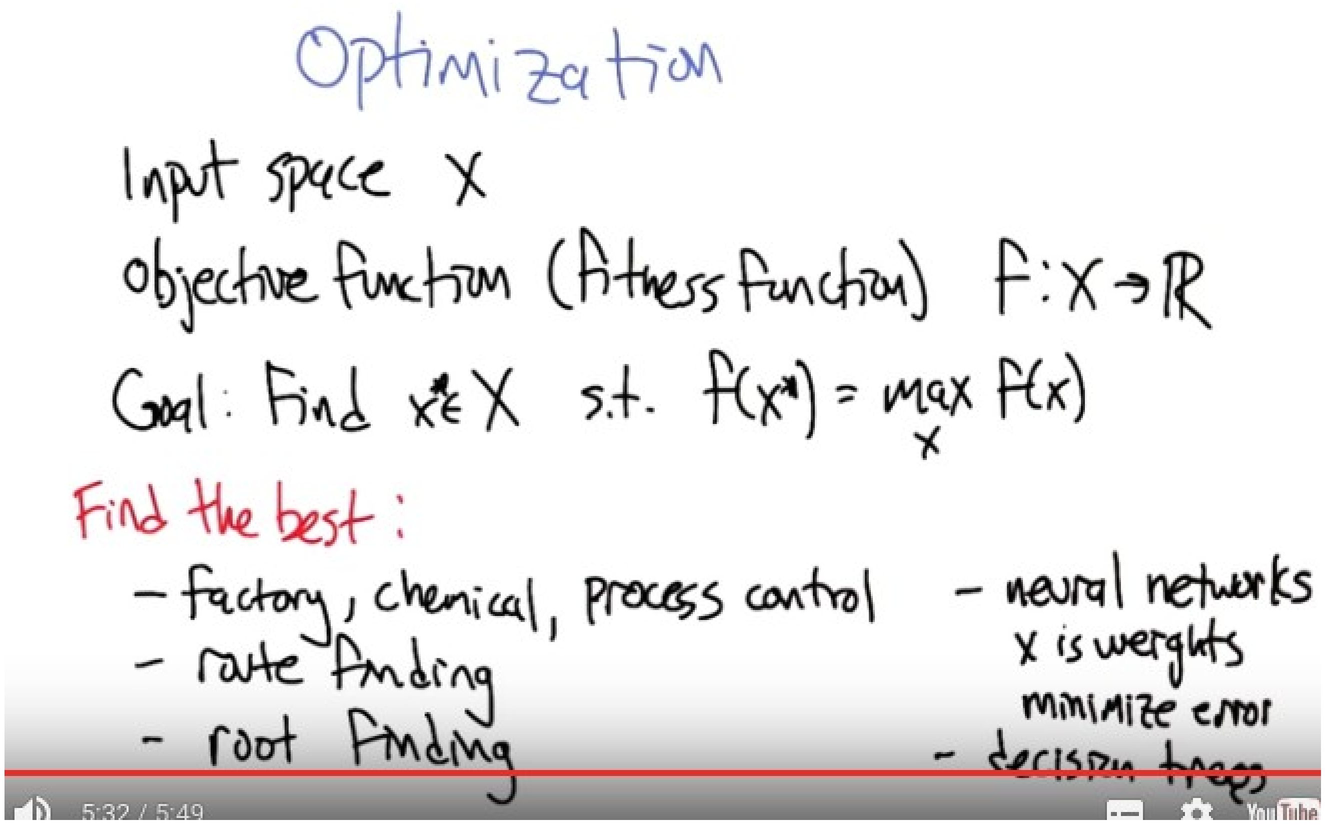
本文件的內容就是 找函數最大值 的幾個算法

UN-Supervised Learning

Mini course



Optimization

Input space X

objective function (fitness function) $f : X \to \mathbb{R}$

Goal: Find $x^* \in X$ s.t. $f(x^*) = \max_x f(x)$

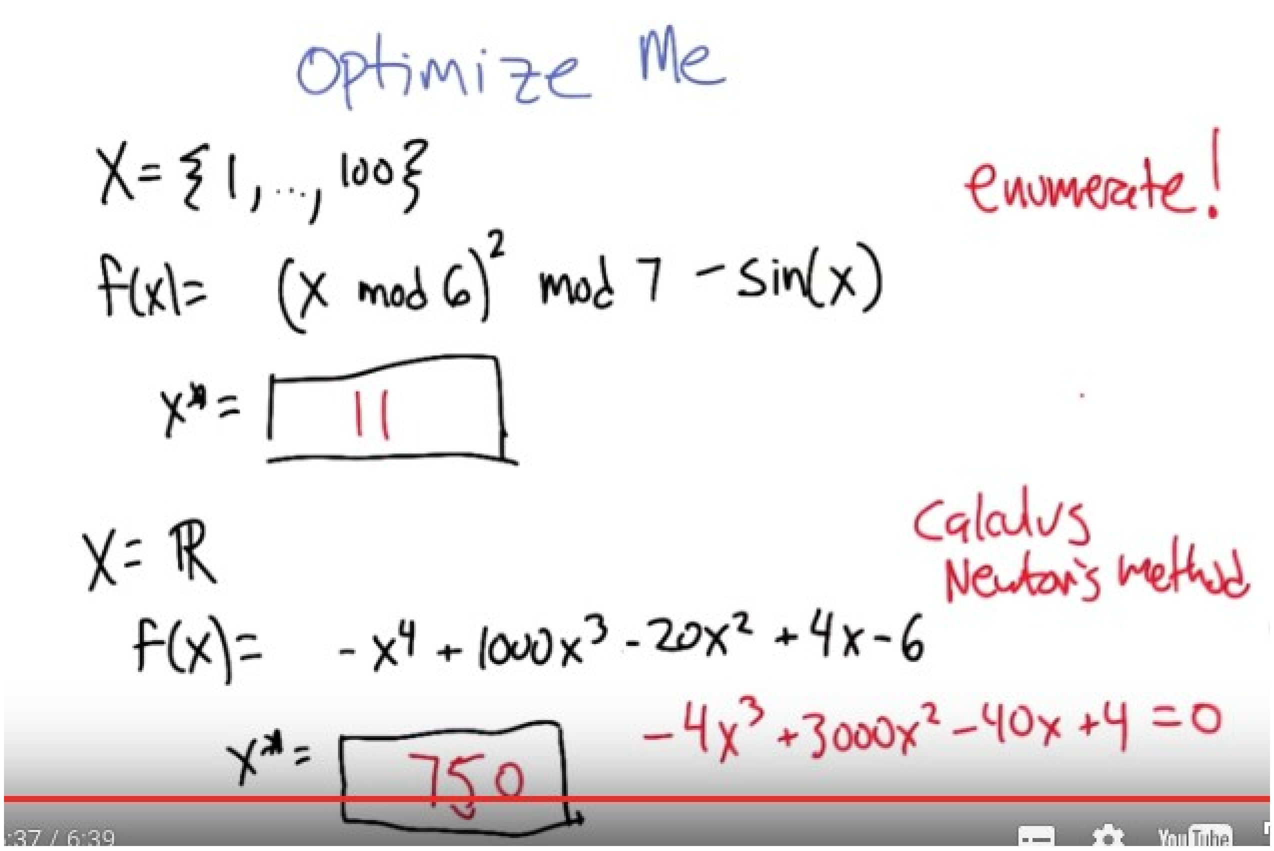
Find the best:

- factory, chemical, process control
- route finding
- root finding
- neural networks
  x is weights
  minimize error
- decision trees

1. Hi everybody, welcome to our second mini course in machine learning. This sections going to be on unsupervised learning. >> That sounds fun. Hi Michael. >> Oh, hey Charles, I didn't even know you were here. >> That's because I'm not really there. >> Oh. [LAUGH] I'm glad to hear your voice regardless. >> Good, I'm always happy to hear your voice Michael, and I'm looking forward to hearing about un-dash-supervised learning. >> Well, unsupervised learning's going to be a series of lectures that we do. But today's lecture Is on PZTTMNIIAOOI. >> Ok... >> Which is randomized optimization. I took the letters of "optimization" and I randomized them. >> [LAUGH] You're such a nerd, I love it! Okay. >> Alright so so the plan is to talk about optimization, and in particular to focus on some algorithms that use randomization to be more effective. But let's let's talk a little bit about optimization for a moment, because that's something that has come up, but we haven't really spent any time on it. So, what I'm thinking about, when I, when I talk about optimization, I imagine that there's some input space X, which is, you know kind of like in the machine learning setting and we've also, we're also given access to an objective function or a fitness function, f, that maps any of the inputs in the the input space to a real number, a score. Sometimes called fitness, sometimes called the objective, sometimes called score. It could be any number of things. But in the setting that we're talking about right now, the goal is to find some value, x*, I didn't mean to put space, such that the fitness value for that x* is equal to or maybe as close as possible to the maximum possible value. >> So that's like [INAUDIBLE] like we were doing with [INAUDIBLE]. Yeah, exactly, right. So of all the possible x's to choose, would we like to choose the one that, that causes the function value to be the highest. >> Okay. >> Yeah, I, I wrote it this way, though, because I thought it would probably be helpful if it's like, yeah, because we'd be pretty happy with the with an x* that isn't necessarily the best, but it's like near arc max, right? Something that's close to the best. >> Okay. >> So, so this is a really important sub-problem. It comes up very often, I was wondering if you could think of examples where that might be a good thing to do. Like in life. >> Well, Like in life. >> Computation, which is life. >> Computation is life. And life is computation. Well, believe it or not, I was actually just talking to someone the other day who's a chemical engineer and works at a chemical plant. And he says there's all these parameters they have to tune when they mix their little magical chemicals. And if they do it just right, they end up with something that, you know, is exactly the right temperature, comes out just right. If they're wrong at all, if some of their temperature is off a little bit or anything is wrong, then it ends up costing a lot of money and not coming out to be as good as they want it to be. So, you know, factories and chemicals and optimization and parameters. >> Factory, chemical. I'm not sure that's a general catergory of problem but I guess, I guess, maybe the one way to think about it is We'll call it process control. So if you've got a process that you're trying to put together. And you have some way of measuring how, how well it's going, like yield or cost, or something like that. Then you could imagine optimizing the, the process, itself, to try to improve your score. >> Okay. Yeah I think that's what it is. >> You know route finding is kind of like this as well. Right. So just. You know, find me the best way to get to Georgia. >> What about root finding? >> Oooh, I see what you did there. >> Mm mm. >> So you could think of root finding as being a kind of optimization as well. If you've got a function and you're trying figure out where it crosses the the origin. You might add you might set that up as a optimization problem. You might say well of all the different positions I could be in, I want to minimize the distance between the axis and the value at the point where I chose it. Find me the best one. Which, you know, is going to be right there. >> Actually, you know, when you put it like that. How about neural networks? >> Nice. So, that's, yeah let's get it back to the learning settings. So, what is, what do you mean by neural networks? >> Well, I mean, everything we've been talking about so far. X, you know is some kind of stand in for parameters of something. A process or I don't know whatever. So if the weights are just parameters of a neural network, then we want to find the x that I guess minimizes our error over some training set, or, or upcoming test sets or something. >> Yeah, so minimizing error is a kind of optimization. It's not a max in this case, but if we I guess, negate it you want to maximize the negative error, that it's maximized when the error is zero. >> Right. >> Cool, is any other learning related

topics that you can think of that, that have optimization in them? >> Well I would guess anything with parameters, where some parameters are better than others and you have some way of measuring how good they are, is an optimization problem. >> So, decision trees? >> Sure. >> So, so what's the parameters there? >> There the order of the nodes, the actually nodes, like what's the first node? Yeah, the whole structure of the tree. So it's not a continuous value like a weight, but it is a structure where we could try to optimize over the structure. There's nothing in the way that I set this up here that makes it so that X has to be continuous or anything like that. We just need a way of mapping inputs to scores. >> Okay. So all of machine learning. Everything we did in the first third of the class is optimization. >> There is some optimization in each of the pieces of it. Yeah, exactly. because often what we say, given this training set, find me a classifier that does wel on the training set and that's an optimization problem. >> Hmm.

2. Alright! So, let's I mean, what we'd really like to get to are algorithms for doing optimization, and I think to do that it's helpful to do a quiz. So, here is a little a quiz that I put together, I call it Optimize Me. And what I want you to do is, is here's two different problems, there's two different input spaces, two different functions, and I want you to find the optimal x-star or something really close to it, if you can't get it exactly right. So, the first question we've got the input space x is the values one to 100. And the function that we're trying to optimize is x mod 6 squared mod 7 minus the sin of x. >> Holy cow. >> Yeah, it's, it is an awesome looking function. Do you want to see it? I, I might be able to plot it for you. >> Sure, I'd love to see plots, because then maybe I'll know how to figure out the answer. >> Yeah oh yeah well maybe I should not then, maybe I should wait until we already have an answer and then yeah, let's, let's wait but I'll show you after it. It's, it's, it's crazy. >> [Laughs] >> Alright it's beautiful. So okay good, so and the second problem we've got x as the set of real numbers, so all, any, any possible number in the reals. And the function that we're trying to optimize is negative x to the 4th plus 1000x cubed minus 20x squared plus 4x minus 6. Alright, so you understand the questions? I understand the questions, yes. >> Good, good luck with them.



Optimize Me

$X = \{1, \ldots, 100\}$

enumerate!

$f(x) = (x \bmod 6)^2 \bmod 7 - \sin(x)$

$x^* = \boxed{11}$

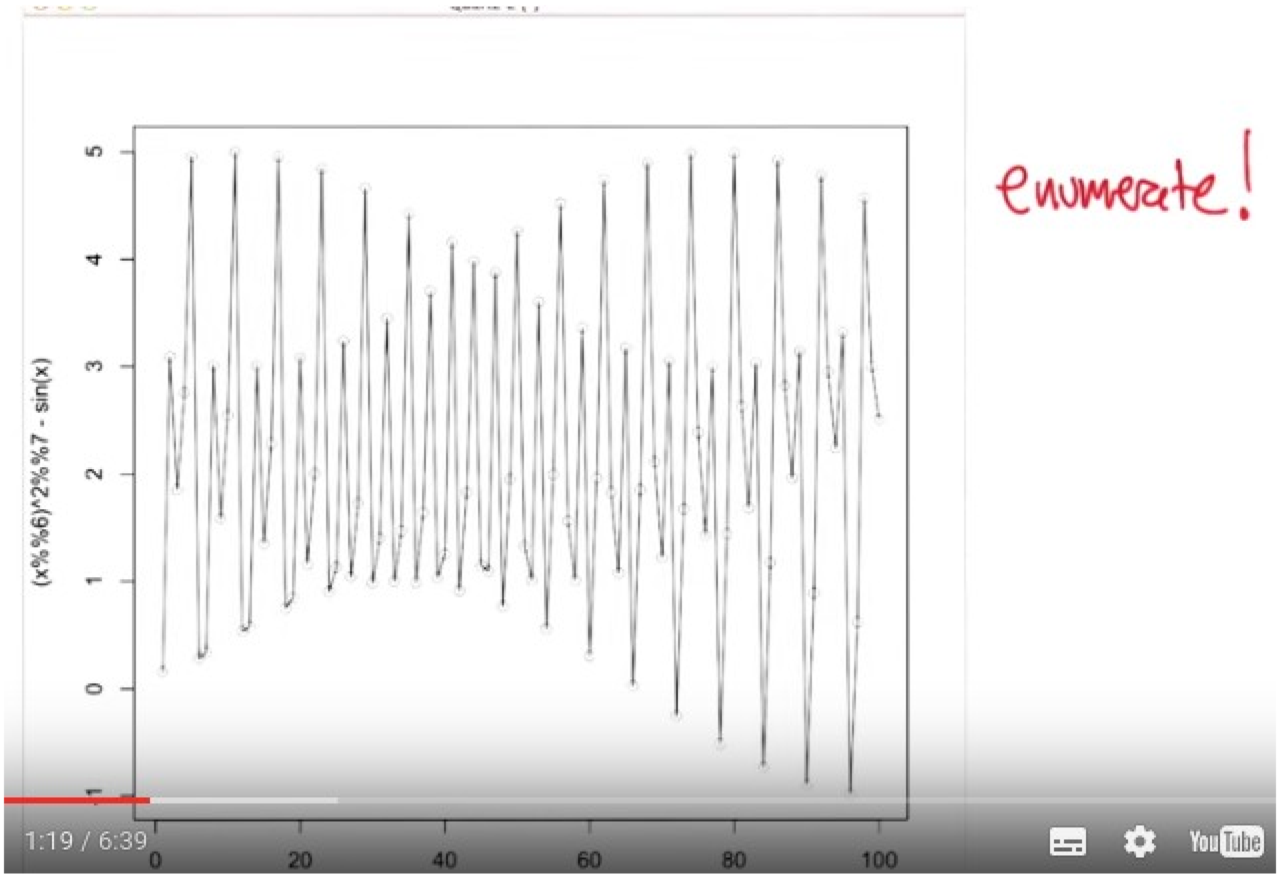
$X = \mathbb{R}$

Calculus
Newton's method

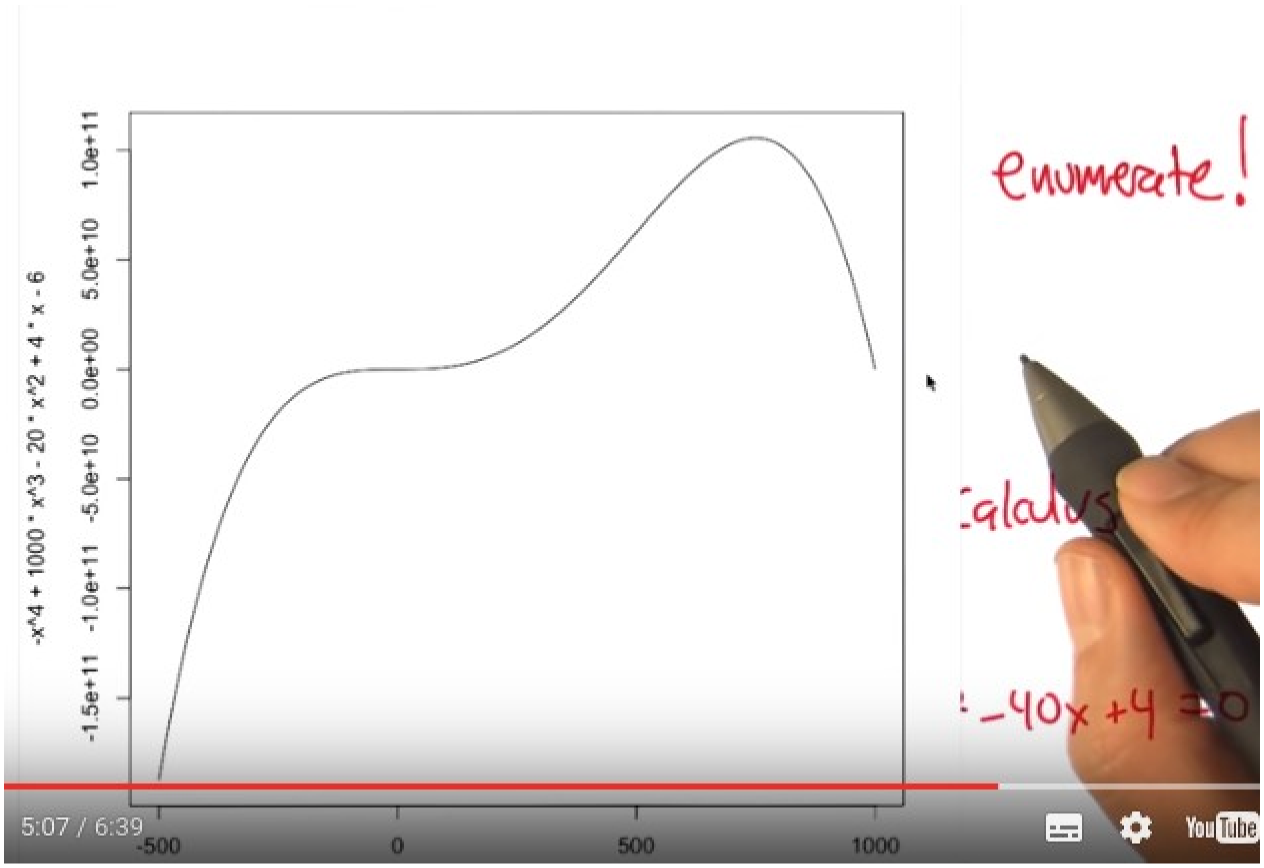
$f(x) = -x^4 + 1000x^3 - 20x^2 + 4x - 6$

$x^* = \boxed{750}$

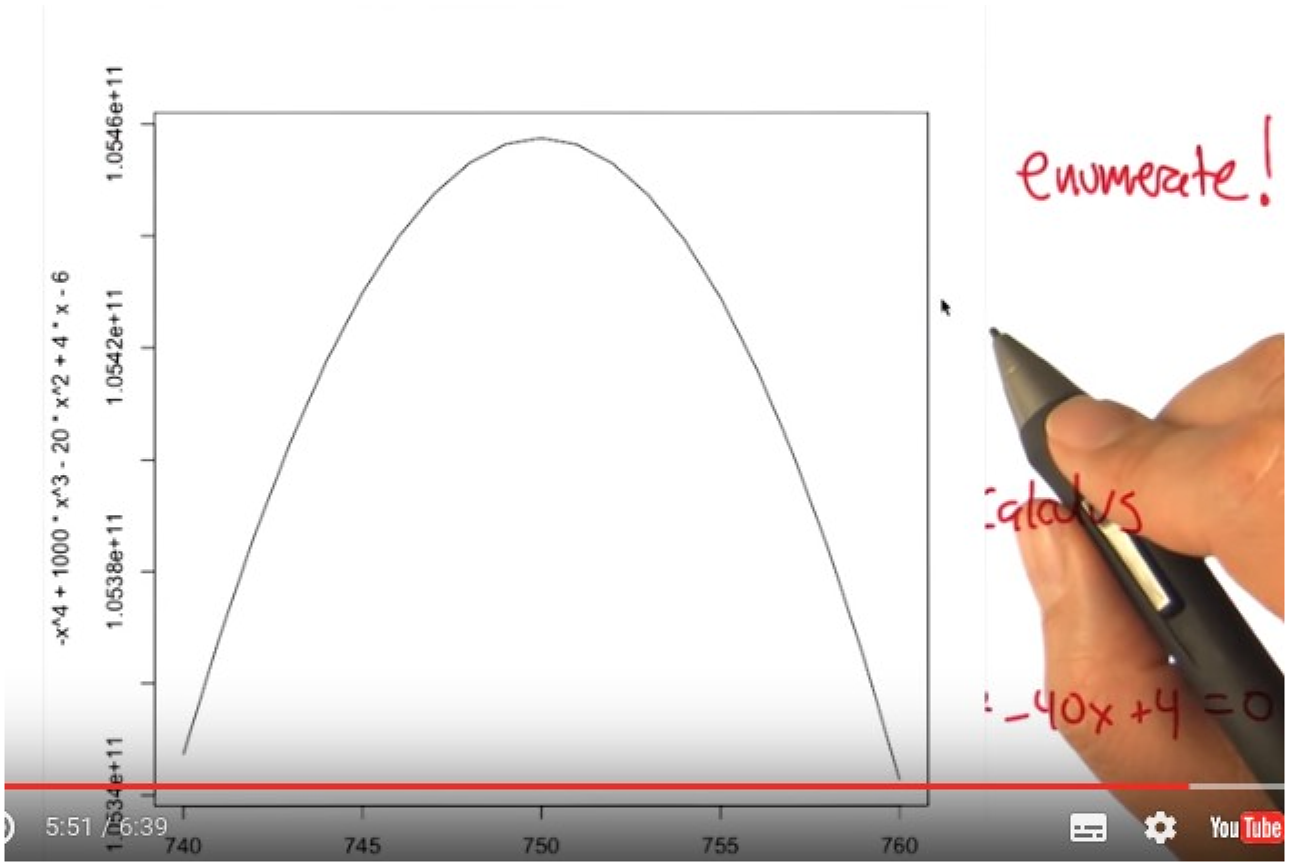
$-4x^3 + 3000x^2 - 40x + 4 = 0$



enumerate!

