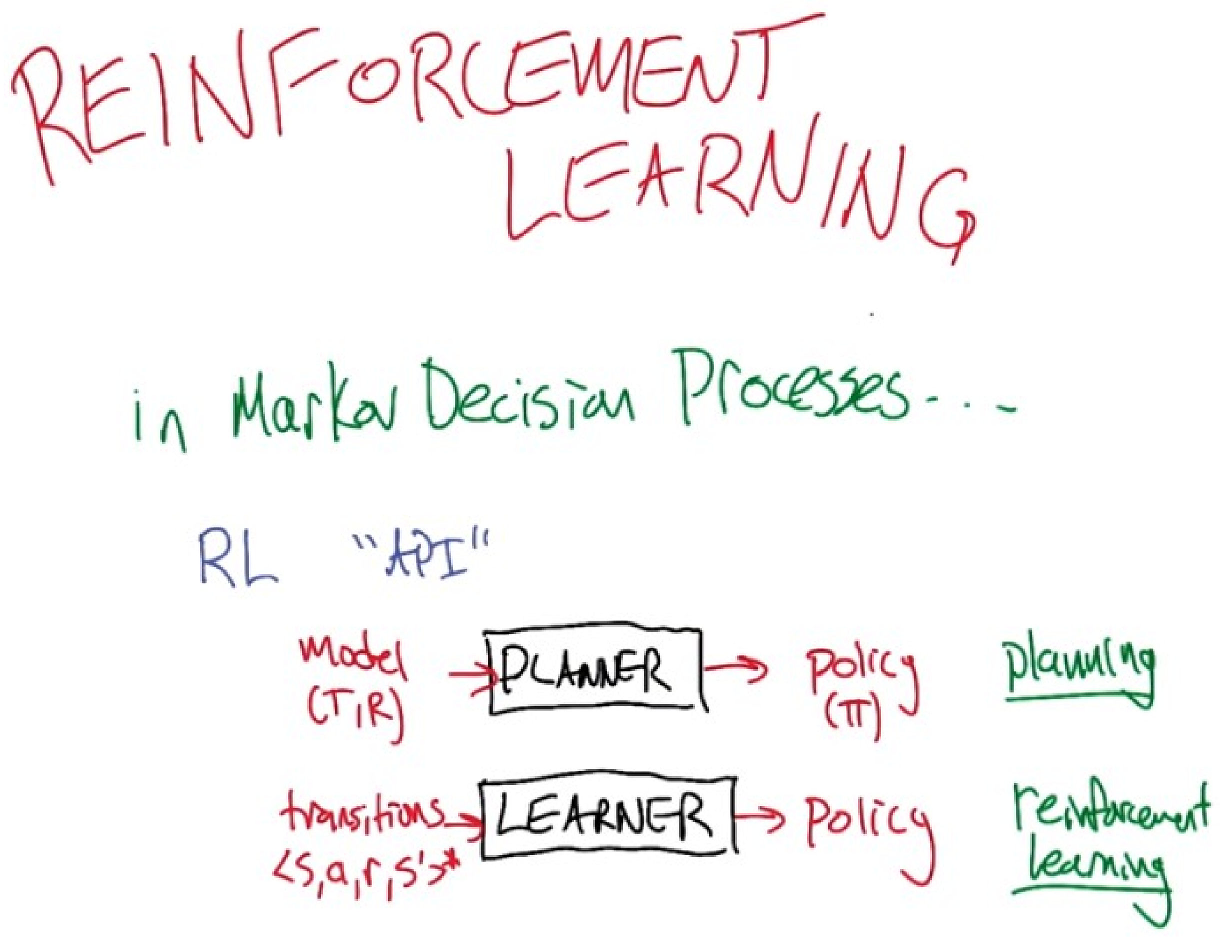
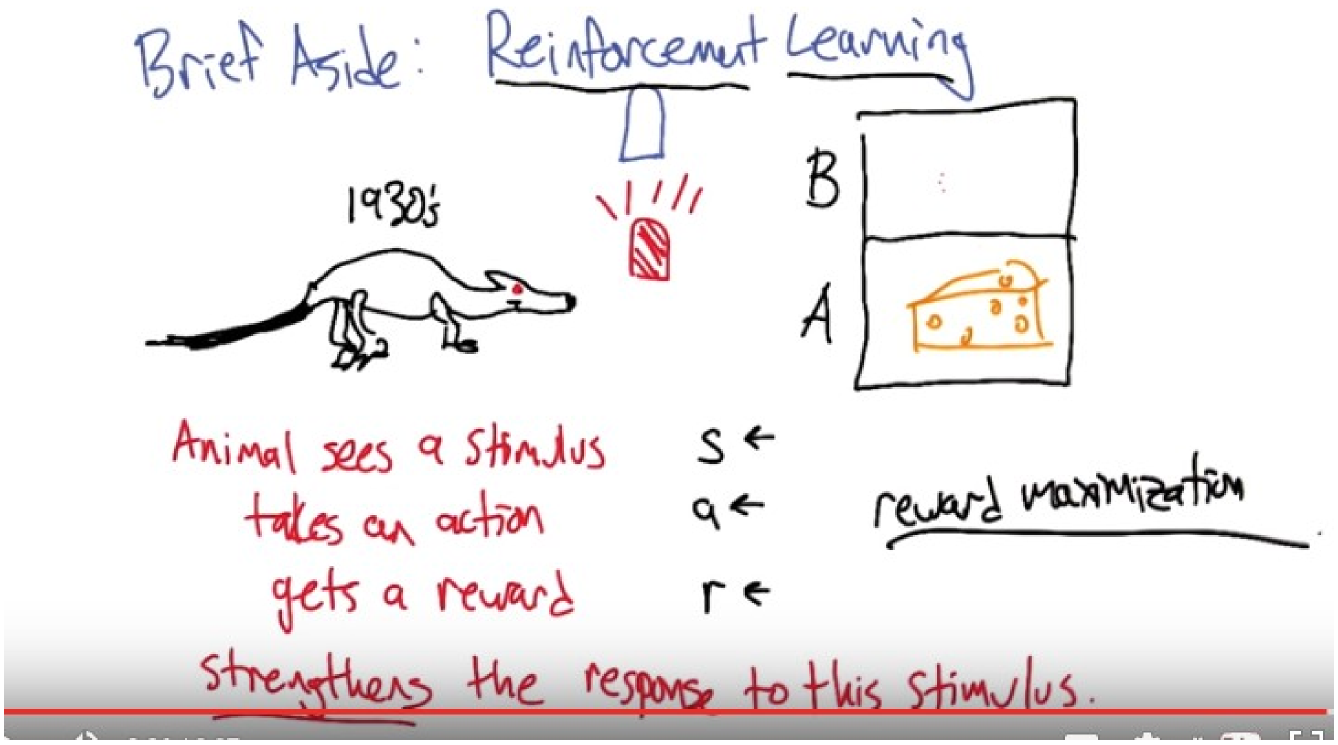


上節的 value iteration 和 policy iteration 都是 planning (後面已驗證),
本節的 Q-learning 才是 reinforcement learning

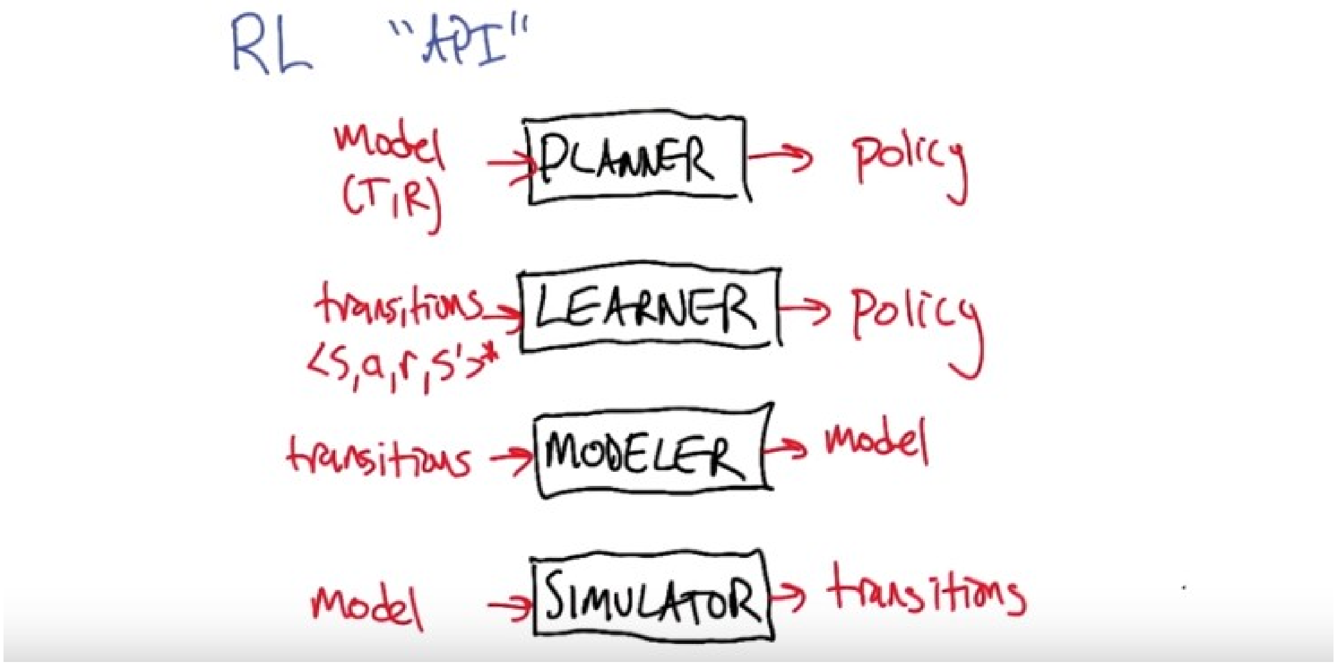
1. Hello Charles. >> Hi Michael. How are you doing? >> I'm doing okay. >> Good. >> It's you know, exciting to be getting to talk about reinforcement learning. >> Mm. Reinforcement learning. It's my favorite type of learning. >> Is that true? >> It is true. >> Wow. I like machine learning. >> I like machine learning too. But of all the kinds of machine learning, reinforcement learning is my favorite kind of learning. So, let's, how bout, how bout, giving the opportunity for the students in the class to learn about reinforcement learning by having us tell them about it. >> Oh, let's do that. You first. >> Alright. We're going to build up on the stuff you told us about last time. >> You mean the fantastic, well defined, and well formalized stuff that we talked about last time? >> Yes, it was fantastic and it laid the groundwork for what I would like to talk about. So you set up Markov decision processes, and I'm going to talk about what it means to learn in that context. >> Excellent. >> I find it useful to start off by thinking about a reinforcement learning API, like application programmer interface. So what you talked about is, is this box here. The idea of being able to take a model of an MDP, which consists of a transition function, T, and a reward function R. And it goes through some code and, you know, it comes out and a policy comes out. Right? And a policy is, is like pi. It maps states to actions. And that whole activity, what would you call that? What would you call the, the, the problem of, or the approach of taking models and turning them into policies? >> Maybe I'd call them planning. >> Yeah that's what I, that's, that's what I was hoping you would say. >> Mmhm. >> Alright, now we're going to talk

about a different set up here. We're still interested in spitting out policies, right? Figuring out how to behave to maximize reward. But a learner's going to do something different. Instead of taking a model as input, it's going to take transitions. It's going to take samples of, being in some state, taking some action, observing a reward and observing the state that is at the other end of that transition. Alright? And we'll, I put a little star on that to say well we're going to see a bunch of these transitions. >> Mm. >> And using that information we're going to instead of computing a policy, we're going to learn a policy. >> I see. So we call that learning? >> Yeah, or even reinforcement learning. >> Mm. By the way, what makes it reinforcement learning? >> That's a question. [LAUGH]. >> It's not a good question, but it's a question. >> I was, I was going to say good question, but I'll, well maybe, maybe there, it's not that it was a bad question, it's that I don't have a particularly good answer for it. So, maybe we need another slide to discuss that. >> Okay.



