4 Second order optimization techniques

4.1 The geometric anatomy of quadratic functions

As we will see throughout this Chapter, quadratic functions naturally arise when studying second order optimization methods. In this brief Section we discuss quadratic functions with an emphasis on how we determine their overall shape - that is whether they are *convex*, *concave*, or have a more complicated geometry.

4.1.1 The general shape of single-input quadratic functions

The basic formula for a quadratic of a single input takes the familiar form

$$g(w) = a + bw + cw^2 \tag{4.1}$$

where a, b, and c are all constant values controlling the shape of the function. The constant c controls the *convexity* of the function, or whether or not the quadratic faces upwards or downwards. Wen the value of c is *non-negative* the quadratic function is *convex* and points *upwards* (regardless of how the other parameters are set). Conversely when the value of c is *non-positive* the quadratic is *concave* and points *downwards* (when the value of $c = 0$ the quadratic reduces to a linear function, which one can say is both convex and concave).

In Figure 4.1 we plot two simple quadratics centered at the origin: the convex quadratic $g(w) = 6w^2$ (top-left panel) and the concave quadratic $g(w) = -w^2$ (bottom-left panel) to illustrate how the value of c controls the shape and convexity of the general quadratic function.

4.1.2 The general shape of multi-input quadratic functions

The N dimensional quadratic function takes a form that is completely generalized from the single input case, which we write as

$$g(\mathbf{w}) = a + \mathbf{b}^T\mathbf{w} + \mathbf{w}^T\mathbf{C}\,\mathbf{w}\,. \tag{4.2}$$

Here the input $\mathbf{w}$ is N dimensional, a is a constant, $\mathbf{b}$ an $N \times 1$ vector, and $\mathbf{C}$ an $N \times N$ *symmetric* matrix. Because this quadratic is defined along many dimensions

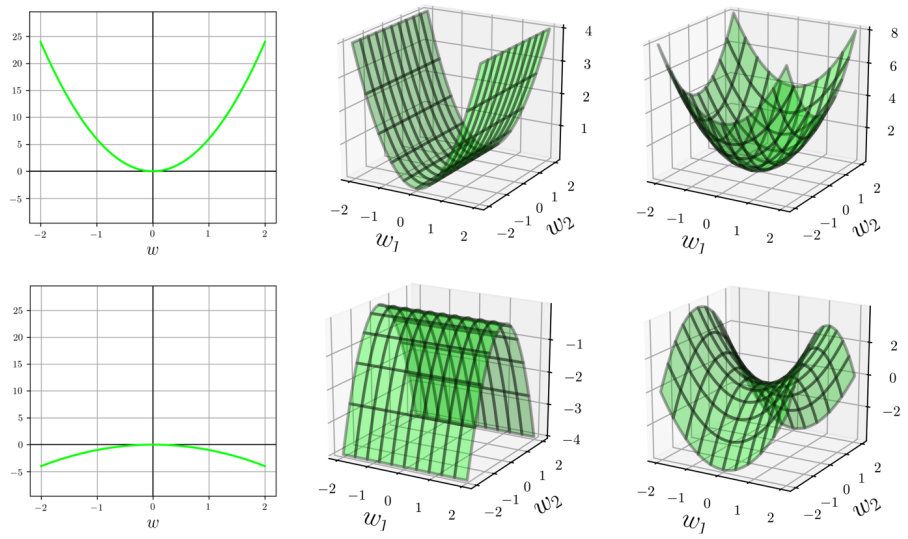

Figure 4.1 An array of quadratic functions. (left column) An example of a convex (top panel) and concave (bottom panel) single-input quadratic function.

it can take on a wider variety of shapes than its single-input analog. For example, it can be convex along certain input dimension and concave along others. The generalization of the single-input test for convexity / concavity reviewed above is *not* whether or not the *values* of $\mathbf{C}$ are positive or negative, but whether its *eigenvalues* are (see Section 7.11.1). If the *eigenvalues* of the matrix *all non-negative* the quadratic is *convex*, if *all non-positive* it is *concave*, if *all equal zero* it reduces to a linear function that is both convex and concave, and otherwise (i.e., if some of its eigenvalues are positive and others negative) it is neither convex nor concave.

In the middle and right columns of Figure 4.1 we show many examples of $N = 2$ multi-input quadratic functions. In all examples we have set the constants a and $\mathbf{b}$ to zero and simply change the values of $\mathbf{C}$. In the middle column we show a convex quadratic (top panel) where $\mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ (whose eigenvalues are 1 and 0) and a concave quadratic (bottom panel) where $\mathbf{C} = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}$ (whose eigenvalues are 0 and -1). In the right column we show another example of a convex quadratic (top panel) where $\mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ (whose eigenvalues are both 1) and and a quadratic that is neither convex or concave where $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ (whose eigenvalues are 1 and -1).

4.2 Curvature and the second order condition for optimality

In this Section discuss we discuss the overall convexity or concavity of quadratic functions - with the second order Taylor series approximation (see Section 7.8 for a review of this concept) in This will help us better understand and quantify the behavior of Newton's method - our main second order algorithm - introduced in the following Section. We then discuss the final calculus-based descriptor of a function's minima and stationary points more broadly speaking: the second order condition for optimality.

4.2.1 Convexity and the second derivative

When discussing convexity / concavity and general mathematical functions we often - since they can take on such a wide swath of different shapes - talk about convexity / concavity *at a point*. If a single-input function g points upwards at a point v we say that it is convex there, and likewise if it points downward there we say it is concave at the point. We can mathematically determine if a function g is *convex* or *concave* at a point v we check its curvature or second derivative information there

$$\begin{aligned} g''(v) \geq 0 \quad &\Longleftrightarrow \quad g \text{ is convex at } v \\ g''(v) \leq 0 \quad &\Longleftrightarrow \quad g \text{ is concave at } v. \end{aligned} \tag{4.3}$$

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*[1].

Example 4.1 Convexity / concavity 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 4.2 is convex or not.

1) $g(w) = w^3$ has second derivative $g''(w) = 6w$ which is not always non-negative, hence g is not convex.

[1] 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 Section 7.13 for additional information regarding convex functions.

