# MITOCW | Lecture 22 | MIT 6.832 Underactuated Robotics, Spring 2009

**RUSS TEDRAKE:** Today is sort of the culmination of everything we've been doing in the model free optimal control. OK. So we talked a lot about the policy gradient methods. So under the model free category here. And we've talked a lot about model free policy gradient methods. And then the last week or so, we spent talking about model free methods based on learning value functions.

OK. Now both of those have some pros and some cons to them. OK? So the policy gradient methods, what's good about policy gradient methods?

**STUDENT:** They scale--

**RUSS TEDRAKE:** They scale with--

[INTERPOSING VOICES]

**RUSS TEDRAKE:** OK. And they can scale well to high dimensions. We'll qualify that. Right? It's actually still only a local search, that's why they scale. And the performance of the model free methods degrades with the number of policy parameters. But if you have an infinite dimensional system with one parameter you want to optimize, then you're in pretty good shape with a policy gradient method. Right?

OK. What else? What are other pros and cons of policy gradient methods? What's a con? Well, I said a lot of it in the parentheses already. But it's local. What are some other cons about policy gradient methods?

**STUDENT:** [INAUDIBLE].

**RUSS TEDRAKE:** Yeah. Right. So this performance degradation typically is summarized by people saying they tend to have high variance. Right? Variance in the update, which can lead to mean that you need many trials to converge.

I mean, fundamentally, if we're sampling policy space and making some stochastic update, it might be that it requires many, many samples, for instance, to accurately estimate the gradient. And if we're making a move after every sample, then it might take many, many trials for us to find the minimum of that. It's a noisy descent. Yeah?

**STUDENT:** You also have to choose like a [INAUDIBLE].

**RUSS TEDRAKE:** Good. That wasn't even on my list. But I totally agree. OK. I'll put it right up here.

There's one other very big advantage to the policy gradient algorithms. We take advantage of smoothness. They require smoothness to work. That's both a pro and a con, right?

But the big one that we haven't said yet, I think, is the convergence is sort of virtually guaranteed. You're doing a direct search. And exactly, you're doing a stochastic gradient descent in exactly the parameters you care about. Convergence is sort of trivial and guaranteed. OK?

That turns out to be probably one of the biggest motivating reasons for the community to have put their efforts into policy gradient. Because if you look at the value function methods, in many cases-- now, I told you about one case with function approximation, still linear function approximation, where there are stronger convergence results. And that was the least squares policy iteration.

But most of the cases we've had, the convergence results were fairly weak. We told you that temporal difference learning converges if the policy remains fixed. OK? But if you're not careful, if you do temporal difference learning with the policy changing, with a function approximator involved, convergence is not guaranteed.

OK? In fact, they often, I mean a lot of these methods struggle with convergence. Not just the proofs, which are more involved. But there's a handful of, I guess, sort of in the big switch from value methods to policy gradient methods, there are a number of papers showing sort of trivial examples of-- can I call them TD control methods? So temporal difference learning where you're actually also updating your policy of TD control methods with function approximation, which diverge. Right?

There was even one that-- I think it might have been, I forget whose-- it might have been Lehman Baird's example. But where they actually showed that the method will actually oscillate between the best possible representation of the value function and the worst possible representation of the value function. And it's sort of stably oscillated between the two. Right? Which was obviously something that they cooked up. But still, that makes the point. Right?

Even the convergence result we did give you for LSPI, least squares policy iteration, still had no guarantee that it wasn't going to, it could certainly oscillate. They gave a bound on the oscillation. But that bound has to be interpreted. Even the LSPI could still oscillate. And that's one of the stronger convergence results we have.

OK. But they're relatively efficient. Right? So we put up with a lot of that. And we keep trying to use them because they're efficient to learn in the sense that you're just learning a scalar value over all your states and actions. That's a relatively compact thing to learn. I told you, I tried to argue last time that it's easier than learning a model even by just dimensionality arguments.

And they tend to be efficient because the TD methods in particular reuse your estimates. And they tend to be efficient in data. They reuse old estimates. They use your old estimate of the value function to update your new estimate of the algorithms. So when they do work, they tend to learn faster. And they can, with the least squares methods, they tend to be efficient in data. And therefore, in time. Number of trials. When these things do work, they're the tool of choice.

The problem is-- and there are great examples of them working-- but there's not enough guarantees of them working. And if you want to sort of summarize why these value methods struggle and why they can struggle to converge and they even diverge, you can sort of think of it in a single line, I think. The basic fundamental problem with the value methods is that a very small change in your estimate of the value function, if you make a little change, can cause a dramatic change in your policy. Right?

So let's say my value functions tip this way. Right? And I change my parameters a little bit. Now it's tipped this way. My policy just went from going left to going right, for instance. And now you're trying to update your value function as the policy changed. And just things can start oscillating out of control. Does that make sense?

OK. That's a reasonably accurate, I think, lay of the land in the methods we've told you about so far. If you can find a value method that converges nicely, use it. It's going to be faster than a policy gradient method. It's more efficient in reusing data. You're learning a fairly compact structure. Value iteration has always been our most efficient algorithm, when it works. But the policy gradient algorithms are guaranteed to work. And they're fairly simple to implement. And they can just be sort of local search in the policy space. Directly in the space that you care about, really your policy.

