Reinforcement Learning with Randomization, Memory, and Prediction

Radford M. Neal, University of Toronto

Dept. of Statistical Sciences and Dept. of Computer Science

`http://www.cs.utoronto.ca/~radford`

CRM - University of Ottawa Distinguished Lecture, 22 April 2016

The Reinforcement Learning Problem

Typical “supervised” and “unsupervised” forms of machine learning are very specialized compared to real-life learning by humans and animals:

- We seldom learn based on a fixed “training set”, but rather based on a continuous stream of information.
- We also act continuously, based on what we’ve learned so far.
- The effects of our actions depend on the state of the world, of which we observe only a small part.
- We obtain a “reward” that depends on the state of the world and our actions, but aren’t told what action would have produced the most reward.
- Our computational resources (such as memory) are limited.

The field of *reinforcement learning* tries to address such realistic learning tasks.

Progress in Reinforcement Learning

Research in reinforcement learning goes back decades, but has never been as prominent as supervised learning: Neural networks, support vector machines, random forests, ... Supervised learning has many prominant successes in large-scale applications from computer vision to bioinformatics.

Reinforcement learn methods have traditionally been first developed in simple contexts with small finite numbers of possible states and actions — a tradition that I will continue in this talk!

But the goal is to eventually migrate such methods to larger-scale problems. This has been very successful in game playing:

- Backgammon (Tesuaro, 1995).
- Atari video games (Mnih, et al, 2013)
- Go (Silver, et al, 2016)

But there is still much to do to handle realistic situations where the world is not fully observed, and we must learn what to remember in a limited memory.

Formalizing a Simple Version of Reinforcement Learning

Let's envision the world going through a seqence of *states*, s_0, s_1, $s_2, \ldots$, at integer times. We'll start by assuming that there are a finite number of possible states.

At every time, we take an *action* from some set (assumed finite to begin with). The sequence of actions taken is a_0, a_1, $a_2, \ldots$.

As a consequence of the state, s_t, and action, a_t, we receive some *reward* at the next time step, denoted by r_{t+1}, and the world changes to state s_{t+1}.

Our aim is to maximize something like the total "discounted" reward we receive over time.

The discount for a reward is γ^{k-1}, where k is the number of time-steps in the future when it is received, and $\gamma < 1$. This is like assuming a non-zero interest rate — money arriving in the future is worth less than money arriving now.

Stochastic Worlds and Policies

The world may not operate deterministically, and our decisions also may be stochastic. Even if the world is really deterministic, an imprecise model of it will need to be probabilistic.

We assume the *Markov property* — that the future depends on the past only through the present state (really the definition of what the state is).

We can then describe how the world works by a transition/reward distribution, given by the following probabilities (assumed the same for all t):

$$P(r_{t+1} = r,\, s_{t+1} = s' \mid s_t = s,\, a_t = a)$$

We can describe our own *policy* for taking actions by action probabilities (again, assumed the same for all t, once we've finished learning a policy):

$$P(a_t = a \mid s_t = s)$$

This assumes that we can observe the entire state, and use it to decide on an action. Later, I will consider policies based on partial observations of the state.

The Q Function

The expected total discounted future reward if we are in state s, perform an action a, and then follow policy π thereafter is denoted by $Q^\pi(s,a)$.

This Q function satisfies the following consistency condition:

$$\begin{aligned} Q^\pi(s,a) \;&=\; \sum_r \sum_{s'} \sum_{a'} \\ &P(r_{t+1}=r,\, s_{t+1}=s' \mid s_t=s,\, a_t=a)\, P^\pi(a_{t+1}=a' \,|\, s_{t+1}=s')\, (r+\gamma Q^\pi(s',a')) \end{aligned}$$

Here, $P^\pi(a_{t+1}=a' \,|\, s_{t+1}=s')$ is an action probability determined by the policy π.

If the optimal policy, π, is deterministic, then in state s it must clearly take an action, a, that maximizes $Q^\pi(s,a)$.

So knowing Q^π is enough to define the optimal policy. Learning Q^π is therefore a way of learning the optimal policy without having to learn the dynamics of the world — ie, without learning $P(r_{t+1}=r,\, s_{t+1}=s' \mid s_t=s,\, a_t=a)$.

Exploration Versus Exploitation

If we know exactly how the world works, and can observe the entire state of the world, there is no need to randomize our actions — we can just take an optimal action in each state.

But if we don't have full knowledge of the world, always taking what appears to be the best action might mean we never experience states and/or actions that could produce higher rewards. There's a tradeoff between:

exploitation:	seeking immediate reward
exploration:	gaining knowledge that might enable higher future reward

In a full Bayesian approach to this problem, we would still find that there's always an optimal action, accounting for the value of gaining knowlege, but computing it might be infeasible. A practical approach is to *randomize* our actions, sometimes doing apparently sub-optimal things so that we learn more.

Exploration While Learning a Policy

When we don't yet know an optimal policy, we need to trade off between exploiting what we do know versus exploring to obtain useful new knowledge.

One simple scheme is to take what seems to be the best action with probability $1-\epsilon$, and take a random action (chosen uniformly) with probability ϵ. A larger value for ϵ will increase exploration.

We might instead (or also) randomly choose actions, but with a preference for actions that seem to have higher expected reward — for instance, we could use

$$P(a_t = a \,|\, s_t = s) \quad \propto \quad \exp\left(Q(s,a)\,/\,T\right)$$

where $Q(s,a)$ is our current estimate of the Q function for a good policy, and T is some "temperature". A larger value of T produces more exploration.

Learning a Q Function and Policy with 1-Step SARSA

Recall the consistency condition for the Q function:

$$Q^\pi(s,a) \;=\; \sum_r \sum_{s'} \sum_{a'} P(r_{t+1}=r,\, s_{t+1}=s' \mid s_t=s,\, a_t=a)\, P^\pi(a_{t+1}=a' \,|\, s_{t+1}=s')\, (r+\gamma Q^\pi(s',a'))$$

This suggests a Monte Carlo approach to incrementally learning Q for a good policy. At time $t{+}1$, after observing/choosing the states/actions s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1} (hence the name SARSA), we update our estimate of $Q(s_t,a_t)$ for a good policy by

$$Q(s_t,a_t) \;\leftarrow\; (1{-}\alpha)\, Q(s_t,a_t) \;+\; \alpha\, (r_{t+1}+\gamma\, Q(s_{t+1},a_{t+1}))$$

Here, α is a "learning rate" that is slightly greater than zero.

We can use the current Q function and the exploration parameters ϵ and T to define our current policy:

$$P(a_t=a \,|\, s_t=s) \;=\; \frac{\epsilon}{\#\text{actions}} \;+\; (1{-}\epsilon)\, \frac{\exp\,(Q(s,a)\,/\,T)}{\sum_{a'} \exp\,(Q(s,a')\,/\,T)}$$

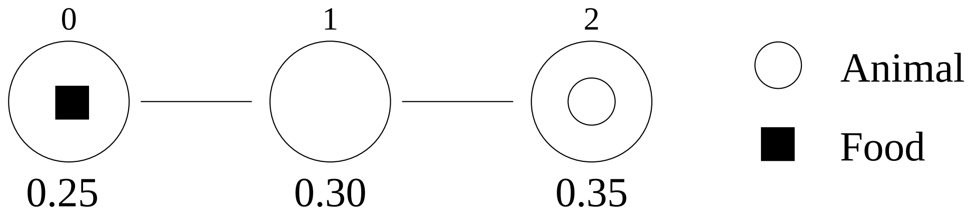
An Example Problem

Consider an animal moving around several locations where food may grow.

At each time step, food grows with some probability at any location without food, the animal may then move to an adjacent location, and finally the animal eats any food where it is.

We assume the animal observes both its location, and whether or not every other location has food.

