Chapter 2

Basic Principles of Optimization

Much of machine learning deals with learning functions by finding the optimal function according to an objective. For example, one may be interested in finding a function that minimizes the squared differences to some targets for all the examples; i.e., $\sum_{i=1}^{n}(f(\mathbf{x}_i)-y_i)^2$. To find such a function, we need to have a basic grasp of optimization techniques.

In this chapter, we discuss basic optimization tools for generic smooth objectives. Many of the algorithms in machine learning rely on a simple approach called gradient descent. We will this first discuss how to minimize objectives using both first and second-order gradient descent. This overview covers only a small part of optimization, but fortunately, many machine learning algorithms are based on these simple approaches.

2.1 The basic optimization problem and stationary points

A basic optimization goal is to select a set of parameters $\boldsymbol{w} \in \mathbb{R}^d$ to minimize a given objective function $c : \mathbb{R}^d \to \mathbb{R}$

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} c(\boldsymbol{w})$$

For example, to obtain the parameters $\boldsymbol{w}$ for linear regression that minimizes the squared differences, we use $c(\boldsymbol{w}) = \sum_{i=1}^{n}(\langle \mathbf{x}_i, \boldsymbol{w}\rangle - y_i)^2$, for dot product

$$\langle \boldsymbol{x}_i, \boldsymbol{w} \rangle = \sum_{j=1}^{d} x_{ij}, w_j.$$

We use the term objective here, rather than error, since error has an explicit connotation that the function is inaccurate. Later we will see that objectives will include both error terms—indicating how accurately they recreate data—as well as terms the provide other preferences on the function. Combining these terms with the error produces the final objective we would like to minimize. For example, for linear regression, we will optimize a regularized objective, $c(\boldsymbol{w}) = \sum_{i=1}^{n}(\langle \boldsymbol{x}_i, \boldsymbol{w}\rangle - y_i)^2 + \sum_{j=1}^{d} w_j^2$, where the second term encodes a preference for smaller coefficients w_j.

The goal then is to find $\boldsymbol{w}$ that minimizes the objective. The most straightforward, naive solution could be to do a random search; i.e., generate random $\boldsymbol{w}$ and check $c(\boldsymbol{w})$. If any newly generated $\boldsymbol{w}_t$ on iteration t outperforms the previous best solution $\boldsymbol{w}$, in that $c(\boldsymbol{w}_t) < c(\boldsymbol{w})$, then we can set $\boldsymbol{w}_t$ to be the new optimal solution. We will assume that our objectives are continuous, and so can take advantage of this smoothness to design better search strategies. In particular, for smooth functions, we will be able to use gradient descent, which we describe in the next section.

Gradient descent enables us to reach stationary points, points $\boldsymbol{w}$ where the gradient is zero. Consider first the univariate case. The derivative tells us the rate of change of the

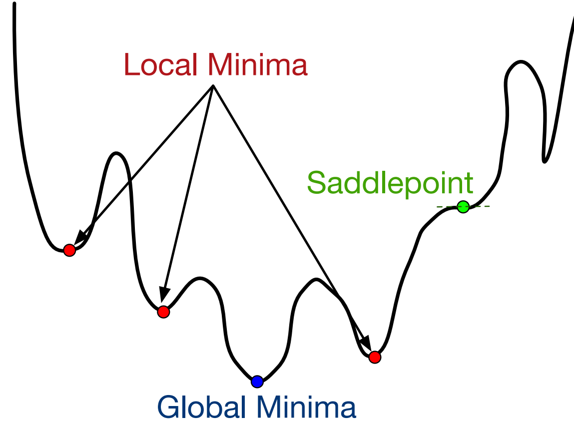

Figure 2.1: Stationary points on a smooth function surface: local minima, global minima and saddle points.

function surface at a point w. When the derivative of the objective is zero at $w \in \mathbb{R}$; i.e., $\frac{d}{dw}c(w) = 0$. This means that locally the function surface is flat. Such points correspond to local minima, local maxima and saddle points, as shown in Figure 2.1. For example, assume again that we are doing linear regression, with only one feature and so only one weight $w \in \mathbb{R}$. The derivative of the objective $c(w) = \sum_{i=1}^{n}(x_i w - y_i)^2$ is

$$\begin{aligned}
\frac{d}{dw}c(w) &= \frac{d}{dw}\sum_{i=1}^{n}(x_i w - y_i)^2 \\
&= \sum_{i=1}^{n}\frac{d}{dw}(x_i w - y_i)^2 \\
&= \sum_{i=1}^{n}2(x_i w - y_i)x_i
\end{aligned}$$

where the last step follows from the chain rule. Our goal is to find w such that $\frac{d}{dw}c(w) = 0$; once we find such a stationary point, we can then determine if it is a local minimum, local maximum or saddle point. Because this objective is convex, we in fact know that all stationary points must be global minima, and so we would not need to do this check. We discuss this further in the last section, where we discuss some properties of objectives.