2. All right, so let's do a brief aside about the [LAUGH], this is like a bad history of reinforcement learning so it's not that it's a bad history, it's just badly told. So but the basic idea is this, once upon a time and we're going to call it, say the 1930s people observed. That if you put an animal, this is supposed to be a rat, in, in a, you know, box, say. And give it choices of looking in place B and place A and, say, one of them has cheese in it and the other one doesn't, but it can't see which because, you know, the doors are closed. And and let's say that we, we consistently do something like this. We turn on a red light whenever the cheese is in the A place. And we turn on a blue light whenever the cheese is in the B place. And what you observe is if you do this consistently, then what you find is that if the animal sees the stimulus like the red light going on and it takes an action like peeking inside the, the or sticking it's head inside the box A and it gets a reward which in this case would be getting to eat some cheese. That that set up strengthens its response to the stimulus. In other words, in the future when it sees the red light it's going to be more likely to go in and look in box A. And this notion of strengthening, you can think of it as reinforcing the connection. Reinforcing just means strengthening. >> Hm. >> So, this was studied for a long time and there's, there's tremendous amount of interesting research about this, but if you start to, you know, think about it as a computer scientist. A natural way of, of kind of thinking about what this problem is, is that you know, a stimulus is kind of like a state,

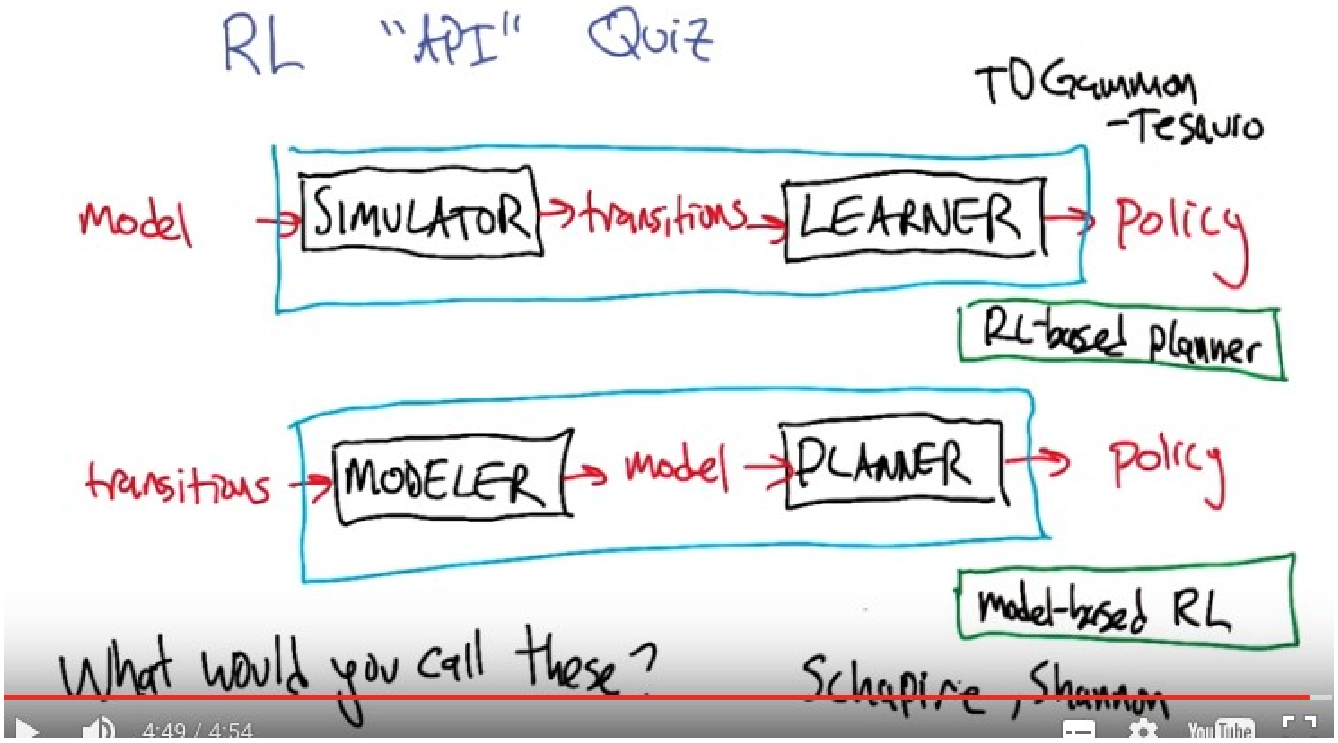
and an action is kind of like an action, and a reward, well, I kind of stole all these words but it's kind of like a reward. And it leads you to this idea that what you really want to do is figure out how to choose actions to maximize reward as a function of the state. And so we started to take this concept and we called it reinforcement learning because that's what the psychologists were calling it. But, for us it just means reward maximization. This notion of strengthening is really not part of the story for us at all. So, we're really taking this word and using it, you know, wrong. >> Hm. Well, that all makes sense except for one thing, which is the idea that this happened in the 1930s. Because we all know that rat dinosaurs did not exist in the 1930s. >> Yeah, they, well they went extinct in the 40s. >> Hm, hm. >> There's this one little epilogue to the story which I find really kind of amusing, and that is the computer scientists did pretty well at figuring out algorithms for solving problems like this. And the psychologists really do care about how stimuli and, and, motor actions and reward all Interact with each other and they've turned to the computer scientists to say, how might the brain be doing this? What, what is the problem that the brain actually might be solving? And so, guess what word [LAUGH] they borrowed back when they started to think about these things. Reinforcement. So, now they talk about reinforcement learning and they don't mean reinforcement, either. They mean reward maximization. >> So we won. >> [LAUGH] Yeah, I do not think that was really our plan but but it's, it's nice to know that we had some you know, impact. >> Well, that's very good you brought planning back into it and so they learned because of our plan, that's pretty good Michael. >> [LAUGH] All right, so that's the end of this aside. >> Okay. >> Let's go, let's go, let's go back to learning things.



3. All right, so now that we're back from our aside, let's go back to thinking about this applications program interface. So I talked about planning and I talked about learning. And it turns out that there's two other sub-processes or sub-components that might be useful to think about that kind of relate these quantities together differently. One is the notion of a modeler. What a modeler can do is take transitions and build a modeler out of it. >> That makes sense. >> And a simulator is kind of the inverse, where you can take a model and you can use it to generate transitions. You can actually kind of imagine running around in the world, Just by simulating that row. This is generally not a very hard thing to do. Though there's certainly applications of reinforcement learning where this simulator is extremely expensive. Because it's simulating a lot of things in the world. And this modeling problem you can think of again as a kind of machine learning problem. Right? Trying to map this kind of

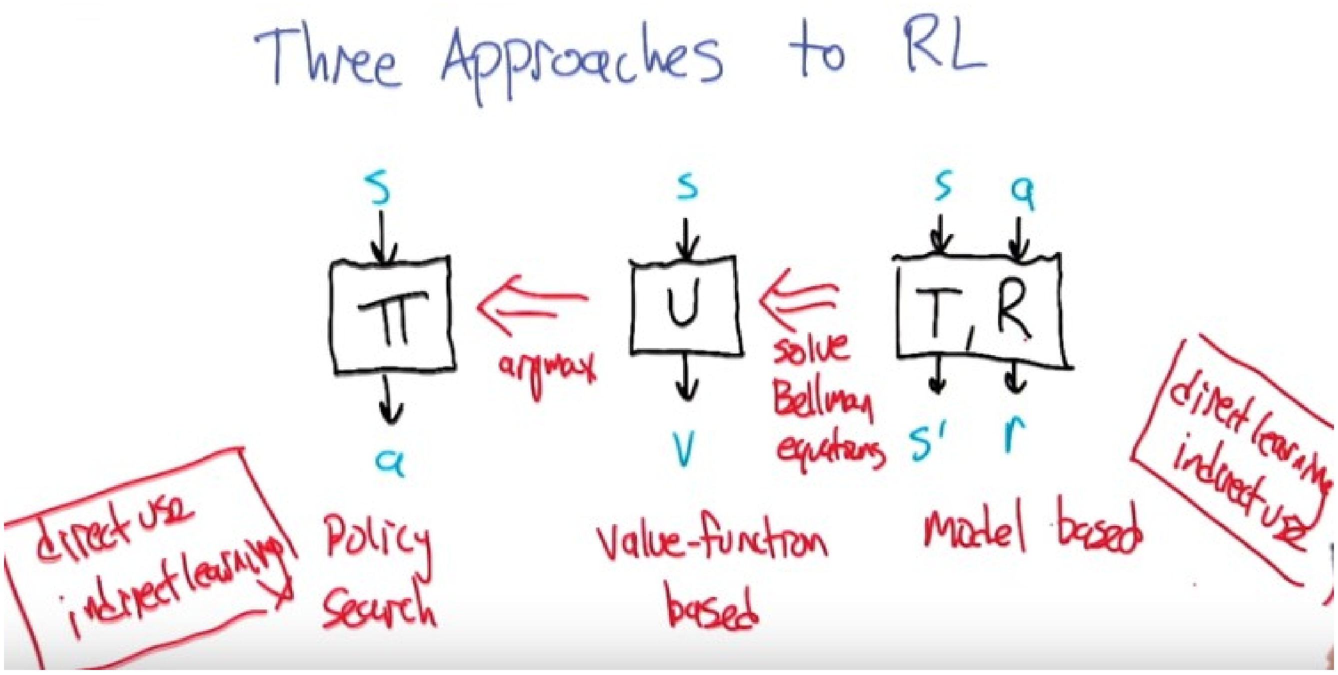
information into models. >> So you call them sub-processes. Does this mean that one way to do learning would be to do modeling and simulating so that I had models. And I knew what the reinforcement function was, and then I could just do planning? >> Yeah, that's a really good idea. In fact, let's look at different ways of gluing these things together. >> Okay.

(圖在下面)
4. So your suggestion was to (下面那個, 即 transitions->model->policty) take a modeler and use it to take turn transitions into a model, but once we have a model, you already told us last time how we can use a planner to spit out a policy. You didn't talk about those planners. What are, what's the name of the algorithms that you described last time? >> Value iteration and policy iteration. >> Yes, right. So, so what you can do is, is run either of those algorithms in here. They both have the same API right, they both take models and spit out policies. >> True. >> Alright, so, so let's do this as a quiz. We'll say, let's use your idea of mapping transitions to model to policy. And what would you call this whole box? Right? And so as a whole box, it takes transitions in and, and produces policy out. So it is solving the reinforcement learning problem. But it's taking a very particular approach to it. But let's contrast it with the, with kind of the opposite idea. Which is, we can also map a model through a simulator into transitions. And then if we have the ability to do reinforcement learning. We can use, turn those transitions into a policy. So again, as a, as a, composed system. This is turning a model into a policy, so it is a kind of planner, but it's a planner that uses a learner inside and this (下面那個, 即 transitions->model->policty) is a learner that uses a planner inside. So just, as, just out of curiosity I would like to, just ask people what they'd want to call these. I'm not going to grade these, but I just I'm just interested. Like, what would you call this approach? Does that make sense? Just type it in the box. >> Let's pretend it makes sense. >> Alright. Go.



5. >> All right Charles, so what would you, what would you call let's, let's let's think about this bottom one first. >> Okay. >> So what would you call an approach to reinforcement learning that what it does is it builds a model and then plans with it? >> A planner? >> Well, it's not a planner. I mean, the

