Chapter 1: Optimization and Stochastic Gradient Descent

Mathematical Foundations of Deep Neural Networks

Spring 2024

Department of Mathematical Sciences

Ernest K. Ryu

Seoul National University

Optimization problem

In an *optimization problem*, we minimize or maximize a function value, possibly *subject to* constraints.

Decision variable: θ

Objective function: f

Equality constraint: $h_i(\theta) = 0$ for $i = 1, \dots, m$

Inequality constraint: $g_j(\theta) \leq 0$ for $j = 1, \dots, n$

$$\begin{aligned} \underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad & f(\theta) \\ \text{subject to} \quad & h_1(\theta) = 0 \\ & \vdots \\ & h_m(\theta) = 0 \\ & g_1(\theta) \leq 0 \\ & \vdots \\ & g_n(\theta) \leq 0 \end{aligned}$$

Minimization vs. maximization

In machine learning (ML), we often minimize a “loss”, but sometimes we maximize the “likelihood”.

In any case, minimization and maximization are equivalent since

$$\text{maximize } f(\theta) \quad \Leftrightarrow \quad \text{minimize } -f(\theta)$$

Feasible point and constraints

$\theta \in \mathbb{R}^p$ is a *feasible point* if it satisfies all constraints:

$$\begin{array}{cc} h_1(\theta) = 0 & g_1(\theta) \leq 0 \\ \vdots & \vdots \\ h_m(\theta) = 0 & g_n(\theta) \leq 0 \end{array}$$

Optimization problem is *infeasible* if there is no feasible point.

An optimization problem with no constraint is called an *unconstrained optimization problem*. Optimization problems with constraints is called a *constrained optimization problem.*

Optimal value and solution

Optimal value of an optimization problem is

$$p^\star = \inf \{f(\theta) \mid \theta \in \mathbb{R}^n, \theta \text{ feasible}\}$$

- $p^\star = \infty$ if problem is infeasible
- $p^\star = -\infty$ is possible
- In ML, it is often a priori clear that $0 \leq p^\star < \infty$.

If $f(\theta^\star) = p^\star$, we say $\theta^\star$ is a *solution* or $\theta^\star$ is *optimal*.

- A solution may or may not exist.
- A solution may or may not be unique.

Example: Curve fitting

Consider setup with data $X_1, \dots, X_N$ and corresponding labels $Y_1, \dots, Y_N$ satisfying the relationship

$$Y_i = f_\star(X_i) + \mathrm{error}$$

for $i = 1, \dots, N$. Hopefully, “error” is small. True function $f_\star$ is unknown.

Goal is to find a function (curve) f such that $f \approx f_\star$.

Example: Least-squares

In least-squares minimization, we solve

$$\underset{\theta\in\mathbb{R}^p}{\text{minimize}} \quad \frac{1}{2}\|X\theta - Y\|^2$$

where $X \in \mathbb{R}^{N\times p}$ and $\mathrm{Y} \in \mathbb{R}^N$. Equivalent to

$$\underset{\theta\in\mathbb{R}^p}{\text{minimize}} \quad \frac{1}{2}\sum_{i=1}^{N}\left(X_i^\top\theta - Y_i\right)^2$$

where $X = \begin{bmatrix} X_1^\top \\ \vdots \\ X_N^\top \end{bmatrix}$ and $Y = \begin{bmatrix} Y_1 \\ \vdots \\ Y_N \end{bmatrix}$.

Example: Least-squares

To solve

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad \frac{1}{2}\|X\theta - Y\|^2$$

take grad and set it to 0:

$$X^\top(X\theta^\star - Y) = 0$$
$$\theta^\star = (X^\top X)^{-1} X^\top Y$$

Here, we assume $X^\top X$ is invertible.

Make sure you understand why

$$\nabla_\theta \frac{1}{2}\|X\theta - Y\|^2 = X^\top(X\theta - Y)$$

LS is an instance of curve fitting

How is LS curve fitting? Define $f_\theta(x) = x^\top \theta$. Then LS becomes

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad \frac{1}{2}\sum_{i=1}^{N} (f_\theta(X_i) - Y_i)^2$$

and the solution hopefully satisfies

$$Y_i = f_\theta(X_i) + \text{small}.$$

Since X_i and Y_i is assumed to satisfy

$$Y_i = f_\star(X_i) + \text{error}$$

we are searching over linear functions (linear curves) f_θ that best fit (approximate) $f_\star$.

