UNIVERSITY OF
COPENHAGEN

Stochastic Gradient Descent

Computational Statistics

Johan Larsson

Department of Mathematical Sciences, University of Copenhagen

March 31, 2026

Last Time

The Expectation-Maximization (EM) Algorithm

A general approach to maximum likelihood estimation with latent variables.

Fisher Information

We saw three different ways to define the Fisher information and how they relate to the EM algorithm.

Recap: The EM Algorithm

The EM algorithm iteratively optimizes

$$\theta^{(t+1)} = \arg\min_{\theta} Q(\theta \mid \theta^{(t)}),$$

with

$$Q(\theta \mid \theta^{(t)}) = \mathrm{E}_{Z|X,\theta^{(t)}} \log p(X, Z \mid \theta).$$

Convergence is **linear** near a local maximum.

Recap: Fisher Information

The Fisher information is defined as

$$i(\theta) = -\operatorname{E}\left(\frac{\partial^2}{\partial\theta^2}\log p(X \mid \theta)\right).$$

We mentioned three ways to compute the empirical observed Fisher information:

1. Differentiate the score function (defined through Q).
2. Compute complete and missing information and using the information identity.
3. Differentiate the EM mapping.

Today

Gaussian Mixture EM Example

We pick up where we left off last time, focusing on the Gaussian mixture example.

Stochastic Gradient Descent

Useful stochastic method for optimization

Can be used in a **mini-batch** version.

EM Gaussian Mixture Example

A Mixture of Two Gaussians

Goal
Fit a two-component Gaussian mixture model to data.

Model
We assume that the data $X_i \sim \mathcal{N}(\mu_k, \sigma_k^2)$ with probability π_k, $k = 1, 2$.

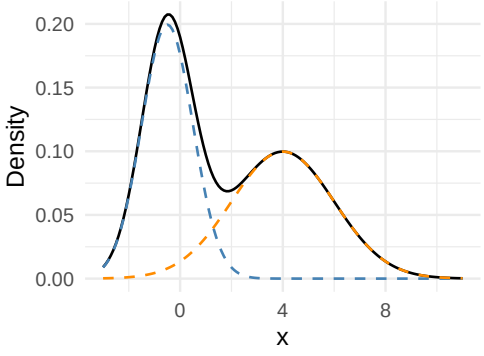


Figure 1: A mixture of two Gaussian distributions.

Finite Mixtures

In a finite mixture, the marginal density of X is

$$p(x \mid \psi) = \sum_{k=1}^{K} \pi_k f_k(x \mid \psi_k)$$

where $\pi_k \geq 0$, $\sum_k \pi_k = 1$, and f_k are densities.

Latent Variable

Assume each X_i comes from one of the K components, but we don't know which one.

We introduce a latent variable $Z_i \in \{1, \ldots, K\}$ such that

$$\Pr(Z_i = k) = \pi_k.$$

Complete-Data Model

Then the complete-data model is

$$p(x_i, z_i \mid \psi) = \pi_{z_i} f_{z_i}(x_i \mid \psi_{z_i}).$$

For Gaussian mixtures, the components are Gaussian:

$$f_k(x \mid \mu_k, \sigma_k^2) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(x-\mu_k)^2}{2\sigma_k^2}}.$$

with $\psi_k = (\mu_k, \sigma_k^2)$.

Complete-Data Log-Likelihood

Letting $\theta = (\pi_1, \ldots, \pi_K, \psi_1, \ldots, \psi_K)$, the complete-data log-likelihood is

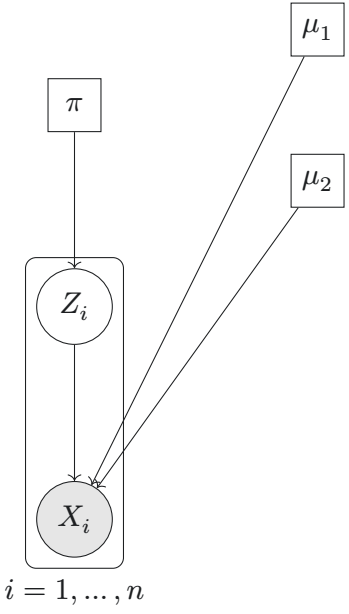
$$\begin{aligned}
\ell_c(\theta) &= \sum_{i=1}^{n} \log p(x_i, z_i \mid \theta) \\
&= \sum_{i=1}^{n} \log \left(f(x_i \mid z_i, \theta) f(z_i \mid \theta) \right) \\
&= \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbf{1}(Z_i = k) \left(\log \pi_k + \log f_k(x_i \mid \mu_k, \sigma_k^2) \right)
\end{aligned}$$

Simple Case: Fixed Component Variances, $K = 2$

Let's simplify and assume $K = 2$ and that σ_1 and σ_2 are known.

Then the model has three unknown parameters:

$$\theta = (\pi, \mu_1, \mu_2).$$



The E-Step

The complete-data log-likelihood is now

$$\ell_c(\theta) = \sum_{i=1}^{n} \mathbf{1}(Z_i = 1) \left(\log(\pi) - \frac{(x_i - \mu_1)^2}{2\sigma_1^2} \right) + \mathbf{1}(Z_i = 2) \left(\log(1-\pi) - \frac{(x_i - \mu_2)^2}{2\sigma_2^2} \right) + \text{const.}$$

Taking expectations gives

$$Q(\theta \mid \theta') = \sum_{i=1}^{n} \tau_i \left(\log(\pi) - \frac{(x_i - \mu_1)^2}{2\sigma_1^2} \right) + (1 - \tau_i) \left(\log(1-\pi) - \frac{(x_i - \mu_2)^2}{2\sigma_2^2} \right)$$

where $\tau_i = \Pr(Z_i = 1 \mid X_i = x_i, \theta')$ is the **responsibility** of component 1 for observation i.

The responsibility tells us how responsible component 1 is for generating x_i.

The Responsibilities

The conditional probability in a mixture model is generally

$$\Pr(Z = z \mid X = x) = \frac{\pi_z f_z(x \mid \psi_z)}{\sum_{k=1}^{K} \pi_k f_k(x \mid \psi_k)}.$$

For the $K = 2$ Gaussian case, this gives us

$$\tau_i = \Pr(Z_i = 1 \mid X = x_i, \theta') = \frac{\pi_1' e^{-\frac{(x_i - \mu_1')^2}{2\sigma_1^2}}}{\left(\pi_1' e^{-\frac{(x_i - \mu_1')^2}{2\sigma_1^2}} + \frac{\sigma_1}{\sigma_2}(1 - \pi_1') e^{-\frac{(x_i - \mu_2')^2}{2\sigma_2^2}} \right)}.$$

The Q-Step

The MLEs of θ given $Q(\theta \mid \theta')$ are

$$\hat{\theta} = \left(\frac{1}{n} \sum_{i=1}^{n} \tau_i, \quad \frac{1}{\sum_{i=1}^{n} \tau_i} \sum_{i=1}^{n} \tau_i x_i, \quad \frac{1}{\sum_{i=1}^{n} (1 - \tau_i)} \sum_{i=1}^{n} (1 - \tau_i) x_i \right).$$

Updating π

We update π as

$$\hat{\pi} = \frac{1}{n} \sum_{i=1}^{n} \tau_i,$$

which is the fraction of all data points "assigned" to component 1.