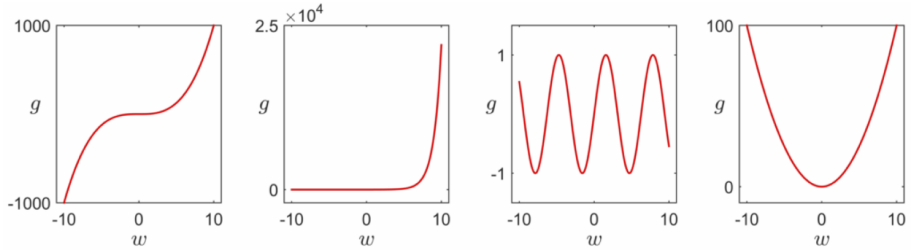

Figure 4.2 From left to right, plots of functions $g(w) = w^3$, $g(w) = e^w$, $g(w) = \sin(w)$, and $g(w) = w^2$.

2) $g(w) = e^w$ has second derivative $g''(w) = e^w$ which is positive for any choice of w, and so g is convex.

3) $g(w) = \sin(w)$ has second derivative $g''(w) = -\sin(w)$. Since this is not always non-negative g is non-convex.

4) $g(w) = w^2$ has second derivative $g''(w) = 2$, and so g is convex.

Example 4.2 **Multi-input quadratics**

The N dimensional quadratic in Equation 4.1.2 has the Hessian matrix $\nabla^2 g(\mathbf{w}) = 2\mathbf{C}$. We can therefore determine the general shape, i.e., if they are convex or concave, by studying the eigenvalues of the matrix $\mathbf{C}$. Indeed we did this in the previous Section in distinguishing the overall shape of the multi-input quadratics shown in in Figure 4.1.

4.2.2 Local curvature and the second order Taylor series

Another way to think about the local convexity or concavity of a function g at a point v is via its Second order Taylor series approximation there (see Section 7.8. This fundamental quadratic approximation, which for a single-input function

$$h(w) = g(v) + \left(\frac{\mathrm{d}}{\mathrm{d}w}g(v)\right)(w - v) + \frac{1}{2}\left(\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(v)\right)(w - v)^2 \tag{4.4}$$

is a true quadratic built using the second derivative of the function. Not only does the second order approximation match the curvature of the underlying function at each point, but if the function is convex at the point v (due to its second derivative being non-negative) then the second order Taylor series is *convex everywhere*. Likewise if the function is concave at v so too is this approximating quadratic concave everywhere.

This concept holds analogously for multi-input functions as well. The second order Taylor series approximation to a function taking in N dimensional input at a point $\mathbf{v}$ is

$$h(\mathbf{w}) = g(\mathbf{v}) + \nabla g(\mathbf{v})^T(\mathbf{w} - \mathbf{v}) + \frac{1}{2}(\mathbf{w} - \mathbf{v})^T\nabla^2 g(\mathbf{v})(\mathbf{w} - \mathbf{v}) \tag{4.5}$$

which not only mirrors the function's curvature at each point but similarly reflects the convexity or concavity of the function g at the point $\mathbf{v}$. When the function g is convex here the corresponding Hessian $\nabla^2 g(\mathbf{v})$ has all non-negative eigenvalues, meaning that the corresponding quadratic function (as discussed in the previous Section) is convex everywhere. Likewise when the function g is concave at the point $\mathbf{v}$, or neither convex or concave there.

Example 4.3 **Local convexity / concavity reflected in the second order Taylor series**

In the top row of Figure 4.3 we show the function (drawn in black)

$$g(w) = \sin(3w) + 0.1w^2 \tag{4.6}$$

along with the second order Taylor series quadratic approximation shown at four example points (one per panel). In each panel the evaluation $g(v)$ of the point v about which the approximation is made is colored green, with the quadratic itself shown in turquoise. The second derivative is shown in the bottom row of the Figure, plotted up to and including the point of expansion. We can see in this Figure that the local convexity / concavity of the function is perfectly reflected in the shape of the associated quadratic approximation. That is, at points of local convexity (like in the first and third column of the Figure) the associated quadratic approximation is convex everywhere. Conversely at points of local concavity (like those shown in the second and fourth column) the associated quadratic approximation is universally concave.

4.2.3 The second order optimality condition

Studying a few simple examples it is easy to come to some far-reaching conclusions about how the second derivative helps unveil the identity of stationary points. In Figure 4.4 we plot three single-input functions along with their first and second order derivatives (shown in the top, middle, and lower rows of the Figure respectively). We mark in green the evaluation of all stationary points by the function in green (where we also show the tangent line in green), as well as the evaluations by the first and second derivatives marking each evaluation in green. In the first and second derivative panels we draw the horizontal zero axis as a dashed black line.

Studying these simple examples in the Figure we can see consistent behavior of certain stationary points. In particular we can see consistency in how the

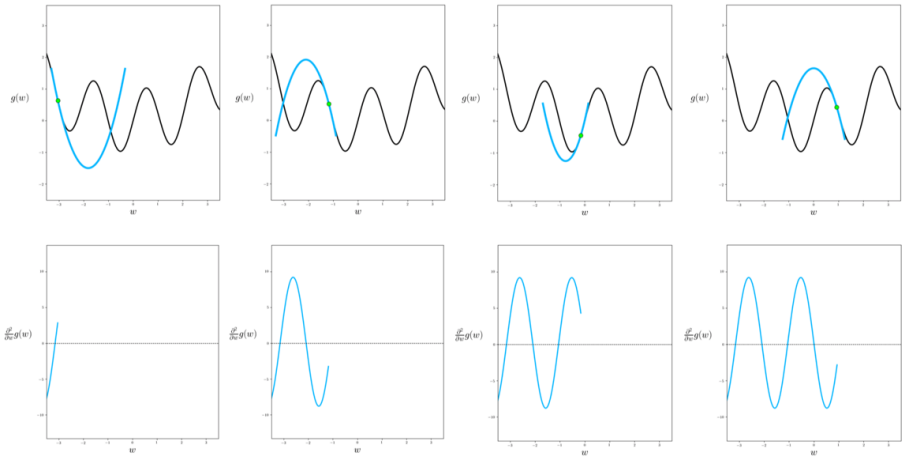

Figure 4.3 Figure associated with Example 4.3. See text for details.

