

03-05 Reinforcement Learning

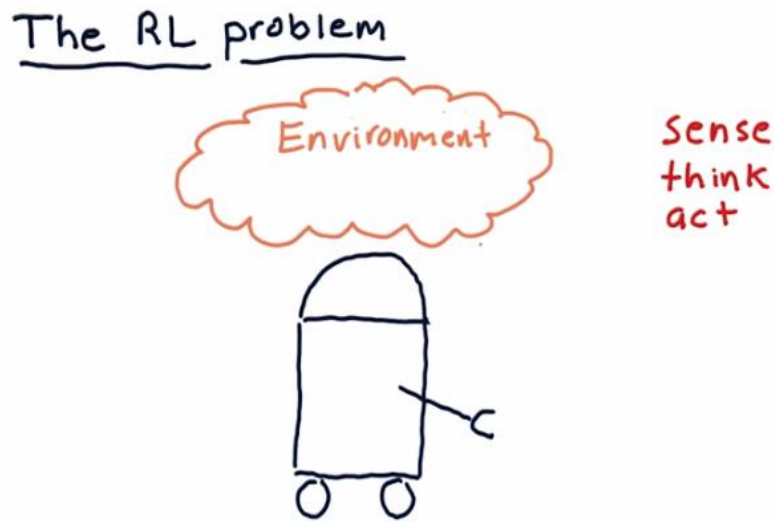
Compiled by Shipra De, Fall 2016

Overview

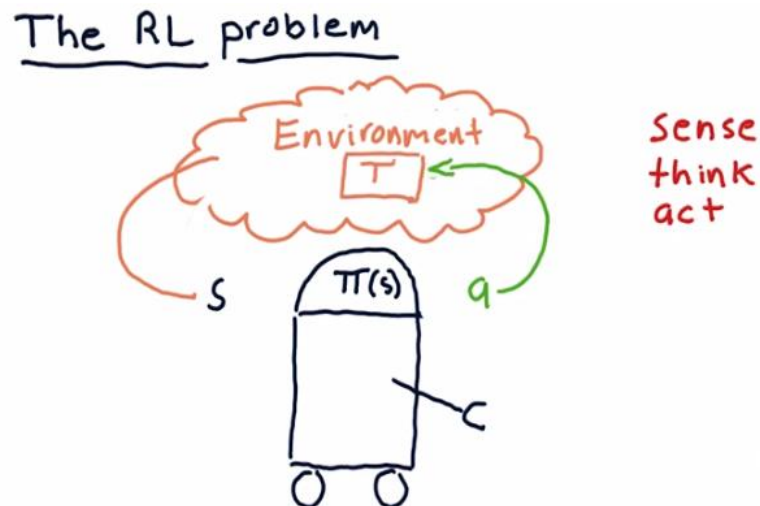


- Up until this point, we've focused on learners that provide forecast price changes. We then buy or sell the stocks with the most significant predicted price change.
- This approach ignores some important issues, such as the certainty of the price change. It also doesn't help us know when to exit the position either.
- In this lesson, we'll look at reinforcement learning. Reinforcement learners create policies that provide specific direction on which action to take.

