Stochastic Gradient Descent (SGD)

The Algorithm: Sum-Structured Objectives

The Problem Setting

Many objective functions in machine learning are structured as a sum:

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x)$$

- f_i is typically the loss function for the i-th data point.
- n is the total number of data points in the training set.

Stochastic Gradient Descent (SGD) Iteration

An iteration of SGD is defined as:

1. Sample $i \in \{1, \ldots, n\}$ uniformly at random.
2. Update x using only the gradient of f_i:

$$x_{t+1} := x_t - \gamma_t \nabla f_i(x_t)$$

The SGD Algorithm: Efficiency

The Advantage of SGD

Efficiency per iteration.

- **Full Gradient (GD):** To compute $\nabla f(x_t) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x_t)$, we must compute n individual gradients.
- **Stochastic Gradient (SGD):** An iteration requires computing only a *single* gradient, $\nabla f_i(x_t)$.
- This makes each SGD iteration n times cheaper than a full GD iteration.

The Core Question

Do these much cheaper iterations still provide meaningful progress toward the minimum?
(Spoiler: Yes, but with trade-offs.)

Unbiasedness of the Stochastic Gradient I

Unbiased Estimator

The update vector $g_t := \nabla f_i(x_t)$ is called a **stochastic gradient**.
This stochastic gradient is an **unbiased estimator** of the true gradient $\nabla f(x)$.
In expectation, over the random choice of i, g_t equals the full gradient:

$$\mathbb{E}[g_t | x_t = x] = \mathbb{E}_i[\nabla f_i(x)] = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x) = \nabla f(x)$$

Unbiasedness of the Stochastic Gradient II

Consequence for Analysis

The key inequality for convex functions, $f(x_t) - f(x^*) \leq \nabla f(x_t)^\top (x_t - x^*)$, may not hold for the *stochastic* gradient g_t.
However, it **does hold in expectation**.

$$\begin{aligned} \mathbb{E}[g_t^\top (x_t - x^*)] &= \mathbb{E}[\nabla f(x_t)^\top (x_t - x^*)] \\ &\geq \mathbb{E}[f(x_t) - f(x^*)] \quad (\diamond) \end{aligned}$$

This is the crucial observation that allows the analysis of SGD.

Convergence: Bounded Gradients

Theorem (Lipschitz-like)

Let f be convex and differentiable with global minimum x^*. Assume:

1. $||x_0 - x^*|| \leq R$
2. $\mathbb{E}[||g_t||^2] \leq B^2$ *for all t (Bounded expected squared norm)*

Choosing a constant stepsize $\gamma := \frac{R}{B\sqrt{T}}$, SGD yields:

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[f(x_t)] - f(x^*) \leq \frac{RB}{\sqrt{T}}$$

This implies an $\mathcal{O}(1/\epsilon^2)$ step complexity.

Convergence: Bounded Gradients. Proof I

Start basic analysis from GD and take expectations:

$$\sum_{t=0}^{T-1} \mathbb{E}[g_t^\top (x_t - x^*)] \leq \frac{\gamma}{2} \sum_{t=0}^{T-1} \mathbb{E}[||g_t||^2] + \frac{1}{2\gamma} ||x_0 - x^*||^2$$

Use our three key facts:

1. $\mathbb{E}[f(x_t) - f(x^*)] \leq \mathbb{E}[g_t^\top (x_t - x^*)]$ (from $\diamond$)
2. $\mathbb{E}[||g_t||^2] \leq B^2$ (Assumption)
3. $||x_0 - x^*||^2 \leq R^2$ (Assumption)

Plugging these in gives:

$$\sum_{t=0}^{T-1} \mathbb{E}[f(x_t) - f(x^*)] \leq \frac{\gamma}{2} B^2 T + \frac{1}{2\gamma} R^2$$

The proof finishes by choosing the optimal $\gamma = \frac{R}{B\sqrt{T}}$ to minimize the RHS. This yields $\sum_{t=0}^{T-1} \mathbb{E}[f(x_t) - f(x^*)] \leq RB\sqrt{T}$. Dividing by T gives the result.

