Methods of Optimization

Daniel Barnhurst

***Abstract*—Optimization is important in many fields of study, especially in math, science, and computer science. The three methods I will talk about today are commonly used, powerful, but understandable. The names of the methods are gradient descent, conjugate gradient, and quasi newton method. All of them are powerful, and each has its use and purpose.**

I. Introduction

Optimization has many different definitions, methods, and purposes. According to Merriam-Webster, optimization is, [1]an act, process, or methodology of making something (such as a design, system, or decision) as fully perfect, functional, or effective as possible specifically : the mathematical procedures (such as finding the maximum of a function) involved in this. Some things would be impossible to complete because they take longer to do than time would allow.

The purpose of the optimization that I will discuss in this presentation is geared toward finding the greatest or least value in a function. More specifically, the 3 optimization methods discussed in this presentation are geared toward finding the minimum. These methods, in computer science, are known as algorithms. Algorithms are basically a set of rules or instructions that must be followed to reach a result. The two things which are necessary to be an algorithm it that it must finish, and it must always produce the right answer. There is always a price to improve in different areas. For example, running faster may take a lot longer of time to code. Different programming languages have different trade offs. Python is really easy to code in, but it is extremely slow compared to other languages such as C++ and definitely slower than Assembly (which is one of the fastest methods short of machine code).

II. Main

A. Gradient Descent

Almost every machine learning project involves a method called gradient descent. It is a common strategy and is popular because it is reasonably simple and doesnt require much math. It is pretty universal and can be incorporated with most algorithms.

A gradient is a term that comes from vector calculus, and is basically a kind of derivative in a vector form. The gradient points in the direction of the greatest increase. It aims for the fastest way to reach a extrema. Once it finds an extrema, (Max or Min), it returns a zero, meaning that it, for lack of better terms, levels off. Even though its possible find global extrema, the gradient can end up on local extremas and get stuck there.

Gradient Descent requires a function in order to operate. Its purpose and goal is to minimize the function, searching for the lowest possible valley in the function. Since gradients are

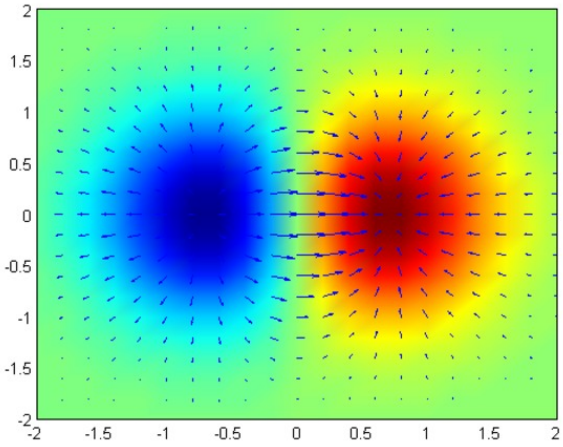

Fig. 1. Simple gradient displaying both negative and positive divergence

used to find the direction of the steepest change, the algorithm is able to find the fastest direction with the steepest descent and moves in that direction for the desired distance, and when it arrives, it does the same thing again.

The distance that is traveled in the direction of most decrease is called the magnitude. If the magnitude is too high, it might jump over the low points and miss the minimum. If it is too slow, it will take a lot of time, get stuck in local minimums, and very likely it wont make it to the end. If functioning properly, every step taken should decrease in value. Making graphs make it easier to see if your magnitude is optimal.

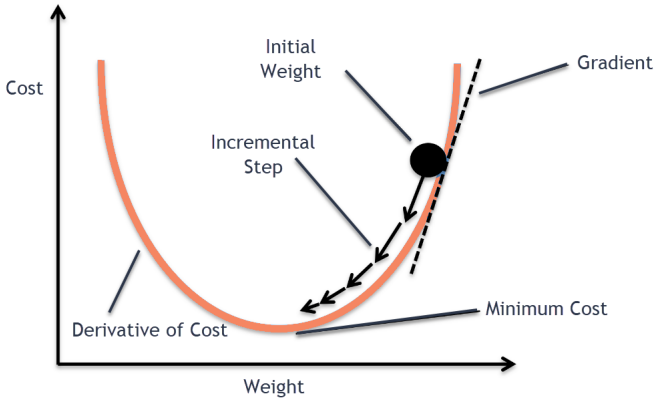

Fig. 2. Optimum magnitude for the function given

There are three separate types of gradient descent. They are Batch gradient descent, Stochastic gradient descent, and Mini

Batch gradient descent. I won't get into details, but explain briefly the differences.

