Comparison of Two different Optimization Algorithms

Ryan Florida

For this project we were to attempt to implement and compare two optimization algorithms, namely gradient ascent and simulated annealing. Both algorithms were meant to find the global maximum of a sum of a given number of Gaussians, with respect to the domain $\mathcal{D} = [0, 10]$. We were provided with the necessary code to generate the sum of a desired number of Gaussians, which is just a term synonymous with the normal curve, given by the normal distribution $f(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, where σ and μ are, respectively, the standard deviation and mean of a given data set.

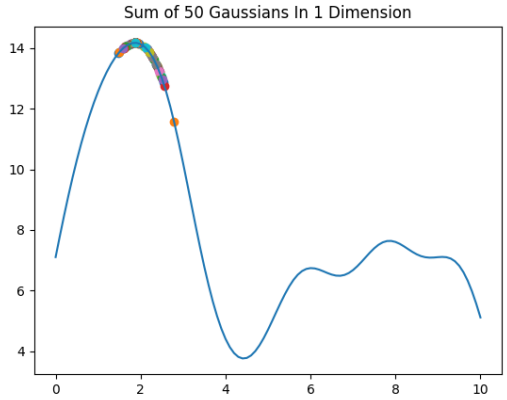
The gradient ascent algorithm, as its name implies, follows the gradient, $\nabla f = \partial_i f \hat{e}^i$, of the Gaussian sum in order to locate the maximum value (see Figure 1.b for the case when $i = 1$). [1] The main idea of the algorithm takes advantage of the—amazing—fact that the gradient of a function is a vector that points in the direction of maximal increase. This fact implies, in two dimensions, that if the gradient is positive at a particular point, then we should take a small step to the right. Conversely, if the gradient is negative at a chosen point, then we should take a small step to the left. This easily generalizes to higher dimensional functions, but a bit more care must be taken when taking a step because left and right are no longer the only choices one must consider. With this in mind, the actual algorithm is implemented as follows:

1. Choose random starting point, $\vec{x}$.
2. Repeat until $|f(\vec{x}_{i+1}) - f(\vec{x}_i)| < \epsilon$ for some strictly positive tolerance ϵ
   (a) Evaluate $\nabla f(\vec{x})$.
   (b) Update $\vec{x} = \vec{x} + \delta \nabla f$ iff $f(\vec{x}) > f(\vec{x} + \delta \nabla f)$, where δ is the chosen step size.

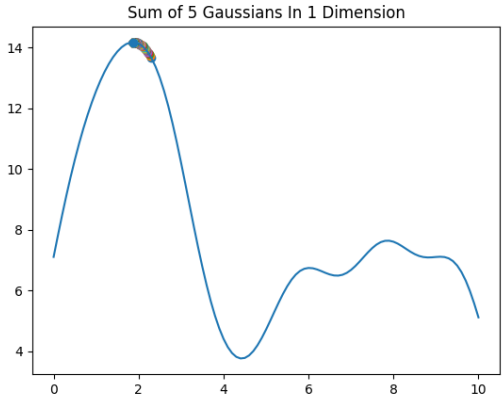
Simulated annealing takes a very different approach to the problem of optimizing a the function. The algorithm is based on that of heating a system to a very high temperature, then cooling it as slowly as possible in order to achieve a maximal strength/stability. The main technique is to first take very large steps in either direction (for a two-dimensional case) and slowly start taking successively smaller steps until the system settles at an optimal point. So, essentially, we are taking a semi-random walk at higher temperatures and then hill-climbing (i.e. following the gradient) at lower temperatures. (see Figure 1.a for a 2-D example). The algorithm is as follows:

1. Choose random starting point, $\vec{x}$
2. Set temperature, T, as high as possible.
3. Do a specified number of times:
   (a) Construct new vector $\vec{y} = \vec{x} + \vec{t}$, where $\vec{t}$ is a vector of random numbers.
   (b) if $f(\vec{y}) > f(\vec{x})$, then update $\vec{x} = \vec{y}$
   (c) else if $e^{\frac{f(\vec{y}) - f(\vec{x})}{T}} > \ell \in (0, 1)$, then update $\vec{x} = \vec{y}$
   (d) Decrease T a small amount.

[1] A note on notation: Einstein's summation convention was used in the definition of ∇f so that i is an index and there is an implied sum over all allowed values of i and hence the corresponding coordinates , $\partial_i \equiv \frac{\partial}{\partial x^i}$, and the $\hat{e}_i$ are the standard unit vectors.