Convergence: Strong Convexity I

Theorem (Strong Convexity)

Let f be differentiable and strongly convex with parameter $\mu > 0$. Assume $\mathbb{E}[||g_t||^2] \leq B^2$ for all t.
With a decreasing stepsize $\gamma_t := \frac{2}{\mu(t+1)}$, SGD yields:

$$\mathbb{E}\left[f\left(\frac{2}{T(T+1)}\sum_{t=1}^{T} t \cdot x_t\right) - f(x^*)\right] \leq \frac{2B^2}{\mu(T+1)}$$

This implies an $\mathcal{O}(1/\epsilon)$ step complexity (for the averaged iterate).

Convergence: Strong Convexity. Proof I

(1). Start from basic analysis and take expectations:

$$\mathbb{E}[g_t^\top (x_t - x^*)] = \frac{\gamma_t}{2}\mathbb{E}[||g_t||^2] + \frac{1}{2\gamma_t}(\mathbb{E}[||x_t - x^*||^2] - \mathbb{E}[||x_{t+1} - x^*||^2])$$

(2). Use ($\diamond$) and strong convexity for the LHS:

$$\mathbb{E}[g_t^\top (x_t - x^*)] = \mathbb{E}[\nabla f(x_t)^\top (x_t - x^*)]$$

$$\geq \mathbb{E}[f(x_t) - f(x^*)] + \frac{\mu}{2}\mathbb{E}[||x_t - x^*||^2]$$

(3). Combine (1) and (2), and use $\mathbb{E}[||g_t||^2] \leq B^2$:

$$\mathbb{E}[f(x_t) - f(x^*)] \leq \frac{B^2\gamma_t}{2} + \frac{(\gamma_t^{-1} - \mu)}{2}\mathbb{E}[||x_t - x^*||^2] - \frac{\gamma_t^{-1}}{2}\mathbb{E}[||x_{t+1} - x^*||^2]$$

Convergence: Strong Convexity. Proof II

Theorem (Jensen's inequality)

Let $f : \mathbb{R}^d \to \mathbb{R}$ *be a convex function,* $x_1, ..., x_m \in dom(f)$ *and* $\lambda_1, ..., \lambda_m \in R_+$ *such that* $\sum_{i=1}^m \lambda_i = 1$. *Then*

$$f\left(\sum_{i=1}^m \lambda_i x_i\right) \leq \sum_{i=1}^m \lambda_i f(x_i)$$

4. Plug in $\gamma_t = \frac{2}{\mu(t+1)}$, multiply by t, sum, and applying the Jensen's inequality we get the thesis.

How to Sample: With or Without Replacement? I

Sampling With Replacement

- **What it is:** In each step, pick an $i \in \{1, \dots, n\}$ uniformly at random.
- **Analysis:** This is the standard assumption used in the proofs.
- **Advantage:** The samples $(g_t, g_{t+1}, \dots)$ are all independent and identically distributed (i.i.d.). This makes the analysis much simpler.
- **Drawback:** In one "epoch" (pass through n samples), it's very likely you miss some data points and sample others multiple times.

How to Sample: With or Without Replacement? II

Sampling Without Replacement ("Random Reshuffling")

- **What it is:**
  1. Create a random permutation (shuffle) of the n data indices.
  2. Pass through the data in that shuffled order.
  3. When all n are used, re-shuffle and start a new epoch.
- **This is what is done in practice!**
- **Advantage:** Guarantees every data point is used exactly once per epoch. Empirically, it converges faster.
- **Drawback:** The samples within an epoch are *not* independent. The analysis is much, much harder.