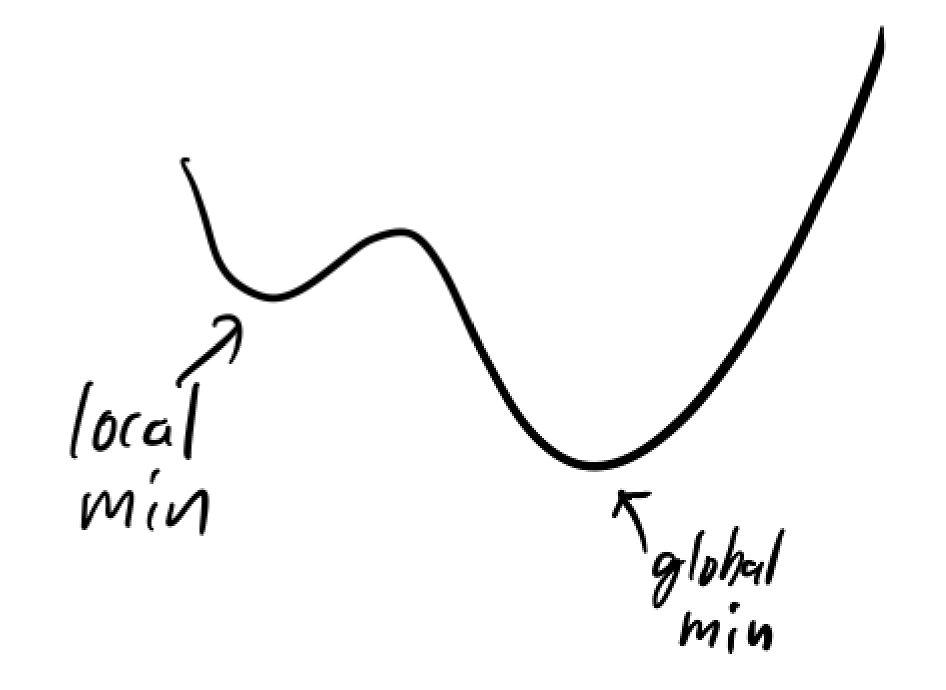
Local vs. global minima

$\theta^\star$ is a *local minimum* if $f(\theta) \geq f(\theta^\star)$ for all feasible θ within a small neighborhood.

$\theta^\star$ is a *global minimum* if $f(\theta) \geq f(\theta^\star)$ for all feasible θ.

In the worst case, finding the global minimum of an optimization problem is difficult*.

However, in deep learning, optimization problems are often “solved” without any guarantee of global optimality.



*The class of smooth non-convex optimization problems is NP-hard.

Gradient descent

Consider the unconstrained optimization problem

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad f(\theta)$$

where f is differentiable.

Gradient Descent (GD) algorithm:

$$\theta^{k+1} = \theta^k - \alpha_k \nabla f\left(\theta^k\right) \quad \text{for } k = 0, 1, \ldots,$$

where $\theta^0 \in \mathbb{R}^p$ is the *initial point* and $\alpha_k > 0$ is the *learning rate* or the *stepsize*.

The terminology *learning rate* is common the machine learning literature while *stepsize* is more common in the optimization literature.

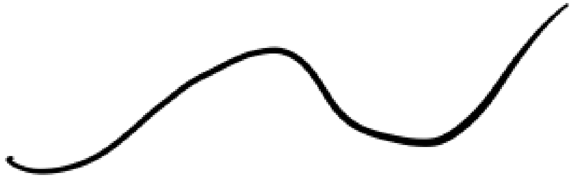
Definition of “differentiability”

In math, a function is “differentiable” if its derivative exists everywhere.

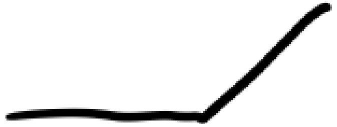
In deep learning (DL), a function is often said to be differentiable if its derivative exists almost everywhere and the function is nice*. ReLU activation functions are said to be differentiable.

