# Chapter 2

# Fundamentals of numerical optimization
[1]

In this Chapter we review fundamental concepts from the field of numerical optimization that will be used throughout the text in order to determine optimal parameters for learning models (as described in Section 1.4) via the minimization of a differentiable function. Specifically, we describe two basic but widely used algorithms, known as *gradient descent* and *Newton's method*, beginning with a review of several important ideas from calculus that provide the mathematical foundation for both methods.

## 2.1 Calculus defined optimality

In this Section we briefly review how calculus is used to describe the local geometry of a function, as well as its minima or lowest points. As we will see later in the Chapter powerful numerical algorithms can be built using these simple concepts.

### 2.1.1 Taylor series approximations

To glean some basic insight regarding the geometry of a many times differentiable function $g(w)$ near a point $v$ we may form a *linear* approximation to the function near this point. This is just a tangent line passing through the point $(v, g(v))$, as illustrated pictorially in Figure 2.1, which contains the first derivative information $g'(v)$.

[1] This document is part of a book currently under development titled "Machine Learning Refined" (Cambridge University Press, late 2016) by Jeremy Watt, Reza Borhani, and Aggelos Katsaggelos. Please do not distribute. Feedback regarding any errors, comments on substance and style, recommendations, etc. is greatly appreciated! Contact: jermwatt@gmail.edu

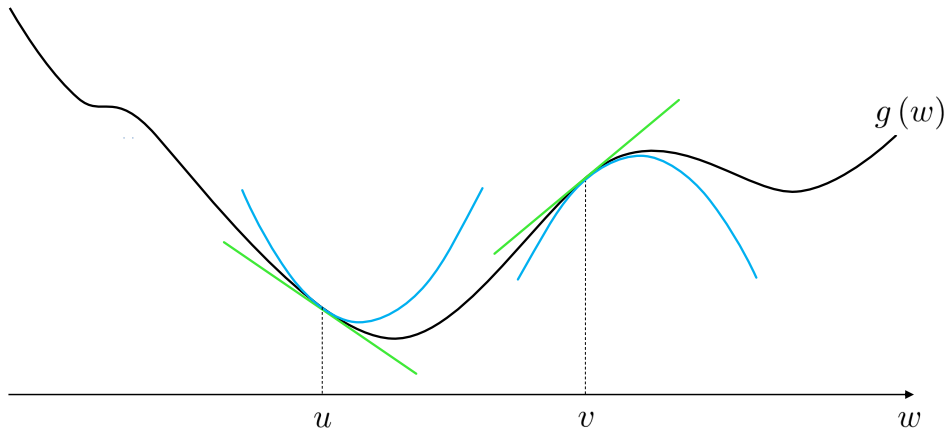


**Figure 2.1.** *Figurative illustration of linear (in green) and quadratic (in blue) approximations to a differentiable function $g(w)$ at two points: $w = u$ and $w = v$. Often these linear and quadratic approximations are equivalently referred to as first and second order Taylor series approximations, respectively.*

Such a linear approximation (also known as a first order Taylor series) is written as

$$h(w) = g(v) + g'(v)(w - v). \tag{2.1}$$

Notice that indeed this function is **a)** linear in $w$, **b)** tangent at to $g(w)$ at $v$ since $h(v) = g(v)$ **and because** contains the first derivative information of $g$ at $v$ i.e., $h'(v) = g'(v)$. This linear approximation holds particularly well near $v$ because the derivative contains slope information.

To understand even more about $g$ near $v$ we may form a quadratic approximation (also illustrated in Figure 2.1) that contains both first and second derivative information $g'(v)$ and $g''(v)$. This quadratic, referred to as the second order Taylor series approximation, is written as

$$h(w) = g(v) + g'(v)(w - v) + \frac{1}{2}g''(v)(w - v)^2. \tag{2.2}$$

This quadratic contains the same tangency and first order information of the linear approximation (i.e., $h(v) = g(v)$ and $h'(v) = g'(v)$) with additional second order derivative information as well at $g$ near $v$ since $h''(v) = g''(v)$. The second order Taylor series approximation more closely resembles the underlying function around $v$ because the second derivative contains so-called curvature information.

We may likewise define linear and quadratic approximations for a many times differentiable function $g(\mathbf{w})$ of vector valued input $\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \cdots & w_N \end{bmatrix}^T$. In general we may formally write the linear approximation as

$$h(\mathbf{w}) = g(\mathbf{v}) + \nabla g(\mathbf{v})^T (\mathbf{w} - \mathbf{v}) \tag{2.3}$$

where $\nabla g(\mathbf{v}) = \left[ \begin{array}{cccc} \frac{\partial}{\partial w_1} g(\mathbf{v}) & \frac{\partial}{\partial w_2} g(\mathbf{v}) & \cdots & \frac{\partial}{\partial w_N} g(\mathbf{v}) \end{array} \right]^T$ is the $N \times 1$ gradient of partial derivatives (which reduces to $g'(v) = \frac{\partial}{\partial w} g(v)$ in the case $N = 1$). We may also generally write the quadratic approximation as

$$h(\mathbf{w}) = g(\mathbf{v}) + \nabla g(\mathbf{v})^T (\mathbf{w} - \mathbf{v}) + \frac{1}{2} (\mathbf{w} - \mathbf{v})^T \nabla^2 g(\mathbf{v}) (\mathbf{w} - \mathbf{v}). \tag{2.4}$$

where $\nabla^2 g(\mathbf{v})$ is the $N \times N$ symmetric Hessian matrix of second derivatives (which is just the second derivative $g''(v) = \frac{\partial^2}{\partial w^2} g(v)$ when $N = 1$) defined as

$$\nabla^2 g(\mathbf{v}) = \left[ \begin{array}{cccc} \frac{\partial^2}{\partial w_1 \partial w_1} g(\mathbf{v}) & \frac{\partial^2}{\partial w_1 \partial w_2} g(\mathbf{v}) & \cdots & \frac{\partial^2}{\partial w_1 \partial w_N} g(\mathbf{v}) \\ \frac{\partial^2}{\partial w_2 \partial w_1} g(\mathbf{v}) & \frac{\partial^2}{\partial w_2 \partial w_2} g(\mathbf{v}) & \cdots & \frac{\partial^2}{\partial w_2 \partial w_N} g(\mathbf{v}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial w_N \partial w_1} g(\mathbf{v}) & \frac{\partial^2}{\partial w_N \partial w_2} g(\mathbf{v}) & \cdots & \frac{\partial^2}{\partial w_N \partial w_N} g(\mathbf{v}) \end{array} \right]. \tag{2.5}$$

### 2.1.2 The first order condition for optimality

Minimum values of a function $g$ are naturally located at 'valley floors' where the line or hyperplane tangent to the function has zero slope. Because the derivative/gradient contains this slope information, calculus thereby provides a convenient way of finding minimum values of $g$. In $N = 1$ dimension any point $v$ where $g'(v) = 0$ is a potential minimum. Analogously with general $N$-dimensional input any point $\mathbf{v}$ where $\nabla g(\mathbf{v}) = \mathbf{0}_{N \times 1}$ is a potential minimum as well. Note that the condition $\nabla g(\mathbf{v}) = \mathbf{0}_{N \times 1}$ can be equivalently written as a system of $N$ equations

$$\begin{array}{c} \frac{\partial}{\partial w_1} g = 0, \\ \frac{\partial}{\partial w_2} g = 0, \\ \vdots \\ \frac{\partial}{\partial w_N} g = 0. \end{array} \tag{2.6}$$

However for a general function $g$ minima are not the only points that satisfy this condition. As illustrated in Figure 2.2 a function's maxima as well as saddle points (i.e., points at which the curvature of the function changes from negative to positive or vice-versa) are also points at which the function has a vanishing gradient. Together minima, maxima, and saddle points are referred to as *stationary points* of a function.

In short while calculus provides us with a useful method for determining minima of a general function $g$, this method unfortunately determines other undesirable points (maxima and saddle points) as well[3]. Regardless, as we will see later in this Chapter the condition $\nabla g(\mathbf{v}) =$

[3] Although there is a second order condition for optimality that can be used to distinguish between various types of stationary points, it is not often used in practice since it is much easier to construct optimization schemes based solely on the first order condition stated here.

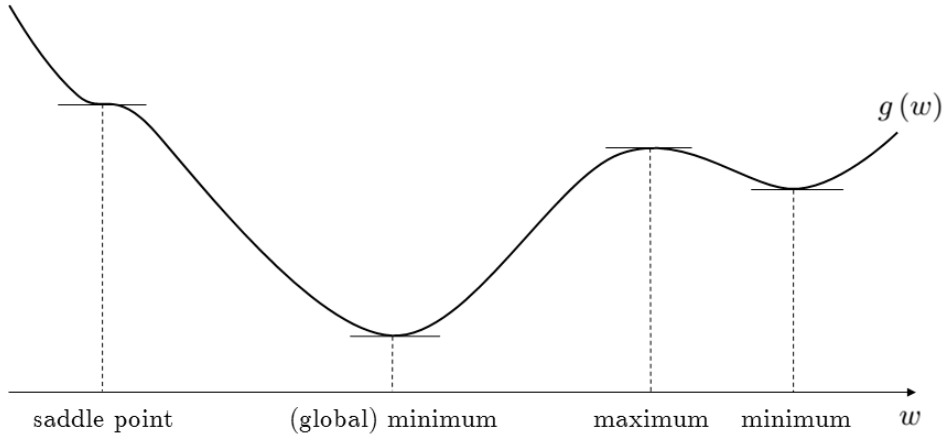


**Figure 2.2.** *Stationary points of the general function g include minima, maxima, and saddle points. At all such points the gradient is zero.*

$\mathbf{0}_{N\times 1}$ is a hugely important tool for determining minima, generally referred to as the *first order condition for optimality*, or in short the first order condition.

> Stationary points of a function $g$ (including minima, maxima, and saddle points) satisfy the first order condition $\nabla g(\mathbf{v}) = \mathbf{0}_{N\times 1}$.