For the multivariate case, we need to consider gradients instead of derivatives. For $\boldsymbol{w} \in \mathbb{R}^d$ where $d > 1$, we need to ask: how does the function change locally, depending on how each element of $\boldsymbol{w}$ is changed? To quantify this, we use the gradient which is composed of partial derivatives

$$\nabla c(\boldsymbol{w}) = \left[\frac{\partial c}{\partial w_1}(\boldsymbol{w}) \quad \frac{\partial c}{\partial w_2}(\boldsymbol{w}) \quad \ldots \quad \frac{\partial c}{\partial w_d}(\boldsymbol{w})\right].$$

Each partial derivative $\frac{\partial c(\boldsymbol{w})}{\partial w_j}$ represents how the function c changes, when only w_j is changed and the other $w_1, \ldots, w_{j-1}, w_{j+1}, \ldots, w_d$ are kept constant. For example, for

$c(\boldsymbol{w} = (w_1, w_2)) = \frac{1}{2}(x_1 w_1 + x_2 w_2 - y)^2$, the partial derivatives are

$$\frac{\partial c}{\partial w_1}(\boldsymbol{w}) = (x_1 w_1 + x_2 w_2 - y)x_1$$
$$\frac{\partial c}{\partial w_2}(\boldsymbol{w}) = (x_1 w_1 + x_2 w_2 - y)x_2$$

Usefully, we do not have to consider how the whole vector changes jointly in all the variables. Rather, it is sufficient to find stationary points by finding $\boldsymbol{w}$ where the partial derivatives are zero.

2.2 Gradient descent

The key idea behind gradient descent is to approximate the function with a Taylor series approximation. This approximation facilitates computation of a descent direction locally on the function surface. We begin by considering the univariate setting, with $w \in \mathbb{R}$. A function $c(w)$ in the neighborhood of point w_0, can be approximated using the Taylor series as

$$c(w) = \sum_{n=0}^{\infty} \frac{c^{(n)}(w_0)}{n!}(w - w_0)^n,$$

where $c^{(n)}(w_0)$ is the n-th derivative of function $c(w)$ evaluated at point w_0. This assumes that $c(w)$ is be infinitely differentiable, but in practice we will take such polynomial approximations for a finite n. A *second-order* approximation to this function uses the first three terms of the series as

$$c(w) \approx \hat{c}(w) = c(w_0) + (w - w_0)c'(w_0) + \frac{1}{2}(w - w_0)^2 c''(w_0).$$

A stationary point of this $\hat{c}(w)$ can be easily found by finding the first derivative and setting it to zero

$$c'(w) \approx c'(w_0) + (w - w_0)c''(w_0) = 0.$$

Solving this equation for w gives

$$w_1 = w_0 - \frac{c'(w_0)}{c''(w_0)}.$$

Locally, this new w_1 will be an improvement on w_0, and will be a stationary point of this local approximation $\hat{c}$. Moving (far enough) from w_0, however, makes this local second-order Taylor series inaccurate. We would need to check the local approximation at this new point w_1, to determine if we can further improve locally. Therefore, to find the optimal w, we can iteratively apply this procedure

$$w_{t+1} = w_t - \frac{c'(w_t)}{c''(w_t)}. \tag{2.1}$$

constantly improving w_i until we reach a point where the derivative is zero, or nearly zero. This method is called the Newton-Raphson method, or second-order gradient descent. In

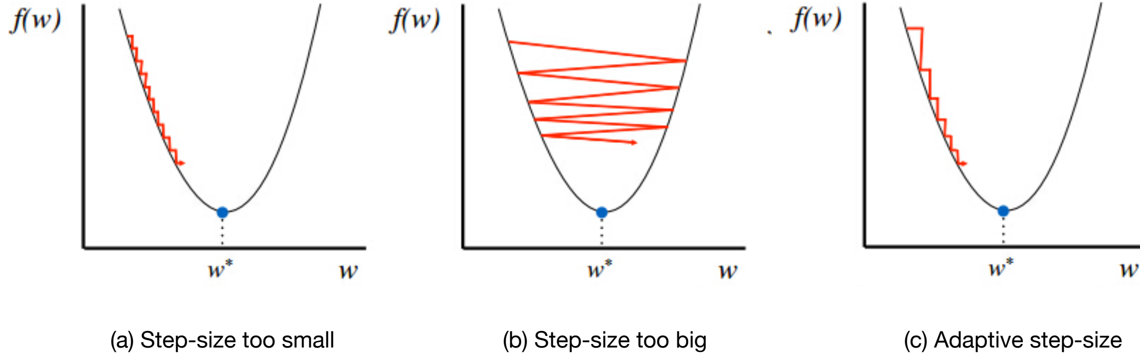

Figure 2.2: Different optimization paths, due to different stepsize choices.

