

Optimization

1 Overview

So far we have studied what to optimize. Often, a learning model can be phrased constructed that has some set of unspecified parameters (for example the weights in a neural network, the number of hidden layers, and the number of hidden units in each layer). Under certain conditions, one gets good generalization, namely having fixed the learning model, the best thing to do within that learning model is to minimize the empirical risk R_{emp} . However, good generalization is not equivalent to a low test error R . One also needs to incorporate prior information. In general one might want to minimize some regularized empirical risk $E(\mathbf{w}) = R_{emp}(\mathbf{w}) + \lambda\Omega(\mathbf{w})$. Thus it is useful to be able to minimize a general function $E(\mathbf{w})$. Optimization is not merely restricted its use for minimizing quantities of interest. For example, suppose we wished to find a solution to the vector equation $\mathbf{f}(\mathbf{x}) = \mathbf{y}$. One approach to finding a solution if it exists is to minimize the function $E(\mathbf{x}) = (\mathbf{f}(\mathbf{x}) - \mathbf{y})^T(\mathbf{f}(\mathbf{x}) - \mathbf{y})$. The study of techniques that perform this minimization is the topic of optimization and is independent of and is the learning, hence can be studied independently of learning, which is what we propose to do here.

We do not pretend to provide a comprehensive treatment of the subject of optimization, for it is truly a vast subject. However, it is hoped that the reader gets some idea of the richness of the subject and some general tools for approaching such problems. First the problem statement.

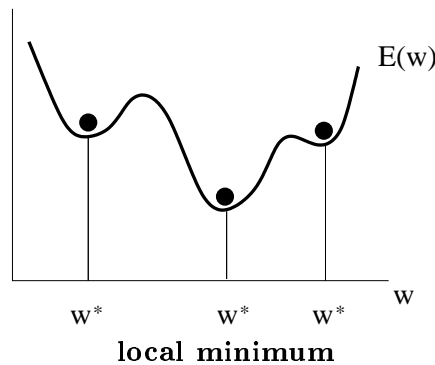
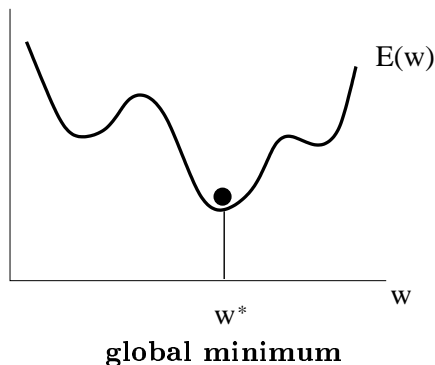
Definition 1.1 (Global Optimization: Problem Statement) *Assume that we are given some function $E(\mathbf{w})$ and a domain D . Find a $\mathbf{w}^* \in D$ such that*

$$E(\mathbf{w}^*) \leq E(\mathbf{w}) \quad \forall \mathbf{w} \in D \quad (1)$$

It turns out that this problem is hard in the formal sense that if this problem could be solved for an arbitrary $E(\mathbf{w})$ fast, then certain other “hard” problems whose fast solution are not known currently would be solved. Thus for our purposes we assume that this problem is “impossible” without further discussion. A problem with an almost identical (but ofcourse different) definition, but that proves to be much more manageable is as follows

Definition 1.2 (Local Optimization: Problem Statement) *Assume that we are given some function $E(\mathbf{w})$ and a domain D . Find a $\mathbf{w}^* \in D$ such that*

$$E(\mathbf{w}^*) \leq E(\mathbf{w}) \quad \forall \mathbf{w} \in D \text{ with } \mathbf{w} \text{ “sufficiently close” to } \mathbf{w}^* \quad (2)$$



The only difference is the term “sufficiently close” to \mathbf{w}^* , which has a formal definition of the form: for all \mathbf{w} in some neighborhood around the local optimum \mathbf{w}^* .

The difference between the solutions to these two problems is illustrated in the figures. Intuitively, one could find such a local optimum, \mathbf{w}^* , by starting a ball rolling on the surface, and following it until it stops dropping. Such method is not guaranteed to work for the global optimum and thus we intuitively see how we have simplified the problem.

Our general approach will be iterative. At time $t = 0$ we start at $\mathbf{w}(0)$ and we iteratively update $\mathbf{w}(t + 1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$, such that (hopefully) $\mathbf{w}(t + 1)$ is closer to a \mathbf{w}^* than $\mathbf{w}(t)$ was. Before we begin our development of the general approach, such iterative techniques beg the question how to start algorithm (how to choose the initial $\mathbf{w}(0)$) and how to stop the algorithm.

2 Initialization and Input Normalization

The approach to this issue depends on the specific nature of the problem, and we illustrate with the case of neural networks minimizing an empirical risk on some data set. For concreteness, we consider a neural network with one hidden layer, with N_H hidden units. We assume that all the activation functions are *tanh*, including the output unit.

Input normalization first puts the data set into a standard form. This is useful because then the same initialization technique can be used for any data set once it has been put into this form. The data set has N inputs \mathbf{x}_i , each assumed to be a d dimensional vector. We would like to transform them into a set of inputs \mathbf{z}_i such that each component of the \mathbf{z}_i 's has zero sample mean and unit sample variance. This is easily achieved by first transforming to the vectors $\mathbf{q}_i = \mathbf{x}_i - \mu$ where $\mu = \frac{1}{N} \sum_i \mathbf{x}_i$. Now, let σ_α be the sample standard deviation for the α component of the \mathbf{q}_i 's. Then, scaling the α component of each \mathbf{q}_i by $1/\sigma_\alpha$ to get \mathbf{z}_i we can achieve our goal because the variance of the α component of \mathbf{z} is $1/\sigma_\alpha^2$ times the variance of the corresponding component of \mathbf{q} which is thus 1. Thus we have for each component α

$$\frac{1}{N} \sum_{i=1}^N \mathbf{z}_i(\alpha) = 0 \quad \frac{1}{N} \sum_{i=1}^N \mathbf{z}_i(\alpha)^2 = 1 \quad (3)$$

We have gone to some pains to ensure that this is so for the data set. We will now see what this has bought us. Lets consider the input to a hidden unit β for data point i .

$$s_\beta = w_{0\beta} + \sum_{\alpha=1}^d w_{\alpha\beta} \mathbf{z}_i(\alpha) \quad (4)$$

We now assume that the weights $w_{\alpha\beta}$ are generated from a distribution with mean μ_w and variance σ_w^2 . Thus the expected value (with respect to $w_{\alpha\beta}$) of the sample mean (with respect to data points) of this input is $E[\mathbf{w}_{0\beta}] = \mu_w$. What about the variance?

$$\text{Var}(s_\beta) = \text{Var}(w_{0\beta}) + E \left[\frac{1}{N} \sum_{i=1}^N \sum_{\alpha=1}^d w_{\alpha\beta} \mathbf{z}_i(\alpha) \sum_{\delta=1}^d w_{\delta\beta} \mathbf{z}_i(\delta) \right] \quad (5)$$

$$= \sigma_w^2 + E \left[\sum_{\alpha=1}^d \sum_{\delta=1}^d \frac{1}{N} \sum_{i=1}^N w_{\alpha\beta} \mathbf{z}_i(\alpha) w_{\delta\beta} \mathbf{z}_i(\delta) \right] \quad (6)$$

where the contribution to the variance due to the term linear in $w_{0\beta}$ vanishes because $\frac{1}{N} \sum_{i=1}^N \mathbf{z}_i(\alpha) = 0$. We make the simplifying assumption that $\frac{1}{N} \sum_{i=1}^N w_{\alpha\beta} \mathbf{z}_i(\alpha) w_{\delta\beta} \mathbf{z}_i(\delta) = 0$ for $\alpha \neq \beta$ which corresponds to saying that the various input dimensions are uncorrelated. It turns out that this assumption could have been avoided by using a different more complicated form of input normalization, but, at the expense of more complicated algebra. To avoid this situation, we will make the assumption with the knowledge that it can be avoided. In this case, we get that

$$\text{Var}(s_\beta) = (d + 1)\sigma_w^2 \tag{7}$$

Now since we want the inputs to the activation to experience the non-linearity of the activation, we want $\text{Var}(s_\beta) \approx 1$ and we want the mean to be zero, we see that we should choose $\mu_w = 0$ and $\sigma_w^2 = 1/(d + 1)$. A similar argument for the weights from the hidden to output layer gives approximately that the variance of those weight initializations should be $1/(N_H + 1)$ (with mean zero). This concludes our discussion of weight initialization by example.

3 When to Stop Training

There is no standard approach to this issue, and one usually approaches it using heuristics. We just describe a few.

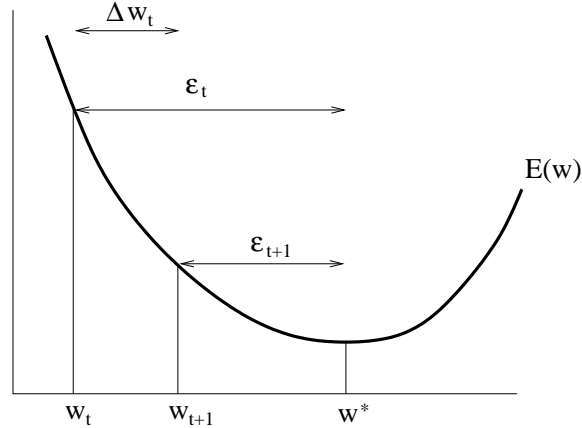
1. Train for a fixed number of iterations or a fixed number of CPU cycles. This has the advantage of being a bounded algorithm, however, different problems need different amounts of training time. This can be taken into account by setting the fixed number of iterations to be huge. However, for small problems that only require a few iterations, this might be inefficient.
2. Train until $E(\mathbf{w}) \leq E_{tol}$ for some tolerance value. This however is not a bounded algorithm as one does not know ahead of time whether such a tolerance can be reached.
3. Train until $\Delta E(\mathbf{w}) \leq \delta_{tol}$ for some tolerance value. Once again this is not a bounded algorithm. Further, it can lead to premature stopping, for example, if one hits a particularly flat part of the error surface that is not near the minimum. This occurs in practice more often than one might suspect.

