**JOHN W. ROBERTS:** We'll be talking about-- I heard last time I had bad handwriting. And I guess this isn't much improved yet, but I will try to be more deliberate if not more skilled. Stochastic-- all right.

So as you may remember last time, we were talking about different assumptions that we've used in all the techniques we've applied so far. Now, we assume that we have a model of the system, that the system is deterministic-- that's not really any better handwriting, but this is the one that last time we talked about getting rid of anyway, right? Stochastic system, stochastic dynamics. So that's what we tried to remove. And then the state is known.

So sort of already gotten rid of this one in some of our discussions. And today we're going to talk about what you do if you don't a model. And this is something that's actually very important in a lot of interesting systems.

And the systems that we work on the lab, some of them we try to model, but some of them help us to model. So this, dealing without this, is a very useful thing to be able to do. So hopefully, you'll all at the end appreciate the tremendous power of model-free reinforcement learning.

So the basic idea is, again, we have this policy parameterization alpha, which somehow defines our policy. And the problem sets that you recently did, it's open loop, so you just have one alpha for every time step. You also can imagine these are gains on a feedback policy, entries of the K matrix, or PD gains, any way you want to parameterize it.

And you think about you use-- these parameters-- now, this is the most simple interpretation. There's a lot more complicated ways of looking at it, but I'm going to look at the most simple way first. You send this into your system. So you can run your system with these parameters.

Now, this is, again, sort of like what you did in the problem set. You have a fixed initial condition, fixed cost function, you give it a policy, you run it, and you see how it does. And so what you get, you get J. You get the cost of running that policy.

So the question is that, previously we've talked about, OK, if you have a model of the system, there's a lot of things you can do. You can do back prop to get the specific gradient; do something like SNOPT, you can do gradient descent using that. You can do, depending on the dimensionality, you can do value iteration. So there's a lot of options when you have a model.

But if you don't, if you don't know how the system works, if it's really just a black box where I have a policy parameterization, I get a cost, how do we achieve anything in that context? We don't have any sort of information about how these things relate to each other.

Well, the thing is we do have some information in that we can execute this black box. We can test it, right? We can run our policy and see how well it does.

So what would you say is the crudest thing you could do if you had a system like this, a black box? You give it an open loop tape, let's say, you run it, and it tells you the cost. What could we do?

**AUDIENCE:** SNOPT could also-- well, not that we'll [INAUDIBLE] SNOPT. But SNOPT could also-- or you have methods for estimating the gradient.

**JOHN W. ROBERTS:** You can do finite differences, right?

**AUDIENCE:** Yeah, do finite--

**JOHN W. ROBERTS:** So finite differences, exactly. So what you can do is you can say, let's talk about, again, this is a notation. [? So ?] what we're using is simple. A lot of times they parameterize it in a different way. But yeah.

So you have pretty much in this context-- let's say we have a deterministic cost. So we don't have a random system. We'll talk about random systems later. Let's say we have a deterministic cost, which is a function of our alpha. So what are our parameters, our parameter vector?

So what we do is we say, OK, let's say I have a 2D system alpha 1, alpha 2. Now, we don't know what this function is right now. But let's just say it's a simple function like this, convex, where these are the contour lines, and what we want is we want to get to the middle. So this is sort of the local min, and we start here.

Now, what-- how SNOPT could get these gradients, and sort of the simplest thing you can imagine doing, would be, all right, well-- one of the simplest things you can imagine doing is actually another very simple thing. You measure here. So you run the system. You get J at this point.

So you run the system, you get J at this point. You run the system, you get J at this point. But you take these differences, divide by your displacement, and what you get is you get some estimate of the local gradient. And if those distances are small enough and your evaluation sort of nice enough, you can get arbitrarily close to the true gradient there. And this will tell you, OK, you want to move in this direction, right?

Now, the problem with this is that you have to do n plus 1, where n is the number of dimensions, evaluations, to get this. So you sort of have to evaluate at alpha and at alpha plus delta 0, 0, 0, dot, dot, dot, alpha plus 0 delta 0, 0, dot, dot, dot, et cetera. Now, obviously, these things just have to be linearly independent, actually. But you might as well do it this way.

Do these finite differences, you get an estimate of the gradient. You can hand it to SNOPT and SNOPT can try to do fancier things. Or you can do gradient descent, where you get this gradient, you compute it, and then you do an update where I say, OK, now my alpha at n plus 1 equals alpha at n plus some delta alpha.

And you can say, OK, delta alpha equals negative eta and then dJ d alpha. That's a vector. And so this is our learning rate. That says, OK, we have the gradient here. How far are we going to move? Setting that can be an issue.

But you update your alpha like this. And you can just keep doing that over and over again, keep evaluating it over and over again. And eventually, you should move in to the 0. You should get to a local min.

The thing is that doing n plus 1 evaluations every time is expensive. Now, you could say you could cut that down a bit if you were to reuse some evaluations and stuff like that. But the point is that you have to do a lot of evaluations to get this local information.

And if you move very far, you sort of have to discard those and do all those evaluations again. So short of doing a lot of evaluation to get sort of an accurate estimate of the gradient right here. Then you're throwing a lot of it away when you move, and the gradient could change.

And so in that sense, doing all these evaluations maybe is wasteful, because you're getting more-- you're sort of being more careful than you have to. And then you're just going to lose that information once you move somewhere else and have to evaluate it again.

So there's another thing. This one, you could say, is even more crude. This is, at least in evolutionary algorithm [? screen, ?] I think they call it just hill climbing. I mean, all these things are sort of hill climbing or valley descending. But what you can also imagine doing is just having a point here, and now we just randomly perturb that.

So I don't do this. This is deterministic, right? I could randomly perturb and just be like, OK, well, what if I'm here? That's worse, right? The cost is higher. So you just throw it out, don't use it.

Do it again here, that's better. So now we just keep this. And we just do this over and over, discarding bad ones, keeping good ones, until we get back there. But the thing is, is that there you're doing all these evaluations. When they get worse, you're just throwing them out, and you're acting like they give you no information.

But there is information in that. Even if it gets worse, and by how much it gets worse, how much it gets better, there's information in all of that. And you're getting information when you do the evaluation. So I throw it away and just sort of like cast out these things that do worse.

So that's sort of the idea of the stochastic gradient descent, is that we're going to follow this sort of like random kind of idea. Well, instead of doing this deterministic evaluation of the local gradient, we're going to randomize the system, and we're going to get an estimate the gradient stochastically, and then we're going to follow that.

And we're going to get as much information out of that as possible. That's one of the important thing, is generally these systems, this evaluation is all the cost. Pretty much everything is dominated by checking your cost of the [INAUDIBLE] policy. So you want to get as much as you can out of each one. And stochastic gradient descent is sort of a powerful way of doing that when you have no model. That's definitely more efficient than hill climbing.

So now the question is, what is the appropriate process for doing this? How do we randomly sample these guys and actually improve our policy? So I'm going to write down an update. This is a common update. It's the weight perturbation update. It also shows up in an identical form in reinforce, if you see any of those. We'll talk about all those.

But when you can look at the [? performance ?] update-- is my handwriting at all legible?

[INTERPOSING VOICES]

**JOHN W. ROBERTS:** Yeah? OK. So you want to look at this [? performance ?] update. Take my word for now that this makes sense. Well, changing the alpha bits. OK.

So I'm saying, change your alpha. Here we have the same learning rate. So this is like in the deterministic gradient descent. And then here's where you evaluate. And this z is noise.

So when you perturb your policy, this is sort of the vector of how you perturb that alpha vector. So this is sort of-- this is a z, this is a z, this is a z. Those z's are these perturbations to this.