first-order gradient descent, the approximation is worse, where we no longer use the second derivative. Instead, when taking a first-order approximation, we know that we are ignoring $O((w-w_0)^2)$ terms, and so the local approximation becomes

$$c(w) \approx \hat{c}(w) = c(w_0) + (w - w_0)c'(w_0) + \frac{1}{2\eta}(w - w_0)^2$$

for some constant $\frac{1}{\eta}$ reflecting the magnitude of the ignored $O((w-w_0)^2)$ terms. The resulting update is then, for step-size η_t

$$w_{t+1} = w_t - \eta_t c'(w_t). \tag{2.2}$$

From this, one can see that, given access to the second derivative, a reasonable choice for the stepsize is $\eta_t = \frac{1}{c''(w_t)}$.

We can similarly obtain such rules for multivariate variables. For example, gradient descent for $c : \mathbb{R}^d \to \mathbb{R}$ consists of the update

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta_t \nabla c(\boldsymbol{w}_t). \tag{2.3}$$

where

$$\nabla c(\boldsymbol{w}_t) = \left(\frac{\partial c}{\partial w_1}(\boldsymbol{w}_i), \frac{\partial c}{\partial w_2}(\boldsymbol{w}_t), ..., \frac{\partial c}{\partial w_d}(\boldsymbol{w}_t)\right) \qquad \in \mathbb{R}^d$$

is the gradient of function c evaluated at $\boldsymbol{w}_t$. We will discuss how to derive this update in the multivariable setting later in this Chapter.

2.3 Selecting the step-size

An important part of (first-order) gradient descent is to select the step-size. If the step-size is too small, then many iterations are required to reach a stationary point (Figure 2.2(a)); If the step-size is too large, then you are likely to oscillate around the minimum (Figure 2.2 (b)). What we really want is an adaptive step-size (Figure 2.2 (c)), that likely starts larger and then slowly reduces over time as a stationary point is approached.

The basic method to obtain adaptive step-sizes is to use *line search*. The idea springs from the following goal: we would like to obtain the optimal step-size according to

$$\min_{\eta \in \mathbb{R}^+} c(\boldsymbol{w}_t - \eta \nabla c(\boldsymbol{w}_t))$$

The solution to this optimization corresponds to the best scalar stepsize we could select, for the current point $\mathbf{w}_t$ with descent direction $-\nabla c(\mathbf{w}_t)$. Solving this optimization would be too expensive; however, we can find approximate solutions quickly. One natural choice is to use a backtracking line search, that tries the largest reasonable stepsize $\eta_{\max}$, and then reduces it until the objective is decreased. The idea is to search along the line of possible $\eta \in (0, \eta_{\max}]$, with the intuition that a large step is good—as long as it does not overshoot. If it does overshoot, then the stepsize was too large, and should be reduced. The reduction is typically according to the rule $\tau\eta$ for some $\tau \in [0.5, 0.9]$. For $\tau = 0.5$, the stepsize reduces more quickly—halves on each step of the backtracking line search; for $\tau = 0.9$, the search more slowly backtracks from $\eta_{\max}$. As soon as a stepsize is found that decreases the objective, it is accepted. We then obtain a new $\mathbf{w}_t$, again compute the gradient and start the line search once again from $\eta_{\max}$.

One can imagine better strategies for selecting the stepsize than this simplistic search; we will in fact discuss some of these in Section 3.5. Nonetheless, this basic line search—an improvement therein—remains a common approach for selecting the stepsize.

Algorithm 1: Line Search($\mathbf{w}_t, c, \mathbf{g} = \nabla c(\mathbf{w}_t)$)

1: Optimization parameters: $\eta_{\max} = 1.0, \tau = 0.7, \text{tolerance} \leftarrow 10e^{-4}$
2: $\eta \leftarrow \eta_{\max}$
3: $\mathbf{w} \leftarrow \mathbf{w}_t$
4: obj $\leftarrow c(\mathbf{w})$
5: **while** number of backtracking iterations is less than maximum iterations **do**
6: $\quad\mathbf{w} \leftarrow \mathbf{w}_t - \eta\mathbf{g}$
7: $\quad$// Ensure improvement is at least as much as tolerance
8: $\quad$**If** $c(\mathbf{w}) <$ obj - tolerance **then break**
9: $\quad$// Else, the objective is worse and so we decrease stepsize
10: $\quad\eta \leftarrow \tau\eta$
11: **if** maximum number of iterations reached **then**
12: $\quad$// Could not improve solution
13: $\quad$**return** $\mathbf{w}_t, \eta = 0$
14: **return** $\mathbf{w}, \eta$

2.4 Optimization properties

There are several optimization properties to keep in mind when reading this handbook, which we highlight here.