Usually a combination of the above methods works best. For example criterion (3) provided that at least some number of iterations have been done or criterion (2) with the restriction that at most some number of iterations can be done. Since stopping is largely a heuristic issue, we do not elaborate any further. We now proceed to the actual development of the optimization algorithms themselves.

4 General Approach to Optimization

As already mentioned, the general approach will be iterative.

$$\mathbf{w}_{t+1} = w_t + \Delta \mathbf{w}_t \tag{8}$$



In general, such iterative algorithms do not get to the local minimum itself. In comparing optimization algorithms, two issues are important. How close one can get to the local minimum and how long it takes to get there. In general one asks one of the following questions of an optimization algorithm

1. Fix an error tolerance with respect to the locally minimal value (either a relative error or absolute error). How long does it take to get to within the tolerance of the locally minimal value?
2. Fix an amount of CPU time. How close (in terms of relative or absolute error) does the algorithm get you to the locally minimal value?

In order to analyse the convergence behavior near the minimum, we look at the residual $\epsilon_t = \mathbf{w}_t - \mathbf{w}^*$. Once we are in the vicinity of the local minimum ($\|\epsilon_t\| < 1$), we are generally interested in convergence behavior of the form $\|\epsilon_{t+1}\| \leq k\|\epsilon_t\|^L$. k is the coefficient of convergence and L is the order of convergence. $L = 1$ is called linear convergence and $L = 2$ is called quadratic convergence. $L > 1$ is called super-linear convergence. It is easy to show by induction that for two times t and $t + \tau$

$$\epsilon_{t+\tau} \leq \begin{cases} k^\tau \epsilon_t & L = 1 \\ k^{(\frac{L^\tau - 1}{L - 1})} \epsilon_t^{L^\tau} & L > 1 \end{cases} \quad (9)$$

We see that the convergence increases very quickly as k decreases and as L increases. In fact there is a qualitative difference between $L = 1$ (exponential convergence in time) and $L > 1$ (super-exponential convergence in time). It is very rare to find methods with $L > 1$. One generally satisfies oneself with linear convergence and tries to beat k down as much as possible. Note that for sufficiently smooth surfaces, convergence of $\epsilon_t \rightarrow 0$ also means that the error itself converges to a locally minimal value.

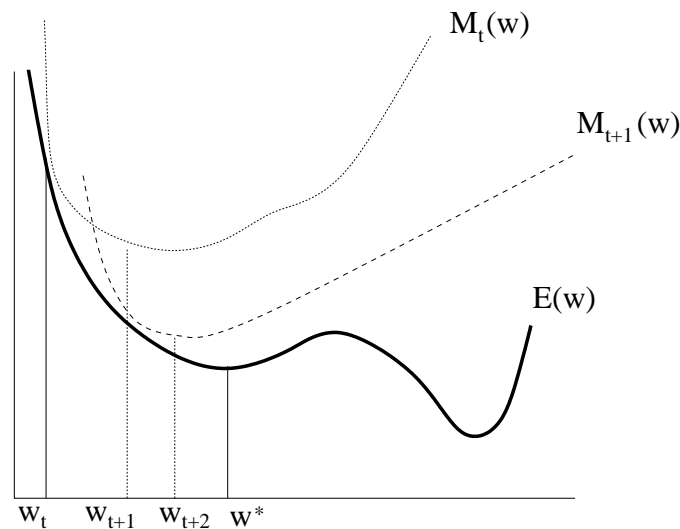
4.1 Iterative Model Estimation-Minimization

We now discuss how we will set about developing the iterative algorithm. The general idea can be broken up into the following iterative scheme

1. At $t = 0$, initialize the parameter to \mathbf{w}_0 .

2. At time step t , build a model of the error surface ($M_t(\mathbf{w})$) using whatever information one has access to.
3. Assuming that the model adequately represents the error surface $E(\mathbf{w})$, use the model to determine \mathbf{w}_{t+1} . This is usually done by setting \mathbf{w}_{t+1} to be the minimum suggested by the model.
4. Update the weight to w_{t+1} , and increment the time step to $t + 1$ and iterate back to step (2) until the stopping criterion is reached.

This general approach is illustrated in the following figure



Two general guidelines for the model that one builds are that it should be easy to build the model and that it should be easy to use the model (i.e. analyse it so as to use it to guide the move). Step (3) will only work if the model is assumed to somewhat resemble the true error surface. However, the two cannot be identical for then we have not made the problem any more tractable than it already was. Thus it is usually reasonable to assume that the model is faithful to the true error surface in some region close to the current point \mathbf{w}_t . Further away, the model may be drastically inaccurate. We will call this region where the model is valid, the *Domain of Validity* (DOV) of the model. This DOV must be kept in mind when using the model to make the next move in step (3). If the model suggests that we move to a point that is outside the DOV, then one needs to be careful in following the model suggestion and it may be necessary to fall back upon a simpler model.

What kinds of information can we assume we have access to when building the model? We assume that at some cost, one can compute the error function itself, $E(\mathbf{w})$ at any point \mathbf{w} , and its derivatives. The models can then be categorized into a hierarchy as follows

1. Zeroth order models: Models that compute only $E(w)$.
2. First order models: Models that have access to $E(w)$ and $\nabla E(\mathbf{w})$ for any assignment of the parameters. Usually such models are linear in the parameter space. One could “cheat” and claim that one is using a zeroth order model and use function evaluation to build up derivative information. We consider such “cheating” as a first order method, and only bona-fide zeroth

order methods fall into class (1). Usually derivative information is more costly than function evaluation. So, one usually tries to minimize the number of calls to the derivative information. However derivative information allows for a better model, hence each iteration will be more efficient.

3. Second order models: Models that have access to the function value, first derivative (gradient), and second derivative (Hessian) information for any assignment of the parameters. Usually such models are quadratic in the parameter space. Since Hessian information is even more costly than gradient information, such models are harder to build but give lots of information once they are built. Thus in general there is a trade off: more information is costly but allows for a better model to guide the search.

One could continue along this hierarchy, considering third, fourth etc. order models. Such models are also feasible to construct, but then the problem becomes that they are extremely hard to analyse, hence the returns are diminishing beyond second order.

4.2 Zeroth Order Models: Monte Carlo and Exhaustive Search

We consider two extremely naive first order approaches to minimization. The motivation is that the techniques actually obtain approximate solutions to the global minimum. However their drawback is that if the dimension of the parameter space is d , then their run time is essentially exponential in d . But any algorithm for global optimization is exponential in d so despite our methods being naive, they essentially tell the whole story.

We assume that we would like to find the minimum of $E(\mathbf{w})$ for \mathbf{w} in some compact d -dimensional domain. Further, without loss of generality, we assume that this domain is the unit cube, $D = [0, 1]^d$, for suppose that g is an invertible mapping from the compact domain to D (such a mapping must exist). Then, consider the function $E'(\mathbf{w}) = E(g^{-1}\mathbf{w})$. E' is the squished version of E onto the cube D . Minimizing E' on D and transforming the solution using g^{-1} is equivalent to minimizing E on the original compact domain.

4.2.1 Montecarlo Minimization

We simply state the algorithm as it is self explanatory. Construct a uniform random sample of size N of parameters $\mathbf{w}_1, \dots, \mathbf{w}_N$. As we sample more points ($N \rightarrow \infty$), we need to do more and more evaluations, but at the same time we expect to get closer to the true minimum. In fact, one can show that for reasonably well behaved error surfaces, for any tolerance $\epsilon > 0$, the probability that one gets to within ϵ of the minimal value approaches 1 as $N \rightarrow \infty$. This fact is sometimes slightly inaccurately phrased one obtains the global minimum with probability 1 as $N \rightarrow \infty$.

Advanced Topic: Convergence Analysis for Montecarlo Minimization

Suppose that the error surface admits a second order approximation in the vicinity \mathbf{w}^* and the global minimum occurs at an interior point and is unique. Some of these assumptions can be relaxed, but we make them to facilitate easier analysis. We assume that we sample from the cube uniformly, although this assumption can also be relaxed. By assumption, near the

minimum, we can write

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \underbrace{\frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*)}_{\leq \epsilon} \quad (10)$$

where \mathbf{H} is a positive definite, symmetric matrix (the Hessian at \mathbf{w}^*). Thus, for ϵ sufficiently small, such that the second order approximation is valid and that the only area where the error can drop below $E(\mathbf{w}^*) + \epsilon$ is in the vicinity of \mathbf{w}^* ,

$$P[E \leq E(\mathbf{w}^*) + \epsilon] = P[\frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*) \leq \epsilon] \quad (11)$$

$$= \int_{\mathbf{x}^T \mathbf{H} \mathbf{x} \leq 2\epsilon} d^d \mathbf{x} \quad (12)$$

$$\stackrel{(a)}{=} \frac{1}{\sqrt{\det \mathbf{H}}} \int_{\|\mathbf{x}\|^2 \leq 2\epsilon} d^d \mathbf{x} = \frac{S_d(2\epsilon)}{\sqrt{\det \mathbf{H}}} \quad (13)$$

where (a) is obtained by making the following changes of variables. First, there exists an \mathbf{A} such that $\mathbf{A}^T \mathbf{A} = \mathbf{I}$ and $\mathbf{A}^T \mathbf{H} \mathbf{A} = \text{diag}[\lambda_1^2, \dots, \lambda_d^2]$. Thus, the first change of variables is to $\mathbf{u} = \mathbf{A}^T \mathbf{x}$, and the next change of variables is to $v_i = \lambda_i u_i$. Then, noting that $\text{Det}(\mathbf{H}) = \lambda_1^2 \lambda_2^2 \dots \lambda_d^2$, the result (a) follows. We use the notation $S_d(r)$ to represent the volume of the d -dimensional sphere of radius r . It is not hard to show that

