Quick Gradient Descent Review

February 12, 2019

Introduction to Gradient Descent

- Let $\mathbf{x} = [x[0], \cdots, x[N-1]]$ be the usual measurements modeled by the distribution $p(\mathbf{x}; \theta)$
- Disclaimer: This is not the ESE optimization class, so notation and derivations are probably not the same. In particular we are only interested in optimizing two families of functions:
- Frequentist Case: $\arg\min_{\theta}(-\log(p(\mathbf{x}|\theta))) = \arg\min_{\theta}(-\Sigma_{n=0}^{N-1}\log(p(x[n];\theta)))$
- Bayesian Case: $\arg\min_{\theta}(-\log\frac{p(\mathbf{x}|\theta)p(\theta)}{p(\mathbf{x})}) =$
  $\arg\min_{\theta}(-\Sigma_{n=0}^{N-1}\log(p(x[n]|\theta)) - log(p(\theta)) + log(p(\mathbf{x})))$
- Sometimes there are constraints on θ, e.g. in a Poisson distribution $\theta > 0$.
- The minimum then occurs either on the boundary of the constraints or where the gradient is a vector of zeros.

Gradient descent

- The gradient defines a vector whose direction is the greatest "ascent", so the opposite direction is the steepest descent.
- Frequentist case: The Gradient Descent formula for the k-th iteration is

$$\theta_{k+1} = \theta_k - \alpha_k \Sigma_{n=0}^{N-1} \nabla \log(p(x[n]|\theta_k))$$

```
theta_h = theta-.05; % need to start close to the solution or it ...
    blows up
 err = 1;
 err_threshold = .001;
 i = 1;
 step_size = .000001; %there are sophisticated ways of choosing ...
     the step size that I am choosing to ignore
 L = 0;
 while err>err_threshold
     prev_DL = D_L;
    %compute likelihood stuff
    [L(i),D_L,I] = Log_Likelihood(x,theta_h(i,:), s2,N);

    theta_h(i+1,:) = theta_h(i,:) - step_size*D_L;
    err = norm(D_L,2); % want the gradient to go to 0;
    i = i+1
 end
```

When to stop, and how to choose the step size?

- At a local minimum, the gradient should be 0. In practice we choose some small threshold ϵ, and compute

$$||\Sigma_{n=0}^{N-1} \nabla \log(p(x[n]|\theta_k))||_2$$

  every iteration as a stopping criteria.
- To choose the step size, pick α_k such that you travel as far along in the gradient direction as possible. i.e. minimize

$$h(\alpha_k) = \Sigma_{n=0}^{N-1} \log(p(x[n]|\theta_k + \alpha_k \Sigma_{n=0}^{N-1} \nabla \log(p(x[n]|\theta_k))))$$

- Now we've added another optimization problem to our first one! We could theoretically use gradient descent on this problem too and keep recursively choosing step sizes.
- Instead we usually use a simpler algorithm, such as the bisection or secant method since the derivative with respect to alpha is not hard to compute.
- Choosing the step size is usually called Line Search.

Example code for bisection line search

Example code for the bisection method, taken from "Numerical Analysis", by Tim Sauer. Here, h is the same function computed on the previous slide.

```
1  %Program 1.1 Bisection Method
2  %Computes approximate solution of f(x)=0
3  %Input: inline function f; a,b such that f(a)*f(b)<0,
4  %     and tolerance tol
5  %Output: Approximate solution xc
6  function xc = bisect(h,a,b,tol)
7  if sign(h(a))*sign(h(b)) ≥ 0
8    error('f(a)f(b)<0 not satisfied!') %ceases execution
9  end
10 ha=h(a);
11 hb=h(b);
12 k = 0;
13 while (b-a)/2>tol
14   c=(a+b)/2;
15   fc=h(c);
16   if fc == 0              %c is a solution, done
17     break
18   end
19   if sign(fc)*sign(ha)<0  %a and c make the new interval
20     b=c;hb=fc;
21   else                    %c and b make the new interval
22     a=c;ha=fc;
23   end
```

