Chapter 5.5 - Genetic Algorithms

10 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=TCFSTYVOY30
https://www.youtube.com/watch?v=tCFSTYVoy30

SUMMARY

Genetic algorithms are optimization techniques that mimic the process of natural selection, where a population of potential solutions is iteratively refined. By generating multiple guesses, evaluating their performance, and retaining the best-performing solutions, genetic algorithms effectively search for optimal solutions in complex, high-dimensional spaces without relying on derivative information.

- *Genetic algorithms start with a set of initial guesses for solutions and evolve them over generations.*
- *The process involves evaluating the performance of each guess and retaining the best ones while discarding the worst.*
- *New guesses are generated based on the retained solutions, allowing for exploration of promising areas in the solution space.*
- *This technique is particularly useful for optimization problems that are nonlinear or lack clear derivative information.*
- *The algorithm iterates through generations, progressively improving the quality of solutions.*
- *While genetic algorithms can be slower than other optimization methods, they are powerful for complex problems where the solution landscape is not well understood.*
- *Implementation can be done in programming languages like Python or MATLAB, with existing libraries available for ease of use.*
- *The convergence of genetic algorithms is often difficult to prove, but they demonstrate practical effectiveness in finding optimal solutions.*

we're going to hit one more op we're going to hit one more optimization technique which is known as genetic algorithms and in some sense what a genetic algorithm is it's it's it's nice you're going to start off with some guesses of solutions and the reason they call a genetic algorithm is you're going to take those Solutions you're going to essentially evolve them or perturb them and Warp them in other words in some sense very much like

what we're going to do is try different solutions the ones that do the best keep them they're the fittest of the solutions discard the worst ones and then take those best Solutions and then perturb them and try new Solutions off them and see if this procedure essentially can in fact walk ourselves into the solution this does not rely on derivative information it is more of a a a search procedure for n-dimensional spaces can be nonlinear functions and

it's a pretty powerful technique commonly used especially for very difficult optimization problems where you don't have derivative information you might have or might not have constraints but this is an algorithm just try lots and lots of things and you find the best Solutions and you build off those that's the idea so the idea is the

following you're going to try to minimize over some function f ofx and so how do you find this well you take some guesses so you might guess

some X of JS and these X of JS can be you know random guesses they can be good guesses could be something that you know something about your system that at least has some constraints on this or you can just really randomly guess just put in a vector of random numbers for the control variables you may not in other words if you don't really have any understanding of that landscape of that objective function you know you don't

necessarily know where to start so you just you started with one guess but why one guess Why not start with a thousand guesses and see which which one of these guesses works well and whichever ones work well you say well maybe I'm getting close there so maybe I should start guessing more around that region and if there's guesses that are terrible you may say okay those don't seem to work well so I'll stay away from those areas so that's the idea we're

going to basically think about setting this up starting off with M guesses so where m is maybe fairly large and what we're going to do is we're going to say for each guess what was the value of my function f ofx right so and what I'm going to do is say well I'm going to keep P of these so out of all those guesses that I tried I keep the best p and throw away from p+

one to M so for instance maybe I tried 100 guesses and I see how all of them did and I keep the 20 best guesses throw away the 80 and from those 20 now I start guessing base partly on those 20 that I have left over and in fact there's a lot of very interesting architecture around genetic algorithms I'm just doing a very basic overview of this because how you would keep these modified to new guesses is

actually an art in of itself and I'm not going to cover that here so let's start with an example and I you know I thought the best way to walk through this is just to give you a quick example of how this might work here's some data and what I'm going to try to do is I'm going to actually try to I've actually shown this previously this was I was trying to fit this previously with the nelder me two

lectures ago a cosine BX plus C so I was going to try to fit that I didn't know what a b and c were and so what we're going to do is going to try to solve that same program problem that we had before but now what I'm going to do is I'm going to use it with genetic algorithm so I'm going to take a thousand generation so each time I guess produce results keep some throw some out

that's one generation so notice it's an iteration procedure the iterations over Generations so I just keep hopefully each generation keeps improving that's the entire concept of the Gen genetic algorithm is that essentially survival of the fittest what that means is the best of your guesses keep surviving and hopefully keep improving towards finding an optimal solution what I'm also going to do here is I'm going to pick 80 number of trials that I'm going to try per no