$$S_d(r) = \frac{\pi^{d/2} r^d}{\Gamma(\frac{d}{2} + 1)} \quad (14)$$

Using the fact that $\Gamma(x+1) \approx x^x e^{-x} \sqrt{2\pi x}$ and the fact that the probability that the observed lowest E is greater than $E(\mathbf{w}^*) + \epsilon$ is given by $(1 - P[E < E(\mathbf{w}^*) + \epsilon])^N$, we find that

$$P[E_{min} > E(\mathbf{w}^*) + \epsilon] \approx \left(1 - \left(\mu(d) \frac{\epsilon}{\sqrt{d}}\right)^d\right)^N \quad (15)$$

where $\mu(d) = \sqrt{\frac{8e\pi^{\frac{d-1}{d}}}{(d \det \mathbf{H})^{1/d}}}$. So for fixed ϵ sufficiently small, we see that $P[E_{min} > E(\mathbf{w}^*) + \epsilon] \xrightarrow{N \rightarrow \infty} 0$. Fix $\eta > 0$, then to ensure that $P[E_{min} > E(\mathbf{w}^*) + \epsilon] < \eta$ we require that

$$N \geq \frac{\log \eta}{\log \left(1 - \left(\mu(d) \frac{\epsilon}{\sqrt{d}}\right)^d\right)} \approx \left(\frac{\sqrt{d}}{\mu(d)\epsilon}\right)^d \log \frac{1}{\eta} \quad (16)$$

Since $\det \mathbf{H} \sim \alpha^d$ where α would be the geometric mean of its eigenvalues which we assume exists, and $d^{1/d}$ converges to 1, we see that as $N \sim \left(\frac{\sqrt{d}}{\mu\epsilon}\right)^d$ as. In otherwords, to achieve a fixed accuracy as d increases, one needs to evaluate the function a number of times that is super-exponential in d .

4.2.2 Exhaustive Search

Suppose that we wish to obtain the minimum to an accuracy ϵ . One can grid each coordinate axis so that every point is at most a distance ϵ from a grid point. The side of the tiny grid cubes must then be ϵ/\sqrt{d} . One now evaluates the error function at each grid point ($\sim (\frac{\sqrt{d}}{\epsilon})^d$ function evaluations), and picks the minimum value. Unfortunately our original goal of finding a parameter vector at most ϵ away from the global minimum is not fulfilled. The Actual minimum could occur inside a grid cube that is far away. However as long as the error surface is well behaved, the value of the minimum we obtain cannot be too far off (up to a factor of ϵ). Once again we see that we need $\sim (\frac{\sqrt{d}}{\epsilon})^d$ function evaluations to get to within ϵ of the globally minimal value.

Suppose that the error function has a continuous gradient on the cube. Then this gradient has bounded norm, $\max_{\mathbf{w} \in D} \|\nabla E(\mathbf{w})\| = B$. If the grid is made up of cubes of size η , then the change in value of the function on one of the cubes is at most $B\eta\sqrt{d}$. If we want to get to within ϵ of the minimum (in value) by looking only at the function value on the grid points, then we require $B\eta\sqrt{d} \leq \epsilon$. And since the number of function evaluations is $\sim (1/\eta)^d$, we see that we need $\sim (B\sqrt{d}/\epsilon)^d$ function evaluations, very similar to the Monte Carlo minimization procedure.

With these two naive methods, we conclude our discussion of first order methods. They provide tools for global minimization that essentially tell the whole story. Namely that even under certain regularity conditions, the to guarantee coming arbitrarily close to the global minimum, one needs exponentially many (in the dimension of the problem) function evaluations.

4.3 First Order Methods

At time t we are at \mathbf{w}_t . We assume that we have access to the error function value, $E(\mathbf{w}_t)$ and its gradient $\nabla E(\mathbf{w}_t)$. To simplify notation, we will often write \mathbf{g}_t for $\nabla E(\mathbf{w}_t)$, when no confusion can arise. Then we can write

$$E(\mathbf{w}) \approx E(\mathbf{w}_t) + \mathbf{g}_t \cdot (\mathbf{w} - \mathbf{w}_t) \quad (17)$$

The first task is to determine what the best direction to move in is. A direction is defined by a unit vector $\hat{\mathbf{d}}$, and using (17), we see that

$$E(\mathbf{w}_t + \eta\hat{\mathbf{d}}) = E(\mathbf{w}_t) + \eta\mathbf{g}_t \cdot \hat{\mathbf{d}} \quad (18)$$

from which we conclude that

$$\frac{d}{d\eta} E(\mathbf{w}_t + \eta\hat{\mathbf{d}}) = \mathbf{g}_t \cdot \hat{\mathbf{d}} \geq -\|\mathbf{g}_t\| \quad (19)$$

so we see that the fastest rate of decrease in the error occurs when $\hat{\mathbf{d}} = -\hat{\mathbf{g}}_t$, where we use $\hat{\mathbf{g}}_t$ to denote the normalized version of \mathbf{g}_t . Thus, moving in the direction of the negative gradient gains most, and can be considered a locally greedy algorithm to minimization. If $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\hat{\mathbf{g}}_t$, then $E(\mathbf{w}_{t+1}) = E(\mathbf{w}_t) - \eta\|\mathbf{g}_t\|$. Since we are not at the minimum, $\|\mathbf{g}_t\| \neq 0$, hence, the conclusion is that if we use the model, we minimize E by choosing $\eta \rightarrow \infty$. This is clearly not a good idea. What has gone wrong? The problem is that while the model tells us to choose $\eta \rightarrow \infty$, this would place out of the DOV, as the linear model is only valid for small η . Thus, we are faced with two competing effects. We want to pick as large an η as possible, without going outside the DOV. Usually one picks some reasonable value for η , such as 0.01. This leads us to fixed learning rate

normalized gradient descent. If we let η decrease in proportion to the magnitude of the gradient, which seems to make sense since as we get closer to the minimum we want to take shorter steps, then we are led to fixed learning rate gradient descent.

4.3.1 Fixed Learning Rate Gradient Descent

The algorithm is as follows.

1. Initialize to \mathbf{w}_0 at $t = 0$.
2. Let $\mathbf{d}_t = -\mathbf{g}_t$.
3. Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \mathbf{d}_t$.
4. Increment t to $t + 1$ and iterate back to (2) until the stopping criterion is reached.

When using gradient descent to minimize R_{emp} on a data set, there are two commonly used approaches. In the batch mode, the error function is $R_{emp}(\mathbf{w})$ computed over the entire data set. In the sequential mode, one considers N error functions, $R_{emp}^{(i)}(\mathbf{w})$, the empirical risk on data point i . One then cycles through each of these error functions, performing a single gradient descent iteration on each error function. Since in the batch mode, each data points contribution to R_{emp} is multiplied by a factor of $\frac{1}{N}$, where as it is not in the sequential mode, it is often found that the sequential mode is more efficient as one is using an “effectively” higher learning rate. As the learning rate, η , approaches zero, the batch and sequential modes yield equivalent results. For finite learning rate, the results of the sequential mode may depend on the order in which the individual error functions are used, and one can get into cycles. It can be shown that if one allows the learning rate to decay with time so that $\sum_t 1/\eta_t = \infty$ and $\sum_t 1/\eta_t^2 < \infty$, then the sequential mode will converge to a local minimum (with probability one). For example one could choose $\eta_t = 1/(t + 1)$. However for most purposes, a fixed learning rate will work well enough.

Advanced Topic: Convergence of Fixed Learning Rate Gradient Descent.

Suppose that we are in the vicinity of a local minimum, \mathbf{w}^* , of the error surface, or that the error surface is quadratic. The expression for the error function is then given by

$$E(\mathbf{w}_t) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w}_t - \mathbf{w}^*)\mathbf{H}(\mathbf{w}_t - \mathbf{w}^*) \quad (20)$$

from which it is easy to see that the gradient is given by $\mathbf{g}_t = \mathbf{H}(\mathbf{w}_t - \mathbf{w}^*)$. The weight updates are then given by $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{H}(\mathbf{w}_t - \mathbf{w}^*)$, and subtracting \mathbf{w}^* from both sides, we see that

$$\epsilon_{t+1} = (\mathbf{I} - \eta \mathbf{H})\epsilon_t \quad (21)$$

Since \mathbf{H} is symmetric, one can form an orthonormal basis with its eigenvectors. Projecting ϵ_t and ϵ_{t+1} onto this basis, we see that in this basis, each component decouples from the others, and letting $\epsilon(\alpha)$ be the α^{th} component in this basis, we see that

$$\epsilon_{t+1}(\alpha) = (1 - \eta \lambda_\alpha)\epsilon_t(\alpha) \quad (22)$$

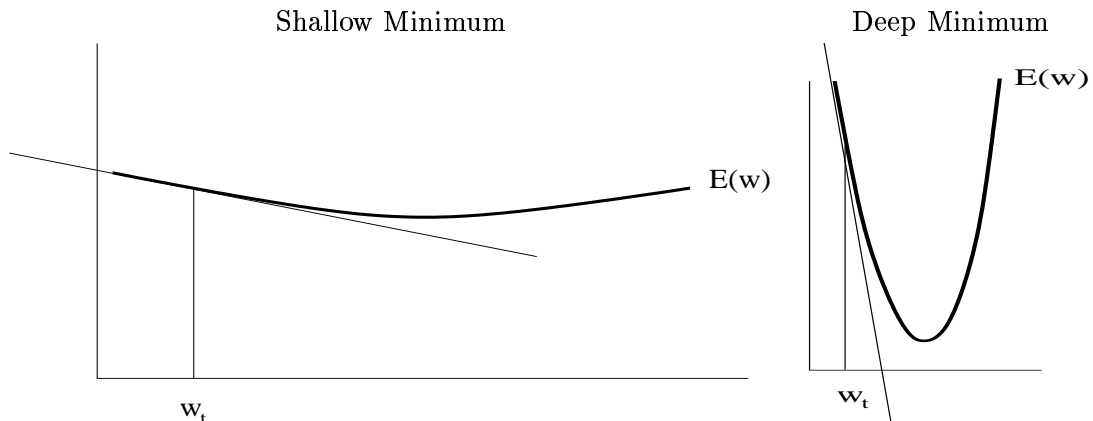
so we see that each component exhibits linear convergence with its own coefficient of convergence $k_\alpha = 1 - \eta\lambda_\alpha$. The worst component will dominate the convergence so we are interested in choosing η so that the k_α with largest magnitude is minimized. Since \mathbf{H} is positive definite, all the λ_α 's are positive, so it is easy to see that one should choose η so that $1 - \eta\lambda_{min} = 1 - \Delta$ and $1 - \eta\lambda_{max} = 1 + \Delta$, or one should choose. Solving for the optimal η , one finds that

$$\eta_{opt} = \frac{2}{\lambda_{min} + \lambda_{max}} \quad k_{opt} = \frac{1 - c}{1 + c} \quad (23)$$

where $c = \lambda_{min}/\lambda_{max}$ is the condition number of \mathbf{H} , and is an important measure of the stability of \mathbf{H} .

