4
Gradient-based Optimization and its Analysis

This chapter sets up the basic analysis framework for gradient-based optimization algorithms and discuss how it applies to deep learning. The algorithms work well in practice; the question for theory is to analyse them and give recommendations for practice. This has happened to some extent, but it has also become clearer that classical ways of thinking about optimization may not be a perfect match with phenomena encountered in deep learning. Nevertheless, the material in this chapter is foundational for thinking about optimization in deep learning. We discus the basic algorithms, and then give the handful of clean analyses of these methods that use minimal assumptions and thus can be applied to deep learning as well. [1]

[1] Note that an analysis with minimal assumptions is the one that is most generally applicable. The downside is that it yields convergence bounds that are far from what one might observe in real life.

The conceptual framework in optimization builds upon simple Taylor approximation (Equation 4.1) of the loss function and thus relies upon derivatives (of various orders) of the loss function.

Gradient descent (GD) If you are reading this book you probably know about gradient descent, a fundamental idea in machine learning and AI. Suppose we wish to minimize a continuous function $f(w)$ over $\mathbb{R}^d$.

$$\min_{w \in \mathbb{R}^d} f(w)\,.$$

The gradient descent (GD) algorithm is

$$\begin{aligned} w_0 &= \text{initialization} \\ w_{t+1} &= w_t - \eta \nabla f(w_t) \end{aligned}$$

where η is called *step size* or *learning rate.* The choice of η is important and discussed later in the rest of the Chapter. [2]

[2] Gradient descent is not guaranteed to find optimum solutions for general loss functions. For instance, complexity theory shows that given a degree 4 polynomial $p(w_1, w_2, \ldots, w_n)$ in n variables of total degree at most 6, it is NP-hard to determine whether or not it is 0 for some assignment to the variables. This can be proven easily using NP-completeness of 3SAT problem.

There is an efficient algorithm for computing the gradient of a loss function for deep learning, the classic back-propagation algorithm (Chapter 3).

Note that today's deep nets often use nonlinear activations, e.g., ReLU, that causes the overall function computed by the net to be non-differentiable. However, this non-differentiability is of the mild sort and does not appear to be an issue in practice.

4.1 Understanding Optimization via Taylor Expansion

4.1.1 First-Order Method: Gradient Descent

The motivation for Gradient Descent (GD) is that the update direction, $-\nabla f(w_t)$, is the direction of steepest local descent. To see why, consider the Taylor expansion of the function $f(\cdot)$ at a point w_t:

$$f(w) = f(w_t) + \underbrace{\langle \nabla f(w_t), w - w_t \rangle}_{\text{linear in } w} + \underbrace{\frac{1}{2}(w - w_t)^T \nabla^2 f(w_t)(w - w_t)}_{\text{quadratic in } w} + \cdots \tag{4.1}$$

Here, $\nabla^2 f(\cdot)$ is the matrix of second-order derivatives called the *Hessian*. Its (i, j) entry is $\partial^2 f / \partial w_i \partial w_j$, and it is a symmetric matrix[3].

[3] The order-k term in the Taylor expansion involves the tensor of all kth-order partial derivatives, consisting of d^k entries.

If we keep only the first-order (linear) approximation, our goal is to find a small step $\delta = w - w_t$ that minimizes this approximation. We constrain the step to have a small norm, e.g., $\|\delta\|_2 \le \epsilon$, to ensure we stay in a region where the approximation is valid.

$$\begin{aligned} \min_{\delta \in \mathbb{R}^d} \quad & f(w_t) + \nabla f(w_t)^T \delta \\ \text{subject to} \quad & \|\delta\|_2 \le \epsilon \end{aligned}$$

Since $f(w_t)$ is a constant, this is equivalent to minimizing the inner product $\nabla f(w_t)^T \delta$.

Problem 4.1.1. *Show that the vector δ that solves the minimization problem above is $\delta = -\alpha \nabla f(w_t)$ for some positive scalar $\alpha > 0$.*

Solution Sketch: The inner product can be written as $\nabla f(w_t)^T \delta = \|\nabla f(w_t)\|_2 \|\delta\|_2 \cos\theta$, where θ is the angle between the gradient and the step δ. To make this quantity as negative as possible, we must choose $\cos\theta = -1$, which means δ must point in the exact opposite direction of the gradient $\nabla f(w_t)$. The magnitude of the step is determined by the constraint ϵ.

This confirms that the optimal local direction to move is precisely the negative gradient. This gives us the standard Gradient Descent update rule, where η is the learning rate:

$$w_{t+1} = w_t - \eta \nabla f(w_t) \tag{4.2}$$

4.1.2 *Second-Order Method: Newton's Method*

The first-order approximation is simple, but it ignores the curvature of the function, captured by the Hessian term. What if we use a more accurate, second-order (quadratic) model of the function around w_t?

$$f(w_t + \delta) \approx f_q(\delta) = f(w_t) + \nabla f(w_t)^T \delta + \frac{1}{2}\delta^T H_t \delta \tag{4.3}$$

where $H_t = \nabla^2 f(w_t)$ is the Hessian at w_t.