### Example 2.1. Stationary points of simple functions

In this Example we use the first order condition for optimality to compute stationary points of functions $g(w) = w^3$, $g(w) = e^w$, $g(w) = \sin(w)$, $g(w) = w^2$, and $g(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{r}^T\mathbf{w} + d$.

- $g(w) = w^3$, plotted in the first panel of Figure 2.3, the first order condition gives $g'(v) = 3v^2 = 0$ with a saddle point at $v = 0$.
- $g(w) = e^w$, plotted in the second panel of Figure 2.3, the first order condition gives $g'(v) = e^v = 0$ which is only satisfied as $v$ goes to $-\infty$, giving a minimum.
- $g(w) = \sin(w)$, plotted in the third panel of Figure 2.3, the first order condition gives stationary points wherever $g'(v) = \cos(v) = 0$ which occurs at odd integer multiples of $\pi/2$, i.e., maxima at $v = \frac{(4n+1)\pi}{2}$ and minima at $v = \frac{(4n+3)\pi}{2}$ where $n$ is any integer.
- $g(w) = w^2$, plotted in the fourth panel of Figure 2.3, the first order condition gives $g'(v) = 2v = 0$ with a minimum at $v = 0$.

- $g(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{r}^T\mathbf{w} + d$ where $\mathbf{Q}$ is an $N \times N$ symmetric matrix (i.e., $\mathbf{Q} = \mathbf{Q}^T$), $\mathbf{r}$ is an $N \times 1$ vector, and $d$ is a scalar. Then $\nabla g(\mathbf{w}) = \mathbf{Q}\mathbf{w} + \mathbf{r}$ and thus stationary points exist for all solutions to the linear system of equations $\mathbf{Q}\mathbf{w} = -\mathbf{r}$.

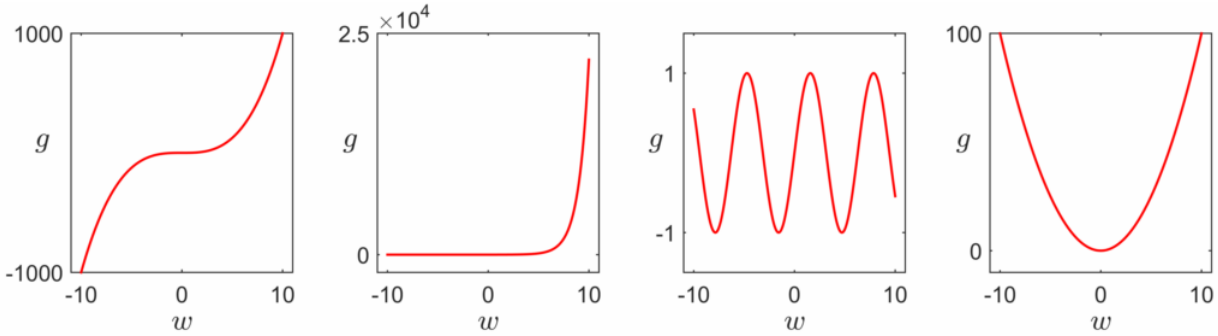


**Figure 2.3.** *From left to right, plots of functions* $g(w) = w^3$, $g(w) = e^w$, $g(w) = sin(w)$, *and* $g(w) = w^2$.

### 2.1.3 The convenience of convexity

As discussed in Section **1.4** solving a machine learning problem eventually reduces to finding the minimum of an associated cost function. Of all (potentially many) minima of a cost function, we are especially interested in the one that provides the lowest possible value of the function, known as the *global minimum*. For a special family of functions, referred to as convex functions, the first order condition is particularly useful because *all stationary points of a convex function are global minima*. In other words, convex functions are free of maxima and saddle points as well as non-global minima.

To determine if a function $g$ is *convex* (facing upward, as at the leftmost point illustrated in Figure 2.1) or *concave* (facing downward, as at the rightmost point in that Figure) at a point $v$ we check its curvature or second derivative information there

$$\begin{aligned} g''(v) \geq 0 \quad &\Longleftrightarrow \quad g \text{ is convex at } v \\ g''(v) \leq 0 \quad &\Longleftrightarrow \quad g \text{ is concave at } v. \end{aligned} \tag{2.7}$$

Here if a statement on one side of the symbol $\Longleftrightarrow$ (which reads 'if and only if') is true then the statement on the other side is true as well (likewise if one is false then the other is false as well). Similarly for general $N$ an analogous statement can be made regarding the eigenvalues of $\nabla^2 g(\mathbf{v})$, i.e., $g$ is convex (or concave) at $\mathbf{v}$ if and only if the Hessian matrix evaluated at this point has all non-negative (or non-positive) *eigenvalues*, in which case the Hessian is called positive semi-definite (or negative semi-definite).

Based on this rule, $g(w)$ is convex everywhere, a convex function, if its second derivative $g''(w)$ is always non-negative. Likewise $g(\mathbf{w})$ is convex if $\nabla^2 g(\mathbf{w})$ always has non-negative eigenvalues. This is generally referred to as the *second order definition of convexity*[6].

A function is convex if and only if $g''(w) \geq 0$ for all $w$ (or $\nabla^2 g(\mathbf{w})$ has non-negative eigenvalues for all $\mathbf{w}$)

**Example 2.2. Convexity of simple functions with scalar input**

In this Example we use the second order definition of convexity to verify whether each of the functions shown in Figure 2.3 is convex or not.

- $g(w) = w^3$ has second derivative $g''(w) = 6w$ which is not always non-negative, hence $g$ is not convex.
- $g(w) = e^w$ has second derivative $g''(w) = e^w$ which is positive for any choice of $w$, and so $g$ is convex.
- $g(w) = \sin(w)$ has second derivative $g''(w) = -\sin(w)$. Since this is not always non-negative $g$ is non-convex.
- $g(w) = w^2$ has second derivative $g''(w) = 2$, and so $g$ is convex.

**Example 2.3. Convexity of a quadratic function with vector input**

In $N$-dimensions a quadratic function in $\mathbf{w}$ takes the form

$$g(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{r}^T\mathbf{w} + d, \tag{2.8}$$

with its Hessian given by

$$\nabla^2 g(\mathbf{w}) = \frac{1}{2}\left(\mathbf{Q} + \mathbf{Q}^T\right). \tag{2.9}$$

Note that if $\mathbf{Q}$ is symmetric, then $\nabla^2 g(\mathbf{w}) = \mathbf{Q}$. Also note that $\mathbf{r}$ and $d$ have no influence on the convexity of $g$. We now verify convexity for three simple instances of $\mathbf{Q}$ where $N = 2$. We discuss a convenient way to determine the convexity of more general vector input functions in the Chapter exercises.

[6] While there are a number of ways to formally check that a function is convex we will see that the second order approach is especially convenient. The interested reader can see Appendix D for additional information regarding convex functions.

- When $\mathbf{Q} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ the Hessian $\nabla^2 g(\mathbf{w}) = \mathbf{Q}$ has two eigenvalues equaling 2, so the corresponding quadratic, shown in the left panel of Figure 2.4, is convex.
- When $\mathbf{Q} = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$ again the Hessian is $\nabla^2 g(\mathbf{w}) = \mathbf{Q}$ has two eigenvalues (2 and 0), so the corresponding quadratic, shown in the middle panel of Figure 2.4, is convex.
- When $\mathbf{Q} = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$ again the Hessian is $\nabla^2 g(\mathbf{w}) = \mathbf{Q}$ and has eigenvalues 2 and $-2$, so the corresponding quadratic, shown in the right panel of Figure 2.4, is non-convex.

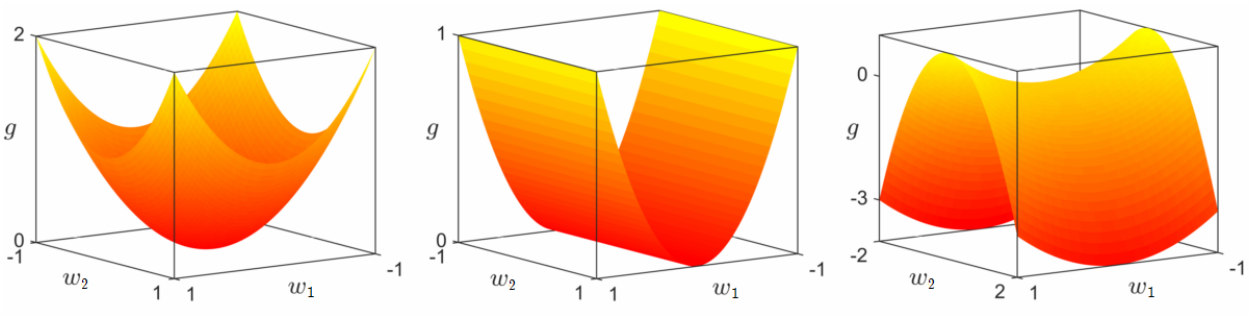


**Figure 2.4.** *Three quadratic functions of the form* $g(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{r}^T\mathbf{w} + d$ *generated by different instances of matrix* $\mathbf{Q}$ *in Example 2.3. In all three cases* $\mathbf{r} = \mathbf{0}_{2\times 1}$ *and* $d = 0$. *As can be visually verified only the first two functions are convex. The last 'saddle-looking' function on the right has a saddle point at zero!*

## 2.2 Numerical methods for optimization

In this Section we introduce two basic but widely used numerical techniques, known as *gradient descent* and *Newton's method*, for finding minima of a function $g(\mathbf{w})$. The formal manner of describing the minimization of a function $g$ is commonly written as

$$\underset{\mathbf{w}}{\text{minimize}} \quad g(\mathbf{w}), \tag{2.10}$$

which is simply shorthand for saying "minimize $g$ over all input values $\mathbf{w}$". The solution to (2.10), referred to as the *optimal* $\mathbf{w}$, is typically denoted as $\mathbf{w}^\star$. While both gradient descent and Newton's method operate sequentially by finding points at which $g$ gets smaller and smaller, both methods are only guaranteed to find stationary points of $g$, i.e., those points satisfying the first order condition (discussed in Section 2.1.2). Thus one can also consider these techniques for numerically solving the system of $N$ equations $\nabla g(\mathbf{w}) = \mathbf{0}_{N\times 1}$.