So what you can do is we can say-- a simple and a very common way is to have the vector z is distributed as a multivariate Gaussian, so where each element of z is iid with the same standard deviation, mean 0. And so you sort of draw this z from your-- you draw a sort of sample z, you evaluate how well it does, you evaluate how well you do with your sort of nominal policy right now, calculate this difference, and then you move in the direction of z.

So I'll try to draw this in 1D and then 2D so it makes sense. So here in 1D, you can say this is our 1 alpha. If this is J, so here's our cost function. So we'll be here.

Now, our z in this case is just a scalar. But so our z is going to be mean 0. And it's going to have a Gaussian distribution. But when you sample from this, you evaluate-- I should actually probably keep that update up at the same time. So you sample, you get this change.

So this is sort of my J alpha. This right here is my J alpha plus z. And imagine this change. That's going to say, OK, the cost went up. It went up by some amount. That's the difference.

I'm going to move in the direction of z. So z is just a scalar. Here it's just going to be sort of the sign and the magnitude of it. And then I'm going to move sort of opposite this. So I perturb z. z went in this direction. It got bigger.

That change, then, is a positive number. So we're going to move down by amount sort of eta that change, right? And so if it gets a lot worse, we move down farther. If it gets a bit worse, we move down a bit. Does that makes sense?

And so when you're measuring here, you're going to get small change for the same z. When you, again, you draw your Gaussian around that, if you get a small change, you're just going to move a bit. When I'm here, where it's really steep, I'll get the same perturbation, I'm going to get bigger change. I'm going to move even farther. And I'll update here.

And so if I do this a bunch of times, you can imagine I descend into local min. Does that make sense? And this is every time you're drawing the stochastically. So you're not doing this [INAUDIBLE] term thing. Every time you do it, you could be updating, you could try worse, you could try better.

But stochastically, you can sort of intuitively see why it's going to sort of descend. Does that make sense?

**AUDIENCE:** This is heavily depending on the fact that the function is sort of [INAUDIBLE] direction?

**JOHN W. ROBERTS:** It's sort of what?

**AUDIENCE:** It's like the function that you're looking at, if you're looking-- if you increase, like in this case, like alpha in one direction versus other one, the changes are sort of similar in both ways.

**JOHN W. ROBERTS:** No. I mean, that can affect the performance of the algorithm. But yeah, I can draw that. These are sort of common pathological cases. Let's look at in 2D. So this is what you're saying, right?

Now, the ideal one would be-- again, we can draw a contour map again. Now, you'd be-- this is about the same, right?

You're saying this is about sort of isotropic or whatever. You're here, you perturb yourself randomly so your Gaussian is going to put you anywhere here. And you measure somewhere. You get better, so you're going to move in that direction, depending on what eta is, and I'll get an update.

You're saying, well, what happens if we're actually in trouble, and we have something that looks like this, right? Saying that's a problem?

Well, that can hurt the convergence of it. It can be slower. But it still works. Because you can see-- like, let's say I'm here. Now, it's really steep here, and it's really shallow here. So what's going to happen is when I perturb it, I'm still going to-- my perturbation in this direction is going have an effect-- maybe it's relatively shallow-- but then in this direction it's going to be very sensitive.

And so when I move it more in this direction and I move very far, and I'm going to go down here first-- I'm going to descend the steep part-- and then slowly converge in on the shallow part. That's sort of called the, I think, the banana problem, where you sort of have this massive bowl, and you go really quick right down here, and then really slowly.

And so the thing is that if it's all very shallow, that's not a problem. You can make your learning rate bigger, you can make your sample further out, and then it sort of just doesn't matter, right? But this asymmetry in these things is an issue. Now, there are some ways of dealing with that if you have an idea of how asymmetric it is. We can talk about this later.

But it'll still descend. And actually, you can show, and I'm about to show, that this update actually follows an expectation it moves in the direction of the true gradient. So, I mean, randomly it can bounce all around. But in expectation, it will move in the right direction.

And if you're having deterministic evaluations-- well, we're going to do a linear analysis at first. But you actually can show that it'll always move within 90 degrees of the true gradient if you have deterministic valuations. So you'll never actually get worse. You can move parallel and not improve, but you'll never move sort of the wrong sign of [? those two ?] parameters.

All right. So then yeah. Let's look at why that is in some detail then. So again, our delta alpha, same up there. So I just won't waste time rewriting it.

And let's do-- let's look at it in a first-order Taylor expansion of our cost function. So look at it locally where you look at sort of like the linear term. So our J-- and we linearize around alpha. So our J of alpha plus z, well, that is approximately equal to-- for small z-- alpha plus dJ d alpha transpose z. So that's the first-order Taylor expansion.

Now, if we examine this then, we plug this in for J alpha plus z in that update, we're going to cancel out of this term, our J alpha term, and we're going to get that delta alpha approximately negative eta dJ d alpha z z.

So now, what does this look like? This is sort of like a dot product between the gradient with respect to alpha and our noise vector. All right?

So and this is going to be about equal to negative eta. This thing we can then write [INAUDIBLE] i is 1 to N of dJ d alpha i zi times vector z.

All right. So here then, if we multiply that out, so we're going to get this vector and eta, because you're multiplying that coefficient times each term individually. You're going to get the vector. And then the same thing. This one's going to be some dJ d alpha zi zN.

Now, if we take expectation of this, we get another distribution. We know that each zi is iid. Do you know iid? That they're all-- they're all distributed the exact same distribution, all the sort of mean 0, Gaussian, standard deviation sigma. And they're all independent.

So if we do that, we can then take the expectation of delta alpha. We can pull that eta out front, because expectation is linear. And what you'll get is you'll get the [INAUDIBLE] again, dJ d alpha-- i is not a random variable. So pull that out. dJ d alpha i, again the sum z-- sorry. Expectation of zi then z1.

Now, this sum goes through all the i's. But the first one only has that z1, right? Now, zi, z1, they're independent, mean 0. So you can sort of split these up, and you're going to get that they're 0 for every term, except for the term where i equals 1, and the second one where equals 2, et cetera, right? All the other terms are going to go to 0.

So it's easy, then. To get the expectations, you go through the sum, and you're going to see that you only have the one where you have expectation of z1 squared, expectation of z2 squared. Now, the expectation-- again, maybe you remember variance equals expected value of x squared minus expected value of x squared, right? Now, we're mean 0, so this is 0. Our variance is sigma squared, So our expected value of x squared is sigma squared.

So that means that each one of these expectations is going to be sigma squared. So you're going to end up where you have negative eta-- now, they all the same sigma, so we can pull that out-- sigma squared. And you're going to have the vector of this dJ d alpha 1, dJ d alpha 2, et cetera. So you're going to get dJ d alpha.

So yeah, so the expectation, this update, when we look at it in this sort of linear sense, is eta sigma squared-- so just these are scalars. They just change the magnitude of it. But it's in the direction of the gradient. And eta is sort of our parameter. We can control it.

That makes sense?

**AUDIENCE:** Is that sigma squared?

**JOHN W. ROBERTS:** Yes, yes. Sorry. Yeah, sorry. Yeah. So the noise you use pops out here.

Comment-- I actually oftentimes in one of the other-- when we look at this algorithm in a different way, they write the update where it's eta over sigma squared, your noise. And then that cancels out that sigma squared, and you purely just get eta dJ d alpha. So you can put that in, too, if you wanted to really just be eta times your true gradient. But the important thing is that you'll move an expectation in the true direction.