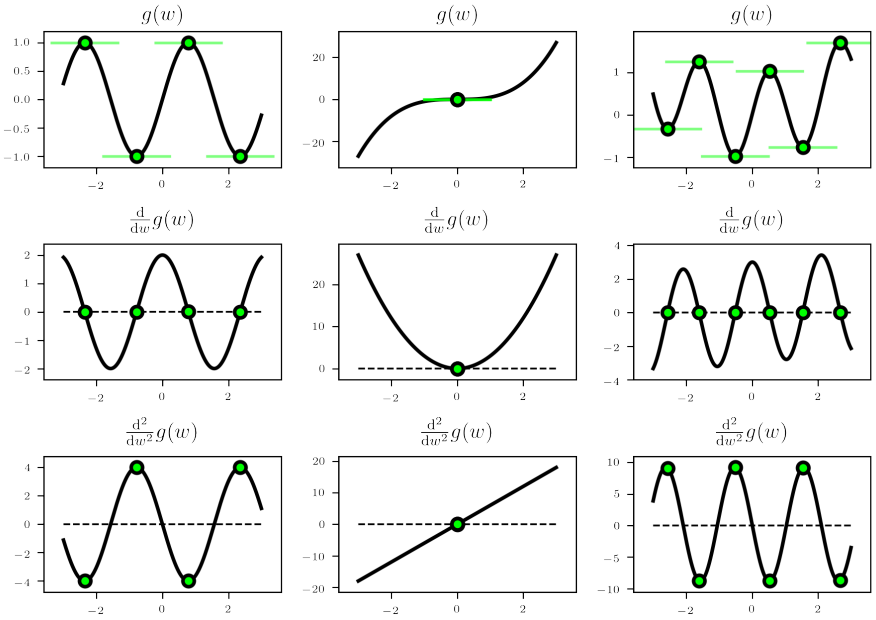

Figure 4.4 Figure associated with Example 4.3. See text for details.

value of a function's second derivative at a stationary point v helps us identify whether it is a local minima, local maxima, or saddle point. In the examples above we see that a stationary point v is

- a local or global minimum if $\frac{\partial^2}{\partial w^2}g(v) > 0$ (since it occurs at *convex* portions of a function)

- a local or global maximum if $\frac{\partial^2}{\partial w^2} g(v) < 0$ (since it occurs at *concave* portions of a function)

- a saddle point if $\frac{\partial^2}{\partial w^2} g(v) = 0$ and $\frac{\partial^2}{\partial w^2} g(v)$ changes sign at v (since it occurs at an *inflection point* of a function, i.e., where a function goes from concave to convex or vice-versa)

These second order characteristics hold more generally as well for any single input function, and taken together are indeed the *second order condition for optimality* for single input functions.

With multi-input functions the analogous second order condition holds. As with all things having to do with convexity/concavity and the second order derivative matrix (a.k.a. the Hessian) the rule translates to the *eigenvalues of the Hessian*. More specifically a stationary point $\mathbf{v}$ of a multi-input function $g(\mathbf{v})$ is

- a local minimum if all eigenvalues of $\nabla^2 g(\mathbf{v})$ are positive (since it occurs at *convex* portions of a function)

- a local maximum if all eigenvalues of $\nabla^2 g(\mathbf{v})$ are negative (since it occurs at *concave* portions of a function)

- a saddle point if the eigenvalues of $\nabla^2 g(\mathbf{v})$ are of mixed values (i.e., some are negative, some are positive) (since it occurs at an *inflection point* of a function)

These rules reduce to those just stated for single-input functions when $N = 1$ since then the Hessian matrix collapses into a single second order derivative.

4.3 Newton's method

Since the first order Taylor series approximation to a function leads to the local optimization framework of gradient descent (Section 3.6), it seems intuitive that higher order Taylor series approximations might similarly yield descent-based algorithms as well. In this Section we introduce a local optimization scheme based on the second order Taylor series approximation - called *Newton's method* (named after its creator, Isaac Newton). Because it is based on the second order approximation Newton's method has natural strengths and weaknesses when compared to gradient descent. In summary we will see that the cumulative effect of these trade-offs is - in general - that Newton's method is especially useful for minimizing *convex* functions of a moderate number of inputs.

4.3.1 The descent direction

In analogy to gradient descent, what would it mean to move in the 'descent direction' defined by the second order approximation at $\mathbf{v}$? Unlike a hyperplane (see Section 3.3), a quadratic does not itself have such a descent direction. However it does have *stationary points* which are global minima when the quadratic is *convex* and maxima with the quadratic is *concave*. As with the descent direction of a hyperplane, we can actually compute the stationary point(s) of the quadratic

approximation fairly easily using the first order condition for optimality (see Section 3.2).

For the single input case the second order Taylor series approximation centered at a point v is shown in Equation 4.2.2 (also see Section 7.8). We can then use the first order condition to solve for the stationary point $w^\star$ of this quadratic (see Example 3.2) by setting its derivative to zero and solving as

$$w^\star = v - \frac{\frac{\mathrm{d}}{\mathrm{d}w}g(v)}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(v)}. \tag{4.7}$$

Note that this update says that to get to the point $w^\star$ we move from v in the direction given by $-\frac{\frac{\mathrm{d}}{\mathrm{d}w}g(v)}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(v)}$.

The same kind of calculation can be made in the multi-input Taylor series quadratic approximations shown in Equation 4.2.2. Setting the gradient of the quadratic approximation to zero (as shown in Example 3.4) and solving gives the stationary point $\mathbf{w}^\star$ where

$$\mathbf{w}^\star = \mathbf{v} - \left(\nabla^2 g(\mathbf{v})\right)^{-1} \nabla g(\mathbf{v}). \tag{4.8}$$

This is the direct analog of the single-input solution, and indeed reduces to it when $N = 1$. It likewise says that in order to get to the new point $\mathbf{w}^\star$ we move from $\mathbf{v}$ in the direction $-\left(\nabla^2 g(\mathbf{v})\right)^{-1} \nabla g(\mathbf{v})$.

So in this notation, what might happen if we have a general function $g(\mathbf{w})$ and at the point $\mathbf{v}$ we form the second order Taylor series approximation there, calculate a stationary point $\mathbf{w}^\star$ of this quadratic, and move to it from $\mathbf{v}$? When might this lead us to a lower point on g? In other words, when might $-\left(\nabla^2 g(\mathbf{v})\right)^{-1} \nabla g(\mathbf{v})$ be a descent direction? Let us examine a few examples to build up our intuition.