Instead of just finding the best direction, we can now find the exact step δ that minimizes this quadratic approximation. For a convex quadratic, the minimum can be found by setting its gradient with respect to δ to zero:

$$\nabla_\delta f_q(\delta) = \nabla f(w_t) + H_t \delta = \mathbf{0} \tag{4.4}$$

$$\implies H_t \delta = -\nabla f(w_t) \tag{4.5}$$

$$\implies \delta = -H_t^{-1} \nabla f(w_t) \tag{4.6}$$

This optimal step is called the **Newton step**. This leads to **Newton's Method**, an iterative optimization algorithm where the update rule is:

$$w_{t+1} = w_t - H_t^{-1} \nabla f(w_t) \tag{4.7}$$

Why is this a good idea? While gradient descent follows the steepest slope, Newton's method directs the step towards the minimum of a local quadratic approximation of the function. This gives it two major advantages:

1. **It accounts for curvature.** If the loss surface is a long, narrow valley (i.e., high curvature in one direction), the gradient might point perpendicularly across the valley. The term H_t^{-1} acts as a preconditioner that rescales the gradient, correcting its direction to point more directly towards the minimum.

2. **Faster Convergence.** Near a minimum, Newton's method exhibits quadratic convergence, which is much faster than the linear convergence of gradient descent. This means the number of correct digits in the solution can roughly double with each iteration.

The Catch: Why isn't it always used? Despite its power, Newton's method is rarely used to train large models like LLMs due to prohibitive computational costs:

1. **Computing the Hessian:** For a model with d parameters, the Hessian is a $d \times d$ matrix, requiring $O(d^2)$ work to compute.

2. **Storing the Hessian:** Storing the matrix requires $O(d^2)$ memory.

3. **Inverting the Hessian:** Inverting the matrix is an $O(d^3)$ operation.

For a model with millions or billions of parameters (d), these costs are infeasible. Furthermore, if the Hessian is not positive definite (e.g., at a saddle point), the Newton step may not even be a descent direction.

This has led to the development of **Quasi-Newton methods** (like BFGS and L-BFGS), which approximate the inverse Hessian iteratively without ever forming the full matrix, offering a compromise between the speed of second-order methods and the efficiency of first-order methods.

4.1.3 *Bounding the Taylor Expansion via Smoothness*

The foundational analysis of Gradient Descent's convergence speed relies on a property of the loss function called smoothness. Intuitively, a function is smooth if its gradient does not change too rapidly.

Definition 4.1.2 (L-smoothness)**.** *A differentiable function f is L-smooth if its gradient is L-Lipschitz continuous, meaning for any two points w_1, w_2:*

$$\|\nabla f(w_1) - \nabla f(w_2)\|_2 \leq L\|w_1 - w_2\|_2 \tag{4.8}$$

If f is twice differentiable, this is equivalent to stating that for any w, all eigenvalues of the Hessian $\nabla^2 f(w)$ lie in the interval $[-L, L]$.

This property allows us to create a tight quadratic upper bound on the function.

Problem 4.1.3. [4] *Prove that if f is L-smooth, then for any w and w_t:*

$$f(w) \leq f(w_t) + \nabla f(w_t)^T(w - w_t) + \frac{L}{2}\|w - w_t\|_2^2 \tag{4.9}$$

[4] Remember Taylor's theorem: $f(w) - f(w_1) - \nabla f(w_1)(w - w_1)$ is a power series and it can be upper bounded by the 2nd order term of the Taylor expansion evaluated at an intermediate point between w, w_1.

This inequality is fundamental. It tells us that the function is never much larger than its local quadratic approximation, with the curvature of the quadratic controlled by L.

4.1.4 *The Descent Lemma for Gradient Descent*

Using the smoothness bound, we can prove that gradient descent makes progress on the objective function with a sufficiently small learning rate, unless it is already at a stationary point (where the gradient is zero).

Lemma 4.1.4 (Descent Lemma)**.** *Suppose f is L-smooth. If we use the gradient descent update $w_{t+1} = w_t - \eta\nabla f(w_t)$ with a learning rate $\eta \leq 1/L$, we have:*

$$f(w_{t+1}) \leq f(w_t) - \frac{\eta}{2}\|\nabla f(w_t)\|_2^2$$

[5]

Proof. Let $\delta_t = w_{t+1} - w_t = -\eta \nabla f(w_t)$. We use the upper bound from Eq. (4.9):

$$\begin{aligned} f(w_{t+1}) &\leq f(w_t) + \nabla f(w_t)^T(-\eta \nabla f(w_t)) + \frac{L}{2}\| - \eta \nabla f(w_t)\|_2^2 \\ &= f(w_t) - \eta\|\nabla f(w_t)\|_2^2 + \frac{L\eta^2}{2}\|\nabla f(w_t)\|_2^2 \\ &= f(w_t) - \eta(1 - \frac{L\eta}{2})\|\nabla f(w_t)\|_2^2 \end{aligned}$$

Descent is guaranteed as long as the term $(1 - L\eta/2)$ is positive, which holds for any $\eta < 2/L$. For the specific rate in the lemma, if $\eta \leq 1/L$, then $(1 - L\eta/2) \geq 1/2$. Substituting this gives:

$$f(w_{t+1}) \leq f(w_t) - \frac{\eta}{2}\|\nabla f(w_t)\|_2^2$$

□

[5] The proof also shows that descent occurs so long as $\eta < 2/L$.

This lemma shows that the function value decreases at each step, and the decrease is proportional to the squared norm of the gradient. This has an important consequence.

Corollary 4.1.5. *For a function that is bounded below, repeatedly applying the Descent Lemma implies that $\|\nabla f(w_t)\|_2^2 \to 0$ as $t \to \infty$. This means gradient descent is guaranteed to converge to a* ***stationary point****, where the gradient is zero.*

For a convex function, any stationary point is a global minimum. However, the loss functions for deep networks are non-convex, meaning they can have many stationary points that are not global minima (i.e., local minima or saddle points).