Updating μ_1 and μ_2

We update μ_1 as

$$\hat{\mu}_1 = \frac{\sum_{i=1}^{n} \tau_i x_i}{\sum_{i=1}^{n} \tau_i},$$

and similarly for μ_2.

This is a weighted mean, weighted by how much each component is responsible for each observation.

Example: Simulation

We start by setting the parameters.

```
sigma1 <- 1
sigma2 <- 2
mu1 <- -0.5
mu2 <- 4
p <- 0.5
```

Then we simulate data.

```
n <- 5000
z <- sample(c(TRUE, FALSE), n, replace = TRUE, prob = c(p, 1 - p))
x <- numeric(n)
n1 <- sum(z)
x[z] <- rnorm(n1, mu1, sigma1)
x[!z] <- rnorm(n - n1, mu2, sigma2)
```

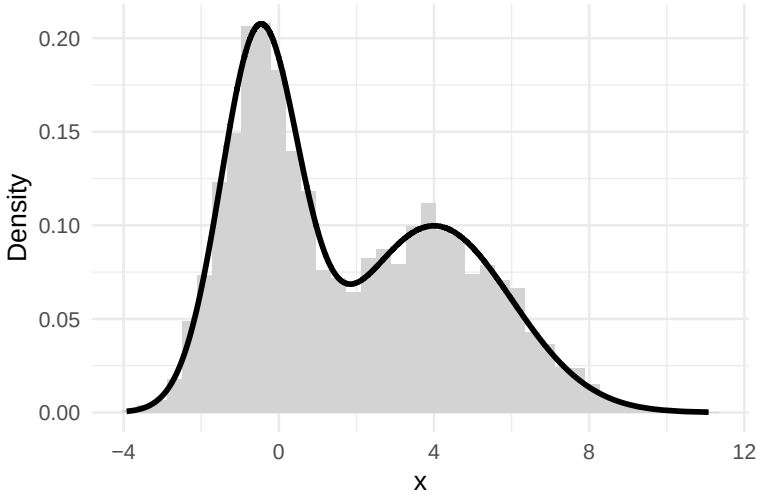


Figure 2: Histogram of simulated data from a two-component Gaussian mixture model.

```
optim_par <- optim(c(0.5, -0.5, 4), neg_loglik, x = x)
optim_par[1:2]
```

```
$par
[1]  0.4915179 -0.5368296  4.0244163

$value
[1] 6928.887
```

But the starting value matters!

```
optim(c(0.9, 3, 1), neg_loglik, x = x)[1:2]
```

```
$par
[1] 1.092782e-07 3.651325e+00 1.782837e+00

$value
[1] 8305.872
```

The EM Algorithm

```r
em <- function(par, em_step, eps = 1e-6, maxit = 20, cb = NULL) {
  for (i in seq_len(maxit)) {
    par0 <- par
    par <- em_step(par)

    if (!is.null(cb)) {
      cb() # Callback function (tracer)
    }

    if (sum((par - par0)^2) <= eps * (sum(par^2) + eps)) {
      break
    }
  }
  par
}
```

Implementation

Next, we implement the E and M steps. Implementations are not shown here, but can be found in the source code

Testing

```
em_par <- em(c(0.5, -0.5, 4), em_gauss_step)
```

```
testthat::test_that("EM and optim give similar results", {
  testthat::expect_equal(em_par, optim_par$par, tolerance = 1e-3)
})
```

```
Test passed
```

```
em(c(0.9, 3, 1), em_gauss_step) # Starting value still matters
```

```
[1] 0.2261480 5.6490188 0.6527314
```

Tracing Convergence

The `tracer` object from the **CSwR** package lets us evaluate arbitrary expressions at each iteration of the EM algorithm via `expr`.

```
library(CSwR)
em_tracer <- tracer(
  c("par0", "par", "nll"),
  Delta = 0, # No printing
  expr = quote({
    nll <- neg_loglik(par, x)
  })
)
```

```
par_hat <- em(c(0.2, 2, 2), em_gauss_step, cb = em_tracer$tracer)

em_trace <- summary(em_tracer)
```

CSwR contains an `autoplot` S3 method for `trace` objects.

```
autoplot(
  em_trace,
  y = nll - min(nll)
)
```

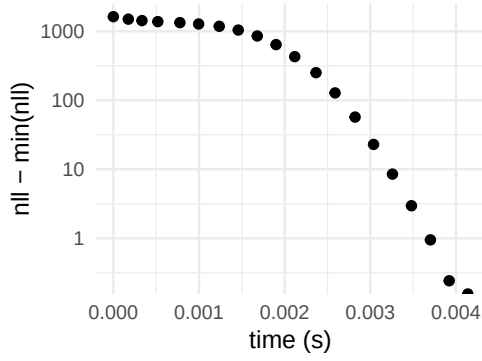


Figure 3: Convergence of the EM algorithm for Gaussian mixture model.