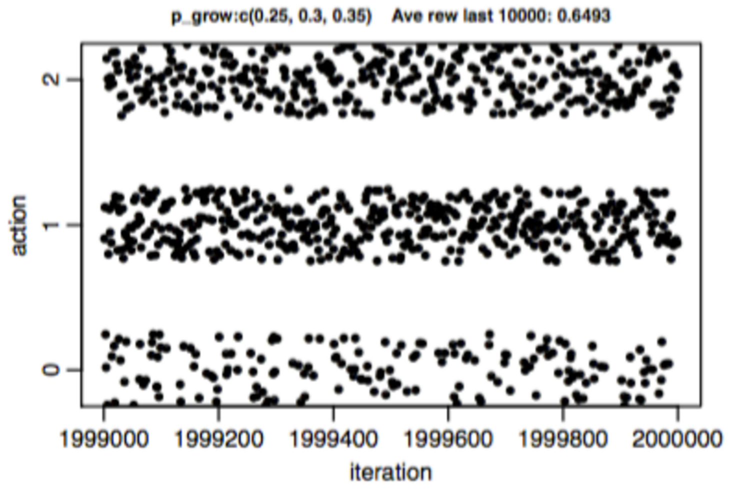
Here's an example with just three locations, with the probabilities of food growing at each location shown below:

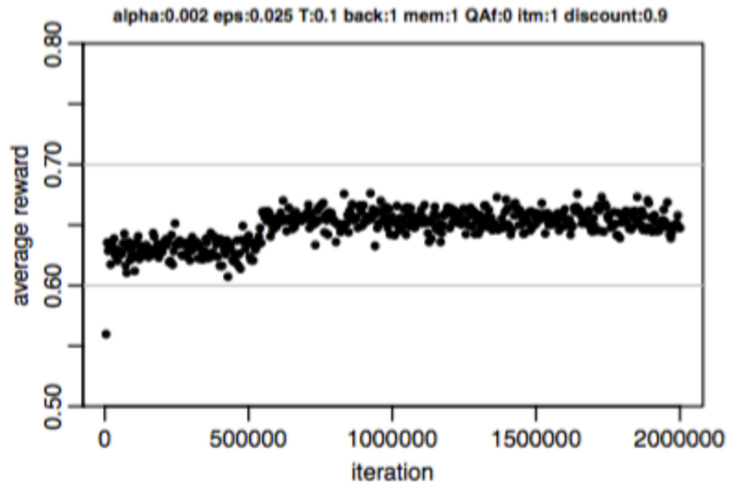


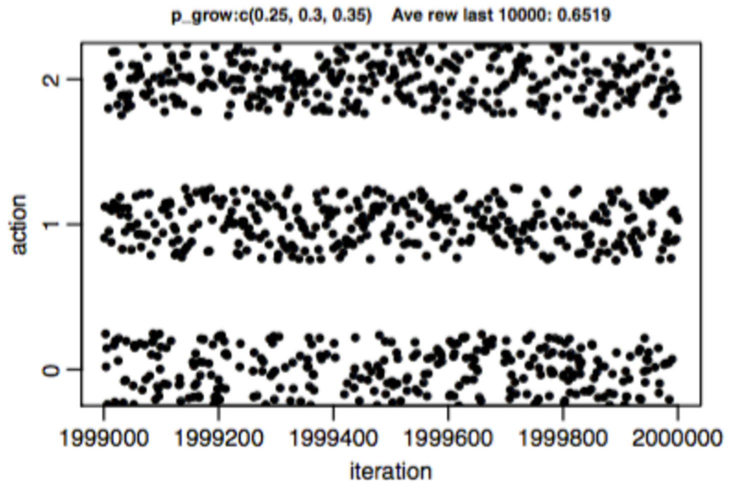
Should the animal move left one step, or stay where it is?

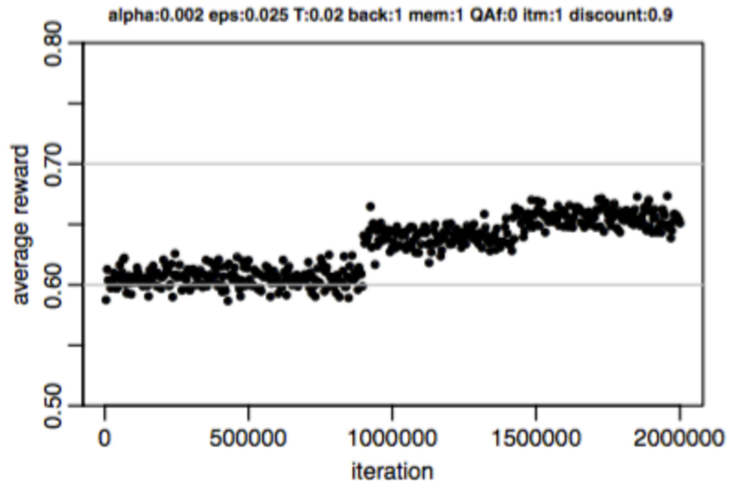
Learning a Policy for the Example with 1-Step SARSA

Two runs, with $T = 0.1$ and $T = 0.02$:









Policies Learned

With $T = 0.1$:

P actions:

	sit	right	left
0-0-0-0	1	98	1
1-0-0-0	49	50	1
2-0-0-0	48	22	30
1-1-0-0	5	1	94
2-1-0-0	17	19	64
0-0-1-0	1	98	1
2-0-1-0	1	1	98
2-1-1-0	1	1	98
0-0-0-1	1	98	1
1-0-0-1	1	98	1
1-1-0-1	1	98	1
0-0-1-1	1	98	1

With $T = 0.02$:

P actions:

	sit	right	left
0-0-0-0	1	98	1
1-0-0-0	98	1	1
2-0-0-0	98	1	1
1-1-0-0	1	1	98
2-1-0-0	1	1	98
0-0-1-0	1	98	1
2-0-1-0	1	1	98
2-1-1-0	1	1	98
0-0-0-1	1	98	1
1-0-0-1	1	98	1
1-1-0-1	1	1	98
0-0-1-1	1	98	1

Learning in Environments with Partial Observations

In real problems we seldom observe the full state of the world. Instead, at time t, we obtain an observation, o_t, related to the state by an observation distribution,

$$P(o_t = o \,|\, s_t = s)$$

This changes the reinforcement learning problem fundamentally:

1) Remembering past observations and actions can now be helpful.
2) If we have no memory, or only limited memory, an optimal policy must sometimes be stochastic.
3) A well-defined Q function exists only if we assume that the world together with our policy is ergodic (visits all possible states).
4) We cannot in general learn the Q function with 1-Step SARSA.
5) An optimal policy's Q function is not sufficient to determine what action that policy takes for a given observation.

Points (1) – (3) above have been known for a long time (eg, Singh, Jaakola, and Jordan, 1994). Point (4) seems to have been at least somewhat appreciated. Point (5) initially seems counter-intuitive, and doesn't seem to be well known.