Definition 4.1.6. *We say w is a* ***stationary point*** *of f if $\nabla f(w) = \mathbf{0}$. If, in addition, the Hessian $\nabla^2 f(w)$ is positive semidefinite at w, then w is a* ***local minimum****.*

Despite the lack of a global guarantee, the stationary points (more precisely, near-stationary points) found by gradient-based methods in practice often correspond to solutions with very low training loss and good generalization.

4.2 *Stochastic Gradient Descent (SGD)*

In modern machine learning, the loss function is typically an average over a large dataset of n samples:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

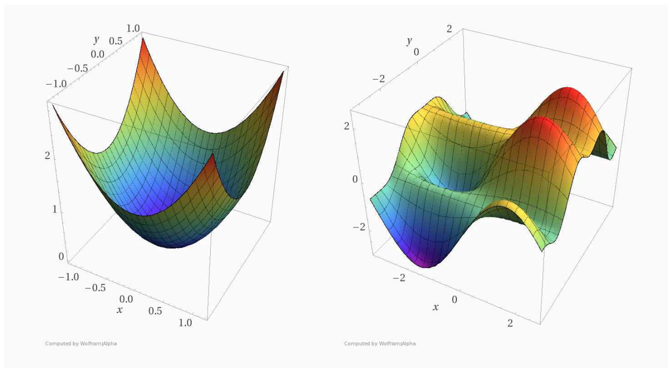

Figure 4.1: Convex and Nonconvex Functions in two variables. For nonconvex functions GD will reach a stationary point, where gradient is zero. (Figure from kdnuggets.org)

where $f_i(w)$ is the loss on the i-th example. Computing the full gradient $\nabla L(w)$ requires a complete pass over the dataset, which is computationally prohibitive for large n.

Stochastic Gradient Descent (SGD) addresses this by approximating the gradient using a small random sample of the data, called a mini-batch. At each iteration t, we sample a mini-batch S_t of size $B \ll n$ and compute the stochastic gradient:

$$g_t(w_t) = \frac{1}{B} \sum_{i \in S_t} \nabla f_i(w_t)$$

This is an unbiased estimator of the true gradient [6] i.e., $\mathbb{E}_{S_t}[g_t(w_t)] = \nabla L(w_t)$. The SGD update rule is then:

$$w_{t+1} = w_t - \eta \cdot g_t(w_t)$$

[6] Here we are using linearity of expectation and linearity of derivatives; very convenient!

While the noisy gradients cause the loss to fluctuate, SGD is superior to full-batch GD in practice for several reasons:

1. **Efficiency:** It allows for far more parameter updates in the same amount of time.

2. **Regularization Effect:** While it may seem that the noise/error in the gradient estimate should hurt the optimization, in actually it seems to help in deep learning! In other words, **stochastic optimization is superior to exact optimization**. One important reason is that in deep learning the ultimate goal is a model that works well on unseen data. This is the topic of generalization, as explained in later chapters. Roughly, the idea is that full GD can end up in sharp local minima, whereas stochastic GD tends to find "flatter" minima that tend to generalize better to unseen data. We will study this "Implicit Bias" effect in more detail in later chapters.)

4.3 *Accelerated Gradient Descent (Momentum)*

The path taken by GD can be highly oscillatory, especially in narrow valleys of the loss landscape. Accelerated methods aim to dampen these oscillations and speed up convergence by incorporating memory of past updates.

The most common version is the **heavy-ball algorithm**, which adds a momentum term:

$$w_{t+1} = w_t \underbrace{-\eta \nabla f(w_t)}_{\text{current gradient}} + \underbrace{\beta(w_t - w_{t-1})}_{\text{momentum}}$$

The parameter $\beta \in [0, 1)$ is the momentum coefficient. This update is analogous to a heavy ball rolling down a hill: the gradient acts as a force, while the momentum term keeps the ball moving in its previous direction, accumulating velocity. This helps it roll past small bumps and speed up along consistent downhill directions.

The update can be equivalently written by tracking the velocity vector $\boldsymbol{v}_t$:

$$\begin{aligned} \boldsymbol{v}_t &= \beta \boldsymbol{v}_{t-1} + \eta \nabla f(w_t) \\ w_{t+1} &= w_t - \boldsymbol{v}_t \end{aligned}$$

This formulation shows that the update is a weighted average of past gradients.

A popular variant is **Nesterov Accelerated Gradient (NAG)**. The key idea is to compute the gradient not at the current position w_t, but at a "look-ahead" position where the momentum is about to carry the iterate.

$$\begin{aligned} \boldsymbol{v}_t &= \beta \boldsymbol{v}_{t-1} + \eta \nabla f(w_t - \beta \boldsymbol{v}_{t-1}) \\ w_{t+1} &= w_t - \boldsymbol{v}_t \end{aligned}$$

By "looking ahead," NAG can react more quickly and correct its course, leading to stronger theoretical convergence guarantees for convex functions. In practice, both methods improve significantly upon standard GD and SGD for minimizing convex functions. For deep learning they often help as well though, though as noted earlier the training loss has multiple minima for deep nets, and thus evaluating the goodness of the optimization algorithm involves looking at the generalization properties, rather than mere optimization speed.