Example 4.4 **Stationary point of approximating quadratics I**

In the top row of Figure 4.5 we show the *convex* function

$$g(w) = \frac{1}{50}\left(w^4 + w^2\right) + 0.5 \tag{4.9}$$

along with a second order Taylor series approximation shown centered at several example input points. In each panel the point of expansion is shown as a red circle and its evaluation by the function as a red 'x', the second order Taylor series is shown in light blue, its stationary point $w^\star$ is shown as a green circle, and the evaluations of both the quadratic approximation and the function itself are denoted by a blue and green 'x' respectively.

Since the function itself is convex everywhere the the quadratic approximation not only matches the curvature at each point but is always convex and facing

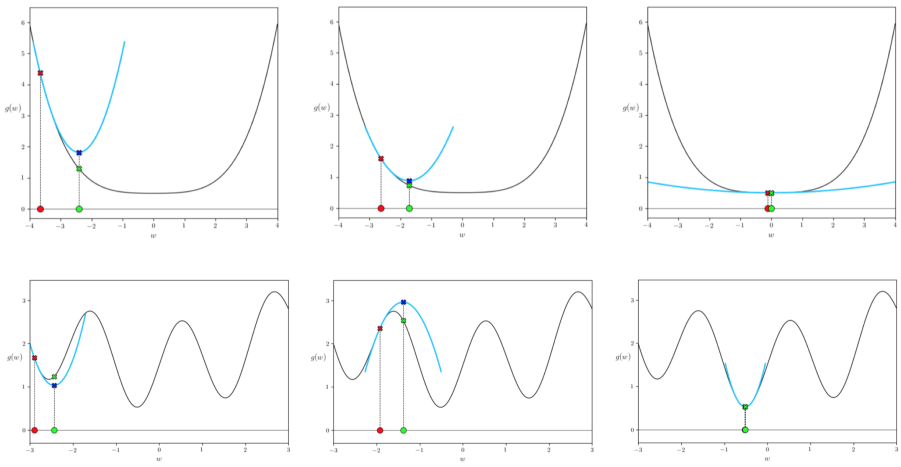

Figure 4.5 Figure associated with Example 4.4 (top row) and Example 4.5 bottom row. See text for details.

upward (as described in the previous Section). Therefore its stationary point is a global minimum (of the quadratic).

Notice importantly that the minimum of the quadratic approximation $w^\star$ always leads to a lower point of the function than where we begin at v.

Example 4.5 Stationary point of approximating quadratics II

Below we illustrate the same concept discussed in the prior example only with the non-convex function

$$g(w) = \sin(3w) + 0.1w^2 + 1.5. \tag{4.10}$$

In the bottom row of Figure 4.5 we show similar panels as those described in the previous Example for this function. Here however the situation is clearly different, with non-convexity being the culprit. In particular at concave portions of the function (like the one shown in the middle panel) - since here the quadratic is also concave - the stationary point $w^\star$ of the quadratic approximation is a global maximum of the approximator and tends to lead towards points that *increase* the value of the function (not *decrease* it).

From our cursory investigation of these two simple examples we can intuit that an idea for a local optimization scheme: repeatedly traveling to points defined by the stationary point of the second order Taylor series approximation. For convex functions, where each quadratic approximation's stationary point seems to lead to lower portions value of a function, this idea could provide an

efficient way of minimizing a cost function. This is indeed the case, and the resulting idea is called the *Newton's method* algorithm.

4.3.2 Newton's method

Newton's method is a local optimization algorithm produced by repeatedly taking steps to stationary points of the second order Taylor series approximations of a function. At the k^{th} step of this process for a single input function we make a second order Taylor series approximation centered at the point w^{k-1}

$$h(w) = g(w^{k-1}) + \left(\frac{\mathrm{d}}{\mathrm{d}w}g(w^{k-1})\right)(w - w^{k-1}) + \frac{1}{2}\left(\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(v)\right)(w - w^{k-1})^2 \tag{4.11}$$

and solve for its stationary point to create the update w^k as

$$w^k = w^{k-1} - \frac{\frac{\mathrm{d}}{\mathrm{d}w}g(w^{k-1})}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w^{k-1})}. \tag{4.12}$$

With the analog for N dimensional input at the k^{th} step we form the second order quadratic approximation

$$h(\mathbf{w}) = g(\mathbf{w}^{k-1}) + \nabla g(\mathbf{w}^{k-1})^T(\mathbf{w} - \mathbf{w}^{k-1}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^{k-1})^T\nabla^2 g\left(\mathbf{w}^{k-1}\right)(\mathbf{w} - \mathbf{w}^{k-1}) \tag{4.13}$$

and solve for a stationary point of this approximator giving the update $\mathbf{w}^k$ as

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \left(\nabla^2 g(\mathbf{w}^{k-1})\right)^{-1}\nabla g(\mathbf{w}^{k-1}). \tag{4.14}$$

This is a local optimization scheme that fits right in with the general form we have seen in the previous two Chapters

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha\mathbf{d}^k \tag{4.15}$$

where in the case of Newton's method the direction $\mathbf{d}^k = -\left(\nabla^2 g(\mathbf{w}^{k-1})\right)^{-1}\nabla g(\mathbf{w}^{k-1})$ and $\alpha = 1$. Here the fact that the steplength parameter α is implicitly set to 1 follows naturally from the derivation we have seen.

Note that although in the formula for the stationary point in Equation 4.3.2 we invert the Hessian - an $N \times N$ matrix, where N is the dimension of the input - in practice this is computationally expensive to implement. Instead $\mathbf{w}^k$ is typically found via solving the equivalent symmetric system of equations

$$\nabla^2 g\left(\mathbf{w}^{k-1}\right)\mathbf{w} = \nabla^2 g\left(\mathbf{w}^{k-1}\right)\mathbf{w}^{k-1} - \nabla g(\mathbf{w}^{k-1}) \tag{4.16}$$

since this is a more efficient calculation. [2]

As illustrated in the top panel of Figure 4.6 - where these steps are drawn for a single-input function - starting at an initial point $\mathbf{w}^0$ Newton's method produces a sequence of points $\mathbf{w}^1$, $\mathbf{w}^2$, ... that minimize g by repeatedly creating the second order Taylor series quadratic approximation to the function, and traveling to a stationary point of this quadratic. 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. However this reliance on quadratic information also makes Newton's method naturally more difficult to use with non-convex functions since at concave portions of such a function the algorithm can climb to a local maximum, as illustrated in the bottom panel of Figure 4.6, or oscillate out of control.

