NPTEL

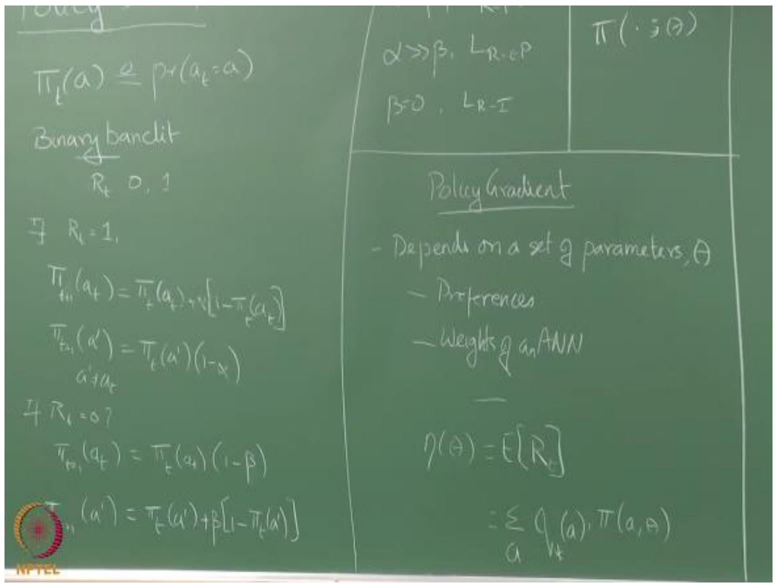
NPTEL ONLINE CERTIFICATION COURSE

REINFORCEMENT LEARNING

REINFORCE

Prof. Balaraman Ravindran
Department of Computer Science and Engineering
Indian Institute of Technology Madras

(Refer Slide Time: 00:17)



$\pi_t(a) \triangleq Pr(a_t = a)$

Binary bandit

R_t 0, 1

If $R_t = 1$,

$\pi_{t+1}(a_t) = \pi_t(a_t) + \alpha[1 - \pi_t(a_t)]$

$\pi_{t+1}(a') = \pi_t(a')(1-\alpha)$, $a' \neq a_t$

If $R_t = 0$?

$\pi_{t+1}(a_t) = \pi_t(a_t)(1-\beta)$

$\pi_{t+1}(a') = \pi_t(a') + \beta[1 - \pi_t(a')]$

$\alpha \gg \beta$, $L_{R-\epsilon P}$

$\beta = 0$, L_{R-I}

$\pi(\cdot\,;\theta)$

Policy Gradient

- Depends on a set of parameters, θ
  - Preferences
  - Weights of an ANN

$\eta(\theta) = E[R_t]$

$= \sum_a q_*(a) \cdot \pi(a, \theta)$

Let us denote the set of parameters by θ so I am going to come up with some kind of a performance measure for a specific choice of θ, so I will call it η(θ) right so for what does the specific choice of θ correspond to, suppose my θ is like composed of θ_1, θ_2 … θ_K so I give you values for θ_1, θ_2 … θ_K so what does this correspond to? ONE policy when I say I am looking at a specific value for θ it corresponds to one policy.

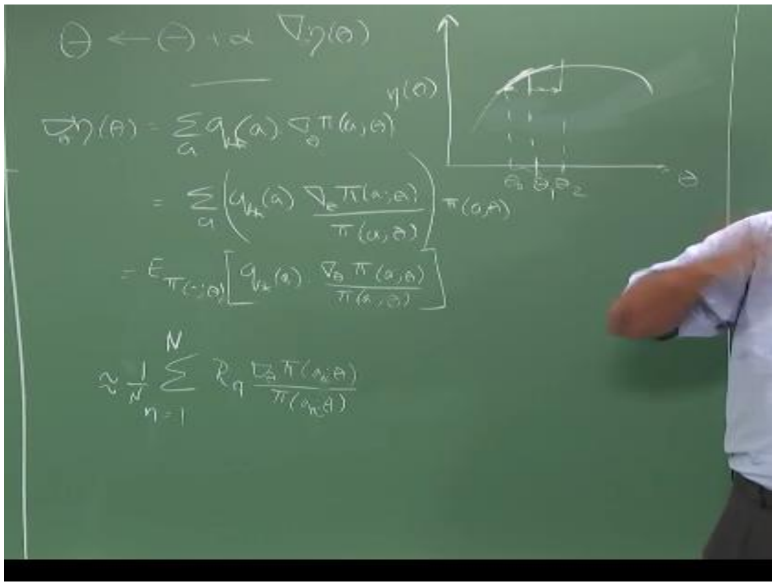
$$\begin{aligned}\eta(\theta) &= E[R_t] \\ &= \sum_a q_*(a)\pi(a;\theta)\end{aligned}$$