Figure 4: Iterations of the EM algorithm

Figure 5: Iterations of the EM algorithm

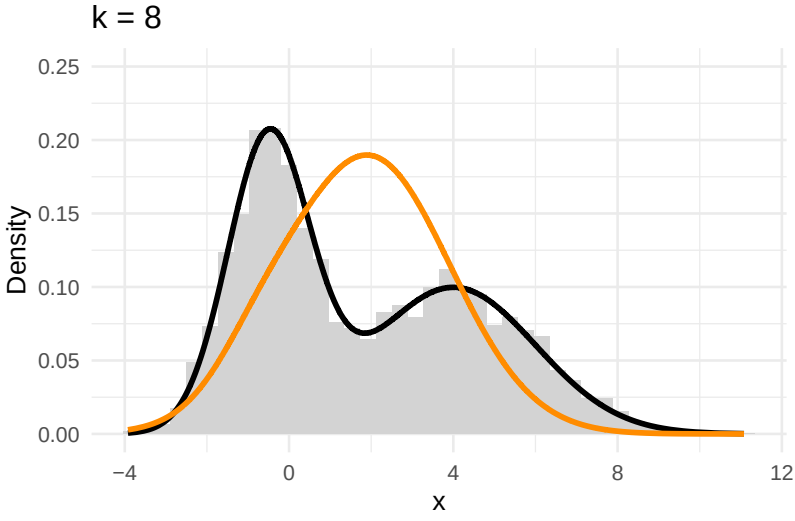


Figure 6: Iterations of the EM algorithm

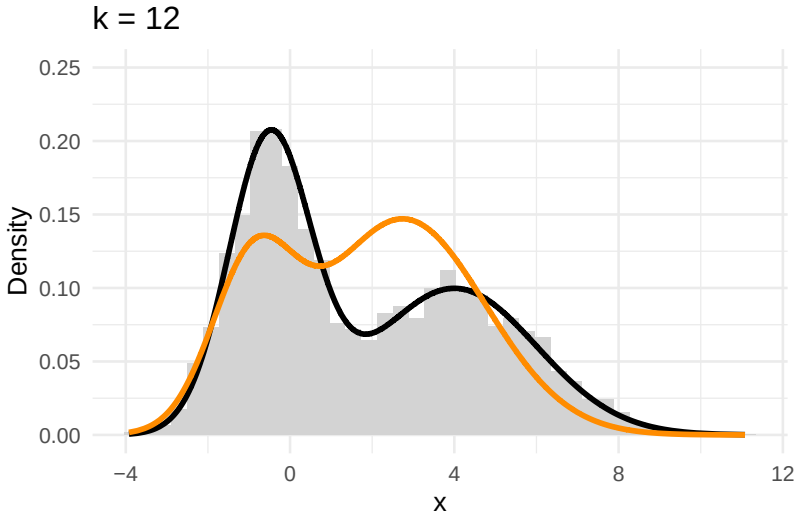


Figure 7: Iterations of the EM algorithm

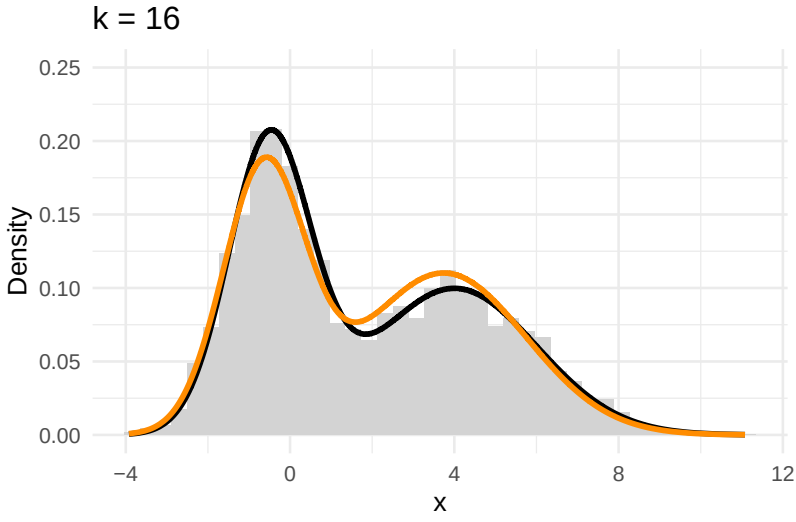


Figure 8: Iterations of the EM algorithm

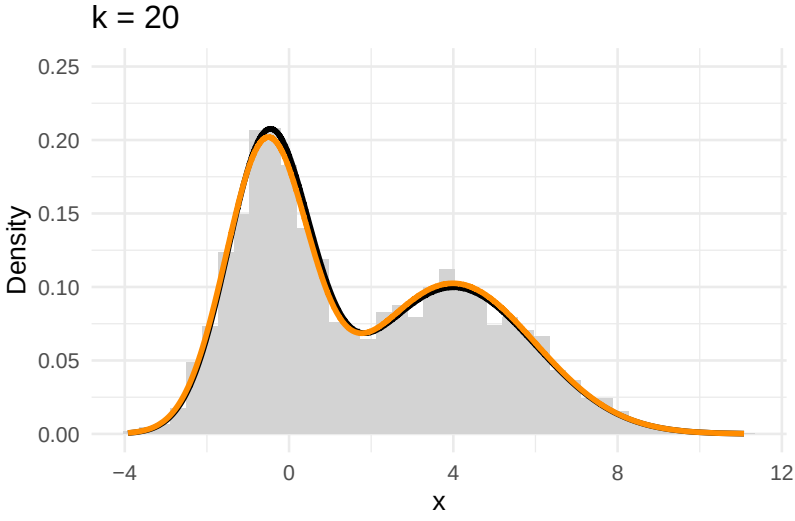


Figure 9: Iterations of the EM algorithm

Benchmarking

You might be inclined to wonder which method is faster: EM or `optim()`?

```
gauss_bm <- bench::mark(
  em = em(c(0.5, -0.5, 4), em_gauss_step),
  optim = optim(c(0.5, -0.5, 4), neg_loglik, x = x),
  check = FALSE
)
```

```
plot(gauss_bm)
```

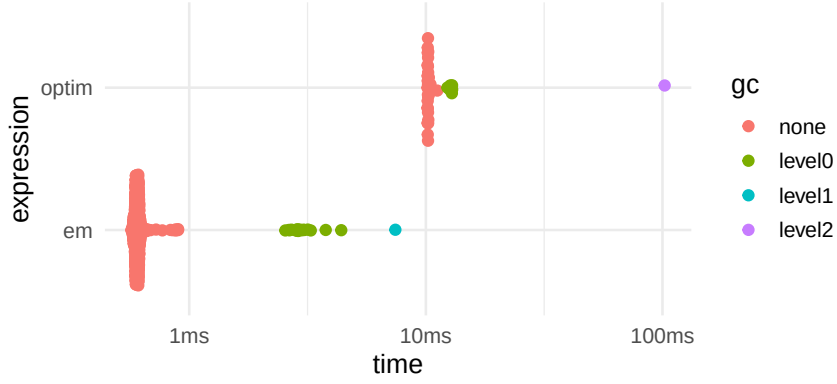


Figure 10: Benchmarking results for EM vs. optim() for Gaussian mixture model.

But this comparison is flawed: **why?**

Label Switching

If we set $\sigma_1 = \sigma_2$, then the likelihood is invariant under switching the labels of the components, i.e., (p, μ_1, μ_2) and $(1-p, \mu_2, \mu_1)$ give the same likelihood.

So the EM algorithm may converge to either of the two symmetric modes.

```
em(c(0.2, 3, -0.5), em_gauss_step_alt)
```

```
[1]  0.1931793  4.0261575 -0.5220438
```

```
em(c(0.8, -3, -0.5), em_gauss_step_alt)
```

```
[1]  0.8068534 -0.5219443  4.0265133
```

In this example, we generated data with $p = 0.8$ and $\sigma_1 = \sigma_2$.

Stochastic Gradient Descent

Minimizing Sums

Many of the functions we are trying to minimize are of the form

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

Gradient Descent
Since $\nabla\left(\sum_{i=1}^{n} f_i(x)\right) = \sum_{i=1}^{n} \nabla f_i(x)$, GD steps are

$$x_{k+1} = x_k - t\sum_{i=1}^{n} \nabla f_i(x_k).$$

Stochastic Gradient Descent
SGD instead takes steps

$$x_{k+1} = x_k - t\nabla f_i(x_k)$$

where i is an index in $\{1, \ldots, n\}$.