Memoryless Policies and Ergodic Worlds

To begin, let's assume that we have no memory of past observations and actions, so a policy, π, is specified by a distribution of actions given the current observation,

$$P^\pi(a_t = a \,|\, o_t = o)$$

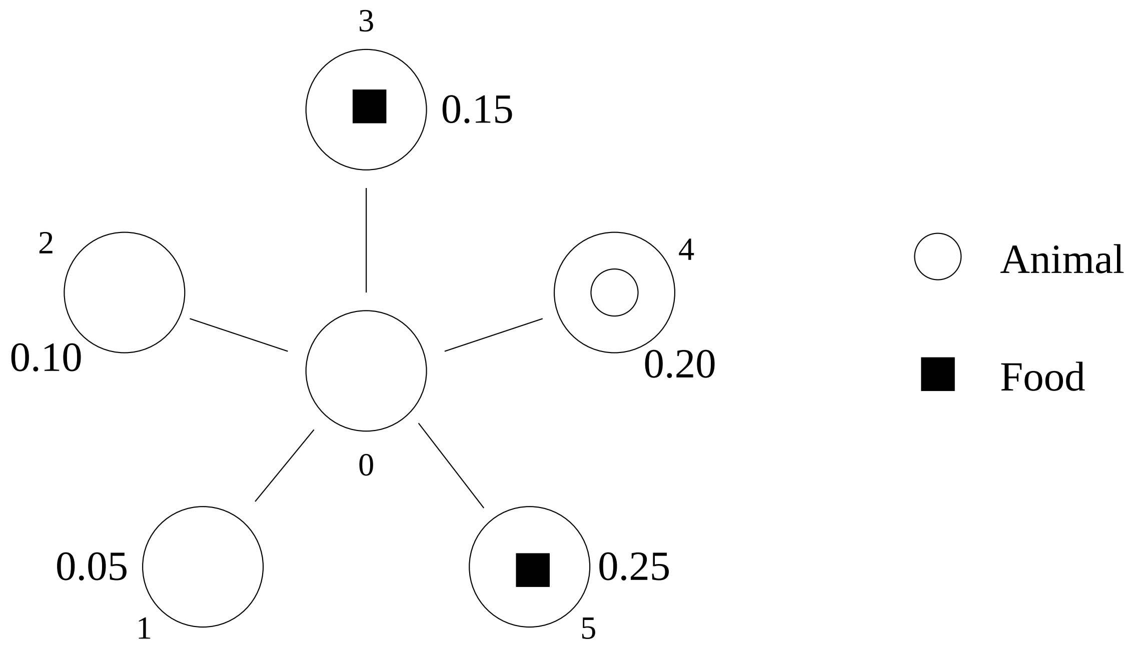
We'll also assume that the world together with our policy is *ergodic* — that all actions and states of the world occur with non-zero probability, starting from any state. In other words, the past is eventually "forgotten".

This is partly a property of the world — that it not become "trapped" in a subset of the state space, for any sequence of actions we take.

If the world is ergodic, a sufficient condition for ergodicity of the world plus a policy is that the policy give non-zero probability to all actions given any observation. We may want this anyway for exploration.

Grazing in a Star World: A Problem with Partial Observations

Consider an animal grazing for food in a world with 6 locations, connected in a star configuration:



Each time step, the animal can move on one of the lines shown, or stay where it is.

The centre point (0) never has food. Each time step, food grows at an outer point $(1, \ldots, 5)$ that doesn't already have food with probabilities shown above. When the animal arrives (or stays) at a location, it eats any food there.

The animal can observe where it is (one of $0, 1, \ldots, 5$), but not where food is.

Reward is $+1$ if food is eaten, -1 if attempts invalid move (goes to 0), 0 otherwise.

Defining a Q Function of Observation and Action

We'd like to define a Q function using observations rather than states, so that $Q(o, a)$ is the expected total discounted future reward from taking action a when we observe o.

Note! This makes sense only if we assume ergodicity — otherwise $P(s_t = s \,|\, o_t = o)$, and hence $Q(o, a)$, are not well-defined.

Also...

- $Q(o, a)$ will depend on the policy followed in the past, since the past policy affects $P(s_t = s \,|\, o_t = o)$.
- $Q(o, a)$ will *not* be the expected total discounted future reward *conditional* on events in the recent past, since the future is not independent of the past given only our current observation (rather than the full state at the current time).
- But with an ergodic world + policy, $Q(o, a)$ will approximate the expected total discounted future reward conditional on events in the distant past, since the distant past will have been mostly "forgotten".

Learning the Q Function with n-Step SARSA

We might try to learn a Q function based on partial observations of state by using the obvious generalization of 1-Step SARSA learning:

$$Q(o_t, a_t) \;\;\leftarrow\;\; (1-\alpha)\, Q(o_t, a_t) \;+\; \alpha\,(r_{t+1} + \gamma\, Q(o_{t+1}, a_{t+1}))$$

But we can't expect this to work, in general — $Q(o_{t+1}, a_{t+1})$ is *not* the expected discounted future reward from taking a_{t+1} with observation o_{t+1} *conditional* on having taken action a_t the previous time step, when the observation was o_t.

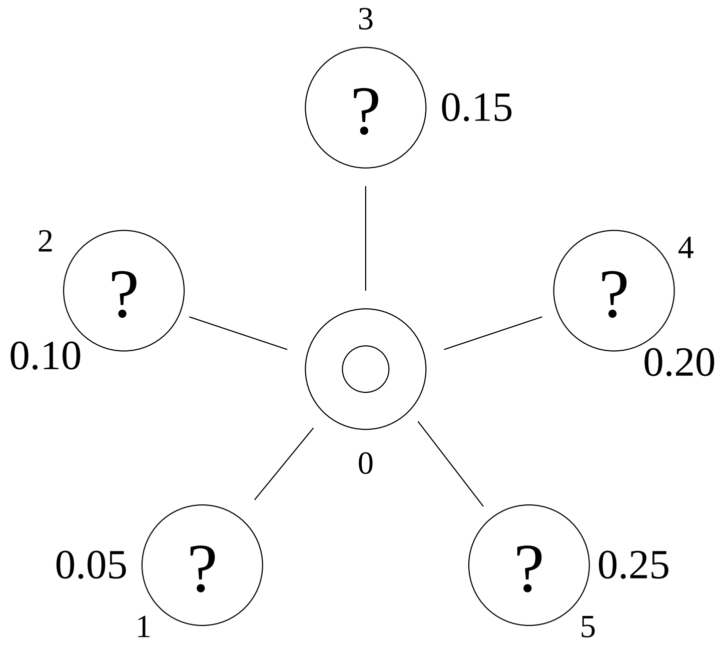
However, if our policy is ergodic, we should get approximately correct results using n-Step SARSA for sufficiently large n. This update for $Q(o_t, a_t)$ uses actual rewards until enough time has passed that a_t and o_t have been (mostly) forgotten:

$$Q(o_t, a_t) \;\;\leftarrow\;\; (1-\alpha)\, Q(o_t, a_t) \;+\; \alpha\,(r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n\, Q(o_{t+n}, a_{t+n}))$$

Of couse, we have to delay this update n time steps from when action a_t was done.

Star World: What Will Q for an "Optimal" Policy Look Like?

Here's the star world, with the animal in the centre. It can't see which other locations have food:



Suppose that the animal has no memory of past observations and actions.

What should it do when it is at the centre?

What should it do when at one of the outer locations?

What will the Q function be like for this policy?

The Optimal Policy and Q Function

In the star world, we see that without memory, a good policy must be stochastic — sometimes selecting an action randomly.

We can also see that the values of $Q(o, a)$ for all actions, a, that are selected with non-zero probability when the observation is o must be *equal*.

But the probabilities for choosing these actions need not be equal.

So the Q function for a good policy is not enough to determine this policy.

But What Does “Optimal” Mean?

But I haven’t said what “optimal” means when the state is partially observed. What should we be optimizing?

The most obvious possibility is the average discounted future reward, averaging over the equilibrium distribution of observations (and underlying states):

$$\sum_o P^\pi(o) \sum_a P^\pi(a|o)\, Q(o,a)$$

Note that the equilibrium distribution of observations depends on the policy being followed, as does the distribution of state given observation.

But with this objective, the discount rate, γ, turns out not to matter!

But it seems to be the most commonly used objective, equivalent to optimizing the long-run average reward per time step.

I’ll instead continue to learn using a discounted reward, which can perhaps be justified as finding a Nash equilibrium for a “game” between policies appropriate when seeing different observations.

Learning a Q Function and an A Function

Since Q for an optimal stochastic policy does not determine the policy, we can try learning the policy separately, with a similar A function, updated based on Q, which is learned with n-Step SARSA.

