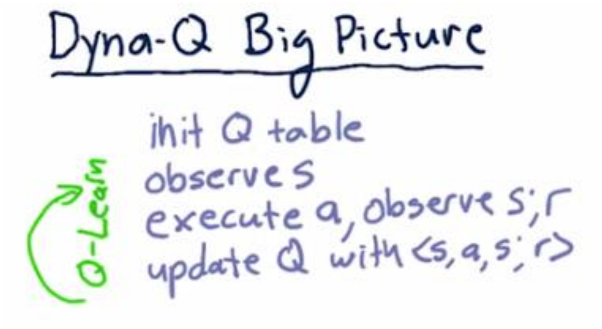
03-07 DYNA

Compiled by Shipra De, Fall 2016

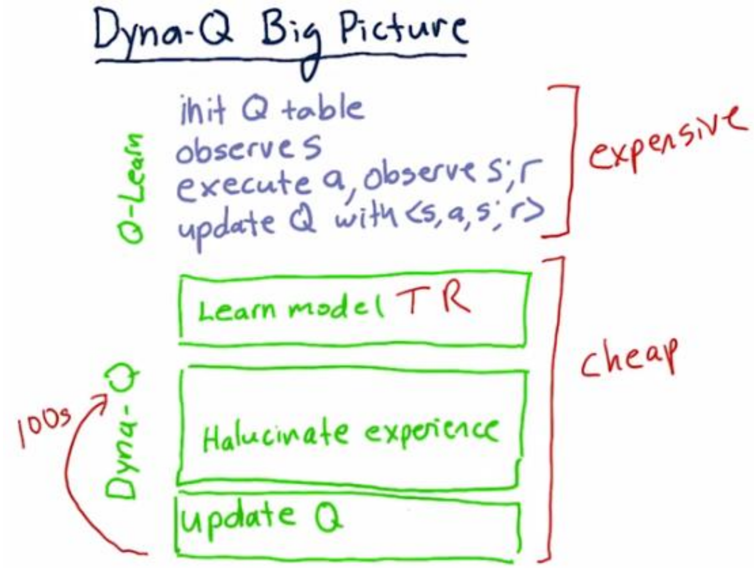
Overview

- One problem with Q learning is that it takes many experienced tuples to converge. This is expensive in terms of interacting with the world, because you have to take a real step, in other words execute a trade, in order to gather information.
- To address this problem, Rich Sutton invented Diana. Diana works by building models of T, the transition matrix and R the reward matrix. Then after each real interaction with the world, we hallucinate many additional interactions, usually a few hundred. That are used then to update the queue table.