### 2.2.1 The big picture

All numerical optimization schemes for minimization of a general function $g$ work as follows:

(1) Start the minimization process from some *initial point* $\mathbf{w}^0$.

(2) Take *iterative* steps denoted by $\mathbf{w}^1$, $\mathbf{w}^2$, ..., going "downhill" towards a stationary point of $g$.

(3) Repeat step (2) until the sequence of points converges to a stationary point of $g$.

This idea is illustrated in Figure 2.5 for the minimization of a non-convex function. Notice that since this function has three stationary points, the one we reach by traveling downhill depends entirely on where we begin the optimization process. Ideally we would like to find the *global minimum*, or the lowest of the function's minima, which for a general non-convex function requires that we run the procedure several times with different initializations (or starting points).

As we will see in later Chapters many important machine learning cost functions are convex and hence have only global minima, as in Figure 2.6, in which case any initialization will recover a global minimum.

The numerical methods discussed here halt at a stationary point $\mathbf{w}$, that is a point where $\nabla g(\mathbf{w}) = \mathbf{0}_{N\times 1}$, which as we have previously seen may or may not constitute a minimum of $g$ if $g$ is non-convex. However this issue does not at all preclude the use of non-convex cost functions in machine learning (or other scientific disciplines), it is simply worth being aware of.

### 2.2.2 Stopping condition

One of several *stopping conditions* may be selected to halt numerical algorithms that seek stationary points of a given function $g$. The two most commonly used stopping criteria include:

(1) When a pre-specified number of iterations are complete.

(2) When the gradient is small enough, i.e., $\left\|\nabla g\left(\mathbf{w}^k\right)\right\|_2 < \epsilon$ for some small $\epsilon > 0$.

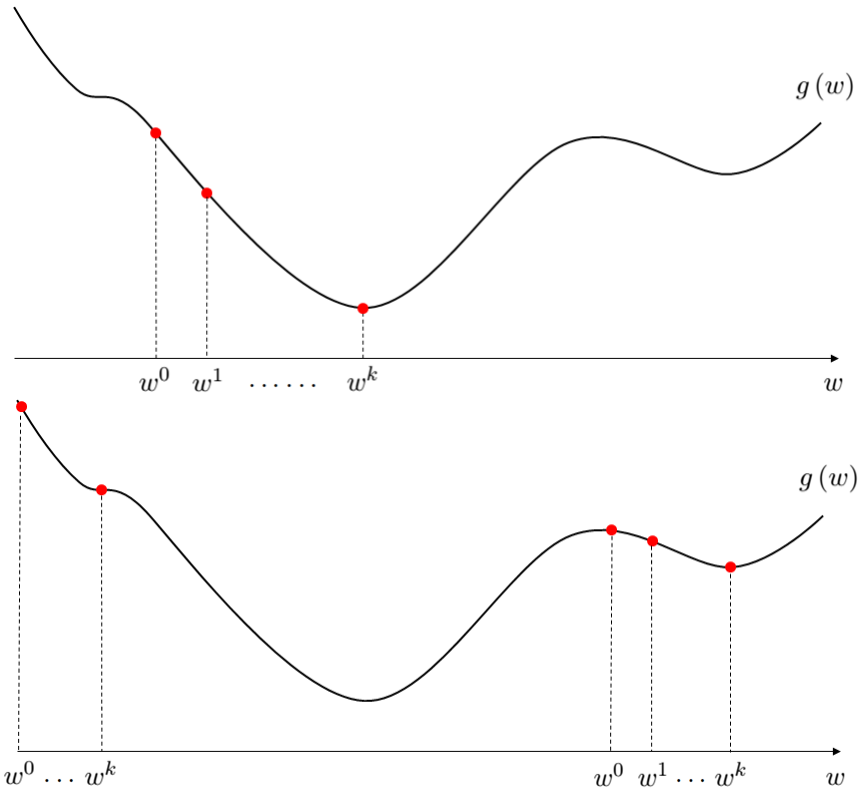


**Figure 2.5.** *The stationary point of a non-convex function found via numerical optimization is dependent on the choice of initial point $w^0$. In the top panel our initialization leads us to find the global minimum, while in the bottom panel the two different initializations lead to a saddle point on the left, and a non-global minimum on the right.*

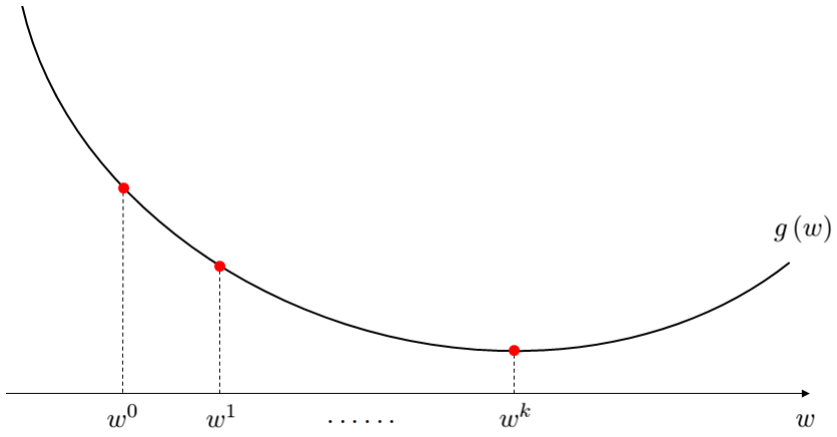


**Figure 2.6.** *A global minimum of a convex function is found via numerical optimization regardless of our choice of the initial point $w^0$.*

Perhaps the most naive stopping condition for a numerical algorithm is to halt the procedure after a pre-defined number of iterations. Note that this extremely simple condition does not provide any convergence guarantee, and hence is typically used in practice in conjunction with other stopping criteria as a necessary cap on the number of iterations when the convergence is achieved slowly. The second condition directly translates our desire to finding a stationary point at which the gradient is by definition zero. One could also stop the procedure when continuing it does not considerably decrease the objective function (or the stationary point itself) from one iteration to the next.

### 2.2.3 Gradient descent

The defining characteristic differentiating various numerical optimization methods is the way iterative steps are taken for reducing the value of $g$. The two classic methods, gradient descent and Newton's method, use local models for the function at each step in order to find smaller and smaller values of the function. As illustrated in Figure 2.7 the basic idea with gradient descent is to build a linear model of the function $g$, determine the 'downward' direction on this hyperplane, travel a short distance along this direction, hop back on to the function $g$, and repeat until convergence. Starting at an initial point $\mathbf{w}^0$ and by carefully choosing how far we travel at each step, the gradient descent procedure produces a sequence of points $\mathbf{w}^1$, $\mathbf{w}^2$, $\mathbf{w}^3$..., that shrinks the value of $g$ at each step and eventually reaches a stationary point of $g$.

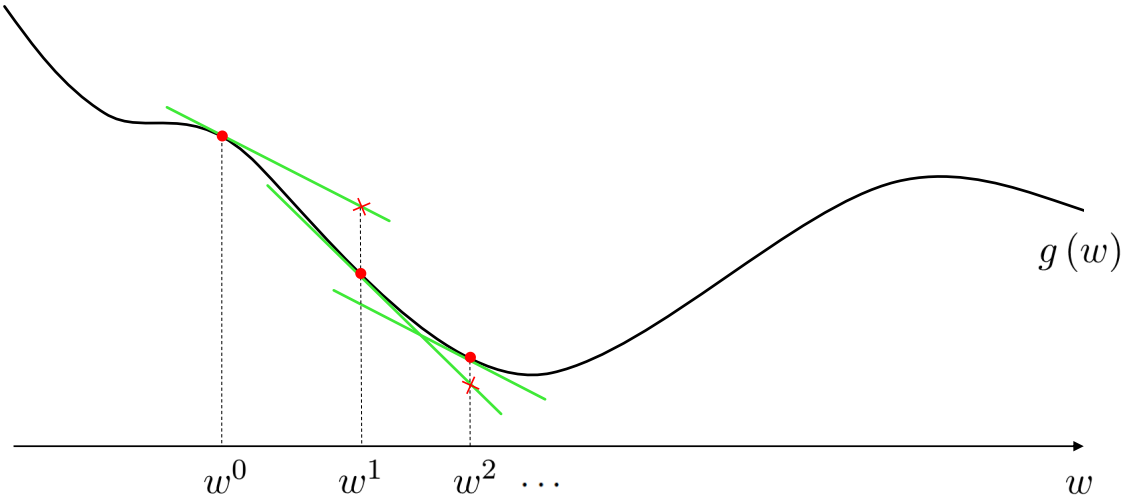


**Figure 2.7.** *With the gradient descent method, we travel in the downward direction of a linear approximation, hop back onto the function, and repeat in order to find a stationary point of g.*

Formally, beginning at an initial point $\mathbf{w}^0$ the linear model of $g$ at this point is given precisely

by the first order Taylor series approximation in (2.3) centered at $\mathbf{w}^0$

$$h\left(\mathbf{w}\right) = g\left(\mathbf{w}^0\right) + \nabla g\left(\mathbf{w}^0\right)^T\left(\mathbf{w} - \mathbf{w}^0\right). \tag{2.11}$$

We now take our first step by traveling in the direction in which the tangent hyperplane most sharply angles downward (referred to as the *steepest descent direction*). Using a simple calculus-based argument[13] it can be shown that this steepest descent direction is given precisely as $-\nabla g\left(\mathbf{w}^0\right)$. Thus we descend in the direction of the negative gradient (hence the name of the algorithm, *gradient descent*) taking our first step to a point $\mathbf{w}^1$ where

$$\mathbf{w}^1 = \mathbf{w}^0 - \alpha_1 \nabla g\left(\mathbf{w}^0\right). \tag{2.12}$$

Here $\alpha_1$ is a positive constant, called a *step length* (sometimes referred to as a *learning rate*), that controls how far we descend in the negative gradient direction from our initial point $\mathbf{w}^0$. We then repeat this procedure constructing the first order Taylor series approximation at $\mathbf{w}^1$, travel in its steepest descent direction which is again given by the negative gradient $-\nabla g\left(\mathbf{w}^1\right)$, taking our second step to the point $\mathbf{w}^2$ where

$$\mathbf{w}^2 = \mathbf{w}^1 - \alpha_2 \nabla g\left(\mathbf{w}^1\right). \tag{2.13}$$

Once again $\alpha_2$ is a small positive step length, perhaps different from $\alpha_1$, that controls the length of our travel along the negative gradient from $\mathbf{w}^1$. This entire procedure is repeated with the $k^{\text{th}}$ step being given analogously as

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha_k \nabla g\left(\mathbf{w}^{k-1}\right), \tag{2.14}$$

where $\alpha_k$ is the step length associated with the $k^{\text{th}}$ gradient descent step. Note that this procedure only stops when at some iteration $\nabla g\left(\mathbf{w}^{k-1}\right) \approx \mathbf{0}_{N\times 1}$, that is when we have approximately satisfied the first order condition and essentially reached a stationary point $\mathbf{w}^k$ of $g$. For easy reference we give the gradient descent procedure (with a fixed given step length) in Algorithm 2.1.