The algorithm does the following at each time $t+n$:

$$\begin{aligned} Q(o_t, a_t) &\leftarrow (1-\alpha)\, Q(o_t, a_t) + \alpha\, (r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n\, Q(o_{t+n}, a_{t+n})) \\ A &\leftarrow A + fQ \end{aligned}$$

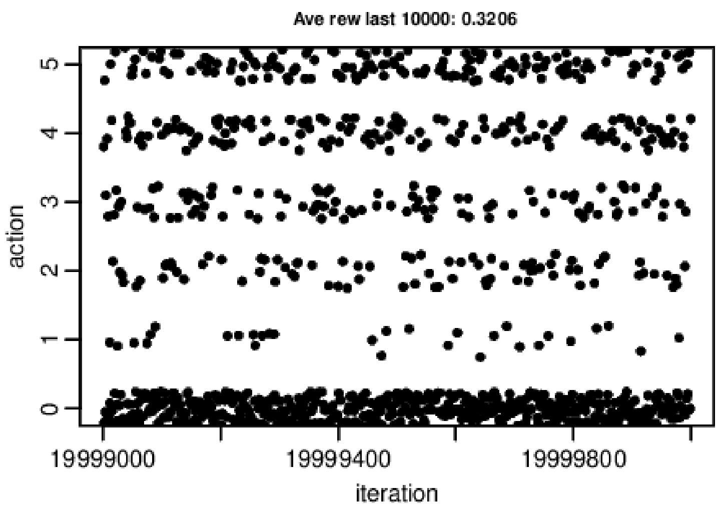
Above, T is a positive “temperature” parameter, and α and f are tuning parameters slightly greater than zero.

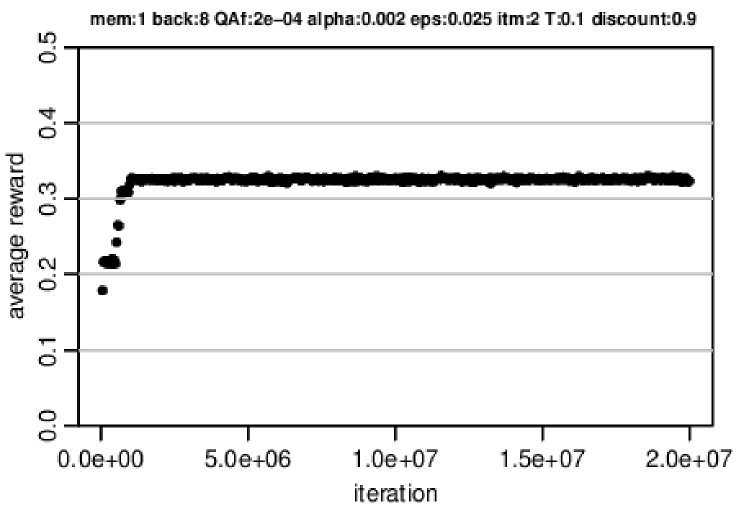
The policy followed is determined by A:

$$P(a_t = a \,|\, o_t = o) = \frac{\epsilon}{\#\text{actions}} + (1-\epsilon)\, \frac{\exp\,(A(o,a)\,/\,T)}{\sum\limits_{a'} \exp\,(A(o,a')\,/\,T)}$$

This is in the class of what are called “Actor-Critic” methods.

Star World: Learning Q and A





Q:

	0	1	2	3	4	5
0	3.164	3.407	3.360	3.404	3.418	3.380
1	3.135	2.928	2.134	2.154	2.146	2.141
2	3.074	2.103	2.937	2.090	2.118	2.159
3	3.069	2.085	2.093	2.977	2.108	2.120
4	3.059	2.056	2.060	2.092	2.962	2.071
5	3.015	2.059	2.079	2.044	2.072	3.026

P action:

	0	1	2	3	4	5
0	0	5	17	19	28	30
1	98	0	0	0	0	0
2	98	0	0	0	0	0
3	98	0	0	0	0	0
4	98	0	0	0	0	0
5	98	0	0	0	0	0

The rows above are for different observations (of position). The Q table shows Q values for actions; the P table shows probabilities of actions, in percent (rounded).

Is This Method Better Than n-Step SARSA?

This method can learn to pick actions randomly from a distribution that is non-uniform, even when the Q values for these actions are all the same.

Contrast this with simple n-Step SARSA, where the Q function is used to pick actions according to

$$P(a_t = a \,|\, o_t = o) \quad = \quad \frac{\epsilon}{\#\text{actions}} \;+\; (1-\epsilon)\,\frac{\exp\left(Q(o,a)\,/\,T\right)}{\sum_{a'} \exp\left(Q(o,a')\,/\,T\right)}$$

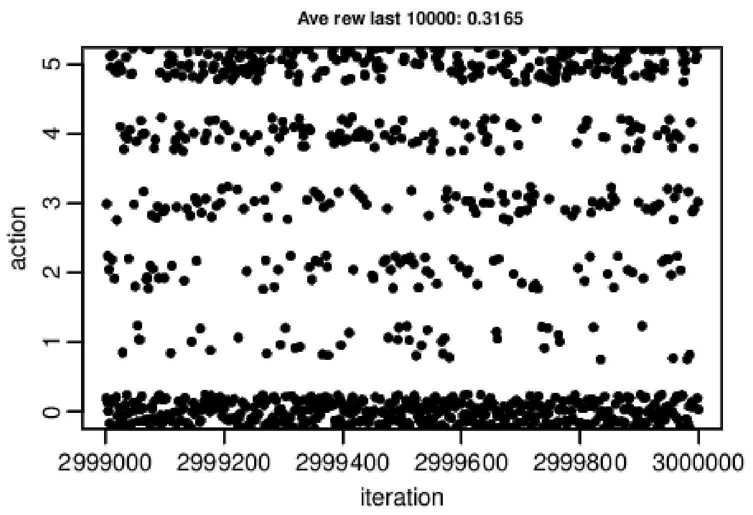
Obviously, you can't have $P(a_t = a \,|\, o_t = o) \;\neq\; P(a_t = a^* \,|\, o_t = o)$ when you have $Q(o,a) \;=\; Q(o,a^*)$.

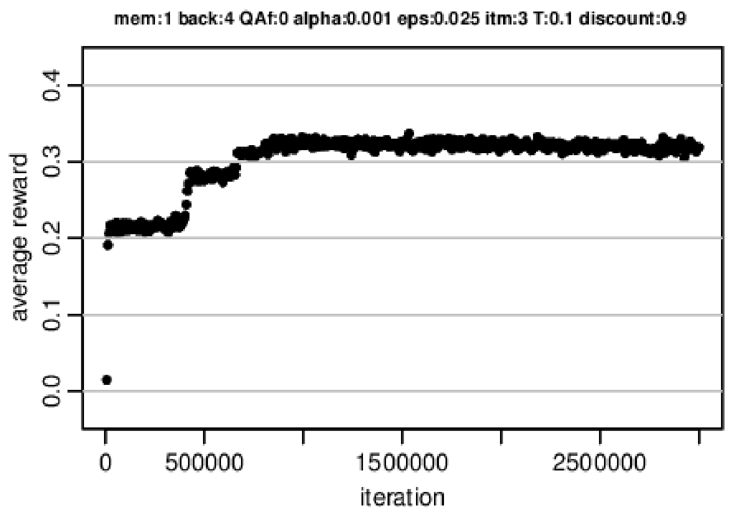
Or is it so obvious? What about the limit as T goes to zero, without being exactly zero?