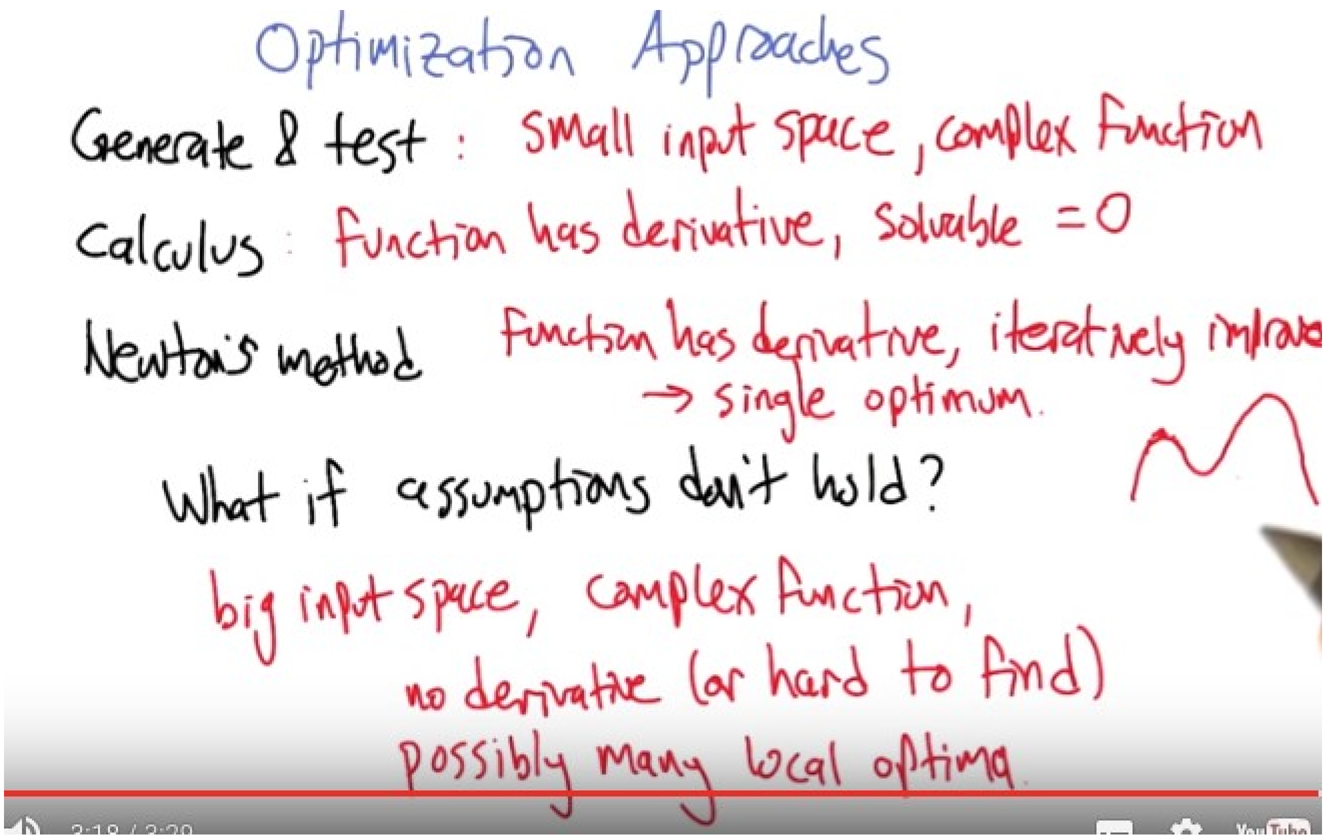


-x^4 + 1000 * x^3 - 20 * x^2 + 4 * x - 6
1.0e+11
5.0e+10
0.0e+00
-5.0e+10
-1.0e+11
-1.5e+11
-500
0
500
1000
enumerate!
calculus
-40x + 4 = 0
5:07 / 6:39



3. Alright, everybody, welcome back. So, let's, let's work through these. Charles, what is, what are you thinking for, for this first one. >> I'm thinking I should write a program, and just run it through, 'because that's the easiest thing to do. >> So what you're saying is this f function is so kind of wacky that it's hard to imagine actually analyzing it, and doing anything with it. But, on the other hand. The X, the space of X values is very small. >> Right. >> Good so right so we can enumerate all the possible values and just see which one gives the highest output. >> Yeah that's pretty straight forward if I just wrote the code, in fact it's like one line of code. >> So I wrote it for you. >> Oh good. >> I'm nice that way. >> You are nice that way, you optimized our time. >> Our did, for what it's worth, this is what the function looks like. I'm not even sure that's a function. >> Isn't that beautiful? >> It's something. It looks DNA gone wrong. >> Yeah, totally wrong, there's just mutations everywhere. It's like a bow tie that then tried to hide itself in another dimension. I mean like, you can see there's this interesting up and down-ness to it like a, like the sine wave part. >> Mm. >> But they're, they're interdigitated with each other. It's all, it's just crazy to me. >> Interdigitated. That is an excellent word. >> Yeah, it means, it means to put you're fingers together. >> Hm. >> So, >> So what happens if you plot that but with, connecting lines? >> Oh, oh, hee, hee. Sure. [LAUGH] >> Wow. >> You get somethin' like that. Wow. >> This is a kind of unfriendly function. >> Yeah, this is not the friendliest function I have ever seen. >> And now it looks a bit like, actually it looks a bit like an insane bow tie. >> Nice. Electric bow tie. >> Yeah. >> So it turns out that there's two points that are really really close to the top. There's this one at 11 and there's this one I don't know a little bit after 80 or so. But the 1 and the 11 is actually the largest. >> Hm. This function goes too loud. >> [LAUGH] It does indeed. So, I'm going to say 11. I wanted to make a crazy function that was the, was the first thing that popped into my head. Actually I did, the second one was a mod three at first. >> Hm, well, it's an interesting thing, right because if you tried to reason this out, it would be very difficult to do. I mean, first off, mod

isn't very nicely you know, it's very easy, it's not very easy to deal with algebraically in closed form but you can start reasoning and say, well, the dominating factor is mod seven in the sense that you're never going to be greater than six and of course the mod six is never going to be greater than five so the best you can get there is five and then you minus the sign, which is going to be some small number so you can kind of pick all the values that are going to give you something like. Five there, and then whichever one has the smallest Sine, that's the one you go for. I wonder if that would get you anywhere close to the right answer. So, interesting that that's not quite true here because 11 mod six is, no 11 mod six is five, right? >> Hmm. >> Squared is 25. Mod seven is four. Is that right? >> Anyway. >> Hm, okay. >> Lets do the second one. >> Okay. >> Alright so it turns out that, again, 11 and 80 are really close but 11 is slightly larger and it's just almost five, but not quite. Alright, so for the second problem, now we can't use of enumerating because our X's constitute the entire space of the, reals. >> So, there's like, 200 of them. >> More. So we need to do something else. And I think what you were pointing out to me is that because this is a nice polynomial. We can actually use calculus to figure out where the maximum is. >> Right, just take the derivative and set it equal to 0. >> So unfortunately, I made that a little harder than than probably you'd appreciate. So the derivative would be this, and setting it equal to 0 would give us this equation, which unfortunately is a cubic. >> Right. >> So it should have three solutions. Perhaps, and you could just check each of them. But solving the cubic is sort of a pain. >> Mm mm. >> I guess I could have been a bit nicer. >> You could have been. >> Sorry. >> But it all depends what you're optimizing. >> So how would you go about solving this now? >> I would go on Google and I look up how do you solve cubics. >> [LAUGH] Sure, that's not a bad idea. Any other things, because it is nice and smooth, right? So let's, here, let's, let me actually find it for you. So here's the function, and I just plotted a subrange of it, from minus 500 to 1000. You know you could ask, could it be that the maximum is actually someplace outside this range. You know how you'd answer that? >> No. So because it's a fourth order polynomial, like think of like a, a quadratic, >> Yeah. >> It's got the one bump in it, a cubic can have as much as you know one up bump and one down bump. And a qua, and a quartic ca, a 4th degree polynomial can have an up bump, a down bump, an up bump and then back down. >> M-hm. >> So we're actually seeing all the activity here, outside this range it just plumettes negatively. It, it can't turn around and come back up again. So because we can see the two bumps we are good. >> OK. >> And in this case, it looks like the bump that we want is somewhere between 500 and a 1000, somewhere in the 700 kind of zone and well so one thing we could do because it's really well behaved, we can kind of zoom in on the function at that range. Mm-hmm. >> So there we are now zoomed in from between 500 and 1,000. We can see, okay, well, actually it looks like the peak is maybe between 600 and 800, you think? >> I'd say 700 and 800. >> Between 700 and 800. All right, well let's take a look at that. >> Ooooo. >> Nice and Pointy. Alright. So how bout that? >> 740. How bout 740 and 760? >> So we really are kind of getting into the nitty gritty here. So, so, for the quiz we'll you know, we accepted anything, between you know, 745 and 755. But in fact we can keep zooming in and we can use calculus, and we can, we can really hone on what that, the tippy top is there. But that's not so important right now, we could also something called Newton's method, you know Newton's method? >> I remember it. >> Newton, newton's method said if, you know, guess a position like you know, 455 sorry 755, and use the derivative at that point which is actually really easy to compute, doesn't involve solving the, the cubic, it just involves, Writing down the cubic and evaluating at that point, we can actually you know, fit with the straight line is here. And we can take it, take steps in the direction of that line, it's a, it's a gradient ascend kind of method. >> Yeah. >> And, and that'll allow us to hone in, as we get closer and closer to the top. The slope is going to flatten out, we're going to take smaller steps And this process converges to whatever the peak is, I think it's a little bit under 750. >> That looks about right.



Optimization Approaches

Generate & test : small input space, complex function

Calculus : function has derivative, solvable =0

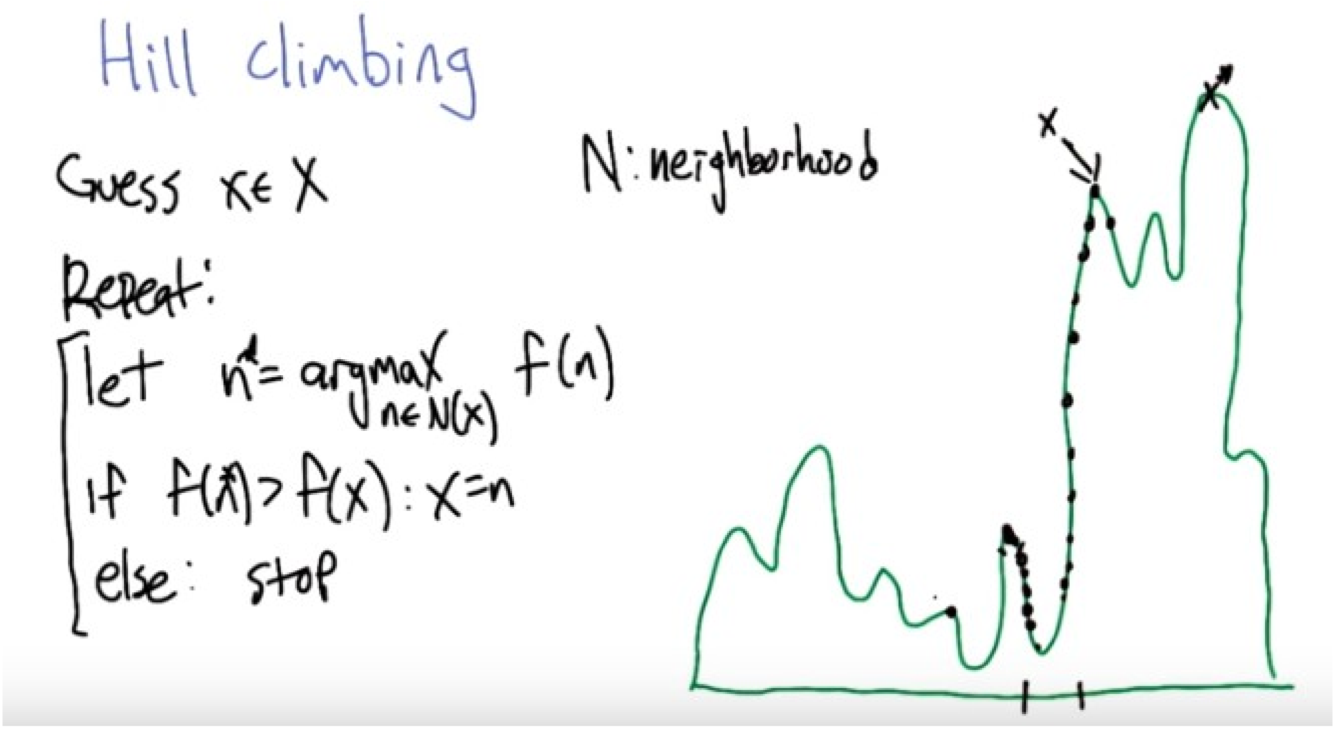
Newton's method function has derivative, iteratively improve → single optimum.

What if assumptions don't hold?

big input space, complex function, no derivative (or hard to find) possibly many local optima.

4. So in terms of optimization approaches, we actually just looked at a couple of different ideas. We, we just looked at generate and test, this sort of idea that you can just run through all the different values in the input space and see which one gives the maximum. When is that a good idea and when is that not a good idea? >> Well, it's probably a really good idea when you have just a small set of things you need to generate and test. >> Yeah, it really does require that we have a small input space. And it's also particularly helpful if it's a complex function, right? Because there really isn't any other choice if the function has kind of crazy behavior, like in the mod example that I gave. Alright, for things like calculus, like, just solving for what the optimum is analytically. When, when is that a good idea and when is that not such a good idea? >> Well, it's a good idea when you have a function where you can do that. Where, there, well, first off, it has to have a derivative. >> Right, and so, it seems like, well, how are you going to write down a function that doesn't have a derivative. Well for the thing we're trying to optimize is some kind of, it could be, crazy because we defined it to be crazy. We can define it with little, you know, like, ifs and things like that, to make it only piecewise continuous. Or it might be the, the, the thing we're optimizing is some process in the real world that we just don't have a functional representation of. All we can do is evaluate the fitness value at different inputs. >> Right, or if the inputs are discrete. >> Right, like in the first example, right. So it might not actually give us any feedback if the derivative is, is, if we're not moving in a smooth continuous space. >> Hey is mod, does mod have a derivative. >> Almost everywhere. >> Just not everywhere. >> Yeah. There's that, you know, because mod kind looks like ner, ner, ner, ner. So it has this nice derivative everywhere except for at these, at the jumps. >> Which happen pretty often. >> Yeah, there's a lot of them. But they're, you know, measure zero. >> Okay. >> Like, if you just picked a random number, it wouldn't be at the jump. So it's not only important that the function has a derivative, but the derivative, we need to be able to solve it, solve that derivative equal to zero. Newton's method can be helpful even outside of that case, where we have a derivative, and we have time to kind of iteratively improve, right? Just

keep querying the the function. Creeping up on what the optimum turns out to be. So, okay, but then what do we do if these, these assumptions don't hold? So we have a real, what would that mean? It would mean, what, big input space. >> Still have a complex function. >> Complex function. >> No derivative, or difficult to find derivative. >> I knew I had that derivative around here someplace. >> [LAUGH] You know it's always the last place you find it. >> [LAUGH] Indeed. Yeah. Exactly. And, and actually I, I, I should have mentioned one more thing about Newton's method. Which is that the function has a derivative you iteratively improve. And it really wants you to have just a single optimum, because even Newton's method can get stuck if it's in a situation where you have a, a curve say like, like that. because Newton's method is just going to hone in on the, the local peak. >> So that would be bad if you have lots of local maxima in this case, or optima in this case. >> Yeah, yeah, yeah, yeah. So, so that's, you can list this as well, possibly many local optima, right. The, the, the function has a kind of a peak. That things around the peak are less than the peak but that peak itself may be less than some other peak somewhere else in the space. >> Makes sense. >> So, as you might expect that our answer to this hard question is going to be randomized optimization. The topic of today's lecture.


Hill climbing

Guess $x \in X$

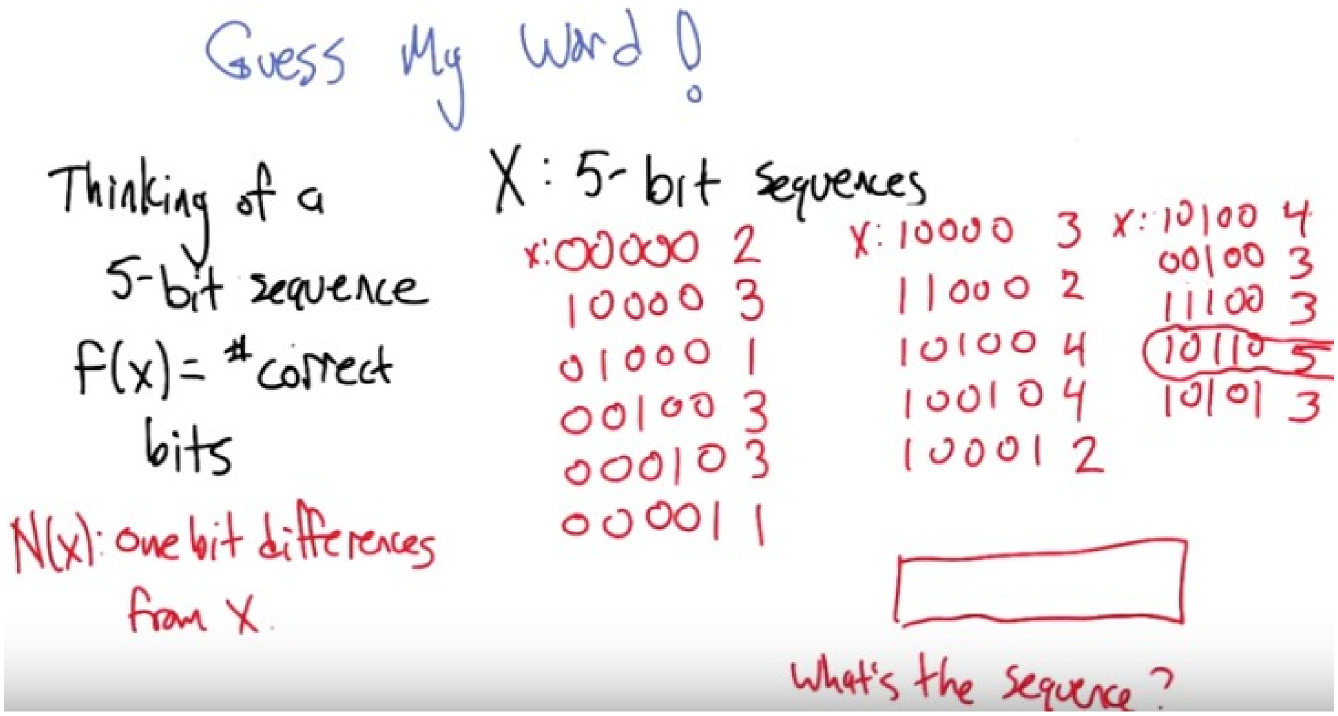
N: neighborhood

```
Repeat:
    let $n^* = \operatorname{argmax}_{n \in N(x)} f(n)$
    if $f(n^*) > f(x)$: $x = n$
    else: stop
```

From online:
What is the difference between argmax and max ?
To use a mathematical example, consider the function f(x) = 1 - x^2.
Then Max f(x) = 1
but Argmax f(x) = 0
as x = 0 is the (unique) value for which f(x) = max f(x) = 1.

5. So here's an algorithm that you'd probably either already know about or would be able to come up with rather quickly. Which is the idea of hill climbing. So if this is the function that we're trying to find the maximum of, then one thing we could do is imagine just guessing some x, I don't know, say, Which has some particular f of x value, and then to say, okay, well, let's move around in a neighborhood around that point, and see where we can go that would actually improve the function value. So we, here the neighborhood might be, you know, a little to the left, a little to the right on the x axis, and, what we

find in one direction it's going down and the other direction it's going up. So what hill climbing says is find the neighbor that has the largest function value. This is steepest ascent hill climbing. And if that neighbor is above where we are now, has a higher function value than we are now then move to that point. Otherwise, we stop because we are at a local optimum. So, what this is going to do is it's going to iterate moving up and up and up and up this curve, always in a better and better and better direction until it hits this peak here. Then, it's going to look on both sides of it. The neighborhood, everything in the neighborhood is worse. So, it just stops there and this is the x that it returns. Which is, you know, not bad answer. It's not the best answer. But, it's a good answer. >> Hm. What if you had started out just a little bit more to the left? Say, on the other side of that valley. >> Oh, like here? >> Yeah. >> Hm. So then, well, let's see. What would it, what would it do? We start there. We take a step. We, we say, what's the neighborhood? The neighborhood or the points, just a little bit to the left, and just a little bit to the right. One of them is an improvement. So it takes the improvement. And again, it keeps finding it more and more improved. And then it gets to this top of this little bump and it says, okay, both the points in my neighborhood are worse than where I am now. Hurray, I'm done. >> Hm. >> So, that's not even worst of the local optima, you could actually get stuck here as well, which is even lower. >> Well, you could get stuck anywhere. Well, at any peak. >> Yes, exactly right and that's the lowest of the peaks.

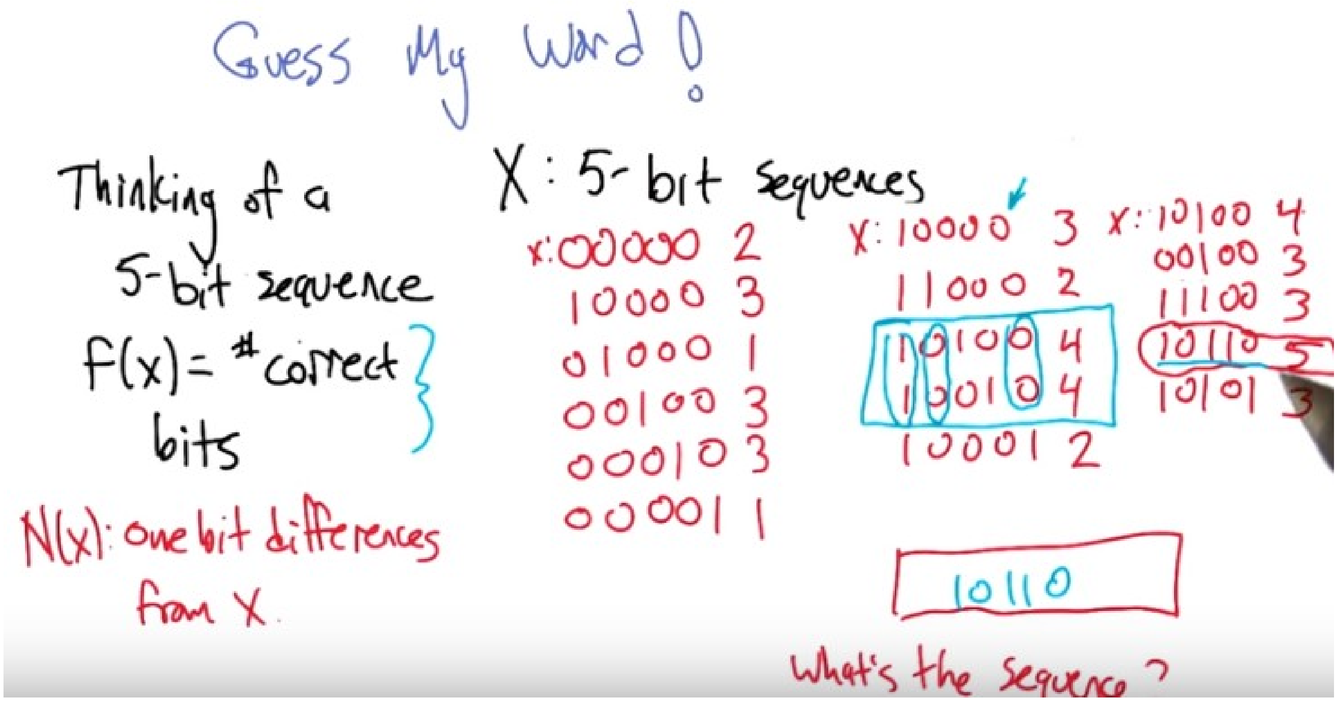


每一個紅色 neighbor 右邊的紅數即 number of correct bits

6. Alright. I want you to guess my word, using this hill climbing approach that we've just been talking about. And unfortunately, well, we tried this will actual words and the space is a little bit big and frustrating. But what we can do instead is pretend that a five bit sequence is a word. So, I'm thinking of a word and by word I mean five bits. And what we're going to do is I'm going to give you, for each time you guess a 5-bit sequence, I'm going to tell you the number of correct bits. So in each position it's, it's, if you matched what I am thinking in that position, then I give you an additional point for that. >> Okay. >> And you're going to now step through the hill-climbing algorithm. Okay, so three things. 1, if

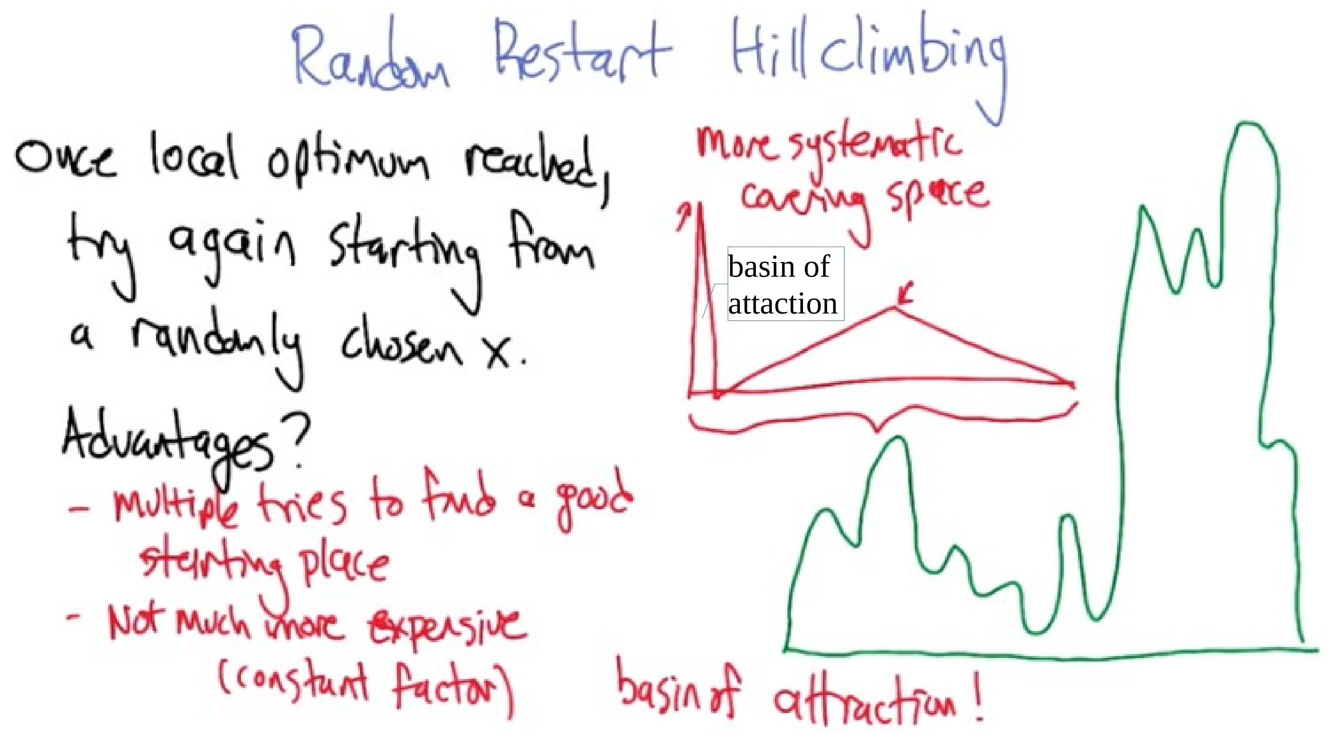
you had done eight bits, that would be a word. And. >> Oh. Nice. >> 2, four bits is a nibble. Okay, here's the third thing. We need to define a neighbor function. So, I'm going to define a neighbor function. So, I can think about this. This is all one bit differences from where you are. >> That's good and, and an interesting question is, was it your job to come up with that or was it my job to come up with that? Now is it, is it part of the problem or is it part of the algorithm that's trying to solve the problem? >> Huh. I'm going to say it's a part of the algorithm that's trying to solve the problem. >> That's how I think of it, too. >> Alright, so, we have to start somewhere, so let's just pretend for the sake of argument that we start it at all five zeros. >> And I'm going to give you a score of two for that. >> Okay. And so now I have to do all possible neighbors, and there are five of them. So there is one followed by four zeros. 01 followed by three zeroes and you know, the rest of the identity. >> Alright, here is all your neighbors for the x, you're aware. >> Mm-hm. >> And here is the scores for all of those neighbors. >> So, there are three of them that are maximal but for the sake of simplicity and drawing, I am going to stick with the first one. So you can get rid of the bottom four. >> Alright, so here is our new x. >> Okay. And now I have to try all different. So one of the neighbors would be all zeroes. But I already know what that is. So I don't have to worry about that. >> That's good. So the algorithm, the pseudocode that we wrote didn't do that, do that optimization but in general if as long as the neighborhood function is symmetric then you can save yourself a little bit of effort. >> Right, so, or we can just keep track of everything that we've ever seen, although that gets very space inefficient very quickly. And since it's the identity maker's [UNKNOWN] with the one matrix. [LAUGH] Okay, so what are the numbers to that one? Wow, that's pretty cool! So, you said we're getting there very quickly. So, it's going to be, so, [CROSSTALK] >> In fact, we now, have enough information, that smart human being could actually jump to the optimum. >> Yeah, I think we do. I was actually thinking about that. >> Let's see what happened.