How do we choose a proper value for the step length at each iteration? As illustrated in the top panel of Figure 2.8 it cannot be set too large, because then the algorithm travels too far at each step and will bounce around and perhaps never converge. On the other hand, if $\alpha_k$ is made too small at each step as illustrated in the bottom panel of Figure 2.8, the algorithm crawls downward far too slowly and may never reach the stationary point in our lifetime!

[13] Note that for a unit length direction $\mathbf{d}$, $h$ in (2.11) can be written as $h\left(\mathbf{d}\right) = g\left(\mathbf{w}^0\right) - \nabla g\left(\mathbf{w}^0\right)^T\mathbf{w}^0 + \nabla g\left(\mathbf{w}^0\right)^T\mathbf{d}$, where the first two terms on the right hand side are constant with respect to $\mathbf{d}$. Thus the unit length direction $\mathbf{d}$ that minimizes the inner product $\nabla g\left(\mathbf{w}^0\right)^T\mathbf{d}$ will result in the sharpest descent in $h$. From the inner product rule (see Appendix IV) this is smallest when $\mathbf{d} = -\frac{\nabla g\left(\mathbf{w}^0\right)}{\|\nabla g(\mathbf{w}^0)\|_2}$, and so the steepest descent direction is indeed $-\nabla g\left(\mathbf{w}^0\right)$.

**Algorithm 2.1** Gradient descent (with fixed step length)

**Input:** differentiable function $g$, fixed step length $\alpha$, and initial point $\mathbf{w}^0$

$k = 1$

**Repeat until stopping condition is met:**

$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$

$k \leftarrow k + 1$

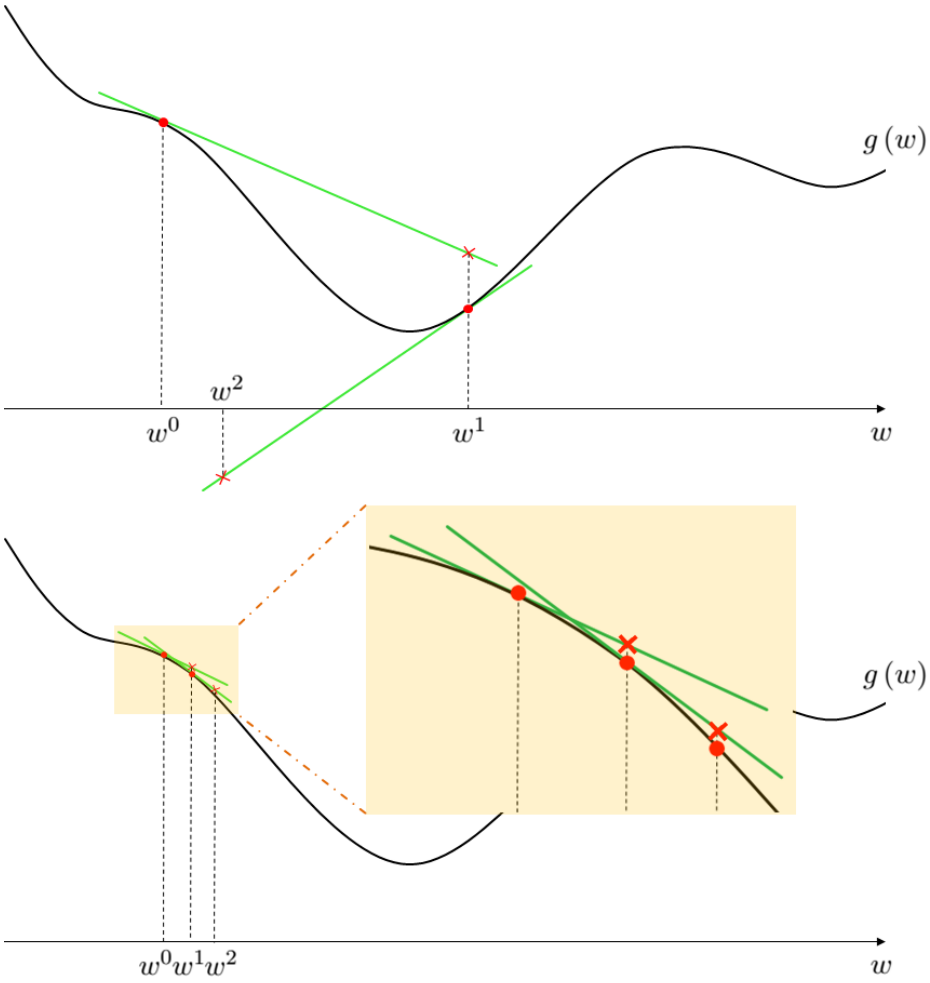


**Figure 2.8.** *The effect of step length on the convergence of gradient descent. (top) Too large of step length and we may overshoot the minimum and possibly never converge. (bottom) A small step length makes the gradient descent converge to the minimum very slowly.*

For machine learning problems it is common practice to choose a value for the step length by trial and error. In other words, just try a range of values, for each performing a complete run of the gradient descent procedure with the step length at every iteration fixed at this value. One can then choose the particular step length value that provides (the fastest) convergence (you can practice this in Exercise 2.14). There are also formal methods that do not rely on trial and error for determining appropriate step lengths which guarantee convergence. These methods are detailed in Section 8.1 and 8.2.

> The step length for gradient descent can be chosen by trial and error and fixed for all iterations, or determined using the formal schemes described in Section 8.1 and 8.2.

**Example 2.4. Gradient descent for a cost function with scalar input**

Suppose that $g(w) = \log\left(1 + e^{w^2}\right)$, whose first derivative is given as $g'(w) = \frac{2e^{w^2}w}{1+e^{w^2}}$. As illustrated in Figure 2.9 $g$ is convex, and with the initial point $w^0 = 1$ and a step length fixed at $\alpha = 10^{-1}$ (determined by trial and error) gradient descent requires a fair number of steps to converge to the global minimum of $g$. Specifically, it takes 15 steps for gradient descent to reach a point at which the absolute value of the derivative falls below $\epsilon = 10^{-3}$.

**Example 2.5. Gradient descent for a cost function with vector input**

Take $g(\mathbf{w}) = -\cos\left(2\pi\mathbf{w}^T\mathbf{w}\right) + 2\mathbf{w}^T\mathbf{w}$, where $\mathbf{w}$ is a 2-dimensional vector, and the gradient of $g$ is given as $\nabla g(\mathbf{w}) = 4\pi\sin\left(2\pi\mathbf{w}^T\mathbf{w}\right)\mathbf{w} + 4\mathbf{w}$. In Figure 2.10 we show the objective value of gradient descent steps $g\left(\mathbf{w}^k\right)$ with two starting points where a fixed step length of $\alpha = 10^{-3}$ (determined by trial and error) was used for all iterations of each run. In this instance one of the starting points ($\mathbf{w}^0 = \begin{bmatrix} -0.7 & 0 \end{bmatrix}^T$) allows gradient descent to reach the global minimum of the function, while the other ($\mathbf{w}^0 = \begin{bmatrix} 0.85 & 0.85 \end{bmatrix}^T$) ends up at a local minimum of the surface.

### 2.2.4 Newton's method

Like gradient descent, Newton's method works by using approximations to a function at each step in order to lower its value. However with Newton's method a quadratic approximation, again generated via the Taylor series approximation, is used. As illustrated in the top panel of Figure 2.11, starting at an initial point $\mathbf{w}^0$ Newton's method produces a sequence of

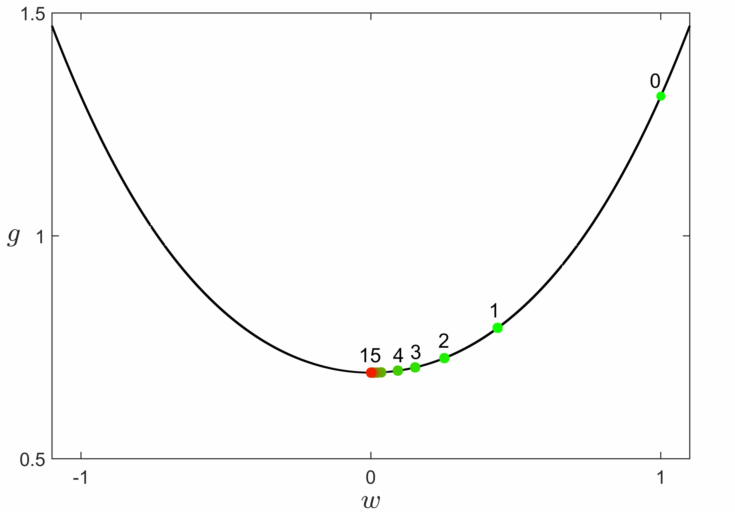


**Figure 2.9.** *Gradient descent applied for minimizing a convex function with scalar input (see Example 2.4) initialized at* $w^0 = 1$. *At each step* $w^k$ *of the procedure, the point* $\left(w^k, g\left(w^k\right)\right)$ *is colored green and numbered on the function itself, with the final point colored red. Notice that only the first* 5 *points as well as the final point are numbered in this Figure since the rest blur together as the minimum is approached.*

points $\mathbf{w}^1$, $\mathbf{w}^2$, ..., that minimizes $g$ by repeatedly creating a quadratic approximation to the function, traveling to a stationary point of this quadratic, and hopping back onto the function. Because Newton's method uses quadratic as opposed to linear approximations at each step, with a quadratic more closely mimicking the associated function, it is often much more effective than gradient descent (in the sense that it requires far fewer steps for convergence [39, 13]). However this reliance on quadratic information makes Newton's method more difficult to use with non-convex functions[16], since at concave portions of such a function the algorithm can climb to a maximum, as illustrated in the bottom panel of Figure 2.11.