So when I say $\eta(\theta)$ that means I am evaluating that policy somehow okay so what is the most natural evaluation measure you can think of for a policy, expected pay off, right expected the expected payoff is the evaluation for the policy so yes I am going to do I am going to do that there okay, so what is the expected payoff you know what the expected payoff is, this is the expected payoff I will get for taking action a.

This is a probability that I will take action a when I am selecting actions according to θ right right so this is a standard notation I do not know if people are familiar with this, this means π is a function that is parameterized by θ right and whatever specific value I give for θ will define what that π is okay and then this means it is going to take some other argument okay, so this is the input to the function okay.

Which is defined by these parameters θ okay so this is how I, when you put a ; here that means it is parameterized form okay, so essentially since $\pi(a;\theta)$ means that is the policy defined by the current settings of parameters, θ times the $q_*(a)$ right because $q_*(a)$ is the true expectation for taking arm a right and then sum over all a, I will basically get the expected reward but if I am having a deterministic policy that means essentially only one arm I will take all the other arms of probability 0 then the payoff will be q_* of that arm right okay, so now once I have this kind of a performance measure so what do I do is essentially I take θ.

(Refer Slide Time: 03:16)



$$\theta \leftarrow \theta + \alpha \nabla \eta(\theta)$$

$$\nabla_\theta \eta(\theta) = \sum_a q_{k}(a) \nabla_\theta \pi(a,\theta)$$

$$= \sum_a \left(q_{k}(a) \frac{\nabla_\theta \pi(a,\theta)}{\pi(a,\theta)} \right) \pi(a,\theta)$$

$$= E_{\pi(\cdot,\theta)} \left[q_{k}(a) \frac{\nabla_\theta \pi(a,\theta)}{\pi(a,\theta)} \right]$$

$$\approx \frac{1}{N} \sum_{n=1}^{N} R_n \frac{\nabla_\theta \pi(a_n,\theta)}{\pi(a_n,\theta)}$$

$$\theta = \theta + \alpha\nabla\eta(\theta)$$

Right I take θ, I find the gradient of the performance okay so where the gradient is taken with respect to θ right so I look at how the performance changes with changing θ and I will try to move θ in the direction of the increase in the performance, okay. So this is called gradient ascent so people might have heard about gradient descent earlier so this is gradient descent, so I am going up the gradient, right.

If the performance improves if I change θ in some direction then I will change my θ in that direction, okay does that make sense people see this or do you need more pictorial analogies, so let us say that I have one parameter θ, I have one parameter θ right and that is $\eta(\theta)$, let us say this is my is this a good way to do it, right let us say that I have my performance whatever is an acceptable range of θ so my performance goes like this right.

So I initially start off with some guess for θ right that will be my initial guess for θ right then what I do, I measure this and then I compute the gradient right so the gradient is there so if I look at which direction I should move my θ, essentially I should move in this direction so I move a little bit and set that as θ_1 okay then I measure the gradient again okay still I have to move in this direction more so I keep doing that and I think I will declare that I have reached the highest performance there.