So the big idea, which is the culmination of the methods we've talked about in the model free stuff so far, is to try to take the advantages of both by putting them together. Represent both a value function and a policy simultaneously. There's extra representational costs there. But if you're willing to do that and make slower changes to the policy based on guesses that are coming from the value function, then you can overcome a lot of the stability problems of the value methods. You get the strong convergence results of the policy gradient. And you get some of the more, ideally, efficiency. You can reduce your variance of your update. You make more effective updates by using a value function. OK?

So the actor is the playful name for the policy. And the critic is your value estimate telling you how well you're going to do. And one of the big ideas there is you'd like it to be a two time scale algorithm. Policy is changing slower than the greedy policy from the value function.

OK. So the idea is an actor critic are actually very, very simple. The proofs are ugly. There's only a handful of papers you've got to look at if you want to get into the dirt. But these, I think, are the algorithms of choice today for a model free optimization. OK. So just to give you a couple of the key papers here. So Konda and Tsitsiklis. John's right upstairs. Had an actor critic paper in 2003 that has all the algorithm derivation and proofs. Sutton has a similar one in '99 that's called Policy Gradient. But it's actually the same sort of math as in Konda and Tsitsiklis.

And then our friend Jan Peters has got a newer take on it. He calls it Natural Actor Critic, which is a popular one today. It should be easy to find. OK. So I want to give you the basic tools. And then instead of getting into all the math, I'll give you a case study, which was my thesis. Works out.

So probably John already said quickly what the big idea was. Right? So John told you about the reinforced type algorithms and the weight perturbation. In the reinforced algorithms, we have some parameter vector. Let's call it alpha. And I'm going to change alpha with a very simple update rule. In the simple case, maybe I'll run my system twice.

I'll run it once with the-- I'll get once, I'll sample the output with alpha. And then once I'll do it with alpha plus some noise. Let's say I'll run it from the same initial condition. Compare those two. And then multiply the difference times the noise I added. Right? And that's actually a good estimator, a reasonable estimator of the gradient. And if I multiply by the learning rate, then I've got a gradient descent type update. OK?

So this is not useful in its current form. John told you about the better forms of it, too. But the problem with this is that I have to run the system twice from exactly the same initial conditions. You don't want to run two trials to simulate the thing exactly twice for every one update. And it sort of assumes that this is a deterministic update. The more general form here would be to not keep, not run the system twice. But use, for instance, some estimate of what reward I'd expect to get from this initial condition. And compare that to the learning trial.

So we just went from policy gradient to actor critic just like that. This is the simplest form of it. But let's think about what just happened. So if I do have an estimate of my value function, I have an estimate of my cost to go from every state. Right? Then that helps me make a policy gradient update. Because if I run a single trial, then I can compare the reward I expected to get with the reward I actually got very compactly. OK?

So this is the reward I actually got. I run a trial, one trial. Even if it's noisy with my perturb parameters, I change my parameters a little bit. I run a trial. And what I want to efficiently do is compare it to the reward I should have expected to get, given I had the parameters I had a minute ago. Right? That's nothing but a value function right here. OK?

So the simplest way to think about an actor critic algorithm is go ahead and use a TD learning kind of algorithm. Every time I'm running my robot, go ahead and work on in the background learning a value function of the system. And simply use that to compare the samples you get from your policy search. Do you guys remember the sort of weight perturbation type updates enough for that to make sense? Yeah?

**STUDENT:** So in this case, that [INAUDIBLE] into your system but just through some expectation.

**RUSS TEDRAKE:** Excellent. That's where you're getting it. From temporal difference learning. In the case of a stochastic system, where both of these are going to be noisy random variables, this actually can be better than running it twice. Because this is the expected value accumulated through experience. Right? And that's what you really want to compare your noisy sample to the expected value. So in the stochastic case, you actually do better by comparing it to the expected value of your update.

What you can show by various tools is that comparing to the expected value of your update, which is the value function here, can dramatically reduce the variance of your estimator. OK? You should always think about policy gradient as every one of these steps trying to estimate the change in the performance based on a change in parameters. But in general, what you get back is the true gradient plus a bunch of noise, because you're just taking a random sample here in one dimension of change. If this is a good estimate of the value function, then it can reduce the variance of that update. Question?

**STUDENT:** [INAUDIBLE].

**RUSS TEDRAKE:** The guarantees of convergence are still intact because you're doing gradient descent. You can actually do, you can do almost anything here. This can be zero. And gradient descent, the policy gradient actually still converges. It doesn't converge very fast. But you can still actually show that it'll, on average, converge. OK? So it's actually quite robust to the thing you subtract out. Because, especially if this thing doesn't depend on alpha, then it has zero expectation. So it doesn't even affect the expected value of your update. So it actually does not affect the convergence results at all.

So the convergence results are still intact. But the performance should get better because you have a better estimate of your J. Right? And that should be intuitively obvious, actually. Right? If I did something and I said, how did I do?

And [INAUDIBLE] just always said, you should have gotten a four every single time. If I got a lousy estimator of how well I should have done, I'd say, OK. Look, I got a six that time. And he says, you should have had a four. Six, you should have had a four. Then he's giving me no information. And that's not helping me evaluate my policy. Right?

If someone said, OK. We did something a little different. I expected you to get a six, but you got a 6.1. Well, that's a much cleaner learning signal for me to use.

