We want to minimize a convex, continuous, and differentiable loss function $\ell(\omega)$.

Algorithm

Initialize $\bar{w}_0$

Repeat until convergence {

$\bar{w}^{t+1} = \bar{w}^t + \bar{s}$

If $\|\bar{w}^{t+1} - \bar{w}^t\|_2 < \varepsilon$

break

}

Trick: Taylor Expansion

How can you minimize a function, ℓ, which you don't know much about? The trick is to assume it's simpler than it actually is.

This can be done w/ Taylor's approximation.

Provided the norm $\|\vec{s}\|_2$ is small, (i.e $\vec{w} + \vec{s} \approx \vec{w}$), we can approximate the function $\ell(\vec{w} + \vec{s})$ by its first and second derivatives:

$$\ell(\bar{w} + \bar{s}) \approx \ell(\bar{w}) + g(\bar{w})^T \bar{s}$$

Assumes ℓ is twice differentiable and more expensive to compute, but is more accurate

$$\ell(\bar{w} + \bar{s}) \approx \ell(\bar{w}) + g(\bar{w})^T \bar{s} + \frac{1}{2} \bar{s}^T H(\bar{w}) \bar{s}$$

Here, $g(\bar{w}) = \nabla \ell(\bar{w})$ and $H(\bar{w}) = \nabla^2 \ell(\bar{w})$.

Both approximations are valid if $\|\bar{s}\|_2$ is small.

Gradient Descent: Use First Order Approximation

In gradient descent we only use the gradient.

In other words, we assume the function ℓ around $\bar{w}$ is linear and behaves like $\ell(\bar{w}) + g(\bar{w})^T \bar{s}$.

Our goal? Find a vector $\bar{s}$ that minimizes this function.

In steepest descent we simply set

$$s = -\alpha g(\bar{w})$$

for some small $\alpha > 0$.

Straightforward to show

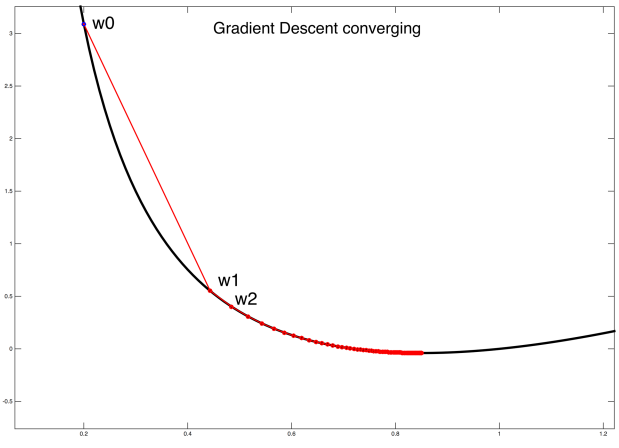
$$\underbrace{\ell(\bar{w} + (-\alpha g(\bar{w})))}_{\text{after one uptate}} \approx \ell(\bar{w}) - \underbrace{\alpha g(\bar{w})^T g(\bar{w})}_{>0} < \underbrace{\ell(\bar{w})}_{\text{before}}$$

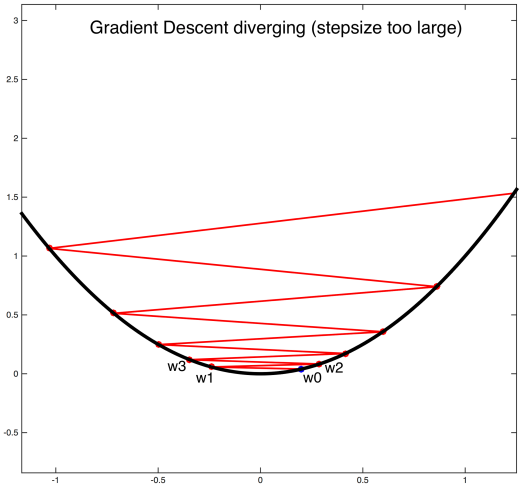
Setting the learning rate $\alpha > 0$ is a dark art; takes practice.

Too large, we diverge.

Too small, will converge but takes a lot of computations.

Safe choice is $\alpha = \frac{t_0}{t}$, which guarantees it will become small enough to converge; but play around with it.





Adagrad

One option is to set the step-size **adaptively** for every feature. Adagrad keeps a running average of the squared gradient magnitude and sets a **SMALL** learning rate for features w/ large gradients, and a **LARGE** learning rate for features w/ small gradients.

Algorithm

Initialize $\vec{w}_0$ and $\vec{z}$: $\forall d: w_d^0 = 0$ and $z_d = 0$

Repeat until convergence {

$$\vec{g} = \frac{\partial f(\vec{w})}{\partial \vec{w}}$$

$$\forall d: z_d \longleftarrow z_d + g_d^2$$

$$\forall d: w_d^{t+1} \longleftarrow w_d^t - \alpha \frac{g_d}{\sqrt{z_d + \varepsilon}}$$

If $\|\vec{w}^{t+1} - \vec{w}^t\|_2 < \delta$

break

}

Newton's Method: Use 2nd Order Approximation

Assumes $\ell(\vec{w})$ is twice differentiable and uses the approximation w/ Hessian

The Hessian Matrix is defined as

$$H(\vec{w}) = \begin{bmatrix} \frac{\partial^2 \ell}{\partial w_1^2} & \frac{\partial^2 \ell}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 \ell}{\partial w_1 \partial w_n} \\ \vdots & \cdots & \cdots & \vdots \\ \frac{\partial^2 \ell}{\partial w_n \partial w_1} & \cdots & \cdots & \frac{\partial^2 \ell}{\partial w_n^2} \end{bmatrix}$$