Gradient descent with fixed learning rate, though it works, tends to be slow especially near the minimum because the condition number may be very close to zero. Thus it is desirable to beef up gradient descent. The natural extension is to allow the learning rate to be variable.

4.3.2 Gradient Descent with Variable Learning Rate



The problem with a fixed learning rate is illustrated in the figure above. For shallow surfaces, the DOV can be large and thus a large learning rate is called for. Whereas if the surface is quite deep, then the DOV is small and we want to use a small learning rate. The general idea is to increase the learning rate (exponentially) when the previous iteration resulted in an error reduction, and to decrease the learning rate if it did not result in a decrease, in some sense, allowing the surface to tell us what an appropriate learning rate is. The algorithm is as follows.

1. Initialize to \mathbf{w}_0 at $t = 0$, and set η_0 to an initial value.
2. Let $\mathbf{d}_t = -\mathbf{g}_t$.
3. Let $\mathbf{x}_{t+1} \leftarrow \mathbf{w}_t + \eta_t \mathbf{d}_t$.
 - i. If $E(\mathbf{x}_{t+1}) < E(\mathbf{w}_t)$, accept the new weight vector, by setting $\mathbf{w}_{t+1} = \mathbf{x}_{t+1}$ and increment η to $\eta_{t+1} = \alpha\eta_t$ where $\alpha > 1$.
 - ii. If $E(\mathbf{x}_{t+1}) \geq E(\mathbf{w}_t)$, reject the new weight vector, by setting $\mathbf{w}_{t+1} = \mathbf{w}_t$ and decrease η to $\eta_{t+1} = \beta\eta_t$ where $\beta < 1$.

4. Increment t to $t + 1$ and iterate back to (2) until the stopping criterion is reached.

In practice one finds that it is usually good to be conservative on the increments ($\alpha = 1.03 - 1.05$), while it is better to be aggressive on the decrements ($\beta = 0.7 - 0.8$). After a little thought, one might wonder why we need a learning rate at all. Once the direction in which to move, \mathbf{d}_t , has been determined, why not simply continue along that direction until the error stops decreasing? This leads us to steepest descent.

4.3.3 Steepest Descent

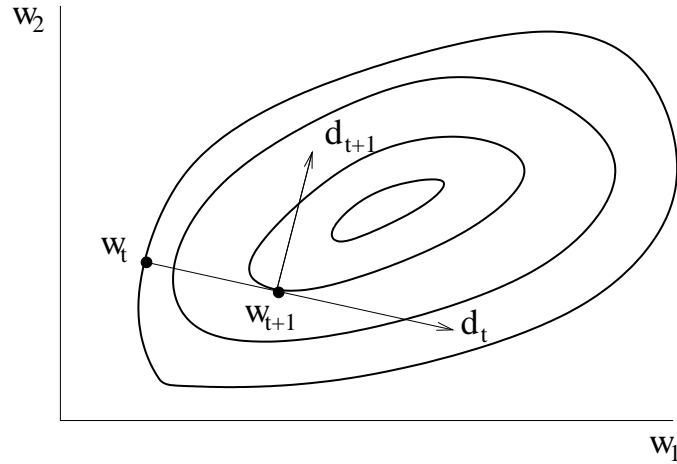
The algorithm is as follows.

1. Initialize to \mathbf{w}_0 at $t = 0$.
2. Let $\mathbf{d}_t = -\mathbf{g}_t$.
3. Consider the function of one variable λ given by $E(\mathbf{w}_t + \lambda\mathbf{d}_t)$. Increasing λ corresponds to moving in the direction \mathbf{d}_t away from \mathbf{w}_t . Continue to increase λ until the error stops decreasing. In other words find a local minimum λ^* of the one dimensional function $E(\mathbf{w}_t + \lambda\mathbf{d}_t)$.
4. Update the weight $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \lambda^*\mathbf{d}_t$.
5. Increment t to $t + 1$ and iterate back to (2) until the stopping criterion is reached.

The algorithm is self explanatory, however we will still have to discuss step (3). The algorithm seems rather circular in the sense that we are discussing optimization and requiring one to perform an optimization in the optimization algorithm itself. We now describe a method for performing this auxiliary optimization that is quite efficient in practice. First we describe geometrically what is going on, illustrating on a 2-d case. Suppose that one is moving along a direction until the error hits a local minimum along that direction. At that point, the component of the derivative of the gradient of the error in that direction must be zero, for suppose that it were less than zero, then by continuing more, the error would decrease further, contradicting the fact that we are at a local minimum. If it were greater than zero, we could retract a little and decrease the error, again contradicting the fact that we are at a local minimum. Thus, when we stop moving, it must be that $\mathbf{d}_t \cdot \nabla E(\mathbf{w}_t + \lambda^*\mathbf{d}_t) = 0$. Formally, this is easily seen by setting the derivative of $E(\mathbf{w}_t + \lambda\mathbf{d}_t)$ with respect to λ equal to zero:

$$\frac{d}{d\lambda}E(\mathbf{w}_t + \lambda\mathbf{d}_t) = \mathbf{d}_t \cdot \nabla E(\mathbf{w}_t + \lambda\mathbf{d}_t) = 0 \quad (24)$$

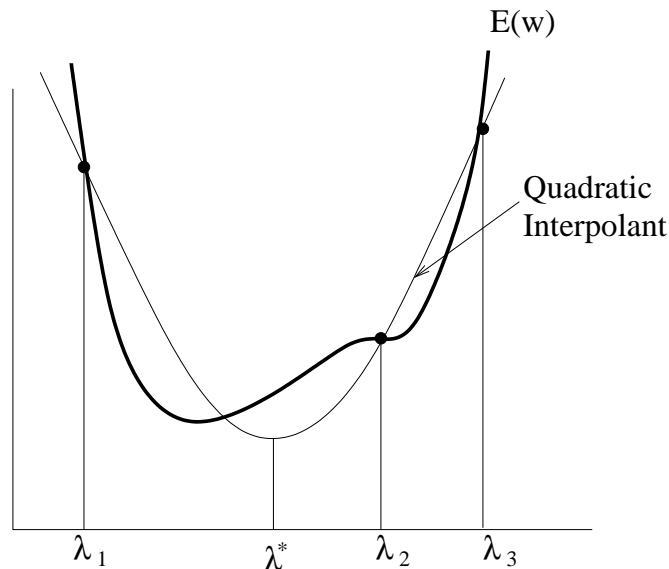
Pictorially, suppose that one considers the contours of constant E . The gradient at any point is perpendicular to the contour passing through that point. Thus we initially start moving perpendicular to the contour. We stop when the direction we are moving in is tangent to another contour, as is illustrated in the following figure.



These geometrical ideas will be used later in developing further techniques. We now discuss how to implement this one dimensional minimization in λ . This one dimensional minimization is sometimes referred to as *line-search*. For shorthand, we will write $E(\lambda)$ to mean $E(\mathbf{w}_t + \lambda \mathbf{d}_t)$.

Quadratic Interpolation Algorithm For Linesearch

1. The first step is to obtain 3 values of λ , $\lambda_1 < \lambda_2 < \lambda_3$ such that $E(\lambda_2) < E(\lambda_1)$ and $E(\lambda_2) < E(\lambda_3)$. For good reason, we will call this a *U-arrangement*. This is illustrated in the following picture.



The reason for finding such an arrangement of 3 points is because it is now known that a local minimum exists between λ_1 and λ_3 . How does one find such an arrangement? Start at $\lambda = 0$. Increment λ by some default step, that can be reasonably large. If the error increases, we have then found a candidate λ_1 and λ_1 . Now start halving the step size until one obtains a suitable λ_2 or stop if the precision of the machine has been reached. If the error decreased on the first step, one has found a candidate λ_1 and λ_2 . Now continue increase the

step geometrically until the error increases, and one has found a λ_3 . In practice, in the course of this search one might end up with many possible choices for the λ 's. In this lucky event, one should choose the set for which $\lambda_3 - \lambda_1$ is smallest (i.e., for which the location of the minimum is most narrowly defined).