**STUDENT:** [INAUDIBLE] the worst possible--

**RUSS TEDRAKE:** Yeah, absolutely. So that's the important point is that it's got to be uncorrelated with the noise you add to your system. OK? If it's not correlated with the noise you add in, then it actually goes away in expectation. So the variance can be very bad if you have the worst possible value estimate. But the convergence still happens.

Like I said, zero actually works. Right? Which is sort of surprising. Right? If I have a reward function that always returns between zero and 10, and I'm trying to optimize my update, then I would always move in the direction of the noise I add. But I move more often in the ones that gave me high scores. And actually, it still does a gradient descent on the cost function. It's actually worth thinking about that. It's actually pretty cool that it's so robust, that estimator. But certainly with a good estimator, it works better.

I don't know how much John told you. But we don't actually like talking about the variance. We like talking about the signal to noise ratio. Did you tell them about the signal noise ratio, John?

**STUDENT:** I don't remember.

**RUSS TEDRAKE:** Quickly? Yeah. So John's got a nice paper. Maybe he was being modest. John has a nice paper analyzing the performance of these with a signal to noise ratio analysis, which is another way to look at the performance of the update. So that's enough to do, to take the power of the value methods and start putting them to use in the policy gradient methods. OK?

The cool thing is, like I said, as long as it's uncorrelated with z, it can be a very bad approximate of the value function. It won't break convergence. The better the value estimate, the faster your convergence is. OK? This isn't the update that people typically use when they talk about actor critic updates. The Konda and Tsitsiklis one has a slightly more beautiful thing. This is maybe what you think of as an episodic update. Right?

This is, I just said we started initial condition x. Maybe I should right an x zero or something. But we just start with initial condition x. We run our robot for a little bit with these parameters. We compare it to what we expected. And we make an update maybe once per trial. That's a perfectly good algorithm for making an update once per trial.

There's a more beautiful sort of online update. Right? If you actually want to, let's say you have an infinite horizon thing. Infinite horizon problem. There's actually a theorem, I've debated how much of this to go into. But I'll at least list the theorem for you because it's nice.

They call it the policy gradient theorem, which says partial J partial alpha, where in the infinite horizon case typically there's different ways to define infinite horizons. This is typically done in an average reward setting. It can be made to work for other formulations. But I'll just be careful to say the one that I know it's a correct proof for.

The policy gradient can actually be written as-- let me write it out. This guy is the stationary distribution of executing of the state action, of executing pi of alpha. This guy is the Q function executing alpha, the true Q function. And this is the state action. And this guy is actually the gradient of the log probabilities, which is the same thing we saw in the policy gradient algorithms. The log probabilities of executing pi.

Yeah. Gradient of the log probability. I'm not trying to give you enough to completely get this. But just I want you to know that it exists and know where to find it. And what it reveals is a very nice relationship between the Q function and the gradients that we were already computing in our reinforced type algorithms. OK?

And it turns out an update of the form-- this is gradient of the log probabilities again. I'll just write it. That would be doing gradient descent on this if you're running from sample paths. This term disappears if I'm just pulling x and u from the distribution that happens when I run the system that gives me this stationary distribution coefficient for free. OK?

And then if I could somehow multiply the true Q function times my eligibility-- this one, I definitely have access to. This one, I can only guess, because I have access to my policy. I can compute that.

But this guy, I have to estimate. OK? So if I put a hat on there, then that's actually a good estimator of the policy gradient using an approximate Q function. And in the case where you hold up your updates for a long time and then make an estimate in an episodic case, it actually results in that actual algorithm. OK?

Getting to that from with a more detailed explanation is painful. But it's good to know. I think the way you're going to appreciate actor critic algorithms, though, is by seeing them work. OK? So let me show you how I made them work on a walking robot for my thesis. I've already done this. Is it going to turn on?

Since I think everybody's here, maybe we should do, while it's booting, I'll do a quick context switch. Let's figure out projects real quick. And then we'll go back. I don't want to run out of time and forget to say all the last details about the projects. Yeah? Somehow, I never remember to post the syllabus with all the dates on there. We're posting it now. But I can't believe I didn't post a long time ago on the website.

But I hope you know that the end of term is coming fast. Yeah? And you know you're doing a write up. Right? And that write up, we're going to say that the 21st, which is basically the last day I can possibly still grade them by, the write up as described, which is sort of I said six pages-- sort of an [INAUDIBLE] type format-- is going to be due on May 21 online.

OK. But next week, last week of term already, we're going to try to do oral presentations so you guys can tell me-- eight minutes each is what works out to be. You get to tell us what you've been working on. OK? For each project, there are a few of you that are working in pairs. But we'll still just do eight minutes per project. And we have 19 total projects. So I figure we do eight-- sorry, nine-- next Thursday, which is going to be the 14th. Is that right? 5-14. And nine on 5-12, working back here, which leaves some unlucky son of a gun going on Thursday.

And the way I've always done this is I have a MATLAB script here that has everybody's name in it. Yeah. Why is it not on here? OK. I have a MATLAB script with all your names in it. OK? And I'm going to do a rand perm on the names. And it'll print up what day you're going.

**STUDENT:** Maybe in fairness to that person, would we all be happy to stay an extra eight minutes on whatever it is? Tuesday?

**RUSS TEDRAKE:** Let's do it this way first. And then we'll figure it out.

[LAUGHTER]

And yes. So I'm going to call rand perm in MATLAB. And for dramatic effect this year, I've added pause statements between the print commands.