Example 4.6 **A single-input convex function**

In Figure 4.7 we show the process of performing Newton's method to minimize the function

$$g(w) = \frac{1}{50}\left(w^4 + w^2 + 10w\right) + 0.5 \tag{4.17}$$

beginning at the point $w^0 = 2.5$ marked as a green dot (in the top-left panel) and corresponding evaluation of the function marked as green 'x'. The top-right panel of the Figure shows the first Newton step, with the corresponding quadratic approximation shown in green and its minimum shown as a magenta circle along with the evaluation of this minimum on the quadratic shown as a blue 'x'. In the lower-left panel shows the next iteration of Newton's method with the quadratic approximation in yellow, and the bottom-right panel the iteration following (with its quadratic approximation shown in red).

Example 4.7 **Comparison to gradient descent**

In Figure 4.8 we apply a single Newton step to completely minimize the convex quadratic function

$$g(w_1, w_2) = 0.26(w_1^2 + w_2^2) - 0.48 w_1 w_2. \tag{4.18}$$

This can be done with a single step because the second order Taylor series - being a quadratic (local) approximation to a function - of a quadratic function is

[2] One can solve this system using coordinate descent as outlined in Section 3.2.2. When more than one solution exists (e.g., if we are in a long narrow half-pipe of a convex function) the *smallest possible solution* is taken - this is typically referred to as the *pseudo-inverse*.

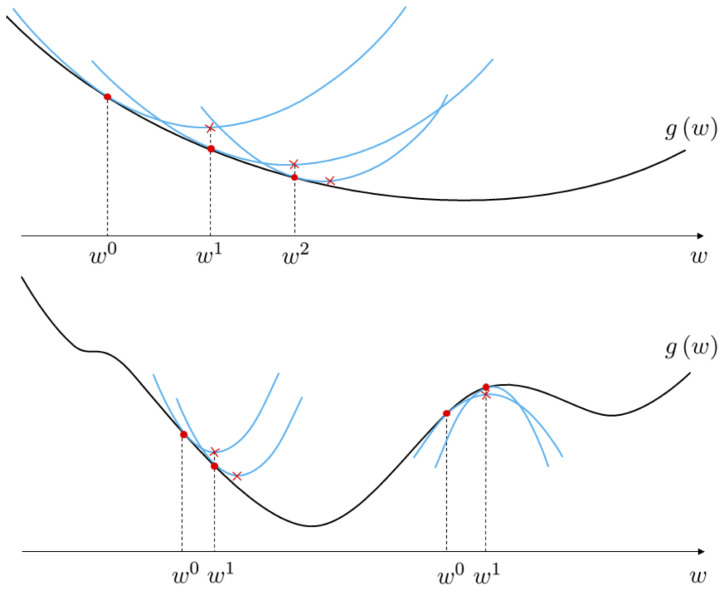

Figure 4.6 Newton's method illustrated. To find a minimum of g Newton's method hops down the stationary points of quadratic approximations generated by g's second order Taylor series. (top panel) For convex functions these quadratic approximations are themselves always convex (upward facing) and so their 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.

simply the quadratic function itself. Thus Newtons method reduces to solving the first order system of a quadratic function (a linear system of equations - described in Example 3.4). We compare the result of this single Newton step - shown in the right panel of the Figure - to a corresponding a run of 100 steps of gradient descent in the left panel of the Figure.

4.3.3 Ensuring numerical stability

Near flat portions of a function both the numerator and denominator of the single-input Newton update in Equation 4.3.2, that is $\frac{\mathrm{d}}{\mathrm{d}w}g(w^{k-1})$ and $\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w^{k-1})$, are both nearly zero valued. This can cause serious numerical problems since once each (but especially the denominator) shrinks below 'machine precision' (that is, the smallest value a computer can interpret as being non-zero) a machine will interpret $\frac{\frac{\mathrm{d}}{\mathrm{d}w}g(w^{k-1})}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w^{k-1})} \approx \frac{0}{0}$.

One simple and common way to avoid this potential disaster is to simply add a small positive value ϵ to the second derivative - either when it shrinks below

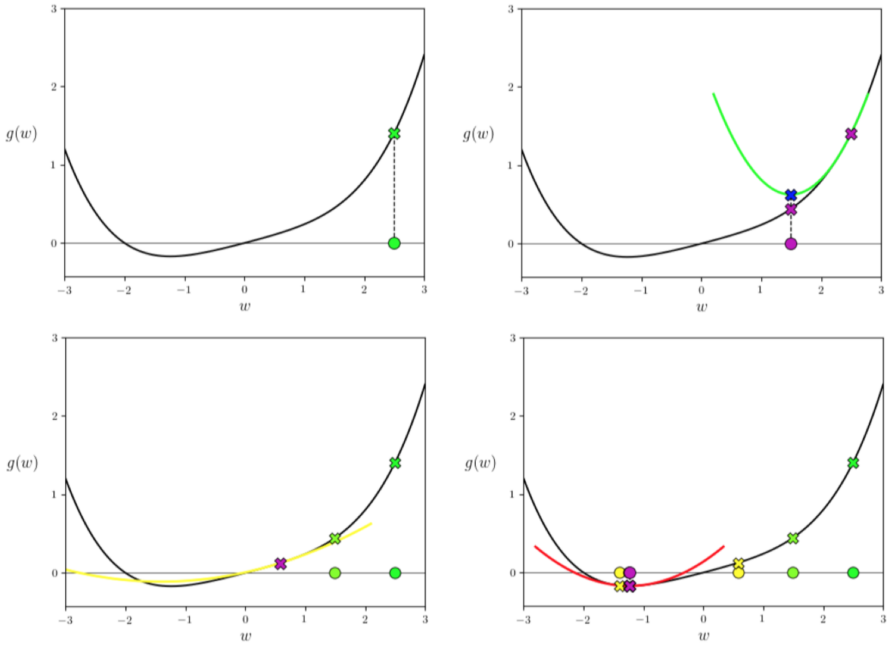

Figure 4.7 Figure associated with Example 4.6 animating a run of Newton's method applied to a single input function. See text for further details.