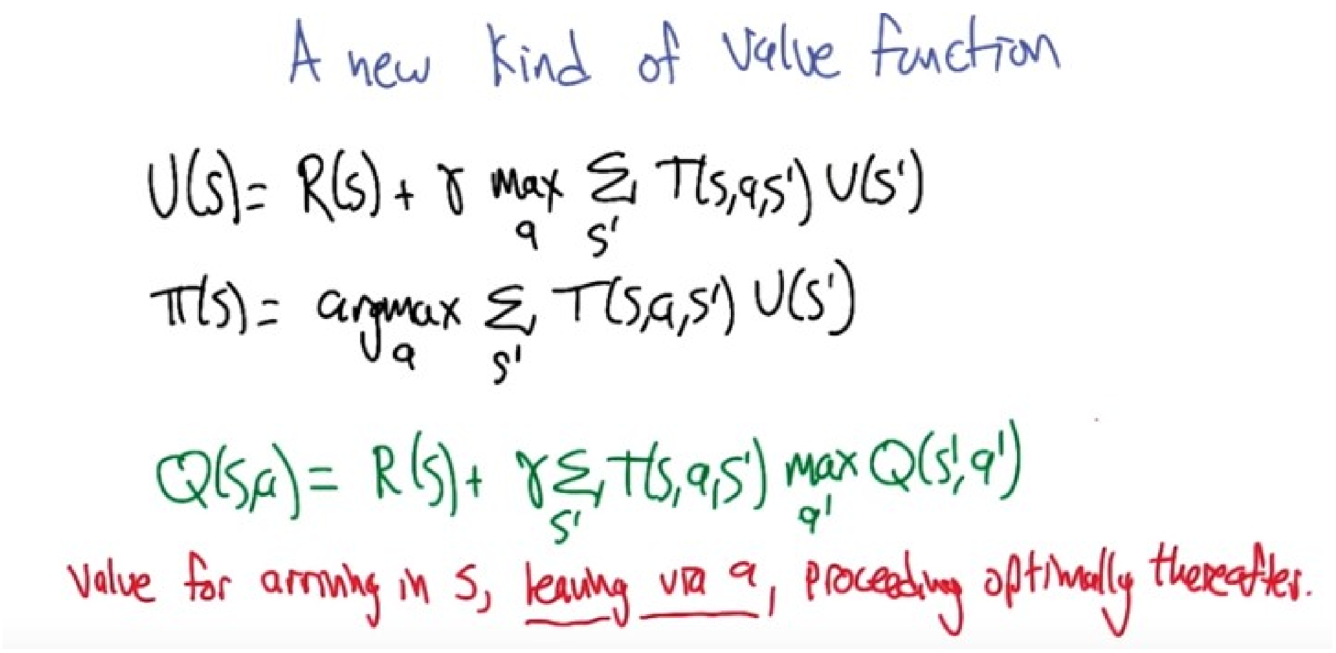
planner's inside of it. >> Sure. >> The overall system, this sort of blue box. Turns transitions into policies so it's kind of a reinforcement learner. >> Yeah. >> But it's one that builds a model inside. >> I would call that. You know what I would call that, I would call that model-based learning or model-based planning. >> Actually it's called model-based reinforcement learning. >> Mm-hm. >> So this is, this is, in fact, my, you said, reinforcement learning is your favorite kind of learning? >> Mm-hm. >> Model-based reinforcement learning is my favorite kind of of reinforcement learning. >> Mm. >> But we're not going to get to talk about it very much, so I wanted to at least have this slide to give people a chance to, at least, you know, these are all pieces that you can imagine doing. Right? This, this piece you can think of as being what we did in, when we're talking about supervised learning. And this piece is what you talked about last time. So you know, you could build a model based reinforcement learner that way. >> Yeah, that makes sense. >> Alright, how about this other idea (the top one), where, where you start off with a model and then you pretend you don't have a model, you pretend you're just in a learning situation by turning that model into, into transitions just by simulating them. Then the learner interacts with the simulator. And then spits out a policy. >> Well, I could come up with one of two answers. >> Okay. >> So I could try to do pattern matching on the answer to the second one. And since that one's model based, then this one's transition based, which is kind of cute. Or I could say, well one difference between at least inside the kind of blue box is it's sort of a model free reinforcement module. because the learner does not ever get to see the model. >> This is true, the learning piece is model free but we're using model free learner in service of planning. >> Okay. >> So, I don't know, I don't have a good this like model free planner, or an RL-based planner, maybe. Like I said, I wasn't going to grade it, I just was kind of interested to see what, what you'd say. >> Hm. Well I, I was pretty, I felt pretty good about the model-based RL one. >> Yeah, well this is the actual term that's used in the field. I'm, I'm sure there are some terms that are used here but nothing, nothing really very consistently. >> Hm. What if I called it the blue box planner? >> You could call it that, and, but had I, had I drawn it with a black box, it would, it had a better name. >> mm, oo, that would have been really cool, black box planner. I like that. Okay. >> But but I want to point out that one of the most successful applications of reinforcement learning are least most celebrated was Jerry Tassara's backgammon playing program, which used exactly this approach. Backgammon is a board game. So we have a complete model of it, but we don't really know how to plan it, it's a very complex, very large state space. So what he did is, he actually used a simulator backgammon, to generate transitions, and then used reinforcement learning approach TD, to, to create a policy for that. So it, his TD Gammon system followed exactly this, this overall pattern. >> Yeah, it was very influential work and generated many mildly embarrassing Master's theses. >> [LAUGH] I can only think of one. Oh, actually I can think, that's not true, I can think of two. >> Oh really? >> Well, which one are you, you're thinking of yours. >> Yes, I'm thinking of mine. I try not to think about it. >> The, the other one is, Justin Boyin, who's a good friend. And, a highly influential reinforcement learning contributor, and now Googler. Who, his Masters thesis was also, it's basically another TD. He did backgammon with RL. >> Oh, I didn't realize that. I, I actually like him. I think he's very cool. [LAUGH]. So, I'm sure that, his was not mildly embarrassing. >> I mean, you know, everybody's master's thesis are mildly embarrassing because you're, you're struggling to learn about how to talk about research, and so you're not going to get it perfect. >> Yeah, and in my case, it was it definitely model free. >> Right, the only non-embarrassing master's thesis that I'm aware of. >> Shanon's. >> Oh sure, alright, well Shannon. What was his master's thesis? >> It was information theory. >> Oh, yeah. It's pretty impressive. >> Yeah. >> Rob Schapire's master's thesis is pretty cool. >> I have no doubt. >> The boosting guy. He did, he did pom de pe learning. >> For his master's thesis? >> Yeah. With the diversity representation. >> I am a terrible human being. [LAUGH]. I mean wow. I mean it is true though, you look at someone like Shannon and you just think to yourself, oh, for his Master's thesis he did, he invented information theory. [LAUGH] You know, like wow, I wonder what he did for his PhD thesis? You know what he did for his PhD thesis? Nobody knows, nobody cares, because he based it on. [LAUGH] >> He, he had already done the information theories. They're like here, fill out a piece of paper, you can pick up a PHD. He did something on AI, I can't remember what it was, but it was, it was totally unimportant and nobody cares about it. But information theory? Yeah. [LAUGH]. >> Thanks. >> All right, well, yeah, okay. [LAUGH]. >> II think, I think Schapire's PHD was boosting. >> Thanks Rob. Make us all look bad, why don't you.



上面的三個方框沒有從右向左的關係, 它們都是互相獨立的方法. 後面要討論的, 是中間那個方法(即 U).

6. Alright, we're going to drill down and talk about a specific reinforcement learning algorithm in just a moment. But I wanted to remind everybody about, some of the quantities that Charles introduced in the lesson last time when he was talking about MDP's, and use them to describe three different approaches for solving reinforcement learning problems. So this first box here pi, maps states to actions, and what did you call this Charles? >> A policy. >> That's right. And reinforcement learning algorithms that work directly on trying to find the policy, are called policy search algorithms. So the good thing about policy search is, you're learning the quantity that you directly need to use. Right? You're learning the policy. That's supposed to be the output. So that seems like a really good thing. Unfortunately, learning this, this function is very indirect. We don't get direct access to, I was in this state, what actions should I have chosen? Right? So this is, what did you call this, Charles, the the temporal credit assignment problem, right? >> Right. >> So, the data doesn't tell you what action to actually choose. >> Right. And this is why it's not exactly like the way we've formulated supervised learning in the past. >> Well, at least if we're trying to do policy certs, that's right. But what we're going to do is now consider, well, maybe that's not the quantity that we want to learn. Let's, let's think about learning this function U, which you had said maps states to values. So, what was this guy? >> U is a utility. >> Yeah. >> Yeah the true utility of the state, sometimes I, I think I called it the, the value of the state. >> Yeah so, so, so sometimes it's referred to as a value function, and learning methods, reinforcement learning methods that target that as, as what they're trying to learn directly, are called value function based approaches. >> Mm-hm. >> And, let's say that we, that we try to learn this, we're trying to learn to map states to values, well the good news is, at least if we're acting in the world, we're getting to see, okay I was in some state, I took some action, duh, duh, duh, and I can, I can observe the values that actually result from that, and maybe use that to, to make updates. So you can kind of imagine learning this, so that the,

the learning's not quite as indirect. How do we use this to decide how to behave? So we, we need to turn it into a policy. >> Mm hm. >> And we didn't quite, we sort of talked about how to do that, in terms of, of looking at the Bellman Equations. We're going to, we're going to, it turns out it's actually kind of hard to use U directly to do that. But we're going to talk about a different form of the value function that's going to make that easy. So this is actually, it turns out it'll be a relatively easy kind of arc max operation, once we have the right kind of value function. >> Okay. >> So, there's some computation we have to do, and there's some indirectness to the learning, but, you know, it's okay. Alright, so the other quantities that you told us about were T, which is, I think of it as the transition function, I think you had a different name for it. >> I just call it the transition model. >> The, the model, right, yeah. So this is the model, and and the, the R is the reinforcement function, the reward function. >> Mm-hm. >> And what do these things do, they take states and actions and the transition function, or the transition model, predicts next states, maybe probabilistically, and the reward function tells you, next rewards, but just rewards. And so this is a model, jointly, and I already mentioned this, but we call methods that learn this quantity, learn these, these functions and then use them to do decisions, model based reinforcement learners, so how do we go from T and R to something like U? >> Well if we had those, if we had T and we had R, and we could see the S's, and we, and the actions we took, then we could do you know, something like value iteration we did before the learned values. >> Right, which value iteration was used to solve the Bellman Equations. >> Right. >> So that is somewhat heavy weight computation to do, but it's doable. So, what would happens over here, is we actually have fairly direct learning. Why is that? Because when we're trying to learn the transition and reward function, we get state action pairs as input. And then when we receive next state reward pairs as output, so we can solve this as a supervised learning problem. >> Hm. >> So, so learning is rather direct, the usage of this is a little bit computationally complex because you actually have to do the planning and then the optimizing to actually develop what you're, you know the policy doesn't come directly out of that. But but this kind of gives you a sense of three different ways, three different ways, places that you can target the pieces of the MVP so that you can do reinforcement learning. >> I like that. >> Now, we're going to focus on this, this middle piece. Partly because, you know, it's kind of the Goldilocks situation. It's you know, the learning's not so indirect and the usage is not so indirect. There's just been a tremendous amount of work focused on, on value function based approaches. And it's, you know, remarkably simple it turns out, if you do it right. And also very powerful. That there's lots of ways to use these simple ideas to, to learn hard problems. So, I think this is a good place to focus. >> Okay. I buy that.



A new kind of value function

$$U(s) = R(s) + \gamma \max_{a} \sum_{s'} T(s,a,s')\, U(s')$$

$$\pi(s) = \arg\max_{a} \sum_{s'} T(s,a,s')\, U(s')$$

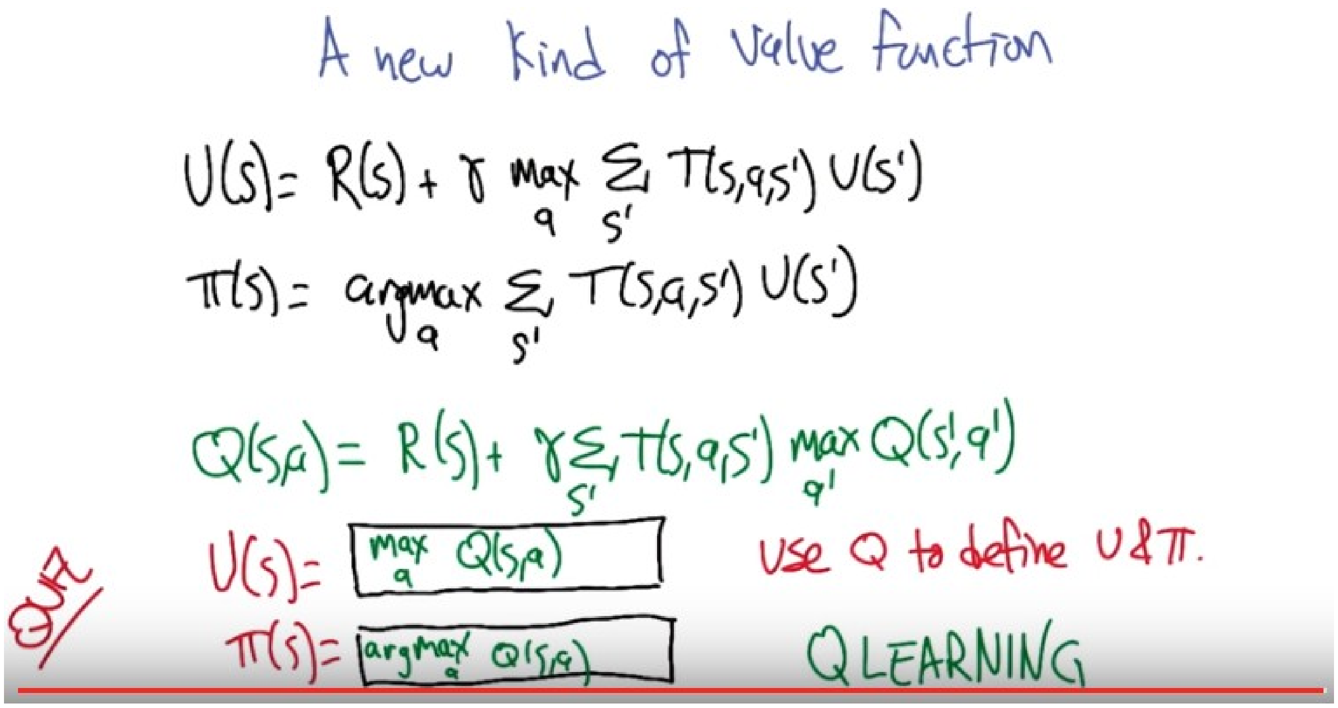
$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

Value for arriving in S, leaving via a, proceeding optimally thereafter.