The Goal in Machine Learning I

Optimization vs. Generalization

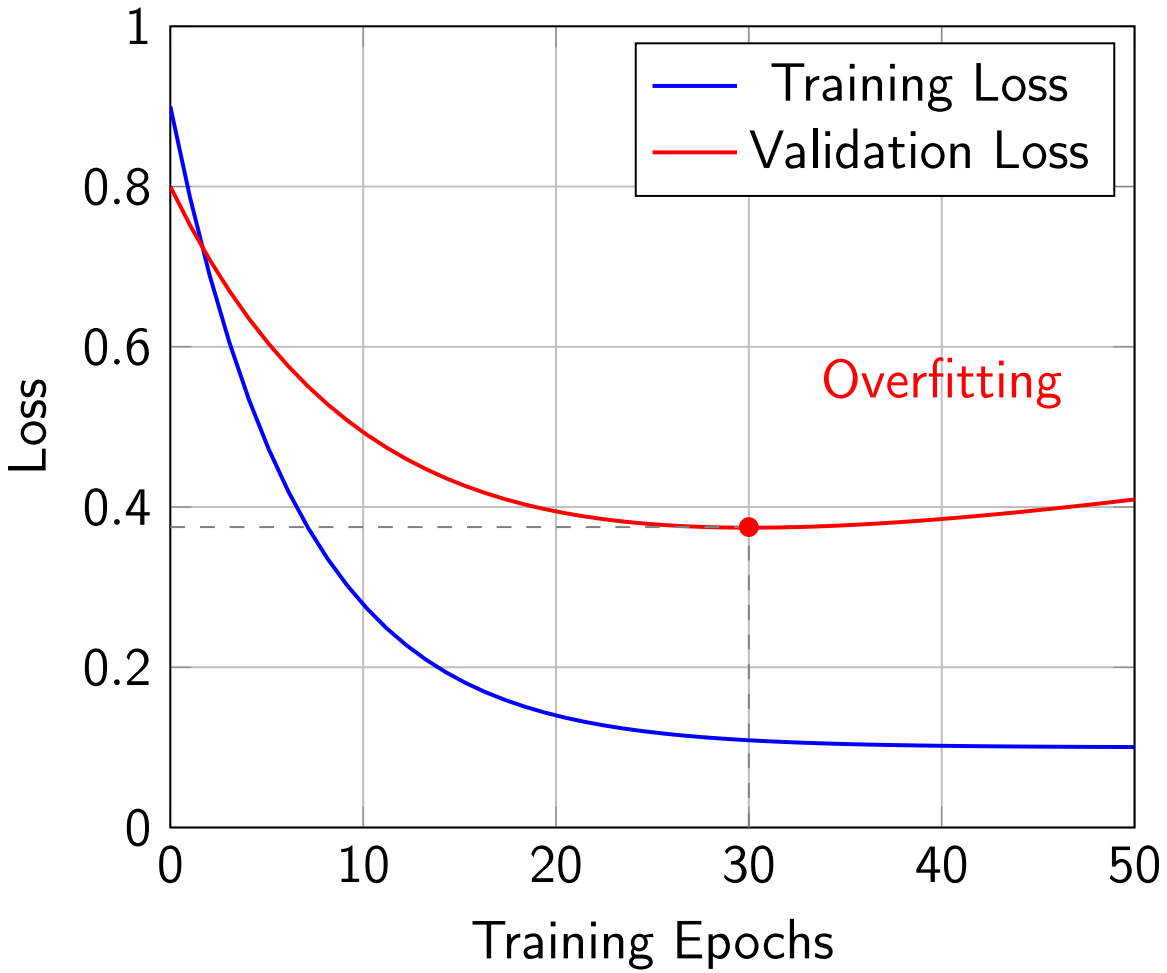
- In pure optimization, we want to find the *exact* minimum of the given function $f(x)$.
- In Machine Learning, $f(x)$ is the **training loss**. We don't care about minimizing it perfectly.
- We really care about the **test loss** (or "generalization error"): how the model performs on new, unseen data.
- Minimizing the training loss *perfectly* can lead to **overfitting**, where the model memorizes the training data and fails to generalize.

The Goal in Machine Learning II

Early Stopping

- Because of this, we almost never train SGD until convergence.
- We monitor the loss on a separate **validation set** (data not used for training).
- We **stop training** when the validation loss stops improving (or starts getting worse), even if the training loss is still going down.
- This "early stopping" acts as a powerful regularizer.