So a couple of interesting properties to this. Here, you see we still have to do-- we still have to do two evaluations to give rid of the update, right? If we want to cancel out that J alpha term, we're going to have to evaluate it twice. Now, it doesn't matter for three-dimensional, and we only have to evaluate it twice. But we still have to evaluate it two times.

And the question is, well, what happens if you don't evaluate it at J alpha? What happens if you only evaluate it once? Well, that's a very common thing to do, actually, and doesn't actually affect your expectation at all.

Lots of times, instead of this sort of like your perfect baseline where you evaluate it, people sometimes average the last several evaluations to get that baseline-- oh, sorry. I don't think I defined baseline. This right here, whatever it is, is your baseline.

Now, there doesn't have to be J of alpha. It can be an exponentially decaying average of your last several evaluations. That's going be approximately J of alpha. And it won't be perfect, but the point is that it's not going to affect it, and we're going to see that. Maybe you'd expect that you need to get rid of that term for you're still moving in the direction [? of ?] [? your ?] gradient, because you can imagine if you don't have that, if you don't know that term, you could evaluate-- if it's always positive, you'll-- I'll draw a diagram, make this clear.

If you don't have that, and you're here, if I-- let's say I just make that 0. I'm going to evaluate here, that's going to be a positive number. So I'm moving in the opposite direction. If I evaluate here, it's going to be positive number. So you're going to move in the opposite direction. So maybe you think like, oh, without that baseline we could be in bad shape.

But actually, you'll move more in this direction when you do that sample than you move in this direction when you do the other sample. And so that scale here, the fact that you move proportional to how big the change is in your cost, it means that in expectation, you'll still move in the direction of the true gradient.

Now, in practice you won't do as well. It makes sense that you won't do as well. Really, when you think about it, that's going to be bouncing all around crazily. But it'll still move in the direction the gradient.

And you don't just have to take my word for that. If you look at this update again, now we can do linear expansion again, and you'll get this dJ d alpha z plus, say, some scalar-- this is uncorrelated with the noise. That's an important thing, though. It's uncorrelated with the noise z.

Now, use expectation again. Expectation is linear. So we have expectation of this term. That's the same as it was before. That's the gradient. And then we have expectation of negative eta dz.

Now, E is uncorrelated with the noise. These are both scalar, so you can actually pull them out. Expectation of z, it's mean 0. So this won't affect it at all. So really, your expected update will not depend it all on what you use here.

So you could put a constant there. You could put in the exact one. You could put in some decaying average, anything you want. It will still move, in expectation, in the right direction.

But in practice, it can a huge difference. I don't know if anyone's implemented these things on-- but a good baseline can be the difference between success and getting completely stuck and not moving anywhere. So if you do small updates, you should still be OK. But performance depends a lot on getting a baseline. Or it can depend a lot. Sometimes it doesn't matter.

Right. So the-- yeah. So again, a common thing to use here is that you're evaluating, you're updating. Let's say every time I do one evaluation, I update. If I took my last 10 of them, I averaged them with decaying sort of weight so that the most recent one is the most heavily weighted, then I'm sure you'll get an approximation of how much should it be around here. And then I update based on that. And that way you don't have to evaluate it twice every time. And so that way, you can actually get sort of improved performance. And it's still going to work.

And another cool thing, this is sort of when we go back to our assumptions about deterministic. It doesn't have to be deterministic, either. Let's say in the same way we put in this, instead let's say we put in noise like, again, like a scalar noise to evaluation w. Oh, I just got [? color. ?]

Now, that's going to show up in here again. Now, it's not a-- now it's a random variable, so it has an expectation. But if they're uncorrelated, we can split them up. We can-- that'll be equal to negative eta expectation w expectation z.

Now, we know that z is mean 0 again. That's 0. So it's not going to affect either. We're still going to get this term. And so you can add sort of additive random noise, and you'll still move that through expected direction, the gradient.

So that's sort of cool. This is quite robust. You can have these errors in this baseline. You can have noisy evaluations. You can have all sorts of these things. And still, expectation will move in the right direction. So that's nice.

We're going to see that that has a lot of practical benefits. Is everybody with me here? I don't know if I went through this quickly or if-- everyone's sort of being quiet. They look sort of--

**AUDIENCE:** w is baseline there?

**JOHN W. ROBERTS:** No, no, sorry. This w I change it to noise. Sorry, this is a noise. Maybe you'd prefer it to be called like xi or something like that. But this is just added noise. So you could say that z is drawn from-- it doesn't really matter the distribution as long as it's uncorrelated. We could say it's drawn from some other Gaussian.

And so it's expectation-- I mean, expectation of this really can be 0, too. Because if it's not non-zero-- it's not mean 0 noise, then you might as well just put that in your cost function and make it mean 0 again, right? Yes?

**AUDIENCE:** So the idea is to add this into the term J alpha? Or replace the term J alpha with different baseline?

**JOHN W. ROBERTS:** Replace it, right.

**AUDIENCE:** OK. And then so what cancels-- so when we talk about a Taylor expansion? What cancels-- what--

**JOHN W. ROBERTS:** Nothing. Nothing cancels it. You see, that's the thing. Yeah, so I put an E here-- maybe I'm reusing too many things.

**AUDIENCE:** Oh, is it J alpha is also uncorrelated to z?

**JOHN W. ROBERTS:** Well, J alpha, J alpha is just a scalar, right? I mean, it is some number. So it is--

**AUDIENCE:** z is your mean, so.

**JOHN W. ROBERTS:** Yeah, so z is your mean. So whether-- we could put in J alpha. We could put an estimate of J alpha that has some error. And then our J alpha minus this is going to be some number-- doesn't matter. If we just put in nothing at all, then our error is sort of that J alpha term. That J alpha term is just, again, some number that's uncorrelated, gets rid of it.

Does that make sense? Everyone looks sort of just--

**AUDIENCE:** So it's actually, putting another constant in that equation for the update makes you move more in some random z-direction. But on average, you're still going down the gradient the same way.

**JOHN W. ROBERTS:** Yeah. I mean, you can move more. Yeah. I mean, if you put some-- if you put some giant constant every time you update, maybe you'll bounce around farther. But on average, you'll still move in the right direction. Because you'll move farther in the right direction than you move in the wrong direction. So they sort of cancel out.

So everybody is on board here? OK. I just really want you to--

**AUDIENCE:** Why wouldn't you include the actual J alpha?

**JOHN W. ROBERTS:** Well, because if you get it by evaluating the function, if you run a policy, it can be expensive to get that J alpha, right? Because for example, I use this in some work I did where we had this flapping thing. I'll show you videos of it. Maybe I'll start setting that up right now.

But so we have this flapping system. And we get-- we sort of have souped it up now so it's a bit quicker. But it used to be every time I wanted to evaluate the function, I had to sit there for 4 minutes and have this sort of plate flap in this water and measure how quickly it was going, all these things.

And so to evaluate that function once, it took me 4 minutes. And so avoiding evaluations is important. And so if you can just take your several previous evaluations, average them together-- now, it's not going to be a perfect assignment, but maybe it's an OK estimate, and then you don't have to spend any more time. And so in that sense, it's sort of cheaper.

Please ask as many questions as possible, because this is--

**AUDIENCE:** But at some point you have to measure every time, right?

**JOHN W. ROBERTS:** You have to. Yeah, you have to measure every time when you want to do an update. But the thing is that-- here. Let's say i, a tiny one-- but the question is, if I have some estimate of that-- let's say my current sort of alpha is here. Now, I need to randomly sample something, so I have to do that evaluation.

Now, the question is, do I have to evaluate it here, too? Because this is my J alpha. Do I evaluate that? Now, I could estimate this, because have a bunch of other evaluations from however I got here, right? So I've already evaluated. If I average those together, I'll get a pretty good idea of what this is.