[LAUGHTER]

So we should have a good time with this, I think. I will, at least. OK. Good. Let's make this nice and big. I actually was going to just use a few slides from the middle of this. But I thought I'd at least let you see the motivation behind it, which very well. And I'll go through it quickly. But just to see at least my take on it in 2005, which hasn't changed a whole lot. It's matured, I hope.

But I've told you about walking robots. We spent more time talking about passive walkers than we talked about some of the other approaches. But there's actually a lot of good walking robots out there. Even in 2005, there were a lot of good ones. This one is M2 from the Leg Lab. The wiring could have been cleaner. But it's actually a pretty beautiful robot in a lot of ways. The simulations of it are great. It hasn't walked very nicely yet. But it's a detail.

[LAUGHTER]

Honda's ASIMO had sort of the same sort of humble beginnings. As you can imagine, it's not really fair that academics have to compete with people like Honda. Right? I mean, so our robots looked like what you saw on the last page. And ASIMO looks like what it looks like. But it's kind of fun to see where ASIMO came from. So this is ASIMO 0.000. Right? And this is actually the progression of their ASIMO robots.

That's the first one they told the world about in '97. Rocked the world of robotics. I was in the Leg Lab, remember. At the time, we were kind of like, oh wow. They did that? Wow. That sort of certain changes our view of the world. That's P3. And that's ASIMO. Right? Really, really still one of the most beautiful robots around.

You know about under-actuated systems. I don't have to tell you that. You know about acrobots. You know walking is under-actuated. Right? Just to say it again-- and I said it quickly-- but essentially, the way ASIMO works is they are trying to avoid under-actuation. Right? When you watch videos of ASIMO walking, it's always got its foot flat on the ground. There's an exception where it runs with an [? arrow ?] phase that you need a high speed camera to see. But--

[LAUGHTER]

It's true. And that's just a small sort of deviation where they sort of turn off the stability of the control system for long enough. And they can recover. Their controller is robust enough in the flat on the ground phase that they can catch small disturbances which are their uncontrolled aerial phase.

So for the most part, they keep their foot flat on the ground. They assume that their foot is bolted to the ground, which would make them fully actuated. Right? And then they do a lot of work to make sure that that assumption stays valid.

So they're constantly estimating the center of pressure of that foot and trying to keep it inside the foot, which means the foot will not tip. And this is if you've heard of ZMP control, that's the ZMP control idea. OK? And then they do good robotics in between there. They're designing desired trajectories carefully. They're keeping the knees bent to avoid singularities. They're doing some-- depends on the story. I've heard good claims that they do very smart adaptive trajectory tracking control. I've heard more recently that they just do PD control. And that's good enough because they've got these enormous gear ratios. And that's good enough.

OK. So you've seen ASIMO working. The problem with it is that it's really inefficient. Right? Uses way too much energy. Walks slowly. And has no robustness. Right? I've told you that story. Here's one view of everything we've been doing in this class. The fundamental thing that ASIMO is not doing in its control system is thinking about the future. OK?

So if you were taking a reinforcement learning class, you would have started off with talking about delayed reward. And that's what makes the learning problem difficult. Right? I didn't use the words delayed reward in this class. But it's actually exactly the same thing. The fact that we're optimizing a cost function over some interval into the future means that I'm thinking about the future. I'm planning over the future.

I'm doing long term planning. And if you think about having to wait to the end of that future to figure out if what you did made sense, that's the delayed reward problem. It's exactly the thing that reinforcement learning folks use to convince other people that reinforcement learning is hard. OK?

So the problem in walking is that you could do better if you stopped just trying to be fully actuated all the time. We start thinking about the future. Think about long term stability instead of trying to be fully actuated. OK? The hoppers, there are examples of really dynamically dexterous locomotion.

But there's not general solutions to that. That's what this class has been trying to go for. So we do optimal control. We would love to have analytical approximations for optimal control for full humanoids like ASIMO. Love to have it. Don't have it. We're not even close. You know the tools that we have now.

But even if we did have an analytical approximation of optimal control-- maybe we will in a few years, who knows-- we'd still like to have learning. Right? All this model free stuff is still valuable because, if the world changes, you'd like to adapt. Right? So my thesis was basically about trying to show that I could do online optimization on a real system in real time. And I told you about Andrews Helicopters.

There's a lot of work on Sony Dogs that do loop trajectory optimization from trial and error. So Sony came out. And they had this sort of walking gait. Right? And then people start using them for soccer. And they said, how fast can we make this thing go?

It turns out the fastest thing they do on an IBO is to make it walk on its knees like this. And they found that from a policy gradient search where they basically made the dog walk back and forth between sort of a pink cone and a blue cone, just back and forth all day long doing policy gradient. And they figured out this is a nice fast way to go. And then they won the soccer competition.

[LAUGHTER]

Not actually sure if that last part is true. I don't know who won. But I'd like to think it's true. There are people that do a lot of walking robots. I think I showed you the UNH bipeds that were some of the first learning bipeds. Right? I told you about these all term. Right? So there's large continuously in action spaces, complex dynamics. We want to minimize the number of trials. The dynamics are tough for walking. Because of the collisions. And there's this delayed reward.

So in my thesis, the thing I did was tried to build a robot that learned well. That was my goal. I simultaneously designed a good learning system but also built a robot where learning would work really well. Instead of working on ASIMO, I worked on this little dinky thing I call Toddler. Yeah? And I spent a lot of time on that little robot. So you know about passive walking.