Formally, beginning at a point $\mathbf{w}^0$ the quadratic model of $g$ at this point is given precisely by the second order Taylor series approximation in (2.4) centered at $\mathbf{w}^0$

$$h\left(\mathbf{w}\right) = g\left(\mathbf{w}^0\right) + \nabla g\left(\mathbf{w}^0\right)^T\left(\mathbf{w} - \mathbf{w}^0\right) + \frac{1}{2}\left(\mathbf{w} - \mathbf{w}^0\right)^T \nabla^2 g\left(\mathbf{w}^0\right)\left(\mathbf{w} - \mathbf{w}^0\right). \tag{2.15}$$

We now wish to travel to a stationary point of this quadratic which is a minimum in the

[16] A number of procedures exist that adjust Newton's method at concave portions of such a function in order to make it more effective for use with non-convex functions. See exercise 3.14 as well as [39, 48] for further information.

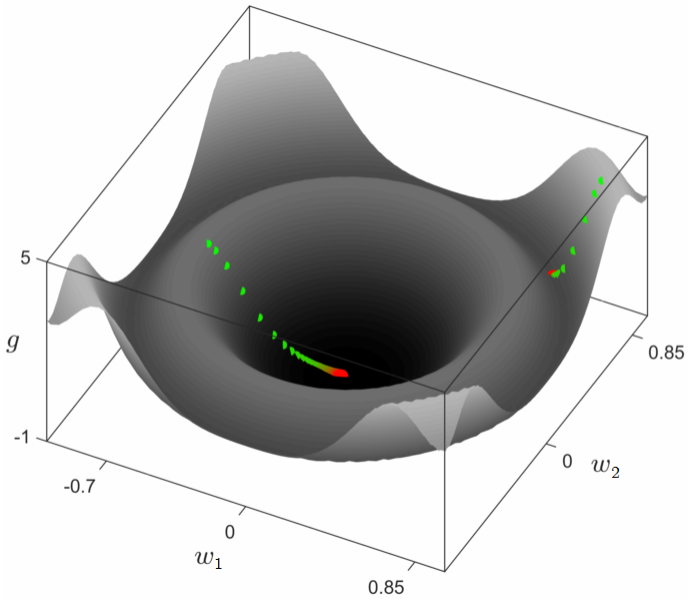


**Figure 2.10.** *Gradient descent applied for minimizing a non-convex function with 2-dimensional input (see Example 2.5). Of the two initializations used here only one leads to the global minimum. For each run the point* $\left(w_1^k,\, w_2^k,\, g\left(w_1^k, w_2^k\right)\right)$ *is colored green at each step of the procedure, with the final point colored red.*

case where $g$ is convex[18]. To do this we can use the first order condition (see Section 2.1.2) by setting the gradient of $h$ to zero and solving for $\mathbf{w}$. This gives the $N \times N$ system of linear equations[19]

$$\nabla^2 g\left(\mathbf{w}^0\right)\mathbf{w} = \nabla^2 g\left(\mathbf{w}^0\right)\mathbf{w}^0 - \nabla g\left(\mathbf{w}^0\right). \tag{2.16}$$

A solution to this system of equations gives the first point $\mathbf{w}^1$ traveled to by Newton's method. To take the next step we repeat this procedure, forming a quadratic Taylor series approximation of $g$ (this time centered at $\mathbf{w}^1$) and determine a stationary point of this quadratic by checking the first order condition. This leads to the same kind of linear system of equations

$$\nabla^2 g\left(\mathbf{w}^1\right)\mathbf{w} = \nabla^2 g\left(\mathbf{w}^1\right)\mathbf{w}^1 - \nabla g\left(\mathbf{w}^1\right), \tag{2.17}$$

a solution of which provides the second step to the point $\mathbf{w}^2$. This entire procedure is repeated until convergence, with the $k^{\text{th}}$ Newton step $\mathbf{w}^k$ defined as a stationary point of the

[18] A common way of adjusting Newton's method for use with non-convex functions is to add a so-called 'regularizer', described in Section 3.3.2, to the original function. See e.g., [39] for further details.

[19] Setting the gradient of $h$ to zero we have $\nabla h\left(\mathbf{w}\right) = \nabla g\left(\mathbf{w}^0\right) + \nabla^2 g\left(\mathbf{w}^0\right)\left(\mathbf{w} - \mathbf{w}^0\right) = \mathbf{0}_{N\times 1}$. Solving for $\mathbf{w}$ then gives the linear system of equations $\nabla^2 g\left(\mathbf{w}^0\right)\mathbf{w} = \nabla^2 g\left(\mathbf{w}^0\right)\mathbf{w}^0 - \nabla g\left(\mathbf{w}^0\right)$, which can be written more familiarly as $\mathbf{A}\mathbf{w} = \mathbf{b}$ where $\mathbf{A}_{N\times N} = \nabla^2 g\left(\mathbf{w}^0\right)$ and $\mathbf{b}_{N\times 1} = \nabla^2 g\left(\mathbf{w}^0\right)\mathbf{w}^0 - \nabla g\left(\mathbf{w}^0\right)$ are a fixed matrix and vector, respectively.

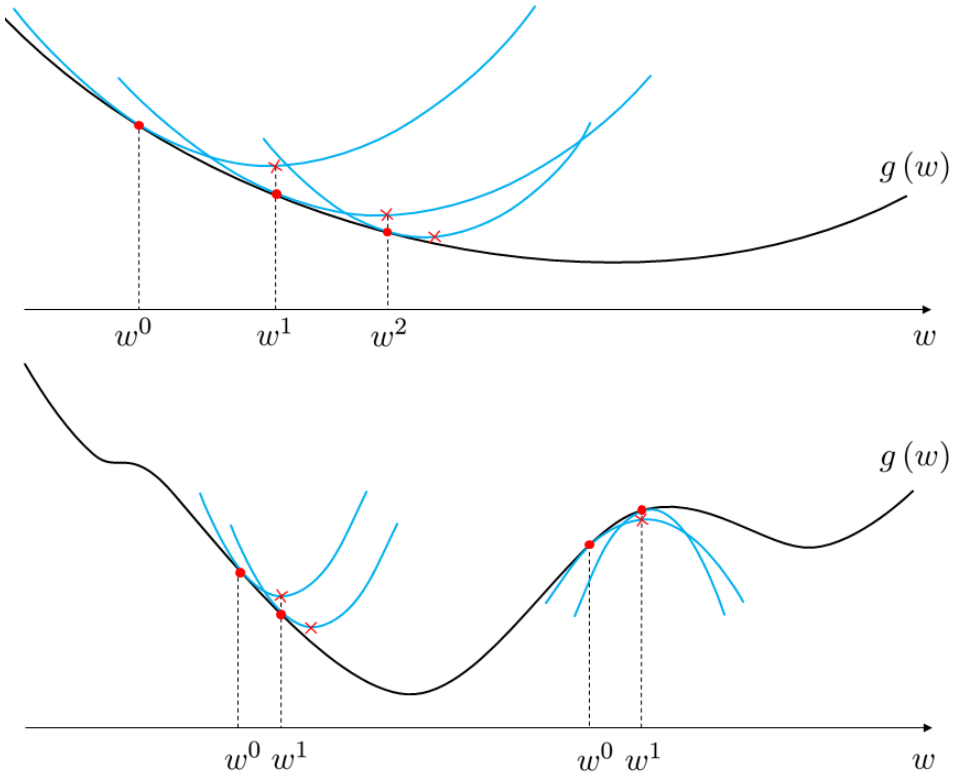


**Figure 2.11.** *Newton's method illustrated. To find a minimum of g Newton's method hops down the stationary points of quadratic approximations generated by g's second order Taylor series. (top panel) For convex functions these quadratic approximations are themselves always convex (upward facing) and so their stationary points are minima, and the sequence leads to a minimum of the original function. (bottom panel) For non-convex functions quadratic approximations can be concave or convex depending on where they are constructed, leading the algorithm to possibly converge to a maximum.*

quadratic approximation centered at $\mathbf{w}^{k-1}$

$$h\left(\mathbf{w}\right) = g\left(\mathbf{w}^{k-1}\right) + \nabla g\left(\mathbf{w}^{k-1}\right)^T\left(\mathbf{w} - \mathbf{w}^{k-1}\right) + \frac{1}{2}\left(\mathbf{w} - \mathbf{w}^{k-1}\right)^T \nabla^2 g\left(\mathbf{w}^{k-1}\right)\left(\mathbf{w} - \mathbf{w}^{k-1}\right) \tag{2.18}$$

which, again applying the first order condition for optimality and solving for $\mathbf{w}$, gives a linear system of equations

$$\nabla^2 g\left(\mathbf{w}^{k-1}\right)\mathbf{w} = \nabla^2 g\left(\mathbf{w}^{k-1}\right)\mathbf{w}^{k-1} - \nabla g\left(\mathbf{w}^{k-1}\right). \tag{2.19}$$

As with gradient descent, notice that these steps halt when at some $k^{\text{th}}$ step we have $\nabla g\left(\mathbf{w}^{k-1}\right) \approx \mathbf{0}_{N\times 1}$, i.e., when we have approximately satisfied the first order condition

essentially recovering a stationary point $\mathbf{w}^k$ of $g$. For convenience we give the Newton's method scheme in Algorithm 2.2.