We won’t be too concerned with this distinction.

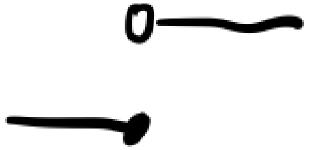
Differentiable in DL & Math



Differentiable in DL but not in Math



Not differentiable in DL or Math



*So no weird functions like the Cantor function. Absolute continuity probably captures the DL scholars’ working definition of “differentiability”.

Why does GD converge?

$$\theta^{k+1} = \theta^k - \alpha_k \nabla f\left(\theta^k\right)$$

Taylor expansion of f about θ^k:

$$f(\theta) = f\left(\theta^k\right) + \nabla f\left(\theta^k\right)^\top \left(\theta - \theta^k\right) + \mathcal{O}\left(\left\|\theta - \theta^k\right\|^2\right)$$

Plug in θ^{k+1}:

$$f\left(\theta^{k+1}\right) = f\left(\theta^k\right) - \alpha_k \left\|\nabla f\left(\theta^k\right)\right\|^2 + \mathcal{O}\left(\alpha_k^2\right)$$

$-\nabla f\left(\theta^k\right)$ is steepest descent direction. For small (cautious) α_k, GD step reduces function value.

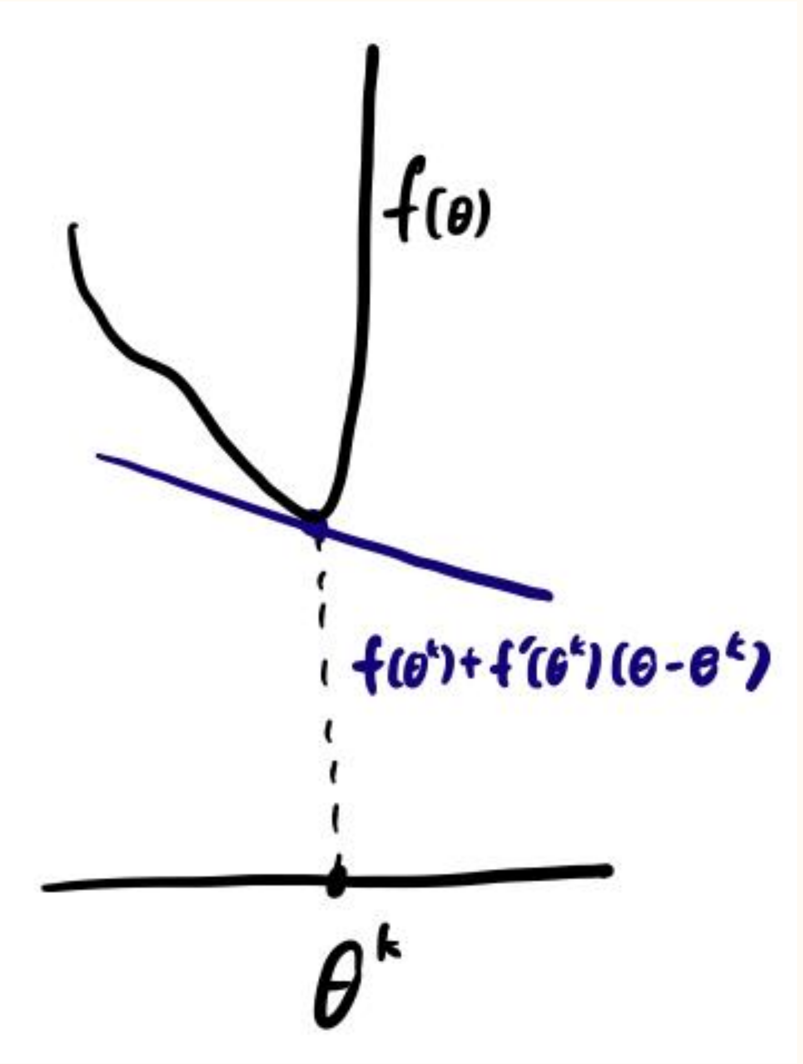
Is GD a “descent method”?

$$\theta^{k+1} = \theta^{k} - \alpha_k \nabla f\left(\theta^{k}\right)$$

Without further assumptions, $-\nabla f\left(\theta^{k}\right)$ only gives you directional information. How far should you go? How large should α_k be?