The RL Problem

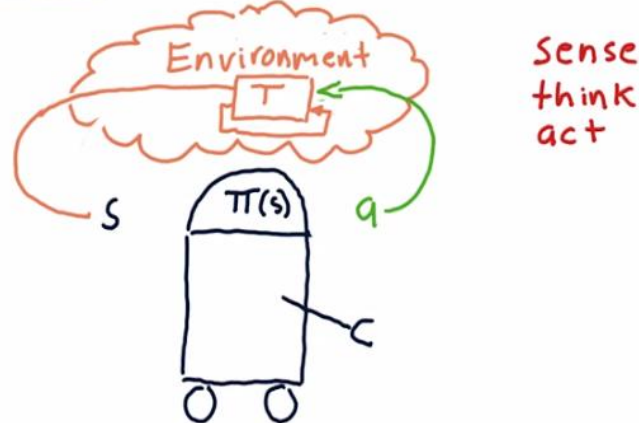


- It's important to point out that when we say reinforcement learning, we're really describing a problem, not a solution. In the same way that linear regression is one solution to the supervised regression problem, there are many algorithms that solve the RL problem.
- Because I started out as a roboticist, I'm going to first explain this in terms of a problem for a robot. So here's our robot here and our robot is going to interact with the environment. So we represent the environment as this sort of cloud up here.
- So the robots going to take actions that'll change the environment. It will sense the environment, reason over what it sees and take another action. In robotics, we call this the sense, think, act cycle and you don't have to implement it only using reinforcement learning.
- There's many ways that you could implement sense, think, act, but we're going to focus on how to do that with reinforcement learning.



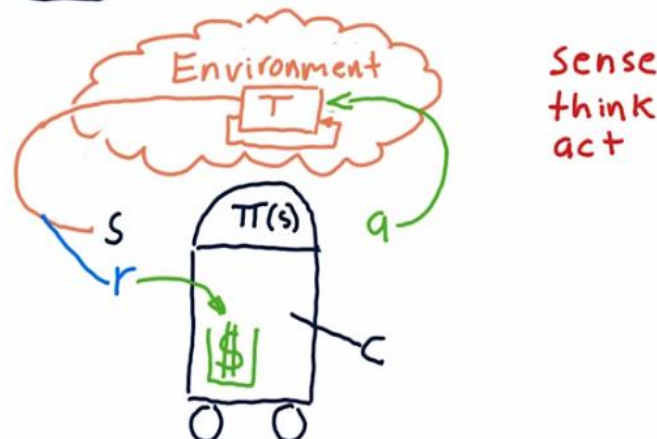
- Okay, so our robot observes the environment and some form of description of the environment comes in. Let's call that the state s , so s is our letter that represents what we see in the environment.
- Now the robot has to process that state somehow to determine what to do. And we call this π or policy, so the robot takes in the state s and then outputs an action. We'll call that action a and it affects the environment in some way and changes it.

The RL problem



- Now this is a sort of circular process, the action a is taken into the environment and the environment then transitions to a new state. So T is this transition function that takes in what its previous state was and the action and moves to a new state.
- And that new state comes out, boom, back into the robot. Robot looks at his policy, action comes out.

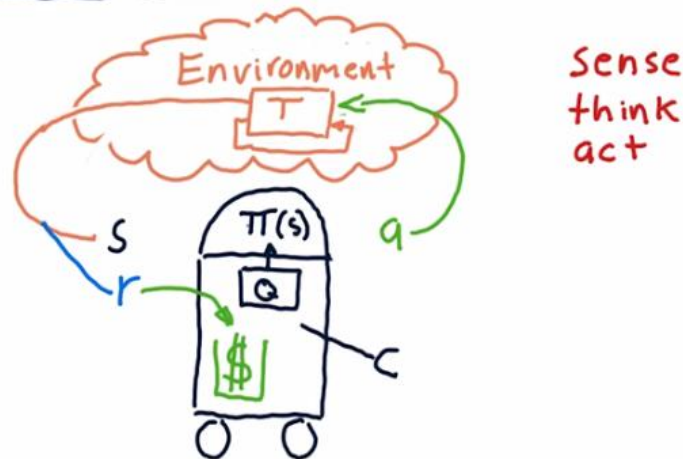
The RL problem



- Now there's a question, how do we arrive at this policy? How do we find π ? Well, that's what we're going to spend a couple lessons on, but this whole puzzle is missing a piece and that's the thing that helps us find π .

- And part of that piece is well, there's this other part called r which is the reward. So every time the robot is in a particular state and it takes an action there's a particular reward associated with taking that action in that state and that reward comes into the robot.
- And you can think of the robot has having a little pocket where it keeps cash and that's what that reward is. And the robot's objective is, over time, to take actions that maximize this reward.