Example: Ordinary Least Squares

The loss function is

$$f(\beta) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - x_i^\top \beta)^2.$$

so

$$f_i(\beta) = \frac{1}{2} (y_i - x_i^\top \beta)^2.$$

And the gradient is

$$\nabla f_i(\beta) = -x_i (y_i - x_i^\top \beta).$$

Basic Implementation of SGD

```
sgd_basic <- function(
  par,
  grad,
  n, # number of data points
  t, # step size
  sampler = sample, # function that samples indices
  maxit = 100,
  ...
) {
  for (k in seq_len(maxit)) {
    for (i in seq_len(n)) {
      ind <- sampler(n)
      par <- par - t * grad(par, ind, ...)
    }
  }
  par
}
```

Unbiasedness

If i is drawn uniformly at random from $\{1, \dots, n\}$ then

$$\mathrm{E}(\nabla f_i(x_k)) = \nabla f(x_k).$$

So SGD gradients are **unbiased** estimates of the full gradient.

Iteration Cost
The main benefit of this is that the cost of each iteration is much lower: $O(p)$ vs. $O(np)$ (for full gradient descent with p variables).

Epoch
When discussing SGD, we often refer to an **epoch** as n iterations of SGD: a full pass over the data.

Example: Logistic Regression

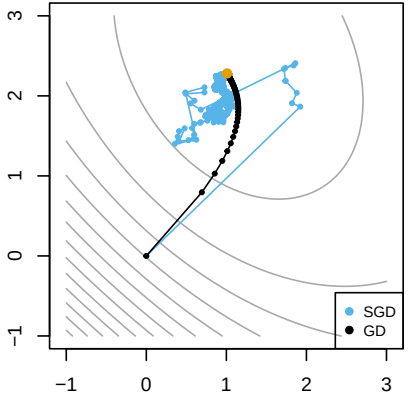
With $y_i \in \{0, 1\}$, the logistic regression loss is

$$f_i(\beta) = -y_i x_i^\top \beta + \log\left(1 + e^{x_i^\top \beta}\right).$$

and

$$\nabla f_i(\beta) = x_i \left(\frac{e^{x_i^\top \beta}}{1 + e^{x_i^\top \beta}} - y_i \right).$$

SGD typically converges quickly at the start but slows down as it approaches the minimum.



Step Size (Learning Rate)

For gradient descent, we could use a fixed step size (learning rate), and guarantee convergence.

But this is **not** the case for SGD.

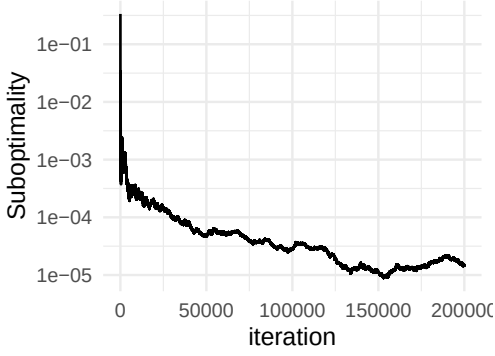


Figure 11: Convergence of SGD for logistic regression with a fixed step size.

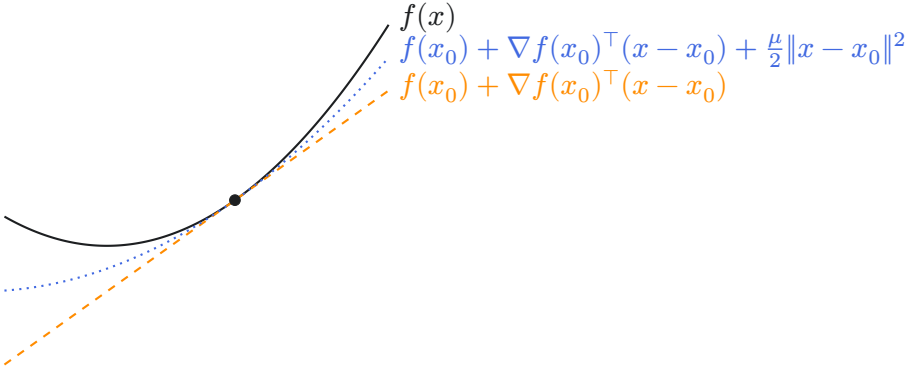
Can we find conditions on the step sizes t_k that ensure convergence of SGD? **Yes!**

But first, we need to introduce the concept of **strong convexity**.

Strong Convexity

A differentiable function $f : \mathbb{R}^p \to \mathbb{R}$ is **strongly convex** if

$$f(x) \geq f(x_0) + \nabla f(x_0)^\top (x - x_0) + \frac{\mu}{2}\|x - x_0\|_2^2, \quad \forall x, x_0 \in \mathbb{R}^p.$$



Example: Strong Convexity

The function

$$f(x) = x^2$$

is strongly convex with $\mu = 2$ since

$$\begin{aligned} f(x) - f(x_0) - \nabla f(x_0)^\top (x - x_0) &= x^2 - x_0^2 - 2x_0(x - x_0) \\ &= (x - x_0)^2 \\ &\geq \frac{\mu}{2}(x - x_0)^2. \end{aligned}$$

The inequality holds for $\mu \leq 2$, and in particular for $\mu = 2$.

Relationship to Lipschitz Gradient

If f is strongly convex with parameter μ and has Lipschitz gradient with constant L, then

$$\mu I \preceq \nabla^2 f(x) \preceq LI.$$

- Strong convexity provides a **lower** bound on the curvature of f
- Lipschitz gradient provides an **upper** bound on the curvature of f

Example

The function $f(x) = x^2$ is strongly convex with $\mu = 2$ and has Lipschitz gradient with $L = 2$, since $\nabla^2 f(x) = 2$.

This is *identical* to the strong convexity parameter.

Example of Strong Convexity–Lipschitz Gap

Let's take the function

$$f(x) = x^2 + \sin(x).$$

This is twice differentiable with

$$\nabla^2 f(x) = 2 - \sin(x).$$

And since

$$1 \leq 2 - \sin(x) \leq 3,$$

f is strongly convex with $\mu = 1$ and has Lipschitz gradient with $L = 3$.

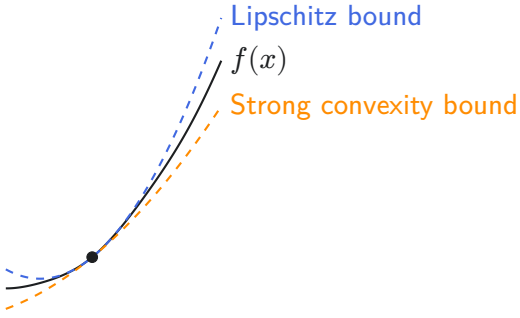


Figure 12: The function $f(x) = x^2 + \sin(x)$ is strongly convex with $\mu = 1$ and has Lipschitz gradient with $L = 3$.

The **condition number** of a strongly convex function with Lipschitz gradient is

$$\kappa = \frac{L}{\mu} \geq 1.$$

The closer κ is to 1, the better conditioned the problem is.

Example

For $f(x) = x^2$, we have $\kappa = 2/2 = 1$, so the problem is perfectly conditioned.

We reach the minimum in one step with gradient descent and step size $t = 1/L = 1/2$.