Maximizing versus minimizing We have so far discussed the goal of minimizing an objective; an equivalent alternative is to maximize the negative of this objective.

$$\underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}}\, c(\mathbf{w}) = \underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmax}} -c(\mathbf{w})$$

where argmin returns $\mathbf{w}$ that produces the minimum value of $c(\mathbf{w})$ and argmax returns $\mathbf{w}$ that produces the maximum value of $-c(\mathbf{w})$. The actual min and max values are not

the same, since for a given optimal solution, $c(\mathbf{w}) \neq -c(\mathbf{w})$. We opt to formulate each of our optimizations as a minimization, and do gradient descent. It would be equally valid, however, to formulate the optimizations as maximizations, and do gradient ascent.

Convexity A function $c : \mathbb{R}^d \rightarrow \mathbb{R}$ is said to be convex if for any $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d$ and $t \in [0, 1]$,

$$c(t\mathbf{w}_1 + (1-t)\mathbf{w}_2) \leq tc(\mathbf{w}_1) + (1-t)c(\mathbf{w}_2) \tag{2.4}$$

This definition means that when we draw a line between any two points on the function surface, the function values between these two points all lie below this line. Convexity is an important property, because it means that every stationary point is a global minimum. Therefore, regardless of where we start our gradient descent, with appropriately chosen stepsize and sufficient iterations, we will reach an optimal solution.

A corresponding definition is a concave function, which is precisely the opposite: all points lie above the line. For any convex function c, the negative of that function $-c$ is a concave function.

Uniqueness of the solution We often care if there is more than one solution to our optimization problem. In some cases, we care about *identifiability*, which means we can identify the true solution. If there is more than one solution, one might consider that the problem is not precisely posed. For some problems, it is important or even necessary to have identifiability (e.g., estimating the percentage of people with a disease) whereas for other we simply care about finding a suitable (predictive) function f that reasonably accurately predicts the targets, even if it is not the unique such function. We will not consider identifiability further in this document, but it is important to be cognizant of if your objective has multiple solutions.

Equivalence under a constant shift Adding or multiplying by a constant $a \neq 0$ does not change the solution

$$\operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} c(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} a\ c(\mathbf{w}) = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^d} c(\mathbf{w}) + a.$$

You can see why by taking the gradient of all three objectives and noticing that the gradient is zero under the same conditions

$$\nabla a\ c(\mathbf{w}) = 0 \iff a\nabla c(\mathbf{w}) = 0 \iff \nabla c(\mathbf{w}) = 0$$

and

$$\nabla(c(\mathbf{w}) + a) = 0 \iff \nabla c(\mathbf{w}) = 0.$$

Chapter 3

More advanced optimization principles

Given the optimization background in Chapter 2, and seeing how it is useful in the following chapters, we can now turn to more advanced optimization approaches. We will now discuss more in-depth how we obtain the second-order gradient descent update for the multivariate case. We then discuss some computational improvements on these methods, particularly through the use of improved step-size selection techniques, by using stochastic gradient descent and some small modifications to deal with non-differentiable points. Finally, we will also provide some basics on constrained optimization. When moving to the multivariate case, it will be useful to get used to multivariate calculus. We provide some basic rules in Section A.1; a more complete reference for these rules can be found in the (highly useful) matrix cookbook [15].

3.1 Multivariate gradient descent

We can generalize the discussion on obtaining the gradient descent update in Section 2.2 from the univariate case to the multivariate case using the multivariate Taylor series approximation. The second-order Taylor approximation for a real-valued function of multiple variables can be written as

$$c(\mathbf{w}) \approx \hat{c}(\mathbf{w}) = c(\mathbf{w}_0) + \nabla c(\mathbf{w}_0)^\top (\mathbf{w} - \mathbf{w}_0) + \frac{1}{2} (\mathbf{w} - \mathbf{w}_0)^\top \mathbf{H}_{c(\mathbf{w}_0)} (\mathbf{w} - \mathbf{w}_0),$$

where

$$\nabla c(\mathbf{w}_0) = \left(\frac{\partial c}{\partial w_1}(\mathbf{w}_0), \frac{\partial c}{\partial w_2}(\mathbf{w}_0), ..., \frac{\partial c}{\partial w_d}(\mathbf{w}_0) \right) \qquad \in \mathbb{R}^d$$