4.4 *Convergence Analysis: Warmup via Quadratic Models*

This being a book on Theory of Deep Learning, we are interested in understanding convergence behavior of different algorithms: how fast do they find a "good enough" solution? The underlying issues can be understood by analyzing GD on a simple convex quadratic function:

$$f(w) = \frac{1}{2} w^\top A w$$

where $A \in \mathbb{R}^{d \times d}$ is a positive semidefinite matrix. The gradient is $\nabla f(w) = Aw$ and the Hessian is $\nabla^2 f(w) = A$.

To simplify the analysis, we can work in a coordinate system defined by the eigenvectors of A. If $A = U \Sigma U^\top$ is the eigendecomposition of A, we can define $\widehat{w} = U^\top w$. The objective becomes $f(\widehat{w}) = \frac{1}{2} \widehat{w}^\top \Sigma \widehat{w}$, where $\Sigma = \operatorname{diag}(\lambda_1, \ldots, \lambda_d)$ contains the eigenvalues. The function is now a sum of decoupled quadratics:

$$f(\widehat{w}) = \frac{1}{2} \sum_{i=1}^{d} \lambda_i \widehat{w}_i^2$$

The GD update for each coordinate $\widehat{w}_i$ is independent:

$$\widehat{w}_i^{(t+1)} = \widehat{w}_i^{(t)} - \eta \lambda_i \widehat{w}_i^{(t)} = (1 - \eta \lambda_i) \widehat{w}_i^{(t)}$$

For this to converge, we need $|1 - \eta \lambda_i| < 1$ for all i, which implies $\eta < 2/\lambda_{\max}$, where $\lambda_{\max} = \max_i \lambda_i = L$ is the smoothness constant.

The convergence rate is dictated by the slowest-converging coordinate. The rate of convergence for coordinate i is $(1 - \eta \lambda_i)$. To make progress on the fastest-curving direction (corresponding to $\lambda_{\max}$), we must set $\eta \approx 1/\lambda_{\max}$. With this learning rate, the convergence factor for the slowest direction (corresponding to $\lambda_{\min}$) becomes $(1 - \lambda_{\min}/\lambda_{\max})$. This leads to a key principle:

The convergence rate of GD is governed by the **condition number** $\kappa = \lambda_{\max}/\lambda_{\min}$, which is the ratio of the largest to the smallest eigenvalue of the Hessian.

A high condition number means the loss surface is a steep, elongated valley. GD will oscillate across the narrow direction while making slow progress along the flat direction.

Need figure showing the elliptical contour lines of a poorly-conditioned quadratic. The figure would show GD taking many small, zig-zagging steps, whereas a preconditioned method like Newton's would point directly to the minimum.

4.4.1 *Preconditioning and Adaptive Methods*

The analysis above shows that a single learning rate is not optimal for all directions. Ideally, we would use a per-coordinate learning rate of $\eta_i \approx 1/\lambda_i$. This is the core idea of **preconditioning**. In the original coordinate system, this corresponds to the update:

$$w_{t+1} = w_t - A^{-1}\nabla f(w_t)$$

For a general function, this becomes Newton's method, using the inverse Hessian as a preconditioner:

$$w_{t+1} = w_t - (\nabla^2 f(w_t))^{-1}\nabla f(w_t)$$

As discussed previously, computing and inverting the Hessian is too expensive. **Adaptive methods** approximate this preconditioning by using information from the gradients themselves to dynamically adjust a per-parameter learning rate.

4.5 *Popular optimizers*

In the past 15 years a raft of optimization methods were invented that try to use the central insight of 2nd order methods in a more efficient update rule. The idea is to maintain some running average of the squared gradients for each parameter to estimate the curvature (i.e., Hessian).

Adagrad (Adaptive Gradient Algorithm) [7] scales the learning rate for each parameter inversely proportional to the square root of the sum of all past squared gradients.

[7] Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. J. Duchi, E. Hazan, Y. Singer. JMRL 2011

- **Problem:** The accumulated sum only grows, causing the learning rate to monotonically decrease and eventually become infinitesimally small, prematurely stopping training.

RMSprop (Root Mean Square Propagation), an improvement in G. Hinton's unpublished notes, fixes this by using an exponentially decaying moving average of squared gradients instead of a sum. This allows the learning rate to adapt and not just decrease. Let $\boldsymbol{s}_t$ be the moving average of squared gradients:

$$\begin{aligned} \boldsymbol{s}_t &= \gamma \boldsymbol{s}_{t-1} + (1-\gamma)(\boldsymbol{g}_t \odot \boldsymbol{g}_t) \\ w_{t+1} &= w_t - \frac{\eta}{\sqrt{\boldsymbol{s}_t} + \epsilon} \odot \boldsymbol{g}_t \end{aligned}$$

where $\odot$ is element-wise multiplication and ϵ is a small constant for stability.

Adam (Adaptive Moment Estimation) [8] is a further modification that has become one of the most popular optimization algorithms today. It combines the momentum idea (first moment of the gradient) with the adaptive learning rates of RMSprop (second moment). It

[8] ADAM: A Method For Stochastic Optimization. D. Kingma and J. Ba. ICLR 2015

maintains moving averages for both the gradient and its square:

$$\begin{aligned} \boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t && \text{(Momentum)} \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)(\boldsymbol{g}_t \odot \boldsymbol{g}_t) && \text{(RMSprop-like)} \end{aligned}$$

After bias correction (not shown), the update combines these two:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\boldsymbol{v}_t} + \epsilon} \odot \boldsymbol{m}_t$$

Adam has been a popular choice for training deep neural networks due to its robust performance across a wide range of tasks.