The RL problem



- And somewhere within the robot, there's an algorithm that takes all this information over time to figure out what that policy ought to be.
- So let me recap a little bit.
 - o S is the state of our environment and that's what the robot senses in order to decide what to do.
 - o It uses its policy π to figure out what that action should be. And by the way, π can be a simple look up table.
 - o Over time, each time the robot takes an action, it gets a reward and it's trying to find the π that will maximize its reward over time.
- Now in terms of trading, our environment really is the market and our actions are actions we can take in the market, like buying and selling or holding. S are factors about our stocks that we might observe and know about and r is the return we get for making the proper trades.

Trading as an RL Problem

Q: Trading as an RL problem

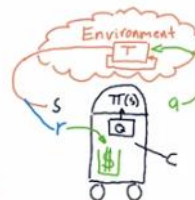
	S	a	r
• Buy	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
• SELL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
• Holding Long	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
• Bollinger Value	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
• return from trade	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
• daily return	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



- Now as you know, we want to use reinforcement learning algorithms to trade with. So let's think now about how we can map the trading problem to reinforcement learning. Okay, so consider each of these factors. Buy, sell, holding long, Bollinger value, return from trade, daily return.
- And then consider, is that item a description of our state that we ought to consider before we make a trade? Is it an action that we give to the market to cause a trade to occur? Or is it a potential reward that we would use to inform our algorithm for learning how to trade? And it's potentially the case that some of these may serve more than one role.

Q: Trading as an RL problem

	S	a	r
• Buy	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
• SELL	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
• Holding Long	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
• Bollinger Value	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
• return from trade	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
• daily return	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>



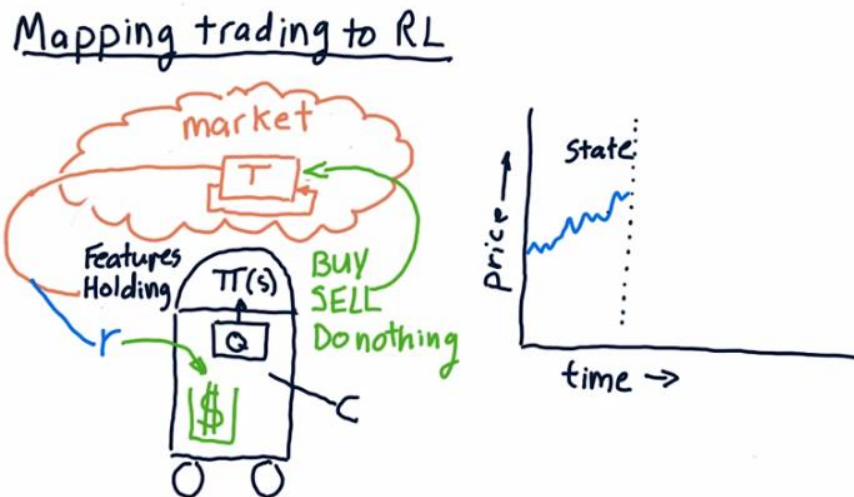
- Okay, let's step through these one at a time. Buy and sell are actions. So, those are directives we give to the market or the environment to change it, and potentially change our state.
- Holding long is a part of the state, it tells us whether we are holding the stock or not. We might also be holding short if we had shorted of the stock. So holding long is a part of the state.
- Bollinger value, that's a feature, a factor that we can measure about a stock, and that's part of the state as well. That would inform us whether we wanted to act on it in some way with an action.

- Return from trade, when we finally exit a position. That is our reward. We might lose money, so it would be a negative reward if we lost money. We might make money and that'd be a positive reward, so that's R a reward.
- Daily return, that could be either a state, in other words a factor we consider for deciding what to do, but it could also be a reward, we'll get into that more later and you'll see how it could be one or the other.