This is the simplest, this is the first passive walker I built. Passive walking 101 here. So it's sort of a funny story. I mean, I was in a neuroscience lab. I worked with the Leg Lab. But my advisor was in neuroscience. They spent lots of money on microscopes and lots of money. So at some point, I said, can I spend a little bit of money on a machine shop? And I promise it'll cost less than that lens you just spent on that one microscope?

And so, he gave me a little bit of money to go down. I was basically in a closet at the end of the hall. My tools looked like things like this. Like, I couldn't even afford another piece of rubber when I cut off a corner. And that's actually a CD rack that I got rid of somewhere.

And that's my little wooden ramp that I was using for passive walking. But I built these little passive walkers with a little sureline CNC mill that walked stably in 3D down a small ramp. Yeah? I don't know why it's playing badly.

So that was the first steps. If we're going to do walking, it's not hard. Those feet are actually CNC-ed out. I spent a lot of time on those feet. They're a curvature that was designed carefully to get stability.

**STUDENT:** It's just a simple [INAUDIBLE].

**RUSS TEDRAKE:** Yeah. Just a pin joint. That's a walking robot. At the time, people had been working on passive walkers for a long time. But nobody had sort of done the obvious thing, which is add a few motors and make it walk on the flat. Nobody had done it. So that's what I set out to do with the learning. Turns out a few people did it around the same time. So we wrote a paper together.

But the basic story was we went from this simple thing that was passive to the actuated version. The hip joint here on this robot is still passive. OK? Put actuators in at the ankle. So we had a new degrees of freedom with actuators so that it could push off the ground but still keep its mostly passive gait. Actually, it's extruded stock here stacked with gyros and rate gyros and all the kinds of sensors. It's got a 700 megahertz Pentium in its belly, which kind of stung.

In retrospect, I couldn't make very many efficiency arguments about the robot because it's carrying a computer the size of a desktop at the time. You know? And so, there's five batteries total on the system. Right? Those four are powering the computer. There's one little one in there that's powering the motors. And still those big four drained like 50% faster than the other ones. But it's computationally powerful. Right? I actually ran a little web server off there just because I thought it was funny.

[LAUGHTER]

And the arms look like I've added degrees of freedom. But actually, they're mechanically attached to the opposite leg. So when I move this, that bar across the front was making that coupling happen, which is important for the 3D walking. Because if you want to walk down, if you have no arms actually and you swing a big heavy foot, then you're going to get a big yaw moment. And the robots often walk like this and went off the side of the ramp. So you put the big batteries on the side and then everything walks straight. And it's good.

So in total, there's nine degrees of freedom if you count all the things that could possibly move. And there's four motors to do the controls. So that's under-actuated. Right? We've got the robot dynamics. Oops. I've used a Mac now. I used to use a Windows. So apparently my u is now O hat.

[LAUGHTER]

Sorry. That's actually tau. OK. So tau. Yeah. So I had most almost the manipulator equations. But I had to go through this little hobby servo. So it wasn't quite the manipulator equation. And the goal was to find a control policy pi that was-- so it was already stable down a small ramp.

And the way I formulated the problem is I wanted to take that same limit cycle that I could find experimentally down a ramp and make it so it worked on whatever slope. So make that return map dynamics invariant to slope. And to do that, you need to add energy. And you need to find a control policy.

So my goal was to find this pi, stabilize the limit cycle solution that I saw downhill to make it work on any slope. So this was just showing that Toddler, with its computer turned off, its motors are turned on-- actually, this one is even the motors are off. And there's just little splints on the ankle. Just showing that it was also a passive walker. And showing that I dramatically improved my hardware experience by getting a little proform treadmill that was off of the back lot. And I painted it yellow and stuff.

So this thing would actually walk all day long. It would. So it's a little trick. At the very edge, in the middle, there's nothing going on. But at the very edge of the treadmill, I put a little lip there. So if it happened to wander itself over to the side, it had that lip and walked back towards the middle. OK? And I put a little wedge on the front and on the back so it sort of would try to stay in the middle of the treadmill. And that thing would just walk all day long. It would drive you crazy hearing those footsteps all day long.

[LAUGHTER]

But it worked. It worked well. It still works today, most of the time. So I use the words policy gradient. But this was really an actor critic algorithm. So I used linear, it's actually a very centric grid in phi. But a linear function approximator. And the basic story was policy gradient. OK?

So it was something in between this perfectly online at every dt, make an update. And it was not quite the episodic run a trial, stop, run a trial, stop. The cost function was really a long term cost. But I did it once per footstep. OK?

So every time the robot literally took a footstep, I would make a small change to the policy parameters. See how well it walked. See where it hit the return map. And then change the parameters again. Change the parameters again. And every time that foot hit the ground, I would evaluate the change in walking performance and make the change in W based on that result. OK. I'll show you the algorithm that I used a second, which you'll now recognize.

So the way to think about that sampling in W is that you're estimating the policy gradient. And you're performing online stochastic gradient descent. Right? So the time, the way I described the big challenge is, what is the cost function for walking?

And how do you achieve fast provable convergence, despite noisy gradient estimates? You guys know about return maps. This is my picture of return maps from a long time ago. So this is the Van der Pol Oscillator. This is the return map here.

