**JOHN W. ROBERTS:** Our investigation of stochastic gradient descent. This time, we'll be talking about the stochastic policy interpretation. And I have some examples. Well, a very simple sort of system example. But a whole bunch of ways of looking and fooling with it in policy parameters that hopefully give you some feel for how to actually go about using this, if you want to use in your project or something like that.

So here. OK. So I gave a little introduction to the idea of a stochastic policy last time. But this time, I'll give maybe a little review of the things and also a sketch of the algorithm. I think someone asked, Joe asked for pseudocode. To get a feel of the process you need to go through. So just in sort of words, this is the process. I'll just get right into it.

Set. Alpha. And again, there's a bunch of ways of actually implementing this and going through the details of it. But this is a very simple way and easy understand. And most of the other forms are pretty clearly derived from this. We can sort of tweak it maybe to get a little bit better performance or fill in the details in different ways. But set alpha to an initial guess. Wow. Sorry. So this is what you start your policy off as. You can start it as zero or a bunch of random numbers. And this can affect your results a lot actually. Good initial guesses will obviously converge faster. But also they can be a bit different base of of attraction for different local minima.

So it also makes sense sometimes to run it several times from different initial guesses and just see where it goes. And if they all converge to the same thing, that means you're at least in a local minima that has a large base of attraction. While, if they are go to very different things, that means that you sort of have a local minima problem. And you can take the minimum of all those runs or you can try to investigate exactly why you keep on getting stuck.

So the initial guess isn't just like come up with the best thing you can. It also means sort of map out the space of policy parameters somewhat so that, when you get stuck in a local minima, you have some confidence about how good that policy really is. So then this is again, you don't have to do this. But run a policy pi alpha. So this is the policy with your parameter set of your alpha. And store the cost as your baseline. So that gives you your first baseline.

Then let's see. The Do While loop here. So do draw noise z. So if you remember before, maybe I should sketch this first one and fill both these boards with the sketch, z is the noise you add to your policy. So when you're searching, when you sample, that's the vector you add. I'll write out the details of it over there. So draw your noise z from some distribution. Now, yesterday we talked about the example of a Gaussian. That's a common one. But there's a number of other ways. And actually, today we'll talk about a different distribution and sort of why you'd want to use it.

Run system with pi alpha plus z. So this is sort of when we're evaluating how well that policy does. And then we'll say we'll store this just in J. All right. Now, we do the update. Remember? That we were talking about. So alpha gets alpha plus negative eta J minus b. b is our baseline up here. So J minus b z. And then update your baseline.

Now you can do this with a second run where you run your new policy. That's sort of the more expensive way. But you can have confidence in that baseline, as long as it's not too random when you evaluate the system. You could do a decaying average. You could have, for example, b gets 0.2 times your J plus 0.8 times your old baseline. Right? And so, this one is sort of an exponentially decaying average where I take my new one, smooth it out by averaging with the previous one. And then if you think about what this does, sort of every measurement gets sort of exponentially decayed away and its significance as you go through time. That's the update.

And then you can do this while not converged. So many times, you can just do a four loop, too. And you can just run it for 100 iterations or several iterations and see whether or not it looks to have converged. That's what I often do. But this is really what I guess you're doing. It's just the four looping stuff is sometimes easier and prevents you from getting stuck where your convergence conditions ever met and you have to Control C. So hopefully this is clear enough that, if you wanted to implement this, you can go do it and try for your project if that's what you interested in.

All right. So--

**STUDENT:** [INAUDIBLE].

**JOHN W. ROBERTS:** I'm sorry. This is et cetera. There's different ways you could update your baseline. You could do it with a-- yeah. Sorry. I mean, these are two common ones. I mean, these are the two that I usually use. But you could have critics and stuff like that, too, where you have more complicated updates for having more complicated baseline.

All right. So when should you use this? We'll just say stochastic gradient descent. Well, it's important to remember, one, that it never does better than true gradient descent. So if it's cheap to evaluate the true gradient, you can follow that true gradient down. And you're in pretty good shape. Now the fact that it's random maybe gives you some robustness to very tiny local minima. But you could add that in your gradient descent, too, if you're just [INAUDIBLE] yourself around a bit.

So stochastic gradient descent, if you have an easy way to compute the gradients, you can do gradient descent using those. Or you can even do something like SNOPT or some high order method. So if it's easy to compute the gradients, exactly. It doesn't necessarily make sense to do stochastic gradient descent. If those gradients are impossible to compute or extremely expensive to compute, then it can make sense.

Then also, if you have a good parameterization, it can be quite efficient. Again, I guess this isn't a prerequisite because it will work with high dimensions. And just sort of like in naive parameterizations. Wow. You just leave params. But it'll be really slow. But is slow. And it can get stuck in local minima. So if you have a good parameterization, it can be much more reasonable to use this. An example is, in our lab, we have this glider we're trying to have perch. So we launch it out this catapult. It flies through, executes a maneuver, and then tries to catch this line. Right?

Now, in that case, if your parameterization were the kind of open loop tapes we've previously used, where just what elevator angle did you try to serve it to, that could be a bad parameterization. It's a course that's going to demand sort of rough things. And so, trying to do learning on that can be very slow. Especially because evaluation is so expensive. So if we can come up with a good parameterization though, suddenly it becomes reasonable to do this. If we can knock down to five parameters that parameterize a very sort of nice class of policies, now maybe we don't need that much data. And we can actually get it just by launching it. So if we can come up with a good parameterization, doing this should work. Yeah?

**STUDENT:** What if you had a good estimate of what your trajectory should be? Do you parameterize [INAUDIBLE]?

**JOHN W. ROBERTS:** Well, I mean, you could start with a good initial guess, which maybe means that you converge reasonably quickly. But the thing is that the learning is going to struggle to get a lot of improvement. It will get some improvement. It'll show in these examples you have ugly parameterizations, it'll still learn a bit. But the problem is that, if you parameters in this open loop tape, let's say we have some really nice smooth trajectory like this. And we parameterize it. Is this all clear? Is this is sort of off on a tangent? But this is just sort of about bad parameterizations.