Mapping Trading to RL

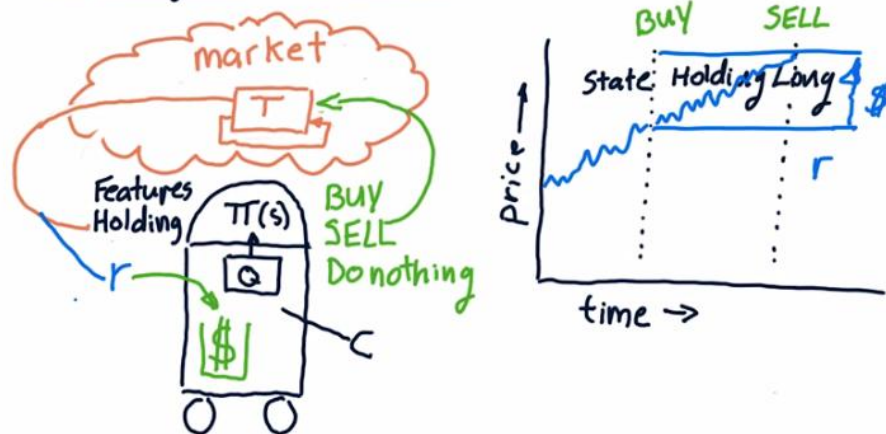


- Let's consider now a little more carefully how we map trading to an RL problem. So first of all the environment here is really the market. Our state that we're going to consider includes things like market features, prices, whether we're holding the stock.
- I'll list a few of those items right here. Our actions are things like buy and sell, and potentially do nothing is also an allowable action.



- So let's think about this in the context of trying to learn how to trade a particular stock. So we've got this historical time series, and let's say this vertical line is today.
- Now we can look back over time to infer the state of the stock. So what are the Bollinger Band values and things like that. Now we process that and decide what's our action.

Mapping trading to RL



- Let's suppose that we decide to buy. So once we buy, we're now holding long. That's part of our state. We go forward. We're now on a new state where the price has gone up. We're holding long. Let's suppose we decide to sell at that point.
- So we've had two actions. Well we've been in two states. In state one we were not holding. We executed the action buy, went forward in time, we're holding long now, and then we execute the action sell. Note that we made money here and that's our reward, r .
- So just to recap for a moment, the policy that we learn tells us what to do at each time we evaluate state, and we're going to learn that.
- We haven't talked yet about how we learned the policy. But we're going to learn the policy by looking at how we accrue money or don't based on the actions we take in the environment.

Markov Decision Problems

Markov decision problems

- Set of states S
- Set of actions A
- Transition function $T[s, a, s']$
- Reward function $R[s, a]$



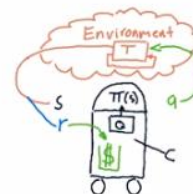
- Let's formalize this a little bit. What we've been working with is something called a Markov decision problem. And here's what makes up a Markov decision problem.
- There are a set of states S . Those are all the values that this S can take as it comes into the robot.
- There's a set of actions A , which is these potential actions we can take to act on the environment.
- There's a transition function. This is the T within the environment. And this is a little bit complicated, but let's just step through it. T is a three-dimensional object, and it records in each of its cells the probability that if we are in state S and we take action A , we will end up in state S prime.
- Something to note about this transition function is, suppose we're in state, a particular state S and we take a particular action A . The sum of all the next states we might end up in has to sum to one.
- In other words, with probability one, we're going to end up in some new state, but the distribution of probabilities across these different states is what makes this informative and revealing.
- Finally, an important component of Markov decision problems is the reward function. And that's what gives us the reward. If we're in a particular state and we take an action A , we get a particular reward.
- So if we have all of these things defined, we have what's called a Markov decision problem.

Markov decision problems

- Set of states S
- Set of actions A
- Transition function $T[s, a, s']$
- Reward function $R[s, a]$

Find

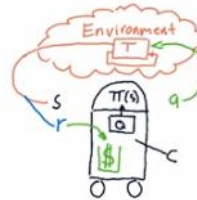
policy $\pi(s)$ that will maximize reward



- Now, the problem for a reinforcement learning algorithm is to find this policy π that will maximize reward over time.