Visualization: Early Stopping



Example: $f(x) = \frac{1}{2N} \sum_{i=1}^{N} (a_i x - b_i)^2$ I

The Setup (1D Linear Regression)

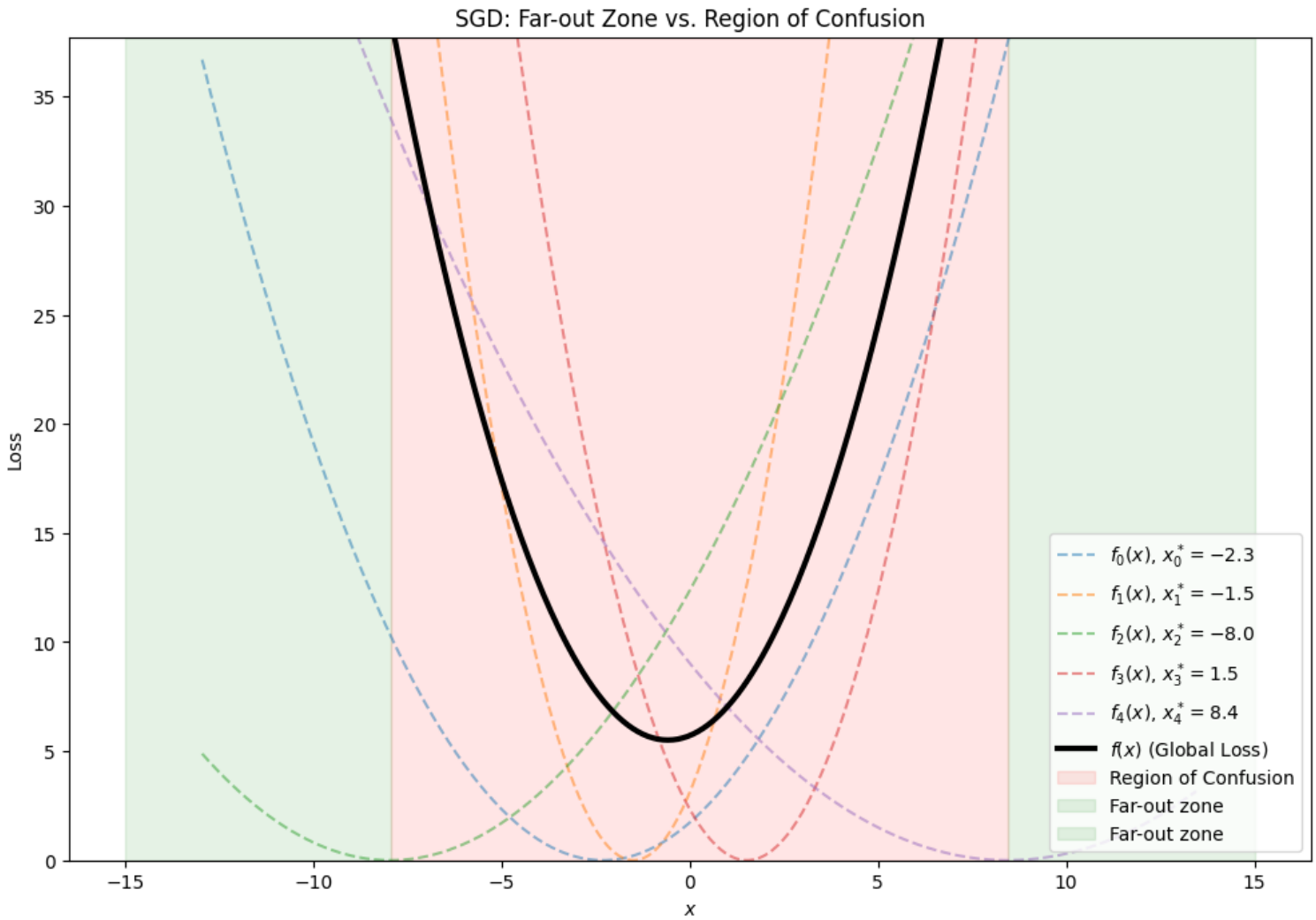
- Let $f_i(x) = \frac{1}{2}(a_i x - b_i)^2$. This is a single parabola.
- The minimum of f_i is at $x_i^* = b_i / a_i$.
- The full-batch loss $f(x)$ is the *average* of all these parabolas.
- The global minimum x^* is at $\nabla f(x^*) = 0$, which gives $x^* = \frac{\sum a_i b_i}{\sum a_i^2}$.