AdamW (Adam with Decoupled Weight Decay) [9] is a variant of Adam that adjusts its implementation of L2 regularization. In standard Adam, the weight decay is implicitly affected by the adaptive learning rates, as it becomes part of the gradient term $\boldsymbol{g}_t$. This couples the decay to the second moment estimate $\boldsymbol{v}_t$, making it less effective.

[9] Decoupled Weight Decay Regularization. I. Loshchilov and F. Hutter. ICLR 2019

AdamW decouples the weight decay from the gradient update. The momentum and RMSprop-like parts are unchanged:

$$\begin{aligned} \boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)(\boldsymbol{g}_t \odot \boldsymbol{g}_t) \end{aligned}$$

The weight decay is then applied directly to the parameters in the final update step, where λ is the weight decay rate:

$$w_{t+1} = w_t - \eta \left(\frac{\widehat{\boldsymbol{m}}_t}{\sqrt{\widehat{\boldsymbol{v}}_t} + \epsilon} + \lambda w_t \right)$$

This seemingly small change often leads to significantly better generalization performance, and AdamW has become the default choice in many modern deep learning frameworks.

LION (Evolved Sign Momentum) is a novel optimizer discovered by a Google Brain team[10] through an automated search process, rather than by manual design. It is simpler and more memory-efficient than Adam, as it does not require a second-moment estimate. LION's core idea is to use the sign of the momentum to determine the update direction, applying a uniform update magnitude across all parameters.

[10] Symbolic Discovery of Optimization Algorithms, Chen et al. 2023

The update is split into two parts: generating the update direction and updating the momentum itself, using two distinct hyperparameters β_1 and β_2.

$$\begin{aligned} \boldsymbol{u}_t &= \text{sign}(\beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t) && \text{(Update direction)} \\ \boldsymbol{m}_t &= \beta_2 \boldsymbol{m}_{t-1} + (1-\beta_2)\boldsymbol{g}_t && \text{(Momentum update)} \end{aligned}$$

The parameter update is then simply a step in the direction of $\boldsymbol{u}_t$:

$$w_{t+1} = w_t - \eta \boldsymbol{u}_t$$

By discarding the magnitude information from the gradient and relying only on the sign, LION decouples the update from the adaptive learning rates seen in Adam and RMSprop. This simplification saves memory (no v_t vector) and has been shown to achieve strong performance, particularly in large-scale training for models like Vision Transformers.

4.5.1 *Exotic LR schedules*

Another feature of modern deep learning is proliferation of *learning rate schedules*, which are recipes for adjusting the LR during training. These adjustments are to the final update direction computed according to one of the above methods.

For language models the popular choice is cosine schedule[11] given by

$$\eta_t = \frac{1}{2}(1 + \cos(\frac{t-1}{T}\pi)) \qquad 1 \leq t \leq T+1.$$

[11] I. Loshchilov, and F. Hutter. Decoupled weight decay regularization. ICLR, 2019.

We will return to these later in the book, and present some analyses for why they work.

4.6 *Convergence rates under smoothness conditions*

As mentioned earlier, gradient-based methods cannot in general find the optimum value of even simple nonconvex functions such as low-degree polynomials. But we did note that if the function is differentiable and smooth, then with a suitably small learning rate, loss does decrease monotonically so long as the gradient is nonzero. In other words, the process ends up with a *stationary point,* where $\nabla = 0$. This chapter establishes upper bounds on how long it takes to get close to a stationary point. See Chapter 7 for analysis of convergence rate to a stronger type of solution: local optimum.

As usual the objective/loss function is denoted $f(w)$ where $w \in \Re^d$. The procedure has T iterations, and the parameter vectors in these iterations are denoted $w_1, ..., w_T$ respectively. We assume *boundedness*: i.e., there is a known M such that $|f(w_t)| \leq \frac{M}{2}$ for all $t = 1, \ldots, T$. We also assume f is L-smooth, which implies

$$f(w) \leq f(w') + \nabla f(w')(w - w') + \frac{L}{2}\|w - w'\|^2. \qquad (4.10)$$

Throughout the chapter, ∇_t is shorthand for $\nabla f(w_t)$.

4.6.1 *Need for smoothness: A simple lower bound*

In constrained non-convex optimization, finding a stationary point presents difficult computational challenges. Even when objective functions are bounded, local information may provide no information about the location of a stationary point.

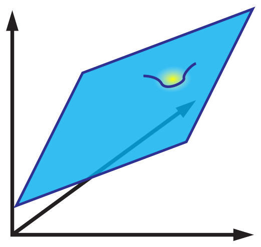

Figure 4.2: A difficult "needle in a haystack" case for non-convex optimization. A function with a hidden valley, with small gradients shown in yellow.

Consider, for example, the function sketched in Figure 4.2. In this construction, defined on the hypercube in $\mathbb{R}^n$, the unique point with a vanishing gradient is a hidden valley, and gradients outside this valley are all identical. Clearly, it is hopeless in an information-theoretic sense to find this point efficiently: the number of value or gradient evaluations of this function must be $\exp(\Omega(n))$ to discover the valley.

To circumvent such inherently difficult and degenerate cases, we require that the objective function be smooth. As we shall see, this allows efficient algorithms for finding a point with small gradient.

4.6.2 *Convergence rates for GD*

This section analyses gradient descent given exact gradient. Next section analyses stochastic GD.