is the gradient of function c evaluated at $\mathbf{w}_0$ and

$$\mathbf{H}_{c(\mathbf{w}_0)} = \begin{bmatrix} \frac{\partial^2 c}{\partial w_1^2}(\mathbf{w}_0) & \frac{\partial^2 c}{\partial w_1 \partial w_2}(\mathbf{w}_0) & \cdots & \frac{\partial^2 c}{\partial w_1 \partial w_d}(\mathbf{w}_0) \\ \frac{\partial^2 c}{\partial w_2 \partial w_1}(\mathbf{w}_0) & \frac{\partial^2 c}{\partial w_2^2}(\mathbf{w}_0) & & \\ \vdots & \vdots & \ddots & \\ \frac{\partial^2 c}{\partial w_d \partial w_1}(\mathbf{w}_0) & \cdots & & \frac{\partial^2 c}{\partial w_d^2}(\mathbf{w}_0) \end{bmatrix} \qquad \in \mathbb{R}^{d \times d}$$

is the Hessian matrix of function c evaluated at $\mathbf{w}_0$. We provide some intuition for the Hessian in the next section, but here it can be intuitively considered analogous to the second derivative. Like the second derivative, it provides information about the curvature of the function, and so provides useful information about how much to step in the direction of the gradient for each w_i.

As a reminder about matrix-vector multiplication, the product of a $d \times d$ matrix $\mathbf{H}$ and $d \times 1$ vector $\mathbf{w}$ is a $d \times 1$ vector $\mathbf{Hw}$. Then, taking $\mathbf{w}^\top \mathbf{Hw}$ is the dot product between a $1 \times d$ vector $\mathbf{w}^\top$ and $d \times 1$ vector $\mathbf{Hw}$, resulting in a scalar. For matrix-vector multiplication,

$$\mathbf{Hw} = \begin{bmatrix} \mathbf{H}_{1:} \\ \mathbf{H}_{2:} \\ \vdots \\ \mathbf{H}_{d:} \end{bmatrix} \mathbf{w} = \begin{bmatrix} \mathbf{H}_{1:}\mathbf{w} \\ \mathbf{H}_{2:}\mathbf{w} \\ \vdots \\ \mathbf{H}_{d:}\mathbf{w} \end{bmatrix} = \begin{bmatrix} \langle \mathbf{H}_{1:}, \mathbf{w} \rangle \\ \langle \mathbf{H}_{2:}, \mathbf{w} \rangle \\ \vdots \\ \langle \mathbf{H}_{d:}, \mathbf{w} \rangle \end{bmatrix}$$

When performing matrix-vector multiplication, you can just imagine the vector $\mathbf{w}$ turning sideways and multiplying each row of $\mathbf{H}$. For matrix-matrix multiplication, $\mathbf{AB}$, you have to ensure that the second dimension of $\mathbf{A}$ equals the first dimension of $\mathbf{B}$. The matrix-matrix multiplication decomposes into matrix-vector multiplication, for each column of $\mathbf{B}$.

As before, to get the incremental update, we can take the gradient of this approximation and obtain the (local) stationary point. Using the basic rules summarized below in Section A.1, the gradient of $\hat{c}(\mathbf{w})$ is

$$\nabla \hat{c}(\mathbf{w}) = \nabla c(\mathbf{w}_0) + \mathbf{H}_{c(\mathbf{w}_0)} (\mathbf{w} - \mathbf{w}_0) \,.$$

Again, we want to find $\mathbf{w}_1$ such that this gradient is zero. If you are not yet familiar with the inverse of a matrix, this will be discussed more in later sections of these notes (particularly for linear regression in Chapter 6). For now, to solve for $\mathbf{H}_{c(\mathbf{w}_0)} (\mathbf{w} - \mathbf{w}_0) = -\nabla c(\mathbf{w}_0)$, one can compute the inverse $\mathbf{H}^{-1}_{c(\mathbf{w}_0)}$ and multiply both sides of the equation by this inverse. This is again analogous to the inverse of a scalar: $h^{-1}h = 1$. The corresponding multivariate update, extended beyond Equation (2.1) for the scalar case, is

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \left(\mathbf{H}_{c(\mathbf{w}_i)} \right)^{-1} \nabla c(\mathbf{w}_i). \tag{3.1}$$

In Equation 3.1, both gradient and Hessian are evaluated at point $\mathbf{w}_i$.

The size of the Hessian makes the choice between first-order and second-order gradient descent less obvious in the multivariate case. Unlike the scalar setting, computing the Hessian itself is expensive (quadratic in the size of $\mathbf{w}$) and it is further even more expensive to compute the inverse of the Hessian. For this reason, more light-weight first-order updates are often preferred. For example, if computing the Hessian costs $O(d^2 n)$ as it does for the linear regression objective, then the computational complexity of the second-order gradient descent is $O(d^3 + d^2 n)$ in each iteration, assuming $O(d^3)$ time for finding matrix inverses. On the other hand, again for linear regression, the computational complexity for first-order gradient descent is only $O(dn)$ per iteration.