So this is essentially how I do it okay there are a couple of questions which you might ask me here, why do I have to do this in this kind of an incremental fashion why cannot I just take the derivative of that and find the highest point and settle down there, sorry I do not have the function at all so forget about closed form, I do not have the function at all what is that function that is $q_*(a) \times \pi$ right I know π but I do not know q_*

If I know q_* my problem is solved right, I do not need to do any of these things, that function itself I do not know right. So how am I evaluating this function and trying to find the gradient, by sampling so I am pulling a lot of arms right according to my current policy π right and then I am trying to evaluate the policy π and then trying to find the gradient at that point right, so because everything is done through sampling.

So whatever gradient I compute at this point will not be the correct gradient right this is too simple I mean it has to be either that direction or this direction, this is too simple right but think of a multi dimensional case when there are many many parameters, the right direction you will have to compute can get messed up right so instead of going in all the parameters instead of identifying the right direction some parameters you might get the right direction some parameters you will get the wrong direction and so on so forth right.

Therefore every time you make only small steps because if we tried if we say oh I have computed the direction of the gradient I will just keep moving in that direction until my.. or I will take a huge step in the direction until my performance drops or something that would not work because the direction itself might be wrong so in this case I mean we have always moved in the right direction every step.

It is possible that because you are estimating the gradient you might actually make the wrong computation and move in the wrong direction sometimes okay so what really you can hope for from these kinds of methods which are called stochastic gradient approaches okay they are called stochastic gradient approaches because you do not know the true gradient and every time you make an estimate of what the true gradient will be and then you are using that for making moves.

I mean the most popular form of this will be called SGD which is stochastic gradient descent okay it is a very popular optimization technique nowadays but in this case you are using stochastic gradient ascent but still it is the same stochastic gradient idea and so you have to be careful about how you use it and the best that you can hope to get is that in an expected sense over many many updates right.

I did I did now I showed you two updates so if you make this update says this kind of updates many times in an expected sense you will move in the right direction of the gradient right so that is the kind of guarantee that typically you expect for these kinds of methods okay, so is it clear now is clearer now if people are not hundred percent clear about what I am talking just stop and ask right I can elaborate at the risk of slowing down yeah.

Good point yeah so we will come to that so we are going to talk about that, so we do not know the function itself right how are you going to estimate the gradient right, so we will come to that in a minute so we know we have a functional form for this now I am going to write down the gradient for this expression right this is the expression we have, I am going to write down the gradient for this expression.

$$\nabla_\theta \eta(\theta) = \sum_a q_*(a) \nabla_\theta \pi(a;\theta)$$

$$= \sum_a q_*(a) \frac{\nabla_\theta \pi(a;\theta)}{\pi(a;\theta)} \pi(a;\theta)$$

$$= E_{\pi(;\theta)}\left[q_*(a) \frac{\nabla_\theta \pi(a;\theta)}{\pi(a;\theta)} \right]$$

$$\approx \frac{1}{N}\sum_{n=1}^{N} R_t \frac{\nabla_\theta \pi(a_n;\theta)}{\pi(a_n;\theta)}$$

This is a batch mode update.

Okay so this is assuming that $q_*(a)$ because it is a parameter of the problem right it has nothing to do with the policy right as far as your policy parameterization is concerned q_* is a constant right because it is a it is something to do with the problem so I can I do not have to do this I just have to differentiate this with respect to theta right, so now I am going to do some hand waving. So what did I do here, I multiplied and divided by π, okay.

So for this to work what do I need, π is non zero for all a so this is one condition that you need for all of this any kind of π that you assume okay has to be nonzero for all a it has to assume some nonzero probability, now if you think about this particular expression right I will put a bracket around that right so what does this remind you off, does it look like an expectation computation I am looking at the expected value of some function right.

Where I am taking samples according to probability π, right this looks like an expected value right, I am summing over all possible outcomes of a all possible values here right that is the probability so I am essentially taking the expectation taken according to π of… is it fine right. So everybody on board with that right so that I can write that expectation oh okay sorry about the font, so the expectation is taken with respect to π, okay.