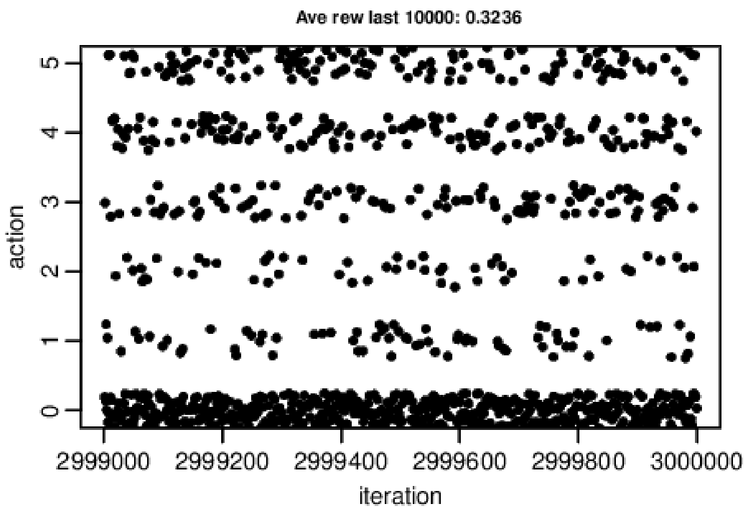
I figured I should checked it out, just to be sure...

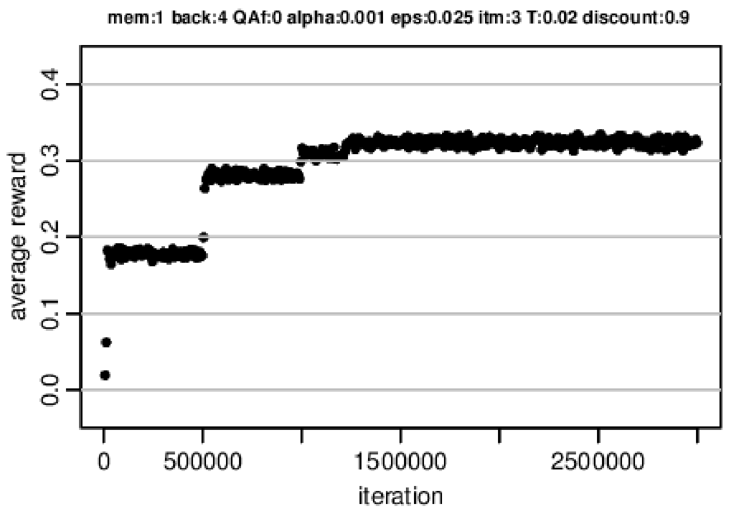
Using Simple n-Step SARSA With Small T Actually Works!

Here is 4-Step SARSA with $T = 0.1$ versus $T = 0.02$:









The Policies Learned

The numerical performance difference seems small, but we can also see a qualitative difference in the policies learned:

```
4-Step SARSA, T=0.1:          4-Step SARSA, T=0.02:

P action:                      P action:
     0  1  2  3  4  5              0  1  2  3  4  5
0    2 10 15 22 27 24         0    0 10 14 25 23 27
1   98  0  0  0  0  0         1   98  0  0  0  0  0
2   98  0  0  0  0  0         2   98  0  0  0  0  0
3   98  0  0  0  0  0         3   98  0  0  0  0  0
4   98  0  0  0  0  0         4   98  0  0  0  0  0
5   60  0  0  0  0 39         5   98  0  0  0  0  0
```

The rows above are for observations (of position). The table entries are action probabilities in percent (rounded).

Comparison of Methods

These methods have different potential deficiencies:

- When learning A using Q, we need to learn Q faster than A, to avoid changing A based on the wrong Q. So f may have to be rather small (much smaller than α).

- When learning only Q, with T very small, the noise in estimating Q gets amplified by dividing by T. We may need to make α small to get less noisy estimates.

Why and How to Remember

When we can't see the whole state, remembering past observations and actions may be helpful if it helps the agent infer the state.

Such memories could take several forms:

- Fixed memory for the last K past observations and actions. But K may have to be quite large, and we'd need to learn how to extract relevant information from this memory.
- Some clever function of past observations — eg, Predictive State Representations (Littman, Sutton, and Singh, 2002).
- Memory in which the agent explicitly decides to record information as part of its actions.

The last has been investigated before (eg, Peshkin, Meuleau, Kaelbling, 1999), but seems to me like it should be investigated more.

Memories as Observations, Remembering as Acting

We can treat the memory as part of the state, which the agent always observes.

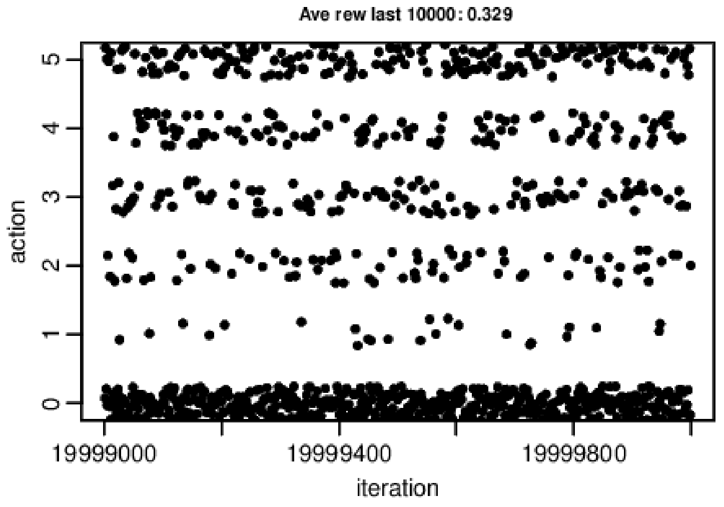
Changes to memory can be treated as part of the action. Most generally, any action could be combined with any change to the memory. But one could consider limiting memory changes (eg, to just a few bits).

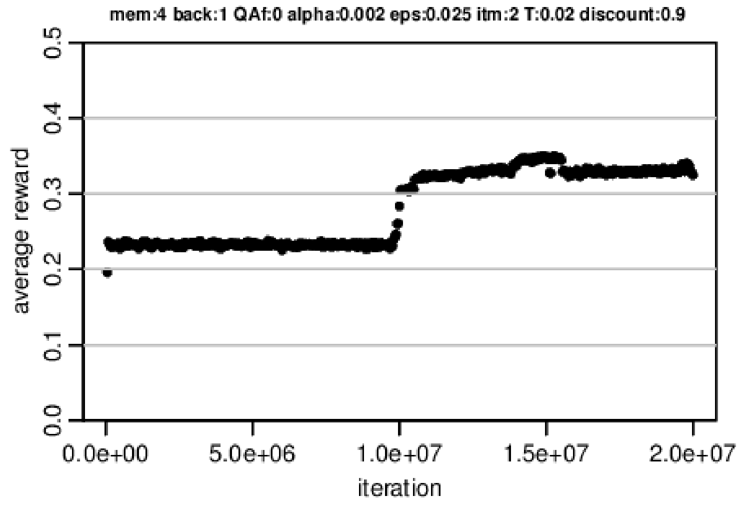
Exploration is needed for setting memory as well as for external actions.

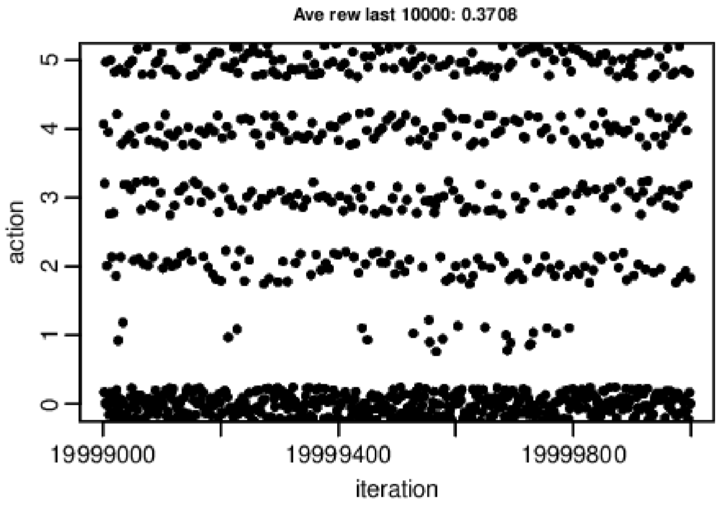
In my experiments, I have split exploration into independent exploration of external actions and of internal memory (though both might happen at the same time, with low probability).