7. >> All right, let's figure it out. >> well, so, first off, I think the first thing, if I'm right is. So, is the following statement true? Once we know four of them are right, we will stumble onto the right answer when we do the neighborhood. >> Well, so, okay. So, that's a good idea. We could, actually, do one more step. >> Mm-hm. >> And see what happens. So yeah, because, in this particular case, that because of the fitness function is the way that it is, the number of correct bits. There's always a neighbor that is one step closer to the target. >> Right. >> So we're always going to be incrementing as we go. >> Mm-hm. >> So this is a very friendly fitness function. >> Mm-hm. >> It's only, only a global optimum. >> Right. So, do you want to do that? And then maybe we could talk about how we could have been even more clever? So, let's just pick the first one. Okay. And so, now we have to do the fourth one, so it's 10110, and then the fifth one is 10101. >> Okay. All right, so this if is x and these are the neighbors of x, these are the scores. And it turns out that one of them actually has all five matches. So that was in fact the pattern I was thinking of. Here, just to prove it 10110. And if we now continue the search from here, all the neighbors are going to be worse, obviously. >> Right.



>> So, (不重要) how could we have figured out the sequence without that extra information that I just gave you (即只有前兩列)? >> Well, if we look at the two that have four, we can basically just see that where the, the, they must have three in common. Which they do. one, the first, the second, and the last one. >> Okay. >> And each is off by exactly one. So, that means that, well, since I know the answer, I know that you just take one, you just take one from each. So either they have to both be ones or they both have to be zeros. >> All right, and so do we know that the 01 doesn't work? We probably know that the 01 doesn't work. >> because that, then the answer would be 10000, 10000. >> Which we already have. >> Which is exactly, we already know is a three. >> Right. So then it has to be 10110, which in fact it is. >> Which it was. Cool >> Right, so this was in fact a very nice fitness function. and, and, and, you know, it seems to me that we had a, a, nice little advantage here in that we really understood the structure of our space. But if this were a 5000 bit sequence, where the fitness function was some weird thing. Where, in fact, we didn't even know what it was. We just knew that there was one. Then, that would've been a whole lot harder. actually, here's a question for you, Michael. If I had told you, if I had given you this, but I haven't, didn't tell you what the fitness function was. Just that when you gave me a sequence, I evaluated it for you. >> Uh-huh. >> We wouldn't have been able to do any of the stuff that we were doing. I mean, we'd still be able to do it, but we wouldn't have been able to jump ahead. Like, we just, like we could've done with those two that were four, because we wouldn't even know what the maximum fitness value was. >> Yeah, we would have to done this last step to, to find that one of them has a score of five. And then we would have to done that step again to make sure that there wasn't anything locally better than five, because we wouldn't of known that if we didn't have a kind of semantic understanding of the fitness function. But, but the rest of the algorithm was perfectly implementable without knowing that, right? All, all that you were basing your decisions on was, you gave me bit sequences, I gave you scores, and you used those to figure out what next bit sequence to ask. It, it didn't use the fact that the fitness function had a particular form. >> That's right. That's right. >> Now, this wouldn't, this particular problem is very, very well behaved, because it has one global optimum, this bit sequence that has a score of five. Nothing else had a five. And once you're at five, and, and no other bit sequence actually could you be stuck. Right, so it was always the case that you could always get closer to the target pattern, so your score can always go up. But, in general, that's

not how things work. That you may actually be able to get stuck in local optima and we need a way of dealing with that.

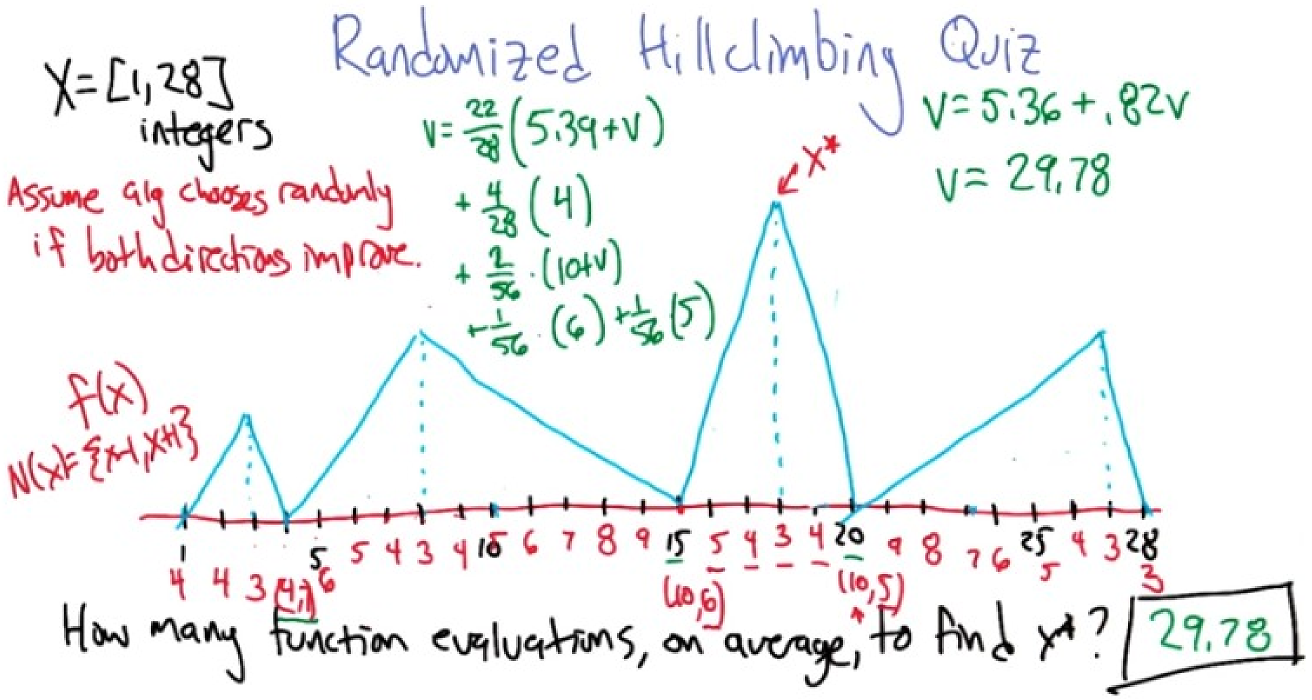

Random Restart Hillclimbing
Once local optimum reached, try again starting from a randomly chosen x.
Advantages?
- multiple tries to find a good starting place
- Not much more expensive (constant factor)
more systematic covering space
basin of attaction
basin of attraction!

8. So Random Restart (這不是人名) Hillclimbing is going to give us a way to deal with the fact that hillclimbing can get stuck, and the place where it gets stuck might not actually be the best place to be. And so we can go back to that function that we looked at before. So what are some places that this could get stuck? >> There, there, and there. >> And others to boot. So what randomized hillclimbing's going to do is once a local optimum is reached, we're just going to start the whole thing again from some other randomly chosen x. It's like, you know, sort of what you do if you were trying to solve a problem and you got stuck. You're like, okay let me just try to solve it again. So why is this a good idea? >> Hm. Well one it, it, it takes away the luck factor of I happened to pick a good starting place. Although I suppose it replaces it with the luck factor that I randomly happened to pick a good place. But that's okay because I'm going to keep randomly picking good places. Or randomly picking starting places >> Yeah. So you get multiple tries to find a good starting place. That there could be various places where you start that don't do so well but as long as there's places where you do well. Then you might luck into starting one of those places and climb up to the tippy top and win. >> Another advantage is that it's actually not much more expensive. So, whatever the cost is, of climbing up a hill, all you've done is multiply it by, you know, a factor, a constant factor, which is how many times you are willing to do a random restart. >> Yeah, that's a good way of thinking about it. That it's, it's really just random hill climbing, repeated how every many times you feel like you have time to repeat it. And it can actually do much, much better. Now, if there really is only one local. Sorry, if there is only one optimum and there is no local optimum, then what's going to happen when we do random restart hill climbing? >> We'll just keep getting the same answer. >> Yeah, over and over again. So, could be that we, that we might keep track of that and notice, okay, you know, what we seem in a space where these random restarts aren't getting us any new information. So, you might as well stop now. >> Well, that's one answer but I could think of another answer. What's that? >> So, maybe you just keep starting too close to the same place you were starting before. I mean, it may be random, but you can get unlucky in

random right? So, maybe you should make certain your next random point is far away from where you started, so that you cover the space. You want to cover the space as best you can. Yeah the randomness should do that, at least on average. But you're right, so we could try to be more systematic. So here might be an example (中間那個紅圖) of a kind of function where that would be really relevant. So imagine we've got, here's our input space and most of the random points that we choose are all going to lead us up to the top of this hill. But a very small percentage are actually going to lead us to the top of the, the real hill. The top that we really want. The optimum. So yeah, we, we might not want to give up after a relatively small number of tries because it could be that this tiny little, I know, I'm going to call it a basin of attraction. >> Mm-hm. I like that. >> And if it's small enough, it might take lots of tries to hit it. In fact, it could be a needle in a haystack. In which case, there's only one place that you could start that has that optimum. And that could take a very, very long time to luck into. In fact, the hill climbing part isn't doing anything for you at that point. >> Yeah, but you know. If you're in a world where there's only one point that's maximum. And, but there's only one way to get to it by having to land on a single point. Then, you're in a bad world anyway. [LAUGH] >> Right, I mean, there's going to be, nothing is going to to work out. Nothing is going to help you in that world. >> Yeah. >> Right, and in fact, I would, I would claim since everything has some kind of inductive bias. Right you're, you're making some assumption about like local smoothness or something. that, that makes sense here because if you, if you are worried about a world where there is always some point of the infinite number of points you could be looking at say. That is the right one and there is no way of finding it other than having to have stumbled on it from the beginning. Then you can't make any assumptions about anything. You might as well look at every single point and what's the point of that. >> Fair enough. >> Mm-hm. >> So the, the assumption here I guess is that there, you can make local improvements and that local improvements add up to global improvements. >> Right. Hey I got a question. I got a semantic question for you, a definition question for you. >> Sure. >> You decided that you didn't, if you only had one optimum, you didn't want to call it a local optimum. So, is a local optimum, by definition, not a global optimum? >> So, if I was being mathematical about it, I would define local optimum in a way that would include the global optimum but it's just awkward to talk about it that way. because it feels like the local optimum is someplace you don't want to be and the global optimum is someplace you do want to be. But yeah that's right the global optimum is a local optimum in the sense you can't improve it using local steps. >> Okay.

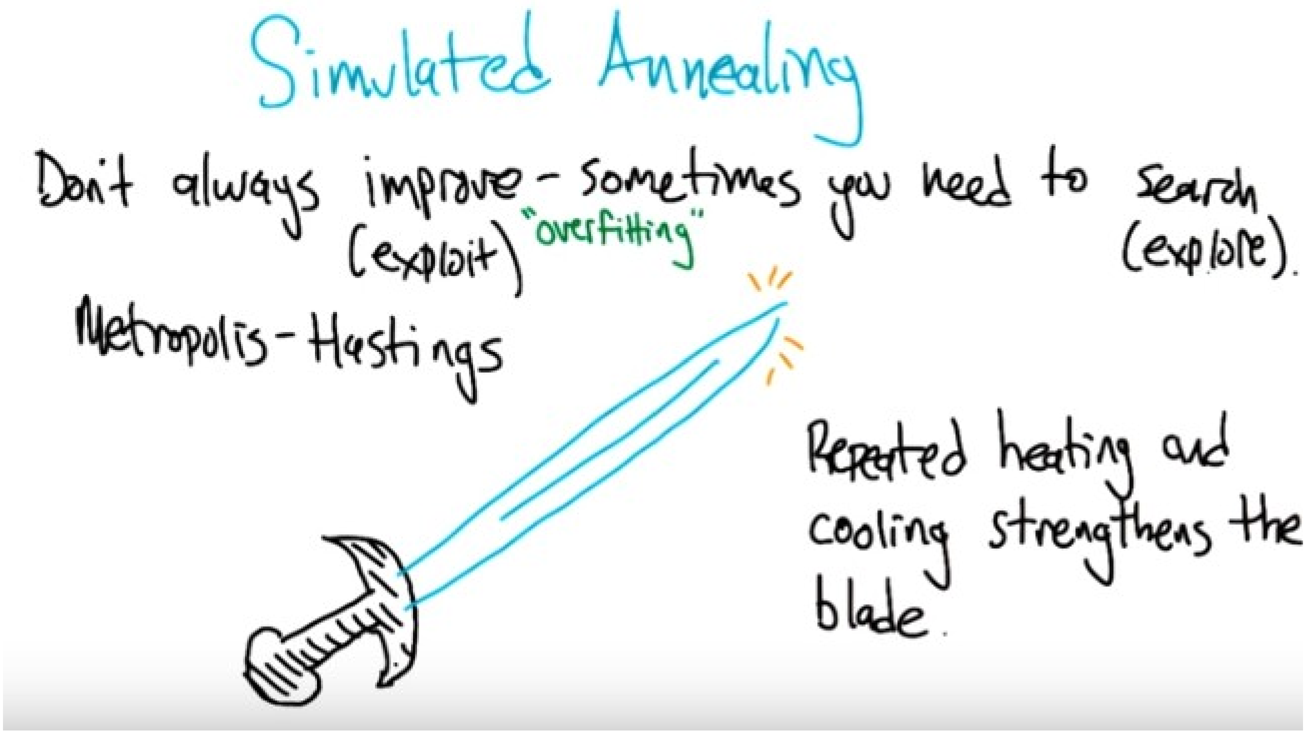
9. I think we can kind of get at some of these issues with a randomized hillclimbing quiz. So what was, kind of what is the advantage of, of doing hillclimbing and, and what does it lead to in terms of being able to find the maximum. So, so here, [LAUGH] yeah, Charles, this is going to be a quiz, and I, I guess you're not particularly happy about it. But let's, let's take a look here. We've got an input space of 1 to 28, the integers. And so here it is kind of out on the screen, all the values from 1 to 28. And here's the fitness function, or the objective function for each of those points. It sort of follows this, not exactly sawtooth, kind of piecewise, jaggy, I don't know what to call it. >> Drunk. >> [LAUGH] It is what it is. >> [LAUGH] >> And, here's the the global optimum. And what we're going to do is, we're going to run randomized hill climbing. And we're going to run it this way. We're going to assume that the algorithm chooses randomly, in both, if both directions improve. So it just flips a coin 50-50, then it does whatever evaluations it needs to do to figure out which way to move to go uphill. We'll use as of the neighborhood of x to be the thing immediately to its left and immediately to its right, unless of course you're at 1 or 28, in which case there's just one neighbor. So we're just going to use a simplification of the hillclimbing that we had before. Which is, it's going to check the neighbors, check both the neighbors if there's two of them, and if one of them is an improvement, it's going to go in that direction. If neither of them improvement then it's going to declare itself at a local optimum. And if both are an improvement, it's just going to flip a coin to decide where to go. And once it gets to the, to the local optimum, let's say it sort of climbs up this hill and ends up at this peak here, it's going to

realize that it's at the, at a local optimum, and then it's going to trigger random restart and pick a new x position to start at. And now, the question is, how many function evaluations on average is it going to take to find x star, the, the tippy-top of this peak? So, you know, maybe you could write code to do this, or maybe you could do it with math. This is probably a little bit more involved than most of our quizzes, but it should give you a chance to really dive in and get a feel for this notion that it matters kind of where these basins of attraction are. Alright, you think that's clear enough to give it a shot? >> Sure. >> Alright, let's go.


Randomized Hillclimbing Quiz
X=[1,28] integers
Assume alg chooses randomly if both directions improve.
V = 22/28 (5.39+V) + 4/28 (4) + 2/56 · (10+V) + 1/56 · (6) + 1/56 (5)
V=5.36+.82V
V= 29.78
x*
f(x)
N(x)={x-1,x+1}
How many function evaluations, on average, to find x*?
29.78

10. All right, so there's there's a bunch of ways you can imagine doing this. And here's, here's how I would suggest looking at it. So, for each of these 28 positions, there's some number of steps that it's going to take before it either resets, that is to say, you know, re, re, re-chooses at random, or has actually reached the tippy top. So, let's, let's actually evaluate these. So if you, if you happen to start at 1, how many steps is it before you realize that you, that you're at ~local ~optimum? You have to evaluate 1, you have to evaluate 2, you have to evaluate 3, and you have to evaluate 4 to see that you're at local optimum. >> Mm-hm. >> So, there's going to be 4 evaluations if you start at 1. If you start at 2, then it's going to be 1, you're going to have to evaluate 1, 2, 3, and 4 again (因為此時還要考查 2 左邊的 1). If you start 3, then you only need to evaluate 2, 3 and 4 to know that you are at a local optimum. >> Hm-mm. >> If you start 4, then you have to evaluate 4 and then if you randomly step in this direction, you are also going to need to evaluate 2, 2 and 3. All right. So then, I, I finished writing down what all these different numbers are. For each, at each position here, I wrote down in red, how many steps it's going to take before it hits a local optimum including the ones here around the actual peak. And I wrote two numbers for the places that were these cusps where it can go either way. So, now, we can actually work out what the expected number of steps to the global optimum is. It turns out for 22 out of 28 of these points, the average 5.39 before they discovered that they're at a local optimum, at which point we have to start the whole process over again. And so whatever the expected value of reaching the local optimum is, we have to, we incur that same cost again (故 5.39 後要再加個 V). >> That makes sense. >> Good. Okay. So, for 4 out of 28. These guys here. Then, they're going to take 4 steps on average and end up with the peak. Then, the only other things that left to figure out or what

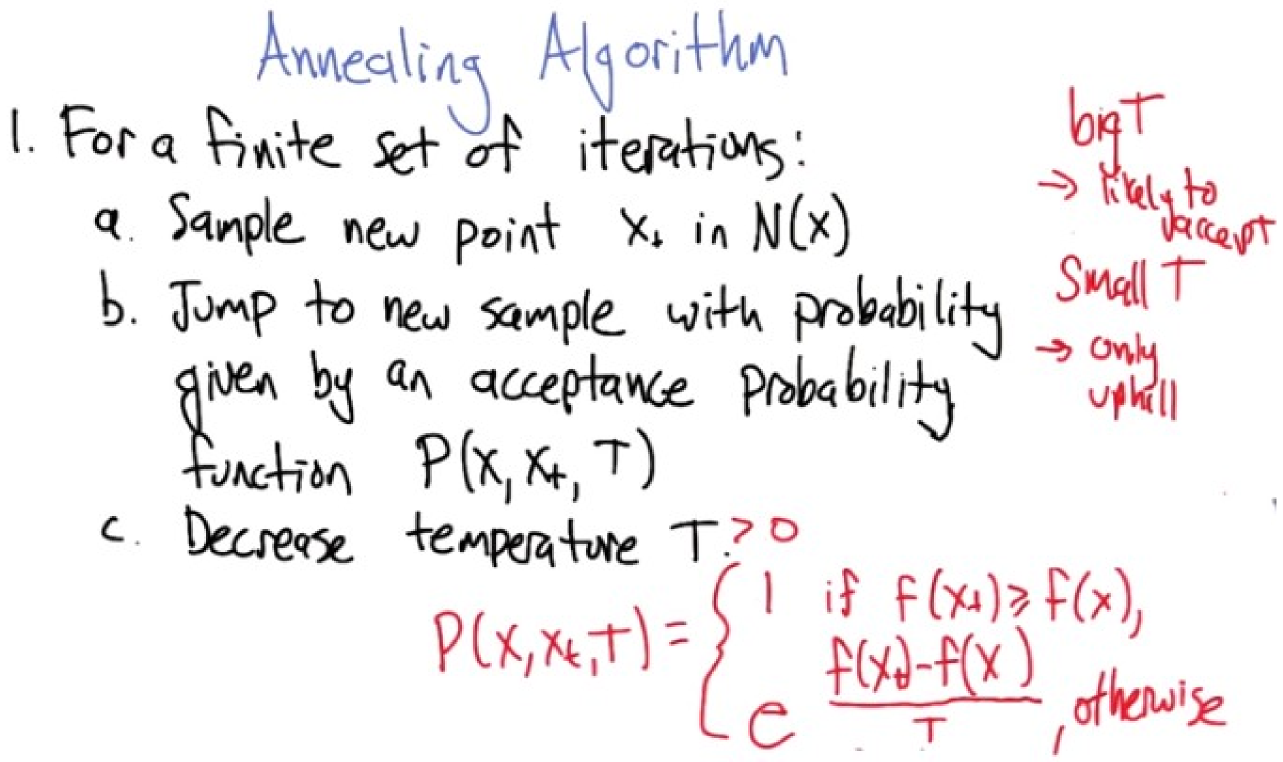
happens at these two weird cusps where just a flip. It's going to flip a coin and either get stuck on a local optimum that's not global or go to the global optimum. So, in 2 out of 56 of these cases it's going to have chosen to go to the local optimum (為何是 56? 因為此時我們可以想像所有的出發點都像 15 和 20 那樣有兩個選擇: 往左還是往右走, 即總共有 56 總可能. 2/56 中的 2 是因為有 15 和 20 兩個這樣的點, 它其實寫成這樣更好懂: 1/56 (10+V) + 1/56 (10+V)). The non global local optimum. And it just so turns out that it's 10 in both of those cases. It's 10 steps before you realize you're stuck again. At which point it's going to start the whole process all over again because it's stuck. Then in 1/56th of the cases, we're going to be here and choose to go to the right and, and take six steps to get to the global and here 1/56th of the cases we're going to take five steps to get to the global. >> Okay, so just to be clear, so the way you got 28 is that there are 28 numbers and you uniformly chose where to start. >> Yep. >> Where you got 56 is, you decided that for two of those numbers 15 and 20 you end up getting them 1/28th of the time. And then half of the time you do the local optimum and half the time you end up doing the global optimum. And it just turns out that in the case, in the cases where you end up going through the local optimum, those both happen to take 10 steps. And that's how you got 2 out of 56. And the last thing to notice is that for point number 4, where you could go to the left or to the right, since you do each half the time, it's like saying the number of steps there is just uh,11 over 2. >> On average. Yeah. Very good. So that was folded into this 5.39 number. You're right. Thanks for pointing that out. >> Okay. Sure. So this all makes sense and obviously if you just add those all up you can solve for V and the answer is [COUGH] exactly. >> So, just algebraically simplifying this equation gives us V equals 5.36 plus 0.82 V. Solving for V gets us 29.78. So, it only takes 29.78 function valuations before we first hit the maximum. Ta-da. >> Michael? >> Yes sir. >> So, that seems unfortunate. Because here's a really simple algorithm, for i equals or, hey let's say x. For x equals 1 to 28. Compute f(x). If it's bigger than my previous one, then hold on to it. And then just return whatever is the biggest number I see. So, that's linear and that's 28 evaluations, which is less than 29.78. >> Yeah, that's a good algorithm in this case. But you, I, I guess your point is that this, this whole idea of hill climbing and using this, this local information is actually costing us a little bit compared to just enumerating all the possible inputs and just testing them off. >> Right. Which I guess makes sense because the numbers are really small and [UNKNOWN] and you have a whole bunch of local optima, and so it's going to take you a long time to luck into the right one. But, it seems kind of sad that there's nothing clever you can do here to do better than 28, which is kind of what you want to do. >> Well, there's a couple clever things we could do, right? So one is, we could random restart faster. It turns out [INAUDIBLE] magic [INAUDIBLE]. So here's a question. What happens if you randomly restart after each function evaluation? How many function evaluations do we have to do, on average before we reach the maximum? >> I don't know. Oh, well, I guess it would be less than 28. >> It would be exactly 28. >> Oh it would? >> Yeah. >> 28's less than 28 so that's, that's fair. >> It's less than or equal to it, sure. >> Yeah, yeah, that's what I meant. >> And even better what happens if we actually keep track of points that we visited in the past? >> Then we don't pay the cost for re-evaluating them. >> That's right. >> So, then the only question is how long does it take us to luck up into getting between 15 and 20. >> Yes. Well, once we. Yeah, once we fall into that zone, then we're going to be done shortly afterwards. >> Right. so, how much of a, how much of a win is that on average? >> That's a good question. It's a little bit hard to tell. Because what's going to happen is if we follow this algorithm with the additional thing, additional attribute that says. If you already know the function valuation from som, from some previous iteration. We just reuse it, we don't get, we don't pay the cost for that. Then what's going to happen is we're going to get dropped off in random places in this space. We're going to hill climb visiting some things along the way. And possibly get drops in place where we've already done the function evaluation. >> Right. >> And therefore don't have to pay that cost again. But it's not as simple as saying how many different hops do we take before we land in this basin because if we land in say this basin here, the second one from the left. If, if we you know get dropped onto point 14, it's going to stay in that zone for a while, which is actually kind of bad because it's taking lots of function evaluations in that same part of the space. >> But look. But what you just said is important, right? Since we're only, we're only going to pay the cost of doing the function evaluation once, that means in the worst case. We would end up hitting the three bad basins before we hit the fourth basin. In fact, we would end up hitting the three bad basins enough times that we would end up visiting all of the points in them. So we'd land in point 14 which would get us all the way over to the peak and just to the left, and then we would land in 5, which would get us to the right, so we cover everything in the basin. So, the worst case is we cover every single point in all three of the basins before we get lucky enough to land in the, the good basin. >> Right. >> And then. But, the moment we land in the good basin, we will get to. Am I right? We will get to avoid at least one of those numbers. >> No. Not necessarily. So, for example, if we start at 15 and we randomly move to the right. To know that we're at the optimum here, we have to check the point 19. So we've actually touched everything. >> No, we didn't touch 20. >> Well, we may have already, unluckily, hit 20 because it's part of the right hand basin as well. >> That's true. >> But anyway, the point is that you're right. If we keep track of which points we've already visited, then we can't do worse than 28. Like, even if we're really, really unlucky, we can't do worse than 28. >> Right. >> Maybe, maybe on average it will be a little bit better than that. But but only one or two. >> But it will still be better. >> I think it still be better. >> On average, on average it will be better. >> Yeah, its true. I think, I think in this case it's, it's just a contrived example with lots of local optima and just a linear space so that there's a the, the attraction base ends up being relatively small compared to the size of the space. >> Hm. Okay, that makes sense. So, what, what should I learn from this? So I guess what I learned is randomized optimization or randomized hill climbing, here, may not do better than evaluating all this space in the worst case, but it won't do any worse. And every once in a while, you will get lucky and do much better. At least if your cost is function evaluations, as opposed to steps. >> That's right. Yeah and also, well and I, and, the thing that I was hoping, that you'd notice is this idea that randomized hill climbing, depends a lot on the size of the attraction base and around the global optima. So if we make that really big, then this is a huge win. If we make it really small then it's less of win. >> No and that makes a lot is right. So if, if, if the third one, had gone all the way from say number 3 to number 26. Then you would fall into it immediately and you would just, you would get there very quickly. >> Right. >> Okay, yeah, I see that, that makes, that makes a lot of sense. So, so when you asked me before what the advantages are, the advantages are, you know, you, you're in a place where the attraction basin for your global optima are in fact, relatively big. If you could, once you luck into that part of the space and there's lots of ways to luck into that part of the space, you can get there very quickly. So it's a big win. >> Agreed. >> Okay, good. You know, okay, that makes sense to me, I like that.