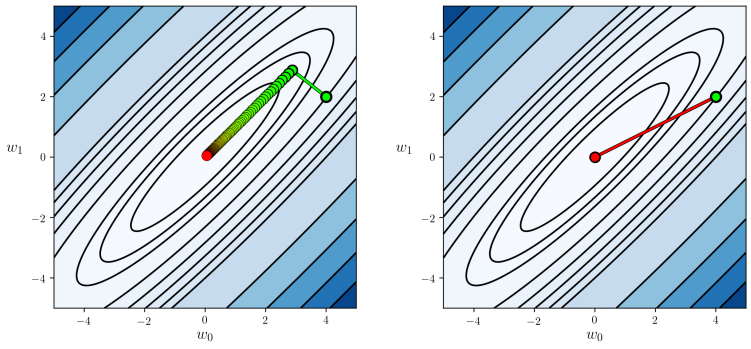

Figure 4.8 Figure for Example 4.7. A run of gradient descent (left panel) compared with the single Newton step required to perfectly minimize a convex quadratic function.

a certain value or for all iterations. This *regularized* Newton's step looks like the following

$$w^k = w^{k-1} - \frac{\frac{\mathrm{d}}{\mathrm{d}w} g(w^{k-1})}{\frac{\mathrm{d}^2}{\mathrm{d}w^2} g(w^{k-1}) + \epsilon} \tag{4.19}$$

The value of the *regularization parameter* ϵ is set by hand to a small positive

value (e.g., 10^{-7}). Note this is an adjustment made when the function being minimized is known to be convex, since in this case $\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w) \geq 0$ for all w.

The analogous adjustment for the multi-input Newtons's method update in Equation 4.3.2 is to add $\epsilon \mathbf{I}_{N\times N}$ - an $N \times N$ identity matrix scaled by a small positive ϵ value - to the second derivative Hessian matrix. This gives the completely analogous regularized Newton's step update for the multi-input case

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \left(\nabla^2 g(\mathbf{w}^{k-1}) + \epsilon \mathbf{I}_{N\times N}\right)^{-1} \nabla g(\mathbf{w}^{k-1}). \tag{4.20}$$

Adding this to the Hessian means that the matrix $\left(\nabla^2 g(\mathbf{w}^{k-1}) + \epsilon \mathbf{I}_{N\times N}\right)$ is always invertible provided a large enough value for ϵ is used. [3]

4.3.4 Steplength choices

While we have seen in the derivation of Newton's method that - being a local optimization approach - it does have an steplength parameter α it is implicitly set to $\alpha = 1$ and so appears 'invisible'. However note that one can explicitly introduce a steplength parameter α and - in principle - use adjustable methods, e.g., backtracking line search (see Section 3.10) in order to tune α. Adding a steplength parameter α this explicitly weighted Newton step looks like

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left(\nabla^2 g(\mathbf{w}^{k-1})\right)^{-1} \nabla g(\mathbf{w}^{k-1}) \tag{4.21}$$

with the standard Newton's method step falling out when $\alpha = 1$.

4.3.5 Python implementation

Below we provide a simple implementation of Newton's method in Python, leveraging the excellent `autograd` automatic differentiation (see Sections 3.5 and 7.6) and numpy libraries. In particular we employ the `grad` and `hess` modules from `autograd` to compute the first and second derivatives of a general input function automatically.

```
# import autograd's automatic differentiator
from autograd import grad
from autograd import hessian

# newtons method function
def newtons_method(g,max_its,w,**kwargs):
    # compute gradient / hessian using autograd
    gradient = grad(g)
    hess = hessian(g)
```

[3] As with the original Newton step in Equation 4.3.2 it is virtually always more numerically efficient to compute this update by solving the associated linear system $\left(\nabla^2 g(\mathbf{w}^{k-1}) + \epsilon \mathbf{I}_{N\times N}\right)\mathbf{w} = \left(\nabla^2 g(\mathbf{w}^{k-1}) + \epsilon \mathbf{I}_{N\times N}\right)\mathbf{w}^{k-1} - \nabla g(\mathbf{w}^{k-1})$ for $\mathbf{w}$.

```
    # set numerical stability parameter / regularization parameter
    epsilon = 10**(-7)
    if 'epsilon' in kwargs:
        beta = kwargs['epsilon']

    # run the newtons method loop
    weight_history = [w] # container for weight history
    cost_history = [g(w)]# container for cost function history
    for k in range(max_its):
        # evaluate the gradient and hessian
        grad_eval = gradient(w)
        hess_eval = hess(w)

        # reshape hessian to square matrix for numpy linalg
            functionality
        hess_eval.shape = (int((np.size(hess_eval))**(0.5)),int((np.
            size(hess_eval))**(0.5)))

        # solve second order system system for weight update
        A = hess_eval + epsilon*np.eye(w.size)
        b = grad_eval
        w = np.linalg.solve(A,np.dot(A,w) - b)

        # record weight and cost
        weight_history.append(w)
        cost_history.append(g(w))
    return weight_history,cost_history
```

Note: while we used a maximum iterations convergence criterion the potentially high computational cost of each Newton step often incentivizes the use of more formal convergence criterion (e.g., halting when the norm of the gradient falls below a pre-defined threshold). This also often incentivizes the inclusion of checkpoints that measure and/or adjust the progress of a Newton's method run in order to avoid problems near flat areas of a function.

4.4 Two fundamental problems with Newton's method

Newton's method is a powerful algorithm that makes enormous progress towards at each step (unlike e.g., zero and first order methods that can require a huge number of steps to make equal progress). However Newton's method suffers from two fundamental problems that limit its popularity in certain fields of machine learning, which we discuss here. These problems involve its application to minimizing non-convex functions, and issues scaling with input dimension.

4.4.1 Minimizing non-convex functions

As discussed in the previous Section, Newton's method can behave very badly when applied to minimizing non-convex functions. Since each step is based off of the second order approximation to a function if the curvature at a point is *concave* Newton's method will naturally take a *step uphill*.