上圖中黑字是之前的定義, 藍字是新定義

7. >> So lets talk about a new kind of value function, that's going to actually make that optimization part easier and the learning part easier. So but it's really closely related to the stuff you talked about, Charles, so let's let's start with what you told us. Which is here's a definition of, of you. This is, we're going to define you. >> Mm. >> And put you in a box. >> It's been tried before. >> It can't be done. All right, so well, you know, if you, if you're a good dog, I'll give you some pie next. So U, >> [LAUGH] >> [LAUGH] U is defined for each state, the utility being the state, this is, the long-term value of being in a state is the reward for arriving in that state, plus the discounted reward of the future, and what's going to happen in the future? Well, in the future, to leave this state we're going to choose some action, then we're going to take an expectation over all possible next states. And we're going to arrive in some next state, S prime, and then U is representing the utility of that as well. So this is, is recursive and nonlinear, but we know how to solve this, we can use things like value iteration to do that. Agreed? >> Agreed. >> Alright, and you also said, here's how you can use this quantity to decide what to do. That the policy in a state S is, well, let's consider all of the actions we can take to leave that state. We'll look to what their expected values are, so we'll iterate over all the possible next states weighted by their probability of the utility of landing in the state that we'd end up in. >> Mm-hm. >> And that, that tells us how to behave. >> Yes. >> Alright, so these are, these are the value functions, well the value function of the policy that we talked about before. Here's our new kind of value function. It is sometimes called the Q function. Though, you know, some people in the know don't like that. It's, it's called the Q function because it's the letter Q. But it's, some people have said that it stands for quality but it's just, it's just a letter in the latter half of the alphabet, you know, V was taken, U was taken, you know, W is used for weights, like, Q was available. So, it was brought to bear. So, so this is, this is a, a new definition and you can see it's got elements of the other two. Let me maybe write down what it, it, a way to interpret this. >> Okay. >> So, again, here's the, the Q function. And what we're going to think of this as, is, this is the value for arriving in some state, S. And this is, you know, this is the reward we get for, for that arrival. Then what we're going to do is we're going to leave S via action A. So we're going to add to that the discounted expected value that we get for taking action A. It's going to now drop us to some next state S prime, according to this probability, and once we land in S

prime, we're going to take whichever action has the highest Q value from there, okay? So that turned out to be the value for arriving in S, leaving via A, and proceeding optimally thereafter. Like once, once we get to a new state, we're going to be choosing the best actions each time. Does that work for you? >> It makes sense right, because you could, U is basically defined the same way. It's the value for arriving in S and then proceeding optimally thereafter. >> That's right. And all the, the only thing that we're doing here is we're, we're basically tying the algorithm's hands briefly. We're saying, we're going to force you to leave via action A. After that, do the normal thing. But just for a moment I'd like you to just take action A. >> Mm-hm. Okay, that makes sense. So you just, you, you inserted basically a, a kind of, kind of utility step in there that I assume you're about to use in some clever way. >> Indeed, yes, this is going to turn out to be really helpful, because it's going to allow us to compare the values of different actions without having to actually stare at the model to do it.



A new kind of value function

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')\, U(s')$$

$$\pi(s) = \operatorname{argmax}_a \sum_{s'} T(s,a,s')\, U(s')$$

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

$$U(s) = \boxed{\max_a Q(s,a)}$$

use Q to define U & π.

$$\pi(s) = \boxed{\operatorname{argmax}_a Q(s,a)}$$

QUIZ

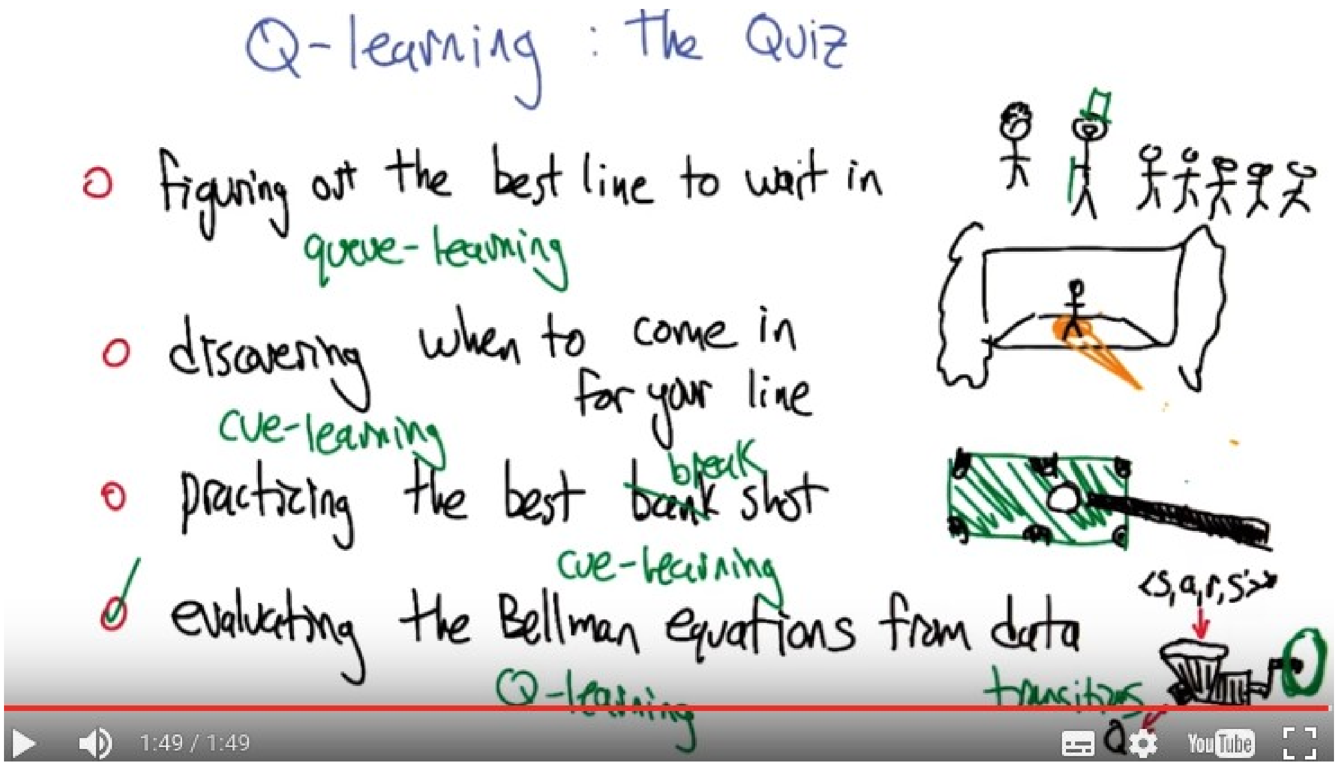
Q LEARNING

上圖的目的就是想說明 Q gives us everything we need.

8. >> Alright, so I just introduced this Q function. It turns out that baked into this one little Q function is everything we need for, for dealing with U and pi without knowing the transition function T or the reward function R. And to you know, to demonstrate this for you, I'm going to let you demonstrate it to me. [LAUGH] So, [LAUGH] here's what I'd like you to do. I'd like you to rewrite these equations, this U and the pi equation, in the little boxes, using Q instead of any references to T and R, alright? So, imagine that we have Q, that somebody's already solved this out for us. How can we define pi and U using Q? Does that seem okay? >> Seems okay to me. I think I might even know the answer. >> Alright, that would be awesome. So let's, let's give people a chance to think about it and then just tell me what you think. >> Okay.

9. Alright, Charles [LAUGH], let's, let's get back to the task at hand, and you thought maybe you had some ideas. So, tell, tell me, how can we define U and pi in terms of Q? >> Okay. Well, so I guess the first thing to observe is that U is a value. Right? >> U are a value. Okay. Yes. U is a value. That's correct. >> [LAUGH] U returns a number. A scalar, in particular, where pi returns an action. >> Yeah, U of s returns a scalar, that's right, and pi of s returns an action, very good. >> Right, okay. And, you

know, we have the definition for U up there near the top, and it's just the reward and then sort of behaving optimally thereafter, where Q is a reward and then taking a specific action, and and behaving optimally there after. So, we can sort of turn Q into U, if we always pick the best action. And we know the best action is, it's the one that, you know maximizes the value that you're going to get from that point on. So, I would say that U of s could be defined as Max over a of Q, s comma a. >> Simplicity itself. >> Mm-hm. >> Nicely done. I have no idea how we're going to grade that automatically, because it's hard to type a's under other a's. But, yeah, that's exactly right, that we have the maximization over all possible actions of the Q value in that state. Like, that's just great! >> Mm-hm >> How about the policy? >> So the policy will look exactly the same, the, the, the policy that you want to follow anyway is the one that maximizes your value going forward except the differences. It's returning an A and not an actual value, so it should be argmax. >> Oh, So it's not exactly the same. >> Right. >> It's almost exactly the same. >> Rr dmax. >> Right, so this is, it's just so trivial, so, so Q gives us everything we need. If only we had a good way of finding Q. >> Yeah, if only, if only we could find a Q, a Q for you. >> [LAUGH] and that's going to be the essense of Q LEARNING. >> Oh, well done.

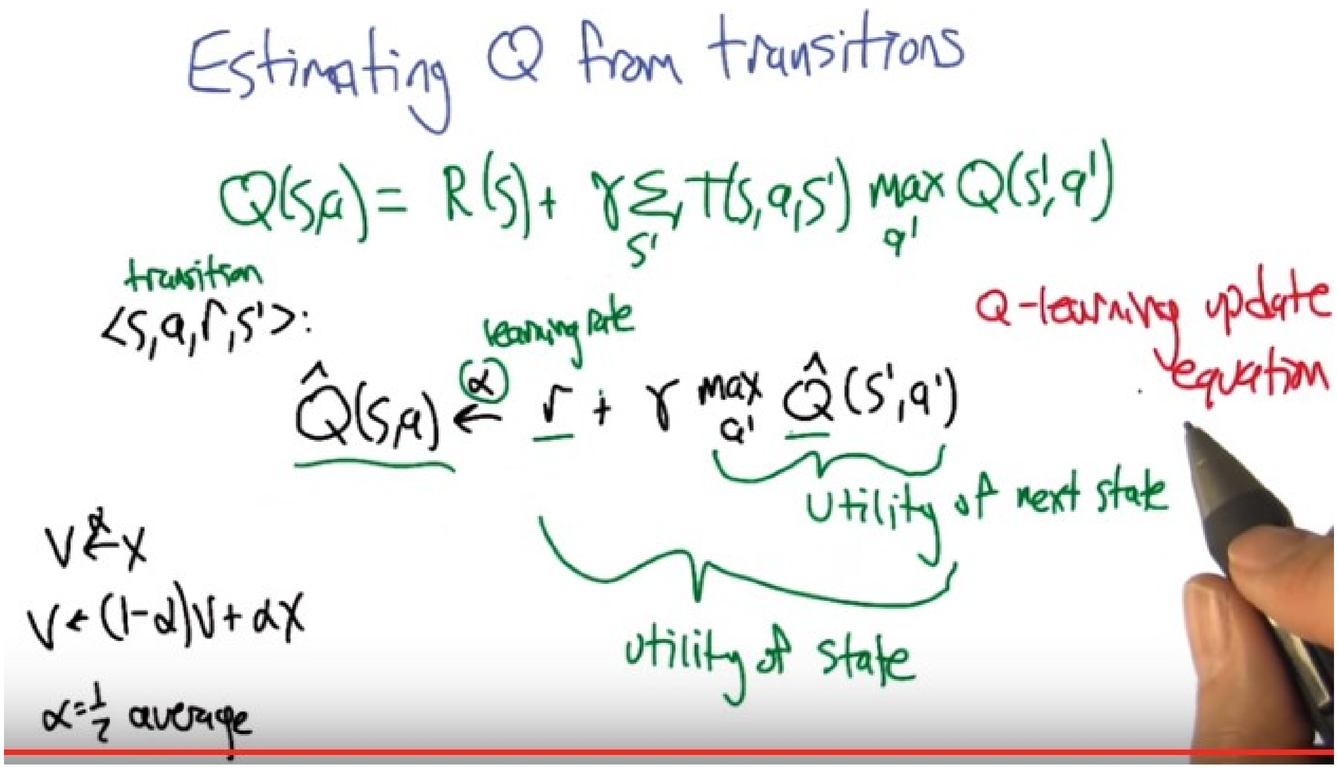

Q-learning : The Quiz
figuring out the best line to wait in
queue-learning
discovering when to come in for your line
cue-learning
practicing the best ~~bank~~ break shot
cue-learning
evaluating the Bellman equations from data
Q-learning
transitions
<s,a,r,s'>
1:49 / 1:49

本 quiz 就是一個搞笑的 quiz, 沒什么深層含義.

10. >> Alright, I apologize in advance but we're going to do a quiz. Q-learning, The Quiz, you know, because quiz starts with Q. So which of these actually describes Q-learning? Is it the problem of figuring out the best line to wait in? Is it discovering when to come in for your line, when you're in a play? Is it practicing the best bank shot? Or is it evaluating the Bellman equations from data? Alright, do you want [LAUGH] are you willing to do this quiz Charles? >> I am, I am willing to do your quiz your quiz learning quiz. >> All right. All right. Let's do it then. >> Okay. Go!