So do you want me to go through all this first? And I can talk about the points in detail?

**STUDENT:** Do you need a microphone?

**JOHN W. ROBERTS:** I believe I have a microphone.

[INTERPOSING VOICES]

Thank you. All right. So are we all on board here? Or are we-- yeah. OK. Good. Yeah. This is sort of disorganized. So if you parameterize by setting all these numbers, you have a nice smooth trajectory. That's the kind of stuff you want. Now if you parameterize by saying each one of these numbers, stochastic gradient descent when it bounces around is going to be like, OK. Send this guy up a bit. Send this guy down a bit. These two guys up. This guy down, this guy down. Up. This guy back in the same place. You know? Something like that. And you get some param policy like this. Right?

Now your physical system is going to smooth that out. It's going to filter it. Right? You're not going to execute something like that necessarily. But it's very unlikely that that's the right kind of policy. Right? I mean, it's very sort of fine-grained tick-tick-tick doesn't really make sense. So if you assume you are going to have something smooth, it's nice to do something that's going to be sort of always relatively smooth. If you were to use fewer points parameters as a spline, that could be a win. So that's what I mean by a good parameterization is something that ideally captures the kind of behaviors you want in your system.

So yeah. I think the main point is don't-- well, this is not the main point. This is a big point that I think a lot of people tend to do, and I can be guilty of myself, is don't discard Back prop through time, RTRL with SNOPT. Many times, because you can see how simple this update is, it's very tempting just to be like, OK, well, I'll put this loop and run it. And I'll be in good shape. I mean, because it's trivial to write that code. I mean, you can see it from pseudocode here. So when you're working on your project, if you're like oh well, I want to optimize. I'll just throw this on there. Bam. I mean, it'll take you 30 minutes to get that code ready.

But maybe you think back prop through time is more confusing, you have to do with all these joint equations and stuff like that. But these give you the true gradient. If your gradients are cheap, doing this can be a win. And then you can get a better policy. You can solve it a lot faster. You can check richer classes of parameterizations because it will solve them so quickly. So when you're thinking about, OK, we're trying to solve this problem and you have your project, don't choose this without being very conscious that it has a lot of limitations. And there's a lot of alternatives which could be better.

So first think, are there very solid sort of things you can do with these using SNOPT converged nicely? If you can't, if you don't have a good model, if your system is too complicated, stochastic range descent maybe is what you want. All right. So that's sort of a discussion coming back from Tuesday on when to do these things and also hopefully code that now that you have it in your notes. Now that you have it in your notes, you'll be able to implement this pretty easily.

So now getting onto new stuff for today, we're talking about stochastic policies. And in particular, we are talking about a certain class of these kind of algorithms. Reinforced algorithms. All right. And I think that these were introduced by Williams. He's at Northeastern in '92 or something. So once again, trying to get across the idea of the stochastic policy.

So this interpretation is very different. We're not going to linearization. We're not going to assume we have this nominal policy, we sample and try something else. We're going to assume that we have some distribution of policy. Our policy is a distribution. So we can think about it as, we parameterize our distribution with alpha. Our policy then has some stochasticity to it, which is going to give us what used to be our alpha plus z. Right? So we know that this is not a sample. This is the output of our policy. This is how we represent our policy. Not as the nominal action, but as some parameterization of the behavior.

Right. So this would be, for example, if I were to play a game where I was flipping a coin and I bet on something, it could be like, OK. My parameter here is the percentage of times I'm going to bet money on it coming up heads or something like that. I mean, that doesn't make sense because you're [INAUDIBLE] is zero or net zero if you're playing a fair game. But the point is that it's something like, this is the probability of doing this.

And so here, we're going to think about this as the mean of a bunch of Gaussians. So our policy is do all these actions with the probabilities described by a multivariate Gaussian with means described by this vector. All right. OK.

[INTERPOSING VOICES]

**STUDENT:** So you just have alpha plus z, is the policy just to add noise to your parameters?

**JOHN W. ROBERTS:** I mean, these are the two different interpretations. Right? So I mean, it is effectively doing that. I mean over here, we have our nominal thing. And then we add this noise. Right? Then we have our nominal policy. We add noise with distribution like this. But this one, what we can have is this is parameterizing a distribution. And our policy is to do these things with this probability. Right?

**STUDENT:** And you said that alpha is the mean--

**JOHN W. ROBERTS:** For us, it's the mean of these Gaussians. But you could parameterize it any way you wanted. I mean, you could have a policy parameterized by, if you have discrete actions, it could be the probability of all these actions and force them to sum up to one. It could be some parameterization of a PDF or something like that. So we're going to talk about it in a limited case. And I think that thinking about that way, at least for me, is easier to think about where we're going. But remember, that's a lot more general. This is just parameterizing some distribution of actions. And so, it could be something very general.

But if you're having trouble putting your head around what all that means, you can think about it as just like, this is describing the means of a bunch of Gaussians. And then our policy is do these actions with the probabilities described by that. Does that make sense? I think that this is sort of, I feel like I'm running around in circles here. All right. I just want to be clear.

So in that case, it doesn't make sense to say, so this goes through our system and produces cost J. But now J is a random variable. All right. So it no longer makes sense to say, what is J for this policy? Because running that policy is going to produce all sorts of different Js. Right? So when you evaluate the policy, I mean you could have a question of, what is the right way? Should you do the min of J? Should you do, what is the worst J it's going to produce? Or what is the maximum J it's going to produce?