The Robbins–Monro Convergence Theorem

Suppose f is strongly convex and we have unbiased stochastic estimates of the gradients, g_k, with $\mathrm{E}(g_k \mid x_k) = \nabla f(x_k)$ and

$$\mathrm{E}\left(\|g_k\|_2^2 \mid x_k\right) \leq A + B\|x_k\|_2^2$$

for some constants A and B.

If x^* is the global minimizer of f then x_n converges almost surely towards x^* if

$$\sum_{k=1}^{\infty} t_k^2 < \infty \quad \text{and} \quad \sum_{k=1}^{\infty} t_k = \infty.$$

The theorem applies to learning rates satisfying

$$t_k \propto \frac{1}{k}.$$

Diminishing Step Sizes

Assume cyclic rule and set $t_k = t$ for n updates. We get

$$x_{k+n} = x_k - t \sum_{i=1}^{n} \nabla f_i(x_{k+i-1}).$$

Meanwhile, **full** gradient descent with step size nt would give

$$x_{k+1} = x_k - t \sum_{i=1}^{n} \nabla f_i(x_k).$$

Difference between the two is

$$t \sum_{i=1}^{n} \Big(\nabla f_i(x_{k+i-1}) - \nabla f_i(x_k) \Big)$$

which does not tend to zero for constant t. So the noise in the gradient estimates does not vanish as we approach the minimum.

Convergence Rates

Gradient Descent

For convex f, GD with diminishing step sizes converges at rate

$$f(x_k) - f^* = O\left(\frac{1}{\sqrt{k}}\right).$$

When f is differentiable with Lipschitz gradient, we get

$$f(x_k) - f^* = O\left(\frac{1}{k}\right).$$

Stochastic Gradient Descent

For convex f with diminishing step sizes, SGD converges at rate

$$\mathrm{E}\left(f(x_k) - f^*\right) = O\left(\frac{1}{\sqrt{k}}\right).$$

But this **does not improve** when f is differentiable with Lipschitz gradient.

Rates for Strongly Convex Functions

When f is strongly convex and has Lipschitz gradient, GD satisfies

$$f(x_k) - f^* = O(\delta^k), \qquad \delta \in (0, 1)$$

and $\delta = 1 - \mu/L$
Under same conditions, SGD gives us

$$\mathrm{E}(f(x_n) - f^*) = O\left(\frac{1}{k}\right).$$

So SGD does **not** enjoy linear convergence rates under strong convexity.

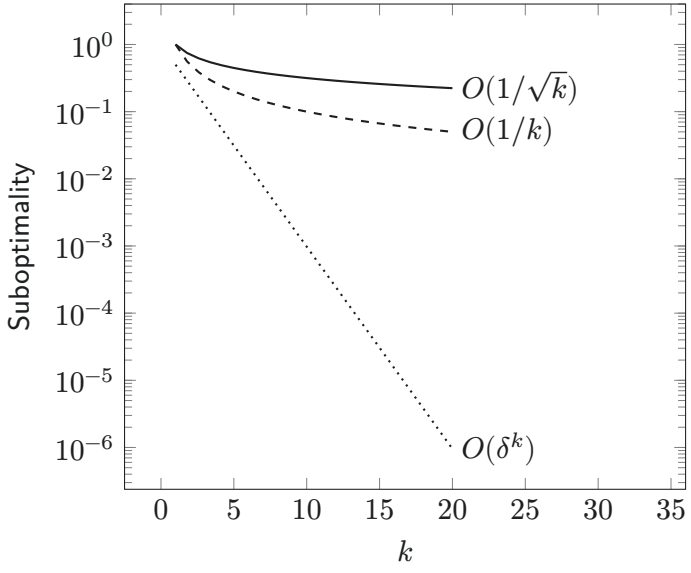


Figure 13: Illustration of convergence rates. Here, $\delta = 0.5$.

Mini-Batch Gradient Descent

Take mini-batch A_k of size b of the data and iterate through

$$x_{k+1} = x_k - \frac{t}{b} \sum_{i \in A_k} \nabla f_i(x_k).$$

Estimates are still unbiased,

$$\mathrm{E}\, \frac{1}{b} \sum_{i \in A_k} \nabla f_i(x) = \nabla f(x),$$

but variance is reduced by a factor of $1/b$.

Under Lipschitz gradient, rate goes from $O(1/\sqrt{k})$ to $O(1/\sqrt{bk} + 1/k)$.

Typically more efficient than standard SGD for various computational reasons, but somewhat **less** robust to local minima.

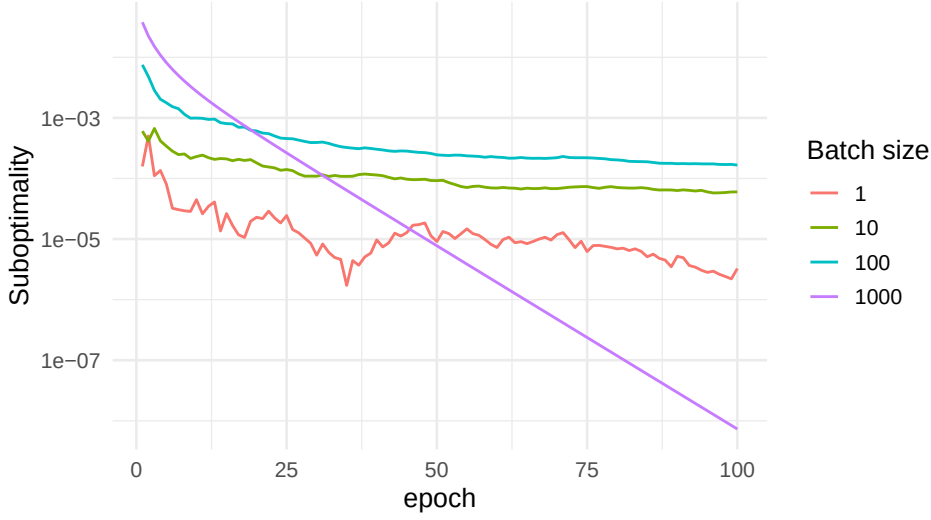


Figure 14: Convergence of SGD on the logistic regression problem, with different batch sizes. Note that batch size 1000 is standard gradient descent.

So, Why Use SGD?

We have seen that convergence rates of SGD are worse than those of GD.

So why even care about SGD?

Reasons to Use SGD

- If $n \gg p$, then we can pick a batch size that is reasonably accurate but still much smaller than n.
- In many applications (e.g. deep learning), we don't *care* about optimizing to high accuracy.
- Sometimes, the stochasticity in SGD helps it escape local minima.

Convergence in Epochs and Iterations

In terms of iterations, gradient descent converges much more quickly than SGD. But in terms of epochs, SGD can be faster.