Batch Gradient descent processes all of the training examples before it picks one. It is efficient computationally, but it doesn't get to the exact answer that the function has. Stochastic Gradient Descent updates the parameters for each training example. This makes it faster and has a more detailed improvement rate. It is more computationally expensive and ends jump around sometimes. The Mini Batch Gradient Descent is a combination of the other two options, it breaks the tasks into smaller batches, (hence the name), which makes it popular.

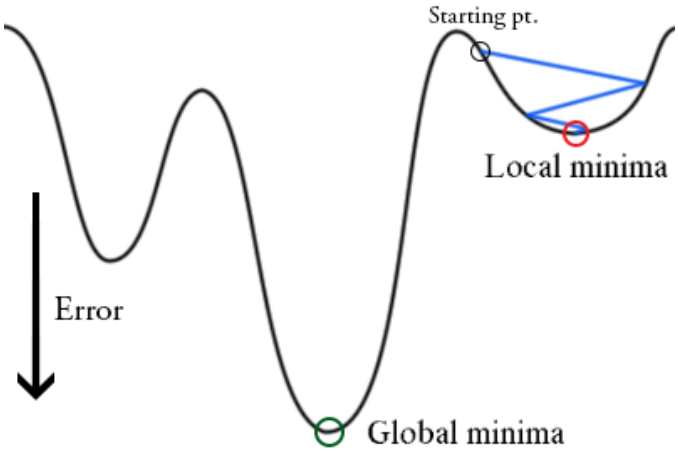

Fig. 3. Stuck in local minima

Most downfalls gradient descent have been addressed and fixed, because it is old and popular, but some still remain. When working with huge data groups, it starts to get really machine hungry, although stochastic gradients fix this to a certain extent. It is also hard to determine the proper learning rate for the individual function. Another problem that most minimizing functions have is that it gets stuck on local minima, which is a big problem, though it can usually be worked around.

B. Conjugate Gradient

The workings behind the conjugate gradient venture back into math, specifically linear algebra. It is an iterative way of solving large systems of linear equations. It was invented in 1951 by Hestenes and Stiefel. It takes much less steps to find the minimum than the plain gradient descent.

Originally, the conjugate gradient method only worked for linear equations, but soon after it gained the ability to handle quadratic equations as well. Now it can do many different equations, making it versatile and powerful. Obviously, the main advantage of the conjugate gradient method is that it only takes a few steps, and is very accurate. That means a lot less jumping around and less calculations need to be made to find the minimum. The trick is that the algorithm moves along the plane until the gradient in the direction it is moving equals zero. This means that the gradient is perpendicular to the direction being traveled. When its reached that point, it is done traveling in that direction, so the only direction needed to go is conjugate to the original direction.