Markov decision problems

- Set of states S
- Set of actions A
- Transition function $T[s, a, s']$
- Reward function $R[s, a]$



Find

policy $\pi^*(s)$ that will maximize reward

policy iteration
value iteration

- And, in fact, if it finds the optimal policy, we give it a little symbol π^* to indicate that it's optimal.
- Now, if we have these, and, in particular, if we have T and R , there are algorithms we can unleash that will find this optimal policy. Two of them are policy iteration and value iteration.
- Now, in this class, we don't start off knowing T and R , and so we're not going to be able to use these algorithms directly to find this policy.

Unknown Transitions and Rewards

Unknown transitions and rewards



- Most of the time we don't have this transition function, and we don't have the reward function either. So the robot, or the trader, whatever environment we're in, has to interact with the world, observe what happens, and work with that data to try to build a policy.
- So let me give you an example here. Let's say we were in state S_1 . So, that's what we observed there. Our robot took action, A_1 . I'm making this little subscript to indicate which step in this series of steps it's at.
- We then find our self in S' . And we get reward R . Now this is an experience tuple. This is very similar to experience tuple in regression learning where we have an X and a Y paired together. That's an experience tuple of you know, when you observe this X you see this Y . Here we're saying when you observe the state, S_1 , you take action, A_1 , you end up in this new state, at least it's an example of you ending up in this new state S_1' , and reward, R_1 .

Unknown transitions and rewards



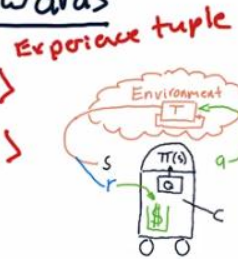
- Now we find ourselves in a new state S_2 , but that's really, this state is where we found our self. We take some new action, A_2 , we end up in some new state, S_2' , and we get a new reward, R_2 .
- When we do this over and over and over and over and over again, gathering experience tuples all along the way. Now, if we have this trail of experience tuples, there's two things we can do with them in order to find that policy π .

Unknown transitions and rewards

- Model-based
Build model of
 $T[s, a, s']$
 $R[s, a]$
Value/policy iteration

- Model-free
Q-Learning.

$\langle s_1, a_1, s'_1, r_1 \rangle$
 $\langle s_2, a_2, s'_2, r_2 \rangle$
...



- The first set of approaches is called model based reinforcement learning. What we do is we look at this data over time and we build a model of T just by looking statistically at these transitions. In other words we can look at every time we were in a particular state and took a particular action and see which new states we ended up in and just build a tabular representation of that. It's not hard.
- Similarly, we can build a model of R . We just look statistically when we're in a particular state, and we take an action, what's the reward? We can just average that over all these instances.
- Once we have these models, we can then use value iteration or policy iteration to solve the problem.
- There's another set of approaches called model-free. And that's the type we're going to focus on. In particular we're going to learn about Q-learning. And model-free methods develop a policy just directly by looking at the data. And of course we'll talk about those soon.

What to Optimize?

What to optimize?



- We didn't go into enough detail about what it is we're trying to optimize here. I just said something vague like we want to maximize the sum of our reward. Well, it's not so simple, in fact, here's a great story to illustrate that.
- There's a great Russian comedian, Yakov Smirnoff, you may remember him or not, but he told this joke once that I really loved. He said, have you heard about the Soviet lottery, it's a million rubles if you win. One ruble a year for a million years.
- So the point is, and if you recall from one of our earlier lessons, that one dollar or one ruble delivered to us a million years in the future is really not as valuable as a dollar or ruble that we get now.
- And so, for instance, if we think about a robot living forever, it might do something just mundane to gather a dollar a year. That's an infinite amount of money, but in practice it doesn't really work that well.
- So to consider that, and to illustrate that, I'm going to show you a little maze problem here, and we'll think about what the robot ought to do that would be optimal in this maze.

What to optimize?



- So here's our robot, and here's the challenge for our robot. We have a reward here of \$1 and a reward over here of \$1 million. So if the robot comes over here and gets this \$1. It's special in that each time he touches it, he gets \$1 and it goes away but then it comes back. So the robot could come here go back and forth and get a dollar each time it moves here.
- This one, once the robot tags it, it's gone. But clearly it's worthwhile to come over here and grab it.
- Now this red area is obstacle, it can't go there. And here I wrote some rewards that the robot, in fact negative one is a penalty. But the penalties the robot would get as it went this way, and zero penalty that way.
- Now, if we say that what we want to optimize is the sum of all future rewards, then it doesn't matter whether we go this way and just get that dollar over and over and over again. Or if we go this way, get the million dollars, come back and get that \$1 over and over and over again. Now there's no difference because they both sum to infinity over time.