2. Let $e_1 = E(\lambda_1)$, $e_2 = E(\lambda_2)$, $e_3 = E(\lambda_3)$. Obtain the quadratic function that interpolates the three points (λ_1, e_1) , (λ_2, e_2) , (λ_3, e_3) .
3. Let the minimum of this quadratic be at λ^* , and let $e^* = E(\lambda^*)$. λ^* is an estimate for the optimal λ .
4. There are 4 cases
 - i. If $\lambda^* < \lambda_2$ and $e^* < e_2$, then $\{\lambda_1, \lambda^*, \lambda_2\}$ forms another U -arrangement with a narrower range in which the minimum can lie. One can now iterate back to step (2).
 - ii. If $\lambda^* < \lambda_2$ and $e^* > e_2$, then $\{\lambda^*, \lambda_2, \lambda_3\}$ forms another U -arrangement. One can now iterate back to step (2).
 - iii. If $\lambda^* > \lambda_2$ and $e^* < e_2$, then $\{\lambda_2, \lambda^*, \lambda_3\}$ forms another U -arrangement. One can now iterate back to step (2).
 - iv. If $\lambda^* > \lambda_2$ and $e^* > e_2$, then $\{\lambda_1, \lambda_2, \lambda^*\}$ forms another U -arrangement. One can now iterate back to step (2).

What if $\lambda^* = \lambda_2$, a degenerate case. Then we perturb λ^* a little (by a tolerance) to get into one of the four cases above.

5. Perform the quadratic interpolation/minimization some fixed number of times, for example 3 times. (Usually 10 will be more than enough).

In general the quadratic interpolations converge very rapidly to the optimal λ . We do not present an analysis here. One note is that the method we have outlined above falls within our paradigm of “build a model” and then “use the model” to go to the minimum, where here the model we build is a quadratic.

To close the loop, it is not hard to obtain λ^* in terms of $\{\lambda_1, \lambda_2, \lambda_3, e_1, e_2, e_3\}$ as follows. Suppose that the interpolating quadratic has the form $E(\lambda) = a\lambda^2 + b\lambda + c$. Then, λ^* is given by $-b/2a$. The following 3 equations hold.

$$e_1 = a\lambda_1^2 + b\lambda_1 + c \quad (25)$$

$$e_2 = a\lambda_2^2 + b\lambda_2 + c \quad (26)$$

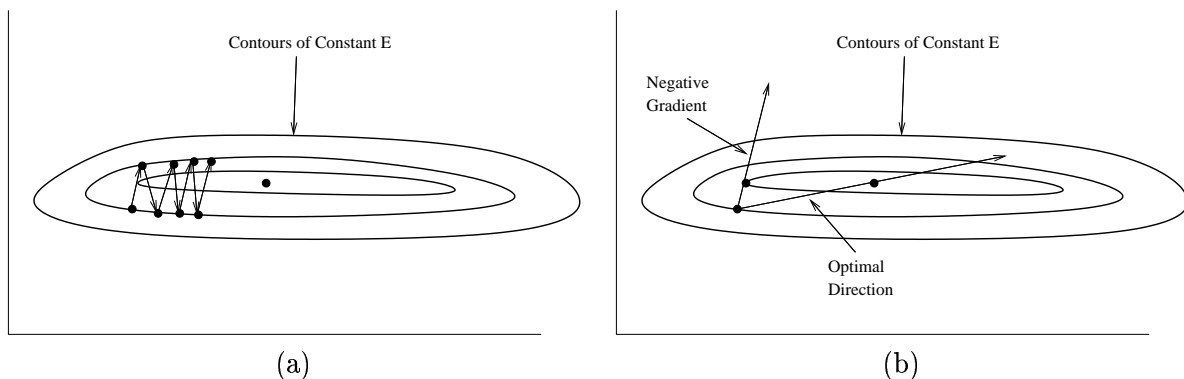
$$e_3 = a\lambda_3^2 + b\lambda_3 + c \quad (27)$$

Solving this set of simultaneous equations for $\{a, b, c\}$ and computing $\lambda^* = -b/2a$, one finds that

$$\lambda^* = \frac{1}{2} \left[\frac{(e_1 - e_2)(\lambda_1^2 - \lambda_3^2) - (e_1 - e_3)(\lambda_1^2 - \lambda_2^2)}{(e_1 - e_2)(\lambda_1 - \lambda_3) - (e_1 - e_3)(\lambda_1 - \lambda_2)} \right] \quad (28)$$

Binary Line Search: Once one has found a U -set, an alternative to the quadratic interpolation that is only slightly inferior, but simpler is to find the minimum by binary search - i.e. step 3 in the linesearch algorithm is replaced by $\lambda^* = \frac{1}{2}(\lambda_1 + \lambda_3)$ instead of the minimum of the model. This method is generally less efficient because it does not use all the information available.

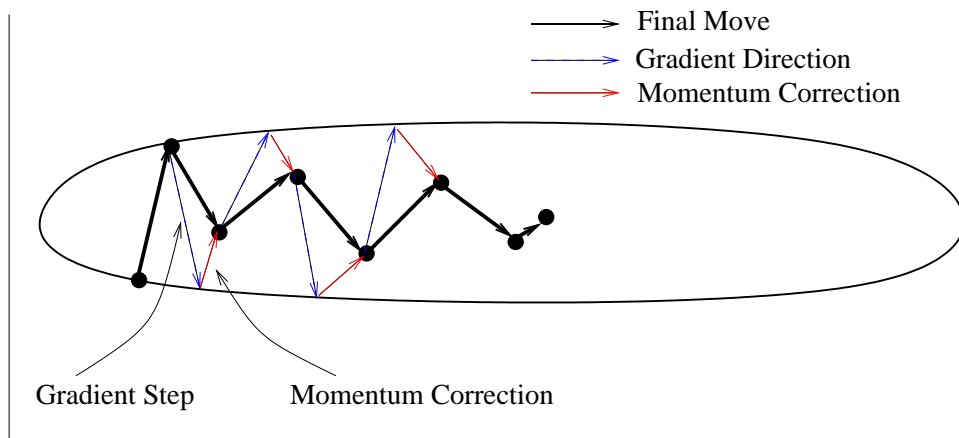
What more can we say about first order methods. It seems like we now have the ultimate first order method. The direction in which to move is the negative gradient, and using our line-search algorithm, we move as much as we can in that direction. The only remaining possible remaining question is whether we should always move in the direction of the negative gradient? It appears that we should if we take a fixed step size, as this is the direction that minimizes the error function. One can consider this as a greedy algorithm, always performing the action that is locally optimal, but this will not necessarily give the optimal algorithm overall. Further, if we consider line-search, the step taken will not necessarily be small, so now the negative gradient direction may not lead to the largest possible decrease. We illustrate some of these issues in the following figures.



With gradient descent, the initial move is always perpendicular to the contour. If the “valley” of the error surface you are in is very narrow (as shown in the figures), then it is possible to hit the other side of the contour and the new gradient doubles back on itself. Thus the approach to the minimum is the extremely inefficient zig-zag path depicted in (a). Figure (b) illustrates how suboptimal the gradient direction can be if one is performing line-search. It is often the case that the error surface has such a valley, hence gradient descent alone will be very inefficient and one needs some way of combatting the zig-zag behavior. The figures should provide ample illustration of this need. The following techniques address this issue.

4.3.4 Momentum

A way to combat the zig-zag behavior is to require that the direction not change too much. This can be enforced by adding to the current negative gradient, a component of the previous move, thus “keeping the previous direction going” unless some force (the negative gradient for the current point) changes it, hence the connection to the physical notion of momentum. This is illustrated in the following figure, where it becomes clear how this can considerably improve upon the zig-zag behavior.



One might remark that suppose that the current gradient turns out to be zero. Then we are at a minimum, yet the momentum term will cause us to move. Of course such a situation RARELY occurs in practice and except in severely badly behaved cases, the situation will rapidly correct itself, so we ignore this worry. Formally speaking, the algorithm is very similar to the previous algorithms, excepting that the direction \mathbf{d}_t is no longer the negative gradient. The algorithm is as follows.

1. Initialize to \mathbf{w}_0 at $t = 0$, and set $\mathbf{d}_{-1} = 0$.
2. Let $\mathbf{d}_t = -\mathbf{g}_t + \mu\mathbf{d}_{t-1}$.
3. Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta\mathbf{d}_t$.
4. Increment t to $t + 1$ and iterate back to (2) until the stopping criterion is reached.

If compared to the gradient descent algorithm with fixed learning rate, the only change is in step (2), where to $-\mathbf{g}_t$, we add a fraction $0 \leq \mu < 1$ of the previous direction, \mathbf{d}_{t-1} to get the new direction \mathbf{d}_t .

There is no reason why such an algorithm should not be used with a variable learning rate. Further one might also use momentum with line-search. However, we will see later that a much superior “momentum-with-line-search-like” algorithm exists so we do not pursue further here.

We can informally understand how momentum is gaining us ground over gradient descent by considering the change total change in the weight vector after many iterations (say T of them).

$$\Delta \mathbf{w} = \mathbf{w}_T - \mathbf{w}_0 = \sum_{t=0}^T \Delta \mathbf{w}_t = \sum_{t=0}^T \eta \mathbf{d}_t \quad (29)$$

$$= -\eta \sum_{t=0}^T \mathbf{g}_t + \mu \eta \sum_{t=0}^T \mathbf{d}_{t-1} \quad (30)$$

$$= -\eta \sum_{t=0}^T \mathbf{g}_t + \mu \Delta \mathbf{w} - \mu \eta \mathbf{d}_T \quad (31)$$

Now, since T is large, we are almost at to the minimum because so the last term can be ignored because μ , η , and especially \mathbf{d}_T are small. Hence, solving for $\Delta \mathbf{w}$ we find that

$$\Delta \mathbf{w} = \frac{-\eta}{1 - \mu} \sum_{t=0}^T \mathbf{g}_t \quad (32)$$