11. >> All right, take me through. >> Okay. So, the first thing I want to say is, I think it's awesome that you made me do this quiz. The second thing I want to say is, in some sense, every single one of them is

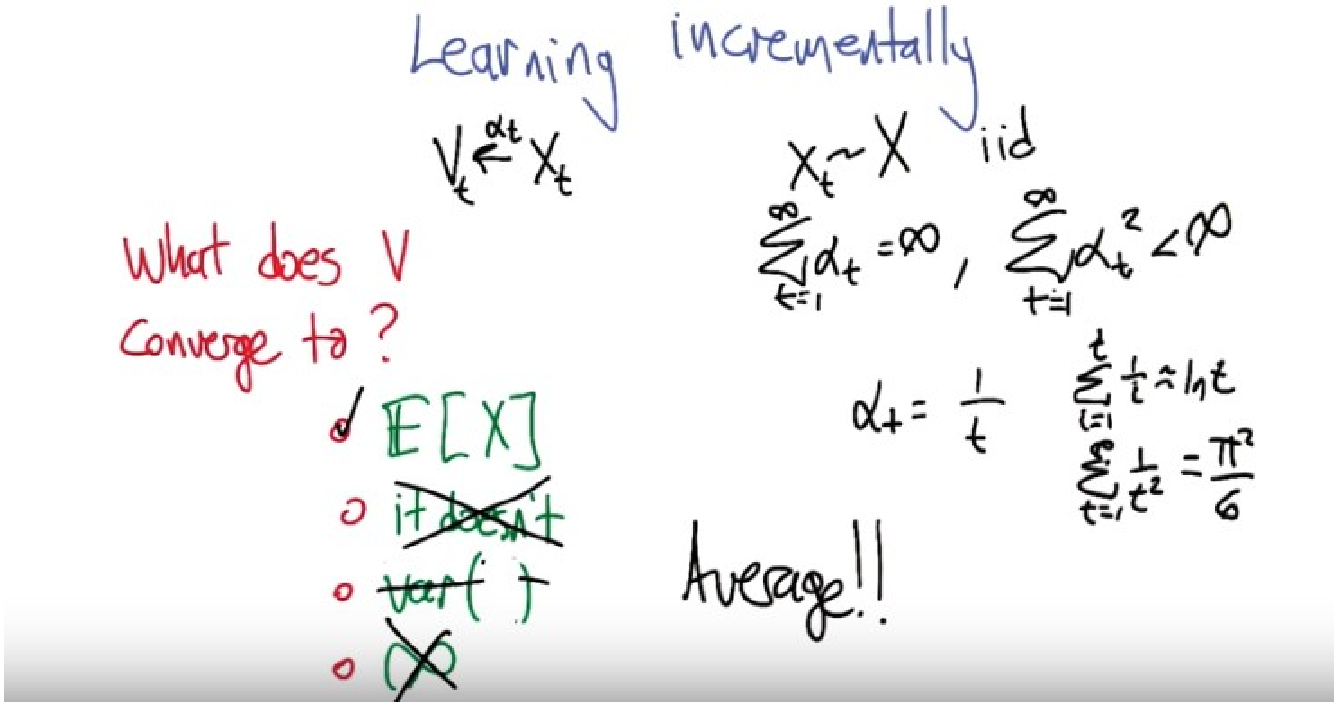
correct. >> Yay! >> But, you put radio buttons down, not check box buttons, thingys, so I have to pick one. So let's just go through. Figure out the best line to wait in. Well, that is a queue, queue which would make a lot of sense if I were, say, English. >> Yes, that's right. In fact, you are, in this picture. >> Oh yeah, great. [LAUGH] That's the English version of Charles, the one with the top hat. Okay. >> Yeah, and a cane, too. >> Yeah, I like it. And a cane, yes, well that is typically what I do whenever I go to London. Discovering when to come in for your line, that is also a queue-learning though a different kind of queue. que-learning. The third practicing your best bank shot, although I probably would've said break shot. >> Ooh, nice. >> That is also cue-learning. >> And it's the same spelling. >> That's the same spelling. >> That's unfortunate. >> Well, you pronounce one cue, and you pronounce the other one cue. >> Yes, well clearly. But, and I spelled them that way too. >> Yeah, and I know. I know. I can see where the emphasis is. And the fourth one is evaluating the Bellman equations from data. That is you take in the states, the actions, rewards and the next states, and you try to learn an actual Q function, which is just the letter Q. So that is Q-learning, and is the only one that is spelled correctly, and so that is the choice that I would make. >> Alright, so, just to be clear here. So this all [LAUGH] the purpose here other than to. >> Demonstrate your ability to show puns? >> [LAUGH] Is that is just to give the definition. So that, so what Q-learning is going to be doing is it's going to use transitions, that is to say data, to directly produce the solution to those Q equations.


Estimating Q from transitions
$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$
transition
$\langle s,a,r,s' \rangle$:
learning rate
$\underline{\hat{Q}(s,a)} \xleftarrow{\alpha} \underline{r} + \gamma \max_{a'} \underline{\hat{Q}}(s',a')$
Q-learning update equation
Utility of next state
utility of state
$V \xleftarrow{\alpha} X$
$V \leftarrow (1-\alpha)V + \alpha X$
$\alpha = \frac{1}{2}$ average

記憶: QrrQ

12. Alright, so what we're going to do to figure out how Q learning works, is we're going to think about what it means to estimate this Q function from transitions. So, let's just remember this is the form of the Q equation, that we've been talking about. And, we can't do this. We can't solve this, because we don't have access to R and T. All we have access to are transitions(後面很快會說甚麼是 transition). >> So this a really, I guess, I'm going to guess you said this before, but when you, when you write out this equation it really jumps out at me. This is the difference between what I was talking about, solving MDPs and reinforcement learning. In solving MDPs we had R and we had T and now we do not have

them, so we have to come up with some other way to solve these kinds of equations. >> That's right. >> Okay. So if we did have R and T, then we could solve this? >> Yeah, this is, I mean, the same things that you talked about, value iteration, policy duration. It can be formulated in terms of Q. So it's, yeah, there's, this is easy to do, well, [LAUGH] it's polynomial to do if you have access to T and R. But but again, in the learning scenario we don't have the model. What we have are transitions. >> Okay, okay. >> So, here's how we're going to use transitions. This is what a transition is (<s, a, r, s'>). We, we observe that we were in some state S of the MDP. And then action A was chosen somehow. And, then a transition happens. We land in a state. We get the reward for landing in that state. And we find out what state we're in. So that's, that's the transition. And what are we going to do with it? Well, what we're going to do, is imagine that we've got an estimate of the Q function, Q hat, and we're going to update it as follows. Here's how we're going to use all these quantities from the transition. We're going to take the, the state and action that we just experienced. And we're going to change it; we going to update it; we're going to move a little bit. Alpha, this is alpha, this is called a learning rate. We're going to move a little bit in the direction of the immediate reward plus the discounted estimated value of the next state. So, we're going to take our estimate Q hat, we going to take the state that we end up in as prime. We're going to look at all the different actions we could take from there and take the maximum. So this together is kind of an estimate of the utility, right? >> Mm-hm. >> And this is the utility of the state that we're going to. This altogether is the utility of the state that, that we're in. To the state S. So this is kind of the utility of the state that we're landing in as prime. And this all together is, is related to the utility of the state, right? You can see that it's related. In that we've got the immediate reward, which kind of matches to this. We've got the discounting. We don't have the sum over transitions but we do have the max A and the lookup in the next, in the Q function. Alright so this, this is the Q learning equation. Alright, let me just say a little bit more about this, this alpha arrow notation, which I really like but is not all that standard. So if you, when I write you know something like V gets with an alpha X. What we mean is we're moving alpha of the way from the current value V towards X, which can be written this way. That V gets 1 minus alpha of V plus alpha of x. And so in particular, if you think about this as so if alpha is 0. That's sort of a learning rate of 0, which shouldn't learn at all, and in fact, if you set alpha to 0 here, it's going to zero out X, and it's going to only assign V to V, so nothing's going to change. So learning rate of 0 corresponds to no learning. And if we set alpha to 1, that's like full learning, so we forget everything that we knew and we just jumped to the new value. And that's what happens here, that 1 minus alpha is 0, so the V goes away, and we just get X assigned to V. So does that make sense? >> That does make sense. And and if alpha were in between 0 and 1 like one half, you're basically making V the average of the old value of V, and the new value X that you see. >> Good.

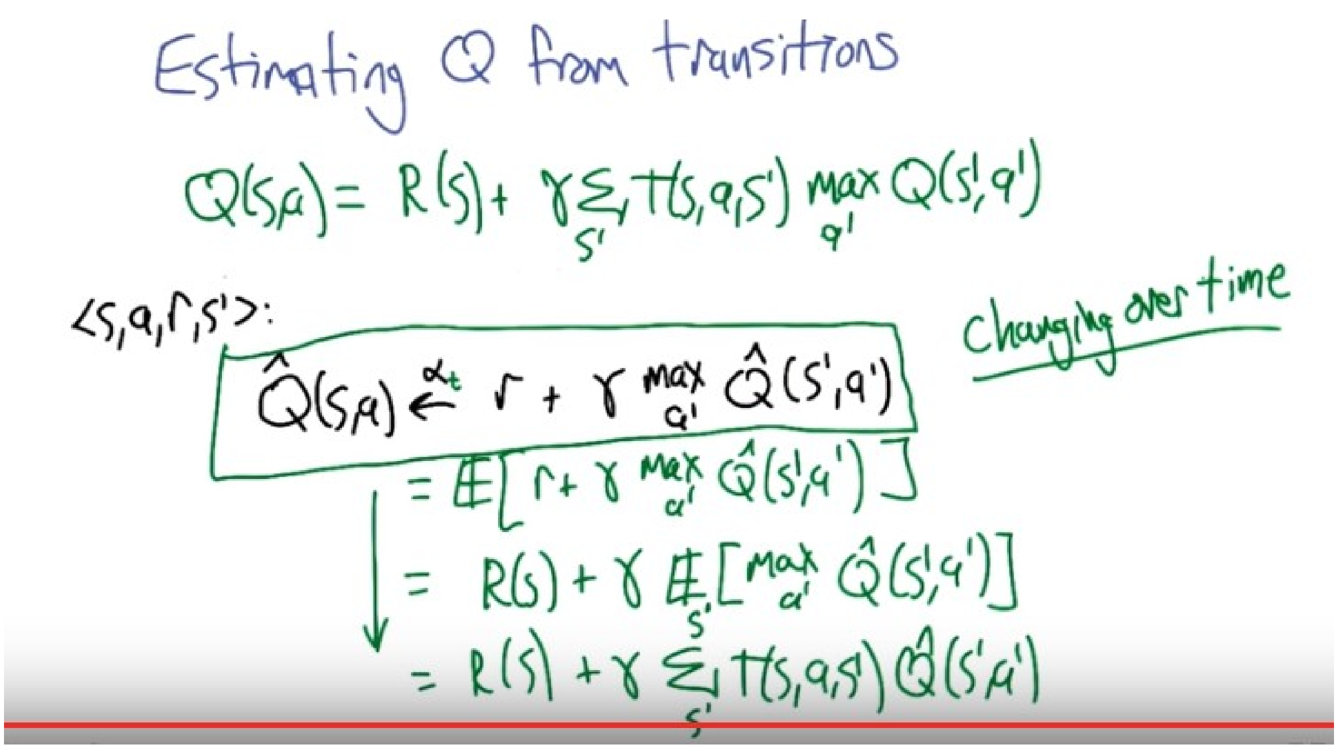


Learning incrementally
$V_t \xleftarrow{\alpha_t} X_t$
$X_t \sim X$ iid
$\sum_{t=1}^{\infty} \alpha_t = \infty$, $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$
$\alpha_t = \frac{1}{t}$
$\sum_{i=1}^{t} \frac{1}{i} \approx \ln t$
$\sum_{t=1}^{\infty} \frac{1}{t^2} = \frac{\pi^2}{6}$
What does V converge to?
✓ E[X]
○ it ~~doesn't~~
○ ~~var( )~~
○ ~~(∞~~
Average!!

13. >> Alright, let's do a little quiz. >> Okay. >> To help us to kind of, make sense of the equations in the Q learning equation by looking at a simpler case. So let's imagine that we got some variable V and some sequencing value X and a sequence of learning rates, alpha sub t. So we're going to draw a series of these X values and use them to update the V values and the learning rate is changing over time and the learning rate is going to have, first of all X is going to be drawn X sub T. Is going to be drawn from some, the distribution of some random variable, big X. >> Mm-hm. >> And the learning rates are going to satisfy two properties. One is that the sum of the learning rates, summed up to infinity. But, the sum of the squares of the learning rate, as we go to infinity, sums up to something less than infinity. So can, can you think of a learning rate sequence that has that property? >> So, the one that I remember is alpha sub t equals one over t. >> Good. So in fact these's a whole range of possible powers of t that work here, but 1/t is a good one. Why is this, why does it satisfy the properties? Well, if you sum up the values up to sum value t, sum up the one over i values, it actually acts like the natural logarithm. And so, as t goes to infinity, the sum goes to infinity. >> But logarithmically. >> Which is still you know, infinity. >> Yeah. >> But if you look at the sums of the squares, that's actually a well known problem called the I think, the Basil problem, the Basil problem? And it turns out to actually be pie squared over six which is kind of crazy. But there it is which is a finite value. >> That's intuitively obvious even to the most casual observer. >> No I really, this, it's, it's, it's whacky. It was an open problem for a long time and then it was finally solved. And it's like, yeah sure pi squared over six. Like where's the pi in here? There's no pi in here. >> Mm, I didn't find pi anywhere. >> [LAUGH] That's neither here nor there. The whole, the point is there's, there's a bunch of sequences of learning rates that satisfy this property, and we're going to imagine that we have a sequence of learning rates that satisfies this property. We're going to be updating V sub-t, with a series of X values drawn from, from distribution big X, and what I'm asking is, what do you think this converges to? Do you think it converges to the expected value of X? Do you think maybe it just doesn't converge at all? Do you think it converges to the variance of X? Of, of the random variable X? Or does it converge to infinity, it just keeps growing without bound? All right. Does that seem like a well formed question? >> sure. >> All right. So let's give you a chance to think about it. >> Okay.