Well, the thing that we generally choose in this world-- well, not Earth, but our discipline-- is you want to look at the expected value of J. And so, this is a philosophical decision to some degree. I think Russ may have talked about this a bit. But when you're an airliner, you probably want to look at the min of J. When you're building a gambling robot, maybe you do want the expected value of J. So we have a lot of money. Right? And so, this is a philosophical decision. But it's analytically tractable. And it makes sense in many cases.

And again, like Russ says, animals aren't doing robust control. I mean, if they were, we wouldn't be sprinting around and jumping, doing gymnastics. Well, some people would be doing gymnastics. I'm probably not doing them anyway. But my expected value of trying to do gymnastics would not be high. I can guarantee that much.

All right. So what we're going to look at is our stochastic gradient set now. Since we can't just dJ d alpha, we're going to have to look at d expected value of J d alpha. Right? So this is now our metric. And so we have to look at the gradient of that expected value. Now, this then is the definition of expected value. Right? We can call, I'm going to call this sort of like the policy parameters that you're running on under this trial. I'm going to call it beta in an attempt to be as unoriginal as possible.

So we're going to call this beta. So the expected value for J we want to integrate over all beta. And your expected value is going to be-- I'm sorry. I've got an error in my notes. So J of beta, probability of beta. So this is just definition of expected value. Now, if you push the d, d alpha through-- yeah, sorry. If you push the d, d alpha through for a beta, what you're going to find is that J of beta is not a function of alpha. We're saying beta up here written up here is a function of alpha. I mean, that's how we've derived it.

But remember that beta is just the actions you've chosen from a distribution. So if you look at running your system with these parameters, that's not a random variable. That is just your number. Right? So the thing that varies though is the probability of getting beta. Right? This is the function, this distribution. So this is what depends on alpha.

All right. Now, this is where a bit of cleverness comes in. You can write this then as a handle over beta, J beta P. So you get the distribution beta. Then d, d alpha of ln of p of beta. d beta. Right? So this is just saying, when you take the derivative of your natural log, you're going to cancel out this one. Right? You'll get the derivative of the one inside. That makes sense? When you do a chain rule on this, you're going to get one over P beta d P beta, d alpha. So this is sort of the trick that you need. So you should see this.

All right. I can erase the pseudocode. So once you do that, if you look at the expression over there, what you can see is that it's the same as the expected value of J beta d ln P beta d alpha. Right? Because we're integrating overall beta, probability of beta. Right? So what's the expected value of this? That means that, if we can make this our update, if we can make effectively this our update, then the expected value of our update is equal to the derivative of the expected value of our performance with respect to our policy parameterization. Do you see?

That's sort of cool. Because this seems like sort of a complicated thing, the derivative of your expected value with respect to your policies, other things. It's this big random thing over a very complicated J. Like a general J. We haven't assumed anything about J. We didn't do a linearization. We didn't do anything like that. Right? So J is still however general you want it to be. It's not local anything. The derivative this with respect alpha, we can make this update. We follow that derivative. So that's pretty cool. Right?

So the question is, now what does this update look like in practice? This is kind of an ugly term. So what happens when we actually try to do an update like this? Well, we can write this as our update in the direction of the gradient is going to be delta alpha again. Now we want a learning rate because this is again sort of a gradient following thing. So generally, in there, you want some sort of learning rate. And you're just going to do a gradient descent.

You get this J of beta, which again now we can write this is alpha plus z. All right. And then something to represent that d ln d alpha P beta. So we'll call that E. And that's called the eligibility. I think that term may come from neural networks. But you can think of it as this eligibility captures how sensitive each parameter should be to an update. So let's say we have 10 alphas. And the eligibility for one of those alphas is very high. That means that, if we do well, that eligibility should go much more in that direction.

The eligibility is sort of just capturing the weights of how much you think each parameter is responsible for affecting that output. Right? So a big eligibility on a parameter means that that parameter is going to move a lot. If the eligibility is small, it's not going to move much at all. So does that makes sense? So that's the way to think about the eligibility. All right. So eligibility is just capturing what we expect the significance to be. Yeah?

**STUDENT:** Is it the quantity inside the expected value brackets? Or is it the expected value?

**JOHN W. ROBERTS:** It's this. It's just this quantity. It's not the expected value. We want our update to be this inside the expected value. Then the expected value of our update is the gradient. So it's just the quantity inside. Right. So I think I remember, when I first learned this, Russ was talking about the eligibility. And I had no idea how to interpret it. And if you still are confused about it, I can try to describe it in more detail maybe with a picture or something like that if you'd like. Do people think they have a good idea of what the eligibility means intuitively? OK. Great.

**STUDENT:** [INAUDIBLE]?

**JOHN W. ROBERTS:** Pardon?

**STUDENT:** I know eligibility [INAUDIBLE]?

**JOHN W. ROBERTS:** I'm going to go through how you calculate it right now. Because E, you see, is going to be this. It's going to depend on what our distribution is. So I said, if we parameterize our policy by the means of a bunch of Gaussians, eligibility is going to be one thing. If you parameterize it as a bunch of Bernoulli variables or something like that, it's going to be different. So it depends on the kind of distribution your policy uses. All right. So-- yeah?

**STUDENT:** If alpha isn't the means of a bunch of Gaussians, then what does alpha plus z mean?

**JOHN W. ROBERTS:** z, alpha plus z is always the action you take. Sort of the-- we actually had some notes on this. If you think about it, let's say that the way you represented your controller, there are some subtle differences here. But yeah. I think it's worth going through.

So let's say I parameterize my controller by three numbers. Like gain on position, gain on the speed, and an interval term. So this is a PID controller. Right? So this is something that most of you I think are probably familiar with. So this is a very simple parameterization of a policy. Right? Now, this how we control it.