(a) Example of simulated annealing



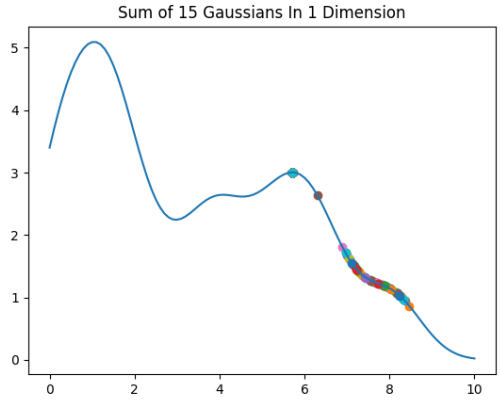
(b) Example of gradient ascent

Figure 1: Visual representation of both algorithms.

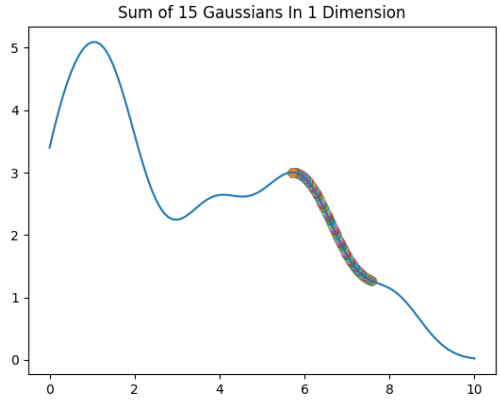
Now let us start to contrast the two algorithms by examining some of their finer details. Gradient ascent will always get us to a maximal value but, unfortunately, this sometimes takes the form of a "local" maximum (See Figure 2.b). Even more troubling is the reality that gradient can also get trapped in saddle points, which are points where the graph changes inflection or a range where the function is constant. This "trapping" happens because when sampling points around a local maximum, or a saddle point, the quantity $|f(\vec{x}) - f(\vec{y})|$ will most certainly be less than our chosen tolerance $\epsilon = 10^{-8}$ for this project. Because the choice of initial position is random, this just adds to the probability that we will get trapped in a local maximum. If one could have more information about a specific problem space, then steps could be taken to avoid falling in to such a trap but that is very problem-specific and not generalizable. Being able to choose the starting point in the problem at hand would not even be beneficial for this problem because, with any input with $\dim(\vec{x}) > 2$, there is little context as to whether we are near a local maximum or not. This is not to say that gradient ascent is not a very good algorithm though, it is far superior to simulated annealing in terms of speed of convergence to its goal since it can stop very quickly if the difference of the f values is less than epsilon. Therefore, the biggest cost associated with gradient descent is accuracy of results.

Simulated annealing, on the other hand, most often avoids falling victim to be trapped in a local maximum because of the semi-random walking nature of the algorithm. If it were to come into contact with a local max, and the temperature was still sufficiently high, it could easily "walk" away from the point and in pursuit of a better one (see Figure 3.a). The actions of simulated annealing are not without consequence though, because it could equally as well end up walking away from a global maximum in pursuit of yet a more optimal point. Annealing could also suffer the same fate as gradient ascent if it is near a local optima when the temperature is very low (See Figure 2.a). Let us look a little deeper into this "influenced" random walk that the algorithm performs. The influencing comes into play in by way of the $f(\vec{y}) > f(\vec{x})$ conditional. So, if the algorithm identifies a better candidate position than its current position, it will immediately take it. However, if the candidate was found to be unsatisfactory and the temperature is still relatively high, then our probability will be given by $P(\text{Take the step}) = e^{-\frac{|f(\vec{y})-f(\vec{x})|}{T}} \approx e^0 = 1$ for $T >> 1$. Hence, at higher temperatures we will have a very high probability of accepting a step in a random direction. The next subtlety in simulated annealing is the

hill-climbing nature that takes over toward the end of the cooling. This comes from the fact that it is very likely to be in the situation that $f(\vec{y}) - f(\vec{x}) < 0 \Rightarrow P(\text{Take the step}) = e^{-\frac{|f(\vec{y})-f(\vec{x})|}{T}} \approx e^{-\infty} \to 0$ as $T \to 0$. Hence, the choice of whether to make a move or not becomes solely dependent on the conditional $f(\vec{y}) > f(\vec{x})$, which is to say that the algorithm will start making the most favorable move at low temperatures. One major drawback of simulated annealing is that it takes a significantly longer time to converge on a solution because there is not really a stop condition that can be imposed on the algorithm, it just runs for the specified number of runs.

