Optimization Theory (DS2) Lecture #13
Life Gets Messy: More on Non-linear Optimization Problems

December 27, 2016

Abstract

So far this semester, the ideas have been new and the mathematics and algorithms sometimes hard, but the problems themselves have been clearly defined. Today, things start to get *messy*. We are going to talk more about non-linear problems, and allude to some ways of solving them, but not really make tremendous advances in that direction. (That's for you, for future work!) We are also going to look at problems with probabilistic outcomes and problems that we don't (yet) understand, using the multi-armed bandit problem as an example.

Some text and examples adapted from A Gentle Introduction to Optimization.

1 Finishing Up Local Optimization

1.1 Newton's Method

We didn't get to it last week, so let's go back and look at it now. It's a *second-order technique*.

1.2 Hill Climbing

Hill climbing is pretty straightforward: move along one axis until you hit a maximum (hill climbing problems are always phrased as optimization problems, though the "hill" might in fact be "minimum energy"). Then pick a different axis, and go that way.

Wikipedia says that hill climbing works for both discrete and/or continuous variables, and that it's different from gradient descent because it changes only a single variable at a time. Gradient descent also has to be differentiable, whereas hill climbing doesn't. (Should check that in Norvig.)

Wikipedia also notes that simplex is like hill climbing, but I'm not sure I agree. True after the basis rotation, I suppose. Might be closer than gradient descent? Simplex requires that the solution space not only be linear, but that it be convex.

Hill climbing, simplex and gradient descent are kind of the same thing, but gradient descent assumes that the minimum you are looking for is contained within the space

of possible solutions, whereas with simplex your goal is to find some edge point, so the focus is on the constraints. Taking the partial derivative of the objective function in a linear problem is child's play, so if you want to, you can start somewhere in the middle of the space (although remember, finding *any* feasible solution is a non-trivial and important first step), take the derivative, and start stepping that way...the hard part is figuring out when you hit some face (or edge or vertex, if you're lucky) of the polytope, then figuring out how to reorient your vector so that you continue advancing. That is, essentially, what simplex already does.

Like gradient descent, researchers have tried many variants over the years, mostly centering on ways to make better choices for which direction to move, including making it non-deterministic, or looking for ways to get it to hop out of local maxima, which leads us to...

2 Heuristics (or Metaheuristics) for Approximate Solutions

Last week when we talked about local versus global maxima/minima, we implied that under some circumstances you must have the global minimum, while in others something close will do. (We also noted, when we talked about computational complexity, that the traveling salesman problem is *NP-complete* when posed as a question with a threshold, "Does a path through all cities of length less than k exist?" whereas the more general, "Find the shortest path through all cities," is *NP-hard*, and the distinction arises due the presence or absence of an easily-verified certificate.)

In general, the only way to find the global optimum in a problem with no known structure is the iterate through all of the possible solutions; has we have seen, in many problems, that means examining an exponential number of possibilities (or even, theoretically, an infinite number if any of the variables is continuous). Methods that find a good enough but not necessarily optimal solution anyway are *heuristics* (or *metaheuristics* if they are really a framework for picking the parameters for evaluation).

Prior to the recent fervor for deep learning (a subfield of machine learning, based on neural networks), there was a constant stream of proposed heuristics for optimization and classification or pattern matching (which, in one sense, can be viewed as optimizing some function that represents the quality of a match): simulated annealing, genetic algorithms, particle swarm, ant colony, and more. Many of these looked to physics, biology or population dynamics for inspiration. We are going to take a *quick* look at two of these, trying to convey the principal idea without going into the details of the solutions.

2.1 Simulated Annealing

Simulated annealing is generally done on problems with a discrete solution space. We evaluate the scalar objective function at our current location, then generate a possible *move* to a nearby state, and evaluate the objective function again. If the objective value is lower, we accept the move. So far, just like hill climbing. But here's the trick: if the value is higher, we still accept the move *with some probability*.