Now the thing is that this is sort of what the actions are. So this would be, a beta would be one of these. An alpha-- and this terminology isn't necessarily standard. I think the alpha and z are. My beta just came up when I was trying to make these notes so I could have a simple term to write down all these expressions. But so this is your beta. This is the thing you're running your system under. Your alpha is the same size as that. But what it is is parameterization for some distribution of how you select these. All right?

So in a Gaussian, I mean I give you a simple one. A Bernoulli distribution, this could be-- let's say KP was either one or five, then my alpha one could be the probability of picking one. And then the probability of picking five is one minus alpha one.

**STUDENT:** So in this case, it's a specific evaluation of your stochastic policy?

**JOHN W. ROBERTS:** This is a specific evaluation. This is the parameterization that defines the distribution. So once you tell me alpha, I can tell you the probability of getting any of these. Right? But when I run it, I have these numbers in there. So this is sort of like a sample from a distribution. And this is what I actually get the cost of. Yeah. All right. And so, if you look at this minus this, that's the noise we talk about in the other interpretation.

**STUDENT:** And the system you're trying to solve is not, [INAUDIBLE] how do you decide [INAUDIBLE]?

**JOHN W. ROBERTS:** Yeah. So we're thinking about this. This is another thing I have a note on. We're going about this in the context of trials. So you run the system under a trial. Right? And then you evaluate the cost. What if your system is just running constantly? Right. Well, you can think about a short period of time. As a trial, you could run it for a while. You can look at discounted cost. And then there's also the average cost or average reward formulation where you can look at how long, what reward you expect to get running it off to infinity. Like, what sort of reward rate.

I think about it right now in the context of, let's just say we can do a trial of something. And if you want to worry about what happens if this is constantly running, there are ways you can still do a trial. And then you can keep track of an eligibility trace, which is different than this eligibility. So that's a confusing thing. What you can do is you can be running constantly. You can just sort of keep iterating and take into account the sort of coupling or [INAUDIBLE]. You can imagine it's sort of like, if I study for my test tonight, I don't have a good night. But I do well my test tomorrow. So I was happy.

And so, it's sort of like, then my action didn't give me an immediate reward. It gave me a reward later. That's sort of the delayed reward problem, which is very common. And there are ways of dealing with this here where you can still just look at your accruing award right now. And you can still sort of give yourself a reward for doing good things in the past. You can still handle that. So I mean, that is a good question. And there's a number of ways to deal with this. But right now, just think about executing a trial. Right?

So then our E here-- again, to make it look like what we want it to look like, to make it-- well, that tells us exactly what our E is. So we can write it as this vector. d d alpha one of ln probability of beta. And this beta is still this vector. But the probability of that is a scalar. Right? So this element of the probability is just a scalar. And that's our entire thing here. Right? So then let's say again, this is where the distribution you choose comes into play.

Let's say that we chose our distribution to be a bunch of Gaussians. All the different parameters ID, the noise ID. But we'll just think about as an independent with the same sigma. So the probability of getting a beta, because they're all independent, is the product over all the different parameters. 1 over square root of 2 pi sigma squared. This is going to cramp it if I do that. So I think I have the opposite problem of anyone else. I write way too big. Video people will be happy. So let just me squish it in here.

So probability of beta is equal to product over all my different parameters with 1 over-- well, you know the Gaussian distribution. It's like this. Right. So if you give me a beta and I have the alpha, I can tell you the probability of us picking that beta. All right. So our ln, the natural log of this, will then allow you to turn this product into a sum. And you'll get the sum of over i equals 1 to n.

Now we can take this derivative. This is a constant. That won't show up. And we'll just have the derivative of-- yeah. Yes. So the derivative, now we'll get rid of this. This one is simple to do now. It's just a product. So the derivative of ln P beta d alpha-- I did this again-- equals. So beta i minus alpha i over sigma squared.

Now if you remember how we talked about it over there, it ends up being [INAUDIBLE]. So here is our eligibility. z over sigma squared. Now what does that mean our update is? So hopefully, this looks familiar. My voice is struggling today. I'm going to come in next time talking like this. It'll be great. So this is our update. Does this look familiar?

[INTERPOSING VOICES]

Yeah. Actually, let me-- yeah. I guess so. Does that look familiar? Is this no, it does not look at all familiar? Or is this yes, it looks so familiar, I don't waste my time and embarrass myself by saying it does? It does. Right? It's the exact same update. Right? Now this one, we don't have a baseline. And we have sigma squared. But that sigma squared is just a scalar. And we already said, baseline or no baseline, whatever you want, it doesn't affect it. Right?

Now, is that true for this case? It didn't affect the other one. This one, it also does. If you think about your reward as-- if you think about your reward as one everywhere, then your derivative, let's see. Where is this exactly?

OK. All right. So if this is constant, then our derivative expected value is zero. That means that the expected value delta alpha over here has to be zero because it's equal to that expected value. That means the expected value of our eligibility is zero. Right? So the expected value of the eligibility is always zero. So that means that some constant in there is not going to affect it. Right? When we do this expected value, when we put in our baseline, that's going to turn into zero. Because the expected value of E is zero. The exact same thing we had previously. Right?

And you can see it's the case for this specific example of z. The expected value of z is zero again. So it's not going to affect the direction of our update and expectation. So yes, you can do anything like this with eligibilities. You can still use a baseline. Right? So this is really the exact same update that we already found. All right. So I think that's pretty cool.

Update that we originally interpreted as locally following the gradient is also following the true gradient, not some sort of local but the true gradient of the expected value of the performance of that policy. All right? So if you have some really ugly value function, you throw this distribution on top of it and you move it around. It's going to be some, it's going to follow in the direction of true improvement there.

And a little test case. I remember when I was trying to think about interpreting these things back when we were originally doing some of this analysis on the signal to noise ratio, if you think of little cusp, if you have a value function that has a little plateau on it, I can write this on a fixed board because it's not going to stay. If you think you have a value function that's sitting around here, my drawing isn't going to be too clear. But here, we'll do it in 1D.