If I wanted to get it exactly, I'd have to run my system here, and then run it again here. And so every update would require two evaluations as opposed to just one. Now, sometimes it still makes sense to do that evaluation, though. Depending on how your system is, if it's really noisy, if you have to do really big updates, it makes sense.

**AUDIENCE:** [INAUDIBLE] using this delta alpha would you calculate [INAUDIBLE]?

**JOHN W. ROBERTS:** Pardon [INAUDIBLE]?

**AUDIENCE:** Yes.

**JOHN W. ROBERTS:** I'm sorry, I didn't hear what you said.

**AUDIENCE:** This new alpha that we have, that we have the [INAUDIBLE] before--

**JOHN W. ROBERTS:** This one? Yeah.

**AUDIENCE:** You calculate it by having a previous alpha, and then we did this thing, and--

**JOHN W. ROBERTS:** And I moved in that direction, right.

**AUDIENCE:** Right. But you're saying that you don't want to calculate the value for this new alpha. Instead we use like, for example, 10 past history of J of alpha, and use that as your estimate.

**JOHN W. ROBERTS:** Yeah. You're saying that doesn't make sense to you?

**AUDIENCE:** It does make sense. In some cases I can think [INAUDIBLE] actually [INAUDIBLE] if the change-- a small change in alpha would have a huge effect on the end value [INAUDIBLE] from J-- like, if you have a very discrete-- like, [INAUDIBLE] condition pass over [INAUDIBLE].

**JOHN W. ROBERTS:** If you move very violently, yeah. So I mean, that's a good example in practice. I mean, there's things that we have in the theory like this expectation stuff. And there's things that I've applied to several systems.

And in practice, when you have like sort of really bad policies, and you need to move really far in state space-- let's say that right now you're trying to swing up a cart-pole, and you're not going anywhere near the top. And your reward function doesn't have very smooth gradients, and so you can't just sort of swing up a bit by bit by bit.

Well, a good thing is, is to put in place possibly very big noise, a very big eta, and then do these two evaluations. Because if you-- it's going to change so much every time you do it. Like for example, if you jump and suddenly you're doing a lot better, then your previous average is not going to be representative. And then you can actually bounce around. You can bounce around so violently in this big space of policies that you never improve, right?

I don't-- maybe I should draw a diagram to make this more clear, what I'm saying. But the key thing is, is that, yeah, if you're moving these really big jumps, and your cost is changing a lot every time, and you still want to sort of move in the right direction, doing two evaluations can make sense. Because if you're stuck to where you don't have good gradients in your cost function, a bunch of little updates which slowly would climb aren't going to give you anything, because maybe they're not even differentiable. Maybe you have some sort of discrete way of measuring a reward, like how many time steps you spend in some goal region, or something, and you don't have any time steps there, there's no gradient at all right now.

And so you need to be violent enough in sort of your policy changes that you eventually get it to where you're into that goal region. And once you get in that goal region, now you have some gradients and you're in good shape.

So that's actually another thing that I was going to talk about. But designing your cost function is extremely important. There are cost functions that can be extremely poor and doing this can work really poorly on. And there's cost functions that can make it a lot easier.

So if you have a cost function which is relatively smooth, if it's-- ideally it doesn't have this sort of banana problem. If it's relatively same in all the different parameters, it can work a lot better. And you can sort of formulate the same task lots of time, since lots of times your cost function isn't what you really want to optimize. It's just of a proxy for trying to get something done.

That's what Russ talked about he didn't care about optimality. It's like, here's a cost function that gives us a means of solving how to do this. And so there's sort of a whole bunch of cost functions you can imagine coming up that try to encapsulate that task.

Now, if you come up with-- for the perch one, for example, this plane perching, which is a difficult problem, and a problem where the models are very bad-- I mean, the aerodynamic models of this plane flying like that are extremely poor. And we have-- we actually have some decent ones. We spent a lot of work trying to get decent ones. But sort of the high-fidelity kind of region, where you really want to just get at the end, it's hard to model that.

So the thing is that, what if you had a cost function, like what we really care about is hitting that perch. So let's say that we give you a 1 if you hit the perch and a 0 everywhere else. Now, that means until we hit the perch, we're getting no information. We could be getting really close, we could be really far away. It's not going to tell us anything.

Now, a lot of actually reinforcement learning has these sort of rewards, like these sort of delayed rewards where you get it here, and then you have to sort of propagate that back. When you're trying to accomplish a task like that, that doesn't necessarily work that well.

If you measure something like distance from the perch of distance from your desired state, if you get a little bit closer to your desired state, you sort of get a little bit better. And then you can measure the gradient. And so that will make a big difference, right? And so if you had something where you have region of state where you have a good gradient in your cost function, and you're out here, and not getting a gradient, the little perturbations you're going to have to random walk sort of have made you no update at all, because you may get no change.

But if you do really big ones, maybe you'll bounce into where you get this region where you're getting some reward. And in that case, these updates are so big that averaging doesn't make sense, a baseline still gives you a big advantage, and maybe two evaluations is worth it.

In some of the flapping stuff I did, I did two evaluations, because when I was moving very violently, because averaging didn't work that well. And getting a good baseline was worth the extra time. But when we ended up getting it working, we put it online, and we actually-- we update it every time we flapped. So it was just 1 second, flap, update, flap, update.

And that way, we pretty much were able to sort of cut our time in half, because our policies were very similar, our average was a pretty good estimate. It's so noisy that one evaluation, anyway, isn't necessarily that great of an estimate of your local value function. And so yeah. We just did an average baseline.

And that's sort of half the running time, right? And so it can be a big one. And so there's a lot of details when you implement it about the right way to sort of put this together, and depending what your cost function is, and how good of an initial policy, what your initial condition and your policy is. But yeah, there's a lot of factors like that.

All right. So now we can do some of-- sorry. I can do example of this. So I keep on talking about this flapping system. That's what I worked on for my master's thesis. And so that's sort of what my brain always goes back to, particularly since we used all these methods.

But all right. So now I wonder if I can do Russ' thing where he makes the font really big. That's also-- the thing I'm about to run, it's this relatively simple lumped parameter simulation of the flapping system.

This is a lumped parameter model of-- let me show you, it's pretty cool-- of this system which I guy in NYU named Jun [? Zhang ?] built this robot that effectively models flapping flight. It's a very simple model. I'll show it to you in a second. But it has a lot of the same dynamics and a lot of the same issues as sort of a bird.

So the system, it's a sort of a rigid plate. Well, the one you see here, we attached a rubber tail to it. But the one-- most of these results are on actually a rigid plate, where it heaves up and down, and what we can do is control the motion it follows.

I hope that the camera can see it.

**AUDIENCE:** [INAUDIBLE] moonlight [INAUDIBLE].

**JOHN W. ROBERTS:** Mood?

**AUDIENCE:** Moonlight.

**JOHN W. ROBERTS:** Oh, moonlight. I was like, mood lighting? OK. Make my lecture more enjoyable. All right. So this is the system.

You can see we drive it up and down. That big cylindrical disk right there is the load cell. So that measures the force we're applying. And then what we do is we control this vertical motion.

How we control it is-- that's an important thing. I talked about how the cost function matters a lot. Well, another thing that matters a lot is the parameterization of your policy. Now, in the last few problems we had open-loop policies, which are pretty simple. You have like 251 parameters or something like that, right?

Now, when you're doing gradient descent using back prop or SNOPT, you have the exact gradient. It's cheap to compute the exact gradient, so you can sort of follow this pretty nicely. But When you do stochastic gradient descent, the probability of being perpendicular sort of to your gradient, or nearly perpendicular to the gradient, increases the number of parameters goes up.