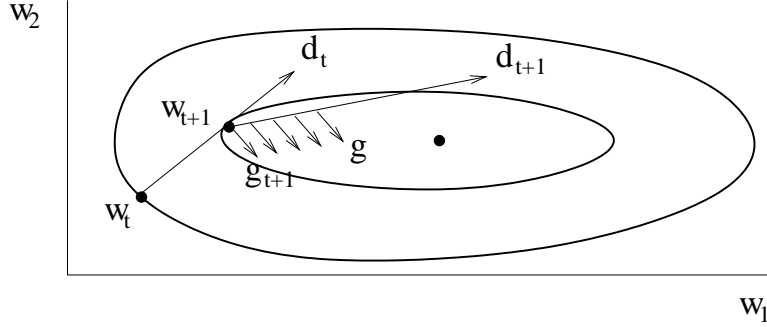
We can interpret this equation as follows. Suppose we made T iterations and at each point the gradient happened to be \mathbf{g}_t . Then plain old fashioned gradient descent would give a change in the weight vector $\Delta \mathbf{w} = -\eta \sum_{t=0}^T \mathbf{g}_t$, which is less than what gradient descent with momentum gives by a factor of $1/(1 - \mu)$. Thus, we can view the momentum term as effectively increasing the learning rate by a factor of $1/(1 - \mu)$. This does not mean that gradient descent with this increased learning rate would give the same result. The momentum term is a little cleverer than that. It also does not mean that gradient descent alone will yield a change in weight vector that is a factor of $1/(1 - \mu)$ less than the change with momentum. This would be a contradiction as both methods starting at the same point should end up at the same minimum, hence, have the same change in the weight vector. The two methods will take much different paths and will encounter different gradients along the way. Momentum will just tend to get there more quickly. Another way to understand how momentum is gaining us is to look at the contribution to $\Delta \mathbf{w}$ due to \mathbf{g}_0 . For simple gradient descent, it is just $-\eta \mathbf{g}_0$. With momentum, the contribution is $-\eta(1 + \mu + \mu^2 + \dots)\mathbf{g}_0$ which is approximately $-\eta \mathbf{g}_0 / (1 - \mu)$, a contribution greater by a factor of $1/(1 - \mu)$.

The natural questions to ask are: How can one pick the momentum parameter, μ ? Could one use a variable momentum parameter, as we did with variable learning rate? What about incorporating line-search - in which case there is no learning rate? These questions are answered in a principled way by the next technique. It is really the king of first order methods and even competes with second order methods.

4.3.5 Conjugate Gradients

To understand the general idea, we consider a necessary condition for w to be a local minimum, namely that $\nabla E(\mathbf{w}) = 0$. Let $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_d$ be any basis for our d -dimensional space. In component form, this necessary condition reads $\nabla E(\mathbf{w}) \cdot \mathbf{u}_i = 0$ for every $i = 1, \dots, d$. Thus, one approach to getting the gradient equal to zero is to get $\nabla E(\mathbf{w}) \cdot \mathbf{u}_1 = 0$, for some \mathbf{u}_1 . Now, keeping $\nabla E(\mathbf{w}) \cdot \mathbf{u}_1 = 0$, get $\nabla E(\mathbf{w}) \cdot \mathbf{u}_2 = 0$ for some \mathbf{u}_2 that is linearly independent of \mathbf{u}_1 . Now, keeping both $\nabla E(\mathbf{w}) \cdot \mathbf{u}_1 = 0$ and $\nabla E(\mathbf{w}) \cdot \mathbf{u}_2 = 0$, get $\nabla E(\mathbf{w}) \cdot \mathbf{u}_3 = 0$ for some \mathbf{u}_3 that is independent of both \mathbf{u}_1 and \mathbf{u}_2 . Continue this process until one has $\nabla E(\mathbf{w}) \cdot \mathbf{u}_i = 0$ for a basis $\{\mathbf{u}_i\}$. Of course this is easier said than done, but we have already accomplished the first step.

Namely, suppose we are at \mathbf{w}_t , and we perform a line-search in the direction \mathbf{d}_t . After this line minimization, we are at \mathbf{w}_{t+1} and we know that $\mathbf{d}_t \cdot \mathbf{g}_{t+1} = 0$, thus we have accomplished our first step with $\mathbf{u}_1 = \mathbf{d}_t$. We now determine the next direction to move in, \mathbf{d}_{t+1} and set about getting the component of the gradient along \mathbf{d}_{t+1} equal to zero. So far, everything is going as planned, except that in performing the line minimization along \mathbf{d}_{t+1} we may mess up the condition $\mathbf{d}_t \cdot \mathbf{g} = 0$. In general this will be so if we choose \mathbf{d}_{t+1} arbitrarily. Thus, the question becomes: Is there a way to choose \mathbf{d}_{t+1} so that in moving along \mathbf{d}_{t+1} to get $\mathbf{d}_{t+1} \cdot \mathbf{g} = 0$ we do not disturb the fact that $\mathbf{d}_t \cdot \mathbf{g} = 0$? The situation is illustrated in the following figure.



We would like the gradient along the new direction of motion to be orthogonal to \mathbf{d}_t . This is exactly what the technique of conjugate gradients attempts to do as follows.

At time t , we were at \mathbf{w}_t , and by minimizing along \mathbf{d}_t we have obtained \mathbf{w}_{t+1} with $\mathbf{d}_t \cdot \mathbf{g}_{t+1} = 0$. We pick \mathbf{d}_{t+1} with the requirement that $\mathbf{d}_t \cdot \mathbf{g}(w_{t+1} + \lambda \mathbf{d}_{t+1})$ remains zero after performing the line search along \mathbf{d}_{t+1} . Using a Taylor approximation,

$$\mathbf{g}(w_{t+1} + \lambda \mathbf{d}_{t+1}) \approx \mathbf{g}_{t+1} + \lambda \mathbf{H}_{t+1} \mathbf{d}_{t+1} \quad (33)$$

Requiring that $\mathbf{d}_t \cdot \mathbf{g}(w_{t+1} + \lambda \mathbf{d}_{t+1}) = 0$ and using the fact that $\mathbf{d}_t \cdot \mathbf{g}_{t+1} = 0$ yields the following necessary condition

$$\mathbf{d}_{t+1}^T \mathbf{H}_{t+1} \mathbf{d}_t = 0 \quad (34)$$

Vectors satisfying such a condition are termed conjugate with respect to \mathbf{H}_{t+1} , hence the name for the method. We now suppose that \mathbf{d}_{t+1} is of the form

$$\mathbf{d}_{t+1} = -g_{t+1} + \beta_{t+1} \mathbf{d}_t \quad (35)$$

In other words, we will be doing line-search with a momentum term (as was promised in the previous section) with a momentum parameter β_t that is yet to be determined. Plugging (35) into (34), we see that

$$0 = -g_{t+1}^T \mathbf{H}_{t+1} \mathbf{d}_t + \beta_{t+1} \mathbf{d}_t^T \mathbf{H}_{t+1} \mathbf{d}_t \quad (36)$$

which upon solving for β_t gives

$$\beta_{t+1} = \frac{g_{t+1}^T \mathbf{H}_{t+1} \mathbf{d}_t}{\mathbf{d}_t^T \mathbf{H}_{t+1} \mathbf{d}_t} \quad (37)$$

One might ask, what if $\mathbf{H}_{t+1} \mathbf{d}_t = 0$ so the division above is not valid. We will assume that \mathbf{H}_{t+1} has full rank so this means that $\mathbf{d}_t = 0$ and if this were to happen, we will show later that this means that $\mathbf{g}_t = 0$ in which case \mathbf{w}_t is a local minimum and though the division is not valid, we don't care, as we would have been done at \mathbf{w}_t . In principle we are done, as we have an expression for β_t , thus we can compute \mathbf{d}_{t+1} . However, we do not know \mathbf{H}_{t+1} . Why? Because we have assumed that only first order information is available. We could try to build up the second order information that we need by perturbing \mathbf{w}_{t+1} and obtaining \mathbf{H}_{t+1} from the observed changes in the gradient, but that would be cheating, and we would then be really using a second order method. Closer inspection shows that we do not really need to know \mathbf{H}_{t+1} , all we need to know is $\mathbf{H}_{t+1} \mathbf{d}_t$. We know that $\lambda_t \mathbf{d}_t = \mathbf{w}_{t+1} - \mathbf{w}_t$. And we have already used the approximation $\mathbf{g}_{t+1} - \mathbf{g}_t = \mathbf{H}_{t+1}(\mathbf{w}_{t+1} - \mathbf{w}_t)$, so let's use it again to get $\lambda_t \mathbf{H}_{t+1} \mathbf{d}_t = \mathbf{g}_{t+1} - \mathbf{g}_t$ which we can substitute into (37) to get

$$\beta_{t+1} = \frac{g_{t+1}^T (\mathbf{g}_{t+1} - \mathbf{g}_t)}{\mathbf{d}_t^T (\mathbf{g}_{t+1} - \mathbf{g}_t)} \quad (38)$$

All the quantities are now known so β_t can be computed. Using the fact that $\mathbf{d}_t^T \mathbf{g}_{t+1} = 0$, that $\mathbf{d}_t = -\mathbf{g}_t + \beta_t \mathbf{d}_{t-1}$ and that $\mathbf{d}_{t-1}^T \mathbf{g}_t = 0$, we get the following simplified expression for β_t

$$\beta_{t+1} = \frac{\mathbf{g}_{t+1}^T (\mathbf{g}_{t+1} - \mathbf{g}_t)}{\mathbf{g}_t^T \mathbf{g}_t} \quad (39)$$

We are now ready to state the conjugate gradient algorithm.

1. Initialize to \mathbf{w}_0 at $t = 0$. Set $\beta_0 = 0$ and $d_{-1} = 0$.
2. Let $\mathbf{d}_t = -\mathbf{g}_t + \beta_t \mathbf{d}_{t-1}$, where β_t is given by (39) for $t > 0$.
3. Minimize E in the direction \mathbf{d}_t to obtain λ^* such that $E(\mathbf{w}_t + \lambda^* \mathbf{d}_t)$ is a local minimum with respect to the single parameter λ .
4. Update the weight $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \lambda^* \mathbf{d}_t$.
5. Increment t to $t + 1$ and iterate back to (2) until the stopping criterion is reached.

The only difference between this algorithm and steepest descent is in step (2) where the direction of minimization has a momentum term with momentum parameter selected in a principled way.