If you have a value function like this, you sit here and follow the local true gradient. It is very small. The other one says that you'll follow that gradient, no matter how big it is. So what it's sort of doing when you have this Gaussian is it smooths out the kinks and stuff like this in your policy. Right? Because you think my local analysis says, OK, well, I'll just get local enough that these little kinks in my value function aren't a problem anymore. But here, we say we didn't say it was local. We said it was global. And so, it's following this gradient, even if we have ugly stuff in our value function.

The reason is because this distribution sort of smooths that stuff out. And now it's able to follow it. So that gradient is still well-defined. It takes the stuff and smooths it into something. Does that makes sense? I think it's just the big sort of difference in interpretation right here. There's the one where it's this local analysis and follows the local true gradient but only locally.

And then we have the other one which follows even, it doesn't require a small sigma or anything like that anymore. It'll follow the expected value, the true gradient of the expected value of your reward. And it's too big of a mouthful for it to be clear just by saying it. But these are the two different interpretations. All right? Good.

**STUDENT:** That function is basically the policy.

**JOHN W. ROBERTS:** Oh yes, I'll draw this better. So here is sort of our alpha. This is our reward. Here we have our [INAUDIBLE] function. Our policy is taking actions with this probability. All right? That's our policy. So you can see that this-- even though this has a kink in it, as it moves that over a bit, this is varying smoothly. So even though this is a little disconnected like that, this is going to slide over still smoothly. Right? So the gradient is always going to be nicely well defined and everything. And this also goes out to infinity. So even if there's some ugly thing way out there, you're never going to suddenly go and start seeing stuff you never saw before. You always sort of saw it. Right? So you'll be following the sort of smooth one all the time.

And that's nice because maybe you won't be following when you get really broad. One interpretation says, well, we're not in the linear regime anymore. So we're not following that local true gradient. This is sort of the issue that brought it up. Let's say this one says it's the linear analysis. And it says, OK. If my sigma is small enough, so I sample close enough so it looks approximately linear, I'm going to follow that gradient. But instead, when it breaks down, you get this really broad Gaussian. So what does that mean?

Well, this one is still following that gradient. So that means that you're seeing stuff all over. And it's following that gradient instead of this local one. Right? Now the problem is that, if you follow that gradient, the stochastic policy isn't necessarily actually the optimal one to follow. When you're actually running your policy in the end, certain systems you do. But in many mechanical systems, you don't actually want these random actions. Right? There's the action that actually does minimize cost.

And so, the sort of local interpretation has some value because you don't necessarily want to stick with the stochastic policy forever. That's why you sort of reduce your sigma. And you get tighter and tighter and tighter. And you follow the gradient so it's more and more biased to the right spot. Because you can imagine, if you have a value function that's a global min here, and then say this is all extremely high.

And this goes down and it's like a low plateau. Now, if you have a really broad Gaussian, it may say, OK, well, even though this is the best spot to execute, I'm going to slide way over here. Because I have a sort of lower expected cost. Right? That put you in the wrong spot. If it was really tight, then I'd say, OK. Sit right here.

So do you see that sort of distinction? That this broad guy could push you in the wrong direction if it's really broad. And you may not, in the end, want to execute a stochastic policy for this exact reason. Your stochastic policy could have a much higher expected cost than a small sigma stochastic policy. Down to where, if sigma is zero, that's when you actually have the minimum expected cost. Yeah?

**STUDENT:** So when is this used? It's used when you're starting and you don't know about systems?

**JOHN W. ROBERTS:** When you don't have a model, it's used in the same context as the other one. Are you saying stochastic gradient descent as a whole? Or--

**STUDENT:** I mean stochastic policies.

[INTERPOSING VOICES]

**JOHN W. ROBERTS:** It's used for exploration. Right? Because our other one, where we had this nominal policy and we add a noise, that doesn't look any different, except from this sort of interpretation, than this. Right? So--

**STUDENT:** So for the [INAUDIBLE] and learning about systems--

**JOHN W. ROBERTS:** Yeah. It's for exploration. And you need it to get more data. Right? Because if I run the same policy all the time, I don't know how things are sensitive. So my stochastic policy gives me this information to where I can learn. Right? So in the end many times, yes, you'll want a deterministic policy. But for the development of that policy, you'll want to execute all sorts of things that could be suboptimal to give you information. And so, that is the interpretation here. It's for that learning.

And if your system is evolving and stuff like that, if it's not constant in time, you may want to always run a stochastic policy. Because you'll be able to constantly learn, if that makes sense. So you may never want to converge. But I think it's-- yes?

**STUDENT:** How do you obtain the parameters of that stochastic policy with that probability? It's very sensitive to sigma, right?

**JOHN W. ROBERTS:** How do I get the parameters of what?

**STUDENT:** Of that distribution.

**JOHN W. ROBERTS:** Distribution? Well, I mean that's what I've set. Right? I said I'm going to act with the probability described by a certain distribution. And I describe the distribution here. So I know these probabilities. Because this, I set. That's from my controller. I designed that myself. What I don't know is this, this value function aspect.

**STUDENT:** So that it should be based on intuition?

**JOHN W. ROBERTS:** No. No. This up here? Oh, how do you decide how broad this? Intuition has something to do with it. Also, you can imagine sampling. I think I mentioned this on Tuesday. You could sample. You can imagine I sample a few points and I'd look at them here. And they look straight to me. And I'm like, OK. Well, what if I sample bigger? And I get these points. And it looks relatively rough. Then I go, OK. Well, I want to be sampling on probably the regime. We're going to see this roughness to some degree. I don't want to be stuck where it's going to take me forever to go anywhere because I'm in the linear regime. So you can do some sampling and get sort of the coarseness of your value function.