Algorithm 2 Gradient descent

```
1: Input: $f$, $T$, initial point $w_1 \in \mathrm{K}$, sequence of step sizes $\{\eta_t\}$
2: for $t = 1$ to $T$ do
3:     Let $w_{t+1} = w_t - \eta_t \nabla f(w_t)$
4: end for
5: return $w_\tau, \tau \in [T]$ s.t. $\nabla_\tau$ is smallest in Euclidean norm.
```

Theorem 4.6.1. *For unconstrained minimization of L-smooth functions and $\eta_t = \frac{1}{L}$, Algorithm 2 satisfies the following for at least one $\tau \in [1, T]$ ($M =$ largest absolute value of any $f(w_t)$ encountered during the algorithm.*

$$\|\nabla_\tau\|^2 \leq \frac{1}{T}\sum_t \|\nabla_t\|^2 \leq \frac{4ML}{T}.$$

Proof. The **Descent Lemma** implies

$$\begin{aligned} f(w_{t+1}) - f(w_t) &\leq \nabla_t^\top (w_{t+1} - w_t) + \frac{L}{2}\|w_{t+1} - w_t\|^2 && L\text{-smoothness} \\ &= -\eta_t \|\nabla_t\|^2 + \frac{L}{2}\eta_t^2 \|\nabla_t\|^2 && \text{algorithm defn.} \\ &= -\frac{1}{2L}\|\nabla_t\|^2 && \text{choice of } \eta_t = \frac{1}{L} \end{aligned}$$

Thus, summing up over T iterations, we have

$$\frac{1}{2L}\sum_{t=1}^{T} \|\nabla_t\|^2 \leq \sum_t (f(w_t) - f_{w_{t+1}}) \leq 2M$$

Thus it follows that at least one suitable τ exists in $[1, T]$. □

4.6.3 *Stochastic gradient descent*

In optimization for machine learning, the objective function f takes the form

$$f(w) = \frac{1}{m}\sum_i \ell(w, z_i),$$

where $z_i, i \in [m]$ are the training set examples, and ℓ is some loss function that applies to the parameters w and datapoint z_i. The key idea of Stochastic Gradient Descent is that a random variable can be used in lieu of the gradient, that has the same expectation. This random variable is simply the average gradient of small *batch* of examples from the training set. The analysis below even allows batch size 1 (see Problem 4.6.3).

We denote by $\widehat{\nabla}_t$ a random variable such that $\mathbb{E}[\widehat{\nabla}_t] = \nabla f(w_t) = \nabla_t$ (where expectation is over randomness used in gradient estimation) and a bound on the second moment of this random variable by

$$\mathbb{E}[\|\widehat{\nabla}_t\|^2] = \sigma^2. \tag{4.11}$$

Algorithm 3 Stochastic gradient descent

```
1: Input: $f$, $T$, initial point $w_1 \in K$, sequence of step sizes $\{\eta_t\}$
2: for $t = 1$ to $T$ do
3:     Let $w_{t+1} = w_t - \eta_t \widehat{\nabla}_t$
4: end for
5: return $w_\tau, \tau \in [T]$ s.t. $\nabla_\tau$ is smallest in Euclidean norm.
```

Theorem 4.6.2. *For unconstrained minimization of L-smooth functions and* $\eta_t = \eta = \sqrt{\frac{M}{L\sigma^2 T}}$, *Algorithm 3 satisfies*

$$\mathbb{E}[\|\nabla_\tau\|^2] \leq \mathbb{E}\left[\frac{1}{T}\sum_t \|\nabla_t\|^2\right] \leq 2\sqrt{\frac{ML\sigma^2}{T}}.$$

Proof. Denote by ∇_t the shorthand for $\nabla f(w_t)$. The stochastic descent lemma is given in the following equation,

$$\begin{aligned} f(w_{t+1}) - f(w_t) &\leq \mathbb{E}[\nabla_t^\top (w_{t+1} - w_t) + \frac{L}{2}\|w_{t+1} - w_t\|^2] && \beta\text{-smoothness} \\ &= -\mathbb{E}[\eta \nabla_t^\top \widetilde{\nabla}_t] + \frac{L}{2}\eta^2 \mathbb{E}\|\widetilde{\nabla}_t\|^2 && \text{algorithm defn.} \\ &= -\eta\|\nabla_t\|^2 + \frac{L}{2}\eta^2\sigma^2 && \text{variance bound.} \end{aligned}$$

Thus, summing up over T iterations, we have for $\eta = \sqrt{\frac{M}{L\sigma^2 T}}$,

$$\begin{aligned} \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T}\|\nabla_t\|^2\right] &\leq \tfrac{1}{T\eta}\textstyle\sum_t \mathbb{E}\left[f(w_t) - f(w_{t+1})\right] + \eta\tfrac{L}{2}\sigma^2 \leq \tfrac{M}{T\eta} + \eta\tfrac{L}{2}\sigma^2 \\ &= \sqrt{\tfrac{ML\sigma^2}{T}} + \tfrac{1}{2}\sqrt{\tfrac{ML\sigma^2}{T}} \leq 2\sqrt{\tfrac{ML\sigma^2}{T}}. \end{aligned}$$

□

We thus conclude that $O(\frac{1}{\epsilon^4})$ iterations are needed to find a point with $\|\nabla f(w)\| \leq \epsilon$. However, each iteration only needs a stochastic estimate of the gradient, and this estimate is a gross upperbound. In practice given the same amount of compute, SGD is much more efficient than GD at reducing loss.