A step of GD need not result in descent, i.e., $f\left(\theta^{k+1}\right) > f\left(\theta^{k}\right)$ is possible.

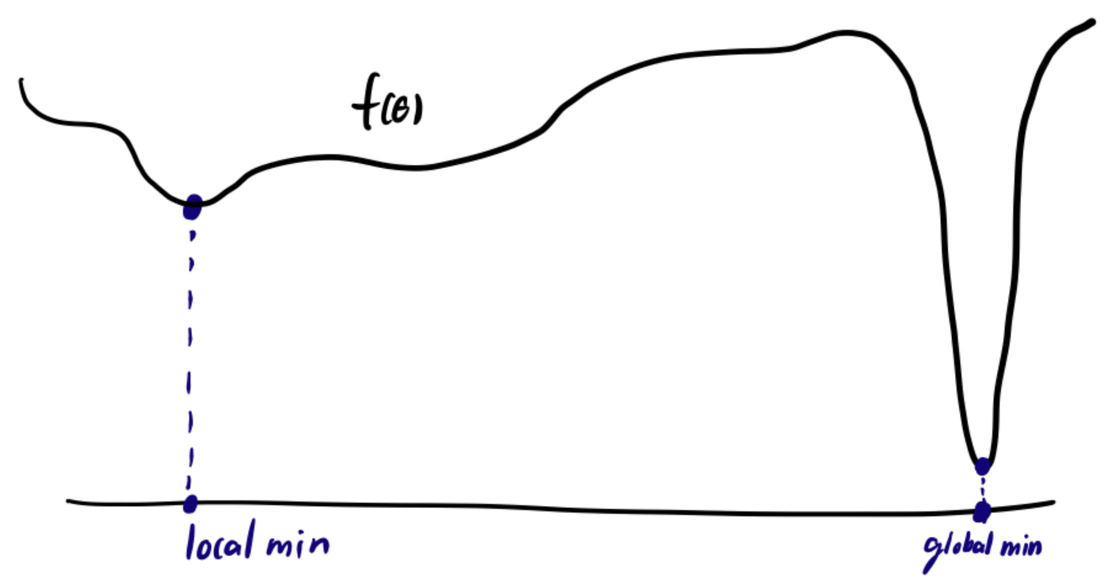
We need an assumption that ensures the first-order Taylor expansion is a good approximation within a sufficiently large neighborhood.



What can we prove?

Without further assumptions, there is no hope of finding the global minimum.

We cannot prove the function value converges to global optimum. We instead prove $\nabla f(\theta^k) \to 0$. Roughly speaking, this is similar, but weaker than proving that θ^k converges to a local minimum.*



*Without further assumptions, we cannot show that θ^k converges to a limit, and even θ^k does converge to a limit, we cannot guarantee that that limit is not a saddle point or even a local maximum. Nevertheless, people commonly use the argument that θ^k usually converges and that it is unlikely that the limit is a local maximum or a saddle point.

Convergence of GD

Theorem) Assume $f: \mathbb{R}^p \to \mathbb{R}$ is differentiable, ∇f is L-Lipschitz continuous, and $\inf_{\theta \in \mathbb{R}^p} f(\theta) > -\infty$. Then

$$\theta^{k+1} = \theta^k - \alpha \nabla f\left(\theta^k\right)$$

with $\alpha \in \left(0, \frac{2}{L}\right)$ satisfies $\nabla f\left(\theta^k\right) \to 0$.

Lipschitz gradient lemma

We say $\nabla f : \mathbb{R}^p \to \mathbb{R}^p$ is L-Lipschitz if

$$\|\nabla f(x) - \nabla f(y)\| \le L\|x - y\| \quad \forall x, y \in \mathbb{R}^p.$$

Roughly, this means ∇f does not change rapidly. As a consequence, we can trust the first-order Taylor expansion on a non-infinitesimal neighborhood.