The first order update for the multivariate case is an even greater approximation, because the whole Hessian is approximated with a scalar $\frac{1}{\eta}$ (making the Hessian approximation a diagonal matrix with $\frac{1}{\eta}$ on the diagonal). The gradient of the first-order approximation then becomes

$$\nabla \hat{c}(\mathbf{w}) = \nabla c(\mathbf{w}_0) + \frac{1}{\eta} (\mathbf{w} - \mathbf{w}_0)$$

and the resulting first-order update is

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta_i \nabla c(\mathbf{w}_i). \tag{3.2}$$

The selection of this step-size is an important consideration. We have already discussed one basic strategy to select the step-size; in Section 3.5, we discuss a few more.

3.2 Properties of the Hessian

Like the second-derivative, the Hessian reflects the curvature of the function at the point $\mathbf{w}_0$. Each entry reflects how the partial derivative for w_j changes when w_i is changed.

For additional intuition, consider the directional derivative. The directional derivative reflects how a (multivariate) function changes when stepping a small amount t in some fixed direction $\mathbf{u}$

$$\lim_{t \to 0} \frac{c(\mathbf{w} + t\mathbf{u}) - c(\mathbf{w})}{t}.$$

Once we restrict ourselves to how the function changes in this one direction, it is easier to imagine and it allows us to use the familiar second derivative rules for the univariate setting. Let

$$\begin{aligned} \mathbf{w}(t) &= \mathbf{w} + t\mathbf{u} \\ g(t) &= c(\mathbf{w}(t)). \end{aligned}$$

We can use the chain rule on $g(t)$ to compute the derivative w.r.t. t.

$$\begin{aligned} g'(t) &= \nabla c(\mathbf{w}(t))^\top \frac{\partial(\mathbf{w}(t))}{\partial t} \\ &= \nabla c(\mathbf{w}(t))^\top \mathbf{u} \\ g'(0) &= \nabla c(\mathbf{w}(0))^\top \mathbf{u} \\ &= \nabla c(\mathbf{w})^\top \mathbf{u} = 0 \end{aligned}$$

where the last equality occurs because $\mathbf{w}$ is a stationary point and so $\nabla c(\mathbf{w}) = \mathbf{0}$. The second derivative is

$$\begin{aligned} g''(t) &= \frac{\partial(\mathbf{w}(t))}{\partial t}^\top \mathbf{H}_{c(\mathbf{w}(t))} \frac{\partial(\mathbf{w}(t))}{\partial t} \\ &= \mathbf{u}^\top \mathbf{H}_{c(\mathbf{w}(t))} \mathbf{u} \\ g''(0) &= \mathbf{u}^\top \mathbf{H}_{c(\mathbf{w})} \mathbf{u} \end{aligned}$$

For this stationary point $\mathbf{w}$ (corresponding to $t = 0$) to be a local minimum, $g''(0)$ has to satisfy the second derivative test: $g''(0) > 0$. This test is only satisfied if $\mathbf{H}_{c(\mathbf{w})}$ is positive definite, by definition of a positive definite matrix. Recall that a positive-definite matrix $\mathbf{H}$ is one for which, given any $\mathbf{u} \neq \mathbf{0}$, $\mathbf{u}^\top \mathbf{H} \mathbf{u} > 0$, or equivalently, has all eigenvalues greater than zero. Since $\mathbf{u}$ was an arbitrary direction away from $\mathbf{w}$, the Hessian must be positive-definite to ensure that $g''(0) > 0$ for all $\mathbf{u} \neq \mathbf{0}$.

The eigenvalues of the Hessian, therefore, reflect the curvature of the function locally. If $\mathbf{H}_{c(\mathbf{w})}$ has a very small eigenvalue λ, then the corresponding eigenvector $\mathbf{u}$—satisfying $\mathbf{H}_{c(\mathbf{w})}\mathbf{u} = \lambda \mathbf{u}$—is a direction away fro $\mathbf{w}$ where the function is almost flat. This is because $g''(0) = \mathbf{u}^\top \mathbf{H}_{c(\mathbf{w})} \mathbf{u} = \lambda \|\mathbf{u}\|_2^2 = \lambda$ is very small.