The important point here, so this is the samples on the return map. This is the velocity at the n-th crossing versus the velocity at the n-th plus 1 crossing. The blue line is the line of slope one. So it's stable, the Van der Pol Oscillator, because it's above the line here and below the line there. And you can evaluate local stability by linearizing and taking the eigenvalues. We've talked about these things.

But I don't know if I made the point nicely before. That if you can pick anything, if you want your return map to look like anything in the world, if you could pick, what would you pick? You'd pick a flat line. Right? That's the deadbeat controller. I used the word deadbeat.

So that's where my cost function came from. The cost function that tried to say that the robot was walking well-- wow-- penalized my instantaneous cost function, penalized the square distance between my sample on the return map and the desired return map, which is that green line.

OK. So basically I wanted, I tried to drive the system to have a deadbeat controller. And I did, and there's limits. There's actuator limits that's going to mean it's never going to get there. But my cost function was trying to force that. Every time I got a sample, it was trying to push that sample more towards the deadbeat controller.

Then basically, it worked. It worked really well. The robot began walking in one minute, which means it started getting its foot cleared. So the first thing, if I set W equal to zero, it was configured so that when the policy parameters were zero, it was a passive walker.

So I put it on flat. I picked it up. I picked it up a lot. And I drop it. It runs out of energy and stands still. Because it was just a passive walker, it's not getting energy from-- it's only losing energy. OK?

So now, I pick it up. I drop it. And every time it takes a step, it's twiddling the parameters at the ankle a little bit. OK? So it started going like this a little bit. And then after about a minute of dropping it-- and quickly, I wrote a script that would kick it into place so I stopped dropping it-- OK. So I gave a little script so it would go like this. And in about one minute, it was sort of marching in place. OK?

And then I started driving it around. I had a little joystick which said, I want your desired body to go like this. And it started walking around. And in about five minutes, it was sort of walking around. I'll show you the video here in a second. And then, I said 20 minutes for convergence. That was conservative.

Most of the time, it was 10 minutes. It would converge to the policy that was locally optimal in this policy class. But it worked very well. And I just sort of sent it off down the hall. And it would walk. OK? And doing the stability analysis, it showed the learn controllers is considerably more stable than the controllers I designed by hand, which I spent a long time on those, too.

And now, here's a really key point. OK? So you might ask, how much is this sort of approximate value function, how important is that? That's sort of the topic for today. Right? How important is this approximate value function? Well, it turns out, if I were to reset the policy, if I just set the policy parameters to zero again but keep the value function from the previous time it learned, then the whole thing speeds up dramatically. So instead of converging in 20 minutes, the thing converges in like two minutes. OK?

So just by virtue of having a good value estimate there, learning goes dramatically faster. And it's only when I have to learn them both simultaneously that it takes more like 10 or 20 minutes. And it worked so fast that I never built a robot. I never built a model for the robot.

Actually, I tried later. It's tough. The dynamics of that-- I mean, it's a curved foot with rubber on it, right? It was just very hard to model accurately. And I didn't need to. It worked. It learned very quickly. Quickly enough that it was adapting to the terrain as it walked.

All right. So here's the Poincaré maps from that little Toddler robot projected onto a plane. So I picked it up a bunch of times. I tried to make it just walk in place here. Before learning, it was obviously only stable at the zero, zero fix point. It was running out of energy on every step and going to zero. After learning, this is what the return map looked like. OK? So it actually could start from stopped reliably. Right?

This is actually far better than I expected it to do. If you do your little staircase analysis of this, so it gets up to the fixed point in two steps or three steps for most initial conditions. Right? And from a very large range of initial conditions, as large as I care to sample from. So you could go up there-- and people did actually.

We had a little-- after we got it working, the press came. And then everybody was asking me, the reporters were saying, can I have my kid play with the robot? Or can we put on a treadmill at the gym?

Rich Sutton put his fingers under it and was like playing with it at dips one time. So it got disturbed in every possible way. And for the most part, it worked really-- I mean, so if you give it a big push this way, it actually takes energy out and comes back and recovers in two steps. You stop it. It goes back up. And it recovers.

And in the worst case, I had some demo to give or something. And I took it out of the case. It had traveled through the airport. The customs people always asked me if it had commercial value. It doesn't have commercial value.

But it broke somewhere in the travel. And I didn't realize it. I picked it up and headed to do its demo. And it's going like this. And it's sort of walking. And it looks a little funny. And people are so relatively happy with it. Turns out the ankle had completely snapped. But in just a few steps, it actually found a policy that was walking with a broken ankle.

[LAUGHTER]

So it works. It really worked. It really did work. I'm not sure-- I mean, yeah. It really worked. OK. So here's the basic video. This was the beginning. I was paranoid. So I had pads on it to make sure it didn't fall down and break. This is the little policy that would kick it up into a random initial condition like that. And now it's learning. It falls down. I don't know why it's playing so badly.

This is after a few minutes. It's stepping in place. It's walking. And then I started driving it around. I say, OK. Let's walk around. And it stumbles. But really, really fast, it learned a policy that could stabilize it. Right? And after a few minutes, this is the disturbance tests. I actually haven't shown these in a long time. It's really robust to those things. And then you can send it off down the hall.

And now, this is a little robot with big feet admittedly. But you know, it's like the linoleum in E25-- this is in E25-- was really not flat. I mean, it's sort of embarrassing to tell people, look at the floor. It's not flat. But for that robot, I mean there's huge disturbances as it walked down the floor. But the policy parameters were changing quite a bit. You could walk off tile onto carpet. And in a few steps, it would adjust its parameters and keep on walking. This was it walking from E25 towards the Media Lab, if you recognize that.