in other words 80 guesses per generation and I'm going to keep the best 40 so I try 80 I keep the top half and then I'm now going to basically use those as a basis for the next round of guesses okay so that's the idea behind this very basic structure we're going to try for right now so here is in fact the initialization process or my guesses I'm going to start off here with a b and c I

had a cosine BX plus c so I'm going to take a it's going to start with 12 I these are my initial guesses I actually did even in that minimize command from the nelder me from two Le few lectures ago and so and I just add some random numbers to it say guess 12 plus some random number B you know so I start just guessing and what I can do is I'm going to produce 80 guesses I'm trying to find

the optimal values of a b and c I don't know what they are I know they're close to this and so now I just get test around and see what works and so here is a full algorithm for doing this so you can download it from the GitHub page right and you can just run it and what we're doing here is basically saying get you know we're going to go through this Loop through a thousand Generations in

each generation we generate some guesses and then what we do is just say how well did each guess fit and what you do is this is the evaluation I'm going to take each guess fit it and say get a score what's my least square fit score and what I'm going to do is basically sort these now and keep the best 40 remember we had 80 guesses I decide the best 40 have the lowest least square error and

then what I do now is I take each I now i' what I've got done is I have new values of a b and c from these and then I randomly generate new guesses off of those and I keep doing this generation after generation after generation in many ways it's kind of as simple as that so what your job is is to provide the guessing the way to create new Solutions go through many generations and hopefully the idea was you keep guessing

you keep refining you keep only the best guesses that are doing the best job and then eventually you keep taking guesses off those and making new guesses off those and then you converge to a solution and so if we implement it that's one way to do it and if you're going to do it in Python here's how you do it you just tell it the function you're doing the data and actually what you're trying to minimize this is the

least square error of that fit and so I also give it lower and upper bounds on what I want to guess and this is how easy it is to do differential Evolution fit the function and this lists the upper and lower bounds on your guesses and then you can run that in mat lab or so python so let's just look at the results of this because it's kind of fun to play around with this because it's it's it's

not the best optimization algorthm for this specific example but this is commonly used in literature when we have very high dimensional systems where we don't have a good understanding of the landscape of f ofx these genetic algorithms can be very quite powerful so here's generation one here's the number of Trials let say 50 trials that I have and here's the Le square error and in the first generation here's my distribution of Errors so I've lined them up these are the guesses notice the

Y AIS here go to 8,000 notice have a lot of error but by the way by the time I get to generation 10 look how many I keep remember keeping only the best guesses and I build off those so my air is dropping notice this is now only to 4,000 dropping I still have some bad guesses 50 Generations in 100 generations and notice my air is dropping significantly here now it's this is a thousand is on that y AIS so

in other words generation after generation I keep doing better and better and better with this idea that I'm going to eventually get really good values of a b and c and keep those and that's kind of converged to the solution in some sense so this is the idea of this so in fact here's the results so what you're looking at here is I so the I believe it's the the the dots are the data the solid line is my little

code that I just wrote there from scratch solution and the dotted line is the python fit through its it's it's its own sulfur and here's the drop in air over time over Generations so remember we did a thousand Generations but you know here's just over 200 Generations you can just see as I keep guessing and throwing away the bad guesses and building on the good guesses my air just starts to drop as I go through these Generations that's the

entire idea so just like grading descent where you're walking downhill this is there's no Hill here because you know in some sense a genetic algorithm doesn't know about you know when we think about that objective function of the landscape this is just simply saying I'm trying to find the best Solutions wherever I do well I use more guesses there wherever I do well I keep improving where I do with this idea that I'll converge to the

solution it's very difficult to prove anything about convergence in these cases however you can see that it works in practice and it's commonly used and in some sense it's a very slow algorithm actually right because generation after generation to try to converge but it's typically used on really difficult problems where you just don't have a good idea of what that solution landscape look like so we're going to finish there with genetic algorithms it's another tool for you to

use in optimization and especially when you have very difficult optimizations this is a something that's there and wellb built algorithms are in Python available for you