**Algorithm 2.2** Newton's method

**Input:** twice differentiable function $g$, and initial point $\mathbf{w}^0$

$k = 1$

**Repeat until stopping condition is met:**

Solve the system $\nabla^2 g\left(\mathbf{w}^{k-1}\right)\mathbf{w}^k = \nabla^2 g\left(\mathbf{w}^{k-1}\right)\mathbf{w}^{k-1} - \nabla g\left(\mathbf{w}^{k-1}\right)$ for $\mathbf{w}^k$.

$k \leftarrow k + 1$

Notice that in cases where the matrix $\nabla^2 g\left(\mathbf{w}^{k-1}\right)$ is invertible we may write the solution to the system in (2.19) algebraically as

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \left[\nabla^2 g\left(\mathbf{w}^{k-1}\right)\right]^{-1} \nabla g\left(\mathbf{w}^{k-1}\right), \tag{2.20}$$

which makes the Newton step look like a gradient step in (2.14), replacing the step length with the inverted Hessian matrix. When the Hessian is not invertible (and there are infinitely many solutions to the system in (2.19)) we may employ the so-called pseudo-inverse (see Appendix C) of the Hessian, denoted as $\left[\nabla^2 g\left(\mathbf{w}^{k-1}\right)\right]^{\dagger}$, and write the update analogously as

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \left[\nabla^2 g\left(\mathbf{w}^{k-1}\right)\right]^{\dagger} \nabla g\left(\mathbf{w}^{k-1}\right). \tag{2.21}$$

While not the most computationally efficient[23] method of solving the Newton system, the solution above is always the *smallest* one possible and so (practically speaking) it can be a useful choice when the alternative is an unfamiliar numerical linear algebra solver (e.g., one which may return very large solutions to the system in (2.19), producing numerical instability in subsequent Newton steps).

As previously mentioned because it uses more precise second order information, Newton's method converges in a much smaller number of steps than gradient descent. This comes at the cost of having to store a Hessian matrix and solve a corresponding linear systems at each step, which with modern computational resources is not typically problematic for functions with up to several thousand input variables[24]. For such (especially convex) functions the standard Newton's method described is highly effective.

[23] It always more computationally efficient to find $\mathbf{w}^k$ by directly solving the linear system in (2.19) using numerical linear algebra software, rather than by calculating the inverse or pseudo-inverse of the Hessian and forming the explicit update in (2.20) or (2.21). See e.g., [13] for further details.

[24] For larger dimensional input storing the Hessian matrix itself can become problematic, let alone solving the associated linear system at each step. For example with a $10,000$-dimensional input $\mathbf{w}$ the corresponding Hessian matrix will be of size $10,000 \times 10,000$, with $10^8$ values to store for the Hessian matrix alone. Several approaches exist which aim at addressing both the storage and computation issues associated with these large

**Example 2.6. Newton's method for a cost function with scalar input**

Let us consider again the function $g(w) = \log\left(1 + e^{w^2}\right)$ from Example 2.4, whose second derivative is given as $g''(w) = \frac{2e^{w^2}\left(2w^2 + e^{w^2} + 1\right)}{\left(1+e^{w^2}\right)^2}$. Note that $g''(w) > 0$ for all $w$, and thus the $k^{\text{th}}$ Newton step in (2.20) for a scalar $w$ reduces to

$$w^k = w^{k-1} - \frac{g'\left(w^{k-1}\right)}{g''\left(w^{k-1}\right)}. \tag{2.22}$$

As illustrated in Figure 2.12, beginning at point $w^0 = 1$ we need only three Newton steps to reach the minimum of $g$ (where the absolute value of the derivative falls below $\epsilon = 10^{-3}$). This is significantly fewer steps than the gradient descent procedure shown in Figure 2.9.

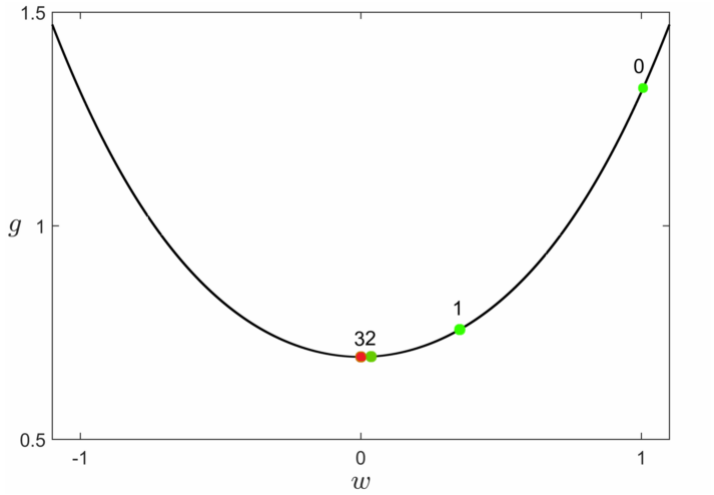


**Figure 2.12.** *Newton's method applied for minimizing a convex function with scalar input (see Example 2.6) initialized at* $w^0 = 1$.

**Example 2.7. Newton's method for a cost function with vector input**

Let $g$ be the quadratic function $g(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{r}^T\mathbf{w} + d$ where $\mathbf{Q} = \begin{bmatrix} 1 & 0.75 \\ 0.75 & 1 \end{bmatrix}$, $\mathbf{r} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$, and $d = 0$. As shown in Figure 2.13, with the initial point $\mathbf{w}^0 = \begin{bmatrix} 10 & 10 \end{bmatrix}^T$

linear systems, e.g., subsampling methods (termed quasi-Newton methods) [39, 48], methods for exploiting special structure in the second order linear system if it is present [13], and Conjugate Gradient methods (particularly useful for sparse Hessians) [62].

only one Newton step is required to reach the global minimum of $g$. This is in fact the case regardless of the initial point chosen since $g$ is quadratic, and thus at any point its quadratic Taylor series approximation is *itself*. This can also be seen by plugging $\nabla g\left(\mathbf{w}^0\right) = \mathbf{Q}\mathbf{w}^0 + \mathbf{r}$ and $\nabla^2 g\left(\mathbf{w}^0\right) = \mathbf{Q}$ into (2.19), giving

$$\mathbf{Q}\mathbf{w}^1 = \mathbf{Q}\mathbf{w}^0 - \left(\mathbf{Q}\mathbf{w}^0 + \mathbf{r}\right) = -\mathbf{r}. \tag{2.23}$$

Note that using (2.19) to compute $\mathbf{w}^2$ results in the exact same linear system since

$$\mathbf{Q}\mathbf{w}^2 = \mathbf{Q}\mathbf{w}^1 - \left(\mathbf{Q}\mathbf{w}^1 + \mathbf{r}\right) = -\mathbf{r}. \tag{2.24}$$

Therefore only one Newton step is required to find the minimum of $g$ at $-\mathbf{Q}^{-1}\mathbf{r} = \begin{bmatrix} -0.57 & -0.57 \end{bmatrix}^T$.

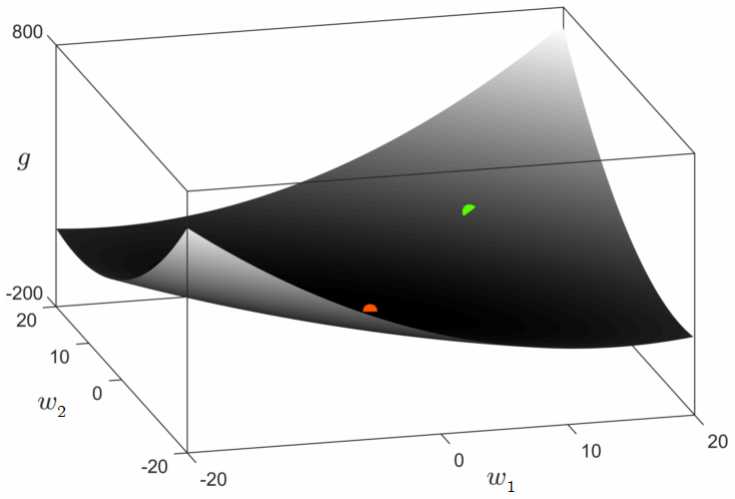


**Figure 2.13.** *Newton's method applied for minimizing a convex quadratic function with two-dimensional input (see Example 2.7). Initialized at any point, only one Newton step is required to reach the minimum.*

## 2.3 Summary

In this Chapter we have seen how to formalize the search for the minima of a general function $g$. In Section 2.1.2 we saw how calculus provides a useful condition for characterizing the minima, maxima, and saddle points of a function (together known as stationary points) via the first order condition for optimality. In the convenient case of a convex function, as discussed in Section 2.1.3, all such stationary points are global minima of the function. As described in Section 2.2 numerical algorithms aim at minimizing a function, but are only

guaranteed to converge to stationary points. Two commonly used numerical methods, gradient descent and Newton's method, employ first and second order Taylor series expansions of a function respectively, in order to produce a converging sequence. Newton's method, which is easier to apply to convex functions, converges in far fewer steps than gradient descent and, unlike gradient descent, requires no step length be determined.