We began with the lofty goal of successively zeroing out the gradient in d independent directions. We now see that our first two steps are successful, but when we perform the third step, to get to \mathbf{w}_3 , we know that \mathbf{g}_3 will be perpendicular to \mathbf{d}_2 and \mathbf{d}_1 , but what about d_0 ? This is the beauty of the algorithm. The perpendicularity of \mathbf{g}_t for all $\mathbf{d}_{t'}$ for $t' < t$ is guaranteed to be so for the case that the error surface is quadratic, and will thus be approximately so as we approach the minimum, where the error surface begins to look more and more quadratic. Further, one can show that the vectors $\mathbf{d}_0, \mathbf{d}_1, \dots, \mathbf{d}_{t-1}$ are all independent. This means that \mathbf{g}_t will be perpendicular to d independent vectors, and hence $g_d = 0$ (if it hadn't already hit 0 at a previous stage). This is a remarkable claim, namely that for a quadratic surface (and hence our surface near the minimum) we will reach the minimum in a finite number of steps, namely at most d steps!

In practice however, since the surface is not exactly quadratic, the \mathbf{d}_t 's are only "approximately" independent, so \mathbf{d}_d will be approximately independent of d vectors. This means that \mathbf{d}_d will be approximately zero, and in making it independent of the other previous d directions, we may have ended up with a direction that is not a good descent direction. For this reason, it is advisable to reset \mathbf{d} to minus the gradient (i.e. set $\beta_d = 0$) about every d or perhaps every $d/2$ iterations. The claim that the convergence for a quadratic surface occurs in a finite number of iterations, which suggests similar convergence for the general error surface is quite striking and hence deserves a proof. This proof will conclude our discussion of first order methods.

Advanced Topic: Convergence of Conjugate Gradient for Quadratic Surfaces.

The error surface being quadratic means that we can represent it as

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*) \quad (40)$$

for some constant matrix \mathbf{H} , where \mathbf{w}^* is the minimum. Once again, the motivation for studying quadratic error surfaces is that the general error surfaces look quadratic near a minimum. The proof of convergence in at most d steps proceeds in two stages.

First we show that if at any time $\mathbf{d}_t = 0$, then $\mathbf{g}_t = 0$ and so we are done. This is true because

$$\mathbf{g}_t \cdot \mathbf{d}_t = 0 = -\mathbf{g}_t \cdot \mathbf{g}_t + \beta_t \mathbf{g}_t \cdot \mathbf{d}_{t-1} = -\|\mathbf{g}_t\|^2 \quad (41)$$

So we continue iterating till we hit $\mathbf{d}_t = 0$, and ask what the maximum number of iterations we have to do is? The following theorem helps us answer this question

Theorem 4.1 *Suppose that we have performed T conjugate gradient iterations and have not hit $\mathbf{d}_t = 0$ for any $0 \leq t \leq T$, then the following conditions hold for all $i < j \leq T$*

- i. $\mathbf{d}_i^T \mathbf{H} \mathbf{d}_j = 0$.
- ii. $\mathbf{d}_i^T \mathbf{g}_j = 0$.
- iii. $\mathbf{g}_i^T \mathbf{g}_j = 0$.

In other words, the \mathbf{d}_i 's are all conjugate to each other, the \mathbf{g}_i 's are all orthogonal to each other and the \mathbf{g}_j 's are orthogonal to all the previous \mathbf{d}_i 's. By construction of the conjugate gradient algorithm, we know that $\mathbf{d}_i \mathbf{H} \mathbf{d}_{i+1} = 0$ and that $\mathbf{d}_i^T \mathbf{g}_{i+1} = 0$, but the amazing thing is that this is true for all $i < j \leq T$.

Lets see how this theorem helps us answer our question regarding the maximum number of iterations that one must perform. Suppose that we have performed $T = d$ iterations and have not yet come to the minimum. We can derive a contradiction in many ways:

1. $\{\mathbf{g}_i\}_{i=0}^d$ represent a set of mutually orthogonal vectors in a d -dimensional space so at least one of them must be 0. ■
2. $\{\mathbf{d}_i\}_{i=0}^d$ represents a set of independent vectors in d dimensions unless one of them is zero. This is true because for any set of α_i , with not all $\alpha_i = 0$ we have from (i) that

$$\left(\sum_{i=0}^d \alpha_i \mathbf{d}_i \right) \mathbf{H} \left(\sum_{j=0}^d \alpha_j \mathbf{d}_j \right) = \sum_{i=0}^d \alpha_i^2 \mathbf{d}_i \mathbf{H} \mathbf{d}_i \geq 0 \quad (42)$$

because \mathbf{H} is positive definite and some α is non-zero, hence $\sum_{i=0}^d \alpha_i \mathbf{d}_i$ cannot be zero, unless all α_i are zero proving the independence of the \mathbf{d}_i . But it is not possible to have more than d independent vectors in d -dimensions hence one of the \mathbf{d}_i must be zero. ■

3. Suppose that $\mathbf{d}_0, \dots, \mathbf{d}_{d-1}$ are not zero. Then since they are independent, \mathbf{g}_d is orthogonal to d independent vectors by (2), hence $\mathbf{g}_d = 0$. ■

Take your pick. No matter which way you go, the conclusion is that by at most d iterations, one has come to the minimum. We now give a proof of the theorem.

PROOF OF THEOREM 4.1: We first show that (ii) \Rightarrow (iii). If $i = 0$, $\mathbf{g}_i^T \mathbf{g}_j = -d_i^T \mathbf{g}_j = 0$ by (ii). If $i > 0$, since $\mathbf{g}_i = \beta_i \mathbf{d}_{i-1} - \mathbf{d}_i$,

$$\mathbf{g}_i^T \mathbf{g}_j = \beta_i \mathbf{d}_{i-1}^T \mathbf{g}_j - \mathbf{d}_i^T \mathbf{g}_j = 0 \quad (43)$$

by (ii). Our proof of (i) and (ii) will be by induction. Let the induction statement for $K \leq T$ be

P_K : $\mathbf{d}_i^T \mathbf{H} \mathbf{d}_j = 0$ and $d_i^T \mathbf{g}_j = 0$ for $i < j \leq K$.

P_1 is true as $\mathbf{d}_0^T \mathbf{H} \mathbf{d}_1 = 0$ and $\mathbf{d}_0^T \mathbf{g}_1 = 0$ by construction. Suppose that P_K is true and consider P_{K+1} . We need to show that for $i < K + 1$, $\mathbf{d}_i^T \mathbf{H} \mathbf{d}_{K+1} = 0$ and $d_i^T \mathbf{g}_{K+1} = 0$. For $i = K$ these statements are true by construction so we need only consider the case $i < K$. Since $H(\mathbf{w}_{i+1} - \mathbf{w}_i) = \mathbf{g}_{i+1} - \mathbf{g}_i$ and $\mathbf{d}_i = (\mathbf{w}_{i+1} - \mathbf{w}_i)/\lambda_i$ for some constant λ_i we see that $\mathbf{H} \mathbf{d}_i = (\mathbf{g}_{i+1} - \mathbf{g}_i)/\lambda_i$. Thus, for $i < K$

$$\mathbf{d}_i^T \mathbf{g}_{K+1} = \mathbf{d}_i^T (\mathbf{g}_{K+1} - \mathbf{g}_K) + \mathbf{d}_i^T \mathbf{g}_K = \lambda_K \mathbf{d}_i^T \mathbf{H} \mathbf{d}_K + \mathbf{d}_i^T \mathbf{g}_K = 0 \quad (44)$$

where the last equality follows by the induction hypothesis. And since (ii) \Rightarrow (iii), we have that for $i < j \leq K + 1$ $\mathbf{g}_i^T \mathbf{g}_j = 0$. Now consider $\mathbf{d}_i^T \mathbf{H} \mathbf{d}_{K+1}$ for $i < K$.

$$\mathbf{d}_i^T \mathbf{H} \mathbf{d}_{K+1} = \mathbf{d}_i^T \mathbf{H} (-\mathbf{g}_{K+1} + \beta_{K+1} \mathbf{d}_K) \quad (45)$$

$$= -\frac{1}{\lambda_i} (\mathbf{g}_{i+1} - \mathbf{g}_i)^T \mathbf{g}_{K+1} + 0 \quad (46)$$

$$= 0 \quad (47)$$

So, P_{K+1} is true. Thus, by induction, P_K is valid for all $1 \leq K \leq T$ and the theorem is proved. \blacksquare

We now move on to second order methods.

4.4 Second Order Methods

At time t we are at \mathbf{w}_t . We assume that we have access to the error function value, $E(\mathbf{w}_t)$, its gradient $\nabla E(\mathbf{w}_t)$ and the Hessian $\mathbf{H}_{ij} = \frac{\partial^2 E(\mathbf{w}_t)}{\partial w_i \partial w_j}$, which we will denote by \mathbf{H}_t . Then we

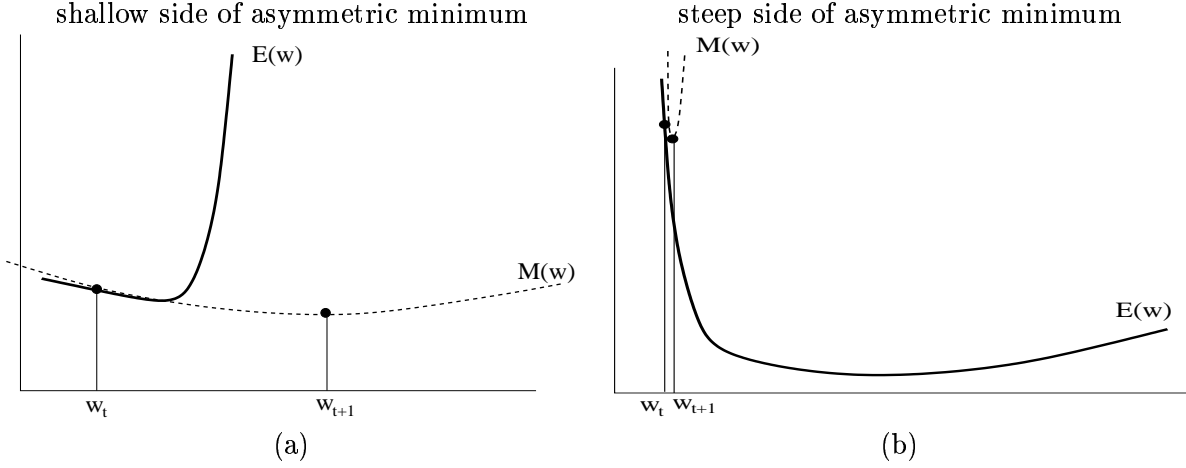
$$E(\mathbf{w}) \approx E(\mathbf{w}_t) + \mathbf{g}_t^T (\mathbf{w} - \mathbf{w}_t) + \frac{1}{2} (\mathbf{w} - \mathbf{w}_t)^T \mathbf{H}_t (\mathbf{w} - \mathbf{w}_t) \quad (48)$$