Is it clear? so now we know how to make an estimate of the expectations right how do you do that we draw sample and then take an average right so we can do that so essentially what I am going to do is pull an arm that is a sample right when you say draw a sample according to π that essentially means I have to pull an arm and what happens when I pull an arm, I get a reward right but the reward does not figure in here.

The expectation of the reward directly figures in here right so I can actually write this as a double expectation you know mean the expectation is taken over not only π but also over the process that generates the reward q_* right it is also the process that generates the reward so my sample here is going to consist of R_t right is it clear I mean this is a very subtle thing, people are here with me on this is it clear.

See q_* itself is already an expectation right so instead of because I do not know this expectation I will have to estimate this expectation as well and that can be done by just taking the R_t so what I will do now is at every time t I will sample an arm, I will pull the arm figure out what the reward is right but is that sufficient, no I have to do this right I just cannot use the R_t directly I will have to use this also right.

So I will take the gradient of π, evaluated at a divided by the.. so this is my one sample right this is because the expectation I am taking is of this quantity so the sample that I draw is actually this right and I have to do this over I do not know some number n, $\pi(a_t; \theta)$ you are right, is it fine , no no no I am not using t I am sorry, I promised you that I will use n for discrete time and t for continuous time.

So we are talking about discrete times here it has to be n I do not know if i used t here, sorry about that but all of you a erase in your notebook that t and write down n, for this part of, sorry about that right so this is clear is there anything missing right, so I need to have 1/n there so this gives me the gradient right so this is fine oh are we still okay, we are not because this is actually a estimation okay.

So I cannot put equal to there, it is only an approximation great so can we compute this, we can right because we know pi we are the ones who determine what π should be right because at the beginning I choose a parameterized representation that I can choose whatever suppose it is softmax well I know how to take the derivative of soft max right or if it is a continuous action I can just use a say a Gaussian right I know how to take the derivative of a Gaussian with respect to its parameters right.

Or if it is a multinomial what do I do you know how to take the derivative of a multinomial as well right so the derivative is very simple so I know what form of the function I choose it so I can go ahead and take the derivative, there are a couple of reasons for it so estimating q_* need not always be the right thing to do okay because I have not introduced all the complications I am hesitating whether I should go there or not.

But the thing is so the q_* function sometimes turns out to be incredibly complex okay as a function of the state space okay not necessarily here right as a function of the state space q turns out to be incredibly complex whereas a direct representation of the policy turns out to be a lot more simple right for example there are many inventory control problems where the policy essentially turns out to be if the inventory is below a certain level you buy, if the inventory is above a certain level you do not buy okay.

But then the value function itself becomes a very complicated function of how many of the items are left of different types and so on so forth so the function itself becomes a lot harder to learn so there are instances like this where this is easier to learn right and there are a couple of other cases especially when you are talking about very large problems for example we will see that policy gradient approaches generalize very easily to problems with continuous actions.

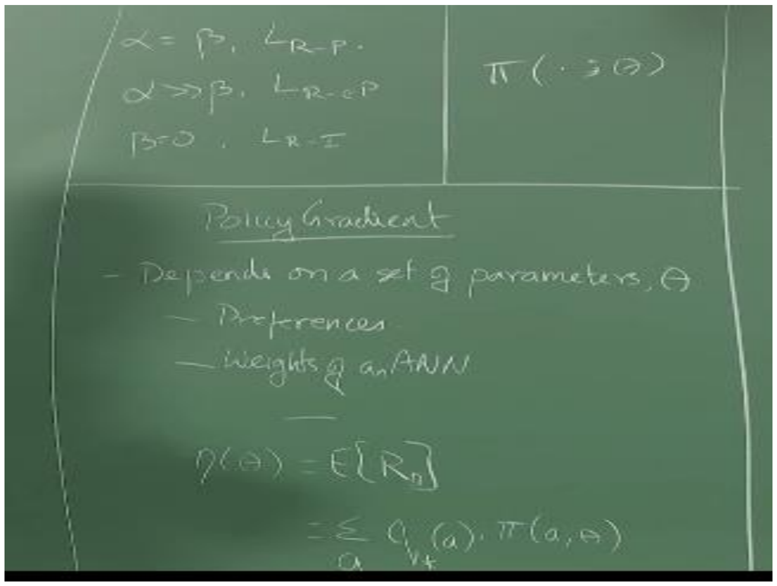
So where there are individual actions so if you are talking about value function methods right where I typically store a value for each action now I am going to have continuous actions it is not clear how to handle it there are ways of handling it but it turns out that they are not that well behaved as policy gradient approaches are when the action space is continuous right so there are many reasons why you want to consider continuous actions I mean policy gradient methods.