And so, I mean that's sort of before you run it, you do some of those things. And you figure out how big these parameters are. Because the alternative would be you set it, you see if it how it does, and you fool around with them all the time. That's another way to do it. But this way, you sort of have a more direct process for trying to be like, OK. I want to be sampling where I actually get some interesting information.

And so, you want to be sampling where you're probably getting distributions that cover linear-- it's fine if it's linear. But you want to actually sample to where you're going to move around to these different features. You want to sort of be able to get, you want your updates when you change your policy to be on the scale, too. Because it was really small, it could take forever to get down to the minimum. Or if I was over here and I had small sigma, I may never sample over here. So I'd never know to move that way. Right?

And so, if you have a bigger sigma, you might be to get out of this local min. So I sample over here and see it's better. And move in that direction. Despite the fact that, locally, it was bad. So there's a number of considerations. But yeah, doing some sampling and stuff like that combined with some intuition. That's probably the best I can give you as to how to set these parameters. I mean, that's a tricky thing. That's something that's nice about SNOPT. There's no eta to set. This one also has a stigma to set.

So please, Russ emphasized that I was to make all of you understand this stuff really well. That's why I had two days. And if you do it on your project and you apply it at the wrong system, I'm going to be blamed.

[LAUGHTER]

Yeah. So if you don't have a model and you don't have anything like that, try that back prop first if you can. Try SNOPT first if you can, I think. If you can't, this maybe won't do as-- this can solve the problem, too, I guess. But sort of appreciate its limitations.

**STUDENT:** So with the stochastic gradient and stochastic policy, you also have to be careful about how you parameterize everything. Right?

**JOHN W. ROBERTS:** Oh, very much.

**STUDENT:** Because, I mean, you are taking the expectation over these betas, which are in fact instantiations of your parameters.

**JOHN W. ROBERTS:** That is a very good point.

**STUDENT:** Are you still having that problem regardless of which two you use?

**JOHN W. ROBERTS:** What do you mean, which two? Oh, whether you're using back prop or this or something like that?

**STUDENT:** Well, no. Between the stochastic policy and the stochastic gradient descent.

**JOHN W. ROBERTS:** OK. I really want to emphasize something right now. They're not different. OK? They're not, they're the same thing. All right? You're correct that your parameterization is going to affect things. Because you're going to get to a minimum, with respect to your parameterization.

**STUDENT:** Right.

**JOHN W. ROBERTS:** You're going to be sampling over your parameters. And so, the cost function is going to be a function of your parameters. So if you imagine, if you have different parameters, the cost function will look completely different. Does that makes sense? If I were to parameterize with an open loop tape or feedback policy, how would the value function dependent on those parameters be completely different? So it's very important. That's true. It's over these instantiated parameters. So picking that affects everything. And picking a sensible one can be a hard problem.

**STUDENT:** So there is almost like another, instead of just having the sigma and the rate, you also have to have essentially these alphas and [INAUDIBLE].

**JOHN W. ROBERTS:** You have it for both, right?

**STUDENT:** Right. Exactly. It's like--

**JOHN W. ROBERTS:** Yeah.

**STUDENT:** It's almost you're over-parameterizing everything just so you can get an answer.

**JOHN W. ROBERTS:** Well, I mean you need to parameterize your policy either way. I mean, whether you choose to parameterize it as an open loop tape where you have 250 parameters or something like that, you've still chosen parameterization. I mean, you still picked a lot of them. But you need to represent it some way. And so, I mean, whether you do back prop or anything, you're still going to have that exact same problem. I mean, you could do back prop and make it more efficient if you had five parameters, too. But it also can handle these big open loop tapes and stuff like that. So there's less of a pressure to find concise and sort of compact representations.

But yeah. But I want to emphasize that these are different ways of looking at it. But update's the same. The algorithm's the same. The only difference is, am I following this local true gradient? Am I following the expected value of my cost?

So a toy example here to show some of the practical issues of some of these things. And please, if you have any more questions about any of these things, even if it seems tangential, just let me know. OK. So here, we have what we just talked about implemented. You can see how simple it is right here. I'm doing a four loop instead of that while converged. But we have a loop through some number of iterations. I get my sigma and eta.

So this is so I can make it dependent on the number of iterations or something like that. So I can make them decrease. I generate the noise according to a Gaussian distribution here. I get my reward. And the rest of this is just sort of plotting stuff and everything like that. And then here. Here's my update. And that's where I've got my baseline. Right? So you can see that this is just sort of that pseudocode right over there. There's nothing more. All these functions are just so that I can switch between different things really quickly. But it's very easy to implement.

**STUDENT:** So you do the baseline. And then the equations, [INAUDIBLE]?

**JOHN W. ROBERTS:** Right. I mean, this is what pops out when you look at that expectation. But as we said, the expected value of E is 0. So if I have a constant in there, my expected value if I subtract off that constant the same way, it's not going to affect the expected direction of my update at all. So these say you don't need a baseline. But a baseline has performance benefits. Right? So that's sort of another, if you go these different directions, this one says, oh you don't need a baseline. You can put it and it helps. The other one's more natural to think about starting with a baseline. Then you can say, oh well, you actually don't need it. So again, they both have the same effect there. Maybe I should have been more clear. But yeah.

So the problem that we're looking at is simply, we have a spline. So I make a spline. And then we're trying to figure out a curve that gets as close to that spline as possible. So it's really just sort of we're trying to fit a curve to this random spline. Right? Now, obviously we don't need to do anything fancy for this. But it's very visual. And so, I think it's probably a good example here. So we'll start. The cost function we're using is the square distance between our sort of guessed curve and the actual curve. And try running it.