Taking the gradient with respect to \mathbf{w} and setting it to zero to find a \mathbf{w}_{t+1} that the model suggests as a minimum, we find that

$$\mathbf{g}_t + \mathbf{H}_t (\mathbf{w}_{t+1} - \mathbf{w}_t) = 0 \quad \Rightarrow \quad \mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}_t^{-1} \mathbf{g}_t \quad (49)$$

Thus, the step taken is given by $\Delta \mathbf{w}_t = -\mathbf{H}_t^{-1} \mathbf{g}_t$ which is sometimes called the Newton step. This could be a big step, and in some sense an optimal step, given our model. The convergence analysis for quadratic error surfaces is extremely easy, since for quadratic error surfaces (48) is exact, the algorithm converges in one step to the minimum. This will also be true once one is close to the minimum, hence the Newton step is extremely fast once one has come close to the minimum (in terms of number of iterations to convergence). The trade off is, of course, that each iteration is costly. One needs to compute the hessian and its inverse. This computation of the inverse is a very costly ($\Theta(d^3)$) operation.

Further, the Newton step can run into some severe problems as are illustrated in the following figures.



In (a) the step taken is too large and takes you out of the domain of validity. In (b), the quadratic model can fit the steep side and the step is very small, and one would like to take a much larger step, as would be taken by say gradient descent, as the gradient is quite large. It is clear that if we didn't take a step, but rather did a line search in the direction $\mathbf{d}_t^N = -\mathbf{H}_t^{-1}\mathbf{g}_t$ (N for Newton), then the convergence property for the quadratic surface would still be valid, and we can avoid the problems illustrated in the figure. A subtle problem may arise though. The direction \mathbf{d}_t^N may not be a descent direction. From (19), we see that that \mathbf{d}_t^N is a descent direction if and only if $\mathbf{g}_t \cdot \mathbf{d}_t^N < 0$ in which case we need

$$\mathbf{g}_t^T \mathbf{H}_t^{-1} \mathbf{g}_t > 0 \quad (50)$$

Thus, it is sufficient that \mathbf{H} be a positive definite matrix. If we are close to the minimum, \mathbf{H} will be positive definite, but in general, away from the minimum, \mathbf{H} may not be positive definite and thus \mathbf{d}_t^N may not generally represent a descent direction. An approach that is sometimes taken is to add an amount $\lambda \mathbf{I}$ to \mathbf{H}_t to make it positive definite so as to guarantee a descent direction, but one starts to get lost in the heuristics, and having spent quite some effort in getting the Hessian, one would like a more principled approach. Suppose we take a fixed step in some direction, we ask what the optimal direction is by minimizing

$$E(\mathbf{w}_t + \delta \mathbf{w}) = E(\mathbf{w}_t) + \mathbf{g}_t^T \Delta \mathbf{w} + \frac{1}{2} \Delta \mathbf{w}^T \mathbf{H}_t \Delta \mathbf{w} \quad (51)$$

with respect to $\Delta \mathbf{w}$ subject to the constraint that the step size is η , ie that $\Delta \mathbf{w}^T \Delta \mathbf{w} = \eta^2$. Setting up the Lagrangian,

$$\mathcal{L} = E(\mathbf{w}_t) + \mathbf{g}_t^T \Delta \mathbf{w} + \frac{1}{2} \Delta \mathbf{w}^T (\mathbf{H}_t + 2\alpha \mathbf{I}) \Delta \mathbf{w} \quad (52)$$

where α is the lagrange multiplier, we find that

$$\Delta \mathbf{w} = -(\mathbf{H}_t + 2\alpha \mathbf{I})^{-1} \mathbf{g}_t, \quad \Delta \mathbf{w}^T \Delta \mathbf{w} = \eta^2 \quad (53)$$

which one can solve for α to find that

$$\alpha = -\frac{1}{2\eta^2} (\Delta \mathbf{w}^T \mathbf{g}_t + \Delta \mathbf{w}^T \mathbf{H}_t \Delta \mathbf{w}) \quad (54)$$

The second term inside the brackets is $\theta(\eta^2)$ but the first is $\sim \eta \|\mathbf{g}_t\|$. Thus we see that $\alpha \sim \|\mathbf{g}_t\|/2\eta$ which is large for η chosen reasonably small (small step size). More formally, (54) implicitly defines

α and since α is large, we can obtain the expansion of the right hand side in terms of $1/\alpha$, which we can solve recursively for α . To leading order, this expansion gives $\alpha = \|\mathbf{g}_t\|/2\eta$, which we substitute into (53) to get the final weight update

$$\Delta \mathbf{w} = - \left(\mathbf{H}_t + \frac{\|\mathbf{g}_t\|}{\eta} \mathbf{I} \right)^{-1} \mathbf{g}_t \quad (55)$$

Let's look at this weight update step more carefully. Suppose that we are far from the minimum. Then $\|\mathbf{g}_t\|/\eta$ is going to be large as η is chosen small and $\|\mathbf{g}_t\|$ is large, hence the dominant term is the second and, thus, far from the minimum, we get that

$$\Delta \mathbf{w} \approx -\eta \frac{\mathbf{g}_t}{\|\mathbf{g}_t\|} \quad (56)$$

which is exactly normalized gradient descent with learning rate η . When we are close to the minimum, the gradient approaches zero, hence, the dominant term is the first term and, thus, as we approach the minimum, we get that

$$\Delta \mathbf{w} \approx -\mathbf{H}^{-1} \mathbf{g}_t \quad (57)$$

which is the Newton step. Thus we have devised an algorithm that has the characteristics of gradient descent far from the minimum, however, close to the minimum the Newton step kicks in. Intuitively, this is exactly what we want. Furthermore, the algorithm automatically selects between the two. Analogous to the difference between normalized and un-normalized gradient descent, it is common to have $\eta \propto \|\mathbf{g}_t\|$ in which case the weight update becomes

$$\Delta \mathbf{w} = - \left(\mathbf{H}_t + \frac{1}{\eta} \mathbf{I} \right)^{-1} \mathbf{g}_t \quad (58)$$

and now analogous to variable learning rate gradient descent one can allow η to vary. This leads us to the celebrated **Levenberg-Marquardt** algorithm:

1. Initialize to \mathbf{w}_0 at $t = 0$, and set η_0 to an initial value.
2. Let $\mathbf{x}_{t+1} \leftarrow \mathbf{w}_t - \left(\mathbf{H}_t + \frac{1}{\eta_t} \mathbf{I} \right)^{-1} \mathbf{g}_t$.
 - i. If $E(\mathbf{x}_{t+1}) < E(\mathbf{w}_t)$, accept the new weight vector, by setting $\mathbf{w}_{t+1} = \mathbf{x}_{t+1}$ and increment η to $\eta_{t+1} = \alpha \eta_t$ where $\alpha > 1$.
 - ii. If $E(\mathbf{x}_{t+1}) \geq E(\mathbf{w}_t)$, reject the new weight vector, by setting $\mathbf{w}_{t+1} = \mathbf{w}_t$ and decrease η to $\eta_{t+1} = \beta \eta_t$ where $\beta < 1$.
3. Increment t to $t + 1$ and iterate back to (2) until the stopping criterion is reached.

In practice one finds that one can be very aggressive with the parameters α and β . $\alpha = \beta = 10$ often works well in practice. Factoring in the inefficiency of having to obtain the Hessian, the Levenberg Marquardt algorithm still probably ranks as the algorithm of choice.

4.4.1 Approximations to the Hessian

Since the Hessian is expensive to compute, we consider some approximations to the hessian.

Sum of Outer Products Approximation Suppose that

$$E(\mathbf{w}) = \sum_i e_i(\mathbf{x}_i; \mathbf{w})^2 \quad (59)$$

as is the case when $E(\mathbf{w}) = R_{emp}(\mathbf{w})$, with e_i being the error on data point i . One then computes that

$$\frac{\partial^2 E(\mathbf{w})}{\partial w_\alpha \partial w_\beta} = 2 \sum_i \frac{\partial e_i}{\partial w_\alpha} \frac{\partial e_i}{\partial w_\beta} + 2 \sum_i e_i \frac{\partial^2 e_i}{\partial w_\alpha \partial w_\beta} = 2 \sum_i \mathbf{g}_i \mathbf{g}_i^T + 2 \sum_i e_i \frac{\partial^2 e_i}{\partial w_\alpha \partial w_\beta} \quad (60)$$

where $\mathbf{g}_i = \nabla_{\mathbf{w}}(\mathbf{x}; \mathbf{w})$. The sum of outer products approximation ignores the second term which is usually the hard term to compute. In this event one is using only first order information.

Building up an Approximation to the Hessian Iteratively Although we do not go through the details here, it is possible to build up iterative approximations to the Hessian (for example BFGS). Using these approximations and linesearch in the Newton direction, one can show that convergence to the local minimum occurs in at most d steps, much like conjugate gradient methods. This concludes our discussion of optimization methods.