Primarily I would say the, one of the primary motivations is to handle continuous actions, okay. So this is some kind of a batch update, hmm because the expectation is taken with respect to the π, yeah you could do more simplifications of that if you want to right but what is your question again, yeah but I need to take this expectation with respect to the sample the probability distribution π right.

So and I cannot separate this I cannot write this out as some I really cannot simplify this further because my q_* is here right so this is the expectation of this whole quantity so I really cannot split this apart and say that okay I will just simplify this expectation and take only the expected value of $q_*(a)$, can I, I cannot take $E[AB] \neq$ AE[B] is it if A is only a constant right not otherwise I am not sure whether we can simplify this further right.

So what we are doing here is some kind of a batch mode so if you think of how we are doing this sampling so I am taking some theta fixing it pulling the arms multiple times right fixing θ pulling arms N times and then taking this average right and then what computing the gradient and then using that to change the parameters right this is one way of doing it another way of doing it to say that okay I will have a θ I will pull an arm I will compute the gradient. And I will change the parameters right so you can have a some kind of an incremental version.

(Refer Slide Time: 21:59)



$\alpha = \beta,\ L_{R-P}$

$\alpha \gg \beta,\ L_{R-\epsilon P}$

$\beta = 0,\ L_{R-I}$

$\pi(\cdot\,; \theta)$

Policy Gradient

- Depends on a set of parameters, θ
  - Preferences
  - Weights of an ANN

$$\eta(\theta) = E[R_n] = \sum_a q_*(a) \cdot \pi(a, \theta)$$

Incremental Version:

$$\theta_{n+1} = \theta_n + \Delta\theta_n$$

$$\Delta\theta_n = \alpha_n R_n \frac{\nabla\pi(a_n; \theta_n)}{\pi(a_n; \theta_n)}$$

$$\Delta\theta_n = \alpha_n R_n \frac{\partial ln\pi(a_n; \theta_n)}{\partial\theta_n}$$

$$\Delta\theta_n = \alpha_n (R_n - b_n) \frac{\partial ln\pi(a_n; \theta_n)}{\partial\theta_n}$$

Here b_n is called the reinforcement baseline.

So where I am saying at every step I will change my parameters by some fraction $\Delta\theta_n$, I will do this at every step right sorry and so this is what I am doing so what I am essentially will end up doing is $\theta = \theta + \Delta\theta$, θ_{n+1} will be $\theta_n + \Delta\theta_n$ right I will do this at every time, every time I pull an arm I will change this right I mean that is essentially the gradient of the ln it turns out this simplifies your life a lot this observation simplifies your life a lot when you are trying to actually solve the solve problems later right now you can immediately think of soft max becoming very easy to differentiate we have now taken log and there is lots of exponents there right.

So all the e power thing will vanish so it will become lot easier to differentiate for you right so what I am going to do now is to include another term here rather arbitrarily called b_n which we refer to as the reinforcement baseline, right and adding the b_n should not affect the whole process that is what the b_n should not really be a function of a, yeah theta basically so it should not be a function of the action I take right.

If it is a function of the action I take then it becomes dependent on θ so as long as it is not a function of the action I take I can keep adding a I can add this thing it is called the reinforcement baseline right so one way of thinking about it is I am when I get a reward right I do not know if the reward is a good reward or a bad reward so giving a baseline and saying that if you are above this baseline it is a good reward if you are below this baseline it is a bad reward okay.

kind of allows you to calibrate the rewards right one typical way in which this reinforcement base line is used is to just take the average of all the rewards I have obtained so far right this is all the rewards regardless of which arm I play I keep accumulating the rewards right now when I play a particular arm if it is above this average then it is a good arm if it is below this average then it is a bad arm right.