So you can think, if you're on-- if you're doing a 1D thing, you're always going to move pretty much-- it doesn't matter if you move in the right direction or the wrong direction. That's one of the benefits of this instead of that hill climbing. But you're always trying to get moving in the right direction to get this measurement. Does that make sense?

If you think in 2D, you have the circle. You're going to be moving around. You're going to be along-- close to the direction of your gradient pretty often. A sphere, it's a lot easier to be pretty far away. I mean, sort of a lot more of the samples you do are going to be relatively perpendicular to your true gradient. And as your dimensionality gets very high, a lot of your samples are relatively perpendicular.

And the thing is that whether you go in the right direction or wrong direction doesn't matter. You'll get the same information either way. Going perpendicular to the gradient gives you no information. Because you'll get no change, and there's no update.

So it's still-- the [? cross ?] dimensionality is alive and well. And very high-dimensional policies can be slower to learn. And so those 251 dimensional policies you use may not be the best representation, because they sort of-- I mean, you probably don't need that many parameters to represent what you want to do.

So for this, what we had-- and this made a big difference, we tried different things, this one worked really nicely-- was a spline. So we said, all right, if you have time, I'm going to set the final time here. Now that's a parameter, too. Then this is the z height. It's in millimeters or whatever you want.

And I'm going to say, OK, I'm going to force it to be at the beginning, in the middle, and at the end-- wow, that's nowhere near the middle, is it? I shouldn't be a carpenter in the 1200s. So what do we do then? We then have five parameters-- now, we've done several versions, but simple one right here-- five parameters that define a spline.

So this is going to be smooth. You can enforce to be a periodic spline, which means that the knot at the end, the connection here, is continuously differentiable as well. And then we force that this parameter-- so this number p1, this one is going to be the opposite of it. So it's a negative p1. And that's true for all these.

So this way, we have this relatively rich policy class that has sort of the right kind of properties. But we do it with only five parameters. So you can imagine, if we want it to be asymmetric top and bottom, that would double our parameters. And we probably wouldn't want to tie this guy to 0, so we'd even add one more.

And when we have the amplitude, you can either fix it or make it free. I can add another parameter. So you can see that as you add this richness, you're going to add all these different parameters. But getting-- using a spline rather than-- this is the height right now, this is the height right then-- it's a huge advantage. Because what's the chance that you're going to want it to move very violently on a sort of like 1 dt time scale?

And if you try to do that, you could actually damage your system. Some of the policies that I-- when I was working on this parameterization, I had the load cell break off and fall into the tank once. Luckily it broke off the wires and lost its electric connection before it fell in there, but yeah.

So if you come up with a parameterization that appropriately captures the kind of behaviors you expect to see, it can be a lot faster to learn. Now, sort of the warning, then, is that you're only going to be optimal-- the only thing, you're going get to a local minimum in this sort of parameterization space.

So if you parameterize-- if I were to parameterize this by saying, OK, well I'm only going to let it be some-- let's say I was going to do like a Fourier series kind of thing and say, OK, it's add this, this, and this-- now, that's not very rich. It's only three parameters. That's good.

But I'm going to do all sorts of things that are probably extremely sub-optimal. Now, it's still going to find the best kind of behavior, or the locally best kind of behavior can using this kind of policy. But it could be quite bad. So the actual optimum could be very different.

So your policy class, you'd like it to include the optimum. And so that sort of is-- it depends on what the question is. You sort of have to just have a feel for what is a good policy class. How do I get [? my ?] dimension as low as possible, while still having the richness to represent a wide variety of viable policies? So when you're trying to implement these things, that can make a big difference.

So yeah. So we set up that. And we could control the shape of that curve. And so that is the policy parameterization we chose. So going back to this code here.

Now, I think I can just run this here. This is going to be doing that bit we talked about on-- again, a simple lumped parameter model of that flapping system.

So here's our curve. It's this, you see this-- well, this is the forward motion of the thing as it's flapping. This is the vertical motion. So this is sort of the waveform it's following. This is where it is in x position. You can see it sort of goes fast, bounces around-- sorry, this is the speed, not the position. So you can see it accelerates from 0, and then as it's pumping, it sort of oscillates a bit.

In practice, there's more inertia and everything, so you don't see these high-frequency oscillations. But this is just a relatively simple, explicit model. This is the shape we follow. So we're following that curve. And we have a little bit of noise to it.

And let me-- so now we're going to perturb it, measure again. Try to measure again, and boom, here we are. We got a little bit better. This is our reward, and then we did another sample, and that's our reward.

Let's do it again, better. You see we improve quite nicely. And also, notice, relatively monotonically. Now, you might be surprised by that. Because even though we're moving-- we have this sort of guarantee we'll move within 90 degrees of the gradient.

That's what I was talking about sort of with, you'll always be within 90 degrees if it's deterministic. And this is deterministic. But it also sort of is this linear kind of interpretation, right?

So as you run it, you'd imagine that you could perturb yourself far enough that you got worse. Now, the reason that's not happening is because I'm perturbing myself very small amounts, and I'm updating very small amounts. So all this sort of linear analysis is appropriate.

And actually, you can see what I talked about that, that you always get pretty close to the true gradient is there. Sometimes it moves up a lot, sometimes it's steep, sometimes it moves up shallowly, but it does a pretty good job. Now, we can change that and try to sabotage our little code here.

Or sometimes you're OK, actually. That's the thing, is that in practice lots of times it's OK if it gets worse sometimes, because allowing it to get worse, being violent enough to get worse, it'll reach the optimum a lot faster. So here, this is our eta parameter. Let's make it bigger a factor of-- let's make it 20.5.

I don't want to risk-- [INAUDIBLE] not get worse.

**AUDIENCE:** Is that the noise or--

**JOHN W. ROBERTS:** Pardon? No, that is the update. So the noise is the same. This noise is still local. But now we're jumping really far. And so you can imagine, we're measuring the gradient. We're moving really far. And now where we've moved to, that gradient may be a poor measurement of sort of the update over that long of a scale.

So let's do this again. This is always fun. Oh, there you go, already. That's better. See now-- but you see, that's a huge increase then. That's what I'm talking about, is that there's sort of a sweet spot.

And you don't necessarily want monotonic increasing. Like, there's limitations on how violent you want it to be in practice, because on a robot, a very violent policy could break your load cell of and have it almost cost you $400. So you don't do something crazy.

But there's also the willingness that-- oh, that's ugly. But you see, I mean, if you bounce pretty far, you can also get huge improvements. And so there's sort of this-- monotonicity in your increasing reward is not necessarily the best way to learn, I suppose. That's from the trenches. I learned that the hard way through many, many hours sitting in front of a machine.

So then the other thing that we can do is this eta. Let's decrease eta. And now let's make our sigma really big. Now, this is going to be really crazy stuff probably.

But you see, now we're going to measure so far. And we're going to get this sort of-- we're going to try to measure the gradient, but it's going to be just way off, because it's moving so far that the local structure is completely ignored. Yeah.

I probably don't have to be nearly as dramatic as this to make my point. But, you know, it's just completely falling apart. Yeah. That's doing as badly as it can, I guess. I think it's like almost [? no net ?] motion, so. Yeah.

So the sweet spot, then, is somewhere in between, where maybe you want an eta of, say, 3, and sigma, I don't know, 0.1. Oh, that's probably still too violent. Yeah, definitely.