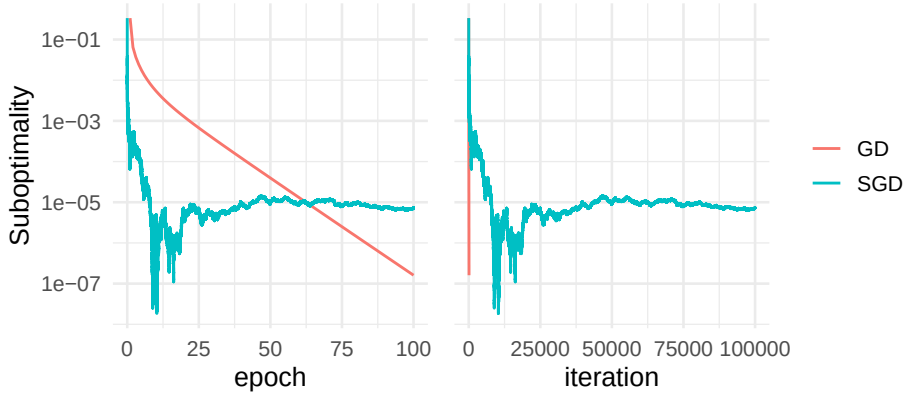


Figure 15: Convergence of gradient descent (GD) and stochastic gradient descent (SGD) for logistic regression in terms of iterations and epochs.

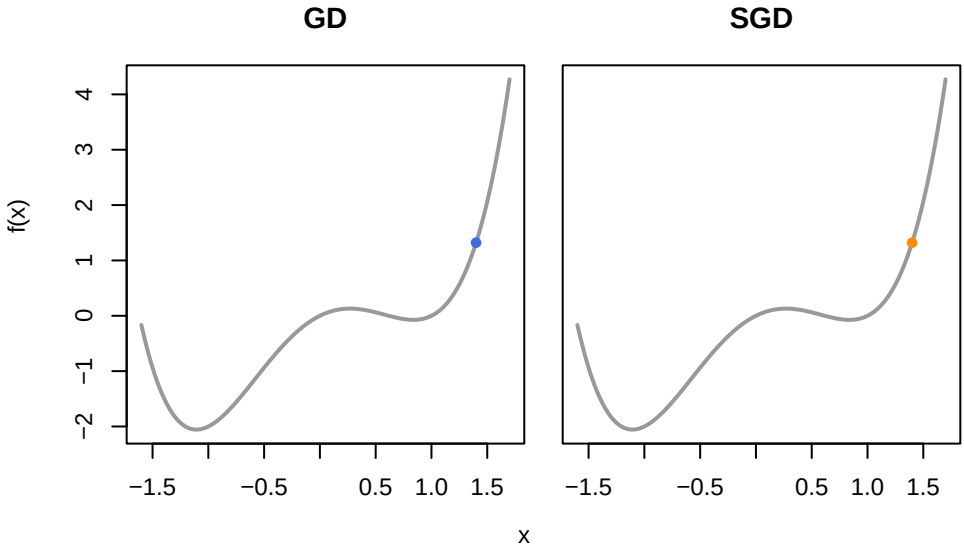


Figure 16: The stochasticity in SGD sometimes helps it escape local minima.

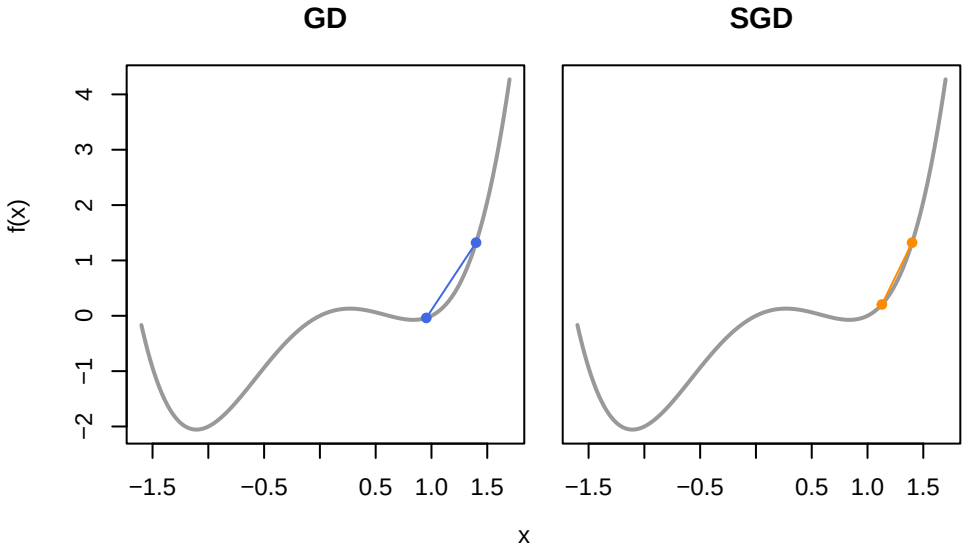


Figure 17: The stochasticity in SGD sometimes helps it escape local minima.

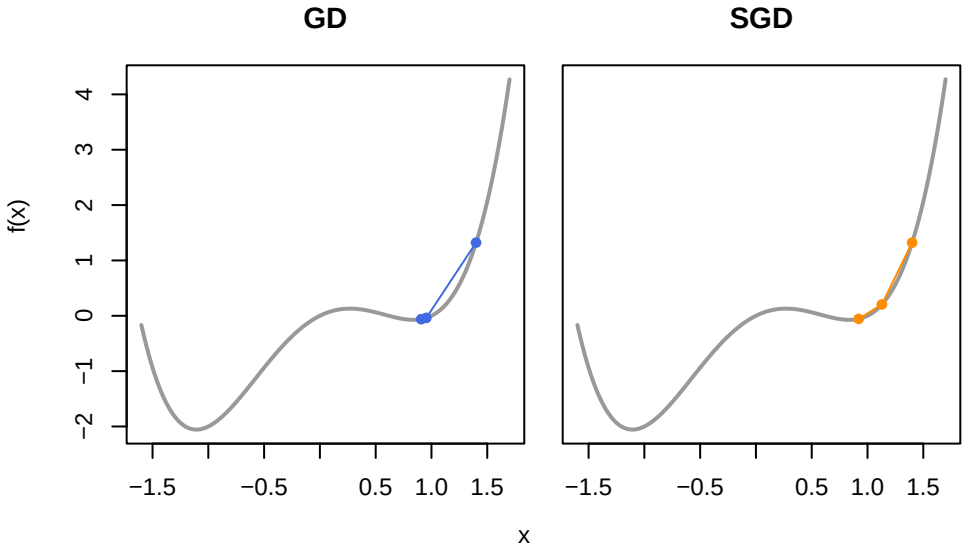


Figure 18: The stochasticity in SGD sometimes helps it escape local minima.