## 2.4 Exercises

### Section 2.1 exercises

**Exercise 2.1. Practice derivative calculations**

Compute the first and second derivatives of the following functions (remember to use the product/chain rules where necessary). *Hint: see the book appendix for more information on how to compute first and second derivatives if these concepts are unfamiliar.*

**a)** $g(w) = \frac{1}{2}qw^2 + rw + d$ where $q$, $r$, and $d$ are constants

**b)** $g(w) = -\cos(2\pi w^2) + w^2$

**c)** $g(w) = \sum_{p=1}^{P} \log(1 + e^{-a_p w})$ where $a_1 ... a_P$ are constants

**Exercise 2.2. Practice derivative calculations II**

Compute the gradient and Hessian matrix of the following (remember to use the product/chain rules where necessary). Note that here $\mathbf{w}$ is an $N \times 1$ dimensional vector in all three cases. *Hint: see the book appendix for more information on how to compute gradients and Hessians if these concepts are unfamiliar.*

**a)** $g(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{r}^T\mathbf{w} + d$, here $\mathbf{Q}$ is an $N \times N$ symmetric matrix (i.e., $\mathbf{Q} = \mathbf{Q}^T$), $\mathbf{r}$ is an $N \times 1$ vector, and $d$ is a scalar.

**b)** $g(\mathbf{w}) = -\cos\left(2\pi\mathbf{w}^T\mathbf{w}\right) + \mathbf{w}^T\mathbf{w}$

**c)** $g(\mathbf{w}) = \sum_{p=1}^{P} \log\left(1 + e^{-\mathbf{a}_p^T\mathbf{w}}\right)$ where $\mathbf{a}_1 ... \mathbf{a}_P$ are $N \times 1$ vectors

**Exercise 2.3. Outer products and outer product matrices**

Let $\mathbf{x}$ and $\mathbf{y}$ be $N \times 1$ and $M \times 1$ vectors respectively. The *outer product* of $\mathbf{x}$ and $\mathbf{y}$, written as $\mathbf{x}\mathbf{y}^T$, is the $N \times M$ matrix defined as

$$\mathbf{x}\,\mathbf{y}^T = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_M \\ x_2y_1 & x_2y_2 & \cdots & x_2y_M \\ \vdots & \vdots & \ddots & \vdots \\ x_Ny_1 & x_Ny_2 & \cdots & x_Ny_M \end{bmatrix}. \tag{2.25}$$

Suppose that $\mathbf{X}$ is an $N \times P$ and $\mathbf{Y}$ is an $M \times P$ matrix and $\mathbf{x}_p$ and $\mathbf{y}_p$ are the $p^{th}$ columns of $\mathbf{X}$ and $\mathbf{Y}$ respectively, verify that $\mathbf{X}\mathbf{Y}^T = \sum_{p=1}^{P} \mathbf{x}_p\mathbf{y}_p^T$ where $\mathbf{x}_p\mathbf{y}_p^T$ is the outer product of $\mathbf{x}_p$ and $\mathbf{y}_p$.

**Exercise 2.4. Taylor series calculations**

Write out the first and second order Taylor series approximations for the following functions.

**a)** $g(w) = \log\left(1 + e^{w^2}\right)$ near a point $v$.

**b)** $g(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{r}^T\mathbf{w} + d$ where $\mathbf{Q}$ is a $N \times N$ symmetric matrix, $\mathbf{r}$ is an $N \times 1$ vector, and $d$ is a constant. In particular show that the second order Taylor series approximation $h(\mathbf{w})$ centered at any point $\mathbf{v}$ is precisely the function $g(\mathbf{w})$ itself. Why is this?

**Exercise 2.5. First order Taylor series geometry**

Verify that the normal vector to the tangent hyperplane generated by the first order Taylor series approximation centered at a point $\mathbf{v}$ shown in equation (2.3) takes the form $\mathbf{n} = \begin{bmatrix} 1 \\ -\nabla g(\mathbf{v}) \end{bmatrix}$.

**Exercise 2.6. First order condition calculations**

Use the first order condition to find all stationary points of each function below.

**a)** $g(w) = w\log(w) + (1-w)\log(1-w)$ where $w$ lies between 0 and 1.

**b)** $g(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{r}^T\mathbf{w} + d$ where $\mathbf{w}$ is 2-dimensional, $\mathbf{Q} = \begin{bmatrix} 1 & 0.75 \\ 0.75 & 1 \end{bmatrix}$ and $\mathbf{r} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$ and $d = 5$.

**Exercise 2.7. Second order convexity calculations**

In Figure 2.14 we show several one-dimensional examples of convex functions. Confirm using the second order definition of convexity that each is indeed convex.

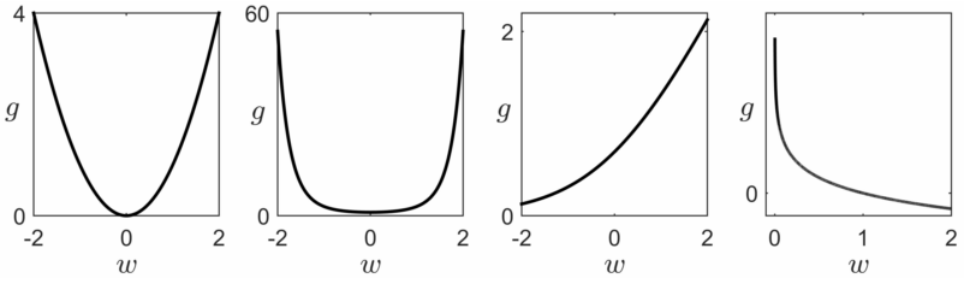


**Figure 2.14.** *From left to right, plots of four convex functions* $g(w) = w^2$, $e^{w^2}$, $log(1 + e^w)$, *and* $-log(w)$.

**Exercise 2.8. A non-convex function whose only stationary point is a global minimum**

**a)** Use the first order condition to determine the stationary point of $g(w) = w\tanh(w)$ where $\tanh(w)$ is the hyperbolic tangent function. To do this you might find it helpful to graph the first derivative $\frac{\partial}{\partial w}g(w)$ and look where it crosses the $w$ axis. Plot the function to verify that the stationary point you find is the global minimum of the function.

**b)** Use the second order definition of convexity to show that $g$ is non-convex. *Hint: you can plot the second derivative* $\frac{\partial^2}{\partial w^2}g(w)$.

**Exercise 2.9. How to determine whether or not the eigenvalues of a symmetric matrix Q are all nonnegative**

In this exercise we investigate an alternative approach to checking that the eigenvalues of a square symmetric matrix $\mathbf{Q}$ (e.g., like a Hessian matrix) are all nonnegative which does not involve explicitly computing the eigenvalues themselves, and is significantly easier to employ in practice.

**a)** Let $\mathbf{Q}$ be an $N \times N$ symmetric matrix. Show that if $\mathbf{Q}$ has all nonnegative eigenvalues then the quantity $\mathbf{z}^T\mathbf{Q}\mathbf{z} \geq 0$ for all $\mathbf{z}$. *Hint*: *use the eigen-decomposition of* $\mathbf{Q} = \mathbf{E}\mathbf{D}\mathbf{E}^T = \sum_{n=1}^{N}\mathbf{e}_n\mathbf{e}_n^T d_n$ *where the* $N \times N$ *matrix* $\mathbf{E}$ *is* $N \times N$ *orthogonal matrix containing eigenvectors* $\mathbf{e}_n$ *of* $\mathbf{Q}$ *as its columns, and* $\mathbf{D} = diag(d_1...d_N)$ *is a diagonal matrix containing the eigenvalues*

*of* $\mathbf{Q}$ *(see the Appendix C for more on the eigenvalue decomposition).*

**b)** Show the converse. That if an $N \times N$ square symmetric matrix $\mathbf{Q}$ satisfies $\mathbf{z}^T\mathbf{Q}\mathbf{z} \geq 0$ for all $\mathbf{z}$ then it must have all nonnegative eigenvalues.

**c)** Use this method to verify that the second order definition of convexity holds for the quadratic function $g(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{r}^T\mathbf{w} + d$, where $\mathbf{Q} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, $\mathbf{r} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$**, and** $d = 1$**.**

**d)** Show that the eigenvalues of $\mathbf{Q} + \lambda\mathbf{I}_{N\times N}$ can all be made to be positive by setting $\lambda$ large enough. What is the smallest value of $\lambda$ that will make this happen?

**Exercise 2.10. Outer product matrices have all nonnegative eigenvalues**

**a)** Use the method described in exercise 2.9 to verify that for any $N$ length vector $\mathbf{x}$ the $N \times N$ outer product matrix $\mathbf{x}\mathbf{x}^T$ has all nonnegative eigenvalues.

**b)** Similarly show that for any set of $P$ vectors $\mathbf{x}_1 ... \mathbf{x}_P$ of length $N$ that the sum of outer product matrices $\sum_{p=1}^{P} \delta_p \mathbf{x}_p \mathbf{x}_p^T$ has all nonnegative eigenvalues if each $\delta_p \geq 0$.

**c) Show that the matrix** $\sum_{p=1}^{P} \delta_p \mathbf{x}_p \mathbf{x}_p^T + \lambda\mathbf{I}_{N\times N}$ where each $\delta_p \geq 0$ and $\lambda > 0$ has all positive eigenvalues.

**Exercise 2.11. An easier way to check the second order definition of convexity**

Recall that the second order definition of convexity for a vector input function $g(\mathbf{w})$ requires that we verify whether or not the eigenvalues of $\nabla^2 g(\mathbf{w})$ are nonnegative for each input $\mathbf{w}$. However to explicitly compute the eigenvalues of the Hessian in order to check this is a cumbersome or even impossible task for all but the nicest of functions. Here we use the result of exercise 2.9 to express the second order definition of convexity in a way that is often much easier to employ in practice.

**a)** Using the result of the exercise 2.9 conclude **that** the second order definition of convexity for vector input functions $g(\mathbf{w})$, that the eigenvalues of the Hessian $\nabla^2 g(\mathbf{w})$ are nonnegative at every $\mathbf{w}$, is equivalently stated as the quantity $\mathbf{z}^T\left(\nabla^2 g(\mathbf{w})\right)\mathbf{z} \geq 0$ holding at each $\mathbf{w}$ for

all $\mathbf{z}$.