All right. So on the left, this is our cost. Well, we have it formulated as our reward here. So it's negative. And then over here, the blue thing is the thing we want. The red is our nominal. And the green is our noise. You see our noise is pretty small. I started it at L zero. You can see it climbing here. Right? And it is getting better. It's bouncing around a bit. Yeah. And so, this is actually parameterized, this spline. So it could be exactly correct if we let it. I mean, it's capable of representing the curve exactly. So the cost could be zero, effectively. Right?

I think actually, if I have run it, it gets stuck in a local minimum, which is a good cautionary tale. Even if a problem is simple like this, you can still get stuck in local minima. So always be careful for that. But see, it's actually doing a reasonably good job. Right? Now you might be like, oh well, it has an ideal parameterization. It's the minimum number of parameters you need to have a spline that can represent it. And this was the same form of spline. So that's a pretty solid parameterization.

But we can try a different one. This computer is showing its age. When I ran these tests on my desktop, it went pfft. And so, I had my number of iterations, like several thousand. And then when I realized it was going to do this, I was like oh. All right. So here. What parameterization do you want? Linear, tripolated, nearest neighbor? A polynomial?

All right. OK. [INAUDIBLE]. There we go. Sorry about this. It's not done. Here we go. Here, you see it's capable of learning. It has this is obviously inferior parameterization. It can't represent the right answer. But you can see it is getting closer here. It's not doing a bad job. Right?

And the question of what the optimum is here is not going to look just like the curve because it has to sort of balance these competing things of going over and going under. So this is a good example of where your policy parameterization doesn't capture the true optimum. And so, when this thing converges to something, it maybe is the best. But it's only the best with respect to that way of parameterizing a policy. Right? So if you guessed that nearest neighbor is rich enough, then you put it in here. You're going to see that it's not optimal. Right? That you've limited yourself.

And sometimes I mean, the other one with the spline, you put yourself in the regime where you have a very good parameterization. You get very close to the right answer. Here, we have poor parameterization. And so, it's still going to learn. It's still going to converge. But it's not going to be as good. So lin interp, we could probably do better. But here. Now, we'll go back to-- well, we'll keep it in nearest neighbor for now. You can see lin interp.

All right. Now I said previously that this is partially nice because it's robust to noise. So here we're going to look at, we're going to add Gaussian noise with a standard distribution of 20. Let's see. Here, it's not doing well. I probably have to reduce my sigma eta. Let's see if we can get back to where it started. Yeah.

**STUDENT:** I think that's the same thing as [INAUDIBLE].

**JOHN W. ROBERTS:** You multiply by sigma. The variance is squared. So this is not doing a great job. You see noise can break it. If we start with a bigger error though, it's effectively standard deviation. Also maybe our eta could have been too big. Because we're getting big measurements just from this error. So like a big eta, let's try reducing our eta.

So it didn't get worse. Learning is going to be slow though, obviously. Because it has such a small signal. But at least we prevented it from doing that terrible thing. That's because it was getting these giant measurements of error and updating based on them. But an expected value will still go where you want it to go. If you did a bunch of updates, you'd be fine. But that just gives you everything out.

If we reduce that down to, let's say, learning will be really slow. So let me turn off plotting. I'm sorry about this. I don't know what just happened. OK. So let's see how this does. Ah, this is smoothed. I ran a boxcar filter on it. But it's not necessarily learning quickly. But with that noise, it still learns. Now if we're going to be doing this, we might as well kick ourselves back to 20 and see if we can do anything.

Yeah. So there, it's got enough noise that it's sort of been walking a bit much. Maybe. If you make your eta smaller and stuff like that, maybe make your sigma bigger so it gets a bigger signal when it measures the change, you might be able to be OK. But I mean, it's still suffered from these things definitely. Now, another point here. I talked about cost functions a bit yesterday. Here, we have one where it's the square distance. Let's say we-- I think I'm going to give away whether it's going to do better or worse by having named it poor grad. But this one, we're going to get a reward of one for each point that's within of 0.05 of the desired point. So if you're far away at zero and if you're close enough, it's one. Right?

**STUDENT:** [INAUDIBLE].

**JOHN W. ROBERTS:** Right. Where did my-- Oh. So you see? This is a much shallower performance. It still looks like it's learning. And it is. Because I mean, even the bad cost functions can learn. But now I'm going to show you what that looks like when you run it and actually look at the plotting. You see? Now we can give it our-- we're still giving it that linear interp. Let's see if we give it the better parameterization and see how that does. You can see it's not learning nearly as quickly as the other cost function did. Right? Let's get all our things back to make eta a bit bigger.

See? So here the only thing that changed between our runs that looked pretty nice and this run right now is the cost function. You see, it is over the course of-- that's a 300 something, it is getting up there-- but if you remember the other one, it's fitting a lot nicer. Right? And the thing is that, if you look at probably the cost of the other one, even using this cost function, once you solve it, the other one would be lower as well. Do you see what I mean?

The problem that, if you solved it with the other one then evaluated this cost function on the solution, that would be better than this after some number of parameters. And it's because learning based on this is hard. Right? You can see that this is very coarse. It's very easy just to lose one or gain one if you don't have good gradient information. And so, this is a bad cost function for learning. Right?

The same task, if this were the task you really cared about, formulating it with the other cost function and solving it, you actually probably do better than solving with this directly. Right? And that's because the gradients in this one are poor. It's not easy for it to define the direction it's supposed to move in. And it's even worse. I was talking before about how you could be in regions where you're getting zero cost or zero reward for a while or there's no gradient at all.

So if we start ourselves with an error here, it's not getting any measurement. You see? Again, it's not changing at all because it can't measure anything. So if we go to-- I said the way to solve that, if you were stuck with it, could be to use more violent kind of sampling. And there, you see we actually are able to get some reward. And it takes a while. But you see that there's more violent sampling at least getting out of that region where we don't get information. Right? Still not the best. But if you're stuck with that problem, that's one way to deal with it.