Lemma) Let $f : \mathbb{R}^p \to \mathbb{R}$ be differentiable and $\nabla f : \mathbb{R}^p \to \mathbb{R}^p$ be L-Lipschitz. Then

$$f(\theta + \delta) \le f(\theta) + \nabla f(\theta)^\top \delta + \frac{L}{2}\|\delta\|^2 \quad \forall \theta, \delta \in \mathbb{R}^p.$$

$f(\theta) + \nabla f(\theta)^\top \delta - \frac{L}{2}\|\delta\|^2 \le f(\theta + \delta)$ is also true, but we do not need this other direction. Together the inequalities imply

$$|f(\theta + \delta) - (f(\theta) + \nabla f(\theta)^\top \delta)| \le \frac{L}{2}\|\delta\|^2 \quad \forall \theta, \delta \in \mathbb{R}^p.$$

(I don’t think this proof is important enough to cover in class, but I provide it here for completeness.)

Proof) Define $g\colon \mathbb{R} \to \mathbb{R}$ as $g(t) = f(\theta + t\delta)$. Then g is differentiable and

$$g'(t) = \nabla f(\theta + t\delta)^\top \delta.$$

Note g' is $(L\|\delta\|^2)$-Lipschitz continuous since

$$\begin{aligned} |g'(t_1) - g'(t_0)| &= \left|\left(\nabla f(\theta + t_1\delta) - \nabla f(\theta + t_0\delta)\right)^\top \delta\right| \\ &\leq \|\nabla f(\theta + t_1\delta) - \nabla f(\theta + t_0\delta)\|\|\delta\| \\ &\leq L\|t_1\delta - t_0\delta\|\|\delta\| \\ &= L\|\delta\|^2|t_1 - t_0| \end{aligned}$$

Finally, we conclude with

$$\begin{aligned} f(\theta + \delta) = g(1) &= g(0) + \int_0^1 g'(t)\,\mathrm{d}t \\ &\leq f(\theta) + \int_0^1 (g'(0) + L\|\delta\|^2 t)\,\mathrm{d}t \\ &= f(\theta) + \nabla f(\theta)^\top \delta + \frac{L}{2}\|\delta\|^2 \end{aligned}$$

■

Summability Lemma

Lemma) Let $V^0, V^1, \ldots \in \mathbb{R}$ and $S^0, S^1, \ldots \in \mathbb{R}$ be nonnegative sequences satisfying

$$V^{k+1} \leq V^k - S^k$$

for $k = 0, 1, 2, \ldots$. Then $S^k \to 0$.

Key idea. S^k measures progress (decrease) made in iteration k. Since $V^k \geq 0$, V^k cannot decrease forever, so the progress (magnitude of S^k) must diminish to 0.

Proof) Sum the inequality from $i = 0$ to k

$$V^{k+1} + \sum_{i=0}^{k} S^i \leq V^0$$

Let $k \to \infty$

$$\sum_{i=0}^{\infty} S^i \leq V^0 - \lim_{k\to\infty} V^k \leq V^0$$

Since $\sum_{i=0}^{\infty} S^i < \infty$, $S^i \to 0$ ■

Convergence of GD: Proof

Theorem) Under the assumptions, if $\theta^{k+1} = \theta^k - \alpha \nabla f\left(\theta^k\right)$ and $\alpha \in \left(0, \frac{2}{L}\right)$, then $\nabla f\left(\theta^k\right) \to 0$.

Proof) Use Lipschitz gradient lemma with $\theta = \theta^k$ and $\delta = -\alpha \nabla f\left(\theta^k\right)$ to get

$$f\left(\theta^{k+1}\right) \leq f\left(\theta^k\right) - \alpha\left(1 - \frac{\alpha L}{2}\right)\left\|\nabla f\left(\theta^k\right)\right\|^2$$

and

$$\underbrace{\left(f\left(\theta^{k+1}\right) - \inf_\theta f(\theta)\right)}_{\geq 0} \leq \underbrace{\left(f\left(\theta^k\right) - \inf_\theta f(\theta)\right)}_{\geq 0} - \underbrace{\alpha\left(1 - \frac{\alpha L}{2}\right)}_{> 0 \text{ for } \alpha \in \left(0, \frac{2}{L}\right)}\left\|\nabla f\left(\theta^k\right)\right\|^2$$

By the summability lemma, $\left\|\nabla f\left(\theta^k\right)\right\|^2 \to 0$ and thus $\nabla f\left(\theta^k\right) \to 0$.