On the other hand because the second order approximation used by Newton's method contains so much local information than when applied to convex functions it can converge to a global minimum in far fewer steps - particularly when it is close to a minimum (since often the quadratic approximation matches a convex functions locally very well).

4.4.2 Scaling limitations

A Newton's method step requires far more in terms of storage and computation than a first order step - requiring the storage and computation of not just a gradient but an entire $N \times N$ Hessian matrix of second derivative information as well. Simply storing the Hessian for a single step of Newton's method, with its N^2 entries, can quickly become challenging for what moderately sized input. For example if the input to a function has dimension $N = 10,000$ the corresponding $10,000 \times 10,000$ Hessian matrix has $10,000^2 = 100,000,000$ entries. The kind of functions used in machine learning applications can easily have tens of thousands to hundreds of thousands or even hundreds of millions of inputs, making the complete storage of an associated Hessian impossible.

4.5 Newton's method, regularization, and non-convex functions

As we saw in Section 4.3 Newton's method is naturally incapable of properly minimizing generic non-convex functions. In this Section we describe a common approach to ameliorate this particular issue which is conceptually simple, we have essentially already seen it before (albeit without the more in depth context we provide here): the regularized Newton step detailed in the Section 4.3.3.

4.5.1 Turning up ϵ

In Section 4.3.3 we saw how adding a very small positive value ϵ to the second derivative of a single-input function, or analogously a weighted identity matrix of the form $\epsilon \mathbf{I}_{N\times N}$ to the Hessian in the multi-input case, helps Newton's method avoid numerical problems in flat regions of a convex function. The general adjusted Newton step took

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \left(\nabla^2 g(\mathbf{w}^{k-1}) + \epsilon \mathbf{I}_{N\times N}\right)^{-1} \nabla g(\mathbf{w}^{k-1}) \tag{4.22}$$

We can interpret this as the stationary point of a slightly adjusted second order Taylor series approximation centered at $\mathbf{w}^{k-1}$. This approximation takes the related form

$$h(\mathbf{w}) = g(\mathbf{w}^{k-1}) + \nabla g(\mathbf{w}^{k-1})^T(\mathbf{w} - \mathbf{w}^{k-1}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^{k-1})^T\nabla^2 g\left(\mathbf{w}^{k-1}\right)(\mathbf{w} - \mathbf{w}^{k-1}) + \frac{\epsilon}{2}\left\|\mathbf{w} - \mathbf{w}^{k-1}\right\|_2^2 \tag{4.23}$$

and the fact that our adjusted Newton step is a stationary point of it can be easily checked via the first order condition.

What do we have here? The first three terms (on the right hand side of the equality) still represent the second order Taylor series at $\mathbf{w}^{k-1}$, and to it we have added $\frac{\epsilon}{2}\left\|\mathbf{w} - \mathbf{w}^{k-1}\right\|_2^2$, a *convex* and perfectly symmetric quadratic centered at $\mathbf{w}^{k-1}$ with N positive eigenvalues (each equal to $\frac{\epsilon}{2}$). In other words: we have a sum of two quadratic functions. When $\mathbf{w}^{k-1}$ is at a non-convex or flat portion of a function the first quadratic - the second order Taylor series - is likewise non-convex or flat: however the second one is *always* convex, and the larger ϵ is set the greater its (convex) curvature. This means that if we set ϵ larger we can *convexify the entire approximation*, forcing the stationary point we solve for to be a minimum and the direction in which we travel is one of guaranteed descent.

Example 4.8 **Regularizing a non-convex quadratic**

In Figure 4.9 we illustrate the regularization of a non-convex function

$$h_1(w_1, w_2) = w_1^2 - w_2^2 \tag{4.24}$$

using the convex regularizer

$$h_2(w_1, w_2) = w_1^2 + w_2^2 \tag{4.25}$$

In particular we show what the resulting sum $h_1 + \epsilon h_2$ looks like over four progressively increasing values of ϵ from $\epsilon = 0$ (left panel) to $\epsilon = 2$ (right panel).

Since h_1 is non-convex, and has a single stationary point with is a saddle point at the origin, the addition of h_2 pulls up its downward facing dimension. Not surprisingly as ϵ is increased the shape of the sum is dictated more and more by h_2. Eventually, turning up ϵ sufficiently, the sum becomes convex.

Example 4.9 **Regularized Newton's method**

In the left column of Figure 4.10 we illustrate five regularized Newton steps (using the update step shown in Equation 4.5.1) to minimize the non-convex function

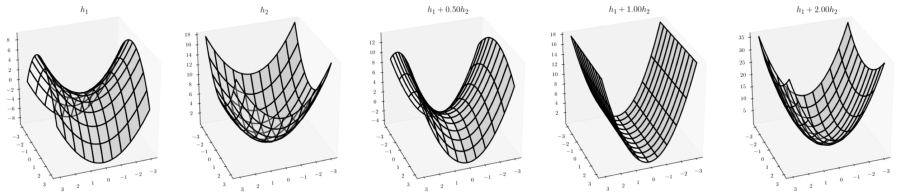

Figure 4.9 Figure associated with Example 4.8. A non-convex quadratic function (left panel) is slowly turned into a convex function via the weighted addition of a convex quadratic (right panel). See text for further details.

$$g(w) = 2 - e^{-w^2} \tag{4.26}$$

We initialize the algorithm at a point of local non-convexity for this function, and gradually increase ϵ from $\epsilon = 0$ (top panel) where Newton's method diverges to $\epsilon = 4$ where it has sufficiently convexified the Newton step leading the algorithm to take steps downward towards the minimum of the function. In the Figure the steps from each run are colored from green (the first step) to red (the final step), as well as the regularized second order approximation at each step (each colored like its respective step). In the right column we show the cost function history plot associated with each associated run.

How high do we need to turn up ϵ in the regularized second order Taylor series approximation (when centered at a non-convex point) in order to make it convex? We know from Section 4.1 that a quadratic function is convex if and only if it has all non-negative eigenvalues. With this in mind a simple analysis of the regularized quadratic quickly reveals the answer: ϵ must be made larger then the magnitude of the smallest eigenvalue of the Hessian $\nabla^2 g\left(\mathbf{w}^{k-1}\right)$ in order for the regularized second order quadratic to be convex. For a single input function this reduces to ϵ being larger in magnitude than value of the function's second derivative at w^{k-1} if it is negative there.