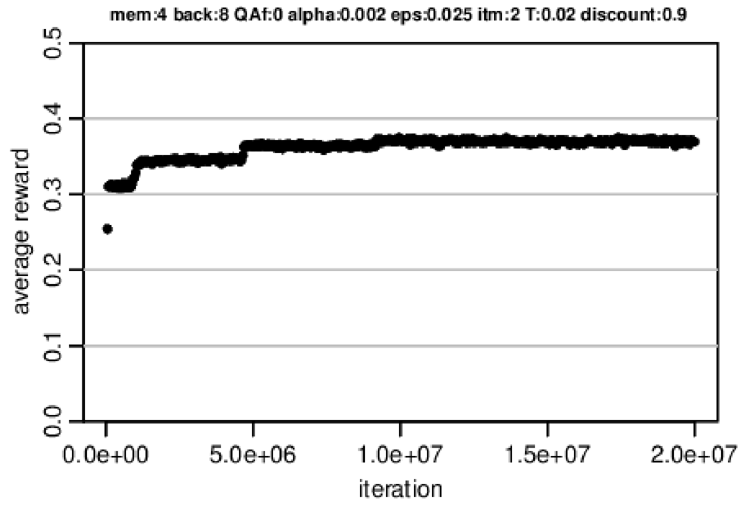
Star World: 1-Step vs. 8-Step SARSA

4-State Memory, Learns Q



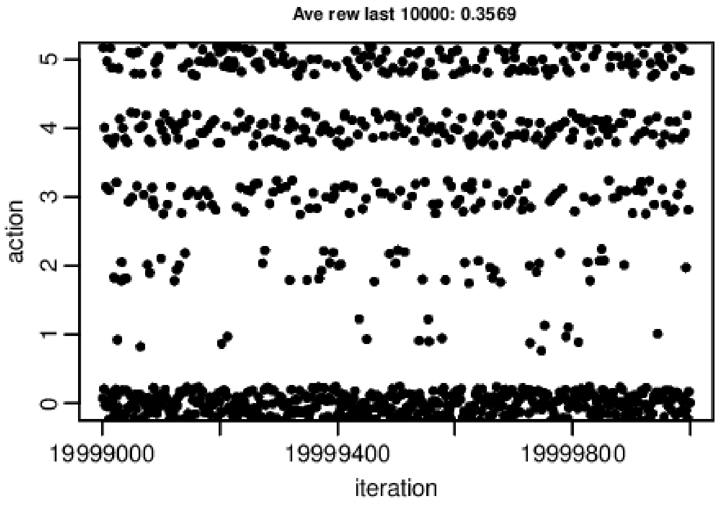


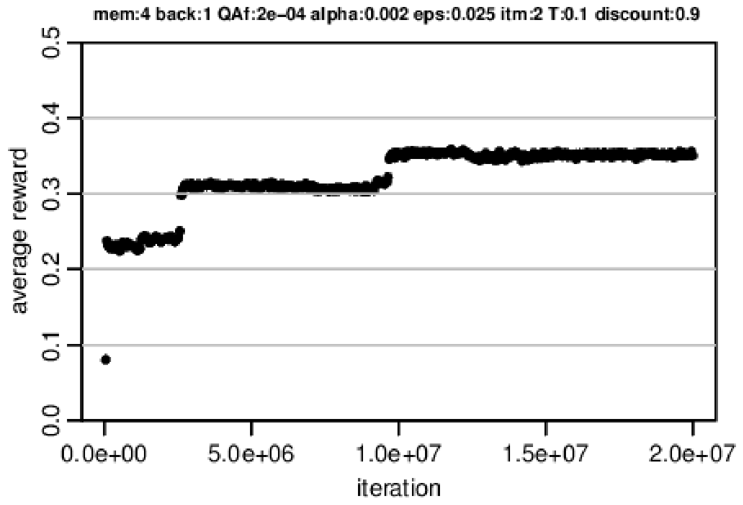


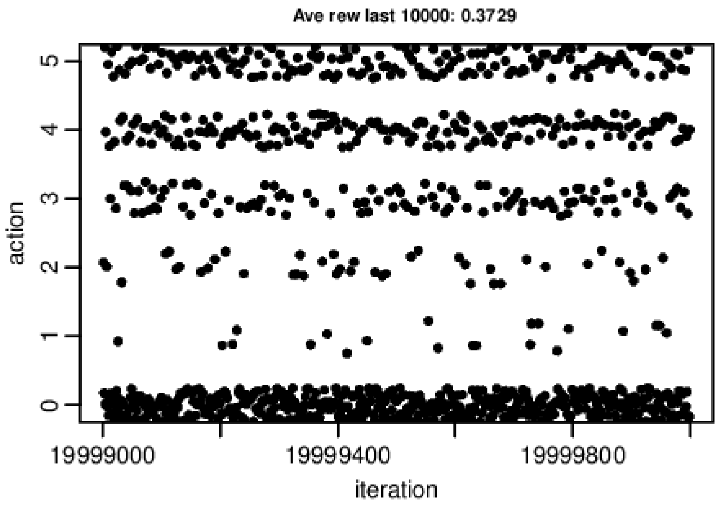


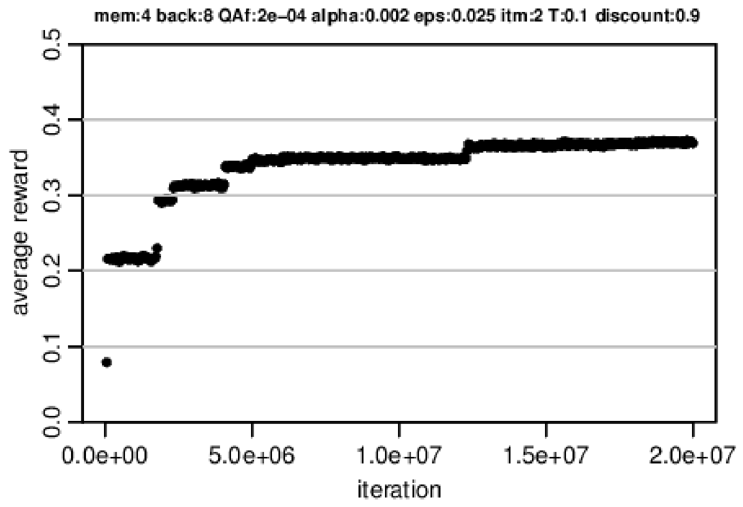
Star World: 1-Step vs. 8-Step SARSA

4-State Memory, Learns Q/A









I. What is Reinforcement Learning?

II. Learning with a Fully Observed State

III. Learning Stochastic Policies When the State is Partially Observed

IV. Learning What to Remember of Past Observations and Actions

V. Using Predictive Performance as a Surrogate Reward

Handling More Complex Problems

Problems arise in trying to apply methods like these to more complex problems:

- The sets of possible observations and/or actions are too large for tables to be a reasonable way of representing a Q or A function. Indeed, observations or actions might be real-valued.

  $\Rightarrow$ Represent Q and A functions by neural networks.

  Done in the applications to Backgammon, Atari games, and Go.

  Will need to handle large memories in a similar way.

- Rewards may be so distant from the actions that influence them that directly learning a complex method for increasing the reward probability is hopeless.

  $\Rightarrow$ Need some surrogate reward.

  Possibility: Reward success in predicting future observations.

  This might, for example, help in learning how to remember things that are also useful for obtaining actual rewards.

From an AI perspective, it's interesting to see how much an agent can learn without detailed guidance — Maps of its environment? Where it is now?

Learning What to Remember When Predicting Text

As a simple test of whether n-Step SARSA can learn what to remember to assist with predictions, I tried predicting text from "Pride and Prejudice" (space + 26 letters), using varying amounts of memory.