■

Purpose of GD convergence analysis

In deep learning, the condition that ∇f is L-Lipschitz is usually not true*.

Rather, the purpose of these mathematical analyses is to obtain qualitative insights; this convergence proof and the exercises of hw1 are meant to provide you with intuition on the training dynamics of GD and SGD.

Because analyzing deep learning systems as is rigorously is usually difficult, people usually

- analyze modified (simplified) setups rigorously or
- analyze the full setup heuristically.

In both cases, the goal is to obtain qualitative insights, rather than theoretical guarantees.

*Due to the use of ReLU activation functions.

Finite-sum optimization problems

A *finite-sum optimization problem* has the structure

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N} f_i(\theta) \coloneqq F(\theta)$$

Finite-sum is ubiquitous in ML. N usually corresponds to the number of data points.

Using GD

$$\theta^{k+1} = \theta^k - \frac{\alpha_k}{N}\sum_{i=1}^{N} \nabla f_i\left(\theta^k\right)$$

is impractical when N is large since $\frac{1}{N}\sum_{i=1}^{N}\cdot$ takes too long to compute.

Finite-sum ≅ Expectation

Although the finite-sum optimization problem has no inherent randomness, we can reformulate this problem with randomness:

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad \mathbb{E}_I[f_I(\theta)]$$

where $I \sim \text{Uniform}\{1, \ldots, N\}$. To see the equivalence,

$$\mathbb{E}_I[f_I(\theta)] = \sum_{i=1}^{N} f_i(\theta)\mathbb{P}(I = i) = \frac{1}{N}\sum_{i=1}^{N} f_i(\theta) = F(\theta)$$

Stochastic gradient descent (SGD)

Stochastic gradient descent (SGD)

$$i(k) \sim \mathrm{Uniform}\{1, \dots, N\}$$
$$\theta^{k+1} = \theta^{k} - \alpha_k \nabla f_{i(k)}(\theta^{k})$$

for $k = 0,1, \dots,$ where $\theta^{0} \in \mathbb{R}^{p}$ is the *initial point* and $\alpha_k > 0$ is the *learning rate*.

$\nabla f_{i(k)}\left(\theta^{k}\right)$ is a *stochastic gradient* of F at θ^{k}, i.e.,

$$\mathbb{E}\left[\nabla f_{i(k)}\left(\theta^{k}\right)\right] = \nabla \mathbb{E}\left[f_{i(k)}\left(\theta^{k}\right)\right] = \nabla F\left(\theta^{k}\right)$$

GD vs. SGD

GD uses all indices $i = 1, \ldots, N$ every iteration

$$\theta^{k+1} = \theta^k - \frac{\alpha_k}{N} \sum_{i=1}^{N} \nabla f_i\left(\theta^k\right)$$

SGD uses only a single random index $i(k)$ every iteration

$$i(k) \sim \mathrm{Uniform}\{1, \ldots, N\}$$
$$\theta^{k+1} = \theta^k - \alpha_k \nabla f_{i(k)}(\theta^k)$$

When size of the data N is large, SGD is often more effective than GD.

Digression: Randomized algorithms

A *randomized algorithm* utilizes artificial randomness to solve an otherwise deterministic problem.

There are problems* for which a randomized algorithm is faster than the best known deterministic algorithm.

The most famous example of this is SGD in deep learning.

* The second most famous example is testing whether a given number is prime.

Why does SGD converge?

Plug θ^{k+1} into Taylor expansion of F about θ^k:

$$F\left(\theta^{k+1}\right) = F\left(\theta^k\right) - \alpha_k \nabla F\left(\theta^k\right)^\top \nabla f_{i(k)}\left(\theta^k\right) + \mathcal{O}\left(\alpha_k^2\right)$$

Take expectation on both sides:

$$\mathbb{E}_k\left[F\left(\theta^{k+1}\right)\right] = F\left(\theta^k\right) - \alpha_k \left\|\nabla F\left(\theta^k\right)\right\|^2 + \mathcal{O}\left(\alpha_k^2\right)$$

($\mathbb{E}_k$ is expectation conditioned on θ^k)

$-\nabla f_{i(k)}\left(\theta^k\right)$ is descent direction *in expectation*. For small (cautious) α_k, SGD step reduces function value *in expectation*.