Example: $f(x) = \frac{1}{2N} \sum_{i=1}^{N} (a_i x - b_i)^2$ II

SGD Dynamics: Two Zones

1. **Far-out Zone:**
   - When x_t is far from all x_i^*, all individual gradients $\nabla f_i(x_t) = a_i(a_i x_t - b_i)$ point in roughly the same direction (e.g., all negative).
   - The stochastic gradient g_t is a good approximation of the true gradient $\nabla f(x_t)$.
   - SGD makes fast, consistent progress, much like full-batch GD.
2. **Region of Confusion:**
   - When x_t is near the global minimum x^*, it lies *between* the individual x_i^*.
   - Some $\nabla f_i(x_t)$ will be positive, others will be negative. They "fight" each other.
   - The stochastic gradient g_t has high variance. $\mathbb{E}[g_t] \approx 0$, but g_t itself is large.
   - SGD cannot converge to x^* (with constant γ) and will "bounce around" this region. ▸ Link ▸ Colab

Visualization: Region of Confusion



$$\mathbb{E}[\|x_t - x^*\|^2] \leq (1 - 2\gamma\mu)^t \|x_0 - x^*\|^2 + \frac{\gamma B^2}{2\mu}$$

Mini-Batch SGD

The Algorithm

A compromise between full-batch (GD) and single-sample (SGD). Instead of one sample, we average m samples (a "mini-batch"):

$$\tilde{g}_t := \frac{1}{m} \sum_{j=1}^{m} \nabla f_{i_j}(x_t)$$

where i_j are m distinct, randomly chosen indices.
The update step is then:

$$x_{t+1} := x_t - \gamma_t \tilde{g}_t$$

- $m = 1$ is standard SGD.
- $m = n$ is full-batch GD.

Mini-Batch SGD: Benefits

Benefit 1: Variance Reduction

- Averaging reduces variance.
- The variance of the mini-batch gradient $\tilde{g}_t$ is m times smaller than the variance of the single-sample gradient g_t.
- $E[||\tilde{g}_t - \nabla f(x_t)||^2] = \frac{1}{m} E[||g_t^1 - \nabla f(x_t)||^2]$
- This makes the gradient estimate $\tilde{g}_t$ much more accurate, leading to more stable convergence.

Benefit 2: Parallel Computation

- The m gradients in a mini-batch, $\nabla f_{i_1}(x_t), \ldots, \nabla f_{i_m}(x_t)$, are all computed at the **same point** x_t.
- They are independent computations and can be perfectly parallelized.
- This is the **primary reason** mini-batching is used in Deep Learning. GPUs (Graphical Processing Units) are massively parallel and can compute all m gradients simultaneously.

Mini-Batch SGD: Drawbacks

Drawback: Overfitting & Generalization

- It is a common belief (and often observed) that the "noise" from small batches (small m) helps SGD escape sharp, non-generalizing local minima.
- This noise acts as a regularizer, pushing the algorithm toward "flatter" minima, which are thought to generalize better.
- A **too-large mini-batch** ($m \to n$) reduces this beneficial noise, making the algorithm behave like full-batch GD, which can get stuck and overfit more easily.
- Finding the "best" m is a key part of hyperparameter tuning.

Stepsize (Learning Rate) Selection I

The Most Critical Parameter

The choice of stepsize γ_t (often called the *learning rate* in ML) is crucial.

- **Too Large:** The algorithm will be unstable, "overshoot" the minimum, and diverge. The loss will explode.
- **Too Small:** The algorithm will be stable, but convergence will be extremely slow.

Stepsize (Learning Rate) Selection II

Common Strategies

1. **Constant Stepsize:**
   - $\gamma_t = \gamma$
   - Simple, but will "bounce around" the minimum. Requires γ to be small enough.
2. **Decreasing Stepsize:**
   - $\gamma_t \to 0$, e.g., $\gamma_t \propto \frac{1}{t}$ or $\frac{1}{\sqrt{t}}$.
   - Guarantees convergence *to* the minimum, but can be slow to tune and may decay too quickly.
3. **Annealing / Schedulers (Practice):**
   - Start with a large, constant γ for fast progress in the "far-out zone".
   - Periodically "decay" (decrease) γ, e.g., divide by 10 every 20 epochs.
   - This is the most common and effective strategy in modern deep learning.