But I think that is-- that's the sort of game you have to play. And how big all these things are depend on a number of factors specific to your system. Like, if your system-- if the change is very small in magnitude, if your cost function is such that it's changed between 10 to the negative fifth and 10 to the negative fifth plus 1 times 10 to the negative sixth, that's changing by very small amounts, right? You could need a very large eta just to make up for the fact that your change is so small.

So a big eta-- like, there's no absolute perception on what is a big eta. It's not like 10,000 is a huge eta. 10,000 could be very small eta, depending on what your rewards are. Same thing with sigma. It depends on how big your parameters are. Because, I mean, my parameters here are of order one, which is sort of convenient. Yeah. So there.

Yeah. So here we're learning pretty quickly. And so all those sort of things, that's sort of a disadvantage of this technique, is that there's a lot of tuning to sort solve these things, is that-- where SNOPT you don't have to set-- SNOPT you don't have to set a learning rate, here you have to set a learning rate. You have to set your sigma.

And when you have really sort of hard problems, there's even more things you have to do. Like, your policy parameterization could affect a lot of things. There's a lot of issues. But sometimes, that's-- sometimes it's the only sort of route you have.

Like, the best this can ever do is gradient descent. It's never going to do better than gradient descent. And so there's of a lot of fancy packages out there. When you have better models and stuff like that, you can do better than gradient descent.

But while even though you're only going to be able to achieve gradient descent, you can achieve it despite the fact that you know nothing about your system, your system is stochastic, and it's noisy, like that. And so in those cases, it can be a big win.

**AUDIENCE:** So when you were doing this in real life, instead of in space each time, you were sitting for 4 minutes in front of a flapping--

**JOHN W. ROBERTS:** I automated pretty much everything, yeah. So I was-- but yeah. I mean, this is a little simulation.

**AUDIENCE:** Every interval was like actually it running and--

**JOHN W. ROBERTS:** Oh, yeah. When I pressed Space, it actually does two-- because this is using a true baseline. I didn't put in the average baseline. So this is running it twice every time I press Space. But yeah. You can imagine every time I'm doing Space, it does this one update and gives me that new point.

What I was doing is I sat there, and it would run, and I'd babysit to make sure it wasn't broken. And it would throw up the curve as it was running so I could make sure that the encoders weren't off, just sort of sitting there keeping track of all these things. I was like a nuclear safety technician. I just eat some doughnuts and go to Moe's, and I would have been a good sitcom character.

But yeah. So I mean, pretty much just babysitting it. But yeah, every time you did it, every time you got a new update-- like, every one of these points cost me 6 minutes or something, because it was like a 3-minute run for-- basically a 3-minute run for an update. Because I wasn't using averaged baseline then either. I was trying to be more violent.

But yeah. And so that's the thing, is that that is the perfect encapsulation of why you want to use this information as carefully as possible. It's because it's very expensive to get a point. Like, here it cost nothing. If I were to turn off the pause, like, this thing would climb up like that.

If you're running on a robot, like we want to use this on the glider, every time you watch that glider, you have to set up the glider, fire it off, take all this data, and reset it by hand, and launch it again. So getting a data point there is going be extremely expensive.

And so we've actually done some work on the right ways to sample. You can imagine trying to come up with the right ways to have a policy. But sampling intelligently can save you a lot of time. We sort of look at the signal-to-noise ratio of these updates.

I don't know if anyone-- some people here probably at least heard about that stuff since they're in my group. But probably talk about that maybe tomorrow. But there's these things you can do that can improve the quality of your performance a lot.

And actually, I test on this exact system. I got put on the system, and I ran it with the sort of results we had that's just, this is a better way to sample, and then just with a naive Gaussian kind of sampling, and you learn faster. And in the context of me sitting there and spending my days in New York City huddled in front of a computer, that's a big win. So anyway.

**AUDIENCE:** So when you say change the sampling, you can just change the variance like you would do to a non-Gaussian distribution?

**JOHN W. ROBERTS:** Right, yeah. So that-- yeah. In fact, we used a very different kind of description overall. You can still-- the linear analysis will still work. But it's just a local-- but yeah, there's work where they change--

We also have something where you change the variance [? to ?] the Gaussian, but your different directions have different variances. And so if you sort of need an estimate of the gradient, then-- but you just estimate the gradient to bias your sampling more in the directions where you think the gradient is, so that more of your sampling is along the directions you think are most interesting.

And so that can be a win when you have a lot of parameters that aren't well correlated. Like if you imagine if you had a feedback policy that was dependent on-- a parameter is active in a certain state-- like, if I was at negative 2 to negative 5, I do this, and let's say I never get there, then that parameter has nothing to do with how well I perform.

And so if you know that, you can sort of-- there's something called an eligibility you can track. And you cannot update that parameter. There's no reason to sort of be fooling around with that parameter when it's not affecting your output. And if you know that, you can do things like that.

And we sort of have a way, a more careful way of, shaping all these-- of shaping this Gaussian to learn faster. And it can. And also, just completely very different kind of sampling. Like, it's-- well, maybe I'll try to talk about it. Because I think it's pretty interesting stuff. The math is a little bit nasty, but I'll skip the really ugly steps.

And actually, the one with the different distribution isn't even that nasty. But yeah. I mean, we ran it here and it [INAUDIBLE] improvement. Yeah, so. Did I answer your question? Yeah. It's not just changing the variances. It's more complicated than that. Although changing the variances can be a big win.

For example, if you knew you had this anisotropy, and if you were to have different etas in different-- if you were to scale everything in your sigma, you could effectively make it squashed in, right? I mean, just a rescaling of this anisotropic bowl will make it right. So if you can evaluate that, you can fix it. But you sort have to know that that's going on.

That's about the times you have adaptive learning rates and stuff. Gradient descent, like if you keep moving the same direction, you have a bigger learning rate. You can have different learning rates, you have different parameters.

This one, as you get close to a local min, you'll decrease your learning rate and your noise, because you want to sort of bounce around. You don't want to be jumping all across this min. So--

**AUDIENCE:** [INAUDIBLE] talked about a basically policy gradient when we were [INAUDIBLE].

**JOHN W. ROBERTS:** Yeah, no. Yeah. I mean, there is-- it's definitely exactly that. It's just stochastic gradient. But yeah, it's all policy gradient ideas. Because we don't-- I mean, these things don't have a critic, right?

But you can combine this with some policy evaluation techniques. And you can turn them into actor-critic algorithms. A very simple-- do people know about actor-critic algorithms? That's going to be a subject I think Russ talks about at the end.

But the thing is that right now-- well, I'll motivate in a completely different way. We talked about how this baseline can affect your performance a lot, right? Now, a good baseline can make you do a lot better. Now, the thing is that, what happens if-- here we start with the same initial condition every time.

But let's say that I actually could be in one of two initial conditions. I can measure this, and then I run it. And the system behaves very differently, or the costs are very different depending on my initial condition. But I want sort of the same policy to cover both of these.

So the thing is, if I just did this and I had one baseline for both of them, and I could randomly be putting these in [? different initial ?] conditions or whatever-- or I mean, I could-- there's probably a more sensible way of saying this, but I don't want to confuse the issue. So if you could have different initial conditions, you can make your baseline a function of your initial condition. Does that makes sense?

Instead of just having B, instead of evaluating it twice, I could have my B of x. And if my x is here, I'm going to say, OK, my cost should be like this. And if my x is here, then it's like, oh, my cost should be like this. And when I evaluate my cost, when I perturb my policy, I have a better idea of how well I'm doing. Does that makes sense?

It probably doesn't, so. All right. So let's say-- now, this is phase space now. Now let's say that I can start in either of these. And let's say that I'm trying to get to-- let's draw this here. I'm trying to get to 0. That's my goal.