Dyna-Q Big Picture

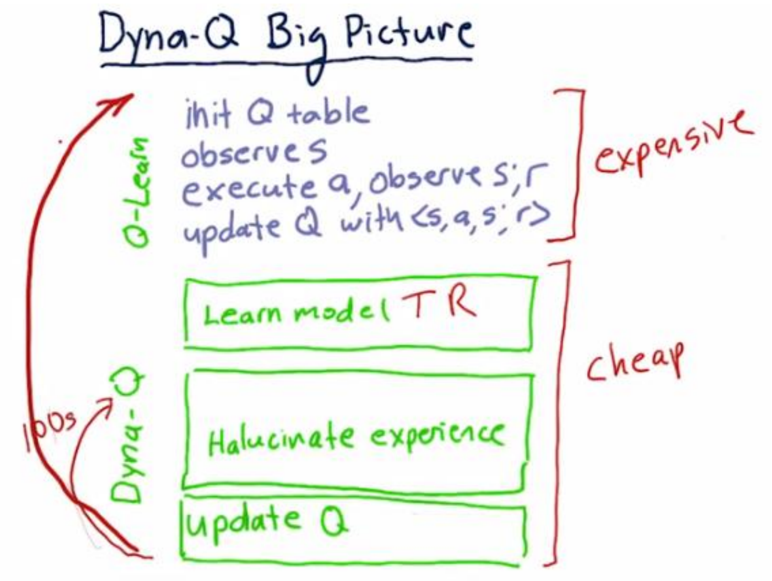


- Dyna-Q is an algorithm developed by Rich Sutton intended to speed up learning or model convergence for Q learning. Remember that Q learning is model free, meaning that it does not rely on T or R, T being our transition matrix and R being our reward function. It doesn't know it. Q learning does not know T or R.
- Dyna ends up becoming a blend of model free and model based methods. Here's how it works.
- So let's first consider plain old Q learning. Dyna is really in addition to Q learning. So let's start with the regular Q learning and then look at how we add Dyna into it.
- So we initialize our Q table and then we begin iterating. We observe s, we execute action a and then we have observe our new state s prime and our reward R within updater our Q table with this experience tubal and repeat.
- And we do that over and over and over again, interacting with the world and our Q table becomes better and better.

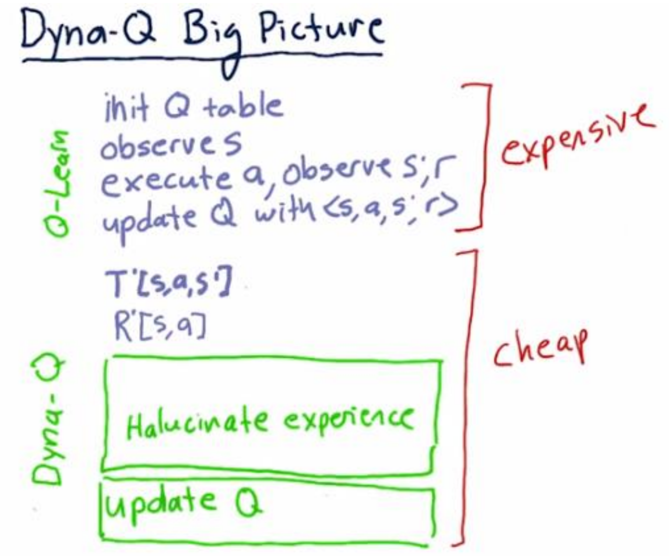


- So when we augment Q learning with Dyna-Q, e add three new components. The first is that we add some logic that enables us to learn models of T and R then for lack of a better term we halucinate an experience.

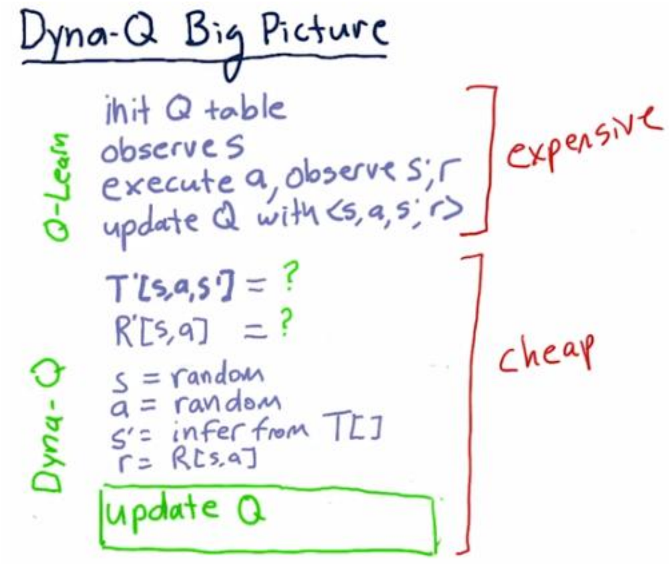
- So rather than interacting with the real world like we do appear with the Q learning part and this is expensive by the way. We hallucinate these experiences, update our queue table and repeat this many times, maybe hundreds of times.
- This operation is very cheap compared to interacting with the real world. So we can leverage the experience we gain here from an interaction with the real world, but then update our model more completely before we step out and interact with the real world again.