14. Alright, what do you think, Charles? >> Okay, so I think a couple of things. Let me just talk this

out loud. So we're trying to learn incrementally. We're trying to figure out, how if we just kind of keep adding these random variables in a sort of way to what you described before what we would end up at. So why do we want to choose alpha t with a particular kind of property well if you look at it, it's clear that alpha t at each time step is moving closer and closer to zero. So it means you start out learning a lot and you sort of believe things less and less and less or you let things change you less and less and less over time. So what is that likely to end up at? Well the first thing I notice that well if it's going to do that then it has to converge. Some point. It just makes sense. So I, that helps me to eliminate the second answer. But actually, that also lets me eliminate the fourth answer because in some ways they, that says the same thing. So that leaves us with the expected value of x and the expected value of x squared. For the expected value of x squared, whichever one it is that is variance. Okay so, I'm going to go with, the expected value of x. >> So, the expected value of x, right. So, so it turns out that this is actually a wave computing the average by just, you know, repeatedly sampling and updating your values. And, and the way to think about it, is that sometimes the x values that we draw are going to be a bit above the average, and it's going to pull the v value up. Sometimes it's a little bit below the average, and it's going to pull the value down. But in the limit, all those pulls and pushes are going to cancel out, and it's going to settle in on the actual average value or the expected value of this random variable. >> Right, because in principle, you're going to see an infinite number of those things. And effectively all you are doing is adding them all up, and sort of, you know. Well you're effectively averaging them one little bit at a time. >> Yeah, that's a good thing about it, that in fact it's adding them up and it's computing a weighted average, but the weights are these alpha's and the weights are decaying over time so we're going to put more weight on the more recent ones, but some more weight on the further away ones but it sort of doesn't matter because the order, since we're drawing this iid, the order doesn't matter and the thing that has to converge to is the mean. >> Right. I like it. >> So let's relate that back to what we do in Q-learning.



Estimating Q from transitions

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

$\langle s,a,r,s' \rangle$:

$$\boxed{\hat{Q}(s,a) \xleftarrow{\alpha_t} r + \gamma \max_{a'} \hat{Q}(s',a')}$$

Changing over time

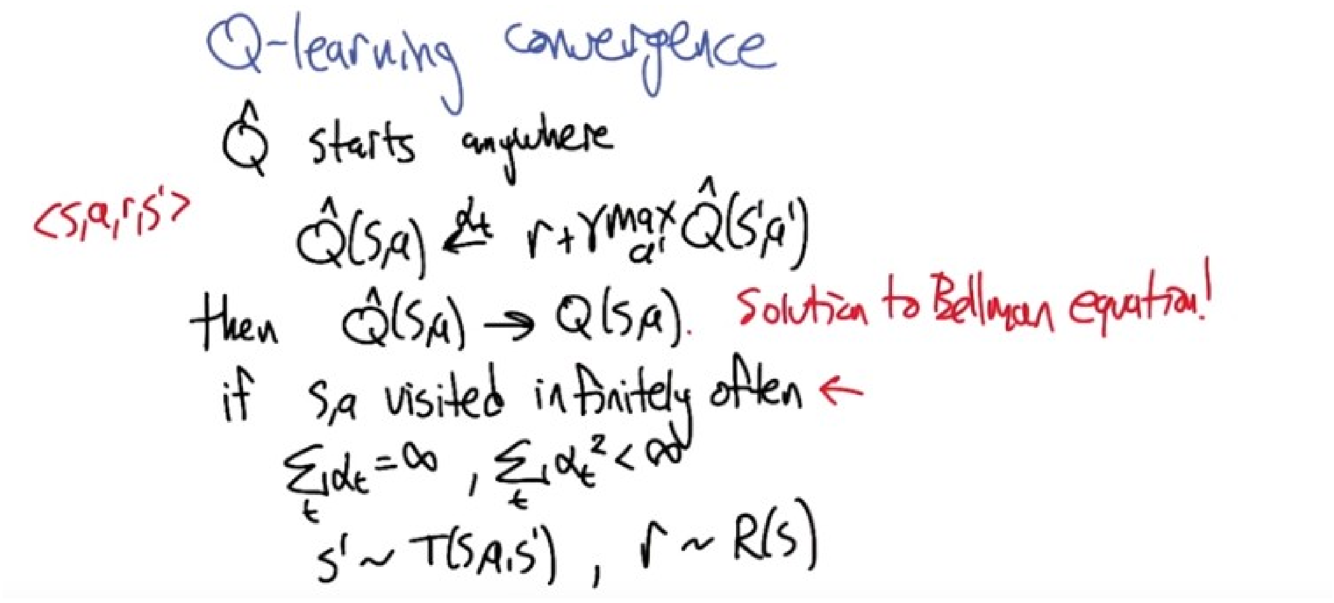
$$= E\left[r + \gamma \max_{a'} \hat{Q}(s',a')\right]$$

$$= R(s) + \gamma E_{s'}\left[\max_{a'} \hat{Q}(s',a')\right]$$

$$= R(s) + \gamma \sum_{s'} T(s,a,s') \hat{Q}(s',a')$$

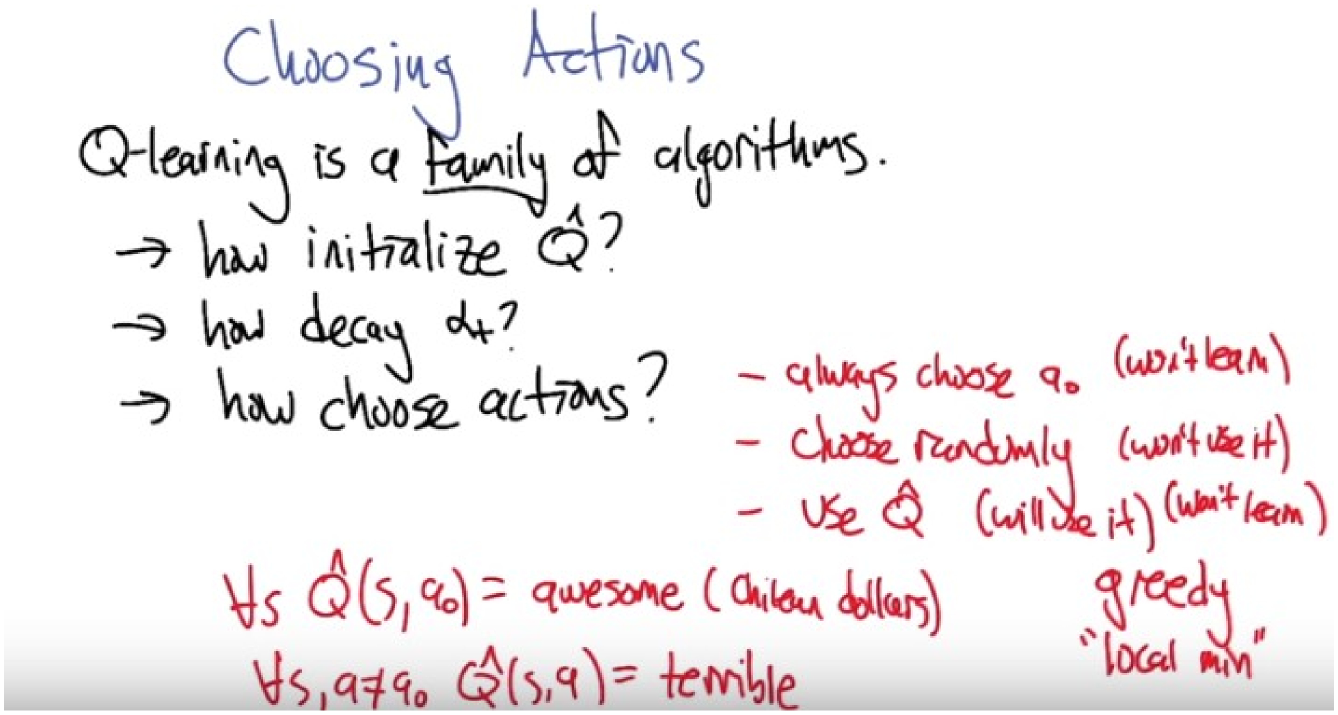
上圖最後的節果其實就等於 Q(S, a)

15. Alright, so this is now the Q in the equation, again. Which again is one of these alphabased things. And, just to be clear, I really do mean, you know, alpha sub t. That we're, that were doing this over time. We're updating our learning rates as we go. It's just, sometimes it's a little irritating to put that, that there. But but yeah. So let's imagine that we're doing that. We're, we're, we're using that same kind of weighted process. Bumping the values around. So, what would, based on what we just talked about, Charles, what would this actually be computing? >> Well, it would be computing the average value that you would get for following, you know, kind of optimal policy after you take this particular action. Yeah, that it's, it's trying to go to this expected value and, and what is that expected value. So the linearity of expectation says that we can actually move the expectation to break up, break up the sum using the expectation. So this, the expected value of the reward is actually R(s). Mm-hm. >> The gamma's going to come out because of linearity of expectation. ANd then what we're left with is the expectation over all next dates of the maximum estimated value of the estimated Q value of the next state. But what is this distribution over S prime? It's the distribution that is determeined by the transtiion function. SO it's this. Which is you know this. So that's good. It's I'm kind of cheating though. Do you see how I'm cheating. >> I think I know how you are cheating but tell me. >> Well so when I told the story about this alpa arrow updating thing. I said that it convergence to the expected value of this quantity here. But this quantity here... Since it's "Q hat", it's actually changing over time. >> Right. >> So this target is actually a moving target. It's changing over time. So we can't quite do this analysis because the first step is a little bit questionable. But it turns out that there really is a theorem that this simple update rule, this Q-learning update rule, this tiny little one line of code (框中的), actually solves Markov decision processes. >> Yeah.


Q-learning convergence
$\hat{Q}$ starts anywhere
<s,a,r,s'>
$\hat{Q}(s,a) \xleftarrow{\alpha_t} r + \gamma \max_{a'} \hat{Q}(s',a')$
then $\hat{Q}(s,a) \to Q(s,a)$.
Solution to Bellman equation!
if s,a visited infinitely often ←
$\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$
$s' \sim T(s,a,s')$, $r \sim R(s)$

注意上圖不是說 Q-Learning 就可以一次 iteration 就得出 Q 的最優解, 因為箭頭上的那個 α_t 代表第 t 次 iteration 的, 故還是要 iteration 很多次才能得出 Q 的最優解. 這在作業中得到了驗證.

16. >> So this is a remarkable fact about this Q-learning rule, and that is if we start Q hat off pretty much anywhere, and then we update it according to the rule that we talked about. Q for, for when we see a transition s,a, r, s prime, then we update (s,a), the Q value for (s,a), move it alpha of the way towards r plus gamma, max a of the, well basically the Q value of the state S prime. Then as long as we do that, then this estimate, this Q^hat (S, a) goes to Q(S, a). The actual solution to the Bellman equation. And I write this with an exclamation mark, because it's like, it's one line of code! It's one line of code, like, how could you not just go out and write this right now? >> Hm. >> But the, the, the, let me just, to finish is, this is only true if we actually visit (S, a) infinitely often. So you know, that's an important caveat. That for this to, to actually hold true, for you to really converge to the solution, it has to run for a long time. It has to visit all state action pairs. The learning rates have to be updated the way that we talked about before (即那兩個 Σ 弄出來的 α_t sequence). The next states need to be drawn from the actual transition probabilities (即 S' ~ T(S, a, S')) but that's, that's cool, if we actually are learning in some actual environment and the rewards need to be drawn from the rewards function. So, this isn't so problematic. This (即 if S, a visited infinitely often) is a little bit problematic, but it is still very reassuring, this idea that we have the right form of an update rule, so that the thing that we converge to is the actual optimal solution to the MDP. >> Cool. And we just have to wait til the heat death of the universe, or infinity, and then we're done. >> Yeah.



Choosing Actions

Q-learning is a family of algorithms.