And I can measure this. But then one of them, I'm going to go [WHOOSH] like that. And the other one I'm going to have to go, I don't know, through whatever torque, [? limited ?] reasons like that or something. So this one always costs more than this one, all right?

It doesn't matter how good my policy is. Like, you can imagine just have a feedback policy. It doesn't matter how bad it is, how good it is. I mean, the same policy is always going to do worse here.

Now, if you believe that a good baseline improves performance-- and trust me, it does-- then I don't want the same baseline. I don't want the same B for both of these situations. Because this guy should always be around 50, and this guy should always be around 20, right?

So what I could do is I could have my baseline be a function of x. And I'm going to be like, OK, here my baseline is 50, here my baseline is 20. And let's say I don't know that from the start. I can learn my baseline while I'm learning my policy. So I can use the same policy for both situations.

And then over here I measure my state, and I'm like, oh, over here I'm doing bad all the time. So my baseline is going to be high. And over here I'm always doing well, so my baseline is going to be low.

And so in that way you can take that into account. Does that makes sense? it does look like--

**AUDIENCE:** [INAUDIBLE] this is basically Monte-Carlo sampling and learning. Because each time that you set your-- so your policy is defined by a set of alphas. And then you fix it, you run it, and you get a sample that says what is the value associated with this starting point given this [INAUDIBLE] policy.

**JOHN W. ROBERTS:** Are you talking about Monte-Carlo for policy evaluation? Because Monte-Carlo [INAUDIBLE]. That's like TD infinity or whatever it is. And that's for policy evaluation. That's how you make a critic. The policy is different, right? The policy, you're doing this update, then you're advancing it a bit.

Your critic, the way I just described making the baseline for this, that would be a Monte-Carlo interpretation. You could do it with t, lambda, or anything you wanted to. But yeah. So the important thing is-- I mean, it looks like the sort of blank faces after I talked about that.

But Russ, I think, is going to go into more detail into actor-critic. But maybe I can talk about that more tomorrow if you want. Yeah. I mean, the important thing is that right now this is a very simple kind of idea we've talked about, where you run the alpha, and then if you ran the same alpha, it would always do the same. Or maybe it just has a little bit of additive noise.

But If actually running the same alpha from different states-- which happens a lot in a lot of systems-- the different states could have different expected performance. And so while you'll still learn without the baseline, having a good baseline everywhere will make you learn faster. And so it's worth learning a baseline and learning the policy simultaneously.

And sort of the thing we talked about, where you just average your last several samples to get your baseline, that's already we're learning a baseline, right? We're just learning it for everywhere in state space. We're saying this is the same everywhere, right?

**AUDIENCE:** That idea of sampling, can you do something like [? smarter ?] using Gaussian processes to do active learning on top of it to sample in areas that are more promising? Instead of just randomly moving somewhere?

**JOHN W. ROBERTS:** I mean, there are ways of biasing your sampling based on what you think the gradient is. I mean, that's one of the things we worked on with signal-to-noise ratio. I'm not sure exactly what--

**AUDIENCE:** I know some people worked on Aibos walking, and they wanted to find a gain which maximizes the speed of the Aibos when they're walking.

**JOHN W. ROBERTS:** I think I read that paper, yeah.

**AUDIENCE:** Yeah, and there are like 12 or 13 dimensions. And it seems like a similar problem--

**JOHN W. ROBERTS:** No, I think they use a very similar algorithm. I think they had a different update, though. It was the same kind of idea. I think that the update structure was maybe different than that. Yeah.

So I won't dwell on critic stuff. That's, I think, the last lecture in the class or something like that. But yeah. So here, I mean, this is sort of the sample system. And you can see how this thing is robust to really noisy systems in practice. Because when I ran it on the flapping thing down at NYU, the consecutive evaluations could be very different-- not because of any change in policy, You run the same policy, you get a big variance.

So that's just because you're running on this physical robot with this fluid system and you're measuring the forces in an analog sensor. And so it's just very noisy. But it's robust to that. And that's what's so nice. Put that here.

So look at that. I mean, this one-- these, luckily, didn't take 3 minutes anymore. They took 1 second. So it wasn't nearly as bad. But, I mean, look how much it's changing. It's changing a significant percentage every time, right?

**AUDIENCE:** These are all with the same [? taping loop? ?]

**JOHN W. ROBERTS:** Yeah. Yeah-- I mean, no, this is playing a different-- this is learning. So the thing is that-- I mean, I showed you how it wasn't monotonic before. But this, you can run the same tape. I mean, up there it's pretty much running the same tape. So up there you get an idea of what the noise looks like when you're running the same policy. Right.

And so you can imagine-- yes.

**AUDIENCE:** Just [INAUDIBLE] went with blue and red.

**JOHN W. ROBERTS:** Oh, blue and red are different ways of keeping track of my baseline. All right. So I mean, I don't worry about the different blue and red. They're just sort of an internal test to see the right way to make these things-- we determined that it didn't make a difference. But yeah.

**AUDIENCE:** It looks like the red is much smoother.

**JOHN W. ROBERTS:** I don't know. It may be plotting. I may have plotted blue on top of red or something, too, you know? I don't know. I remember we decided it didn't make much of a difference. Yeah. I see what you're saying. It does look like the variance is a bit less, but I don't think it was.

But these are trials on the bottom. So that's, every second we sort of did another flap, we did another update. So this is update from the bottom. And yeah. This is-- we actually have a reward instead of cost here. So it's going to go up instead of down. But yeah.

So despite the fact this is really noisy, despite the fact that we had this average baseline, which I was talking about-- so our baseline wasn't perfect-- it still learned. It learned pretty quickly. I mean, 400 samples maybe doesn't seem very good. But that's also less than 10 minutes. So that's like 7 minutes.

So it in practice can work pretty darn well. And solving this thing with other techniques would be very tricky. Well, I mean, you could build a model like this model we have and stuff like that, you can try to solve it with a simulation. That's generally how they solve a lot of these problems, is to do the optimization on a model.

So there's this fly. Jane Wang at Cornell tries to optimize the stroke form for a fly, like a fruit fly. I think it's a fruit fly scale. And so she just built a sort of pretty fancy model of this thing and then simulates it. It does the optimization on a computational fluid dynamics simulation.

And so that's some way we can-- and there you can get the gradients, you can do all the sort of things we've already talked about. Because you have the model, you can do all these things explicitly. But the model takes a long time to run. I think the optimization took months of computer time.

So if you-- that's the thing here, is that the full simulation of this system, where it took me 1 second to get an update, it takes, I think, about an hour per flap. So an hour on a computing cluster to get one full safety simulation of one flap. And that's even the simpler one.

We're working on other ones, too, that have sort of aeroelastic effects, which are where sort of the body deforms in response to the fluid forces. And simulating those is even harder. And so where it takes an hour to get an update, I can get it in a second.

And the thing is my update is going to be noisier and I don't get the true gradient. But when you can get 3,600 updates per update, you're going to win. I mean, I'll get one flap in the time takes me to optimize and sit there for most of an hour, you know? So you can see in those kind of problems, it can be a big win, especially when a simulation is extremely expensive, or computing the gradient is extremely expensive, but you have the robot right in front of you. You can just take that data, accept the noise, do model-free gradient descent.

I think that's what I wanted to talk about. If you have any questions or anything didn't make sense at all, please let me know. Otherwise, maybe I'll introduce something that I'm trying to talk about tomorrow, a different interpretation. I'll just try to get your brain ready for it, I guess.

But if there are any other questions on this, please ask.

