$(x_n)_{n=1}^N$
Reinforcement learning	state space	policy	rewards

There are a great number more “canonical” machine learning problems, but these are the most standard.

Geometric modeling **versus** Probabilistic modeling

Important concepts

If you remember nothing else from this class...

- ▶ Generalization: the key to all ML
- ▶ Knowledge = {data, model}
- ▶ Geometric view of data and decision boundaries
- ▶ Hyperplane (linear) classifiers
- ▶ Kernels as a way to non-linearize classifiers
- ▶ Clustering and PCA
- ▶ Probabilistic models and hidden variables

Have a great summer!

p.s., if you want more about graphical models, take CS7941 in the Fall
p.p.s., if you want more RL, take CS5300 in the Spring