- → how initialize $\hat{Q}$?
- → how decay α_t?
- → how choose actions?
  - always choose a_0 (won't learn)
  - choose randomly (won't use it)
  - use $\hat{Q}$ (will use it) (won't learn)

$\forall s\ \hat{Q}(s,a_0)$ = awesome (Chilean dollars)

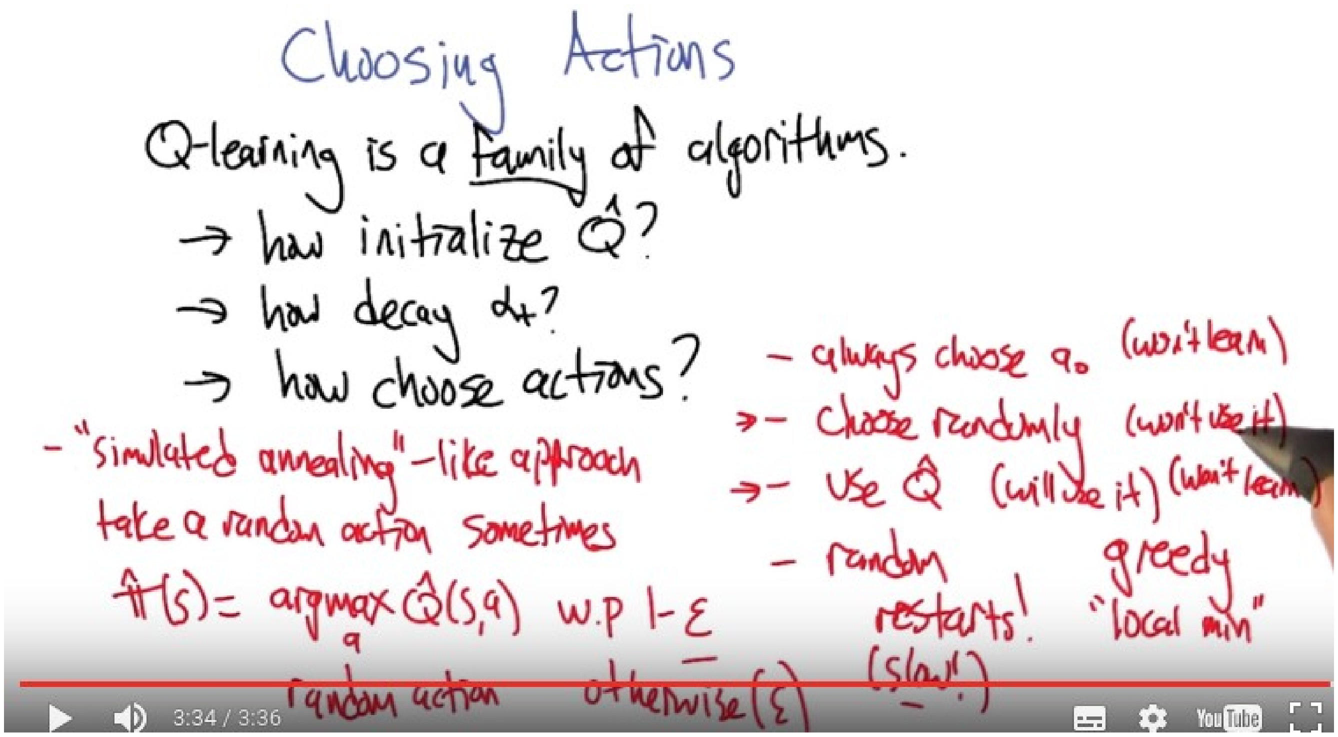
$\forall s, a \neq a_0\ \hat{Q}(s,a)$ = terrible

greedy

"local min"

17. So, Charles, I kind of cheated. >> Oh, tell me more. >> So, Q-learning isn't really an algorithm. Q-leaning is actually a family of algorithms. There's lots of different reinforcement learning algorithms, specific reinforcement learning algorithms that can be reasonably called Q-learning, and they vary typically along these three dimensions. How do we initialize our estimate, Q hat, how do we decay our learning rates, alpha sub-t? And how do we choose actions during learning? >> Hm. >> And different ways of making these choices actually lead to algorithms with fairly different behaviors. In particular when we use this in the context of an MDP, well let's, let me, let me ask you. So like, what do you think might matter about let's start with the last one, choosing actions? >> Well. It see, well there's a bunch of dumb things you could do, right? You could just, just pick an amption, action, action every single time, like the same action every single time independent of what you learned, that's kind of dumb. But, it seems like the obvious smart thing to do is say look, I'm learning, I'm getting better and better, so what I'm going to do is at this next time steps is I actually have to take an action. I'll just pick the action that my Q hat tells me is the best action to take. And I'm done. >> So all right, let me, let me see if I can capture some of what you just said there. So, one way to choose actions really badly is say, pick some action, call it a0. And no matter what state your in. No matter what's happen so far, always choose that action. >> Mm-hm. >> Right, so this possibly can't work. It's going to violate the Q-learning convergence that says we have to visit each state action pair infinitely often, and update them to converge and you know and, and it makes perfect sense. Like if we never try something, like how do you know that you don't like something if you've never even tried it. >> Like spinach. >> Exactly. Another idea would be to choose randomly. And this seems kind of good and that we are going to visit, you know, all the states that are visitable and we will try all the actions that are actionable. And we could actually learn Q this way. But, as you pointed out, this is not a great idea because we may have learned Q, but we haven't really used it. We haven't really chosen actions using what we've learned, so it's like we're wise but we are impotent. >> No, I think it's more like we're wise but we're stupid. I mean. >> We're wise but we still. We know a lot but refuse to actually do anything about it. >> Right.

In particular in some sense the only difference between the two is the the theorem, right? If I, if I'm just choosing randomly, what's the problem with them always choosing a0. Well, you're don't going to converge but the real problem is that you don't learn anything. Or you don't take advantage of anything you learned. Choosing randomly is basically the same thing, you basically, you never take advantage of anything that you learn. What's the point in learning a Q function if you are always going to behave randomly, you've learned enough or you've learned but you [CROSSTALK]. >> [CROSSTALK] right you've actually learned the ultra policy but you're not following it so you're not actually using what you know, so you can't you know it doesn't, it doesn't work all right. So then you had another idea. Which was to use our estimate to choose actions. >> Yeah. >> And that seems like a good idea in that we will use it. Is it possible that it won't learn? >> Well, it will learn something. >> Well, yeah. It might not learn anything all that good though. So for example, what if we do something like this. So we initialize, now we're back up to this, this first point here. We initialize the estimate Q hat so that for every state, a0 looks awesome and all the other actions look terrible. >> Wait [INAUDIBLE] is that metric awesome? Or English awesome? >> Oh you're right I'm sorry, I didn't put units on that. That's in Chilean dollars. >> [LAUGH] >> Oh okay, so it's pretty awesome then. Okay. So, if you do that, then well let's see what happens. It's almost like always taking a0. The only thing that would save you from taking a0 forever, is if, as you take a0, you learn that, you update your Qs and you keep getting really, really bad results. Really, really bad results, in fact, worse than terrible. >> yes. Well let's imagine the terrible is worse than terrible. >> Oh. >> So, but you're right, yeah you're right. So, so there's, there's at least the case that if this terrible value is actually lower than the value of always choosing a0, then we'll continue to use a0 forever. This is, this is called the greedy action selection strategy. >> Mm-mh. >> And that, this is the problem that runs into, it's a kind of local min. >> Oh, I see. Oh and oh, okay, I see that. So, you, you didn't even have to come up with this ridiculous example. If you, if you well, not ridiculous, but you, extremely. >> I was going to say, I'm, I'm sorry, ridiculous how? >> Well, I'm sorry. It, it's a ridiculous situation to be in. It's sort of the extremely unluicky example. >> I think what you're saying is that you don't like people from Chile. >> [LAUGH] Oh no I love people from, I love Chile. Especially with you know, just some good beans and some nice meat. But the thing is that you, if you randomly set your Q hat in such a way that a bad action or let's say a suboptimal action ends up looking much better to begin with then the optimal action. You can get in a situation where you keep choosing the wrong action anyway and so you are only going to learn things that reinforce that action which might be good just not optimal. So you won't actually end up converging onto the true Q hat and that's why you get into a local min. >> That's right and so in that bad situation and you, I admit it's a little contrived because I've never, I've never even been to Chile is that it wont learn. It'd actually be exactly the same as this first case, always choose a sub 0. >> Mm-hm. >> So, that seems problematic too and it interacts in an interesting way with the initialization. Right. So maybe we can do this idea of using Q hat, but we have to be much more careful about how we initialize. >> Hm. So, you know, what I want to do is something like random restarts.



Choosing Actions

Q-learning is a family of algorithms.

→ how initialize $\hat{Q}$?

→ how decay α_t?

→ how choose actions?

- always choose a_0 (won't learn)
- choose randomly (won't use it)
- use $\hat{Q}$ (will use it) (won't learn)
- random restarts! (slow!)
- greedy "local min"

- "simulated annealing"-like approach

take a random action sometimes

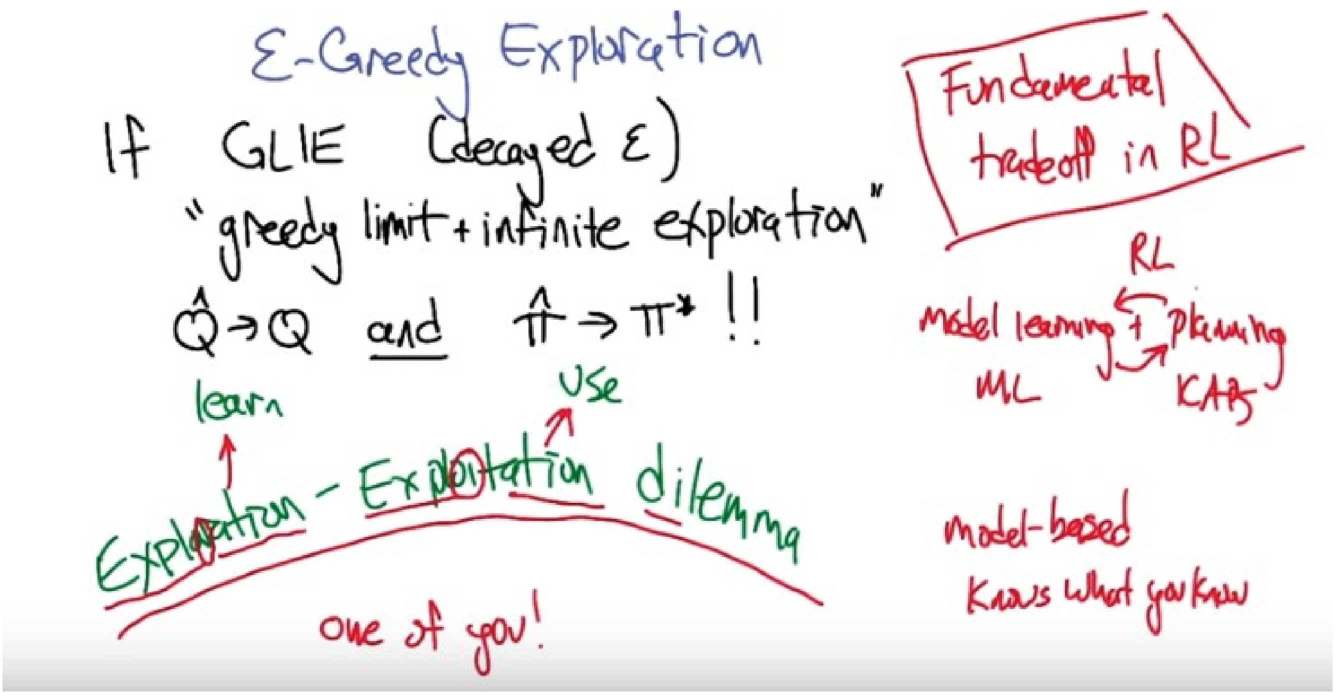
$\hat{\pi}(s) = \arg\max_a \hat{Q}(s,a)$ w.p. $1-\varepsilon$

random action otherwise (ε)