Due to the convexity of ℓ, it is always a symmetric square matrix and positive semi-definite

Note: A symmetric matrix M is positive semi-definite if it has only non-negative eigenvalues or, equivalently, for any vector $\vec{x}$ we must have $\vec{x}^T M \vec{x} \geq 0$

It follows that the approximation

$$\ell(\vec{w}+\vec{s}) \approx \ell(\vec{w}) + g(\vec{w})^T \vec{s} + \frac{1}{2} \vec{s}^T H(\vec{w}) \vec{s}$$

describes a convex parabola, and we can find the minimum by solving the following optimization problem:

$$\underset{s}{\operatorname{argmin}} \; \ell(\vec{w}) + g(\vec{w})^T \vec{s} + \frac{1}{2} \vec{s}^T H(\vec{w}) \vec{s}$$

$\downarrow \partial/\partial s(\,) = 0$

$$g(\vec{w}) + H(\vec{w})\vec{s} = 0$$

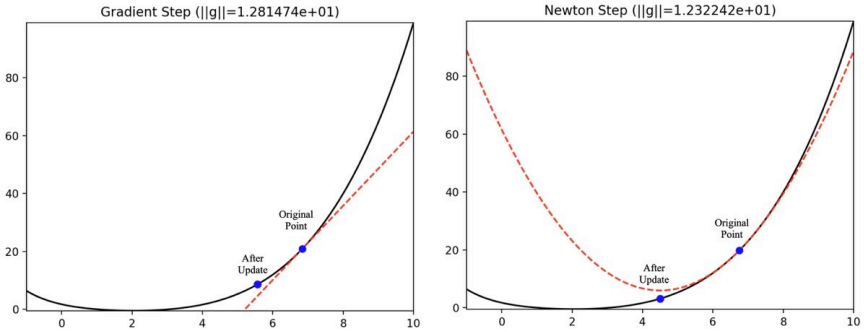
$$\vec{s} = -\left[H(\vec{w})\right]^{-1} g(\vec{w})$$

This choice of $\vec{s}$ converges very fast if the approximation is sufficiently accurate and the resulting step sufficiently small. Otherwise it can diverge.

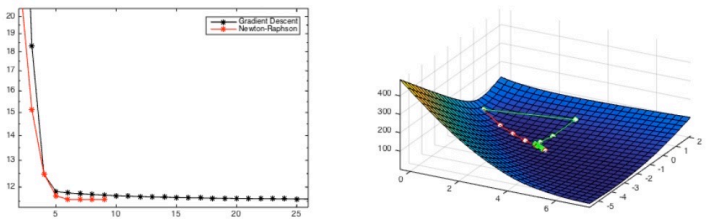
Divergence happens when the function is nearly flat/flat along a dimension. Then the second derivative becomes small and its inverse large → big steps.

Best Practices

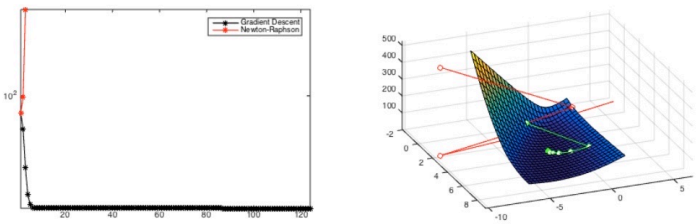
1. The matrix $H(\mathbf{w})$ scales $d \times d$ and is expensive to compute. A good approximation can be to only compute its diagonal entries and multiply the update with a small step-size. Essentially you are then doing a hybrid between Newton's method and gradient descent, where you weigh the step-size for each dimension by the inverse Hessian.
2. To avoid divergence of Newton's method, a good approach is to start with gradient descent (or even stochastic gradient descent) and then finish the optimization Newton's method. Typically, the second order approximation, used by Newton's Method, is more likely to be appropriate near the optimum.



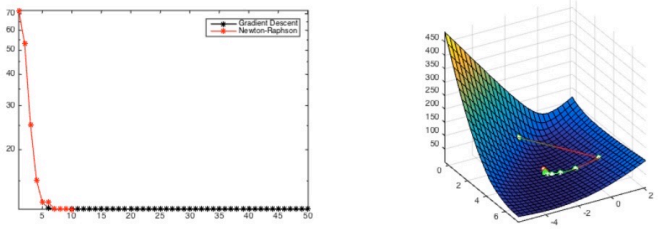
A gradient descent step (left) and a Newton step (right) on the same function. The loss function is depicted in black, the approximation as a dotted red line. The gradient step moves the point downwards along the linear approximation of the function. The Newton step moves the point to the minimum of the parabola, which is used to approximate the function.



(a) A starting point where Newton's Method converges in 8 iterations.



(b) A starting point where Newton's Method diverges.



(c) same starting point as in Figure 2, however Newton's method is only used after 6 gradient steps and converges in a few steps.

The three plots show a comparison of Newton's Method and Gradient Descent. Gradient Descent always converges after over 100 iterations from all initial starting points. If it converges (Figure 1), Newton's Method is much faster (convergence after 8 iterations) but it can diverge (Figure 2). Figure 3 shows the hybrid approach of taking 6 gradient descent steps and then switching to Newton's Method. It still converges in only 10 updates.