Another way to choose the Step Size

- A fancier way to choose the step size is to use the Barzilai-Borwein Method:

$$\alpha_k = \frac{(x_k - x_{k-1})^T \left(\nabla \Sigma_{n=0}^{N-1} \nabla \log(p(x[n]|\theta_k)) - \Sigma_{n=0}^{N-1} \nabla \log(p(x[n]|\theta_{k-1}))\right)}{|| \left(\nabla \Sigma_{n=0}^{N-1} \nabla \log(p(x[n]|\theta_k)) - \Sigma_{n=0}^{N-1} \nabla \log(p(x[n]|\theta_{k-1}))\right) ||_2^2}$$

- This approach ends up decreasing the step size near the minimum, but using line search or BB on the functions we are interested in is guaranteed to converge to a Local Minimum.
- Global minima are only guaranteed when the function is convex - not always the case for us.
- The initial conditions θ_0 determine which local minima the algorithm will find.

Newton-Rhapson

- What if we set the step size for gradient descent with the Hessian matrix?
- Newton Rhapson Formula:
  $\theta_{k+1} = \theta_k - (\Sigma_{n=0}^{N-1} \frac{\partial^2 log(p(x[n]|\theta_k))}{\partial \theta_k \partial \theta_k^T})^{-1} \Sigma_{n=0}^{N-1} \nabla \log(p(x[n]|\theta_k))$
- Newton originally invented this method in the 1-D case to find roots of high degree polynomials.
- Note: Historically, the derivation of Newton-Raphson comes from the taylor expansion of the objective function, so it's not mathematically equivalent to Gradient Descent.
- Finds the optimal value faster than Gradient Descent, but also can fail more easily.
- Requires computing second derivatives, which don't always exist.

Other Common Modifications to the formula

- Let $H_k = \theta_k + \beta(\theta_k - \theta_{k-1})$
- Heavy Ball Method:

$$\theta_{k+1} = H_k - \alpha_k \Sigma_{n=0}^{N-1} \nabla \log(p(x[n]|\theta_k))$$

- Nesterov method :

$$\theta_{k+1} = H_k - \alpha_k \Sigma_{n=0}^{N-1} \nabla \log(p(x[n]|H_k))$$

- Both of these methods aim to speed up the process by incorporating the $k-1$ iteration into the formula.
- They converge on par with Newton's if the objective function has certain properties.