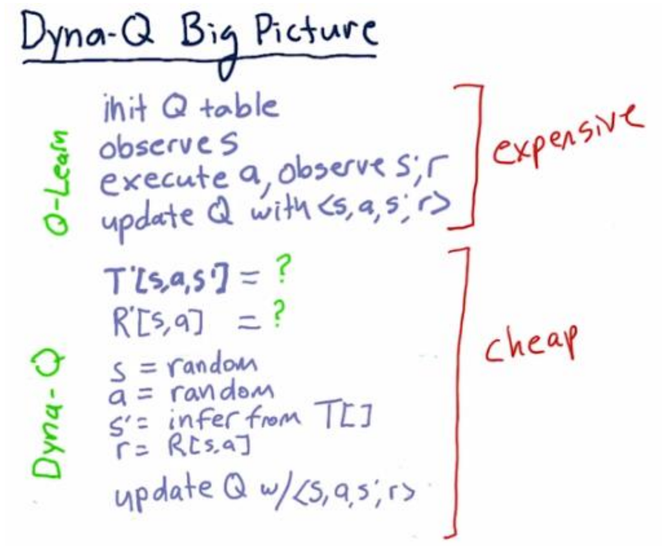
- After we've iterated enough times down here maybe 100 maybe 200 times. Then we return back up here and resume our interaction with the real world.
- The key thing here is that for each experience with the real world, we have maybe 100 or 200 updates of our model here.
- Let's look at each of these components in a little more detail now.



- So in this part where we update our model. What we really want to do is find new values for T and R. The point where we update our model includes the following. We want to update T, so I'm calling it T prime for the moment, which represents our transition matrix and we want to update our reward function.
- Now, remember that T is the probability that if we are in state s and we take action a will end up in s prime. And r is our expected reward if we are in state s and we take action a. I'm going to show you in a moment how we'll update T and R.

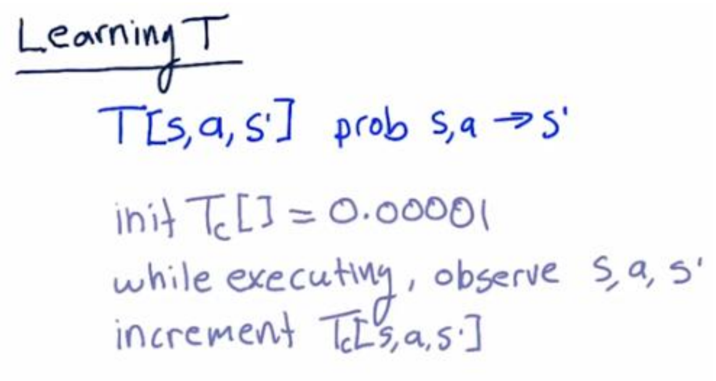


- Here's how we hallucinate an experience. First, we randomly select an s, second, we randomly select an a, so our state and action are chosen totally at random. Then we infer our new state s prime, by looking at T. And we infer a reward, our immediate reward R by looking at big R or R table.
- So, now we've got s, a, s prime and r. Or a complete experience tuple and we can update our Q table using that.



- So, the Q table update is our final step and this is really all there is to die in a queue. We've added these three sections and what's missing and I'll get to in just a moment is how do we update our model of t and our model of r?

Learning T



Learning T

T[s,a,s'] prob s,a → s'

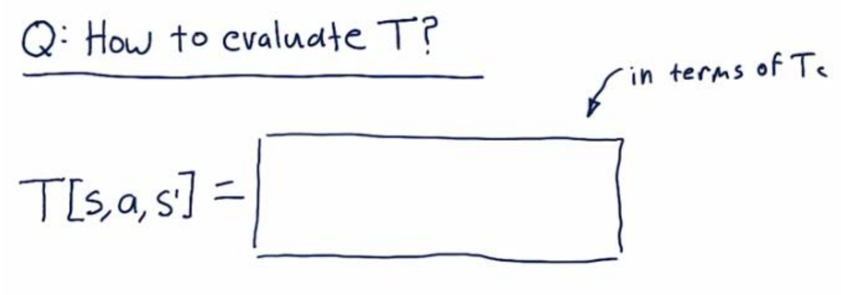
init T_c[] = 0.00001

while executing, observe s,a,s'