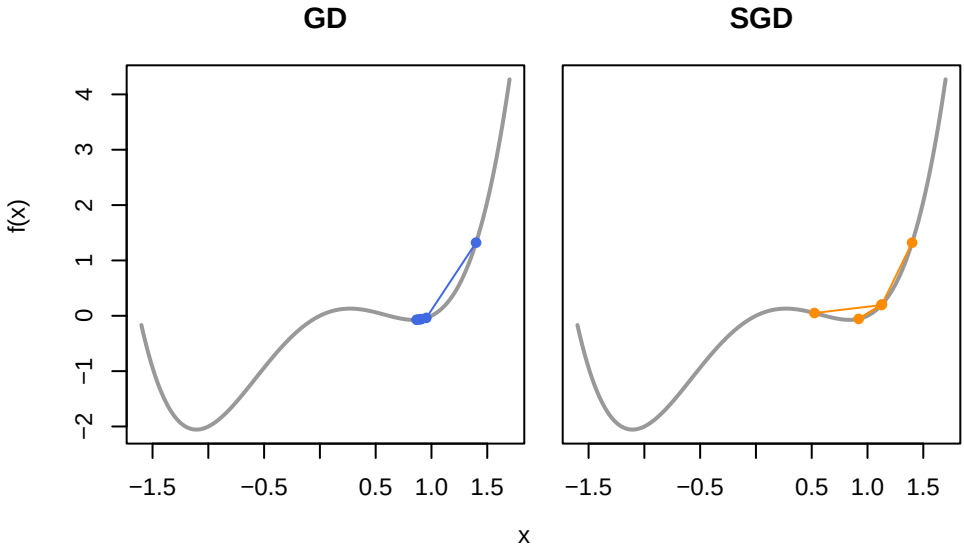


Figure 19: The stochasticity in SGD sometimes helps it escape local minima.

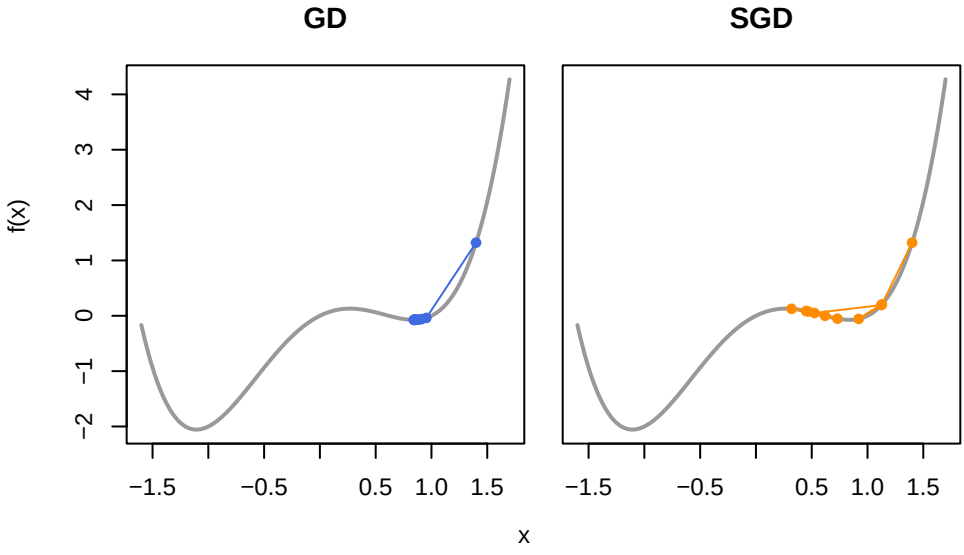


Figure 20: The stochasticity in SGD sometimes helps it escape local minima.

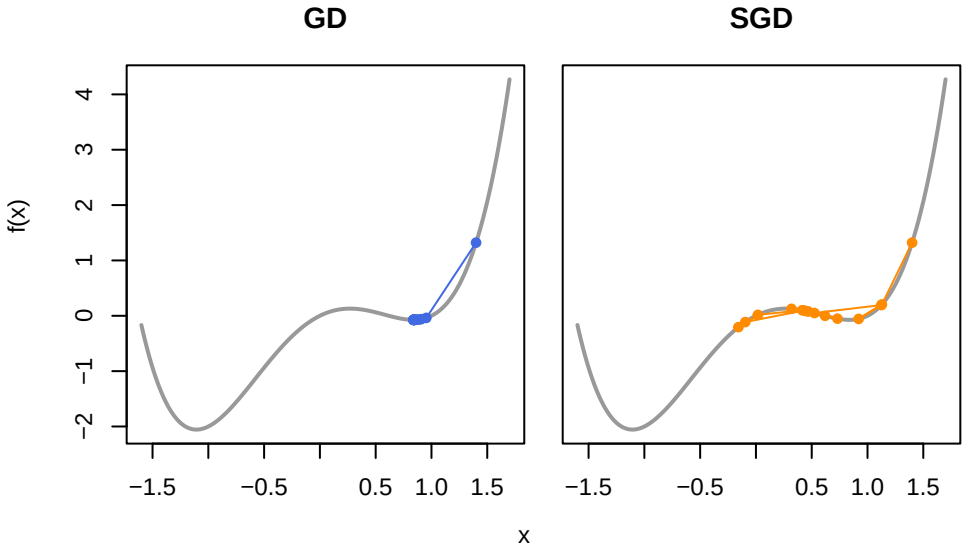


Figure 21: The stochasticity in SGD sometimes helps it escape local minima.

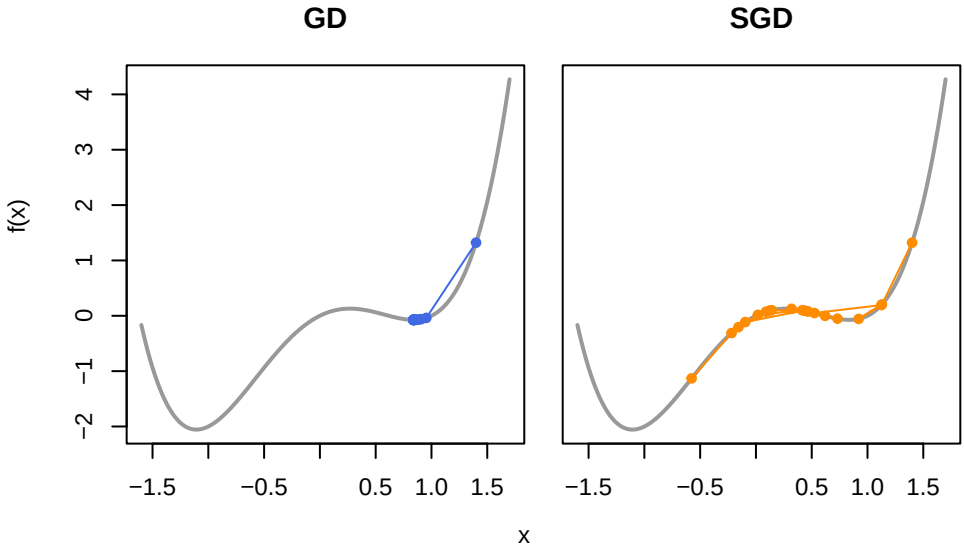


Figure 22: The stochasticity in SGD sometimes helps it escape local minima.