Momentum accelerated gradient descent

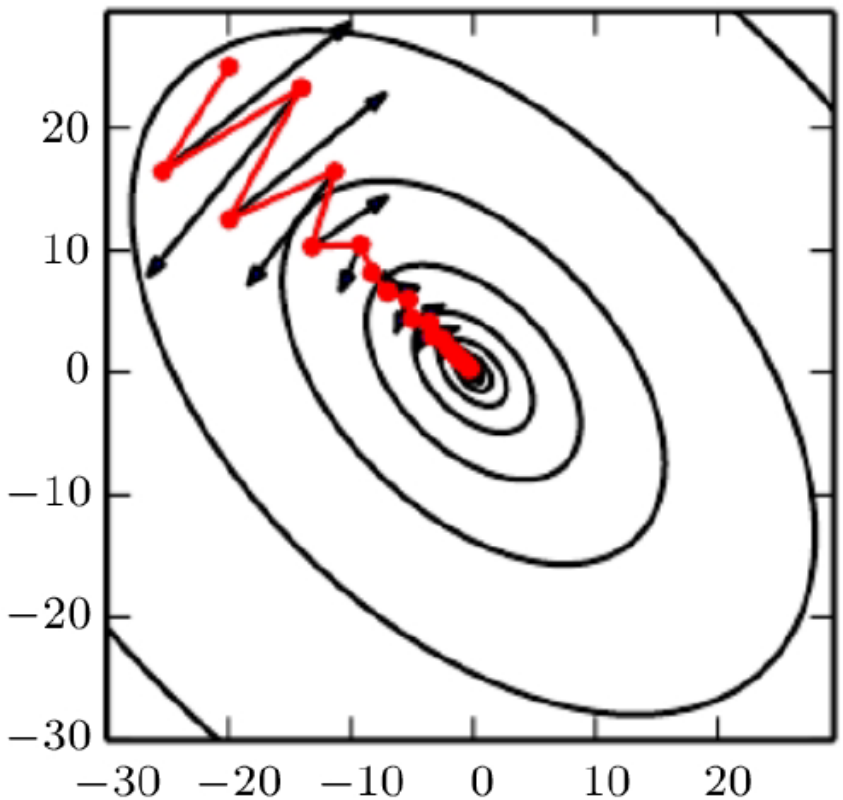

Figure 1: Momentum based methods avoid zigzagging issues that can occur

Stochastic Gradient Descent

- If N = 1,000,000, we end up doing a lot of repetitive computation.
- Instead, choose n_k at random from $0, ..., N-1$ every iteration.
- Stochastic Gradient Descent Formula $\theta_{k+1} = \theta_k - \alpha_k \nabla \log(p(x[n_k]|\theta_k))$.
- Needs more iterations but if an iteration is faster than the original computation the overall program runs faster.

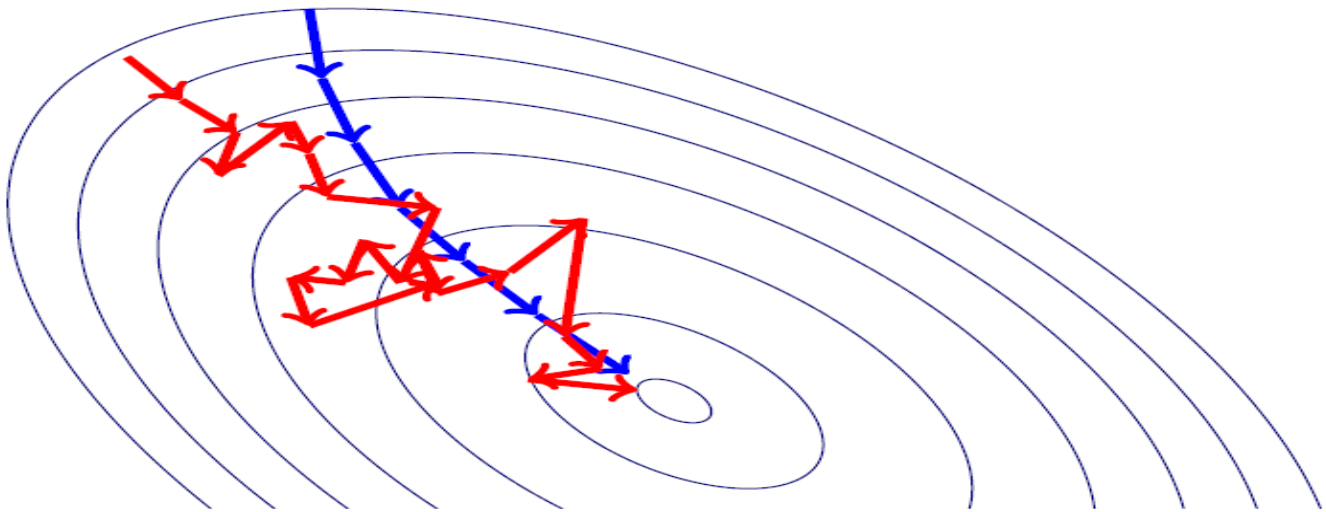

Figure 2: Red line represents the search in stochastic gradient descent, blue line represents normal gradient descent.

Comments

- If the Hessian matrix has a bad condition number or is high dimensional (a lot of parameters), computing it's inverse doesn't work. So in high dimensional cases gradient based methods are used instead of Newton.
- For non-convex problems, we can only find local minima. To mitigate, run the algorithm either close to your desired point or run many times with random initial conditions.
- Sometimes gradients don't have good closed form solutions - use finite differences to approximate.