Simulated Annealing
Don't always improve - sometimes you need to search
(exploit) "overfitting"
(explore).
Metropolis-Hastings
Repeated heating and cooling strengthens the blade.

Aneal: 退火

11. Alright, we're going to look at another algorithm that kind of takes the idea of random restarts one step further. Instead of just waiting until you hit a local optimum to, to decide that you're going to randomly restart, there's always a chance of taking a step in a downward direction while you're trying to do your hill climbing. And the algorithm that we're talking about in concretely is called Simulating Annealing. So the basic idea is that we don't always improve. Sometimes we actually need to search. We need to take the point where we are now and wander away from it, with the hope of finding something even better. And you can think of this as being related to the notion exploring and exploiting. The hill climbing is what it's doing is always trying to exploit. It's always trying to climb its way up the hill as quickly as it can, which can lead it to getting stuck. Exploring is the idea of visiting more of the space with the hope that you can actually climb even further. >> So, which is better then, exploring or exploiting? >> Well you have to trade these off really carefully. So, if you're just exploring, it means you're just randomly wandering around the space. You're not using any of the local information to improve yourself. But if you only exploit then you can get stuck in these local optima. So you need to do a bit of both. >> So, since I'm always trying to connect this back to stuff we talked before. If you exploit all the time that's kind of like overfitting. That's like believing your data too much and not taking any chances at all. Does that make sense or is that too much of a stretch? >> I doesn't make sense to be me but I'm sure it make sense to you. >> [LAUGH] It makes sense to me. >> So you want to think of that as being kind of like overfitting. >> Well, you know, the fundamental problem of overfitting, right, is believing your data too much. And, and dealing with just the coincidences of what you happen to see. And that's like being very myopic. And exploiting, in this case, only taking the direction which you go is like believing the data point where you happen to be. I predict, from this one example, that I should be headed in this direction, and I don't care about anything else. And I'm not going to worry about anything else. Well, that's kind of like believing too much, which is sort of what overfitting is. Meanwhile search on the other hand, the exploring, is like believing nothing and taking advantage of nothing. So obviously as you need to take, as you said, take advantage of the local information. You need to believe it at least a little bit at least some of time time or otherwise it's as if

you've learned nothing. >> Alright, well I definitely agree with you that there's a trade-off there and that there's a resemblance to overfitting. I don't, I don't know how deep that resemblance is. >> Superficial. They're like twin cousins. >> Twin cousins from another family branch. So the simulated annealing algorithm that we're talking about is related to an algorithm called Metropolis-Hastings, which I always thought sounded really cool. And the whole idea of it actually gets into metallurgy. So here I drew a picture of a sword. And we want the sword, when we're making a sword if we're a blacksmith and we're making a sword, we want it to be nice and hard. And the way to make it hard and sharp, and, and, and well structured, is for all the molecules to be, be aligned in the same way. So they fit together very snugly and they kind of lock in. And, so, there is a bit of an optimization process that has to happen. There is all these different ways that the molecules can be arranged and we would like to settle into an arrangement that has the lowest energy. Well, it turns out that the way blacksmiths have figured out to do this is they will make the sword and then do repeated hea, heating and cooling, and that actually show, they can see that that actually strengthens the blade, and, and what's really happening at the molecular level is it's giving the molecules an opportunity to realign themselves, to find an even better solution to the problem of how they can all fit into that space. So, this annealing is, is this idea of heating and cooling. And we're not really literally heating and cooling, though of course when we run the computer it is getting a bit hotter. We're just simulating the idea of the temperature changing.



Annealing Algorithm

1. For a finite set of iterations:
   a. Sample new point x_t in $N(x)$
   b. Jump to new sample with probability given by an acceptance probability function $P(x, x_t, T)$
   c. Decrease temperature T. > 0

big T → likely to accept

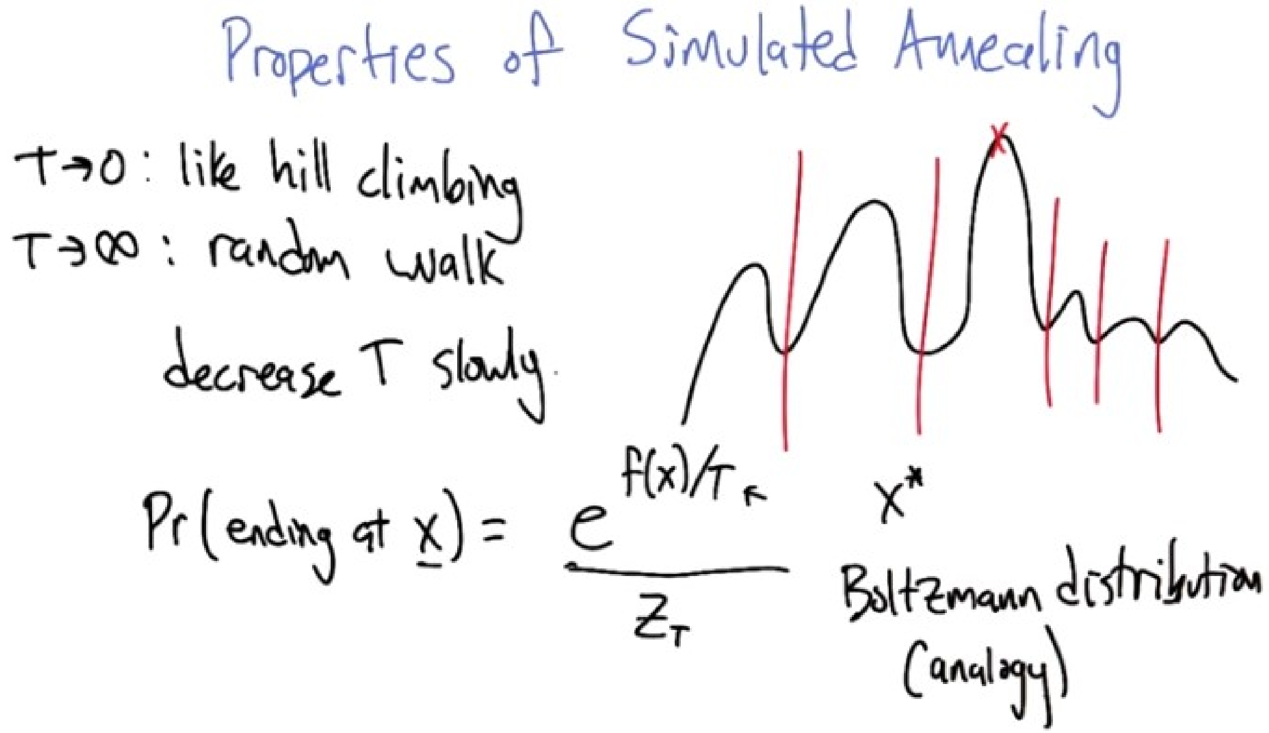
Small T → only uphill

$$P(x, x_t, T) = \begin{cases} 1 & \text{if } f(x_t) \geq f(x), \\ e^{\frac{f(x_t) - f(x)}{T}}, & \text{otherwise} \end{cases}$$

上圖中是 x_t, 而不是 x+
注意上式中 e^(....)是不會大於 1 的, 因為只有 f(x_t)小於 f(x)時, 才會用 e^(....)來算(否則直接為 1), 故可表示概率.

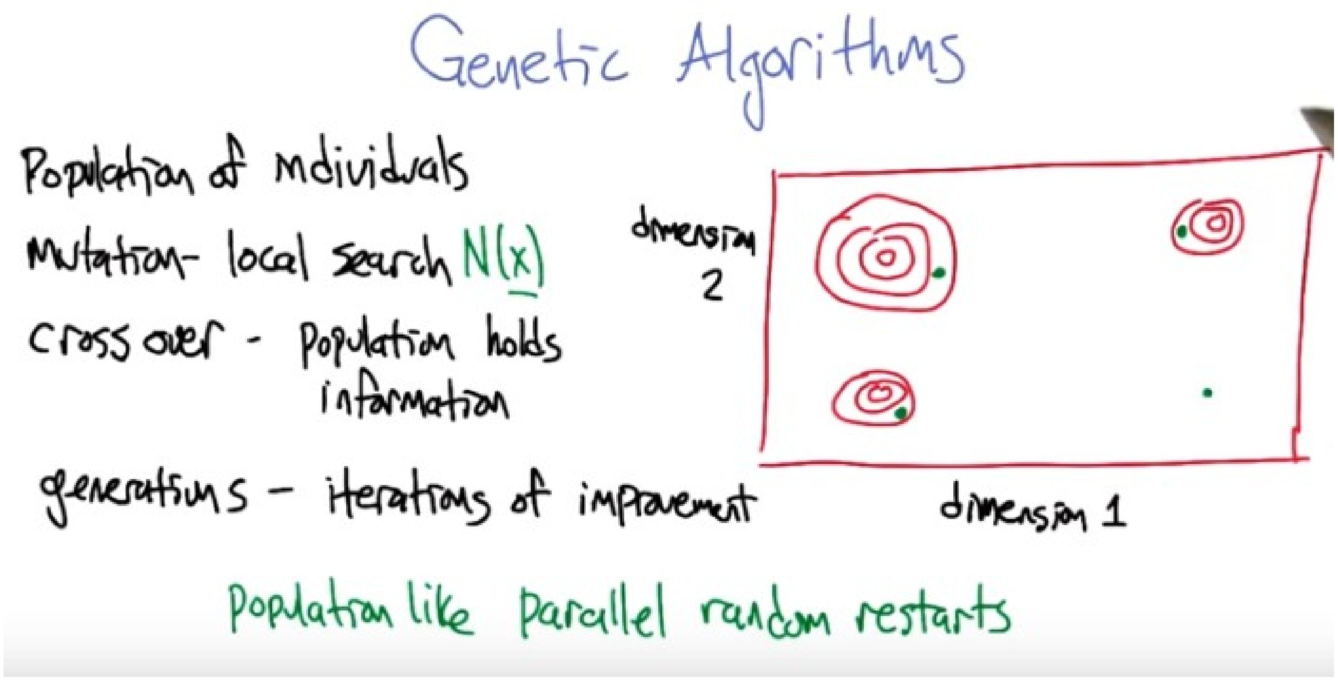
12. So, that's kind of just the motivation for the simulating the Annealing Algorithm, but there's an actual algorithm and it's, it's remarkably simple and remarkably effective. So, the algorithm goes like this, for, we're just going to repeat for some finite set of iterations. We're going to be at some point x,

and we're going to sample a new point x of t, from the neighborhood of x. And then what are we going to do? Well, we're going to move our x to that x t, probabilistically. So in particular, we've got this probability function pxxtt, which is going to help us decide whether or not to actually make the move. And it's, and, and the form of that function is written out here. So, the probability that if we're currently at x, and we're thinking about moving to x sub t, little t, and the current temperature is capital T, then what's going to happen? If the fitness of the new point is bigger than or equal to the old point, we're just going to make the move. Right? So we always hill climb when we are at a point where we can do that. >> So that's just rigorall hill climbing. >> It's kind of like hill climbing, exactly. It's a little different from the hill climbing the way we described it, where we said let's visit all the points in the neighborhood. This is kind of a useful thing to be able to do when the neighborhood's really large, just choose anything in the neighborhood, and if it's an improvement, you know, go for it. >> Okay. >> But, what if it's not an improvement? Well if it's not an improvement, what we're going to do is we're going to look at the fitness difference between the point that we're evaluating, and the point where we are now. Look at the difference between those two, divide that by the temperature, take E to that, and interpret that as a probability. And we either make the move or not with that probability. So, alright, so let's, we've gotta dive in a little bit to this expression to make some sense out of it. So, what happens, Charles, if the point that we currently visited x, then we visit a neighbor point x_t, what if their fitnesses are really close to each other, say you know, just Infinitimally close to each other. >> Well, if they're infinitesimally close to one another, then that means that difference is going to be very close to 0. >> Right, and so we get 0 over doesn't matter what the temperature is, we get 0. e^0 is 1, so we make the move if it's you know, infinitesimally smaller than, than where we are now. >> Mm-hm. Alright, what if it's a, what if it's a big step down? >> Well, then, that means that number's really, really, negative. >> Yes. >> And the negative divided by some positive number T is the temperature, so T's always greater than or equal to 0. >> Yeah, let's say that. >> Okay. >> In fact, probably making it(指 T) equal to 0 could run us into trouble. So let's just say that it's bigger this year. >> So it's Kelvin. So it's in Kelvin. Okay, so >> [LAUGH]. we, that'd be a really, really big negative number, and E to a really big negative number is 1 over E to a really big number. So that makes it very close to 0. >> Good, right. So, in other words, if a giant step down, we probably won't take it. >> Right, okay, that makes sense, so you're sort of smoothly going in directions that kind of look bad as a function of how bad they look, and sort of exaggerating that difference through an exponential function. [LAUGH] Right? >> [LAUGH] Sure. If, if that, if that gets you going. >> Well so 3 isn't 3 times worse than 1. It's you know, 2 to the 3 times worse than 1. >> I see, or E to the. Well so, so good, but let's, let's look at one more thing with this equation here. Lets, lets say that there's You know, a moderate step down. >> Mm-hm. So, it's not so huge that when we divide it by T, it's essentially negative infinity. Let's just say it's a smaller thing, you know, - 5 or something like that. >> Okay. >> Then what, what happens? Now we get some E to the that is going to be some probability between 0 and 1. What does the T, what does T do? In this case. What if T is something really big? What if, what if, what if T is something really small? >> Well, T is something really big. Let's say, infinity then, it doesn't matter what the difference is. It's effectively going to be 0. >> Right, so really big temperature here means that we're going to have some negative thing divided by something else. This is going to be, essentially 0. It's going to be E to the 0, so it's going to be 1. So, when the temperature's really high, we're actually willing to take downward steps. So, we don't even sort of notice that it's going down. >> Right. >> There's lots of randomness happening. >> Right. So, in fact, if key is infinity, it's really, really, really, really hot then even if the neighbor, is much, much worse off. He'll be less worse off than, infinity, and so, basically you're always just going to jump to the next person, to the next neighbor, to the next point (不管 f(x_t)和 f(x)誰大). >> Right. So we're very likely to accept. So the, so it's going to move freely. If T is really small, as T approaches 0 what happens? Well let's just take the extreme case. So T was 0 or effectively 0, then that means any difference whatsoever basically becomes infinity. >> Right, it gets magnified by this very, very small T. >> Right. And so then it's not going to take any downwards steps. It's only going to go uphill. Yeah, so that's kind of the essence of the Similarity Annealing Algorithm, so maybe we can talk about some of its properties. >> OK.


Properties of Simulated Annealing
T→0 : like hill climbing
T→∞ : random walk
decrease T slowly.
Pr(ending at x) = e^{f(x)/T} / Z_T
x*
Boltzmann distribution (analogy)

13. What are some of the properties of simulated amealing? No we already talked about two of them: this idea that as the temperature goes to zero, it acts, it's acting more and more like hill climbing, only taking steps that improve the fitness. And as T goes to infinity, it's like a random walk around the neighborhood. So you, wherever point where it is, it's willing to go to any of the other points in the neighborhood and it just doesn't, it doesn't really pay any attention to the fitness function anymore. >> So huh, so that makes sense right if I wanted to, to go with the analogy. High temperatures like a lot of heat, which is a lot of energy and so molecules bounce around a lot. That kind of has the effect of flattening out the fitness function. T goes to zero, that's no energy. It's like you've frozen, molecules can't move and so you only move to places with lower energy or in this case, higher fitness (即 f(x)值大的). So that all makes sense. So how do I get from one to the other? You see you had any >> You had in the algorithm decrease t, but do we need to decrease t quickly, slowly, what's the right way to do it? >> So in practice, we wanted to decrease the temperature slowly, because it gives it, the system a kind of a chance to explore at the current temperature before we start to cool it out. So, well, one way to think about it is, if you think, if you consider some kind of function that we're trying to optimize, when the temperature is high, it kind of doesn't notice big valleys. It's, it's willing to kind of wander through big valleys. As the temperature gets cooler, the valleys become boundaries and it starts to break things up into different basins of attraction, but smaller valleys it still is able to walk, walk over. As the temperature gets even lower, those become barriers as well. And so what we really want to do is give it a chance to kind of wander to where the, the high value points are before cooling it so much that it can't now bridge the gulf between different local optimums. >> Ok, that makes sense. So, property as it's sort of bouncing between exploration and exploitation. I buy that. So where do you end up? Where do you end up? >> Well, if we're lucky, we end up at the global optimum; yay. >> [LAUGH] Is there anything, you know, we could say that we could characterize how often we end up there, or are we going to have to do randomized restarts and do simulated annealing a million times? >> So, we're not

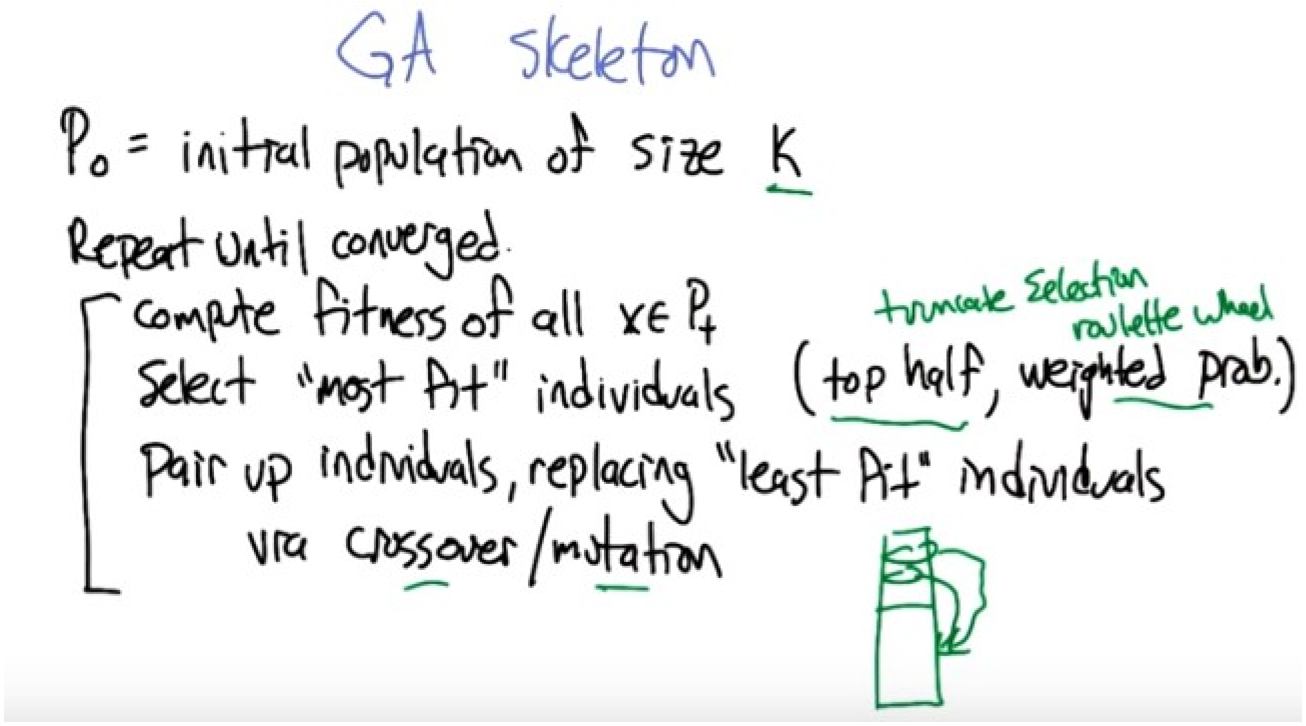
going to be able to go through the argument for exactly why this is true, but there is a remarkable fact about simulated annealing that is worth mentioning. That the probability end at any, any given point x in the space is actually e to the fitness of x divided by the temperature and then normalized, because this is a probability distribution over the input space. So this is, this is pretty remarkable right? So this is saying that it's most likely to be in the places that have high fitness, right? because that's those are where it's going to have the highest probability of being. And you can see now what the relationship with the temperature is here too, that as we bring the temperature down, this is going to act more like a max. It's going to put all the probability mass on the actual x*, the, the, the optimum. And as the temperature is, is higher, it's going to smooth things out and it'll, you know, be randomly all over the place. >> Hm. >> So, so that's why it's really important that we eventually get down to a very low temperature. But if we, again, if we get there too quickly, then it can, it can be stuck. >> Hm. >> Right, because it's spending again, time proportional to the fitness. So, it is spending time ending up at points that are not the optimum, if they're say, close to the optimum. >> Okay, so you know, I actually now that you, you spell that out, I actually recognize this distribution, it has a name it's called the Boltzmann distribution (這都能聯係上, 好神奇) (Z 是不是就是配分函數). >> That's exactly right, and if there's people listening that have a physics background, they're going to probably be poking their eyes out because, huh, it's the case that we, we're using a number of physics concepts, but we don't quite use the same notation and we don't quite do things exactly the way that a physicist would. So, there's definitely a relationship and the Boltzmann distribution is used quite a bit in, in the physics community, but it may not be exactly like that so don't, try not to panic. >> Yeah, it's more like an analogy, like so much of life. It's a simulated Boltzmann Distribution. >> It's like a simile. >> [LAUGH]


Genetic Algorithms
Population of individuals
Mutation - local search N(x)
cross over - Population holds information
generations - iterations of improvement
dimension 2
dimension 1
Population like parallel random restarts

注意是 Genetic, 而不是 Generic.

14. All right. There's one more class of randomized optimization algorithms that is really attractive. It's, it's, the, it's very interesting to people. It has proven itself time and time and again. And it's this notion of genetic algorithms. So the main insight that's exploited in the genetic algorithm setting is this. So let's imagine we've got a two dimensional space. And, it's you know, hard for me to actually draw a, a fitness surface over a two dimensional space. So just kind of think of this as being one of those maps.,

those contour maps (即等高線). And so imagine that we've got, one dimension that X now comes in, in these two different dimensions. And What we're trying to do is find the peak, which happens to be there. So what if, we actually evaluate 3 different points, so these green points here, we actually check what the values are at these points. So what we find is that, from this initial point (右下的點), this green point here. If we increase our dimension 2, we get a better value. But, it's also the case. It's starting from that point (還是右下的點). If we increase on dimension 1, we get a better value. So, maybe, what we, we ought to do, is take kind of elements of these 2 solutions, these 2 inputs and combine them together, and move out on dimension 1 and dimension 2, and maybe that will actually give us a good score as well. And in this particular case, it puts us in the base interaction of the local maxima. So, this turned out to be useful in many spaces, especially spaces that can be specified combinatorially like this. Where there is this separate dimensions that contribute in various ways to the overall fitness value. >> Ok Michael, that sort of makes sense. But, what does this have to do with genetics or algorithms for that matter? >> Well, it's an algorithm in that we're doing, it's an optimization algorithm, and the genetic part, is because what we're going to do, is were going to build an analogy with biological evolution. >> Mm, analogies. >> In particular, instead of thinking about these input points, these little green dots, we're going to think of them as each input point is like an individual, and a group of them taken together is like a population. >> Mm-hm. It's really the same idea, but we're just giving it a different name. Okay. >> The the idea of local search where you make little changes to a, to an input, we're going to now call that mutation. All right? Where you take an individual and kind of tweak it a little bit >> Oh, like we did in the the, the example we did before where we define the neighborhood as every one, difference in every single bit. That's right. So, so the mutations can happen along neighborhoods. It's the same kind of concept as that. >> Okay. >> And then, you know? And, and, you can see that there's, the mutations happening over X. So, I assume that you get X [UNKNOWN]. >> [LAUGH] I think that's a fair point. I like your science. >> Yeah. [LAUGH] Yeah, that's right. It is science. The, those are all concepts that we were already using when we were doing these other randomized optimization algorithms. One thing that's different though, is the notion of crossover. So, what crossover does, is it takes different points, like this green point and this green point and instead of moving them just to their own neighborhood, it gives you a way of combining their attributes together with the hope of creating something even better. So, that is where it starts to actually kind of deviate from the standard notion of local search or randomized optimization. And gets us into something that, that feels a little more like evolution. So, this is kind of like. Dare I say sexual reproduction. Right, where the two parents can actually form a new kind of offspring that you know, if you're lucky is has all the positive attributes of both of the parents. Like my children. >> Uh-huh, and if you're unlucky, it has the worse attributes of the parents, like other peoples children. >> [LAUGH] Exactly. And so, and finally what we were calling iteration before in the context of genetic algorithms, we can call it generation. Because we're going to take, a population of individuals and kind of you know, mate them together to create a new population of individuals. And we're going to, what we hope is improve iteration by iteration. >> Okay, that makes sense. So, If I can just push on this a minute, it seems like, if it weren't for crossover, this is really like doing random restart, except instead of doing restarts you just do 'em all at once cause we have parallel computers. >> Yeah, I think that's I think that's fair. I think that's quite fair actually. Okay. So then the thing that makes it more than that is, crossover. That somehow these parallel, random searches are bleeding information back and forth, help maybe bleeds the wrong. >> [LAUGH] Yeah, you don't want to get too biological about this. >> Yes, right. Well, so they're sharing fluids, metaphorically, with one another >> [LAUGH]. And conveying information that way, just the way genes do. Right. And then, and so that's the sort of interesting concept that now we have information, not just in the individual, like we're moving that one individual around and trying to find better and better solutions, but the population as a whole, represents a distribution over individuals. And that, that distribution might actually be a useful thing to guide the search for higher scoring individuals. >> Okay. That, that makes sense.