So valid after, yeah..hmm fine.. eventually it will eventually it will be all fine right so initially you might be making some mistakes because the initial few the b will be wrong right so you might actually think a good arm is bad and a bad arm is good I mean depending but eventually it will all work out does not matter right so if you think about if the reward is good right then I will go in the direction of the gradient right but if the reward is bad then I will go in the opposite direction of the gradient right.

So in some sense that is a little waving here that but then it is fine it turns out that adding the reinforcement baseline makes the convergence behavior of this algorithm a little more stable okay but it is not necessary you can if you are going to implement this you can do it without the reinforcement baseline also right so this is called the and this term is sometimes called the called the characteristic eligibility.

So what is the characteristic eligibility it essentially tells you, if you do that then you get into actor critic I will talk about that later, I will talk about that later yeah sure yeah you could do that right and that is essentially one of the motivations for getting into actor critic algorithms, so that requires you to maintain q_* estimates as well as the π estimates as well as the π representation.

So the π would be the critic and the q_* would be the actor,uff.. other way around the π would be the actor the q_* will be the critic so that is another class of algorithms right which address some of the drawbacks of policy gradient approaches right I will talk about it but somewhere halfway down the course not now so after this I am going to go back and start talking about the full RL problem and at some point I will come back and do actor critic yeah.

So his question was why do I use R_n right why cannot I plug-in q there right you could and yeah and typically when you do that it performs better the algorithm it is actually a it reduces variants significantly at the cost of adding significant bias right there are other issues you know you well you know what are the other issues going to actor critic but I will talk about it later right and going back.

So why is that called the characteristic eligibility so if you think about it so it essentially chooses which of the θs are most responsible for a change at that point right so that is the term that determines that is the term that is the θ dependent term in your updates right and it tells you which theta is more responsible for change that is happening here there suppose there are three or four different parameters that I have right.

So whichever has the highest gradient whichever direction you have the highest gradient that is the direction where the largest change is going to happen right, so it is essentially this telling you which θ is more eligible to receive the update that is why it is called the characteristic eligibility right did that make sense to people I think i have lost more people today than in the previous classes no great.

So this whole thing right this way of doing this incremental version of doing this update is actually called the called the reinforce algorithm, first proposed by this guy called Williams in 98, Williams proposed reinforce in I think in 1988 and in fact he proposed this in the context of neural networks right so he when he came up with this algorithm neural networks were still ruling right.

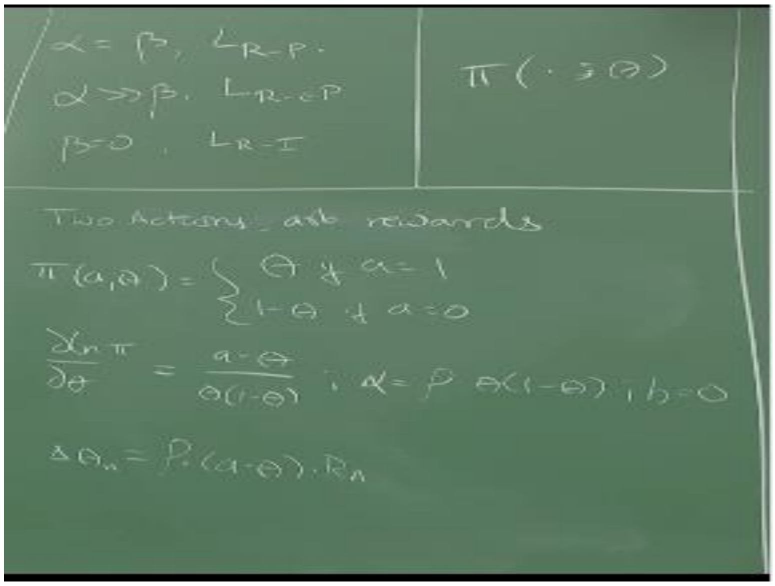
So he proposed this in the context of neural networks he said that your thetas are weights of a neural network right and then he came up with this update and then the biggest contribution he did was he said that hey this looks like a really crazy update right I mean how is this going to work and initially people are very skeptical he actually showed that in an expected sense right even though I am doing this update after just pulling one arm in an expected sense the gradient will be in the right direction right.