**AUDIENCE:** What was the reward function for [INAUDIBLE]?

**JOHN W. ROBERTS:** The reward function for this was the integral of velocity, of spin velocity, over the integral of power input. So it measured the force on it, multiplied that by the vertical velocity. That gives you the rate of power. That gives you power, which is the rate of work. And then it just sort of calculates the distance.

And so that ratio is what we tried to optimize. So it tries to figure out sort of the minimum energy per unit distance. And so it spins around in a circle, but it's a model of it going forward. So we did it for an angle, but you can do it just as easily for if you had a linear test. It's just harder experimentally. And so it's try-- it's sort of an efficiency metric.

Yeah? All right. Turn the lights back up. Make sure I crossed all my Ts, dotted my Is. Oh, yeah. And actually, there's one story, too, before I get into that thing.

So a lot of these things originated, like a lot of the things we've seen for neural networks-- like back prop, like gradient descent. I mean, we learned that [INAUDIBLE] originated in the context of neural networks, RTRL did. And a lot of this did, the reinforce algorithm, which is the thing we're going to talk about-- originated with neural networks.

And one of the reasons they found so appealing, particularly like this kind of stochastic work, is that it seemed biologically plausible. That it could be like, what is the chance that a human brain is doing back prop? I mean, it could be doing some sort of approximate back prop or something like that. I actually don't know that much about neuroscience.

But the thing is that these sort of computationally involved techniques for solving these problems don't seem like they're reasonable as sort of postulations on how the human brain or how neurons solve these problems. But this one, you can see, it's so simple. And the little randomness being part of it and just the sort of like simple update structure does seem biologically plausible. Just sort of intuitively, it makes more sense.

But even more than that, there's examples of-- there's data and evidence that suggests that these kind of things could be one of the aspects of how animals learn. And the coolest one, I think, is there's these song birds that learn how to sing. Like, they don't-- they're not born knowing a certain way to sing.

But they hear their parents sing as they're growing up, and they start singing more and more. And they get better and better. And actually, you can hear them getting better until they sing like their parents did. And you can raise them in captivity and play them Elvis all the time, and they'll do like a song bird impression of Elvis, which I'm surprised you can't buy the CD of that on late night TV.

But the-- right. But so a really cool thing, though, is that there's this part of the brain where, if you measure sort of the signals, they seem to be completely random. Like, they just seem to be random noise. And so it's like-- it's strange that there's not this structure. It's like, what could this part of the brain be doing? Why would it need to be producing random noise?

What they did-- and this is your-- maybe you bird lovers out there won't like it-- they took one of these birds. And while it was learning-- like, they waited till a bird learned like the full song. And then they deactivated, through some means, the part of the brain that produces random noise.

And nothing happened. The bird-- apparently, the bird wasn't like entirely the same. But it still could sing the songs fine, everything like that. Then they took a bird who was in the process of learning the song-- had learned some of it but wasn't perfect yet and was still getting better-- and they deactivated that part of the brain.

And it started just singing the same song. Like, how it'd been singing, it kept singing. It didn't get any better. And so that's some sort of proxy evidence that this random noise was related towards the ability to improve, that it's not storing the signal, that it's not necessarily like the descent itself. But just this random noise could be how it's screwing up its song in an effort to get better and better. It screws up a bit, listens, and maybe it's a little bit better, and it does that.

So that's sort of a pretty-- I mean, and sometimes compelling evidence that, I mean, biology could at least use this as some aspect of its improvement, that you shut down the random noise and it stops learning. I mean, if you get the variance to 0, you're not going to get worse, you're just not going to do anything. You're going to keep singing the same song.

So that's sort of cool, I think. Right, so just give you something to chew on. There's another interpretation of this. So here we sort of talked about this one. Here I think our idea was this sort of sampling, where we have some nominal policy. We perturb it, measure how good we did, how well we did, measure performance, and update.

So this is pretty much what we have, is we have got some policy that we're working at. We add this z to it that changes it a bit. We run it, and then we update.

There's a different interpretation-- my performance got too long. There's a stochastic policy interpretation. Now, in this, the way you think about it isn't that we have some nominal policy and we're adding noise to it. It's that your policy itself acts stochastically. So actions are random.

Doesn't mean that they're completely random. I mean, they're random with some distribution. But you're not saying exactly what you do. And so you can imagine this is sort of like if you're playing Liar's Poker-- you know, where you hold the card above your head. And then you can see anyone else's card but not your own, and then you sort of bet on these things. Do you know the game?

Maybe that game doesn't have enough cultural penetration to be a good example. But if you're playing normal poker, any sort of gambling games, if every time you had the same cards, if you made the exact same bet, people could eventually sort of figure that out, and maybe they could use it to beat you.

There's plenty of games like that, where, say, every time I have a certain card, I always bet this. Then if I bet that way, they're going to be like, oh, he has good cards. I'm going to fold. Or, oh, he always bluffs when he has this card. So this sort of deterministic policy doesn't make sense. The stochastic policy is exactly what you do.

So your policy is going to be like, oh, I've got like pocket kings, 95% of the time I'm going to raise whatever, and [INAUDIBLE] time I'm going to check-- those kind of things, where you're sort of-- there's some noise in what you do.

Now, you can question whether or not that makes sense as whether optimal policies would be stochastic in the kind of problems we look at. But the important thing is just to realize that your policy, don't think of it as it's doing these things. It is these sort of distributions of what you do. All right?

So the parameterization, then, controls distribution. Ooh. My fifth grade teacher would not have liked that. But.

So what you do, then, you can imagine you control, perhaps, the mean of a distribution. So where over here we had this-- you can think of it as really sort of exactly the same, where my other interpretation said, OK, my policy is alpha, and then I add random noise z to it.

While here, my policy is parameterized by alpha, and my action is the same thing. It's just that it's not, this is what I'm doing and I'm sampling something else. It's that, this is actually my policy. If I ran the same policy, I would just do all these things with these probabilities. So it's your actions are stochastic.

Now, that's sort of something that isn't always completely-- well, when I first saw it, it wasn't really easy for me to get my head around all what that meant. But yeah. So this is-- we're going to look at this. Yeah, I won't go into more detail. But tomorrow we'll look at this sort of different interpretation of how to do this.

And you can get the same learning. we'll actually show that the update's the same, the behavior's very similar. But the properties are a little bit different. And the big thing is that you don't have to do this linearization. Here we did this sort of linear expansion and we say, OK, so this is true locally. When you look at it in this context, you can show that you'll always follow the gradient of the expected value of the policy. All right?

And so that's a big difference, right? Here we're saying, OK, we're looking at the local gradient, we're going to follow the local gradient here. But let's say that you have a very broad policy or a very sort of violent value function. Let's look at this 1D one again, where my value function has something like this-- extremely violent.

Well, when I put this random stochastic policy, that smooths it out. And so even though-- it's because I have a stochastic policy. Running my policy, the cost is a random variable now, depending on what my actions are. Even if my dynamics are deterministic, because my policy is stochastic, my cost is stochastic, I'm going to get some-- if you look at this, there's some expected cost for running this policy on this.

And you do this, you're going to sort of-- you can imagine sort of smoothing out some of this, right? Sort of averaging over all of these. And what you follow when you do this-- and the update is really identical. Like, it's actually just the exact same update. Possibly, there's a coefficient up front that you could put in or not. But the structure is the same.

And the thing is that it'll follow the expected value of the performance of the stochastic policy. So it's sort of a different way of thinking. I think that this way is sort of the easier way to sort of first think about it. But tomorrow will be more probability kind of things. And we'll talk about the stochastic policy interpretation and some of the ramifications of that. Yeah. And maybe some other interesting side notes. So yeah.