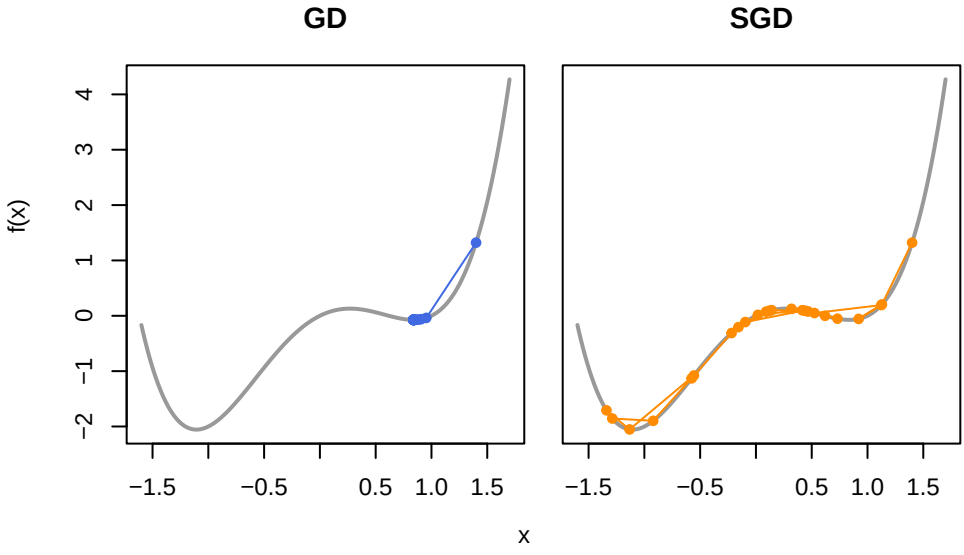


Figure 23: The stochasticity in SGD sometimes helps it escape local minima.

Learning Rate (Step Size)

The learning rate (step size) t is crucial for SGD, but hard to set.

Convergence theorem only gives a few hints.

A Class of Decay Schedules

$$t_k = \frac{t_0 K}{K + k^a} = \frac{t_0}{1 + K^{-1} k^a}$$

with initial learning rate $t_0 > 0$ and constants $K, a > 0$.

Convergence is ensured by Robbins–Monro if $a \in (0.5, 1]$.

Fixing the exponent a and picking a target rate, t_1, to be reached after k_1 steps, we can solve for K and get

$$K = \frac{k_1^a t_1}{t_0 - t_1}.$$

```r
decay_scheduler <- function(
  t0 = 1,
  a = 1,
  K = 1,
  t = NULL,
  k1 = NULL
) {
  force(a)

  if (!is.null(t) && !is.null(k1)) {
    K <- k1^a * t / (t0 - t)
  }

  b <- t0 * K

  function(k) b / (K + k^a)
}
```

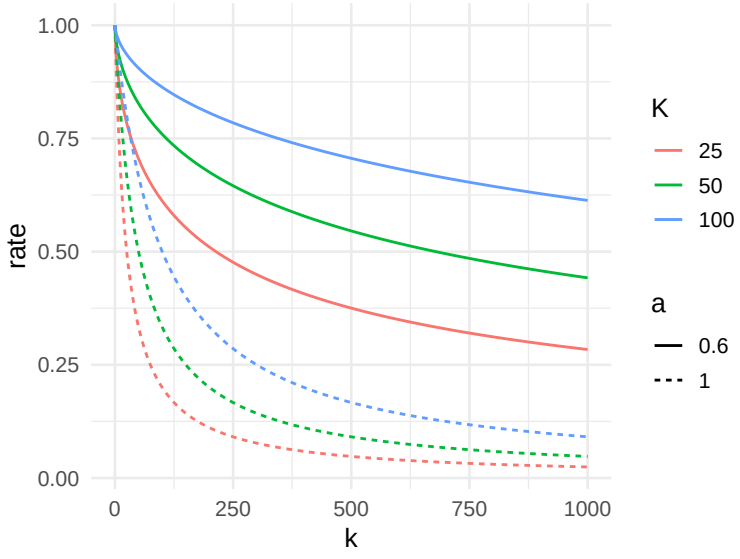


Figure 24: Examples of learning rate schedules, using $t_k = \frac{t_0}{1+K^{-1}k^a}$.

So, Is The Problem of Setting Learning Rate Solved?

No, not really, because it's still hard to pick t_0, K, and a.

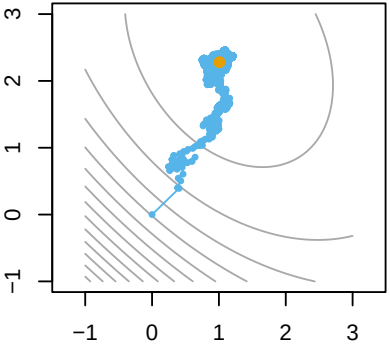


Figure 25: SGD on the logistic regression

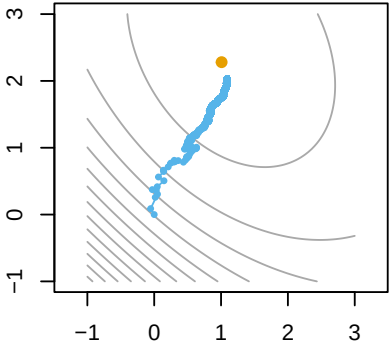


Figure 26: SGD on the logistic regression

Algorithm 1: Stochastic gradient descent (with mini-batches)

Data: Initial step size $t_0 > 0$

for $k = 1, 2, \ldots$ **do**

  $A_k \leftarrow$ batch of data;

  $t_k \leftarrow$ new, smaller, step size;

  $x \leftarrow x - t_k \sum_{i \in A_k} \nabla f(x_i)$;

  if *converged* **then**

    return x

Nested Loops

Implementing an SGD algorithm typically means implementing a nested loop, and we know this isn't R's strongest suite.

Next lecture, when we turn to Rcpp, you will have the tools to solve this problem.

Data Storage

SGD algorithms typically subset rows, or blocks of rows.

In a column-major order matrix, this is not optimal. Transposing the matrix first can help.

Exercise: Single-Variable Ridge Regression

Minimize

$$\frac{1}{n}\sum_{i=1}^{n} f_i(\beta), \quad \text{with} \quad f_i(\beta) = \frac{1}{2}(y_i - x_i\beta)^2 + \frac{\lambda}{2}\beta^2.$$

Step 1

Generate some data, for instance x from `rnorm` and `y <- x * beta + rnorm(n)`. You can pick β and n however you want.

Step 2

Solve this problem using stochastic gradient descent. Plot loss in terms of epochs and time. Test your method against `optimize()` or an analytical solution.

Step 3

Profile your algorithm.

Summary

Stochastic gradient descent is a popular method for large-scale optimization.

Convergence results are relatively weak, but for many applications this does not matter.

In practice, most implementations use mini-batches.

Next Two Lectures

Rcpp

We will look at how to use Rcpp to speed up R code.

Because SGD typically involves nested loops, these algorithms benefit greatly from relying on C++.

Variations on SGD

We will look at some variations on SGD that improve convergence in practice.

These include momentum and Nesterov acceleration.