OK. So one of the things I said is that one of the problems with the value estimate is you make a small change in the value function, you get a big change in the policy. Theoretically, no problem. In practice, you don't probably want that. Right?

One of the beautiful things about the policy gradient algorithms is you make a small change to the policy. It doesn't look like the robot's doing crazy things. So every time, everything you saw there, it was always learning. Right? Learning did not look like a big deviation from nominal behavior. I never turned off learning with this. Right?

It turned out in the policy gradient setting, I could add such a small amount of noise to the policy parameters, which was a very central grid over the state space, such a small amount of noise that you couldn't even tell it was learning. Right? But it was enough to pull out a gradient estimate and keep going. So it didn't look like it was trying random things. But then, if it walked off on the carpet and did a bad thing, it would still adapt. That was something I didn't expect. It just was a very nice sort of match between the amount of noise you had to add and the speed of learning.

The value estimate was a low dimensional approximation of the value function. Very low. Like ridiculously low. One dimension. Right? But it was sufficient to decrease the variance and allow fast convergence. I never got it to work before I put a value function in. And here's this question. So I ended up choosing gamma to be pretty low. Gamma was 0.2. I did try with zero times. What did that mean?

So that's how far I carried back my eligibility, which means how many steps am I looking at it. So that you could think of it as a receding horizon optimal control. How many steps ahead do you look? Right. Except it's discounted. OK?

So 0.2 is really heavy discounted. Really, really heavy. It means I was basically looking one step ahead and not worrying about things well into the future, which made my learning faster but meant I didn't take really aggressive corrections that were multi-step sort of corrections. Only very rarely, if the cost really warranted it. OK. So that was always something I thought would be cool if I could get that higher and show a reason why multi-step corrections made it a lot more stable.

**STUDENT:** Did it not work as well?

**RUSS TEDRAKE:** It didn't learn as fast. At some point, I decided I'm going to try to make the point that these things can really learn fast. And so, I started turning all the knobs. Simple policy, simple value function, low look ahead. And it worked. But it was fast.

**STUDENT:** Is gamma used [INAUDIBLE] the same as lambda?

**RUSS TEDRAKE:** It's a gamma in a discounted reward formulation.

**STUDENT:** So there is no eligibility trace?

**RUSS TEDRAKE:** The eligibility trace for the reinforce in a discounted problem is the same as the discount factor. So in my lab now, we're doing a lot of these model based things. We're doing LQR trees. We're doing a lot of things. In fact, the linear controls are working so beautifully in simulation that Rick Corey, one of our guys, started joshing me. He's like, why didn't you just do LQR on Toddler? And he was giving me a hard time for a long time.

Now he's asking about model free methods again because it's really hard to get a good model of very underactuated systems. I mean, the plane that I'll tell you about more on Thursday, our perching plane we've seen quickly, is one actuator. And depending on how you count the elevator, eight degrees of freedom roughly. And sorry, eight state variables. And it's just very, very hard to build a good model for that that's accurate for the long trajectory, the trajectory all the way to the perch such that LQR could just stabilize it. We're trying. But there's something sort of beautiful about these things that just work without building a perfect model. OK?

The big picture is roughly the class you saw. This is actually, I had forgotten about this. This was one of my backup slides from before. But this is the basic learning plot, which is just one average run here. If I reset the learning parameters, how quickly would it minimize the average one step error? And it was pretty fast. And then actually, that's a lot of steps. That's more than I remember. But this takes steps once a second. And so, in a handful of minutes, it does hundreds of steps.

OK. And this is the policy in two dimensions that it learned. So if you think about a theta role and theta role dot, I don't know if you have intuition about this, but the sort of yin and yang of the Toddler was that you wanted to push when you're in this side of the face portrait and push with this foot when you're on this side of the face portrait. I did things like I mirrored, the left ankle was doing the inverse, the mirror of the right ankle. Right?

So everything I could do to try to minimize the size of the function I was learning. And that's actually sort of a beautiful picture of how it needed to push in order to stabilize and skate. Any questions about that?

All right. So that's one success story from model free learning on real robots. It learns in a few minutes. There's other success stories. I'll try to talk about more of them on Thursday. But at this point, I've basically given you all the tools that we talk about in research to make these robots tick. Their state estimation, I didn't talk about. There's Morse's idea that we didn't talk about. But this is, I've given you a pretty big swath of algorithms here. So really I want to now hear from you next week. And I want to give you a few more case studies so you feel that these things actually work in practice. And you can go off and you use them in your research. Yeah, John?

**STUDENT:** If there is a lot of stuff that's been published and a lot of interest [INAUDIBLE] stochasticity, then it would make sense to have a large gamma [INAUDIBLE]. Right? There'd be no reason, it would be a faulty way of trying to interpret that data. Right?

**RUSS TEDRAKE:** Yeah. I mean, I think that so Katie's stuff, the metastability stuff, argued that for most of these walking systems, it doesn't make sense to look very far in the future anyways. Because the dynamics of the system mix with the stochasticity, which I think is the same thing you just said. Yeah. Yeah.

**STUDENT:** The general dimensions of the robot [INAUDIBLE] when you're designing that robot, thinking about this model free learning when you started? [INAUDIBLE] helps it be a little more stable.