Variants of SGD for finite-sum problems

Consider

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^{N} f_i(\theta)$$

SGD can be generalized to

$$\theta^{k+1} = \theta^k - \alpha_k g^k,$$

where g^k is a stochastic gradient. The choice $g^k = \nabla f_{i(k)}\left(\theta^k\right)$ is just one option.

Sampling with replacement lemma

Lemma) Let $X_1, \dots, X_N \in \mathbb{R}^p$ be given (non-random) vectors. Let $\frac{1}{N}\sum_{i=1}^{N} X_i = \mu$. Let $i(1), \dots, i(B) \subseteq \{1, \dots, N\}$ be random indices. Then

$$\mathbb{E}\frac{1}{B}\sum_{b=1}^{B} X_{i(b)} = \mu$$

Proof)

$$\mathbb{E}\frac{1}{B}\sum_{b=1}^{B} X_{i(b)} = \frac{1}{B}\sum_{b=1}^{B} \mathbb{E}X_{i(b)} = \frac{1}{B}\sum_{b=1}^{B} \mu = \mu$$

■

Minibatch SGD with replacement

Minibatch SGD with replacement

$$i(k,1),\dots,i(k,B)\sim\mathrm{Uniform}\{1,\dots,N\}$$

$$\theta^{k+1}=\theta^{k}-\frac{\alpha_k}{B}\sum_{b=1}^{B}\nabla f_{i(k,b)}\left(\theta^{k}\right)$$

To clarify, we sample B out of N indices with replacement, i.e., the same index can be sampled multiple times.

By previous lemma, $\frac{1}{B}\sum_{b=1}^{B}\nabla f_{i(k,b)}\left(\theta^{k}\right)$ is a stochastic gradient of F at θ^{k}

Random permutations

A *permutation* σ is a list of length N containing integers $1, \dots, N$ all exactly once. We write S_n for the set of permutations of length N.

There are $N!$ possible permutations of length N.

A *random permutation* is a permutation chosen randomly with uniform probability; each of the $N!$ permutations are chosen with probability $\frac{1}{N!}$.

The $N!$ count comes from a basic counting argument. A random permutation can be efficiently sampled using the Fisher–Yates–Knuth shuffle algorithm.

Digression: 0-based indexing and random permutations in Python

In Python, generate random permutations with

```
np.random.permutation(np.arange(N))
```

In Python, array indices start at 0, although in math and in human language, counting starts at 1. We use permutations containing $0,1,\ldots,N-1$ in our Python code.

Sampling without replacement lemma

Lemma) Let $X_1, \dots, X_N \in \mathbb{R}^p$ be given (non-random) vectors. Let $\frac{1}{N}\sum_{i=1}^{N} X_i = \mu$. Let σ be a random permutation. Then

$$\mathbb{E}\frac{1}{B}\sum_{b=1}^{B} X_{\sigma(b)} = \mu$$

Proof)

$$\mathbb{E}\frac{1}{B}\sum_{b=1}^{B} X_{\sigma(b)} = \frac{1}{B}\sum_{b=1}^{B} \mathbb{E}X_{\sigma(b)} = \frac{1}{B}\sum_{b=1}^{B} \mu = \mu$$

■

Minibatch SGD without replacement

Minibatch SGD without replacement

$$\sigma^k \sim \mathrm{permutation}(N)$$

$$\theta^{k+1} = \theta^k - \frac{\alpha_k}{B} \sum_{b=1}^{B} \nabla f_{\sigma^k(b)}(\theta^k)$$

We assume $B \leq N$. To clarify, we sample B out of N indices without replacement, i.e., the same index cannot be sampled multiple times.

By previous lemma, $\frac{1}{B}\sum_{b=1}^{B} \nabla f_{\sigma^k(b)}(\theta^k)$ is a stochastic gradient of F at θ^k.

How to choose batch size B?

Note $B = 1$ minibatch SGD becomes SGD.

Mathematically (measuring performance per iteration)

- Use large batch is when noise/randomness is large.
- Use small batch is when noise/randomness is small.

Practically (measuring performance per unit time)

- Large batch allows more efficient computation on GPUs.
- Often best to increase batch size up to the GPU memory limit.