Example 8: We can now consider the Hessian $H_{c(\mathbf{w})}$ for the linear regression solution. This Hessian will enable us to verify if we indeed found a local minimum, of if instead we found a stationary point that is a local maximum or a saddle point. The Hessian is

$$H_{c(\mathbf{w})} = 2\mathbf{X}^\top \mathbf{X}.$$

This Hessian is positive semi-definite matrix. To see why, consider that for any vector $\mathbf{w} \neq \mathbf{0}$,

$$\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} = (\mathbf{X}\mathbf{w})^\top \mathbf{X}\mathbf{w} = \|\mathbf{X}\mathbf{w}\|_2^2 \geq 0$$

where equality can only happen—for some $\mathbf{w}$—if the columns of $\mathbf{X}$ are linearly dependent. Since the Hessian is positive semi-definite for every $\mathbf{w}$, this verifies the convexity of $c(\mathbf{w})$. Furthermore, if the columns of $\mathbf{x}$ are linearly independent, the Hessian is positive definite, which implies that the global minimum is unique. □

3.3 Handling big data sets

One common approach to handling big datasets is to use *stochastic approximation*, where samples are processed incrementally. To see how this would be done, let us revisit the gradient of the objective function, $\nabla c(\mathbf{w})$. We obtained a closed form solution for $\nabla c(\mathbf{w}) = \mathbf{0}$; however, for many other objective functions, solving for $\nabla c(\mathbf{w}) = \mathbf{0}$ in a closed form way is not possible. Instead, we start at some initial $\mathbf{w}_0$ (typically random), and then step in the direction of the negative of the gradient until we reach a local minimum. This approach is called *gradient descent* and is summarized in Algorithm 2. Notice that here the gradient is normalized by the number of samples n, as $\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$ grows with the number of samples and makes it more difficult to select the stepsize.

Algorithm 2: Batch Gradient Descent$(c, \mathbf{X}, \mathbf{y})$

1: // A non-optimized, basic implementation of batch gradient descent
2: $\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^d$
3: err $\leftarrow \infty$
4: tolerance $\leftarrow 10e^{-4}$
5: max iterations $\leftarrow 10e^5$
6: **while** $|c(\mathbf{w}) - \text{err}| >$ tolerance and have not reached max iterations **do**
7: err $\leftarrow c(\mathbf{w})$ ▷ for linear regression, $c(\mathbf{w}) = \frac{1}{2n}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$
8: $\mathbf{g} \leftarrow \nabla c(\mathbf{w})$ ▷ for linear regression, $\nabla c(\mathbf{w}) = \frac{1}{n}\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$
9: // The step-size η could be chosen by line-search, as in Algorithm 1
10: $\eta \leftarrow$ line search$(\mathbf{w}, c, \mathbf{g})$
11: $\mathbf{w} \leftarrow \mathbf{w} - \eta\mathbf{g}$
12: **return** $\mathbf{w}$

For a large number of samples n, however, computing the gradient across all samples can be expensive or infeasible. An alternative is to approximate the gradient less accurately with fewer samples. In stochastic approximation, we typically approximate the gradient with one sample[1], as in Algorithm 3. Though this approach may appear to be too much of an approximation, there is a long theoretical and empirical history indicating its effectiveness (see for example [5, 4]). With ever increasing data-set size for many scenarios, the generality of stochastic approximation makes it arguably the modern approach to dealing with big data. For specialized scenarios, there are of course other approaches. For one example, see [16].

[1] Mini-batches are a way to obtain a better approximation but remain efficient.

The training algorithm for stochastic gradient descent can now be revised to randomly draw one data point at a time from $\mathcal{D}$ and then update the current weights using the previous equation. Typically, in practice, this entails iterating one or more times over the dataset in order (assuming it is random, with i.i.d. samples). Each iteration over the dataset is called an *epoch*. The conditions for convergence typically include conditions on the step-sizes, requiring them to decrease over time. As with batch gradient descent, these stochastic gradient descent updates will converge, though with more oscillation around the true weight vector, with the decreasing step-size progressively smoothing out these oscillations.

Algorithm 3: Stochastic Gradient Descent($c, \mathbf{X}, \mathbf{y}$)

1: $\mathbf{w} \leftarrow$ random vector in $\mathbb{R}^d$
2: **for** $i = 1, \ldots$ number of epochs **do**
3: Shuffle data points from $1, \ldots, n$
4: **for** $j = 1, \ldots, n$ **do**
5: $\mathbf{g} \leftarrow \nabla c_j(\mathbf{w})$ ▷ for linear regression, $\nabla c_j(\mathbf{w}) = (\mathbf{x}_j^\top \mathbf{w} - y_j)\mathbf{x}_j$
6: // For convergence, the step-size η_t needs to decrease with time, such as
7: // $\eta_t = \eta_0 t^{-1/2}$ or $\eta_t = \eta_0 i^{-1}$ for an initial η_0 (e.g., $\eta_0 = 1.0$).
8: // In practice, it is common to pick a fixed, small stepsize
9: $\eta_t \leftarrow i^{-1}$
10: $\mathbf{w} \leftarrow \mathbf{w} - \eta_t \mathbf{g}$
11: **return** $\mathbf{w}$