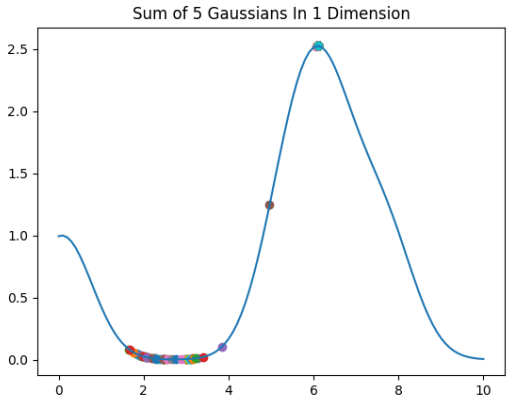


(a) Simulated annealing trapped in local max.

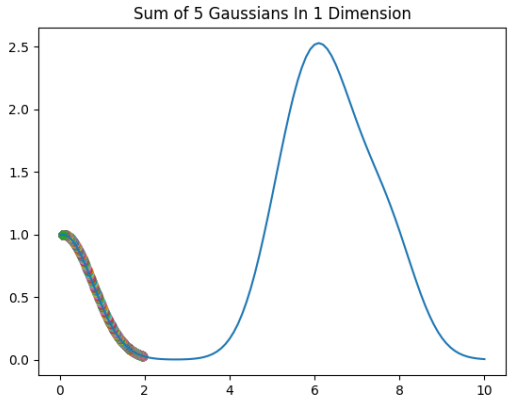


(b) Gradient ascent trapped in local max.

Figure 2: Both algorithms suffering defeat.



(a) Simulated Annealing



(b) Gradient Ascent

Figure 3: Example of simulated annealing out-performing gradient ascent.

For my personal implementation of the algorithms I did not try to optimize the gradient ascent because it is, in its own right, already fairly efficient. The slowest part of the gradient ascent algorithm seemed to be the evaluation of the Gaussians when given the input vector, but it was nothing detrimental to overall performance. When I first implemented the simulated annealing, I noticed that it was much slower than I was expecting, though I quickly realized this was just due to the fact that we go through 10^5 iterations

of the algorithm so I should not expect it to be blazingly fast. For the cooling schedule, I chose to go with exponential multiplicative cooling. This schedule has the form $T = \lambda^n T_0$ for $n \in [0, 10^5], \lambda = 0.999$, and $T_0 = 32767(2^{15} - 1)$. The choice for exponential multiplicative cooling was made because it was the most efficient option that still yields very nice results without having to waste computation time on calculating exponential or logarithmic functions. The choices for T_0 stemmed from the fact that I wanted to start at as high a temperature as possible to keep the random walk going for the majority of the lifetime of the calculation. The choice of λ corresponded to my choice of T_0 and the fact that I wanted to prolong the random walk.

Though I had no control over the sum of Gaussians, I was able to increase the speed and efficiency of my algorithm, while still maintaining the reliability, by making a few micro-optimizations. My first optimization was to find a cut-off value for when to consider T to be zero. I decided this by doing the data-collection portion of the project with multiple cut-off values and found that a cutoff value of 10^{-30} gave me very good results, where "good" here means that it beat the gradient ascent algorithm more than (10^3) times, while still giving me a sufficient number of iterations ($\sim 10^5$) to hill-climb. After doing this testing phase, I was able to reduce the number of comparisons, with no loss in accuracy of results, by ending the "walk-loop" when T fell below the cut-off. With T assumed to be zero, I then started the "hill-climbing loop" from the iteration I left the "walk-loop" without calculating the probability portion of the conditional statement when deciding whether or not to accept the new position values. I neglected the probability term because it would essentially always be zero during this phase, which implies that the comparison statement it was a part of would always yield false. This dramatically reduced my computation time and I was able to complete most calculations—with number of Gaussians ≥ 1000—in almost half the time that my initial implementation could.

In the end, simulated annealing seems to be far superior to gradient ascent in terms of accuracy of results (see table 1) but gradient ascent far out performs simulated annealing in terms of time complexity.

Table 1: Simulated Annealing Performance Compared To That of Gradient Ascent

dim(x)	Wins	Equivalences	Losses
1	293	102	5
2	291	100	9
3	297	100	3
4	298	100	2
5	300	100	0
TOTAL	1479	502	19