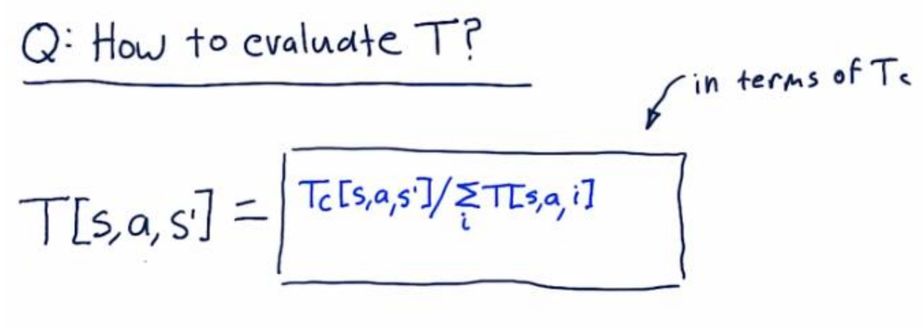
increment T_c[s,a,s']

- So what I'm about to show you are my methods for doing these things they may not correspond exactly to what Rich Sutton did. So keep that in mind but I've used these methods in practice and found them to work really well.
- Let's start with Learning T. Remember T of SA S prime represents the probability that if we are in state S, take action A we will end up in state S prime. To learn a model of T. we're just going to observe how these transitions occur. So in other words we'll have experience with the real world we'll get back on s, a, s prime and we'll just count how many times did it happen.
- So I'm going to introduce a new table I call T count or TC. And it goes like this. So we initialize all of our T count values to be a very, very small number. The reason for that is later on you'll see that if we don't do that we could get in a situation where we have a divided by zero.
- Then we begin executing Q learning. And each time we interact with the real world we observe, s, a, and s prime. And then we just increment that location in our T-count matrix.
- So every time we see it transition from S to S prime with action a, boom, we add a one. And that's pretty, it's pretty simple. That's it.

How to Evaluate T?

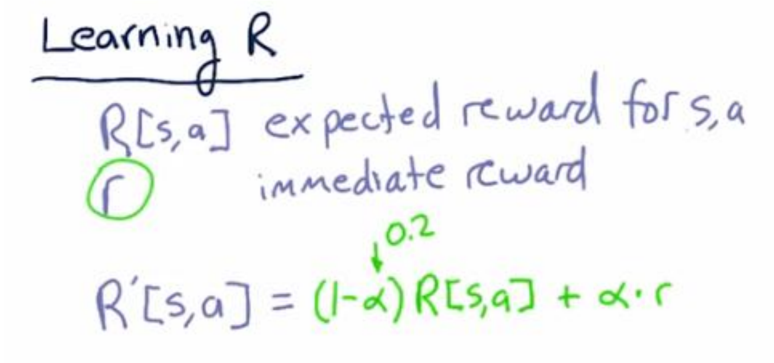


- Okay, assume now we've been watching our interactions with the real world for a while and we would like to consult our model of T. I would like for you to write an expression for T in terms of Tc or T count.
- Okay, so fill in this text box with your opinion of how we can compute T of s, a, s prime in terms of T sub c. And remember T sub c is just the number of times each one of these has occurred.



- Okay, consider that state s and a are somewhat fixed. What we're trying to find is given s and a, what's the probability of a particular s prime? So, we'll begin this by just consulting how many times did this transition occur? In other words, how many times when we were in state s did action a, did we end up in s'? So that's how many times this particular transition occurred.
- Now we just need to divide it by the total number of times we were in state s and did action a. So essentially, it's the sum over i. Where we have i iterate over all the possible states of T[s, a, i]. So this is the number of times in total that we were in state s and executed action a.
- And so this ratio ends up being the probability that will end up in s' if we're in state s and take action a.