What to optimize?



- Now what if we say, okay, I want to optimize my reward over three moves. So I've got a finite horizon. Let's consider the rewards we get with a finite horizon of three if we go this way versus this way.
- So if we go this way, we're going to get rewards of -1, -1, -1, and if we go this way we get zero, \$1, and then we have to move down here, and get another zero. So clearly, starting here, with a finite horizon of three, the best thing to do is go up there.

What to optimize?

infinite horizon
finite horizon

\$1		\$1M	-1	-1
0				-1
jackpot	-1	-1	-1	-1

-1, -1, -1, -1, -1, -1, -1, \$1M
0, \$1, 0, \$1, 0, \$1, 0, \$1

- Now, if we extend the horizon a little bit further, say out to eight, we would find that this is the best thing to do. So if we go this way, we get -1, -1, -1, until we hit the jackpot here and get \$1M. Clearly if you sum this up, it's a pretty good prize.
- If we go this way and touch that \$1 over and over again, we get this. So clearly as we expand our finite horizon trivially up to say eight steps, going this way and tagging at one million is the best thing to do.
- If we carried it even further, we'd discover that then we should come back this way and go to that dollar and tag it over and over and over again.

What to optimize?

infinite horizon
 $\sum_{i=1}^{\infty} r_i$
finite horizon
 $\sum_{i=1}^n r_i$

\$1		\$1M	-1	-1
0				-1
jackpot	-1	-1	-1	-1

- Let me formalize these a little bit. With the infinite horizon what we're trying to maximize is the sum of all rewards over all of the future. So it's the sum of each of these rewards for i equals one to infinity.
- The finite horizon is very similar, it's just we don't go to infinity. So for optimizing over horizon of four steps, n would be four. We're just trying to maximize the sum of the reward for the next four steps.

What to optimize?

infinite horizon


$$\sum_{i=1}^{\infty} r_i$$

finite horizon

$$\sum_{i=1}^n r_i$$

discounted reward

$$\sum_{i=1}^{\infty} \lambda^{i-1} \cdot r_i$$

\$1		\$1/M	-1	-1
0				-1
	-1	-1	-1	-1

- Now, there is yet another formulation that if you think back to that lecture a while back about what's the value of a future dollar. We can dig that up and it makes a lot of sense in terms of reinforcement learning.
- So remember that if it takes us say, four years to get a dollar, that dollar is less valuable than say if it takes one year. And the same way, if it takes, say, eight steps to make a dollar, that dollar is less valuable than a dollar I can get just in one step.
- And the way we represent that is very simple. Just like we represented the sum of future dividends and it looks like this, it's called discounted reward. So instead of just summing up the r sub i 's, we multiply it by this factor γ to the i minus 1, such that our immediate reward, the very next one we get, whatever γ is when it gets raised to the zero power is just one.
- So that means for the very next step we get r . But for the step after it, it's γ to the one. So it devalues that reward a little bit.

What to optimize?

infinite horizon

$$\sum_{i=1}^{\infty} r_i$$

finite horizon

$$\sum_{i=1}^n r_i$$

discounted reward

$$\sum_{i=1}^{\infty} \lambda^{i-1} \cdot r_i$$

\$1		\$1/M	-1	-1
0				-1
	-1	-1	-1	-1

$0 < \lambda \leq 1.0$

- Gamma is a value between zero and one, the closer it is to one, the more we value rewards in the future. The closer it is to zero, the less we value rewards in the future.
- In fact, if gamma is set equal to one, this is exactly the same as the infinite horizon. But gamma relates very strongly to interest rates if you recall.