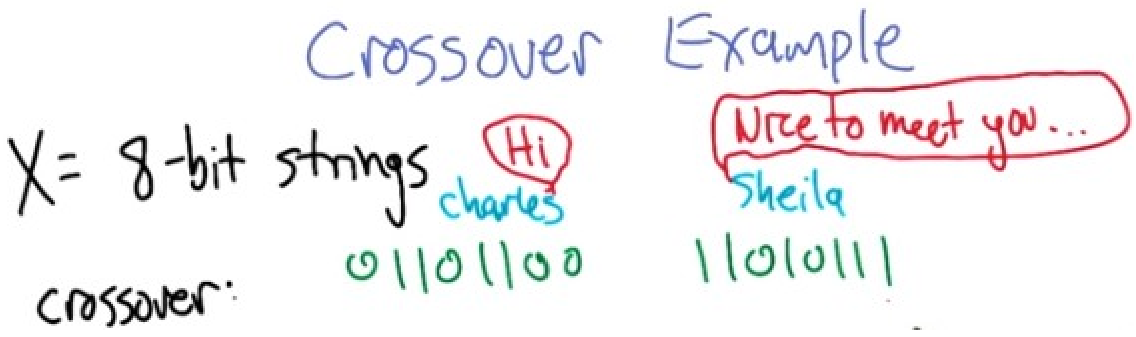


GA skeleton

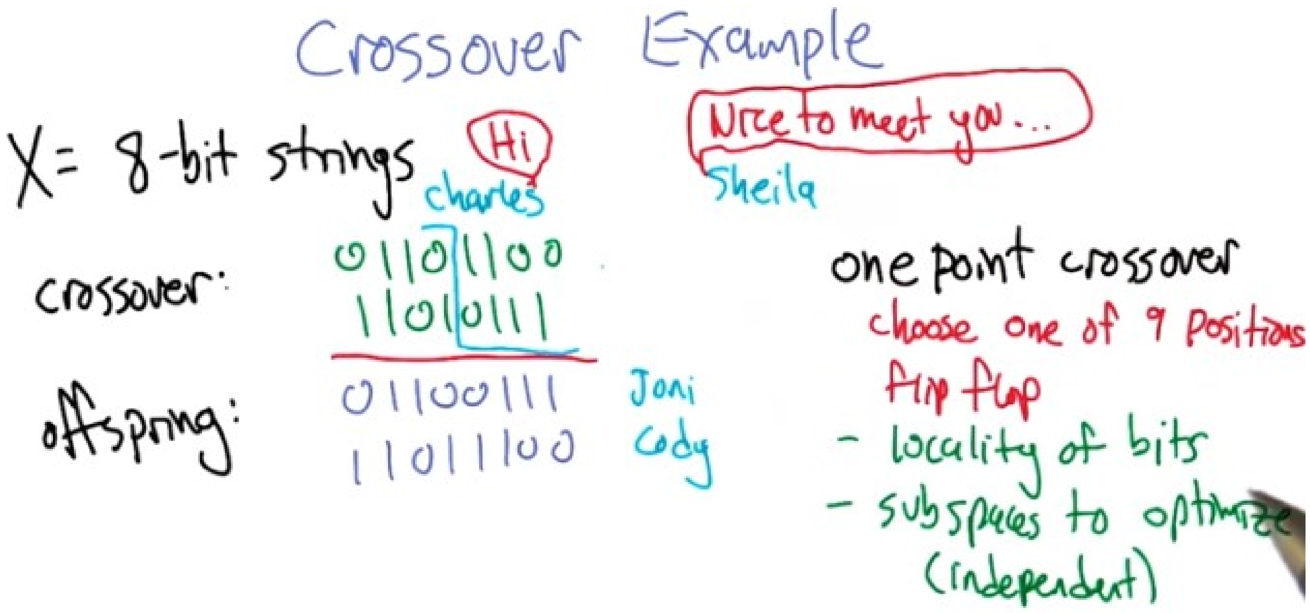
```
P_0 = initial population of size K
Repeat until converged
    compute fitness of all x ∈ P_t
    Select "most fit" individuals (top half, weighted prob.)
    Pair up individuals, replacing "least fit" individuals
        via crossover/mutation
```

15. >> Here's a skeleton of the al, of an algorithm that implements a, a GA. So, [COUGH] what we need to do is we start off with some inital population and usually the population size is fixed to some constant. We'll call it k. So we just generate a bunch of, of random individuals to get things started. Then what we're going to do is we're going to repeat until essentially things converge. We're going to compute the fitness of all of the individuals in the population. And then we're going to select the most fit individuals. So, how do you suppose we might decide on which ones are the most fit? >> The ones you're most likely to go to the prom with? >> Some people choose people differently than just whether they're fit. I guess it depends what you mean by fit. Do you mean like fit, like physical fitness? >> Well, I meant fit like well, I did. But I guess fit, like, whatever the fitness function tells you is fit. >> Great! Okay, so, since we applied the fitness function f to all the individuals we have scores for them, and if we want to select the ones that are most fit, which ones do you think those would be? I guess you're saying those would be the ones with the highest scores. >> Yeah. >> So that is, that's definitely one way to do it, where what happens is you take the, say, top half of the population in terms of their scores, and we declare them to be the most fit and everybody else to be the least fit. But there's other ways you can do it as well. One, this is sometimes called truncation selection. But there's also an idea called roulette wheel selection where what you do is you actually select individuals at random but you give the higher scoring individuals a higher probability of actually being selected. So we don't just strictly choose the best ones, we choose weighted by who's the best. >> So does this get back to exploitation versus exploration then? >> I think it does. I think it, I was certainly having that thought when I was saying it out loud. That this is a, it's a similar kind of idea and in fact you can use Boltzmann distribution type ideas similar to the annealing type idea for doing this selection where you have a temperature parameter. If you set the temperature parameter to zero then you get something like just choosing the top half. And if you set the temperature to be something like infinity then you're just going to randomly choose samples from the population irregardless, no. >> Irregardless is a word. >> Is it really? >> Yes it's actually a word. >> Irrespective is what I wanted to say. Irrespective of the fitness of those individuals. >> Well irregardless I understand what you mean. >> Alright, so then

that's going to, we declare some of them as most fit. Then what we're going to do is pair up those most fit individuals. This is like a dating service now. And let them produce offspring using crossover and maybe a little bit of mutation too. So instead of just taking the combination of the two parent individuals, we take their combination and then we make little, little local changes (相當於基因突變) to it, to mute, mutate them. And we let that, the, the value of that pair that, that new offspring replace one of the least fit individuals in the population. >> So Michael, can I ask you a question? >> Sure. >> I'm having a hard time what is crossover means. Can you draw me a little picture? >> Sure. That we, this might be, you know rated R. >> That's fine. >> No, no, no. No, you're right because we're going to do it well so, it turns out that this is sort of generic, we can define crossover in many different ways. But I think you're right, I think it's worth looking at some concrete examples because it gives you some insight into why this operation might be useful. >> Okay.

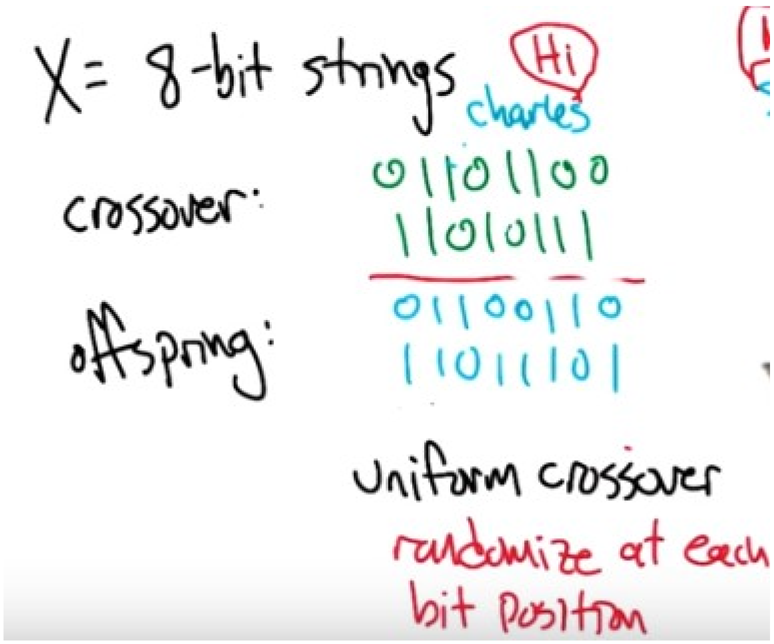


16. So let's do a concrete example of crossover. And so, it turns out that the crossover operation is always going to depend critically on how you represent the input space. So, let's say concretely that our input space is, is eight bit strings. So here's two parents, 01101100 and 11010111. And we're going to introduce them to each other. All right, now they know each other. >> Mm-hm. >> And now we're going to use them to create a new individual, a new offspring. >> So they're really going to know each other.

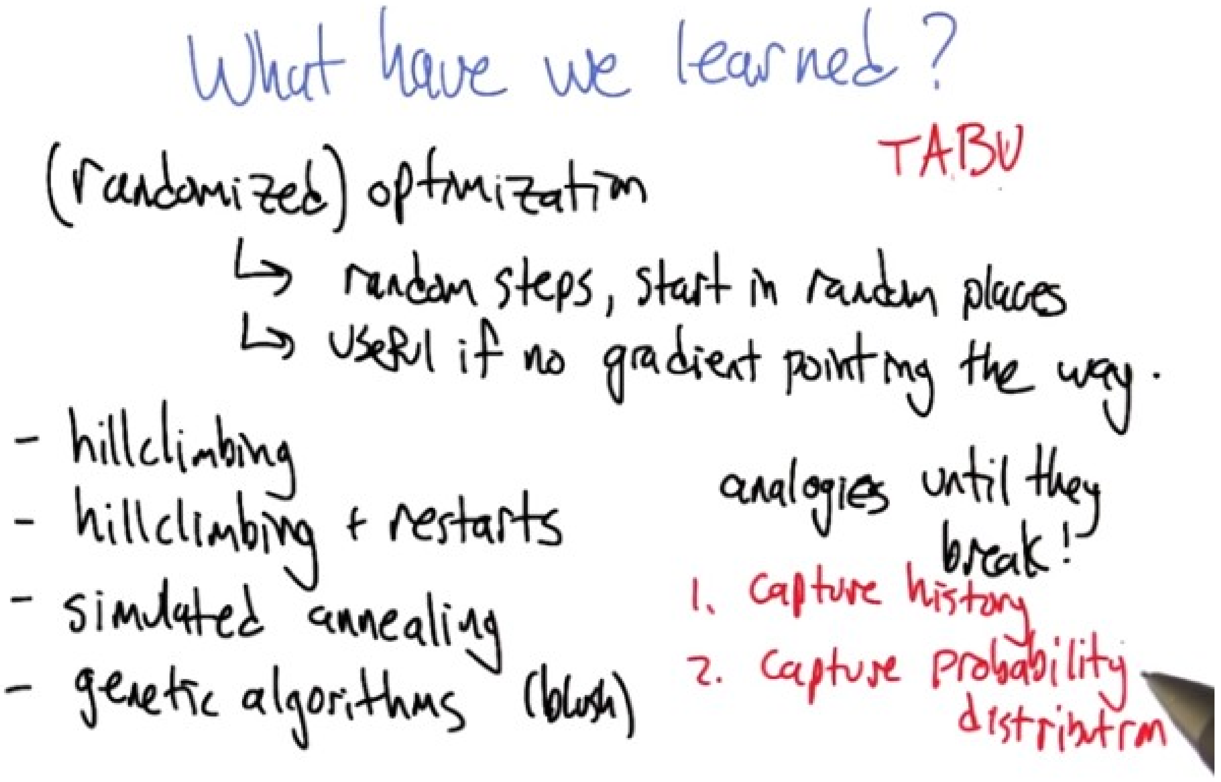


>> All right. Now we've put these two bit sequences together and we've lined up so that the bits correspond in each of the different positions and now we can ask. How can we use this? To generate a

new individual that uses elements of the two individuals that we have. So if you can think of any ways to do that. >> I have some ideas. [LAUGH] >> Yeah okay, what you got? >> I have lots of ways but I don't think any of them can be reproduced for the purposes of this learning lesson. >> All right well so let's, let's do ones that really kind of stick to the bit patterns. >> [LAUGH] I swear there's no way you can say this without getting in trouble Michael. All right. >> But that's okay. So how about, here's an obvious one. Right, if we really push the genetic notion as far as we can then each of those things represent, I don't know, alleles or some other biological term. And so what happens in genetics, right, is you mix and match your chromosones and alleles together. So why don't I say one child is the first four bits of this handsome Charles fellow and the last four bits of this beautiful and wonderful Sheila person (此即基因重組). >> Alright, I see what you're doing there. So what you're saying is we're going to pick, well maybe this isn't quite what you said but I'm going to imagine what you said we're going to pick a random number along the sequence at the half-way point. And what we're going to do is now mix and match and create two offspring. One uses the first half of Charles and the second half of Sheila and then the other one is the other way around. And as you can see it's this last bit that determines the sex. Anyway, so these are the two offspring that these individuals have generated. And this particular way of, of combining where you randomly choose a position and then flip flop, is called one point crossover. Alright, so now I want you to think about this for a second Charles. So I don't know if it's an inductive bias, but what kind of bias do we put in? When we say well we're going to choose one of these points and we're going to flip flop based on where that point is chosen. >> What is that, what is that going to, what kind of offspring is that going to generate? >> Huh. Also, you know what, I see two, I see two kind of assumptions built there. So, maybe that an inductive bias, so. Or a bias of some sort. So, one assumption is that locality of the bits matter. Good. >> Right. So the first by picking halfway through, you are saying, the first four bits are somehow related and the last four bits are somehow related because otherwise they wouldn't make any sense. Now to talk about them being together and that brings it to my second point which I guess really is just a first point. Which is that it assumes that there are subparts of the space that can be kind of independently optimized that you can then put together. Right, and in particular when I say sub spaces to optimize, I mean that they're independent part of the subspace, so that's actually the example that you gave before you said Well there's these two dimensions and each dimension kind of matters independently and the total reward or the total, the total fitness is some kind of linear combination of them. And so I can put them, cause if those two things aren't true than really doing crossover like this won't help you at all. You're just kind of randomly. Mixing things together. >> It's kind of an assumption about the way space works, in that, kind of like the example we did when we were doing bit guessing. That that you can be heading in a good direction, that there're pieces that are right and if we reuse those pieces we can get even righter. >> Sure. >> Alright so, and if it is the case that the sequence of the ordering of bits matters, we have this locality property. This is actually a fairly sensible thing to do.

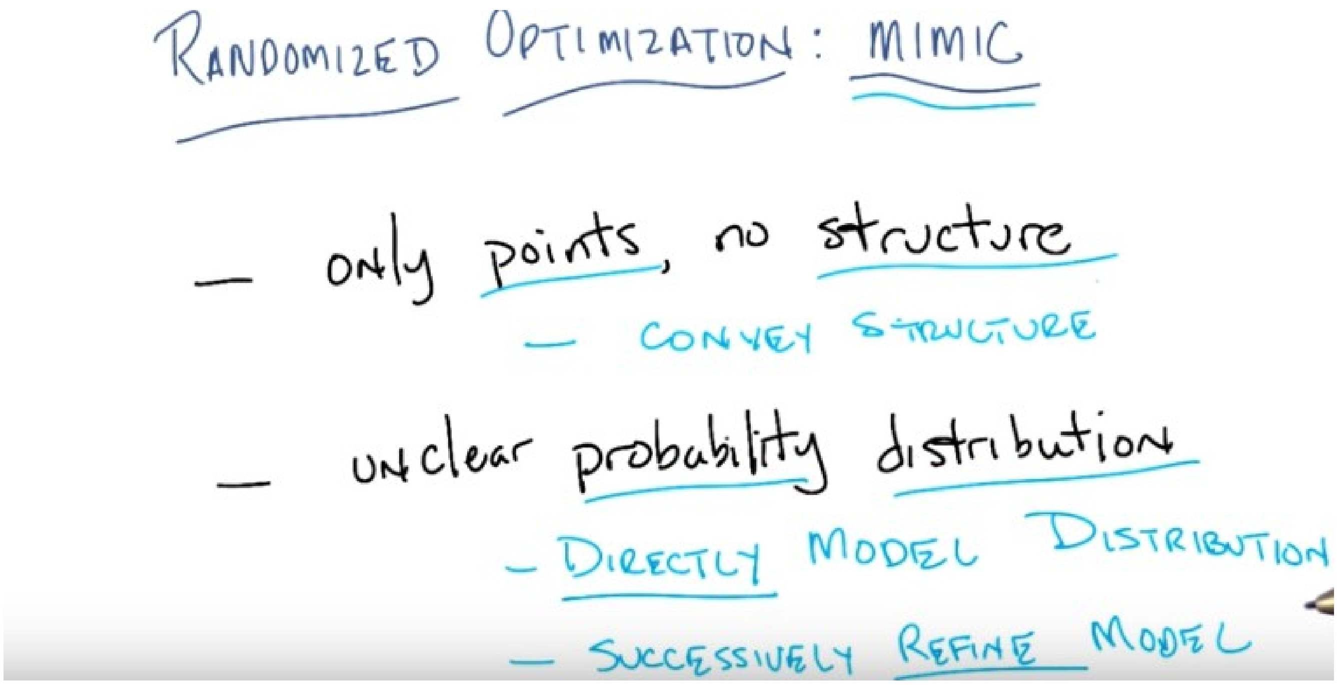


But can you imagine any other way of combing these bits together to get to get offspring? >> Well, I can think of lots. >> Well, so let's, let's focus, you know, you have many different possible ideas, but let's focus on ideas where we still have this subspace to optimize property. But we don't really have a locality of bits property. We don't really, the ordering doesn't matter anymore. So keeping them clumped together like that. We don't think that that's a useful thing. >> Okay. Well, what would that mean? Tell me what that means. >> Well, so, The one point crossover, when we talked about that. It really matters that you know, the two bits that are next to each other are very likely to stay connected, right? That is, it's, it's unlikely that the split will happen to happen exactly between them and so we'll tend to travel as a group. But, if we don't think it's important that the bits that are next to each other need to travel together. If we say that It should be equally likely for any of the bits to kind of remain together. We need to cross over a lot more than just that one time. In a sense we might need to cross over every time. Well so, what I'm trying to get at here is this notion that what we could do is we could generate individuals by just scrambling at each bit position. >> Okay. >> So. The first bit position, maybe which stays the same, the second one flips (flip 意思見下), the third one stays the same, the fourth one stays the same, the fifth one flips (flip 意思為: 約定男上女下, 先將爸爸的全部基因給兒子, 媽媽的全給女兒, 但 fifth one flips 使得 爸爸的第五個基因給女兒, 媽媽的第五個基因給兒子), the sixth one stays the same, the seventh one flips, and the eighth one stays the same. So now, we've got two individuals, and every bit from these individuals comes from one of the parents and so that means that if there is sub pieces that are current that maybe preserved in the offspring but no longer does it matter what the ordering is. We get exactly the same distribution over offspring, no matter how we order the bits. >> Okay. >> So this idea is sometimes called uniform crossover. And essentially, we are just randomizing at each bit position. This kind of crossover happens biologically at the level of genes right so we, we imagine that we get our genes from our parents but the, for each different gene like the gene for eyes and the gene for hair color are not particularly linked (即它們可以隨便來自父母哪方) to each other they're uniformly chosen at each position.


What have we learned?
TABU
(randomized) optimization
↳ random steps, start in random places
↳ useful if no gradient pointing the way.
- hillclimbing
- hillclimbing + restarts
- simulated annealing
- genetic algorithms (blush)
analogies until they break!
1. capture history
2. capture probability distribution

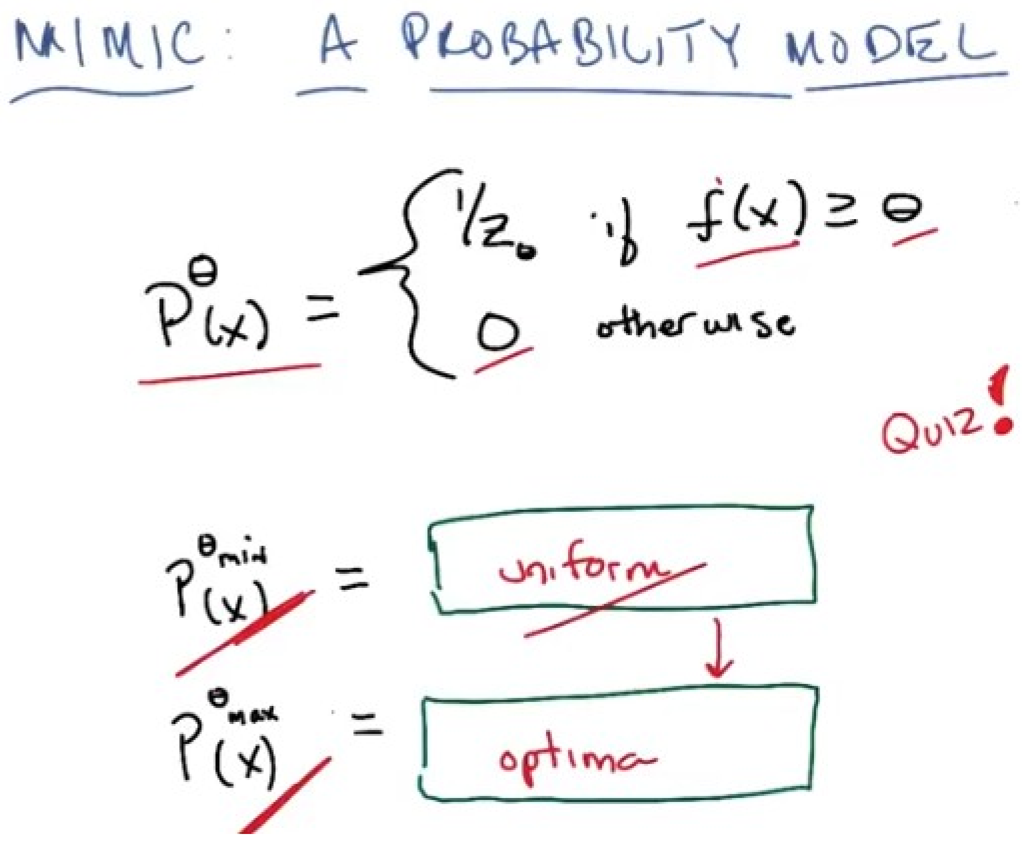
17. So that's really all I wanted to say about genetic algorithms. I mean, there's lots of tweaky things that you need to do to get this to work very effectively. You have some choice about how to represent the input, and you have some choice about how you can do your selection, and your fit to finding your fitness function. But, at a generic level, this is, this is, it's a, it's a useful thing. Some people call genetic algorithms the second best solution to any given problem. >> Hmm. >> So, it's a good thing to have in your toolbox. But I think that's, that's really it. That's all I wanted to say about randomized optimization. So what have we learned? >> Well, we learned about random optimization period. That that there is a notion of optimization in the first place. >> So, we talked about optimization in general and then we, what was the randomized part? >> Well, we'd take random steps where we start off in random places. Or, we'd do random kind, well actually that's really it. You take random steps, you start off in random places and it's a way to overcome when you can't take a natural gradient step. >> That's right. So did we talk, and we talked about some particular randomization. Er, sorry, randomized optimization algorithms. >> Let's see. There was randomized hill climbing. >> And we had 2 flavors of that. >> Right. We did simulated annealing. And, we did genetic algorithms. >> And don' t forget, that we talked a little bit about how this all connects back up with learning, because in many cases, we're searching some parameter space to find a good classifier, a good regression function. A later in this particular sub-unit we're going to be talking about, finding good clusterings. And so, this notion of finding something that's good, finding a way to, to be optimal is pervasive through apache learning. >> Oh that make sense. Well, there, well, if we're trying to remember all these other things we learned. We also learned that AI researchers like analogies. >> [LAUGH] >> Both simulating annealing and generic algorithms are analogies. And they don't just like analogies, they like taking analogies and pushing them until they break. >> Indeed, actually hill climbing [LAUGH] is an analogy too. >> Yeah, actually, right? Every single thing that we did is an analogy to something. Okay, that's good. The other thing that we learned, which I think is important, is that there's no way to talk about cross, crossover without getting a bunch of people in the studio to giggle. [LAUGH] >> Yeah, genetic algorithms make people blush. >> Okay, that's pretty good, but Michael you know, I'm, I'm looking at

all this and now that you put all these words together on one slide, I have 2 observations I want to make. That that are kind of bothering me. So, one is, I'm looking at hill climbing that makes sense, hill climbing restarts makes sense, simulated annealing makes sense, [INAUDIBLE] but you know what, they don't really remember a lot. So, what do I mean by that? So you do all this hill climbing and you go 8 billion steps, and then what happens? You end up with the point. You do simulated annealing. You do all this fancy stuff with slowly changing your temperature, and creating swords with black smiths, and all that other stuff you talked about and in the end, at every step, the only thing you remember is, where you are and maybe where you last were. And with genetic algorithms, it's a little more complicated that because you keep a population, but really you're just keeping track of where you are, not where you've been. So in some sense, the only difference between the 1 millionth iteration, and the 1st iteration is that you might be at a better point. And it just feels like, if you're going to go through all this trouble of going through what is some complicated space that hopefully has some structure, there should be some way to communicate information about that structure as you go along. And that just, that sort of bothers. So that's one thing. The second thing is, what I really liked about simulating annealing, other than, you know, the analogy and hearing you talk about strong swords, is that it came out at the end with a really nice result, which is this Boltzmann distribution, that there's some probability distribution that we can understand, that is actually trying to model. So, here are my questions then. It's the long way of asking a real simple question. Is there something out there, something we can say more about? Not just keeping track of points, but keeping track of structure and information. And is there some way that we can take advantage of the fact that, all of these things are somehow tracking probability distributions just by their very nature of being randomized. >> That's, that's outstanding. Yes, very good, very good question. It is true that these are all kind of amnesic, right? They kind of just wander around, and forget everything about what they learned. They don't really learn about the optimization space itself. And use that information to be more effective. There are some other algorithms that these are, these are kind of the simplest algorithms, but you can re-combine these ideas you know, sort of cross-over style, to get other more powerful algorithms. There's one that's called TABU search, that specifically tries to remember where you've been, and you're supposed to avoid it. Right, they become TABU regions. To stay, with the idea that you should stay away from regions where you've already done a lot of evaluations, so you cover the space better. And then there's other methods that are, have been popular for a while that are gaining in popularity, where they explicitly model the probability distribution over where good solutions might be. So they might be worth actually talking a little more about that. >> Okay, so go ahead. Well, so I kind of added time. Maybe maybe you could look into this, I can give you some references and maybe you can report back. >> Fine. >> [LAUGH] You'll learn, you'll learn very well by doing that. >> Yes sir. >> [LAUGH] Alright, we'll see you then, then. >> Okay, I'll see you then, then too. Bye, Michael. >> Bye Charles.