The reward is minus the total squared prediction error for the next symbol (sum of squared probability of wrong symbols, plus square of 1 minus probability of right symbol).

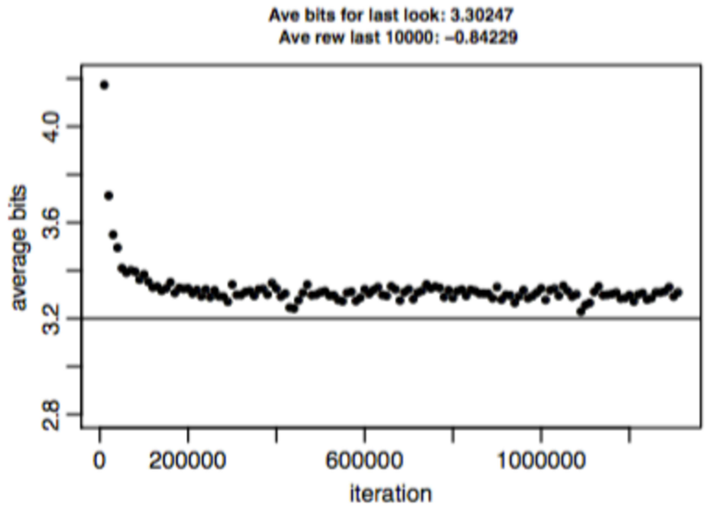
Observations are of the current symbol, plus the contents of memory.

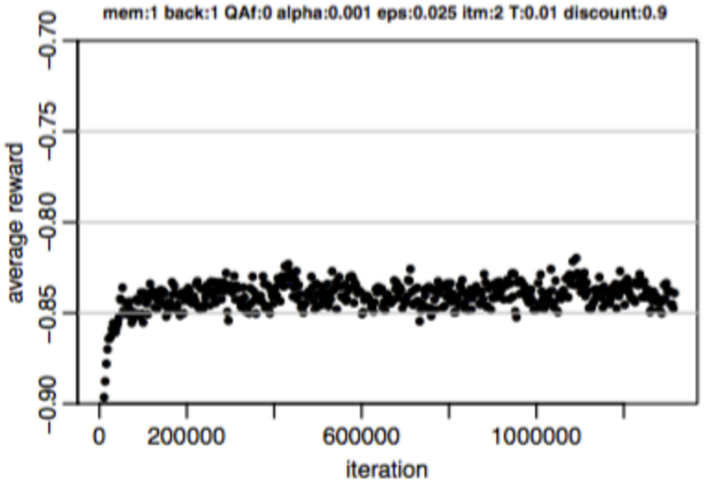
Actions are to change the memory (in any way).

With no memory, we get a first-order Markov model.

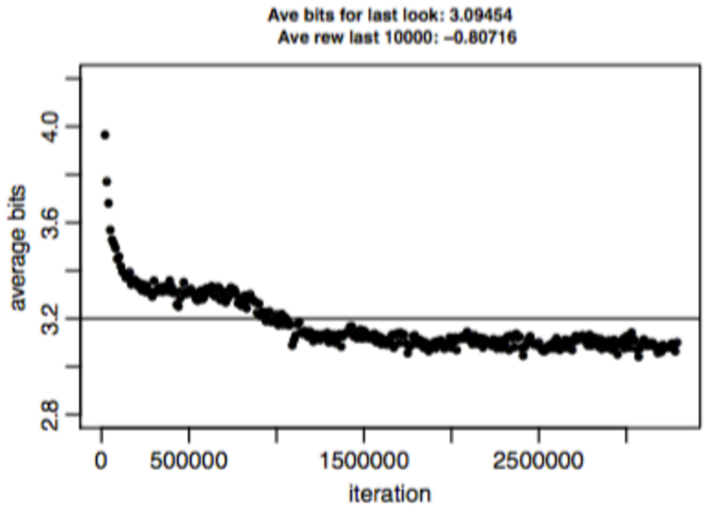
Results on Predicting Text

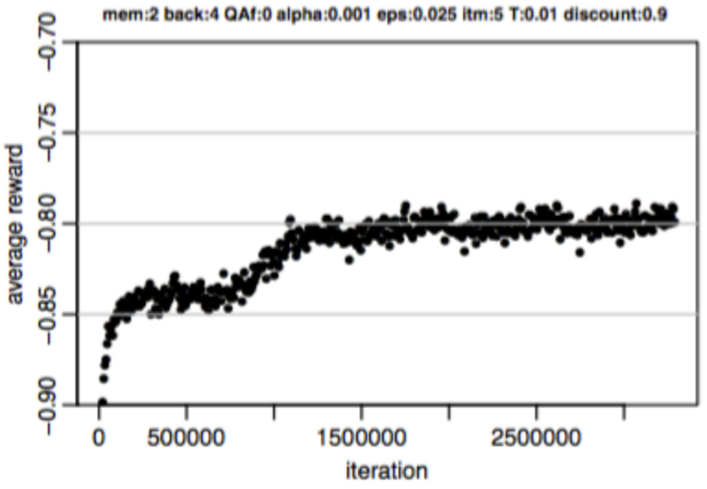
No memory, 1-Step SARSA:





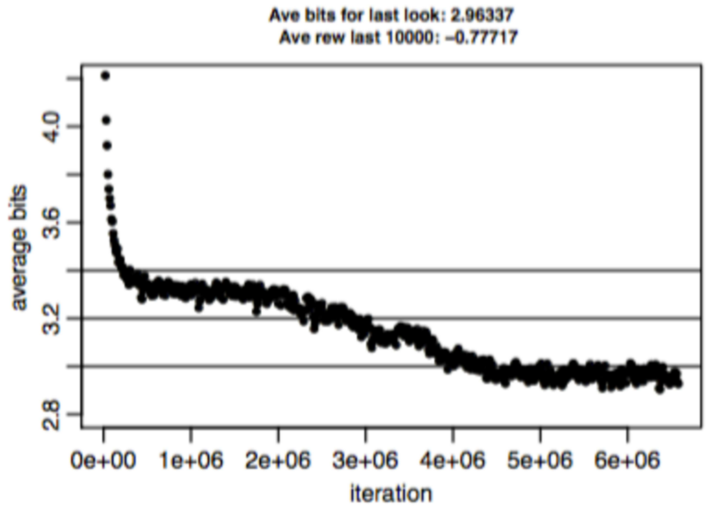
Two memory states (ie, one bit), 4-Step SARSA:

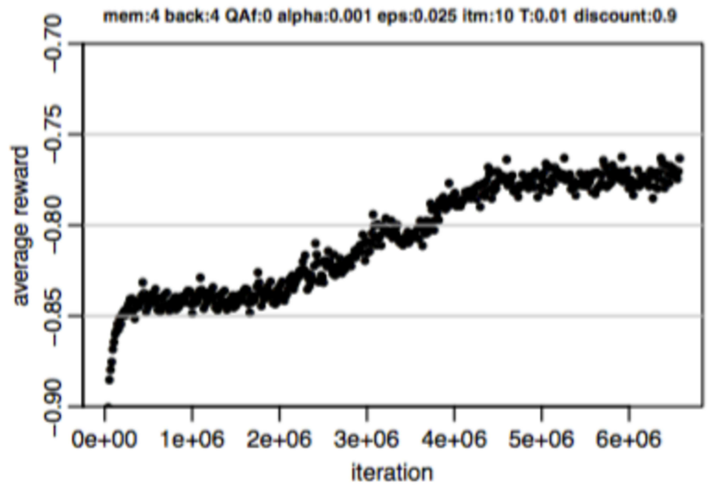




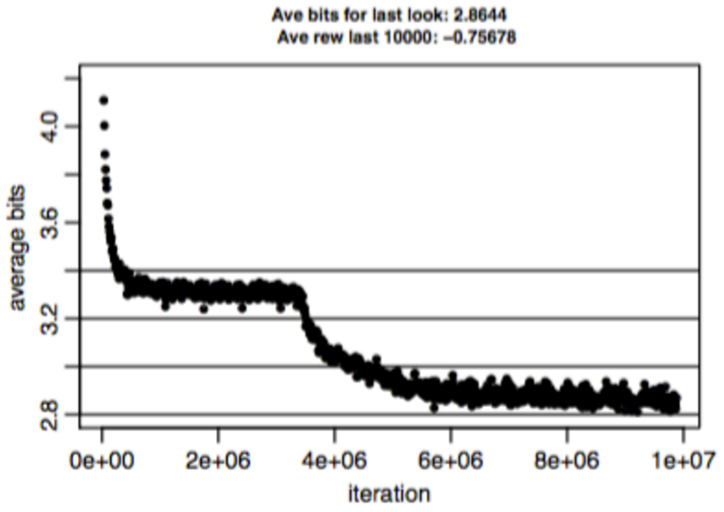
More Results on Predicting Text

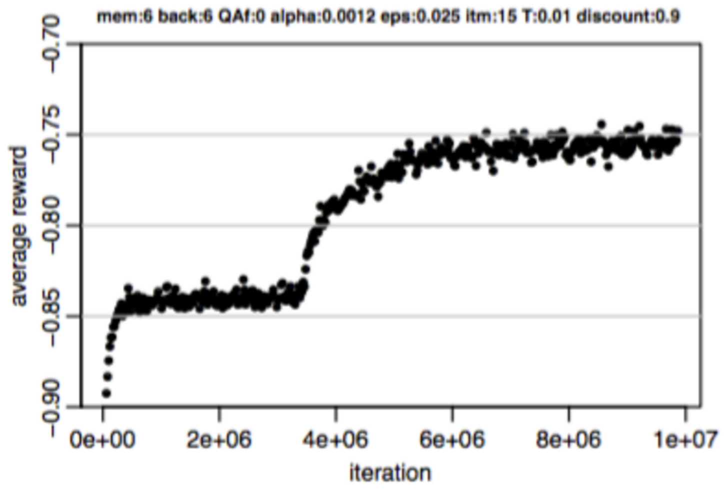
Four memory states (ie, two bits), 4-Step SARSA:





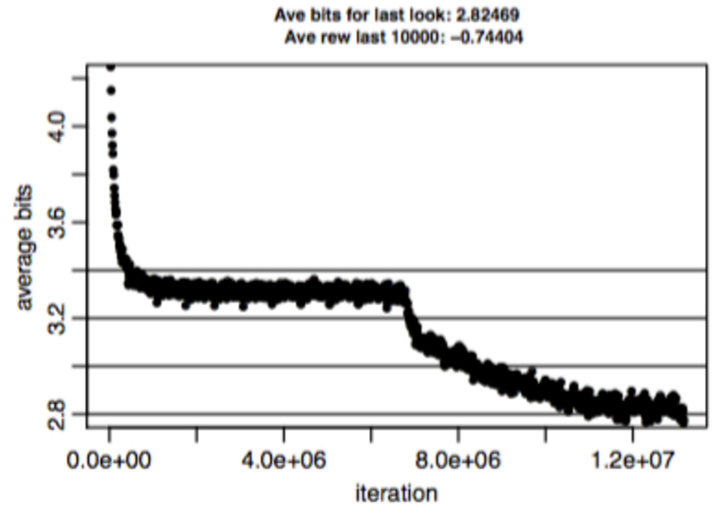
Six memory states, 6-Step SARSA,

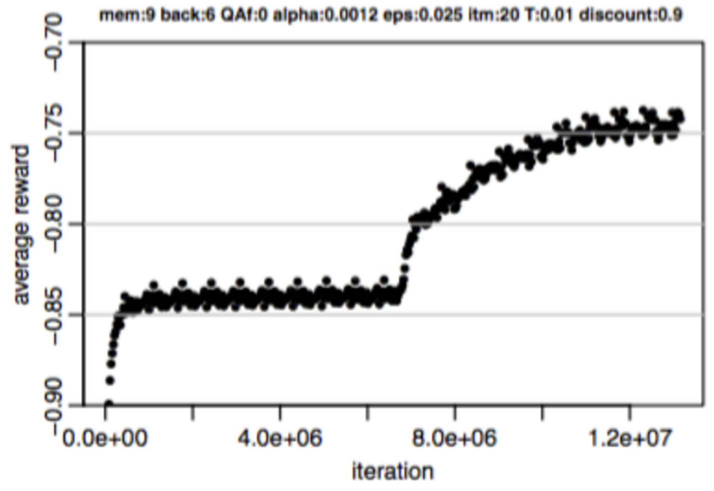




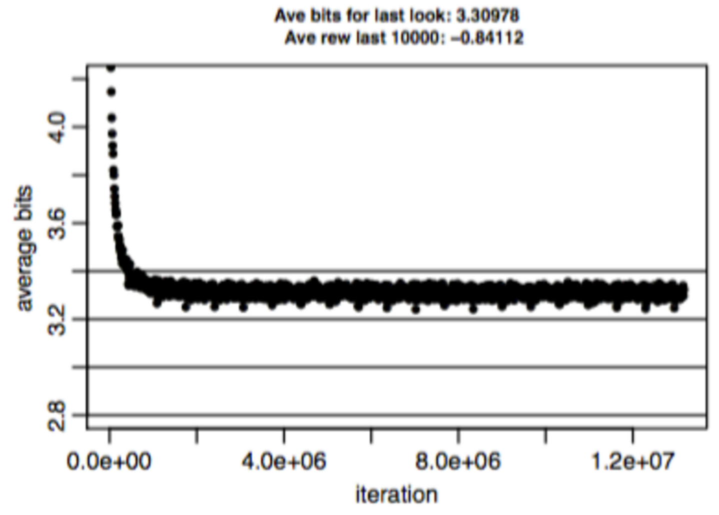
Yet More Results on Predicting Text

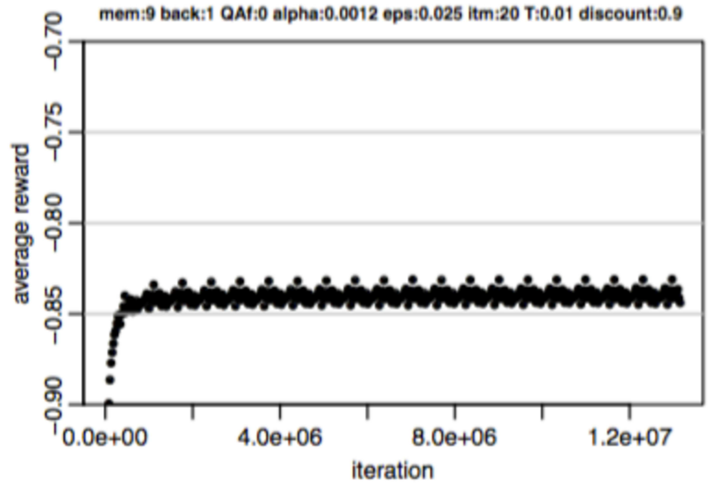
Nine memory states, 6-Step SARSA,





Nine memory states, 1-Step SARSA,





References

Littman, M. L., Sutton, R. S., Singh, S. (2002). “Predictive Representations of State”, NIPS 14.

Minh, V., et al. (2013) “Playing Atari with Deep Reinforcement Learning”, http://arxiv.org/abs/1312.5602

Peshkin, L., Meuleau, N., and Kaelbling, L. P. (1999) “Learning Policies with External Memory”, ICML 16.

Silver, D. et al. (2016) “Mastering the game of Go with deep neural networks and tree search”, Nature, 529.

Singh, S. P., Jaakola, T., and Jordan, M. I. (1994) “Learning without state-estimation in partially observable Markovian decision processes”, ICML 11.

Tesauro, G. (1995). “Temporal Difference Learning and TD-Gammon”. Communications of the ACM, 38(3).