Learning R



Learning R

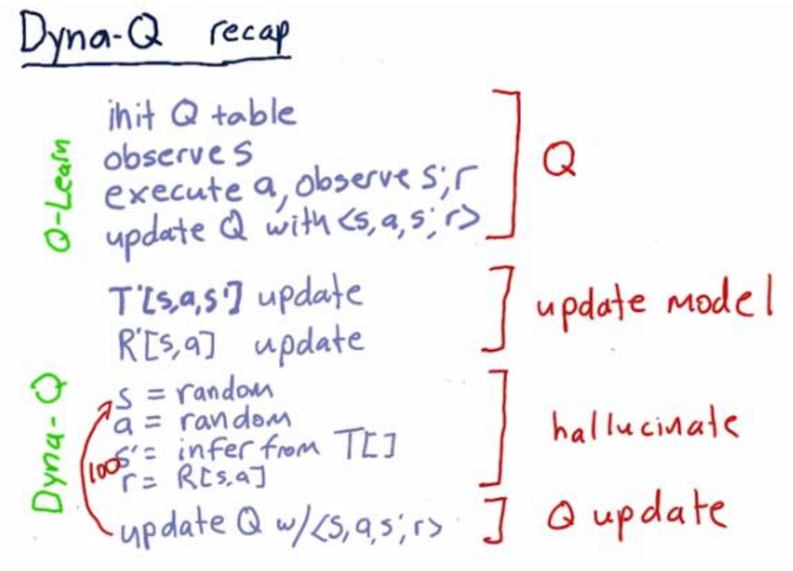
R[s,a] expected reward for s,a

r immediate reward

$$R'[s,a] = (1-\overset{0.2}{\alpha})R[s,a] + \alpha\cdot r$$

- The last step here is how do we learn a model for R? Now remember, when we execute an action a in state s, we get an immediate reward, little r. So again, big R, s, a as are expected reward if we're in state s and we execute action a. Little r is our immediate reward when we experience this in the real world. So big R is our model, little r is what we get in an experience tuple.
- So we want to update this model every time we have a real experience. And it's a simple equation, very much like the q table update equation. What we have is one minus alpha where alpha is our learning rate and again that can typically be something like zero point two.
- Anyways we multiply that times our current value for R and then we add in of course our new estimate of what that value ought to be. And we just use r for the new estimate. So it's alpha times little r, which is our immediate reward, or our new best estimate of what the value should be, plus what the value was before times 1 minus alpha.
- So we're waiting presumably. Our old value more than our new value, so we converge more slowly. But that's it. That's a simple way to build a model of R from observations of interactions with the real world.

Dyna Q Recap



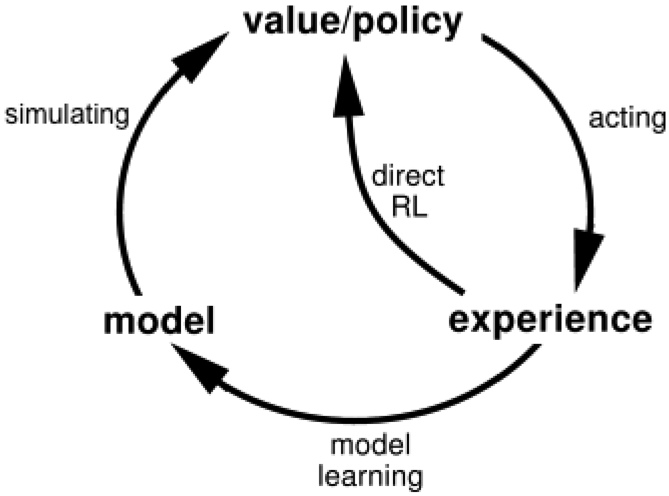
```
Dyna-Q recap
init Q table
observe s
execute a, observe s', r
update Q with <s,a,s',r>
T'[s,a,s'] update
R'[s,a] update
s = random
a = random
s' = infer from T[]
r = R[s,a]
update Q w/<s,a,s',r>
```

- Before we close, let's recap really quick how Dyna-Q works. So, we start with straight, regular Q-Learning here and then we add three new components.
- The three components are, we update models of T and R, then we hallucinate an experience and update our Q table.
- Now we may repeat this many times, in fact maybe hundreds of times, until we're happy. Usually, it's 1 or 200 here.
- Once we've completed those, we then return back up to the top and continue our interaction with the real world.
- The reason Dyna-Q is useful is that these experiences with the real world are potentially very expensive and these hallucinations can be very cheap. And when we iterate doing many of them, we update our Q table much more quickly.

Summary

The Dyna architecture consists of a combination of:

- direct reinforcement learning from real experience tuples gathered by acting in an environment,
- updating an internal model of the environment, and,
- using the model to simulate experiences.



Resources

- Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, TX, 1990. [pdf]
- Sutton and Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. [web]
- RL course by David Silver (videos, slides)
  - Lecture 8: Integrating Learning and Planning [pdf]