RANDOMIZED OPTIMIZATION : MIMIC
– only points, no structure
– CONVEY STRUCTURE
– unclear probability distribution
– DIRECTLY MODEL DISTRIBUTION
– SUCCESSIVELY REFINE MODEL

18. Here's what I like to do. You pointed out some issues that you were concerned about and I thought that maybe you could go and look into it a bit more and you did. And so why don't I turn things over to you so that you can tell us what you've found out. >> Okay, well thank you Michael. Hi again. >> Hi. >> Thank you Michael. So I did go and I started trying to deal with these issues. So just to recap a little bit, there were two problems that I had, more or less. And they were, that we had all these cool little randomized optimization algorithms. And, most of them seemed to share this property that the only thing that really happened over time, is you started out with some point and you ended up with some point, which was supposed to be, you know, the optimal point. And the only difference between the first point, and the last point that you did or, the one millionth point or however how many you iterations you had, is that, that point might have been closer to the optimum by some measure. And very little structure was actually being kept around or communicated. Only the point was being communicated. Now you could argue that that isn't quite true with genetic algorithms, but really you move from a single point to just a few points. The other problem that I had is that we had all of this sort of probability theory that was underneath what we were doing, all this randomization. But somehow it wasn't at all clear in most of the cases, exactly what probability distribution we were dealing with. Now, one thing I really liked about some [UNKNOWN] is that you were very clear about what the probability distribution was. So what I decided to do is to go out there in the world and see if I could find maybe a class of algorithms that took care of these two points for us. And I found something, you'll be very happy hear, Michael. >> Yeah, I would love to, to find out what it is. >> It turns out that I wrote a paper about this, almost 20 years ago. >> [LAUGH] How did you find that? >> I just said, well if I wanted to start looking someplace, I should look at home first. And I stumbled across >> Oh. >> this paper that I wrote. >> I see, I see. So learning about machine learning, really begins at home. >> That's exactly right. So, I had to re-read the paper because, you know, it is a coupl of decades old. And I will point that a lot of other work has been done since this that refines on these ideas. But, this is fairly straightforward and simple, so, I think I am just going to stick with this work. And the paper's available for everyone listening this to right now or watching this right now. So you can read it and all of it's gory details. But I just want to go over the, the high level bit here because I, I, I really think it kind of gets at this idea. So, in particular, the paper that I'm talking about introduced an algorithm called Mimic, which actually stands for something, though I forget what. And it really had a very

simple structure to it. The basic idea was to directly model a probability distribution. >> Probability distribution of what? >> Well, I'll tell you. And, you know, like I said, Michael, I will, define exactly what this, probability distribution is for you for a second and, and hopefully you'll, you'll buy that it seems like a reasonable distribution to model. And given that you have this, probability distribution that you're directly modeling, the, the goal is to do this sort of search through space, just like we did with all the rest of these algorithms. And to successfully refine the estimate of that distribution. >> Hm. >> And the idea is that if you can directly model this distribution and refine it over time that, that will in fact convey structure. >> Structure in particular of what were learning about the search space while we're doing the search. >> Exactly, and not just simply the structure of the search space, but the structure of the parts of the space that represent good points or points that are more optimal than others. Yeah, that seems like a really useful thing. >> So I'm just going to give you, again, this, this simple mimic algorithm that, that sort of captures these basic ideas, because I think it's fairly simple and easy to understand, while still getting some of the underlying issues. But do keep in mind that there's been literally decades of work since then, and optimization space where people have really taken these kinds of ideas and refined them to be sort of much more mathematically precise. But this, I think, does get the idea across, and I happen to understand it, so I thought that I would share it with you. >> Hm. >> Seem fair? >> Yeah, that sounds really exciting. >> Excellent.

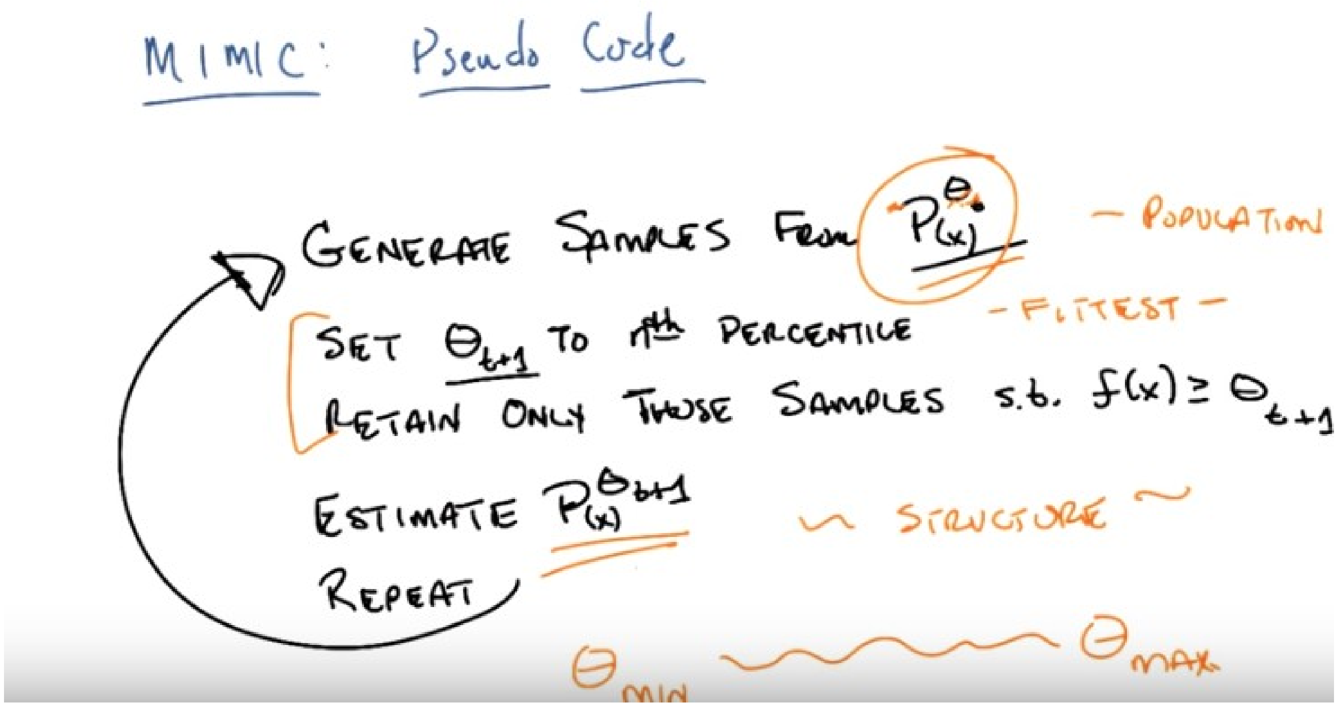

MIMIC: A PROBABILITY MODEL
$P^{\theta}(x) = \begin{cases} 1/Z_\theta & \text{if } f(x) \geq \theta \\ 0 & \text{otherwise} \end{cases}$
Quiz!
$P^{\theta_{min}}(x)$ = uniform
$P^{\theta_{max}}(x)$ = optima

P^max (x) = optima 的意思為 P^max (x) = 1 / (optima 個數).

19. Okay, so here's a probability distribution I'm going to define for you Michael. You'll notice that it's a probability over our Xs. And it's parameterized by θ. So θ is going to turn out to stand for threshold. Okay? Got it? >> Mmhm. >> Okay, so here's a probability distribution. It is 1 over z sub theta. For all values of x such that the fitness function is greater than or equal to theta. >> Yeah. >> And it's 0 otherwise. So do you understand that? >> I think so. So I assume that Z_θ here is, is going to be some

kind of normalization factor that accounts for the probability of landing or choosing an x in that space. In the space of high-scoring individuals above threshold. >> That's exactly right. So another way of saying this is that the this probability is uniform over all values of x whose fitness are above some threshold. >> Yeah so, I'm, I'm, the (此話說得好:) image I have in my hand is kind of like a mountain range and if you put a slice through it then everything that's kind of above that slice, if we are going to sample uniformly from that, that collection. >> That's right. And everything below it, we will simply ignore. >> Doesn't happen, it doesn't get sampled form. >> Exactly. Okay so because you demanded it Michael, here's a quiz. >> [LAUGH] Alright. >> So, two questions in this quiz. The first question is, θ is some threshold and so let's imagine the fitness function has to have some minimum value and some maximum value. Okay? And let's call those theta submin and theta submax respectively. And so I want you to describe in one or two words P sup data min of x. That is the probability distribution whether threshold is its minimum value, and P sup data max of x. That is, the distribution where theta is at it's maximum value. You got it? >> So, just to, just to be clear, sort of the maximum meaningful and minimum meaningful values >> Right. >> Because I can imagine them, you know, if it's just sort of way outside the range of the fitness function then it's kind of trivial, I guess. >> Right, well, let's, it is, but let's assume that θ_min is the lowest value that the fitness function can take on and θ_max is the largest value of the fitness function. >> Okay, yeah, okay, that's how I was imagining it. >> Okay. >> Okay, yeah, I'm good to go. >> Okay. So go!

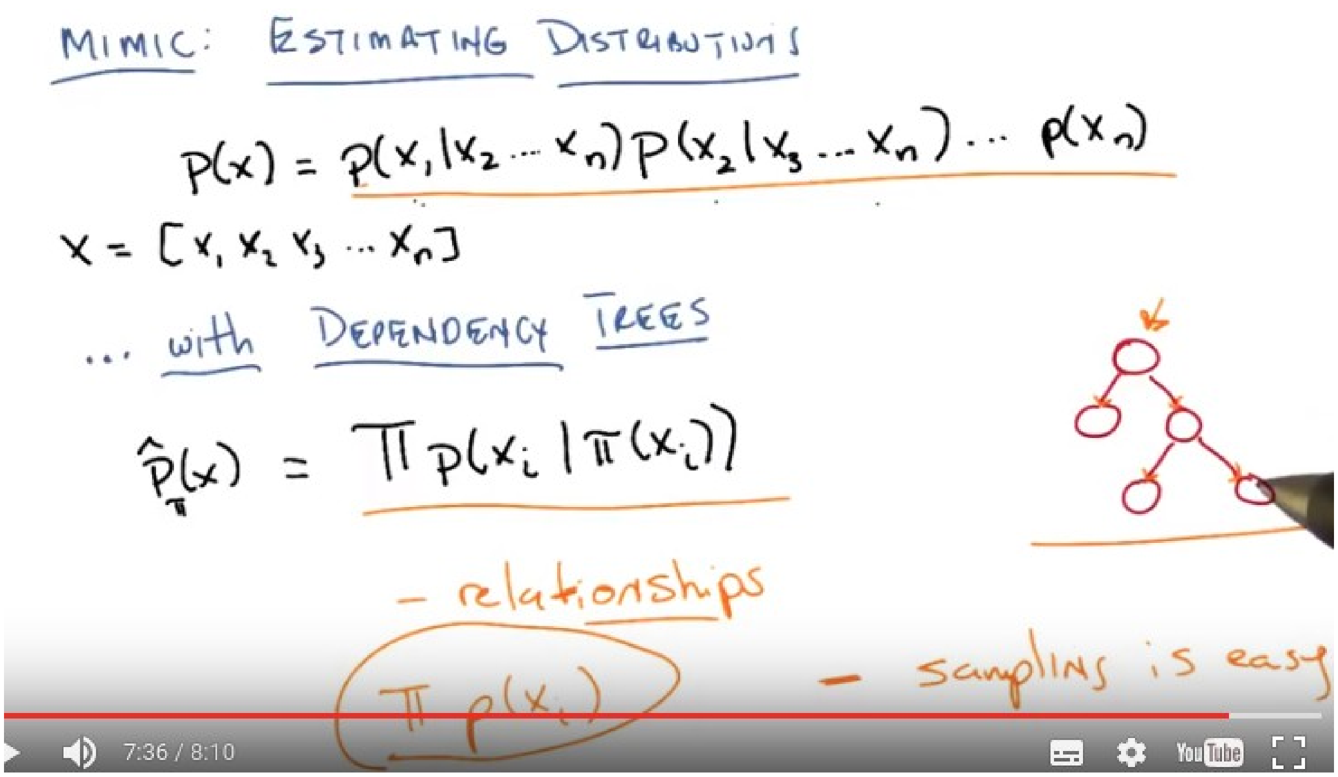
20. Okay Marco, you got an answer for me? >> Yeah, yeah. I, this is, the way I was thinking about it was essentially, well for the theta min. Wait, so we're maximixing, right? >> yes. >> All right. So θ_max, then, is going to be the largest value if, that f can return. >> Yep. >> Which is actually the optimum of the function. So that probability distribution assigns a probability of well one, if there's a unique optimum, it'll decide a probability of one to that single point. >> That's exactly right, but you were getting at something else when you said if there's one optimum. What if there are multiple optima? >> Then it's uniform over all of them. >> Right. So, it is the distribution that generates only optimal points. So, it's a distribution over optima, which is just you said and we'd accept optimum point or anything that sort of captures this idea. >> Okay. >> Okay? Well, what about theta submit? >> So, if it's the minimum that the function can achieve then it, it ought to be the case that everything in this space of X's is part of that. So it should assign uniform probability to all points in the input space. >> That's exactly right. So it is in fact, simply, the uniform distribution. >> Nice. >> Now, this is going to be pretty cool, because in this, in this slide right here we basically have the mimic algorithm. >> I'm not seeing it. >> Well I'm about to tell you, we are basically going to try to estimate this particular distribution; P sub theta of X. We're going to start out with P^min (x), P sub theta min of x, which is the uniform distribution. So we're going to sample uniformly from all of the points, and then somehow use those points to do better and better, and better and better estimates until we get from uniform to a distribution of only the optimal points. And that's the basic idea. >> So, okay, so, and, and the first one being something that's really easy to sample from, because we're just sampling uniformly from the input space. >> Right. >> And the second being something that'd be really useful to sample from, because if we could sample from the set of optima, it's really easy to find an optimum. >> Exactly. So we'll start out here. And we're, the goal is to start out here (P^min (x)) and then to end up here (P^max (x)). Gotcha. >> And so let's actually write out a out rhythm that's does that. >> Yeah, because that's not an algorithm that's just a goal. >> But many algorithms start out as goals. >> Hm.


MIMIC: Pseudo Code
Generate Samples From $P^{\theta_t}_{(x)}$
- Population
Set θ_{t+1} to nth Percentile
- Fittest -
Retain Only Those Samples s.t. $f(x) \geq \theta_{t+1}$
Estimate $P^{\theta_{t+1}}_{(x)}$
~ Structure ~
Repeat
θ_{min}
θ_{max}

Percentile: 百分位数. 一组 n 个观测值按数值大小排列，处于 p%位置的值称第 p 百分位数。

21. Okay, so Michael here's some pseudocode that represents an algorithm of the sort I just described after the last quiz. So here's the basic idea, just see if you can picture this. We have, were in the middle of this process, we have some data at sometimes, step t, and were simply going to generate samples that are consistent with that distribution. So, we're going to shoot our probability distribution is not just something that gives us a probability but something from which we can sample. So, generate samples according to P^θ_t (x) (前面已給出了 P^θ (x)的表達式). Generate a bunch of those samples and now that we have these samples, we're going to come up with a new theta. T plus one. And that θ_(t+1) is going to be the (value of) best samples that we just generated. So, let's say the top half (或圖中的 nth percentile). Now, you'll notice that this should actually remind you of something. Does it remind you of any of the other randomized algorithms that we've looked at so far? >> Maybe a little bit like simulated annealing. >> No. Hm. >> Because it does, you know, change the probability of going up. What we're choosing. Oh, we're choosing different percentile. That's a lot like genetic algorithms. >> Right, it's exactly like genetic algorithms. Except instead of having this population that moves from one bit to other, we generate samples that's like our population. So, we generate a population here. We pick the most fit of that population by retaining only those that are the best ones. And all the stuff that you said before about, well maybe you don't want to pick the best ones. Maybe you want to sample it according to a probability distribution proportional to the fitness. All those things you talked about before would apply here, but let's just stick to the simple case where you, you take the best ones. And, now that we've got this new population, rather than using that population again, we estimate a new distribution that's consistent with those. And then we just lather, rinse, repeat until we come to some conversions. >> Okay, so let me just make sure I'm following. First of all, when you say P sup, you don't mean the food. >> I do not, because I do not like pea soup. >> And you don't mean sup meaning like the largest possible value of theta, because when I was hearing p-sup theta, I was, I kept thinking that you mean the best theta. >> No, soup is short for superscript. [CROSSTALK] When people say p-sub-i, which means subscript. >> Oh, yeah, okay, that, that makes sense. >> Mm-hm. >> And, okay. So now now that's [LAUGH] that's clear. The, this notion of estimating a probability distribution, I guess that's where it feels really fuzzy to me. because if you do the estimate as a kind of, I don't know,

sort of particle filtery kind of thing, where you just keep instances. >> Mm-hm. >> Then it feels really, a lot like the genetic algorithm. >> Right. Except, remember, one of the things that we cared about, a structure. So this structure that we're maintaining remember this, this complaint that I had that we weren't somehow keeping track of structure in a nice way? >> Yeah. >> Is all going to be hidden inside the how we represent these probability distributions. That is going to be the structure that moves from time step to time step rather than just the population of points moving from time step to time step. And I'll, I'll get into some details about that in the very next slide, I just want to make certain that you understand. The high level bit here. >> The high level bit. >> And the high. >> Yeah, I mean I am not, I am not connecting it with theta in the sense of the previous slide. >> So theta is our threshold, right? So, basically, we have some distribution. Now, remember what is P superscript theta? It is you know, it is a probability distribution that is uniform over all points that are, have a fitness value that is greater than theta, greater than or equal to theta. Right. So I have some distribution here. If I just generate samples from that distribution then what this means is I'm going to be generating all the points whose value, whose fitness is at least as good as theta. And then I'm going to take from those the set of points that are much higher than theta, hopefully and use that to estimate a new distribution. And I'm just going to keep doing that. And what would should happen is, since I'm consoling taking the best of those, is that θ over time will get higher and higher, and higher, and I'll move from θ_min over time, the θ_max. And when I get to theta max, I've converged, and I'm done. >> I see. So all right, so the theta right, so I guess I was confused because I, I didn't completely absorb the second line yet. So that we're updating the theta and it's specifically going to be tracking the, I'm not sure what the nth percentile is. >> [INAUDIBLE] >> Adding the number of samples that we draw. >> Oh no. I'm sorry. Nth means you know, 50th percentile or 25th percentile or 75th percentile. >> So, okay. So, if you set that to be the median. >> Mm-hm. >> Which is the 50th percentile. Then what we're doing is we're, we're sampling. We're taking the top half of what's left. We're somehow refitting a distribution into that, set of individuals that were drawn. And repeating in the sense that now, the fitness of those individuals, the median should be higher because we're sampling from kind of a more fit collection. >> Right. That's exactly right. >> Okay. >> So this should work or at least it should match your intuition it would work if there are, sort of two things that are true. The first is that we can actually do this estimate. >> Yeah, that's the part that scares me. Uh-huh. >> Sure. Well given a finite set of data, can we estimate a probability distribution. So that's the first thing that sort of had better be true. And the second thing that needs to be true and this is, I think, a bit more subtle, is that the probability distribution for a piece of theta and a probability distribution for say, piece of theta plus epsilon. That is a slightly higher value of theta, should be kind of close. So in other, and what does that mean? What that means. Is that generating samples from P sub theta of t should give me samples from P sub theta of t plus 1. So I have to that, that means that not only do I have to be able to estimate the distribution, I have to do a good enough job of estimating this distribution such that, when I generate samples from it, I will also generate samples from the next distribution that I'm looking for. >> Okay? >> Okay. >> So if those things are true. Then, I, I hope, you can be sort of imagine that you would be able to move from the part of the space that a theta min, and eventually get to the part of the space, the theta max. >> Yeah, it seems like it should climb right up. Yeah, I agree with that. That's just, assuming that you could represent these distributions in between. Again, the picture I have in my head is an, like a bumpy mountainous landscape. And you put a slice through it. And, in the beginning, yeah, right. So in the beginning it's the whole space, and that should be easy to represent. At the end it's a single point, and that's easy to represent. But in the middle. Yeah, like that. But in the middle, it's this sort of crazy, there's a blob, then there's a space, then there's more blob. Little blob with a hole in it, so like that might be a much harder distribution to represent. >> Right, and that's going to turn out to be an empirical question but as I, as I hope you'll see in a couple of slides it tends to work out pretty well in practice. >> Cool. >> Okay. And by the way when it doesn't work out well in practice, there's all these cute tricks that you can do to make it work out pretty well in practice, and we'll talk about that towards the end, okay


MIMIC: ESTIMATING DISTRIBUTIONS
$p(x) = p(x_1|x_2 \dots x_n)p(x_2|x_3 \dots x_n) \dots p(x_n)$
$x = [x_1\, x_2\, x_3 \dots x_n]$
... with DEPENDENCY TREES
$\hat{p}_\pi(x) = \prod p(x_i|\pi(x_i))$
- relationships
$\prod p(x_i)$
- sampling is easy
7:36 / 8:10

22. Okay Michael, so, let me see if I can convince you that we actually have some way of actually estimating these distributions. So, all I've written up here is the chain rule version of a joint probability distribution, okay? But just to be clear, what these subscripts mean here, every x that we have is made up of a set of features. So, there's, let's just say there is N of these features. And so really, X is a vector. There's feature one. There's feature two. There's feature three, dot, dot, dot. All the way up to feature N. Okay? >> Sure. >> What I really want to know for my piece of theta and I'm going to drop the θ's here for the purpose of just describing this generic distribution. Is I want to say, well the probability of me seeing all of the features yeah, all of the features of some particular example is just the joint distribution over all of those features. Now of course, we could just estimate this, but that's going to be hard. >> Why's it going to be hard? Because. >> Well, the first distribution is conditioned on a lot of things, so it's an exponential sized conditional probability table. >> Exactly, so this is exponential table and if it's an exponential table then we need to have an enormous amount of data in order to estimate it well and this is sort of the fundamental problem. But, we have addressed this in earlier lectures, earlier lessons Michael. >> In the inference lecture maybe? >> Yes, in the inference lecture. >> Whew. >> Where we can try to estimate this incredibly painful joint distribution by making some assumptions about conditional independence, and in particular I'm going to make, one kind of assumption. And that is, that we only care about what's called Dependency Trees. Okay, so what's a dependency tree Michael? Do you remember? >> No I have absolutely no idea, I don't think I've ever heard of such a thing. >> So the Dependency Tree is a special case of a Bayesian network, where the network itself is a tree. That is, every node, every variable in the tree, has exactly one parent. >> I see. So sometimes in Bayesian networks, they talk about polytrees. Polytrees just mean that if you ignore the direction of the edges, you have a tree. But you're actually talking about the, the edges have to go in a particular direction. Everybody's got one parent. >> Right, exactly right. And in, and in, so, you should see these as a directed graph, although I almost never draw them that way. Because, you know, it's sort of obvious by looking at it. That this is the root of the tree. So all we have here now is a tree. Every node

has one parent, and what does that mean? That means that every node, every random variable (x_i), depends on exactly one other random variable. Yeah. >> So we can rewrite this joint distribution here now as a product over each of the features depending upon only its parent. >> I see. So the first capital Π there means product. And the second lower case π there means parent. >> Exactly. So when I compare this representation of a distribution, dependency to representation versus the full joint. What nice properties do I get from doing it this way versus doing it that way? Well the, I guess the main positive is that you're only ever conditioned on, well it's gotta be at most one, right? Not exactly one? >> Right. At most one, so, so in fact you, you picked up on a nice rotational thing here. The parent of the tree, the root of the tree doesn't actually have a parent. So just assume that in this case Pi returns itself or something and so it's the unconditional distribution. And that could happen at any time. >> Got, gotcha, alright, yeah and so, like if nobody has any parents, then that's the same as Naive Bayes, but here it can happen with those one parents. >> No. >> No? >> Naive Bayes has one, exactly one, everyone has one parent and it's exactly the same one. >> Oh, right, good point. >> So, if everyone has no parents, then you're saying that all of the features are in fact, independent of one another. >> Gotcha. Okay. Alright. But the, but that's not what you asked. You asked comparing that probability distribution to the full joint, the main thing is that you're, no ones ever conditioned on more than one, other feature and therefore the conditional probability tables stay very, very small. >> Right. In fact do you, how small do they stay? >> Well it depends on how big this, are there, are there binary features that we're talking about? >> Yeah let's say they're binary. So then it's like two numbers. Right. It's like you're probability when your parent is true and your probability when your parent is false. >> Well that's each table. But what about representing the entire thing? >> So it's going to be linear in the size of the number of variables or. >> Right. >> Features. >> Right and the number, the amount of the data that you need is going to be fundamentally. In order to get this tape, in order to get this tree, it's going to turn out that you only need to keep track of quadratic set of numbers, quadratic in the number. >> We're not going to be given that tree, we're going to have to figure that out? >> Well, remember, one of the steps in the algorithm is you have to estimate the distribution. >> So we're going to have to figure out, of all the trees that I might have, which is the best tree? >> Yeah, that' doesn't exactly follow from the algorithm sketch because you could also say well, then you need to figure out that a tree is the right thing at all instead of something else like a box. >> Oh, that's fair, but what we're going to do is were just going to assume that we're going to do the dependency trees, now why are we doing this? Were actually doing this for a couple of reasons Michael, your, your points pretty your points well taken. Dependency trees have this nice feature that they actually let you represent relationships. The point is that you're actually able to represent relationships between variables. Which in this case are sort of features in our space. But you don't have to worry about too many of them, and the only question here then is how many relationships do you want, what kind, which of all the possible relationships you could have where you only are related to one other thing, do you want to have. Well, in some sense a dependency tree since you can depend on at most only one other thing, is the simplest set of relationships you can keep track of. The next simplest would be, as you point out, not having any parents at all. In which case, you would be estimating simply this. >> Yeah, I wouldn't say that the next simplest. That's even simpler. But it doesn't, doesn't allow any of the inter-relationships. Any of the co-variants essentially information to be captured. >> No, that's fair, that's a fair point. So, we could have started with something like this except here you must you you're forced to ignore all relationships because you're treating everything as independent. And we don't believe that things are independent or at least we think there a possibility there's some dependence. And so by allowing at most that you're connected to one other parent that's sort of the least committed to the idea you could still be while still allowing you to capture these relationships. So that's the basic idea. Now I want to be, I want to be strict here that the mimic algorithm or just this whole notion of representing probability distributions does not depend upon dependency trees. We're going to use dependency trees here because you have to make some decision about how to represent the probability distributions, and this is kind of the easiest thing to do that still allows you to capture relationships. >> I buy that. >> Okay and one other thing worth noting is I you'll I hope you'll see this is a couple of slides if you don't see it immediately, is that at the very least this will allow us to capture the same kind of relationships that we get from cross over in genetic algorithms. >> Huh? >> And that was a bit of the, the inspiration for this. That cross-over is representing structure. In this case structure that's measured by locality, and this is kind of the general form of that. >> So you are saying if there is some locality, then whatever, however we're going to learn the dependency tree from the samples is going to be able to capture that. And therefore, it will kind of wrap its head around the same kind of information that that cross over's exploiting. >> That's exactly right, and in fact, as we'll see in one of the examples that I'll give you in a moment, that it can do better than that because it doesn't actually require locality the way that crossover does. There's one other thing that's worth mentioning here about this distribution and why it's nice. It captures relationships. But the second thing is, something that you, you sort of alluded to earlier is that, it's very easy to sample from. Right? So given a dependency tree where each one of these features, each one of these nodes, represents a feature, it is very simple, very easy to generate samples consistent with it. Right? You just start at the root, generate a sample, unconditional sample according to whatever the distribution is and then you go through the parents and you do the same thing. So it's exactly a topological sort and in trees topological sorting is very easy and this is in fact, linear in the number of features. >> Right. I get that and it is exactly an instance of what we talked about when we did Bayesian inference The thing that is new is how do we figure out dependency tree from the sample. >> Exactly, that's what we're going to do next and that's going to involves math.