GD and SGD without differentiability

In DL, SGD is applied to nice continuous but non-differentiable* functions that are differentiable almost everywhere.

In this case, if we choose $\theta^0 \in \mathbb{R}^n$ randomly and run

$$\theta^{k+1} = \theta^k - \alpha_k \nabla f(\theta^k)$$

the algorithm is usually well-defined, i.e., θ^k never hits a point of non-differentiability.

With a proof or not, GD and SGD are applied to non-differentiable minimization in ML. The absence of differentiability* does not seem to cause serious problems.

*So these are neural networks built with ReLU activation functions.

Cyclic SGD

Consider the sequence of indices

$$\{\mathrm{mod}(k, N) + 1\}_{k=0,1,\ldots} = 1, 2, \ldots, N, 1, 2, \ldots, N, \ldots$$

Here, $\mathrm{mod}(k, N)$ is the remainder of k when divided by N. In Python, this is written with `k%N`.

Cyclic SGD:

$$\theta^{k+1} = \theta^{k} - \alpha_{\mathrm{k}} \nabla f_{\mathrm{mod}(k,N)+1}(\theta^{k})$$

To clarify, this samples the indices in a (deterministic) cyclic order.

Cyclic (mini-batch) SGD

Strictly speaking, cyclic SGD is not an instance of SGD as unbiased estimation property lost.

Advantage:

- Uses all indices (data) every N iterations.

Disadvantage:

- Worse than SGD in some cases, theoretically and empirically.
- In DL, neural networks can learn to anticipate cyclic order.

Shuffled Cyclic SGD

Shuffled Cyclic SGD:

$$\theta^{k+1} = \theta^k - \alpha_k \nabla f_{\sigma^{\lfloor \frac{k}{N} \rfloor}(\mathrm{mod}(k,N)+1)}(\theta^k)$$

where $\sigma^0, \sigma^1, \ldots$ is a sequence of random permutations, i.e., we shuffle the order every cycle. Again, strictly speaking, shuffled cyclic SGD is not an instance of SGD as unbiased estimation property lost.

Advantages :

- Uses all indices (data) every N iterations.
- Neural network cannot learn to anticipate data order.
- Empirically best performance.

Disadvantages:

- Theory not as strong as regular SGD.

Which variant of SGD to use?

Theoretical comparison of SGD variants:

- Not that easy.
- Result does not strongly correlate with practical performance in DL.

In DL, the most common choice is

- shuffled cyclic minibatch SGD (without replacement) and
- batchsize B is as large as possible within the GPU memory limit.

One can generally consider this to be the default option.

Epoch in finite-sum optimization and machine learning training

An *epoch* is loosely defined as the unit of optimization or training progress of processing all indices or data once.

- 1 iteration of GD constitutes an epoch.
- N iterations of SGD, cyclic SGD, or shuffled cyclic SGD constitute an epoch.
- N/B iterations of minibatch SGD constitute an epoch.

Epoch is often a convenient unit for counting iterations compared to directly counting the iteration number.

SGD with general expectation

Consider an optimization problem with its objective defined with a general expectation

$$\underset{\theta\in\mathbb{R}^p}{\text{minimize}} \quad \mathbb{E}_\omega\left[f_\omega(\theta)\right] \coloneqq F(\theta)$$

Here, ω is a random variable. We will encounter these expectations (non-finite sum) when we talk about generative models.

For this setup, the SGD algorithm is

$$\theta^{k+1} = \theta^k - \alpha_k \nabla f_{\omega^k}(\theta^k)$$

where $\omega^0, \omega^1, \ldots$ are IID random samples of ω. If $\nabla_\theta \mathbb{E}_\omega[f_\omega(\theta)] = \mathbb{E}_\omega[\nabla_\theta f_\omega(\theta)]$, then $\nabla f_{\omega^k}(\theta^k)$ is a stochastic gradient of $F(\theta)$ at θ^k. (Make sure you understand why the previous SGD for the finite-sum setup is a special case of this.)

GD for this setup is

$$\theta^{k+1} = \theta^k - \alpha_k \mathbb{E}_\omega\left[\nabla_\theta f_\omega\left(\theta^k\right)\right]$$

However, if the expectation is difficult to compute GD is impractical and SGD is preferred.