Problem 4.6.3. *Suppose the gradient is estimated using a random sample of B datapoints. (a) Let $\widetilde{\nabla}_t^{(B)}$ be the stochastic gradient at time t when the batchsize is B. Suppose the variance of $\widetilde{\nabla}_t^{(1)}$ (defined as $\mathbb{E}\left[\left\|\widetilde{\nabla}_t^{(1)} - \nabla_t\right\|^2\right]$ is bounded by γ_1^2. Show that there exists an upper bound γ_B^2 on the variance of $\widetilde{\nabla}_t^{(B)}$ that scales with $1/B$. (b) Compute the asymptotic size of T to find a point with $\|\nabla f(w)\| \leq \epsilon$ depending on B and ϵ. For simplicity, you only need to consider the case when $\eta \leq \frac{1}{L}$.*

4.6.4 *Convergence Analysis for Adaptive Algorithms and AdaGrad*

Adaptive methods were described earlier. They require more space to store their parameters, usually 2 or 3 parameters for each of the d coordinates in the gradient. But they can have faster convergence, as well as other myterious properties in deep learning setting that are not mathematically understood. In fact, several of these adaptive algorithms are not guaranteed to converge even for convex loss.

We analyse AdaGrad [12], which was a precursor to modern adaptive algorithms and does have a proof of convergence.

[12] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011

Algorithm 4 AdaGrad

```
for $t = 1$ to $T$ do
    Input: Matrices/scalars $P_t$, as below
    Set
        $$w_{t+1} = w_t - P_t \widehat{\nabla}_t$$
end for
return $w_\tau, \tau \in [T]$ s.t. $\nabla_\tau$ is smallest in Euclidean norm.
```

We start with a simple analysis of an adaptive stepsize first, as per the following theorem. In this section, in addition to the aforementioned notation, we also use the shorthand notation $\nabla_{1:t} = \sum_{i=1}^{t} \nabla_i$, and let $G \geq \|\nabla_t\|$ be an upper bound on the gradient norm.

Theorem 4.6.4. *For unconstrained minimization of L-smooth functions and $P_t = \|\widehat{\nabla}^2_{1:t}\|^{-1} \cdot I$, Algorithm 4 satisfies*

$$\mathbb{E}[\|\nabla_\tau\|^2] \leq \mathbb{E}\left[\frac{1}{T}\sum_t \|\nabla_t\|^2\right] \leq \frac{(L + M\log GT) \cdot \|\widehat{\nabla}^2_{1:t-1}\|}{T}.$$

Proof. From the descent lemma:

$$\begin{aligned} -M &\leq f(w_{T+1}) - f(w_1) \\ &= \textstyle\sum_t (f(w_{t+1}) - f(w_t)) \\ &\leq \textstyle\sum_t (\nabla_t^\top (w_{t+1} - w_t) + \frac{L}{2}\|w_t - w_{t+1}\|^2) \quad \text{smoothness} \\ &\leq \textstyle\sum_t (-\nabla_t^\top P_t \widehat{\nabla}_t + \frac{L}{2}\widehat{\nabla}_t^\top P_t^2 \widehat{\nabla}_t) \end{aligned}$$

Let $P_t = \|\widehat{\nabla}^2_{1:t}\|^{-1}$, and let $\sigma^2 \geq \|\widehat{\nabla}_t\|^2$ be an upper bound on the second moment of the stochastic gradient. Then notice that by the Harmonic series,

$$\sum_t \widehat{\nabla}_t^\top P_t^2 \widehat{\nabla}_t = \sum_t \frac{\|\widehat{\nabla}_t\|^2}{\|\widehat{\nabla}^2_{1:t}\|^2} = \sum_t \frac{\|\widehat{\nabla}_t\|^2}{\sum_{i=1}^t \|\widehat{\nabla}_i^2\|^2} \leq \log GT$$

Using this inequality in the previous derivation, we get that

$$\sum_t \nabla_t^\top P_t \widehat{\nabla}_t \leq M + \frac{L}{2}\log GT.$$

Taking the minimal valued LHS, we get

$$\nabla_\tau^\top \widehat{\nabla}_\tau \cdot P_{\tau-1} \leq \nabla_\tau^\top \widehat{\nabla}_\tau \cdot P_\tau \leq \frac{(L + M\log GT)}{T}.$$

Taking expectation over the unbiased gradient estimator, and shifting sides, we get

$$\|\nabla_\tau\|^2 \leq \frac{(L + M\log GT) \cdot \|\widehat{\nabla}^2_{1:t-1}\|}{T}.$$

□

4.6.5 Adagrad Convergence: Diagonal Matrix case

Theorem 4.6.5. *For unconstrained minimization of L-smooth functions and $P_t = diag(\sum_{i=1}^{t-1} \widehat{\nabla}_i \widehat{\nabla}_i^\top + \sigma^2 I)^{-1/2}$, Algorithm 4 satisfies*

$$\mathbb{E}[\|\nabla_\tau\|^2] \leq \mathbb{E}\left[\frac{1}{T}\sum_t \|\nabla_t\|^2\right] \leq (M + L\log GT) \cdot \frac{\sum_j \sqrt{\widehat{\nabla}^2_{1:t}(j)}}{T}.$$

Proof. From the descent lemma:

$$\begin{aligned} M &\geq f(w_1) - f(w_{T+1}) \\ &= \textstyle\sum_t (f(w_t) - f(w_{t+1})) \\ &\geq \textstyle\sum_t (\nabla_t^\top (w_t - w_{t+1}) - \frac{L}{2}\|w_t - w_{t+1}\|^2) \quad \text{smoothness} \\ &= \textstyle\sum_t (\nabla_t^\top P_t \widehat{\nabla}_t - \frac{L}{2} \widehat{\nabla}_t^\top P_t^2 \widehat{\nabla}_t). \end{aligned}$$