But you can see that, with the other one where it doesn't have any regions like that-- if you go back to our nice grad-- sorry, I need to switch my eta. Yeah. Whoops. We can go back to-- See how much nicer that is? So there you go. So here you can see the parameterization can affect it a lot. The cost function can affect it a lot. The eta and sigma can affect it a lot. There's a lot of concerns. Right?

And that's why that should make SNOPT and stuff more appealing to you. Because if you can complete those gradients using back prop, there's no guesswork in that. Right? You give it to SNOPT, there's no parameters to set. I mean, I guess there's some convergence criteria and stuff. But you don't have to play these games. There's a lot of ways to get this wrong. Right? So--

**STUDENT:** So the reason that we introduce the [INAUDIBLE] policy was a way to explore [INAUDIBLE]?

**JOHN W. ROBERTS:** For here, I mean they provide you the exploration. Yeah. There's certain situations-- we talk about gambling and stuff like that-- where stochasticity, if there's an adversarial kind of component, stochasticity could be necessary for optimality sometimes. Right? But in these kind of cases where we have a dynamical system, I don't see how stochasticity in a deterministic system could make you more optimal than deterministic. Right? That's because the system is deterministic and not trying to guess what you're doing and stuff. So that's pretty nice convergence right there.

And you see, if we evaluated this solution with our cost function of one of the poor gradients, I mean this is going to be a lot nicer. Right? And that's even though we wouldn't solve it explicitly. It's just because the gradients of this one are a lot nicer. So you can solve it a lot nicer. So remember that, that cost functions-- even if it's not explicitly what you maybe most care about-- for example, we care about catching the perch. Right? But if you want to catch the perch, it could be that, by learning with something with a distance in it, maybe we'd learn better and actually end up catching the perch more than if our reward was just one that could catch the perch. Right?

So cost function design can matter a lot. Policy parameterization matters a lot. So hopefully you all feel really comfortable using stochastic gradient descent now in the situations where it's warranted.

**STUDENT:** Can you talk a little bit about the baseline? How do you get the baseline?

**JOHN W. ROBERTS:** Yeah. My baseline here. I think I have two limitations, let's see something we can do. So my baseline here, right now I'm using a second evaluation. Right? We can do an average baseline. I screwed something up.

Yeah. I probably-- so that's gone crazy. But we'll try a smaller eta. There. So this is an average baseline. See, it's still improving. It's not doing as nice. And it's also because we have to turn down the eta and stuff. I don't know that error was. I'm not going to try to debug it right now. But you can see it's still learning.

Now the real question is, what if we have no baseline? Right? I made a really smart eta there because now it's not getting rid of anything. Yeah. You don't really have to turn it down even more. But that's a good example of why a good baseline is so critical. Right? I mean, you can see that all we did was change how the baseline, the average baseline was working fine. And it didn't cost us any more. You have two extra evaluations that we just got rid of. We put in that. And you see how much it struggled there.

So I mean, if you just do an average baseline, you get it for free. And it's still a huge advantage over what we're trying to do without any baseline at all.

**STUDENT:** But we did-- I mean, it will eventually work it out. The baseline would just be a lot slower?

**JOHN W. ROBERTS:** Yeah. I mean, it should. I mean, the thing is that, in practice, it can randomly walk around and get stuck in places. Like right here, you see this is that. It's a big run of improvement there. And it did better than it originally did. You see it's just walking all around. And so, yes. I mean there's still this bias towards improvement. And it will work. And if you were to calculate a whole bunch of delta alphas and then add them all together, then you probably wouldn't walk in the wrong direction as often.

But here's the thing. You have to remember that it's sort of always moving in the direction. Right? It's just it's moving more or less, depending on how good it is. And so, yes. This should still work. But in practice, a good baseline helps a lot, as you've seen. So hopefully, all those examples--

**STUDENT:** Is it also the case that that baseline will not hurt you?

**JOHN W. ROBERTS:** Well, how bad? I mean, this could be considered bad. Right? A zero baseline did that.

[INTERPOSING VOICES]

**STUDENT:** Or can you actually have a baseline that gives you worse results than no baseline?

**JOHN W. ROBERTS:** You probably could. Because you could just have it be a bigger error. Right? Just more in the wrong direction.

**STUDENT:** Right.

**JOHN W. ROBERTS:** So Yeah. But I mean that's what I'm saying. If you average over the past several trials like we did and then reduce the rate of your learning so that those averages are relatively representative and you don't move too much on your own--

**STUDENT:** [INAUDIBLE].

**JOHN W. ROBERTS:** Yeah. But I'm saying yeah. I mean, an average baseline is free. There's no extra policy evaluations. And it still helps a lot. So an average baseline makes sense. And I mean, in something like this where it's so cheap, if you want those big jumps and stuff like that, the average baseline is if the big jump can struggle. Because the cost is so different when you've updated. Right? So your average is going to make a lot of sense.

But so they're sometimes going to be better rather than reducing your updates. Like when we're really far away from the right policy and we have a long way to move and we could move in all sorts of ways, that'd still be good. The average baseline, if we turn everything down to the average baseline, and it still works nicely, that probably wouldn't be as fast as doing two policy evaluations and doing big jumps. Right? Where every time, you just re-evaluate it.

**STUDENT:** It just resets it.

**JOHN W. ROBERTS:** And so, yeah. To move it smoothly with an average baseline, we may need more than twice as many iterations as the one does when you do an ideal baseline. That's something that I've seen in practices. When you're doing this big updates, a real evaluation can be fewer evaluations in the end. So anything else or anything else you'd like to see run on the spline fitting problem right here?

OK. No one has any questions? We're all good? I'm going to tell Russ that you understand stochastic gradient descent perfectly. And it's going to be extremely useful in your research in the problems to which it is well suited. All right. Great.