3.4 Non-smooth but still continuous optimization

We assume throughout these notes that our objectives are continuous. However, this need not mean that they are smooth: in some cases, these continuous objectives may have non-differentiable points. For example, the ℓ_1 regularizer is non-differentiable at 0, making $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1$ non-differentiable. One strategy is to use sub-gradient descent; loosely, this amounts to selecting a reasonable choice for the gradient at the non-differentiable point. Here, for example, we could take the partial derivative of ℓ_1 for w_j to be zero at zero, -1 for $w_j < 0$ and 1 for $w_j > 0$. Unfortunately, this descent is slow because there is a tendency to jump around zero. Unlike ℓ_2, the gradient does not gradually decrease near zero, slowly decreasing w_j, but rather jumps between two large values -1 and 1. With such large gradient, it is difficult to gradually decrease w_j to zero, even if that is the optimal solution.

One alternative for such non-smooth objectives is to use proximal methods. The idea is simple: use gradient descent for the smooth component of the optimization (the error term $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$), and then for values in $\mathbf{w}$ that are close to zero, set them to zero. This thresholding idea, though simple, is a theoretically sound approach for optimizing with the non-smooth ℓ_1. This thresholding operator is called the proximal operator, and can be seen as a projection operator. Each time $\mathbf{w}$ is updated with the gradient, it moves it away from a sparse solution; the proximal operator then projects $\mathbf{w}$ back onto the space of sparse solutions. The proximal operator for ℓ_1 is applied element-wise to $\mathbf{w}$, and so is defined on

each $\mathbf{w}_i$ as, with stepsize η and regularization parameter λ,

$$\text{prox}_{\eta\lambda\ell_1}(\mathbf{w}_i) = \begin{cases} \mathbf{w}_i - \eta\lambda & \text{if } \mathbf{w}_i > \eta\lambda \\ 0 & \text{if } |\mathbf{w}_i| \leq \eta\lambda \\ \mathbf{w}_i + \eta\lambda & \text{if } \mathbf{w}_i < -\eta\lambda. \end{cases}$$

The proximal operator on the entire vector $\mathbf{w}$ is defined element-wise: $\text{prox}_{\eta\lambda\ell_1}(\mathbf{w}) = [\text{prox}_{\eta\lambda\ell_1}(\mathbf{w}_1), \ldots, \text{prox}_{\eta\lambda\ell_1}(\mathbf{w}_d)]$. Nicely, the theory states that the stepsize should be no larger than the inverse of the Lipschitz constant for the smooth part of the objective, where intuitively the Lipschitz constants reflects how quickly the function changes. In Algorithm 4, we provide a gradient descent algorithm for the incremental update with the ℓ_1 regularizer, introduced as an algorithm called ISTA [3]. More generally, proximal methods are used for other non-smooth objectives, though in these notes we only consider Lasso.

Algorithm 4: Batch gradient descent for ℓ_1 regularized linear regression $(\mathbf{X}, \mathbf{y}, \lambda)$

1: $\mathbf{w} \leftarrow \mathbf{0} \in \mathbb{R}^d$
2: err $\leftarrow \infty$
3: tolerance $\leftarrow 10e^{-4}$
4: // Precomputing these matrices, to avoid recomputing them in the loop
5: $XX \leftarrow \frac{1}{n}\mathbf{X}^\top\mathbf{X}$
6: $Xy \leftarrow \frac{1}{n}\mathbf{X}^\top\mathbf{y}$
7: // This stepsize is specific to the least-squares loss for linear regression
8: $\eta \leftarrow 1/(2\|XX\|_F)$
9: **while** $|c(\mathbf{w}) - \text{err}| >$ tolerance and have not reached max iterations **do**
10: err $\leftarrow c(\mathbf{w})$
11: // Proximal operator projects back into the space of sparse solutions given by ℓ_1
12: $\mathbf{w} \leftarrow \text{prox}_{\eta\lambda\ell_1}(\mathbf{w} - \eta XX\mathbf{w} + \eta Xy)$
13: **return** $\mathbf{w}$

3.5 More methods to select the step-size

Because selecting the step-size is such an important part of an effective descent algorithm, there are many ways to do so. In addition to line search, one of the most popular methods is to use quasi-second-order (or quasi-Newton) methods. As we saw, the inverse of the Hessian provides a good way to select the stepsize, but is typically too expensive to compute let alone invert. Quasi-second-order methods approximate the Hessian, with as little storage and computation as possible. One of the simplest such approximations is to approximate only the diagonal of the Hessian, and the invert it, which only costs $O(d)$ computation and space. Such an approximation is typically quite poor for even the diagonal of the inverse Hessian, and so is not commonly used. Instead, the most popular methods include LBFGS [13], Adadelta [19] and Adam [10].