**b)** Use this manner of expressing the second order definition of convexity to verify that the general quadratic function $g(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{Q}\mathbf{w} + \mathbf{r}^T\mathbf{w} + d$, where $\mathbf{Q}$ is symmetric and known to have all nonnegative eigenvalues and $\mathbf{r}$ and $d$ are arbitrary, always defines a convex function.

**c)** Verify that $g(\mathbf{w}) = -\cos\left(2\pi\mathbf{w}^T\mathbf{w}\right) + \mathbf{w}^T\mathbf{w}$ is non-convex by showing that it does *not* satisfy the second order definition of convexity.

## Section 2.2 exercises

**Exercise 2.12. Play with gradient descent code**

Play with the gradient descent demo file ***convex_grad_surrogate*** which illustrates the consequences of using a fixed step length to minimize the simple convex function shown in the left panel of Figure 2.4. Try changing the initial point and fixed-step length to see how the gradient descent path changes. For example, find a step-length that causes the algorithm to diverge (meaning that the steps go off to infinity). Also look inside the gradient descent sub-function and see how it mirrors Algorithm 2.1.

You can also play with ***non-convex_grad_surrogate*** which shows the demo, this time for a curvy non-convex function.

**Exercise 2.13. Code up gradient descent**

In this exercise you will reproduce Figure 2.10 by using gradient descent in order to minimize the function $g(\mathbf{w}) = -\cos\left(2\pi\mathbf{w}^T\mathbf{w}\right) + \mathbf{w}^T\mathbf{w}$. Use the wrapper ***two_d_grad_wrapper_hw*** to perform gradient descent, filling in the form of the gradient in the sub-function

[in,out] = gradient_descent(alpha,w0);

This sub function performs gradient descent is almost entirely complete with exception of the gradient. Here *alpha* is a fixed step length and w0 is the initial point (both provided in the wrapper), the *in* variable contains each gradient step taken i.e., $in = \left\{\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha\nabla g\left(\mathbf{w}^{k-1}\right)\right\}_{k=0}^{K}$ and corresponding *out* is a vector of the objective function evaluated at these steps i.e., $out = \left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$ where $K$ is the total number of steps taken. These are collected so they may be printed on the cost function surface for viewing.

**Exercise 2.14. Tune fixed step length for gradient descent**

When minimizing a function with high dimensional input $g(\mathbf{w})$ using any numerical method, it is helpful to store each iteration and the corresponding objective value at each iteration $g\left(\mathbf{w}^k\right)$ to make sure your algorithm is converging properly. In this example you will use gradient descent to minimize a simple function, and plot the objective values at each step, comparing the effect of different step sizes on convergence rate of the algorithm.

Suppose $g(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ where $\mathbf{w}$ is an $N = 10$ dimensional input vector. This is just a generalization of the simple parabola in one-dimension. $g$ is convex with a single global minima at $\mathbf{w} = \mathbf{0}_{N\times 1}$. Code up gradient descent with a maximum iteration stopping criterion of 100 iterations (with no other stopping conditions). Using the initial point $\mathbf{w}^0 = 10 \cdot \mathbf{1}_{N\times 1}$ run gradient descent with step lengths $\alpha_1 = 0.001$, $\alpha_2 = 0.1$ and $\alpha_3 = 1.001$ and record the objective function value $g\left(\mathbf{w}^k\right)$ at each iteration of each run. Plot these on a single graph like the one shown in Figure 2.15.

Make sure to use the gradient descent as described in Algorithm 2.1 with only the maximum iteration stopping condition.

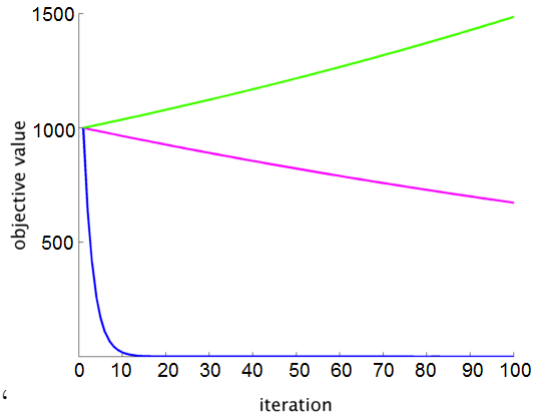


**Figure 2.15.** *Three runs of gradient descent with different step lengths (see text for further details). The smallest step length (in magenta) causes the algorithm to converge very slowly, while the largest (in green) causes it to diverge as the objective value here is increasing. The middle step length value (in blue) causes the algorithm to converge very rapidly to the unique solution of the problem.*

**Exercise 2.15. Geometry of gradient descent step***

The distance between the $(k-1)^{th}$ and $k^{th}$ gradient step can easily be calculated as $\left\|\mathbf{w}^k - \mathbf{w}^{k-1}\right\|_2 = \left\|\left(\mathbf{w}^{k-1} - \alpha\nabla g\left(\mathbf{w}^{k-1}\right)\right) - \mathbf{w}^{k-1}\right\|_2 = \alpha\left\|\nabla g\left(\mathbf{w}^{k-1}\right)\right\|_2$. In this exercise you will compute

the corresponding length traveled along the $(k-1)^{th}$ linear surrogate $l(\mathbf{w}) = g\left(\mathbf{w}^{k-1}\right) + \nabla g\left(\mathbf{w}^{k-1}\right)^T\left(\mathbf{w} - \mathbf{w}^{k-1}\right)$.

In Figure 2.16 we show a detailed description of the precise geometry involved in taking the $k^{th}$ gradient descent step with a fixed step length $\alpha$. Use the details of this picture to show that the corresponding length traveled along the linear surrogate, i.e., $\ell = \left\| \begin{bmatrix} \mathbf{w}^k \\ l\left(\mathbf{w}^k\right) \end{bmatrix} - \begin{bmatrix} \mathbf{w}^{k-1} \\ l\left(\mathbf{w}^{k-1}\right) \end{bmatrix} \right\|_2$, is given precisely as

$$\ell = \alpha\sqrt{1 + \|\nabla g\left(\mathbf{w}^{k-1}\right)\|_2^2}\left\|\nabla g\left(\mathbf{w}^{k-1}\right)\right\|_2 \tag{2.26}$$

*Hint: use the Pythagorean Theorem.*

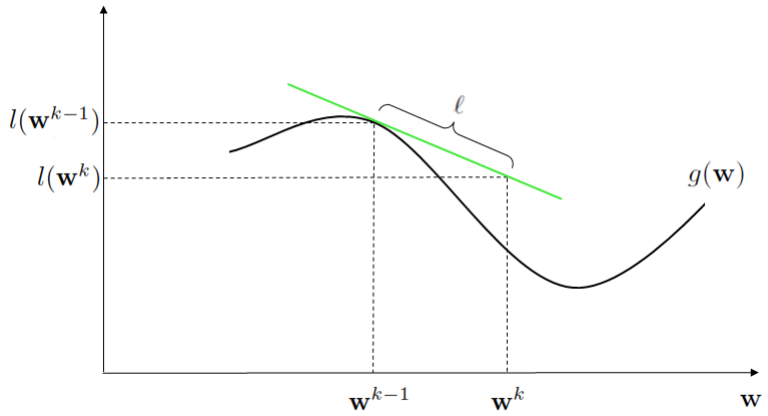


**Figure 2.16.** *The geometry of a gradient descent step.*

### Exercise 2.16. Play with Newton's method code

Play with the Newton's method demo file ***convex_newt_demo*** which illustrates Newton's method applied to minimizing the simple one dimensional function discussed in Example 2.6. Look inside the Newton's method sub-function and see how it mirrors Algorithm 2.2.

Here also play with ***non-convex_newt_demo*** which shows the same kind of demo for a non-convex function. Here note that, as illustrated in Figure 2.11, that if you begin on a concave portion of the function Newton's method will indeed climb to a local maxima of the function!

### Exercise 2.17. Code up Newton's method

**a)** Use the first order condition to determine the unique stationary point of the function $g(\mathbf{w}) = \log\left(1 + e^{\mathbf{w}^T\mathbf{w}}\right)$ where $N = 2$ i.e., $\mathbf{w} = \begin{bmatrix} w_1 & w_2 \end{bmatrix}^T$.

**b)** Make a surface plot of the function $g(\mathbf{w})$ or use the second order definition of convexity to verify that $g(\mathbf{w})$ is convex, implying that the stationary point found in part **a)** is a global minimum. *Hint: to check the second order definition use exercise 2.10.*

**c)** Perform Newton's method to find the minimum of the function $g(\mathbf{w})$ determined in part **a)**. Initialize your algorithm at $\mathbf{w}^0 = \mathbf{1}_{N\times 1}$ and make a plot of the function value $g\left(\mathbf{w}^k\right)$ for ten iterations of Newton's method, as was done in Exercise 2.14) with gradient descent, in order to verify that your algorithm works properly and is converging.

Make sure to follow the Newton's method algorithm as described in Algorithm 2.2 with only the maximum iteration stopping condition, and use the pseudo-inverse solution to each Newton system in your implementation as given in (2.21).

**d)** Now run your Newton's method code from part **c)** again, this time initializing at the point $\mathbf{w}^0 = 4 \cdot \mathbf{1}_{N\times 1}$. While this initialization is further away from the unique minimum of $g(\mathbf{w})$ than the one used in part **c)** your Newton's method algorithm should converge *faster* starting at this point. At first glance this result seems very counterintuitive, as we (rightfully) expect that an initial point closer to a minimum will provoke more rapid convergence of Newton's method!

Can you explain why this result actually makes sense for the particular function $g(\mathbf{w})$ we are minimizing here? Or, in other words, why the minimum of the second order Taylor series approximation of $g(\mathbf{w})$ centered at $\mathbf{w}^0 = 4 \cdot \mathbf{1}_{N\times 1}$ is essentially the minimum of $g(\mathbf{w})$ itself? *Hint: use the fact that for large values of $t$ that $\log(1 + e^t) \approx t$, and that the second order Taylor series approximation of a quadratic function (like the one given in part **b)** of exercise 2.4) is just the quadratic function itself.*