**RUSS TEDRAKE:** Good. So I'm glad you asked that. So it's definitely very stable, which was experimentally convenient. Right? Because I didn't have to pick it up as much. But it actually learns fine when it starts off unstable. So the way I tested that is, if the ramp was very steep, then it starts oscillating and falls off sideways. So just to show that it can stabilize and unstabilize. It's like, oh, the same cost function. It's absolutely no different. I showed that it stabilized that. And it just meant I had to pick it up when it fell down a bunch of times. But the same algorithm works for that.

So it's not really the stability that I was counting on. That was just experimentally nice. The big clown feet and everything were because that's how I knew how to tune the passive gait. Right? In the passive walkers we work on these days, you always see point feet. Because I care about rough terrain now. And those clown feet are not good for rough terrain. So we could try to get rid of that.

**STUDENT:** You're saying if you wanted to scale that out, you had mentioned the [INAUDIBLE] robots [INAUDIBLE] would you have the same success [INAUDIBLE]?

**RUSS TEDRAKE:** Got you. I think it's fine. I think that it would look ridiculous that big maybe. And I wouldn't scale the feet quite that big. Right? That would be ridiculous. But I don't think there's any scaling issues there really. It's the inertia of the relative links that matters. And I think you can scale that properly. At some point you're going to just look ridiculous if you don't have knees and you're that big.

So yeah. Energetically, the mechanical cost of transport, if you just look at the power coming out of the batteries-- sorry, actually the work done by the actuators, the actual work done by the actuators. It was comparable to a human, 20 times better than ASIMO. But if you plot the current coming out of the batteries, it was three times worse than ASIMO or something like that. Because it's got these little itty bitty steps and really big computer. And that was, in retrospect, maybe not the best decision. Although I never had to worry about computation. I never had to optimize my algorithms to run on a small embedded chip.

**STUDENT:** Can you talk a little bit about the [INAUDIBLE]?

**RUSS TEDRAKE:** You can actually see it here. So this is the barycentric policy space that were the parameters. Yeah. So it was tiled over 0.5, 0.5 roughly. And you could see the density of the tiling there. Yeah.

And that was trained. So there was no generalization. So the fact that those looked like sort of consistent blobs was just from experience and eligibility traces carrying through. But those are not constrained by the function approximator to be similar more than one block away.

There's literally a barycentric grid there. And then the value estimate was theta equals zero. The different theta dots. It was just the same size tiles. But a line just straight up the middle.

**STUDENT:** So your joystick would just change theta? Or not the theta? But it would just change the position.

**RUSS TEDRAKE:** The joystick was, so the policy was mostly for the side to side angles, which would give me limit cycle stability. And then I could just joystick control the front to back angles. So this thing, we could just lean it forward. It starts walking forward. Even uphill. That's fine. You lean back, it starts walking back. It was really basically this. Yeah. If you want it to turn, you've got to go like this. And it would do its thing. Right? So that was it. It wasn't sort of highly maneuverable. Yeah.

**STUDENT:** It seems like there are some [INAUDIBLE] to step to step, having each step be like--

**RUSS TEDRAKE:** A trial.

**STUDENT:** So a section on your Poincaré map.

**RUSS TEDRAKE:** Yeah.

**STUDENT:** I don't know if that would work for flapping.

**RUSS TEDRAKE:** Absolutely.

**STUDENT:** If [INAUDIBLE] up or down is a similar kind of thing.

**RUSS TEDRAKE:** I think it would. We were thinking about it that way. So you're absolutely right. So it was nice to be able to, it was very important to be able to add noise by sort of making a persistent change in my policy. So this whole function, adding noise meant this whole function would change a little bit.

And then I would stay constant for that whole run. And then change a little bit. If you add noise every DT, for instance, then you have to worry about it filtering out with motors and stuff. This was actually a very convenient discretization in time on the point grade map. Yeah. So I think that was one of the keys to success. John?

**STUDENT:** The actuators you took, were they pushing off the sort of stance foot?

[INTERPOSING VOICES]

**RUSS TEDRAKE:** Or pulling it back up. But yes.

**STUDENT:** So you just actuated the stance foot. That was the actuator [INAUDIBLE].

**RUSS TEDRAKE:** The units were, I guess they were scaled out. I did actually do the kinematics of the link. So it was literally a linear command in-- those are probably meters or something in the-- no. It's way too big.

[INTERPOSING VOICES]

**STUDENT:** Touching down, but touch down at the same angle?

**RUSS TEDRAKE:** No. The swing foot was also being controlled. So it would get a big penalty actually if it was at a weird angle when it touched down. It would hit and it would lose all its energy. But that was free to make that mistake.

**STUDENT:** So you have two actions then? The address to [INAUDIBLE]?

**RUSS TEDRAKE:** No. Well, it's one action. But the policy is being run on two different actuators at the same time. So one of them is over in this side of the state space. And the other one's over in the side of the state space at the same time.

**STUDENT:** OK. So it just used different data. But they're-- OK.

**RUSS TEDRAKE:** Yeah. So just it was learning on both of those sides at the same time. I'm a big fan of simplicity. It's easy to make things that work. I mean, I think it's a good way to get things working. So that's what the test will be as we go forward in how complex we can make these things. But in sort of the simple case, they work really well.

Great. OK. So thanks for putting up with the randomized algorithm. We'll see you on Thursday.