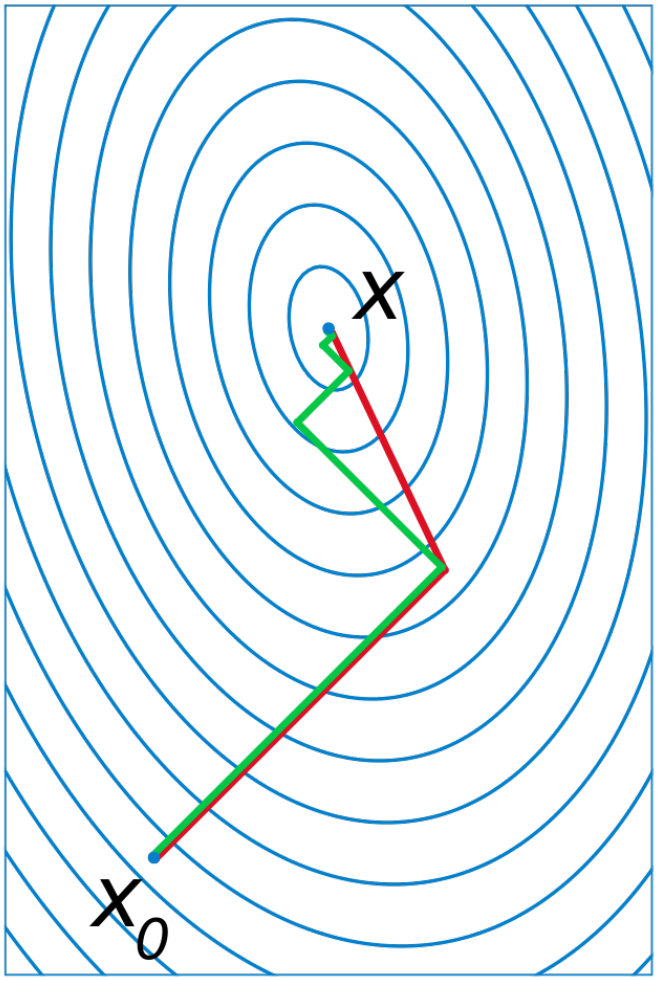

Fig. 4. Conjugate Gradient VS Gradient Descent

C. Quasi Newton

Newtons method is a powerful, accurate way to find zeros and local extrema, but it can be pretty costly. The hard part of Newtons method is finding the Jacobian or Hessian. Sometimes they can't be calculated and sometimes they use too much computational power each iteration, which is the downfall of Newton's method. Luckily there is a new way based on Newtons method that does not require the exact Jacobian, used to find zeros, or the exact Hessian, used to find extrema. The modified versions use an approximate type of Jacobian and Hessian matrix that is built from the gradient as the function progresses. The new way is called the Quasi Newton method. Because we are looking at finding the minimum, the Quasi Newton method is nice because the exact Hessian often requires a lot of work.

Its workings for finding the minimum are actually a side effect of finding zeros. Instead of finding zeros of the function given, the algorithm searches for the zeros of the gradient. If

$$J\left(x_1, x_2, x_3, \cdots x_n\right) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \cdots & \frac{\partial f_3}{\partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \frac{\partial f_n}{\partial x_3} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Fig. 5. Jacobian Example

using the full blown method, we would have to convert the Jacobian into a Hessian, but thanks to this exploit, the quasi newton method does not require either of those.

William Davidson was struggling with an approximation problem, that used a different optimization method. Every time he tried to run it the computer would crash. This is why he came out with the first version of the quasi newton method in 1959. Its not quite as old as the conjugate gradient method, but it is still pretty old. It has many descendants including, DFP, SR1, BHHH, BFGS, and others. All of these methods change the performance of the quasi newton function, but at their core, they use the same principles as the original.

Some benefits over the normal newton method is it is computationally cheaper, faster, no second derivatives,and no linear equations. THe drawback is that it has more convergence steps and is less precise.

III. Computations

Here is a few examples of some different types of minimzation mehtods from the scipy liraries computing different functions in comparison to each other.

$$x^4 + 2 * x^3 + 3 * x$$

CG time: 88.1942675113678 sec
TNC time: 37.70553803443909 sec
BFGS time: 61.31789040565491 sec

$$x^2 + 2 * x + 3$$

CG time: 32.65203809738159
TNC time: 24.223828077316284
BFGS time: 25.20459508895874

$$y^4 - x^2 * y^2 - 3 * y$$

CG time: 164.2236795425415
TNC time: 31.774510860443115
BFGS time: 56.557291746139526