If I repeat this experiment multiple times and I watch how the weights evolve right they will actually move in the same direction as it would have happened if I had computed the correct gradient and then taken steps in that direction so essentially he established that for this reinforce update and it since then it has kind of been the one sole hope of convergence for all kinds of complex function approximations and RL okay.

So it works really well but it is extreme extremely slow, reinforce is extremely slow because the variance is very high so when I want to move on from this I will talk about the other problems of reinforce here is a horrible part about the paper so reinforce is actually an acronym there is an expansion each letter stands actually for a word he came up with a name for the algorithm which actually shortened to reinforce.

And then that started the trend for convoluted names in the RL community okay anyone knows what the expansion of reinforce is do you know what the expansion of reinforce is, forgot okay, I cannot for the life of me I refuse to memorize what reinforce stands for, great so let us do some special cases of reinforce right I want to consider a binary bandit.

(Refer Slide Time: 33:26)



$\alpha = \beta$, L_{R-P}.

$\alpha \gg \beta$. $L_{R-\epsilon P}$

$\beta = 0$. L_{R-I}

$\pi(\cdot\,;\theta)$

Two Actions arb. rewards

$$\pi(a,\theta) = \begin{cases} \theta & \text{if } a = 1 \\ 1-\theta & \text{if } a = 0 \end{cases}$$

$$\frac{\partial \ln \pi}{\partial \theta} = \frac{a-\theta}{\theta(1-\theta)} \; ; \; \alpha = \beta\, \theta(1-\theta) \; ; \; b = 0$$

$$\Delta\theta_n = \beta \cdot (a-\theta) \cdot R_n$$

Or let us say I take a bandit with two actions okay but they can have arbitrary rewards okay here is the parameterization I choose this is essentially the Bernoulli thing right so I have two actions with some probability θ I will select action 1, 1-θ I will select action 0 okay so what do I need to do now to get that update rule we have to find $\frac{\partial ln\pi}{\partial\theta}$ so what this will be, it will be 1 if a is 1, it will be minus 1 if correct.

No!! yeah look at ln π right it is not $\frac{\partial\pi}{\partial\theta}$, it is $\frac{\partial ln\pi}{\partial\theta}$ yeah so it will be 1/θ if a is one and it will be $-1/1-\theta$ if a is zero right, $\frac{\partial ln\pi}{\partial\theta}=\frac{a-\theta}{\theta(1-\theta)}$ we can write it like that, substitute a is 1, so $1-\theta$ and $1-\theta$ will get cancelled out, you will get 1/θ. Substitute a is zero, θ and θ will get cancelled out, you will get $-1/1-\theta$ okay a very compact way of writing it and now I am going to say that I will choose my α to be some $\rho\times\theta(1-\theta)$.

So it turns out that as long as your α is not dependent on the actual reward that you get okay it can be dependent on the θ this is what William showed as long is not dependent on the actual reward you are all fine okay you will converge so that is essentially this is result so I can make it dependent on my θ it should not be dependent on the actual action you sample okay and the actual reward that you receive so as long as that is there it is fine all right.

And I am going to choose my b to be 0, why am I making this specific choices so now I let me plug everything back in so what does my Δθ look like, so α_n which is $\rho\times\theta(1-\theta)\times(R_n-b_n)$ b_n which is 0 so it is R_n into ∂ln this which will be $\frac{a-\theta}{\theta(1-\theta)}$ right so what do I end up getting $\Delta\theta_n=\rho(a-\theta)R_n$ okay so what is the what is the interpretation for this, let us say I take action one okay and I get a reward of 1.