4.5.2 From Newton's method to gradient descent

When we turn up ϵ to a large number the direction we travel actually becomes the gradient descent direction (Section 3.6) at $\mathbf{w}^{k-1}$. We can see this by analyzing the descent direction provided by the regularized Newton step in Equation 4.5.1, in particular it can be shown (see Exercises) that when ϵ is large

$$-\left(\nabla^2 g(\mathbf{w}^{k-1}) + \epsilon \mathbf{I}_{N\times N}\right)^{-1} \nabla g(\mathbf{w}^{k-1}) \approx -\left(\epsilon \mathbf{I}_{N\times N}\right)^{-1} \nabla g(\mathbf{w}^{k-1}) = -\frac{1}{\epsilon} \nabla g(\mathbf{w}^{k-1}) \tag{4.27}$$

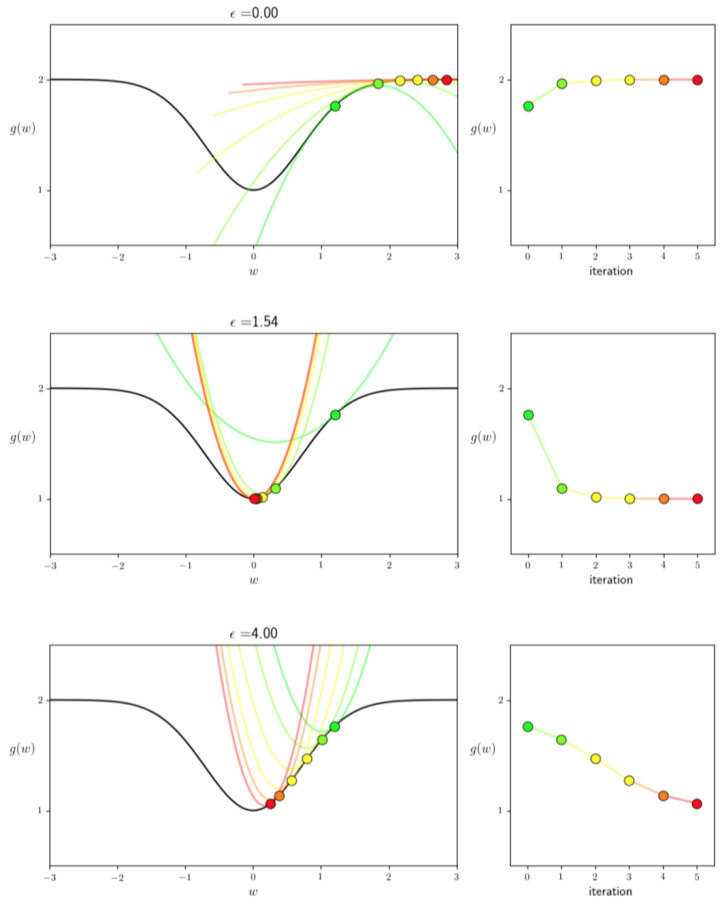

Figure 4.10 Figure associated with Example 4.9. Here Newton's method is initialized at a point of non-convexity of the function shown in the left column. Since this initialization lies at a point of concavity the algorithm climbs to a maximum (top-right panel). As ϵ is increased the algorithm is properly convexified, leading it to determine a proper minimum of the function (bottom-left panel). See text for further details.

So, in other words, when ϵ is large the direction we travel when taking the regularized Newton step becomes the gradient descent direction, albeit with a very small magnitude.

4.6 Exercises

Section 4.1 exercises

1. **Determining the eigenvalues of a symmetric matrix**

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

a) Let $\mathbf{C}$ be an $N \times N$ symmetric matrix. Show that if $\mathbf{C}$ has all non-negative eigenvalues then the quantity $\mathbf{z}^T\mathbf{C}\mathbf{z} \geq 0$ for all $\mathbf{z}$. *Hint*: Use the eigenvalue decomposition of $\mathbf{C}$ (see Section 7.10).

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

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

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

2. **Outer product matrices have all non-negative eigenvalues**

a) Use the method described in exercise 1 to verify that for any N length vector $\mathbf{x}$ the $N \times N$ outer product matrix $\mathbf{x}\mathbf{x}^T$ has all non-negative 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.

Section 4.2 exercises

3. **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 non-negative 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 1 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 1 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 non-negative 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 $g(\mathbf{w}) = a + \mathbf{b}^T\mathbf{w} + \mathbf{w}^T\mathbf{C}\mathbf{w}$, where $\mathbf{A}$ is symmetric and known to have all non-negative eigenvalues and a and $\mathbf{b}$ 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 4.3 exercises

4. **Newtons method I**

Repeat the experiment described in Example 4.6. Instead of plotting the resulting path taken by Newton's method (as shown in Figure 4.7) create a cost function history plot to ensure your algorithm properly converges to a point near the global minimum of the function. You may employ the implementation of Newton's method described in Section 4.3.5 as a base for this Exercise.

5. **Newtons method II**

a) Use the first order condition (see Section 3.2 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.*

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 cost function history plot for ten iterations of Newton's method in order to verify that your algorithm works properly and is converging. You may use

the implementation given in Section 4.3.5 as a base for this part of the Exercise.

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 $\mathbf{g}(\mathbf{w})$ 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?

6. **Newton's method and square roots**

Use Newton's method to compute a square root of the number: 999. Briefly explain how you set up the relevant cost function that was minimized to obtain this square root. Explain how you use zero or first order optimization methods (detailed in Chapters 2 and 3) to do this as well.

Section 4.5 exercises

7. **Non-convex minimization using Newton's method**

Use (regularized) Newton's method to minimize

$$g(w) = \cos(w) \tag{4.28}$$

beginning at $w = 0.1$. In particular make sure you achieve decrease at *every* step of Newton's method.

8. **Show that regularized Newtons method decreases**

Rigorously verify that when ϵ is set large enough to convexify the regularized second order approximation that the corresponding Newton step is indeed a descent direction. That is, regardless of the function g being minimized that ϵ can be set large enough so that a corresponding Newton step can lead to a lower portion of the function i.e., $g\left(\mathbf{w}^k\right) \leq g\left(\mathbf{w}^{k-1}\right)$.