FINDING DEPENDENCY TREES

$$D_{KL}(\underline{p}\,\|\,\underline{\hat{p}_\pi}) = \sum_i p\left[\lg p - \lg \hat{p}_\pi\right]$$

$$= \underline{-h(p)} + \sum_i h(x_i \mid \pi(x_i))$$

$$J_\pi = \sum_i h(x_i \mid \pi(x_i))$$

23. >> Okay Michael, so what I'm going to do, I'm going to step through how you find dependency trees and just stop me if it doesn't make any sense, okay? >> Sure. >> But hopefully it's fairly straightforward, at least if you understand information theory. And again I want to stress that although we're going to, we're going to spend some time doing this, this is just one example of ways that you could represent a probability distribution. It's just going to turn out that this one is particularly easy and powerful, okay? >> Exciting. >> Okay, so the first thing we have to remember, Michael, is that we have some true probability distribution which I'm just going to write as P, that we're trying to do. That's our P^θ in this case. But the general question of how you represent a dependency tree doesn't depend on θ or anything else, there's just some underlying distribution we care about, let's call that P. Okay. And we're going to estimate it with another distribution which I'm going to represent as P-hat because, you know, for approximation. That's going to depend upon that parent function that we defined before. Okay? >> Alright. >> So somehow. You sound skeptical, Michael. >> Well, because I saw that you wrote those train tracks. >> And you figure they must mean something? >> Well I know they mean

something. It's a, it's an information theory thing but I'm not going to remember what it is. You're going to, it's going to, you're going to say something like kl divergence or something. >> That's exactly right. So, somehow we want to not just find a p-hat sub theta, that is a dependency tree that represents the underlying distribution, but we want to find the best one. And so best one here sort of means closest one, or most similar, or the one that would generate the points in the best possible way. And it turns out, for those who remember information theory, that there's actually a particular measure for that and it's called the KL Divergence. You remember what the KL Divergence stands for, Michael? >> Kullback Leibler >> That's right. And it has a particular form and in this case noncontinuous variables. It has basically this form. And that's basically a measure of the divergence, that's what the D stands for, the divergence between the underlying distribution P that we care about and some other candidate distribution P-hat that we're trying to get as close as possible. So if these are the same distributions, if P-hat and P are the same distribution, the Kullback–Leibler divergence is equal to zero. >> Mm. Mm-hm. >> And as they differ or as they diverge this number gets larger and larger. Now it turns out that these numbers are unitless. They don't obey the triangle inequality. This is not a distance. This is truly a divergence. But if we can get this number to be minimized, then we know we have found a distribution P-hat, that is as close as we can get to P, okay? And we have a whole unit that Pushcar will do to remind everybody about information theory where this comes from. But basically this is the right way to define similarity between probability distributions. You just have to kind of take that on faith. >> Okay. Well, I'll, I'll pause this and go back and listen to Pushcar's lecture and then I'll be ready for you. >> Okay, beautiful. Okay. So what we really want to do is we want to minimize the Kullback–Leibler divergence the that is minimize the difference between the distribution that we're going to estimate with a dependency tree and the true underlying distribution. And just by doing some algebra, you end up getting down to what looks like a fairly simple function. So Michael, if you were, if you were paying close attention to the algebra, you will realize that well p log p, now that you've come back from Pushcar's lecture is just entropy. >> Yep. >> So or it's minus the entropy. And so I can rewrite this as simply minus the entropy of the underlying distribution (-h(p)), plus the sum of the conditional entropies, for each of the x_i, given its parent. Which has some, you know, sort of intuitive niceness do it, but, whatever. This is what you end up with just by doing the substitution. P log P gives you minus entropy of P minus P log P hat, which gives you the conditional entropy according to the function, the parent function pi. Okay. Well, in the end all we care about is finding the best π. So, this term (-h(p)) doesn't matter at all and so we end up with a kind of cost function that we would like to minimize. Which I'm going to call here J, which depends upon pi which is just the sum of all the conditional entropies. Basically the best tree that we can find will be the one that minimizes all of the entropy for each of the features, given its parents. Does that make any intuitive sense to you? >> Yeah. I think so. Cause we want, we want to choose parents that are going to give us a lot of information about the values of the corresponding features.

FINDING DEPENDENCY TREES

$$D_{KL}(\underline{p} \| \underline{\hat{p}_\pi}) = \sum p[\lg p - \lg \hat{p}_\pi]$$

$$= \underline{-h(p)} + \sum h(x_i | \pi(x_i))$$

$$\underline{\min}\ J_\pi = \sum h(x_i | \pi(x_i))$$

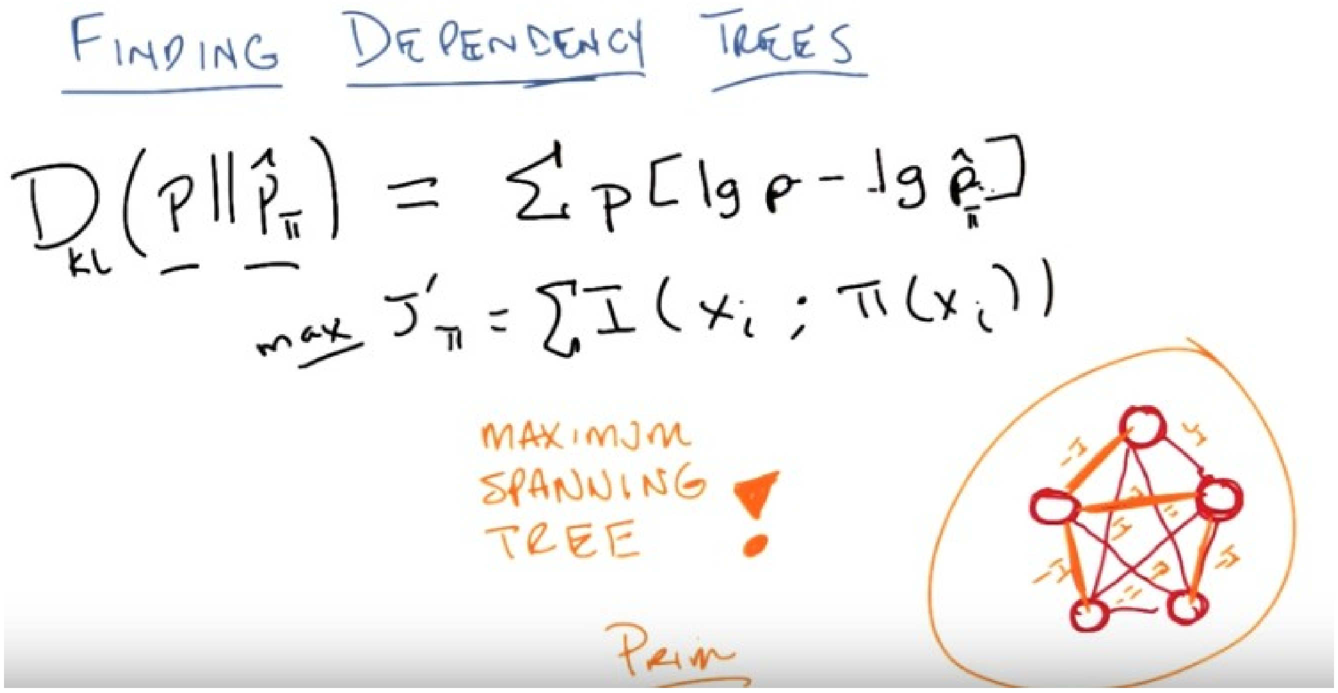
$$\underline{\min}\ J'_\pi = -\sum h(x_i) + \sum h(x_i | \pi(x_i))$$

$$\underline{\min}\ J'_\pi = -\sum I(x_i ; \pi(x_i))$$

$$\underline{\max}\ J'_\pi = \sum I(x_i ; \pi(x_i))$$

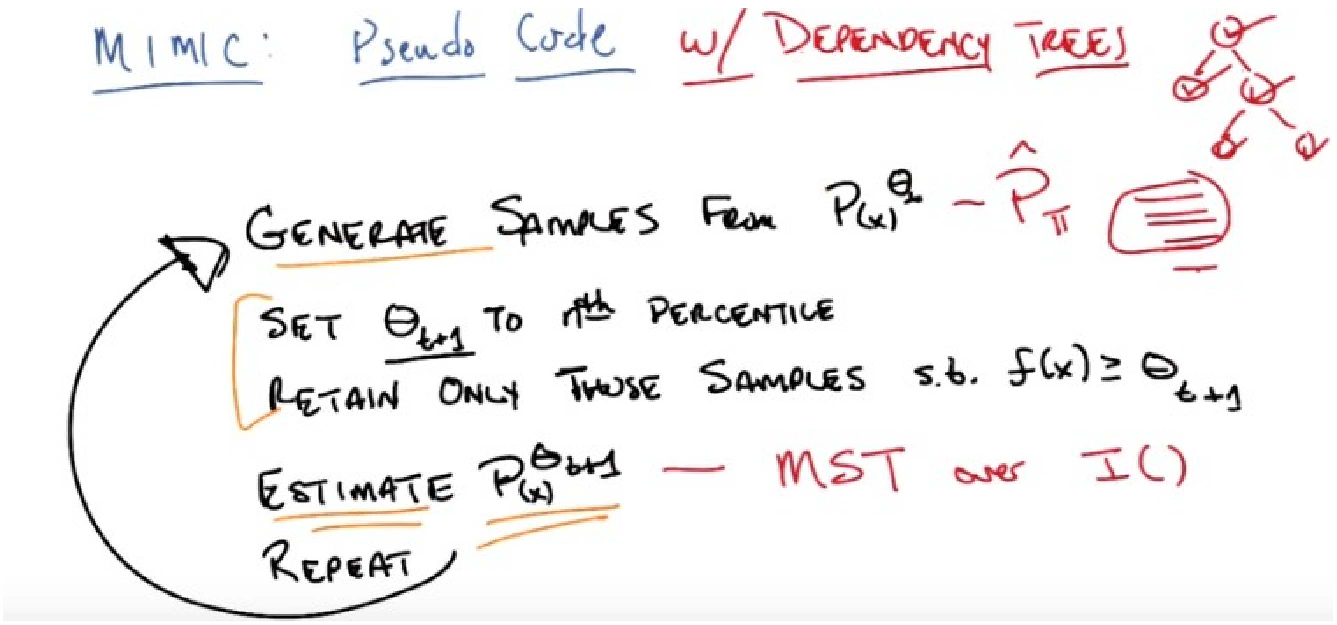
24. Right. So if, in order for this (J) to minimized you would have to have picked a parent that tells you a lot about yourself. All right. Because entropy is information. Entropy is randomness (entropy 越大, 就越混亂, 信息就越少. 故要想 entropy 小, 信息就要多). And if I pick a really good parent then knowing something about the parent tells me something about me and my entropy will be low. So if I can find the set of parents for each of my features such that I get the most out of knowing the value of those parents then I will have the lowest sort of sum of entropies, conditional entropies. You with me? >> Yeah. >> Okay, now this is very nice and you would think we'd be done except it's not entirely clear how you would go about computing this. Actually, I know how you go about computing it, and let me tell you, it's excruciatingly painful. But it turns out that there's a cute little trick that you can do to make it less excruciatingly painful, and what I'm going to do is I'm going to define a slightly different version of this function. And I'm going to call it, J prime. So as I said before, we want to minimize this particular cost function, j. Which we get directly from the Kullback–Leibler divergence. So all I've done is define a new function j prime, where I've added this term. Just minus the sum of all of the unconditional entropies of each of the features. Now I'm able to do this because nothing in this depends upon π and so doesn't actually change the proper pi. That makes sense? >> Yeah, except I keep thinking about pie. >> Mm, pie. Mm, I'm sorry, you got me distracted. Okay but do you see how minimizing this versus minimizing this should give me the same pi? >> I do. I mean, it's sort of like adding a constant if you've got a max. It doesn't change which element gives you the max. >> Right. But by adding this term I've actually come up with something kind of cute. What is this expression, Michael? Do you know? >> It looks kind of familiar from Information Theory. Is that cross-entropy? >> No, though sort of, but no. Is it mutual information? >> It is in fact, mutual information. >> In fact. >> It's the negative of, mutual information. So, minimizing this expression, is the same thing as maximizing mutual information. Now, I went through this for a couple of reasons Michael. One is, I think it's easier to, kind of see what mutual information is. But also because, this going to induce a very simple algorithm. Which I'll show you in a second, for figuring out how to find a dependency tree. And the trick here is to realize that, conditional entropies are directional. Right? >> Wait, so conditional entropies is, is, oh, is that, that's the quantity that we started up? >> Right. >> At the top with the hq,

okay, huh. >> So, this is, this is directional right? Xi depends upon this >> Yeah. >> And if I were to do h of pi given xi, I would be getting a completely different number. Mutual information on the other hand is bi-directional. >> Interesting. >> It's going to turn out to be easier to think about. But before we do that let's just make certain that this makes sense so we wanted to minimize a couple of liable deductions because that's what Shannon told us to do. We work it all out and it turns out we really want to minimize the kind of cost function another way of rewriting this of conditional entropy. We threw this little trick in, which just allows us to turn those conditional interviews into mutual information. And what this basically says is that to find the best π, the best parents, the best dependency tree, means you should maximize the mutual information between every feature and its parent. >> Nice. >> Right. And that sort of makes sense, right? I want be associated with the parent that gives the most information about me. >> Charles, I have a question. >> Yes? >> Just a little bit confused. The the xis, what are they, where's that bound? >> Oh, by the summation. >> The, what summation? >> You know, the summation that's always been there the entire time I've been talking. >> Oh, I'm sorry I missed that. >> I don't know how you missed it man, its right there. >> Yeah, I mean, you didn't actually put the index in the summation so I was, guess I was, I was confused. >> My fault right, so just to be clear of anyone who noticed that the [INAUDIBLE] version is a summation over all possible variables in the distribution And so we've done here is carry that all the way through and so these two steps here in particular. This new cost function that we're trying to minimize is a sum over the negative mutual information between every variable, every feature, and its parents. And that's the same thing as trying to maximize the mutual information, the sum of the mutual informations, between every feature and its parent. >> Hmm, interesting. >> Right, and again I think that makes a lot of sense, right? You, basically the best tree is, the best dependency tree is the one that captures dependencies the best. >> I see. Alright. That's cool. Alright, so now we need to figure out how to do that optimization. >> Exactly right. And it turns out it's gloriously easy.


FINDING DEPENDENCY TREES
$D_{KL}(\underline{p} \| \underline{\hat{p}_\pi}) = \sum_i p[\lg p - \lg \hat{p}_\pi]$
$\underline{\max}\ J'_\pi = \sum_i I(x_i ; \pi(x_i))$
MAXIMUM SPANNING TREE !
Prim

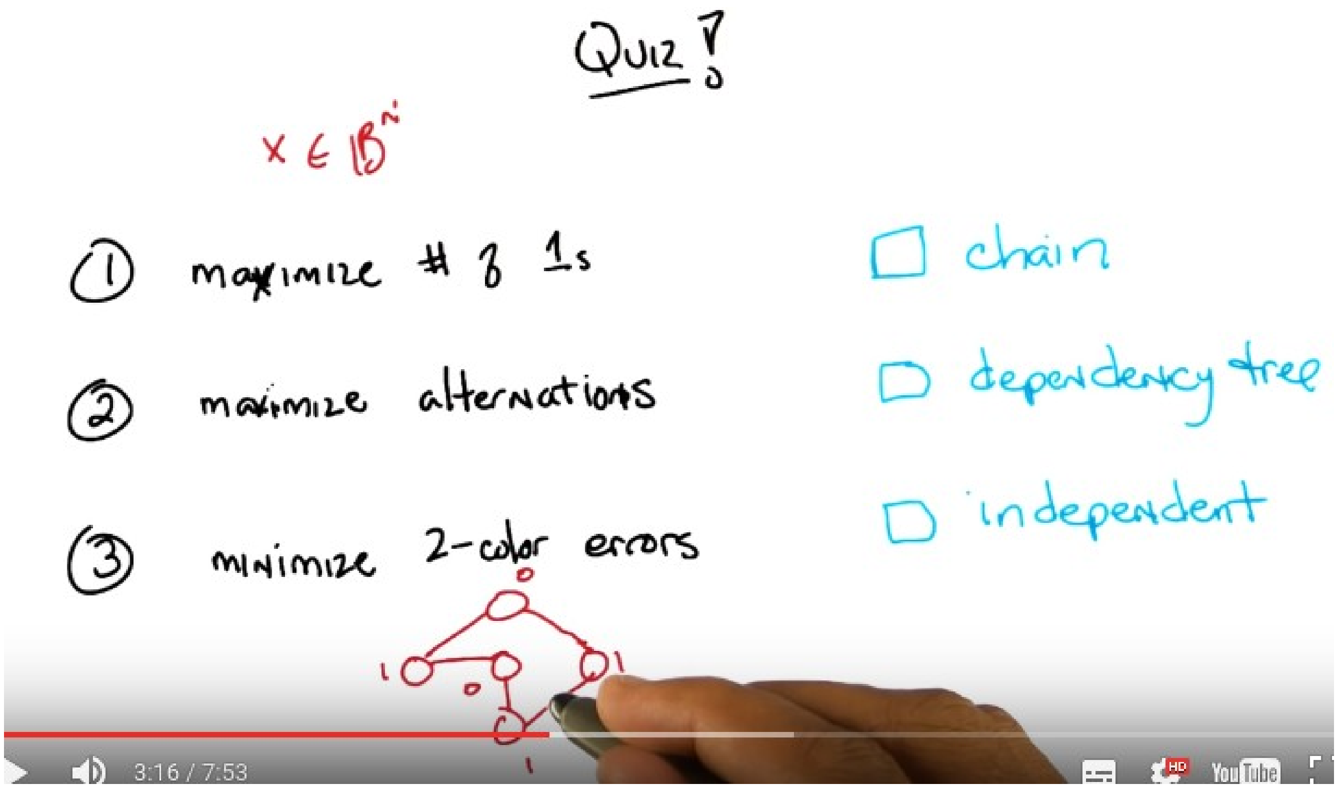
25. Alright Michael, so see if you can tell me where we're going here. So, just as a reminder I've erased a bunch of stuff so I can make some room. We're trying to maximize this cost function, J'π, which is basically the sum of the mutual information between every single feature and its parents. And I want to

point out that that actually induces a graph. In fact a fully connected graph. So here, I've drawn a bunch of nodes as part of a graph, and all the edges between them. Each one of these nodes is going to represent a feature, an x_i, and what's going to go on the edges, is going to be the mutual information between them. So, I now have a fully connected graph, which has the mutual information between each pair of nodes. All in squared edges, and I want to find a sub graph of this graph, in particular a tree. That maximizes these sums. So, any ideas how I do that, Michael? First off, does that make sense what I just said? >> Yeah, yeah, yeah, I mean it's, it's very interesting right. So we're trying to pick, each, each time we pick an edge, it's actually saying that what we, we're going to care about the relationship between this pair of variables. And we're only allowing edges. We're not allowing, because they, they, that involves 2 nodes. We're not getting sets of 3, or 5, or however many nodes larger than that. >> Mm-hm. And so what we want, is we want to be considering a subset of edges, that form a tree, that has the highest total information content. This is correct. >> And that's a minimum spanning tree. >> Well, not quite. >> That's a maximum spanning tree. >> That's exactly right. It turns out that finding the tree consistent with this graph, such that this is true, is in fact the same thing as finding the maximum spanning tree. Which we all vaguely remember from our algorithms class. >> So you said, it was terribly painful, but it's actually not. This is a >> No, I said, it's, it's terribly easily painful. >> So, that's really neat. We turn to problem of finding the best distribution, and in particular the best dependency tree, independent of the true underlying distribution, I want to point out, into a problem of some, of a well-known, well understood computer science problem, of finding the maximum spanning tree. If we find the maximum spanning tree. Then we have found the best dependency tree. >> So, to be honest, I don't remember maximum spanning tree. Minimum spanning tree, can we just solve a maximum spanning tree problem by, I don't know, negating the edges or, or >> Yes. >> Adding, okay. >> That's exactly right. So, everywhere in there, where you pick a max, you pick a min or you just label them with the negative of the mutual information and it's the same thing. I think that this particular form of writing as a maximum mutual information, is easier to think about. >> Okay. >> And what you'll do when you find the maximum spanning tree, is you'll end up with, in fact some tree. And you're done. >> Well, so now you need to tell me, because the structure was a directed structure. So, dude, so can we then pick any node then call it the root and then do a, like a depth first reversal or something? >> Absolutely. So, first off, I want to point out there are two different algorithms that you could use. Do you remember the two algorithms you could use to find the maximum spanning tree? Or the 2 famous ones anyway? >> I'm going to go with Prim and Kruskal but I know there's others. No, those are the two that, that they teach you in algorithms class. And it turns out for this particular one, you want to use Prim. Because, prim is, >> Proper. >> [LAUGH] It is proper, prim and proper. Prim is the proper algorithm to use whenever you have a strongly connected graph, it just happens to be faster. >> You mean densely connect to graph. >> What did I say? >> Strongly. >> Okay, densely connected. Right, a densely connected graph and by the way, this is a fully connected graph, which is about as dense a connected graph as you can get. [LAUGH] So, Prim's algorithm runs in time quadratic or polynomial anyway in the number of edges. And so it's actually fairly efficient as things go, because as we know in theory if it's polynomial, it might as well be free. There you go. >> [LAUGH] >> So use Prim's algorithm, you find the Maximum Spanning Tree, and you are done. >> Alright, I'm going to need reminding where we, where this is all plugging in. >> Okay. >> because. >> That's exactly what I was going to. >> because, this is a cool diversion, but. >> Right, so this is actually worth pointing out this was a bit of a diversion that we had to do. Let's go back to the original algorithm that we had, and just point out what we would be doing here.


MIMIC: Pseudo Code w/ Dependency Trees
Generate Samples from $P_{(x)}^{\theta_t}$ ~ $\hat{P}_{\pi}$
Set θ_{t+1} to nth percentile
Retain only those samples s.t. $f(x) \geq \theta_{t+1}$
Estimate $P_{(x)}^{\theta_{t+1}}$ — MST over I()
Repeat

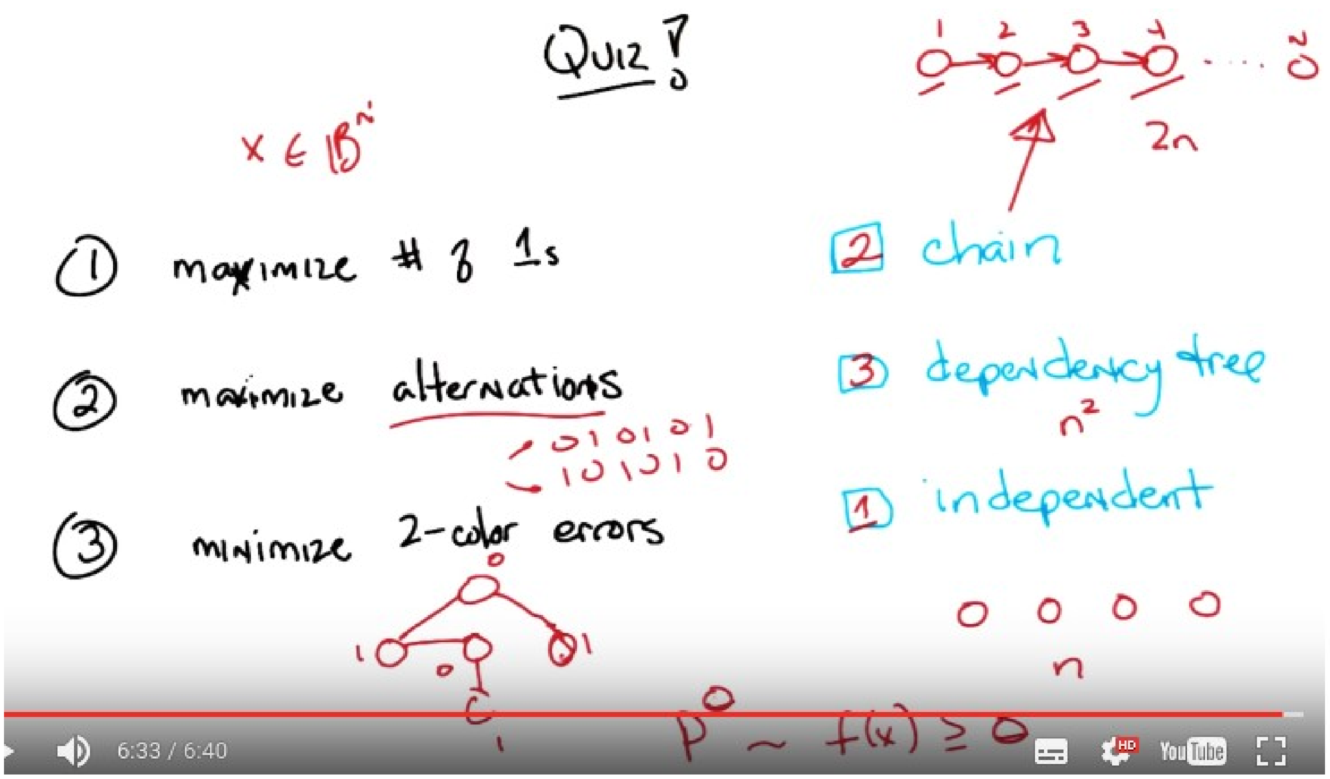
26. Okay Michael so let me take you back to the pseudo code that we, we have for MIMIC and see if we can plugin some of the stuff that we we just talked about just as a way of refreshing. So as you recall, the goals to generate samples from some P^θ_t (x). So in this case were just going to estimate that by finding the best dependency tree we can, that is the best Pi function, okay? So we're going to be starting with some dependency tree, and we're going to generate samples (sample 應該是指 P(a | b, c)這樣的東西, 我在 Bayes 那兩個文件中某個中說個) from that, and we know how to generate samples from that given a dependency tree like say this. We simply started a node, generate according to it's unconditional distribution and then generate samples from each according to it's conditional distribution given it's parents. So, first we generate a sample from here. Then we generate one from here, then from here, then from here, and then from here. Now where do we get these conditional probability tables, Michael? >> That's a good question. I think it's pretty direct from the mutual information. >> Exactly. In order to compute the mutual information, that is in order to compute the entropies, we have to know the probabilities. So, we generate all of these samples, and that tells us now, for every single feature, X of I, how often that was say, taking on a value of one, or taking on a value for zero. If we assume everything's binary. So that means we have unconditional probability distributions. Or at least estimates of unconditional probability distributions, for every single one of our features, yes? >> Yep. >> And at the same time, because we have these samples, we know for every pair of features. The probability that one took on a value, given a value for the other. So, already have the conditional probability table for each of those as well. So, just the act of generating these samples and building that mutual information graph, gives us all the conditional and unconditional probability tables that we need. So given that we can generate samples and time linear and the number of features, and we're done. So this part is easy. By the way you asked me a question before. I just want to make it clear, that we come back with an undirected graph. And, you can pick any single node as the root, and that will induce a specific directed tree. And it actually just follows from the chain rule. I'll leave that as an, an exercise to the, the viewer. But there you go. So now we know how to generate samples. From some tree that we have. We generate a bunch of these samples. We find the best ones. Retaining only those samples, and now we know how to estimate the next one, by simply doing the maximum spanning tree over all the mutual information. And then we just repeat. And we keep doing it. And there you go. A particular version of mimic. With dependency trees. Got it? >> Cool. >> Right. So, again, just to really drive this point home, you don't have to use dependency trees. You can use

unconditional probability distributions, which are also very easy to sample from and very easy to estimate. You could come up with more complicated things, if you wanted to. Try to find the best Bayesian network. You can do all of these other kinds of things and that'll work just fine. Dependency Trees though are very powerful, because they do allow you to capture these relationships, that is to say they give you a probability distribution that has structure that we were looking for, while not having to pay an exponential cost for doing the estimation.