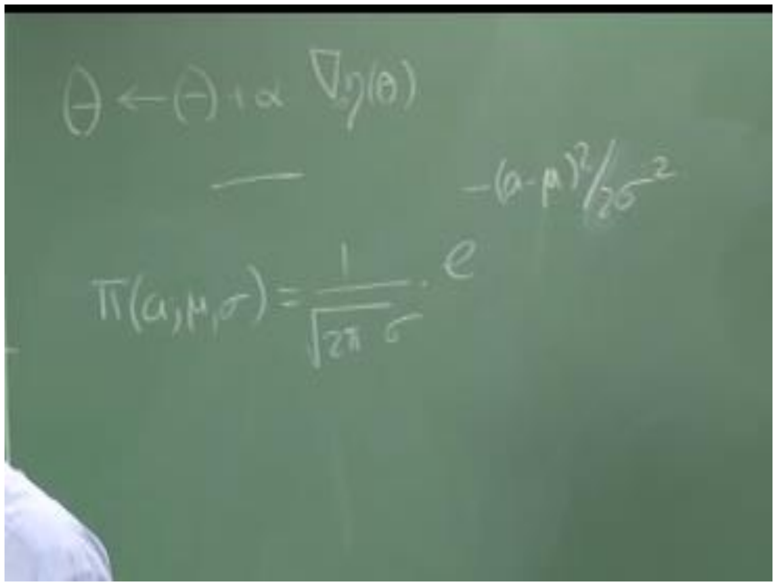
So what do I get, $\rho\times(1-\theta)$ that is how much I change my value by so which is essentially this right $\rho\times(1-\theta)$ right suppose I took action but what about I took action one right and my reward was one right what will I do for 0 oh there is no there is only one parameter right it is automatically my other parameter will get adjusted because it is $(1-\theta)$ suppose I took action one and got a reward of 0 what happens no change.

Suppose I took action of 0 and got a reward of 1 what happens, $-\theta R_n$ right so that is essentially what I have here right so this is $-\theta R_n$ right so θ is my $\pi_t(a_t)$ if you remember right and then this

R so that is essentially what I have, so this is this is what $L_{R\text{-}I}$ right so essentially $L_{R\text{-}I}$ is a is a gradient following algorithm actually it is a reinforce algorithm right so you can try to look at different choices for the alphas and the different choices for pi right and try to come up with this kind of update rules right.

We will do the soft max action selection as I described in the previous class right so do that right where I replace all my qs with some arbitrary θ suppose I have K actions or K arms then I will have θ_1 to θ_K these are my parameters and the probability of selecting arm i will be $\frac{e^{\theta_i/\beta}}{\sum_j e^{\theta_j/\beta}}$ right so this is my softmax expression so derive my reinforce update rules assuming that is the expression, right that is one I am going to give you another homework. The same thing, I said one of the nice things about doing reinforce.

(Refer Slide Time: 40:14)


$\theta \leftarrow \theta + \alpha \nabla_\theta \eta(\theta)$
$\pi(a, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-(a-\mu)^2/2\sigma^2}$

Is that it allows us to handle continuous actions right allows us to handle continuous actions, actions need not be discrete so far we have been looking at the cases where actions are discrete so I am going to use, that as my probability distribution that is my policy okay $\pi(a; \mu, \sigma) =$

$\frac{1}{\sqrt{2\pi}\sigma} e^{-(a-\mu)^2/2\sigma^2}$ give me the reinforce update rules so how many rules will I have for reinforce update rules here 2 - one for the μ, one for the σ right.

Get me the reinforce update rules and yeah so try to simplify as much as possible because when you do the ln μ, I mean the ln π derivatives will get all kinds of weird constants coming there so you can choose your α appropriately so that some of these constants get cancelled out so that I can have just like we did here right I chose my ρ to be $\theta \times (1-\theta)$ likewise you can choose your constants in this case also right the α s to be something different so that you end up with a good nice-looking form okay great.