What to optimize?

infinite horizon
 $\sum_{i=1}^{\infty} r_i$

finite horizon
 $\sum_{i=1}^n r_i$

discounted reward
 $\sum_{i=1}^{\infty} \lambda^{i-1} \cdot r_i$

$0 < \lambda \leq 1.0$.95

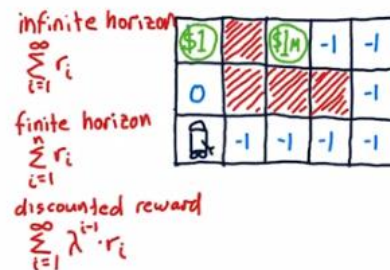
\$1		\$M	-1	-1
0				-1
	-1	-1	-1	-1

- So, if say, gamma were 0.95 it means each step in the future is worth about 5% less than the immediate reward if we got it right away. This is the method that we use in q learning. One reason is that the math turns out to be very handy, and it provides nice conversion properties.

Which Approach Gets \$1M

Q Which gets \$1M?

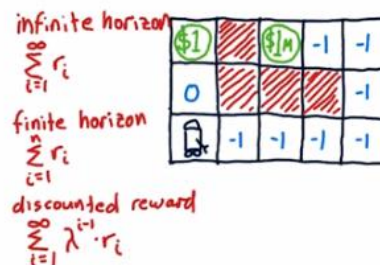
- ☐ infinite horizon
- ☐ finite n=4
- ☐ finite n=10
- ☐ discounted $\lambda=.95$



- I want you to consider each of these optimizations and answer which of those will get us to the \$1 million.
- In other words, if the robot is trying to maximize the sum over these horizons, which ones will lead it to a policy that causes it to reach that \$1 million?

Q Which gets \$1M?

- ☒ infinite horizon
- ☐ finite n=4 X
- ☒ finite n=10
- ☒ discounted $\lambda=.95$



- So there are actually several that satisfy that. Infinite horizon is a little bit iffy because the robot can go this way and get a dollar on every other move and that will add up to infinity.
- It can go here and get the \$1 million and then come back and do that and it will add up to infinity. So it's possible that infinite horizon will cause it to do that but there's two equivalent solutions.
- Finite with n=4, no it won't get to that \$1 million. Because if it tries to go that way, it'll only get negative reward here, but it'll get positive reward if it goes that way.
- However, if we let n go out to 10, boom, it'll reach that \$1 million.
- And finally, discounted reward, where each dollar in the future is only worth further and further into the future. Still, by the time we get to the eight steps that it takes to reach this reward it's still so huge that that's clearly the optimal thing to do.
- Okay, so those are the answers to which horizons will cause us to get to that \$1 million.

Summary

RL summary

- RL algos solve MDPs
- $S, A, T[s,a,s'], R[s,a]$
- Find $TT(s) \rightarrow a$
- Map trading to RL



- Let's summarize things and wrap up this lecture. I just want to repeat the points so you so reinforcement learning is something that we can use in trading.
- The problem for reinforcement learning algorithms is a Markov decision problem. And reinforcement learning algorithms solve them.
- A Markov decision problem is defined by S, A, T , and R , where S is the potential states, A are the potential actions, T is a transition probability, which is given I'm in state s , I take action a , what's the probability I'll end up in state s' , and R is the reward function.
- The goal for reinforcement learning algorithm is to find a policy, π , that maps a state to an action that we should take, and its goal is to find this π such that it maximizes some future sum of the reward.
- We talked about that being either infinite horizon, fixed horizon, or discounted sum. We can map our task for trading to reinforcement learning and it works out like this.
 - S , our states, are features about stocks and whether or not we're holding a stock.
 - Actions are buy, sell, or do nothing.
 - The transition function here is the market.
 - And finally, the reward function is how much money we get at the end of a trade.
- So, we can apply reinforcement learning algorithms to find this policy. We've mentioned a few of those algorithms, for example policy iteration, and value iteration, and Q learning, but we haven't talked in detail what they are, and that's the subject of lessons coming up.
- Okay, that's it for reinforcement learning, I'll see you again soon.