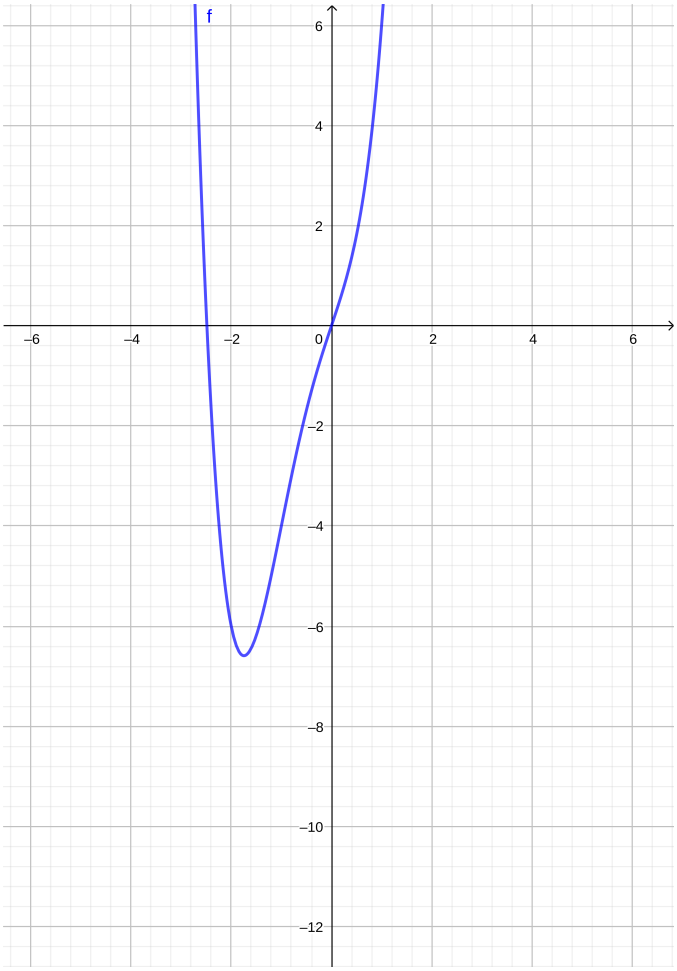

Fig. 6. $x^4 + 2 * x^3 + 3 * x$

IV. Conclusion

The fastest function that was demonstrated today was the TNC, which is short of the truncated Newton method. It is one of the many quasi Newton methods. The next fastest function was the BFGS function, which did a good job and almost won in the quadratic function. BFGS stands for Broyden, Fletcher, Goldfarb, Shanno, which are the creators. It is also a quasi Newton method. Last place was the conjugate gradient, which is not surprising. THe other methods are continually updating while the original conjugate gradient formula that we used has stayed the same. All of the functions are quite good at what they do, and especially the built in functions. Without a computer, none of this would be possible, meaning that we would be held back from many scientific discoveries do to the lack of computers, which make the problems trivial.

Going through this project has taught me how important building on previous successes are. Almost all minimization functions that we have studied involve the Gradient Descent at their heart. Even with it being old, it is still one of the most used algorithms in machine learning. Its also one of the easiest to understand. Simple is a good things, as long as it works.

Having all of these functions at our fingertips is a blessing that we take for granted. Without these methods, we wouldnt

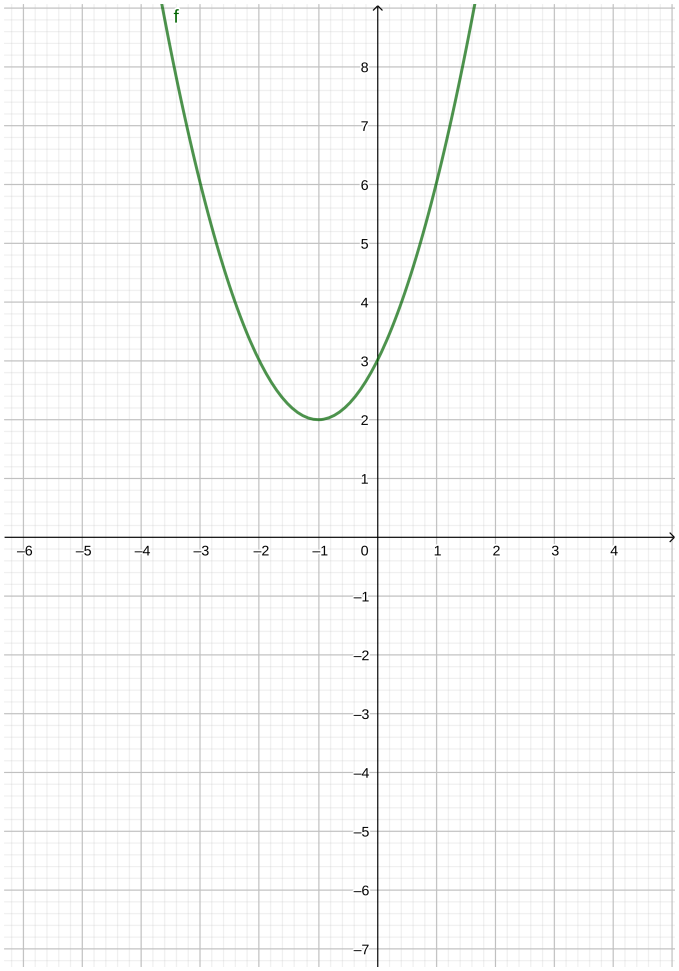

Fig. 7. $x^2 + 2 * x + 3$

be able to get the answers with the precision and speed we get them now, if we could get them at all. Powerful algorithms and computers are an essential part of our future progression, same as the past.

REFERENCES

[1] Optimization. Merriam-Webster, Merriam-Webster, www.Merriam-webster.Com/dictionary/optimization.