w.p.: with probability

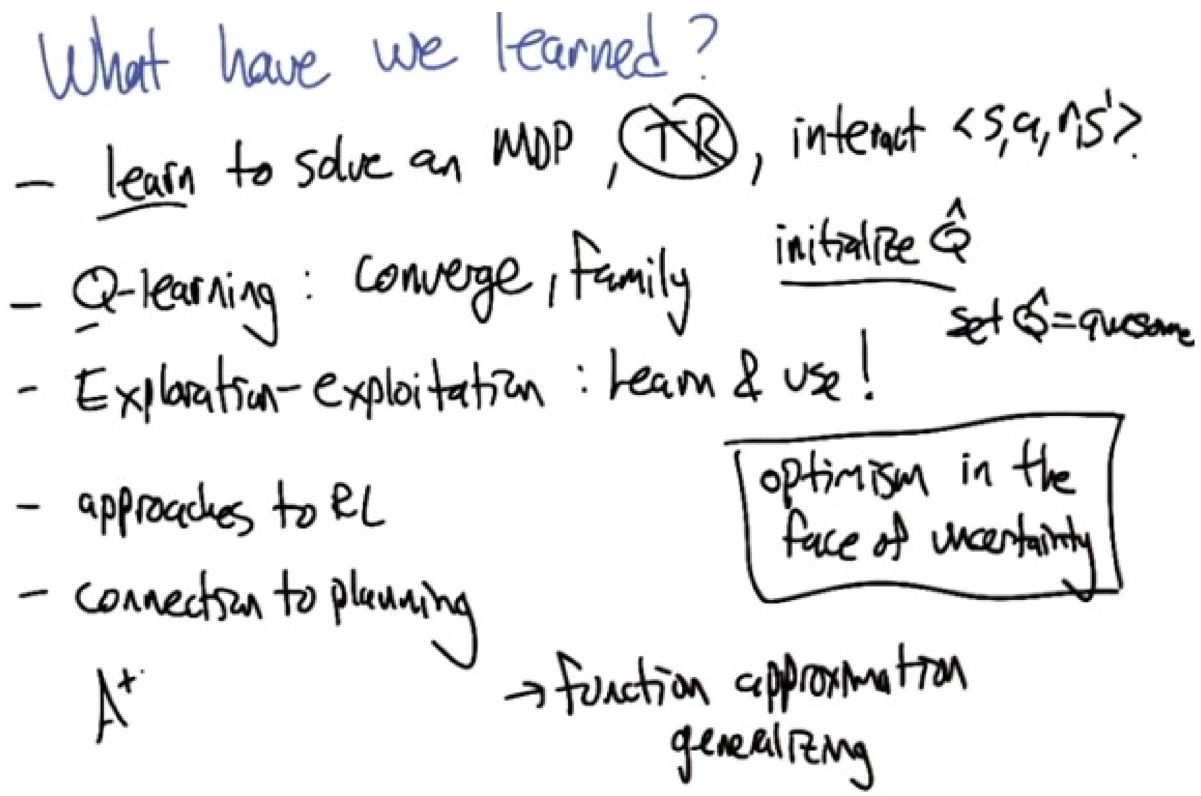
18. So I think that is a clever idea, random restart. That was one of the ways that we got unstuck when we were in local optima when we were doing optimization. Maybe this will also come in handy, in this setting. >> Yeah I like the idea because I came up with it but I can see a couple, you know, but I can see a couple of problems with it. >> But well you can't see a problem with it yet because it's not even clear what you meant. Or maybe that's the first problem. >> Well it was clear what I meant in my head, now if you're going to want my mouth to understand what my head meant, well then your just asking for too much, so I really kind of meant, random restarts where you just kind of start it over, over, over, and over again, the problem with that, the problem with that is, it's already going to take a long time, you know, infinity to get to a good answer. And we thought we might have this issue with randomized optimization, certainly this is going to be a problem here, but it still feels like there's something we could do with the randomness idea that would help us to overcome this problem with using what we know. And only using what we know. >> Alright, alright, so I, I like that direction. So let's think a little bit about what that could mean. So random restarts was one idea that we had when we were talking in optimization about getting unstuck, which is, let yourself get stuck, and then once you realize you're stuck, throw everything out and start over again. >> Right. >> And you're right. That's going to end up being really slow, but we had a different way of Of using randomness to get unstuck. In, for example, algorithms like >> Simulating a kneeling >> That's what I was thinking about. Yeah, so Simulating a Kneeling. The idea is Simulating a Kneeling is that you tend to, you take up hill steps. But occassionally, you're willing to take a random down hill step. >> Right. >> So, it's kind of this mixture of choosing randomly. And using what you know, what seems to be the best. Right, and so, yeah, yeah, I can see that. So then the random restarts thing kind of works if instead of it being a random restart, it's, it's just a random action every once in a while. >> Yeah, excellent alright. So simulated annealing like-approach says we're going to take a random action sometimes. So then our exploration policy, our

approximate policy is going to be. To, we're in state s. Figure out the best action in that state according to our estimate and take that with probability I don't know. let's say 1 - ε. >> Mm-hm. >> In otherwise take a random action and see what happens. >> I have a probability of epsilon. Yeah that's whats left over so. >> Right so that's why you had one minus epsilon because you want ε to be small. So epsilon is going to be. So every once in a while you will randomly act, oh and then that'll, that'll solve the sim annealing problem. Or do what sim annealing helps you with by just taking a random action in what looks like a bad direction, comparatively, sometimes. But it also means that you get to explore the whole space. And so you have a chance of actually learning the true Q. >> Exactly. So, and yet most of the time, you know, we're spending, assuming Epsilon is small, we're spending a lot of our time taking good actions, or actions that we think are good. But we're always holding out some probability of taking other actions to help us improve our Q hat. >> Mm, so if you do that infinitely long, and you've, this will, so this will let you visit SA. An infinite number of times, as long as epsilon is greater than zero. >> That's right and that the mdp is connected, right? So, there could be state action pairs, that just can't ever be reached, in which case you won't reach them. But that's okay, because since you can't reach them, they really don't matter. >> Yeah, they sort of don't exist. >> Exactly. Alright so let's let's focus in on this a little bit more because I think that this is now the first idea that we've had for choosing actions that has the property that it will learn and it will use what it learns. >> Mm, I like that.


ε-Greedy Exploration
If GLIE (decayed ε)
"greedy limit + infinite exploration"
Q̂ → Q and π̂ → π* !!
learn
use
Exploration - Exploitation dilemma
one of you!
Fundamental tradeoff in RL
RL
model learning + planning
ML
KAB
model-based
Knows what you know

19. >> Alright, so let's say something that we actually know about this approach to action selection which is called ε-Greedy Exploration. And what we know is that if the action selection is GLIE, which means Greedy in the limit with infinite exploration, and that basically means that we are decaying our ε for ε Greedy. That we start off more random and over time we get less and less random and more and more Greedy. Then we have two things that are true. One is that Q hat goes to Q, which is, which comes from kind of a standard Q learning convergence result. But, we also have something cooler in this case which is that the policy that we're following, pi hat, is getting more and more like the optimal policy over time. >> Hm. >> So, not only do we learn stuff, but we use it, too. And this is an example of the Exploration-Exploitation Dilemma. And exploration exploitation is really, really important, not just because they're two words that surprisingly start with the same five letters, like unlikely letters. But

also, that it is strike, it is talking specifically about this issue. Exploitation is about using what you know. And exploration is about getting the data that you need so that you can learn. And this is one particular way of doing, it turns out there's lots of other ways of making this trade-off and the reason it's a trade-off is because there's only one of you. There's only one agent acting in the world and it has these two actually somewhat conflicting objectives. One is to take actions that it doesn't know so much about so it can learn about them, and the other one takes actions that it knows are good so that it can get high reward. >> Hm, that makes sense. You know, I didn't, I never realized that exploration and exploitation share the first five letters. >> Hm. >> I always knew that they shared the last five letters. >> Oh, that's interesting too. >> Huh. So if you take an r and turn it into an it, you might move from exploration to exploitation. I feel like an entire political movement could be founded on that. [LAUGH]. But I'm not sure exactly what it would be yet. Maybe I'll work on that, before our next lesson. >> So I have an algorithm. There, there's a standard, lemma. >> Mm hm. >> In reinforcement learning theory called the exploration exploitation dilemma. Sorry, [LAUGH], no, lemma. The other kind of lemma. The exploration exploitation lemma, which has to do with taking actions that are either exploring or exploiting. But I have one where you actually do teaching. >> Mm-hm. >> You can actually, each time you take an action it's either going to teach the agent something, it's going to explore or exploit. So I call that exploration exploitation, or explore, exploit, explain. >> Oh, nice. But you could have called it the exploration exploitation dilemma because you used dilemma. And di means two sometimes. And you do the two things. >> Well, in fact, dilemma does literally mean two. Its a choice between two things. >> Right, so its a dilemma. >> Nice >> Its a lemma about two things. >> Alright, so there is, it turns out there's a lot of other approaches to exploration and exploitation. And some of them in the, in the model based setting. You can do a lot more with it, a lot more powerful things with it because you actually can keep track of what you've learned effectively in the environment and what you haven't, so the algorithm can actually know what it knows and can use that information to explore things that it doesn't know and exploit things that it does know. Q learning doesn't really have that distinction. It's a much harder thing to do. So, so that's what I wanted to tell you in terms of, you know, thinking about exploration-exploitation, does that make sense to you? >> It does make sense to me. I think what, the main point I got out of this, or a main point I got out of this, other than our incredible ability to get caught up in letters and coincidences of spelling, is that the exploration-exploitation dilemma really is a dilemma. It's like the fundamental tradeoff in reinforcement learning. You have to exploit because you have to use what you've got, but you have to learn or otherwise you might not be able to exploit profitably. So you have to always trade off between these things, and if you don't, you're bound to either learn nothing or to get caught in local minima. >> I couldn't agree with you more. In some sense, if you think of reinforcement learning as being the question of model learning plus planning, there's nothing new here because model learning is well studied in the machine learning community and planning is well studied in the planning and scheduling community. >> Planning. >> And so, like, what are we adding to this? And what we're adding is the fact that these two processes interact with each other and depend on each other and that's exactly the exploration, exploitation dilemma and that's information has to go back and forth between these two processes that other people understand and we're the glue. >> Or the glee. >> [LAUGH] The glee glue. >> I like it. That's beautiful.



What have we learned?

- learn to solve an MDP, $\cancel{TR}$, interact $\langle s,a,r,s' \rangle$
- Q-learning: converge, family initialize $\hat{Q}$ set $\hat{Q}$=awesome
- Exploration-exploitation: learn & use!
- approaches to RL optimism in the face of uncertainty
- connection to planning

A* → function approximation generalizing

20. So that's at least in a nut shell the reinforcement learning story. There's there's a lot of other topics and I think we are planning to get to some of them in a topics lesson a little bit later, but there's you know, courses and books worth of stuff to study in this area. Just like supervised learning as a whole. But, we're going to just kind of end it there with the idea that we now have a handle on how we can use learning to do decisions with delayed rewards. So, can you help us summarize what we what we learned in this lesson? >> Okay, sure. I think what did I learn here today, I think I learned a lot of things, some of which had to do with reinforcement learning mainly that you can actually learn how to solve an MDP. I think that's actually a pretty big deal. >> Right. Meaning that we don't know T and R. >> Mm-hm. >> But we just have access to the ability to interact with the environment and recieve transitions. >> And, that's actually that's actually pretty impressive. And, a very powerful thing. Because, we're often not, if we assume the world is an MDP, but we don't know T and R. If we don't have some way of learning in that we don't really have much we can do. And, you've showed me that their is something we can do. >> Cool. >> I think that's the biggest thing. We learned some specific things. In particular, we learned about Q-learning. >> Several kinds of Q-learning, but one is actually a real word. >> Yes [LAUGH] indeed they're all real, Michael. We learned about Q-learning. And I think the other most important thing that we learned about is the exploration versus exploitation trail. >> And with Q-learning, we learned a little bit about when it converges. >> Mm-hm. >> And that it's actually a family of algorithms. And different members of that family have different behaviors associated with them. Oh, there's one other thing I wanted to say on that topic, actually. >> Okay. Which is, that one way to achieve this exploration-exploitation balance, was to randomly choose actions. So to change the we're doing action selection. But there's another one too, which is that we can

actually by manipulating the initialization of the Q function. We can actually get another kind of exploration, can you see how that might work? >> Oh, I know what you do. If you could, if you set, say the Q values to all be the highest possible value they could be. >> Great, so if we initialized the Q hat to awesome values, then what the Q lettering algorithm would do, even with greedy exploration, what it will do is it will try things that it hasn't tried very much, and it still thinks are awesome. And little by little, it gets a more realistic sense of how the environment works. And. >> So it's very optimistic? >> That's right, exactly and it's referred to often as optimism in the face of uncertainty and it's a similar kind of idea that's used in algorithms like, A*, if you're familiar with search algorithms in AI. >> Oh yes, I remember those. >> But this is, this is a really powerful idea and it's used in, in reinforcement learning and banded algorithms and planning and And search. >> Okay, and that makes sense because if everything is awesome. Then your true key value can only go down if awesome is bigger than the biggest Q value you could ever have. And so that means you're going to look at every single action. And as you learn more about them, then you will just get more depressed about them. And that's good. >> [LAUGH] Yes the world slowly beats you down. [LAUGH] So is that it? Is that all we really talked about? I guess that's about right. We talked about what a que function was. >> Right. >> And how that kind of binds everything together and we talked about different approaches including policy search and model base reinforcement learning. >> Yeah that was very nice. We tied it all back into planning. >> So one, one thing we didn't talk about is connecting to function approximation, and the issues in machine learning that are really important things things like over fitting. They come up in the reinforcement learning setting, but not in this simplified setting that were looking at here where we learn a separate value. For each state action pair, we're going to have to start generalizing to see the importance of that. And that's, we're going to do in a later lesson. >> Okay. I like it. And we also learned a bunch of things about letters. Like exploration versus exploitation. >> In fact, we know enough that we can now get an A in letters. >> [LAUGH] I like it. Okay. Well, I think we learned a lot, Michael. >> [LAUGH] Okay. Well, good. Well, thanks. It's very nice to get a chance to talk to you about this stuff. >> Cool. And so I guess. What are we going to talk about next. >> Well Whatever it says on the syllabus. >> I think it's game theory. >> That's pretty cool. >> Oh. I see why we're going to do that. Because all we've been talking about the world as if there were just one agent and nobody else. And now we're going to see what happens, when there are other people. >> Right. Other people show up at the party, next time. >> On, Machine Learning.