Taking conditional expectation, and the definition of P_t which is conditionally independent of $\widehat{\nabla}_t$, we get

$$\begin{aligned} M &\geq \textstyle\sum_{i=1}^d \left\{\sum_t (\nabla_t^2(i) P_t(i) - \frac{L}{2}\widehat{\nabla}_t^2(i) P_t^2(i))\right\} \\ &\geq \textstyle\sum_{i=1}^d \left\{\sum_t (\nabla_t^2(i) P_t(i) - \frac{L}{2}\log \sigma^2 T)\right\} \\ &\geq \textstyle\max_{i=1}^d \sum_t \nabla_t^2(i) P_t(i) - \frac{L}{2}\log\sigma^2 T, \end{aligned}$$

where the second inequality is due to the Harmonic series,

$$\sum_t \widehat{\nabla}_t^2(i) P_t^2(i) = \sum_t \frac{\widehat{\nabla}_t^2(i)}{\widehat{\nabla}^2_{1:t-1}(i) + \sigma^2} \leq \sum_t \frac{\widehat{\nabla}_t^2(i)}{\widehat{\nabla}^2_{1:t}(i)} \leq \log \sigma^2 T.$$

We conclude that any j,

$$\sum_t \nabla_t^2(j) P_t(j) \leq \max_i \sum_t \nabla_t^2(i) P_t(i) \leq M + \frac{L}{2}\log\sigma^2 T.$$

Let c_j be a random variable which is equal to $\nabla_t^2(j)$ with probability $\frac{1}{T}$. Then the above implies that

$$\mathbb{E}[c_j] \leq \frac{M + \frac{L}{2}\log\sigma^2 T}{T P_t(j)} = (M + L\log GT) \cdot \frac{\sqrt{\widehat{\nabla}^2_{1:t}(j)}}{T}.$$

Thus, summing over the coordinates j, we get

$$\mathbb{E}\,\|\nabla_\tau^2\|^2 = \mathbb{E}[\sum_j c_j] \leq (M + L\log GT) \cdot \frac{\sum_j \sqrt{\widehat{\nabla}^2_{1:t}(j)}}{T}.$$

□

4.7 Discussion: Degree of match between Theory and Practice

The classical optimization theory presented provides a crucial foundation, but its assumptions often clash with the reality of modern deep learning.

Setting the Learning Rate. The theory requires the learning rate $\eta < 2/L$, where L is a global smoothness constant. In practice, L is unknown and varies across the loss landscape. While one could estimate the local smoothness using the power method to find the top eigenvalue of the Hessian (via efficient Hessian-vector products as explained in Section 3.4.1), this is rarely done. Instead, the learning rate is treated as a critical hyperparameter, often tuned via trial and error or decayed according to a schedule (e.g., reduced by a factor of 10 at certain epochs).

The Edge of Stability Phenomenon. Classical theory posits that for the loss to decrease, the learning rate η must be set based on the smoothness L. However, recent work [13] suggests that in deep learning, the causality is reversed: the optimizer, when given a fixed learning rate η, actively seeks out regions of the landscape where the local smoothness L rises to meet the stability limit, i.e., $L \approx 2/\eta$. Training in this "edge of stability" regime, where the loss can oscillate non-monotonically but trends downward over the long term, appears to be correlated with good final model performance. It highlights a key difference between classical convex optimization and large-scale deep learning. We will later see some theoretical attempts at explaining this effect.

[13] Jeremy Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. *ICLR*, 2021

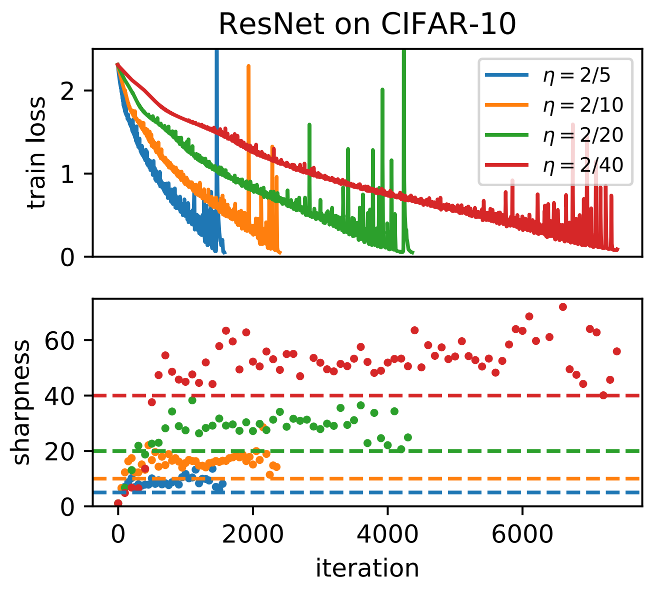

Figure 4.3: **Edge of stability phenomenon.** Figure 5 in Cohen et al. 2021 shows a ResNet trained on $5k$ examples. When doing GD (as opposed to SGD) with a small learning rate η, the smoothness is observed to rise to $2/\eta$ and slightly beyond (figure on right). After this point one sees loss go up and down during iterations, with a long-term downward trend. No theoretical explanation is known as of now. The authors use “sharpness”instead of smoothness, which actually makes some sense because higher L corresponds to a more uneven landscape.