Simulated annealing was developed by Kirkpatrick, Gelatt and Vecchi at IBM, with the first major paper published in 1983. They pay homage to the Monte Carlo algorithm of Metropolis *et al.*, which we are not going to discuss in detail.

Given the original physical motivation of annealing, they call the objective function the "energy" of the state. The probability of accepting the move is

$$P(E, E', T) \tag{1}$$

where E and E' are the energies of the current and proposed states, and T is known as the *temperature*. This function has to go to zero,

$$P(E, E', T) \to 0 \text{ for } E' > E \text{ as } T \to 0. \tag{2}$$

You begin with a high temperature, so that the probability of accepting any move is nearly one, then gradually decrease the temperature toward zero. The intention is that this will create a high probability of settling into a very low energy state, perhaps even the *ground state* of the entire system. This process is known as the *cooling schedule*. The original authors talk about the desirability of starting with a high temperature and cooling slowly initially, then rapidly toward the end. Others have suggested that the cooling should be adaptive, and watch for *phase transitions* where the energy suddenly drops substantially.

I can tell you from experience that this is difficult to use well...

2.2 Genetic Algorithms

...and I can tell you with even great certainty that genetic algorithms are even harder to use well!

As you can probably guess, you run with a *population* of individuals, and let them breed, looking for the one that solves your problem best.

To run a genetic algorithm, you need several things:

1. A representation of a solution to your problem as a *genome* (not AGCT, as in biology, but as a string of some length where each entry comes from some fixed alphabet).
2. A *fitness function* (our objective function again, with yet another name),and
3. some way to use the fitness function to decide whether the individuals are allowed to breed.
4. Some way to match members of your population for breeding (purely random?).
5. A *crossover operator*, where two individuals in your breed to create one or more offspring.
6. A *mutation operator*, where each individual is randomly subjected to some set of changes to its genome.

There are other parameters, such as your population size, that also impact the success of your overall system.

3 The Multi-Armed Bandit Problem

Who wants chocolate?

What problem was, "formulated during the war, and efforts to solve it so sapped the energies and minds of Allied analysts that the suggestion was made that the problem be dropped over Germany, as the ultimate act of intellectual sabotage," in the words of Prof. P. Whittle of Cambridge?

From our pedagogical point of view, so far this semester, every objective function has had a *fixed* value, or reward, and even if the objective function is too complex to solve directly for the global optimum, the objective function has at least been *known*. But what if the outcome is probabilistic, and worse, not completely understood?

For example, what if you are given the task of managing a portfolio, either of stocks or researchers, each of which has a probability of paying off big and a probability of failing completely, and you initially have no information about which are good and which are duds?

We will assume here that our goal is to maximize the *expectation value* of the return over a large set of trials, although in the real world you would typically have the additional competing goal of minimizing the probability of your total assets falling below some threshold, or of protecting against certain events through some amount of diversity, or of using your power of allocation here to achieve some other social goal.

At the beginning, since we know little or nothing about the expectation value, we are in the *exploration phase*. At some point, we may feel like we know enough, and we just want to use the bandit that produces the highest returns; now we are in the *exploitation phase*.

4 References for this Lecture

1. Wikipedia on hill climbing
   `https://en.m.wikipedia.org/wiki/Hill_climbing`

2. Kirkpatrick *et al.* in *Science* (1983), and Kirkpatrick alone in *J. Statistical Physics* (1984) have good descriptions and figures for simulated annealing.

3. There are *many, many, many* references on genetic algorithms out there in the world, and unfortunately I don't have a specific compact reference or two that I recommend...

4. Gittins, J. Royal Stat. Soc. (1979) for one of the key papers on multi-armed bandits,
   `https://www.jstor.org/stable/2985029?seq=1#page_scan_tab_contents`

5. "so hard the Allies considered dropping it on the Germans," also from Gittins,
   `https://www.jstor.org/stable/2985029?seq=1#page_scan_tab_contents`

6. And Wikipedia,
   `https://en.m.wikipedia.org/wiki/Multi-armed_bandit`