27. Okay so, we're going to have a quick quiz. Michael insists on having one, because Michael likes that sort of thing, and I like Michael, so I'm going to go with him and give you a quick quiz to see if you follow along with what we talked about. Now this quiz is less about the details of mimic itself. Than it is about the probability distribution. So I've been harping on this idea that mimic doesn't care about your probability distribution. You should just pick the best one. So we want to make certain that you get that by giving your three problems and three probability distributions you might use and see if you can tell us which one goes with which. Okay? So here are the three problems. The first problem is you're fitness function is maximize the number of 1. So the first thing you need to, to see is that; we're going to assume in all of these cases, that our input value is x binary strings of length N. Okay? So, 00000001001011. 11100111, whatever. There's going to be a string of 100. So, they're binary digits. And the first problem we want to maximize the number of 1s that appear in that sample. Do you get that, Michael. >> Yeah. I mean it's, I mean that's going to be maximized by just making everything a 1, right? >> That's right. That's right. >> Okay, but, but we want, we want to think about mimic finding that. Right, because you don't know that, that's the true underlying fitness function. That just happens to be what it is. >> I see. Okay. >> Okay. So, in the second problem, we want you to maximize the number of alternations between bits. So, 1010101 is better than all 1s, much better the all 1s in fact. because the fitness you'd get from maximizing alternations. Between 010101 is in fact, N minus 1 which is the largest that could be. But if you add all the digits being the same, your fitness would be 0. You see that Michael? >> And minus 1, oh, I see and minus 1 because it switch every time it switches.

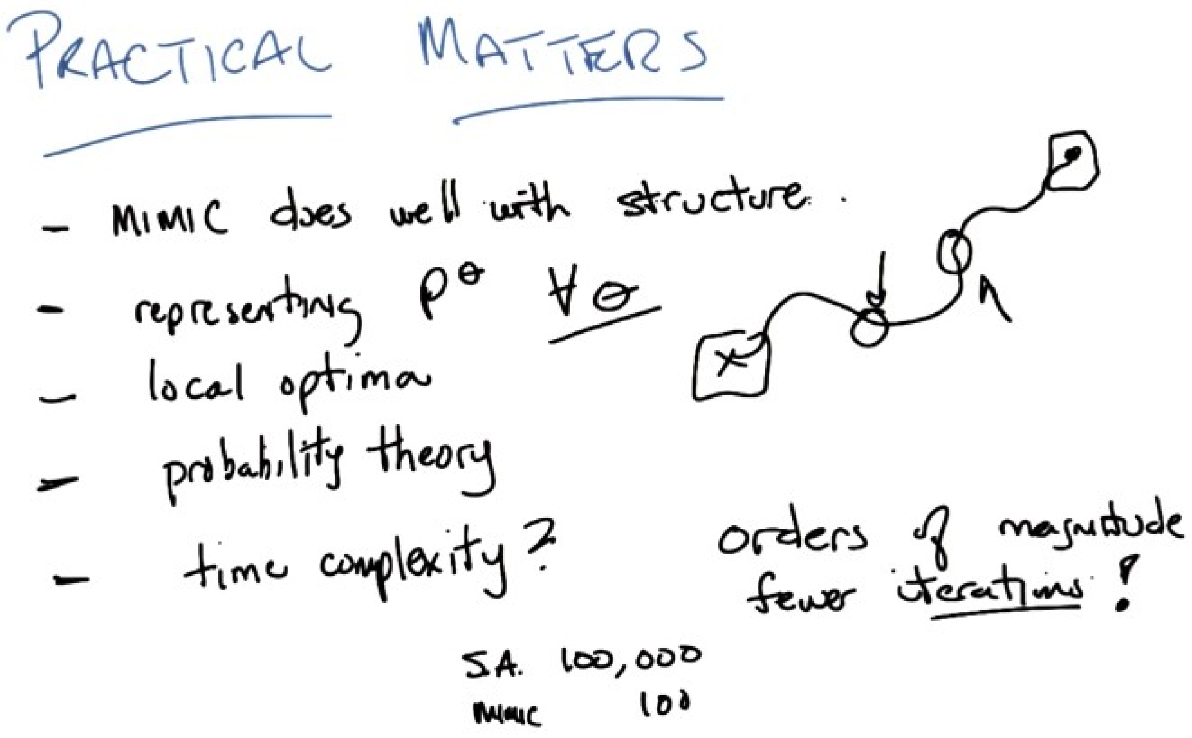
>> Right. >> Got it. >> So what would maximize the number of alternations and say, hey. Five digit string. >> Like 01010. >> Or? >> The compliment of that, 10101. >> Right. So two values both act as maxima here. Okay. The third problem (本問題中只有兩個 color: 0 和 1, 其實也是一個二進製數的問題) is you want to minimize two color errors in a graph. Now that one is a little bit harder to describe, so I'm going to draw a little graph for you. So, the two color problem you might remember is given a graph with some edges, like say, this graph here. You want to assign a color to each node such that every node has a different color than its neighbor. Okay? So, we assume there's only two colors here. One is 0. So let's say I assign This the color, value of 1. This the color value of zero. This the color value of zero. This the color value of 1. And this the color value of 1. So how many errors are there? Michael. >> Wait, we're, oh, we should be trying to maximize. But, okay, so I guess not. So minimize two color errors. So an error would, in this case would be, an edge that has the same color on both ends. And I see one of those, the bottom sort of rightish. >> Right. So here these do not match. So that's good. These do not match. These do not match. But these two match. Even though that one doesn't. So there's exactly one error (即圖中筆指向的那個邊). So in these first two examples we're trying to maximize the fitness. Here we're trying to minimize the cost just to make things. Slightly more difficult for you.



If we removed this edge here, then you'll notice that this has no mismatches whatsoever, and this would be an optimum. >> Got it. >> Okay? So we need to figure this out. Okay? So there we go. To maximize the number of ones in your string or maximize the alternations by adjacent bits. Or to minimize the number of two color errors in a graph. Alright. Those are the three problems. Now here are the three distributions. The first distribution, these (指那三個 distributions) are in alphabetical order, is a chain. So a chain would be in kind of Bayesian network speak, a chain would be a graph. Like this. Where basically every feature depends on its previous neighbor. So, in a four-bit string, I'm saying that the first bit depends on nothing, the second bit depends on the value of the first bit, the third bit depends on the value of the second bit and the fourth bit depends on the value of the third bit. So, the way you would generate samples is you would generate a sample from here. And then generate a sample on here dependent upon this value. Given that value, you generate a sample here. Given that value, then you generate a sample given that value. Got it Michael? >> I think so. So, and, and we are imagining that the ordering is given. So, that, so, we know which ones depend on which. >> Right. So, in fact, it's not just a chain. It's a specific chain. >> And is it the same chain as, say, the ordering of the bits? >> Yes. In this particular case, it's the same chain as the ordering of the bits so, I'm going to call this that chain. >> [LAUGH] Alright. >> Okay. The second one is what we've been talking all about along. It's a dependency tree. Unlike in the case with this chain here where I am giving you a specific chain. This (chain 中) is the first bit, the second bit, the third bit, the fourth bit and so on to the nth bit. I don't know which is the dependency tree is or you have to find it but there is some dependency tree I want you to represent. And the third one is the easiest to think about and that's where everything is independent of everything else. So, if I were to draw that. It would be a bunch of nodes with no edges between them, directed or otherwise. And so, the joint probability across all your features is just a product of the unconditional probabilities. So, the simplest probability distribution you can have. Okay, You got that, Michael? >> Yeah (此藍是起分界線作用), so. And, okay, if I'm understanding correctly, each of these is representable by a dependency tree. >> Yeah, each one is, actually. So then why, why would any one be better than any other one? >> I don't know, Michael, you tell me? >> I guess in the case of, well could be, you could think of it maybe in the terms of number of parameters that need to be estimated. So in the independent case, since we know that it's independent or at least we're imagining it's independent, we just have to estimate one probability per node, which is like N. >> Yep. In the chain case, we have a conditional probability per node. So it's, like 2n parameters. >> Mm-hm, yep. >> And in the dependency tree case, it's. I think what you were saying is that we're estimating kind of n squared parameters. And then, then pulling the tree out of that. Exactly. Now, of course, like you point out Michael. A chain is a dependency tree. Independents are a dependency tree and a dependency tree is surprisingly enough a dependency tree. >> [LAUGH] >> But these numbers and parameters do matter, because although a dependency tree can represent and independent set of variables. The way your going to discover that their independent is that your going to need a lot of data to estimate those in square parameters in order to realize that the conditional probability tables effectively don't mean anything. So, it will be very easy for a dependency tree to over fit in the case where all the variables or all the features are independent. Okay. >> Okay. Alright. So, you got it? >> Alright. Nope. One more question. >> Yes? >> Well, other then I need to fill the boxes with the numbers one, two, or three, right? >> Yep. Of the algorithm that they're best for, or the the problem that they're the best for. >> Got it. And each one is used exactly once? Yes. >> Okay, and then finally just to kind of make sure that I am wrapped head around why we are doing probability distributions at all. So in the context of mimic, we need some way of capturing the, I guess the space of answers that do at least a certain level. They, they do at least this will and in terms of the fitness. So, it, it, it. >> Hm. >> I guess, I guess I feel a little turn off because. We're not trying to represent the fitness functions, right? You're not telling me which dependency structure should I be using to represent this fitness function. >> Right. Instead I'm asking you to figure out which distribution will represent the optima, the structure between the optimal values. >> Huh, okay. Alright, I'm not sure I 100% get it, but I think I get it enough to answer this quiz. >> Excellent. Okay, so go.

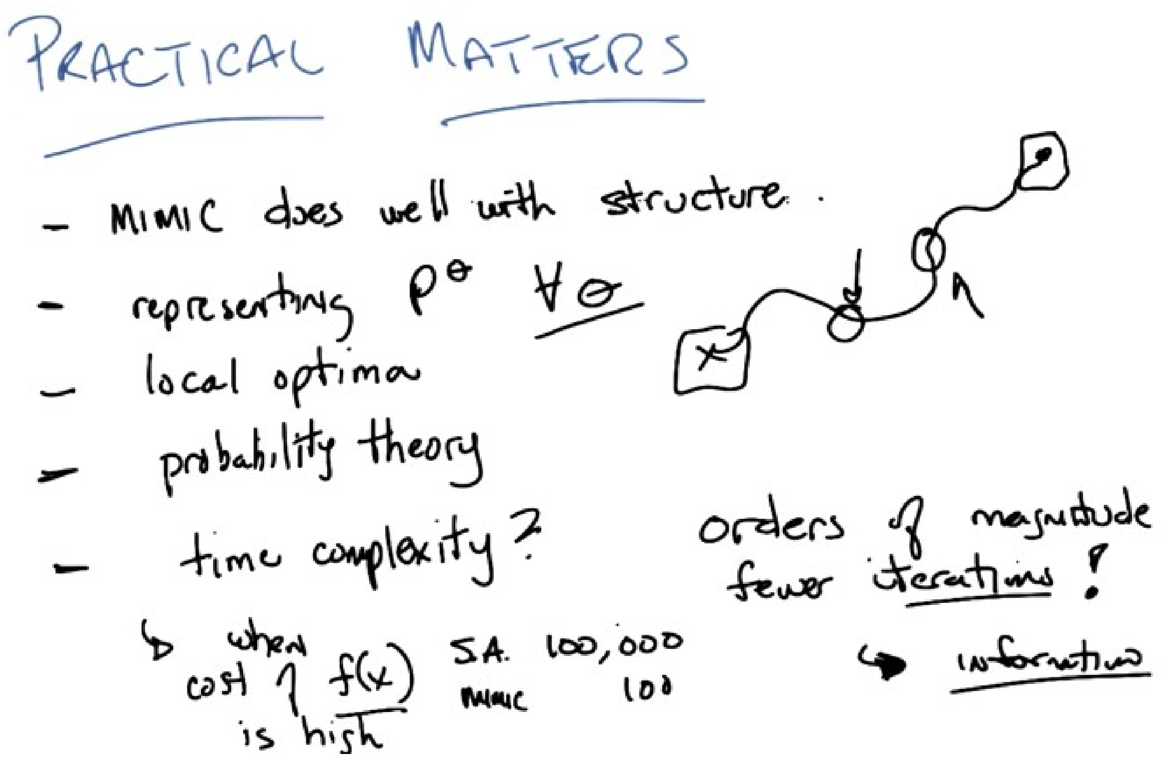
28. >> Okay, Michael. >> Okay, so problem one, where you're maximizing the number of 1s, to represent the optima here, or the influence of any given bit on the fitness value, they're all independent. They all just contribute whatever they contribute. And so I don't see any reason to capture other dependencies. We're, we're, you know, local information is enough. Is that, is that enough to say that the answer, that the one goes in the last box for independent? >> That's correct. >> But it's, I guess, I guess what I don't quite understand is in, in the case of the, so we want to know, what's the probability, yeah, it's a funny thing we're representing, right? >> Yeah. >> So if we, the, like, the third position, we're saying, well what's the probability that if this is a 1 that we're going to be above a certain value,

and what's the probability if, if it's a zero, we're going to be above a certain value. >> Mm-hm. >> And that doesn't really exactly capture the way the fitness function works. But I guess, oh, I guess because we're drawing from, we're drawing the rest of the things at random, the rest of the values at random? >> Mm-hm. Yeah, we are. >> So is that where the meaning comes from, then? >> Yeah, think about it this way. Remember, the probability distribution that we care about, p sub theta, is basically uniform for all x's such that the fitness value is greater than or equal to theta. So imagine we start out with theta equal to 0, let's say, which is the lowest value that you could get? Well then, what's probability of bit 1 being a 1? >> Okay >> For all values greater than or equal to, well, it's 1/2. >> Okay. >> Because every single value is okay. And so in order to get a uniform distribution across n bits, I can sample each bit independently, uniformly, and that will give me an overall uniform distribution. Agreed? >> I do agree with that, yeah. >> Okay. Now, at the very, very end, when I have all 1s as the maximum value, that's also easy to represent. Because the probability, what's the probability of bit 1 being 1? For the optimum value. >> Yes, the probability of bit 2 is? >> And the probability of bit 3? >> Point, no, 1. >> Exactly, and they all can be independently represented. So we can definitely represent the minimum theta value. >> Streams, yeah. >> Yeah, so, we can represent the maximum. The question here is, or at least in terms of, you know, you're likely to find the answer. The question here is, can this distribution represent values of theta in between? So imagine theta was, let's say I had four bits here, like we're doing here. And let's imagine theta was 2. >> Good. >> Now, notice that when theta is 2, it's not just the number of points that will give you exactly 2, it's the ones that will give you at least 2. 2. >> Right. And 3, and 4. So how many different ways are there to get 4? Well, there's only one way to get a value of 4, and that is all 1s. How many different ways can you do 3? Well, there's 4 ways to get 3. You basically have to choose the one that you give as 0, right? And each one of those bits, each one of these values will be a 1, three quarters of the time. And how many different ways can you get 2? Well it's n choose 2, which is, something. >> 6, so you can actually write all of those out and count the number of times that each one is a 1. And those are all your samples, and you just simply estimate it. And you'll end up with a uniform distribution which will be consistent with the examples that I just sampled from, but will probably not exactly capture p sub theta. because it will sometimes be able to generate with probability greater than 0, say, a value of all 0s. >> Yes, okay, good. That was what I was concerned about. And so this is, this is just an approximation, even in the simple case. >> Right. And so, and that's actually pretty interesting, right? So, yeah, we know that the extreme values can be represented by an independent distribution, but it's not clear that all the values in between can be represented by such a simple distribution. But that's always going to be the case. What you want is a distribution that will definitely capture the optimum and, or optima, and gives you a good chance of generating samples that get you closer to the optimum along the way. So even though in the case before where theta is 2, we might still generate examples where you get a fitness of 0 or 1, you have a high probability if you generate enough samples of actually getting values of theta greater than 2, 3, or 4, even. >> Got it. >> Okay. >> Alright, so I, so, given that, I think I can do the next two without too much more difficulty. So, for number 2 problem 2, maximize the number of alternations, we pointed out that it's really important to, to know who your, what your neighbor is, because you want to be a different value than your neighbor. >> Mm-hm. >> And the chain gives you exactly that information without anything extra. >> Right. >> So I would put the 2 in the first box. >> Yep. >> And finally, well, there's only one box left. So I'd put the 3 in the middle box. But I guess the interesting to, to look at here is that it is surprisingly appropriate, right? So, that the coloring problem, it is specifically saying, well, the, my value depends on, you know, what's a good value for me depends on, I guess, several of my neighbors, not just one (neighbor). >> Right. >> But I feel I could, you could capture a lot of the necessary information by finding a good dependency tree. >> Right. And it turns out that in practice, as our readers and listeners will see, from one of the resources that we're making available to them, that in practice, mimic does very well on problems like this. Even where you have really complicated graph structures. It tends to do much better than a lot of the other randomized optimization problems. And that's because what we've got here in this graph is structure. And what mimic is trying to represent, is in fact, structure. In fact, in all of these cases, well, except for the, the first case. There's not a single answer often. There are possibly many answers, but the answers all have in common their relationship to one, to some underlying structure. So in the maximizing alternations case, you have 010101 or 101010. These are two completely different values. In fact, as you pointed out when I asked you this earlier, they're complimentary. But, each one has a very simple structure, which is, every bit is different from its neighbor to the l, actually both of its neighbors. Every bit is different from its neighbors. And that doesn't matter what their values are. Given the value of one of them, it actually completely determines the value of everything else. And so mimic doesn't get lost trying to bounce back and forth between this possible maximum and this possible maximum. Instead, it represents both of them at the same time. Because the only thing that matters is the relationships. So, in fact, that gets us to the last bit of things I wanted to talk about, which are sort of practical features of mimic and what we kind of learn by thinking this way.


PRACTICAL MATTERS
- MIMIC does well with structure.
- representing P^θ $\forall\theta$
- local optima
- probability theory
- time complexity?
orders of magnitude fewer iterations!
S.A. 100,000
MIMIC 100

29. Okay. So here's some practical matters, Michael. I mentioned one of them before, and that is that mimic does well with structure. When the optimal values that you care about depend only on the structure, as opposed to specific values, mimic tends to do pretty well. By contrast Randomize hill climbing, genetic algorithms. These other things that we've looked at before can sometimes get confused by two different values that are both optima but look very different from one another. Where it's the structure that matters and not the actual values. So, the chain example before where you had alternating values as one of those cases where it's easy for randomized algorithms that only look for point values to get confused. 'because their basically being drawn in multiple directions at once. The quiz also brought up another point which is worth mentioning here was that mimic and with anything that tries to do this kind of search through probability space is that it's an issue of representing everything. That is it's not enough just to be able to represent a probability distribution of the optima. You really want to be able to represent everything in between as you move through probability space toward your answer. This is the universal symbol for moving through probability space. You don't just

want to represent here at the end and here at the beginning which is pretty easy because uniform distribution, but can you represent this point? Can you represent this point? And if you can't are you going to end up getting stuck. And actually turns out that mimic can get stuck in local optimal though it typically does not optima because you get randomize your search for optima. >> Hm. I see. >> But the problem of local optima is still a problem of local optima. Now, when I say something like you get randomized restarts for free, I'm actually cheating a little bit and hiding something which is a little bit more important, which is what you really get for free is probability theory. So there's a hundred, literally, hundreds of years of work on how to think about representing probability distributions and what you can do with them and there are terms like important sampling and rejection sampling And all these kinds of tools that we have for representing probability distributions that you ca actually inherit with something like Mimic for dealing with these painful cases where you might not be able to represent distributions. >> But the single most important thing to me about Mimic or what to get out of here is that representing structure does matter. But you pay a price. And that price basically boils down to, time. So the question we might ask ourselves, is, what is the sort of practical time complexity of MIMIC? And, it really boils down to something very simple. Let me just share a fact with you Michael. Okay. I have run this algorithm on many, many, many examples and I've compared it to Simulated Annealing. I've compared it to [INAUDIBLE] algorithms. I've compared it to randomized hill climbing. And it works pretty well for the sorts of examples I've come up with. And here's a little fact that I just want to give you. MIMIC tends to run orders of magnitude fewer iterations. And I'm not exaggerating here; I mean that if I run Simulated Annealing, it might take 100,000 iterations for something like Simulated Annealing, but for Mimic, it might take only a hundred. And this is consistently true, so it turns out that that's not good enough. It turns out that the fact that Mimic can do something in three, four, five, six, seven orders of magnitude fewer iterations Is it an argument for always using it? Can you imagine one now? >> because it might give a worse answer? >> Well, in practice it doesn't. So, these are cases where both simulated [UNKNOWN] and mimic, or randomized hill climbing or genetic algorithms actually eventually do find the answer. Mimic just finds it in orders of magnitude, fewer iterations. >> But you're counting iterations. >> Mm-hm. >> You didn't control for the fact that Different algorithms, can take different times for a single iteration. >> That's exactly right. So, what do you think? If I were to compare simulated annealing to mimic, which do you think take, the, takes more time for any given iteration? >> Well simulated annealing just does this little tiny step, right? It like, computes a bunch of neighbors and then does a probability comparrison, and then, takes a step. Mimic is drawing this sample, estimating a bunch of parameters, then yeah. I guess the other way around. It's drawing from a distribution. It's computing which things are say above the median performance, and then it's re-estimating a new distribution. And then that's the end of the iteration. Depending on how many samples it takes to do that, it's, it (MIMIC) could take a very long time, and in particular, it's going to always be a lot more samples than what simulated annealing is doing. >> Almost certainly. So, when would you think that MIMIC would still be worth using in a case like that, where we know that we can get to the answer, but simulated annealing will take orders of magnitude more iterations, MIMIC will take fewer iterations? So when would it still be worth it to take the one with fewer iterations even though east, each iteration is expensive? Prints algorithm, quadratic this that and the other. >> Oh yeah, grab at that part, hm.



PRACTICAL MATTERS

- MIMIC does well with structure.
- representing P^θ $\forall \theta$
- local optima
- probability theory
- time complexity?
  - ↳ when cost of $f(x)$ is high

SA. 100,000
MIMIC 100

orders of magnitude fewer iterations!

↳ information

30. Here. Let me put it to you this way Michael. What is MIMIC giving you for all of that work that it's doing in building dependency trees and, and you know, running Prim's algorithm and finding maximum spanning trees. >> I'm going to say structure because that's the word that you've been using repeatedly. >> You get structure. And, and another way of thinking about structure in this case is you're getting information. You get a lot more information because you get structure, you get a lot more information for iteration as well. So, that's the price you're paying, you're getting more information every single time you do an iteration, at the cost of building this maximum spanning trees or whatever it is you're doing in, in estimating your probability distribution. So, why would it be worth it to do that? What's the other source of expense? >> Well, so, all right. So there's all this computation within an iteration, but what it's, what it's doing, what it's trying to do is trying to find inputs that have high scores and so you do have to compute the scores for all those inputs. So the, the fitness calculation is very important. >> Right, so MIMIC tends to work very well when the cost of evaluating your fitness function is high. So, it's really important that I only have to take 100 iterations if every single time I look at a fitness function and try to figure, compute it for some particular x, I pay some huge cost in time. >> Well, have you told us how many function evaluations there are in an iteration? because it seems like, it could be a lot in MIMIC. >> Well, you get one, basically for every sample you generate. >> Yeah and, and so, but so. I guess, you can compare iterations but you can also compare samples. >> Right, so they would depend upon how many samples you feel like you need to generate, and of course you can be very clever because. Remember, theta will generate a bunch of samples for theta plus 1. >> Mm-hm. >> So, if you keep track of the ones that you've values you've seen before, you don't have to recompute them. So it's actually pretty hard to know exactly what that's going to be. But let's imagine that at every iteration, you generate 100 samples. So, at most, you're going to have to evaluate f of x, 100 times. So, MIMIC is still a win over something else, if the number of iterations that it takes is 100 or more than 100 times fewer. >> Wow. >> Right? >> Yeah. >> So, can you think of functions, fitness functions, real fitness functions that might actually be expensive to compute? >> Well, yeah,

sure. I mean a lot of this stuff that is, is important is like that. So, if you're trying to design a rocket ship or something like that, that. Fitness evaluation is, is doing a detailed simulation of how it performs. And that could be a very costly thing. >> Right, actually it's, that's a good example because MIMIC has been used for things like antenna design. It's been used for things like designing exactly where you would put a rocket in order to minimize fuel cost sending it to the moon. These sorts of things where the, the cost really isn't evaluating some particular configuration where you have to run a simulation or you have to compute a huge number of values of equations and so on and so forth, you know, to figure out the answer. Another case where it comes up a lot is where Your evaluation function, your fitness function, involves human beings. >> Oh, interesting, yeah. >> Where you generate something, and you ask a human, how does this look, or does this do what you want it to do because humans, as it turns out, are really slow. >> [LAUGH] Yeah, they're not measured in megaflops. >> That's right. So, you end up with cases where fitness functions are expensive, something like this becomes a big win. And that's a general point to make, though, that when. You are looking at all of the different algorithms, at different models or at everything that we have been doing, both for unsupervised learning and earlier for supervisor learning. A lot of times, your trade-off is just in terms of whether you are going to over-fit or not. It's whether it's worth the price you need to pay in terms of, either space or time complexity to go with one model versus the other. We've been talking about it in terms of sample complexity, but there's still time complexity and space complexity to worry about. >> Cool. I get it. >> Okay, cool. So normally what we would do next is we would say what, whatever we learn, but I actually kind of think we just did that. >> [LAUGH] indeed. >> So, in fact, let me do something about that.

(本段圖跟上一段一樣)
31. >> So, Michael what have we learned? >> That. >> I think you're right. Okay, well, Michael, I was happy to be able to share some of this with you and to remind myself of something I did 20 years ago. >> [LAUGH] And it answered the concern that you had, so it fit into the over all structure. >> Excellent, oh, it fit into the structure. Well done, Michael. >> Oh, I see what I did there. >> Yeah, very well done. Well, I will talk to you later Michael. Thank you so much for listening. >> Sure, thanks. >> Bye.