CS-GY 6923: Lecture 6
Gradient Descent + Stochastic Gradient Descent

NYU Tandon School of Engineering, Akbar Rafiey

Logistic regression

Goal: Minimize the logistic loss:

$$L(\boldsymbol{\beta}) = -\sum_{i=1}^{n} y_i \log(h(\boldsymbol{\beta}^T \mathbf{x}_i)) + (1 - y_i) \log(1 - h(\boldsymbol{\beta}^T \mathbf{x}_i))$$

I.e. find $\boldsymbol{\beta}^* = \arg\min L(\boldsymbol{\beta})$. How should we do this?

Logistic regression

Set all partial derivatives to 0! Recall that $\nabla L(\boldsymbol{\beta})$ is the length d vector containing all partial derivatives evaluated at $\boldsymbol{\beta}$:

$$\nabla L(\boldsymbol{\beta}) = \begin{bmatrix} \frac{\partial L}{\partial \beta_1} \\ \frac{\partial L}{\partial \beta_2} \\ \vdots \\ \frac{\partial L}{\partial \beta_d} \end{bmatrix}$$

Logistic regression gradient

$$L(\beta) = -\sum_{i=1}^{n} y_i \log(h(\beta^T \mathbf{x}_i)) + (1 - y_i) \log(1 - h(\beta^T \mathbf{x}_i))$$

Let $\mathbf{X} \in \mathbb{R}^{d \times n}$ be our data matrix with $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ as rows. Let $\mathbf{y} = [y_1, \ldots, y_n]$. A calculation gives:

$$\nabla L(\beta) = \mathbf{X}^T \left(h(\mathbf{X}\beta) - \mathbf{y} \right)$$

where $h(\mathbf{X}\beta) = \frac{1}{1+e^{-\mathbf{X}\beta}}$. Here all operations are entrywise. I.e in Python you would compute:

```
h = 1/(1 + np.exp(-X@beta))
grad = np.transpose(X)@(h - y)
```

Logistic regression gradient

To find β minimizing $L(\beta)$ we typically start by finding a β where:

$$\nabla L(\beta) = \mathbf{X}^T \left(h(\mathbf{X}\beta) - \mathbf{y}\right) = \mathbf{0}$$

- In contrast to what we saw when minimizing the squared loss for linear regression, there's no simple closed form expression for such a β!
- This is the typical situation when minimizing loss in machine learning. (linear regression was a lucky exception.)
- **Main question:** How do we minimize a loss function $L(\beta)$ when we can't explicitly compute where it's gradient is $\mathbf{0}$?

Minimizing loss functions

Always an option: Brute-force search. Test our many possible values for $\boldsymbol{\beta}$ and just see which gives the smallest value of $L(\boldsymbol{\beta})$.

- As we saw on Lab 1, this actually works okay for low-dimensional problems (e.g. when $\boldsymbol{\beta}$ has 1 or 2 entries).
- **Problem:** Super computationally expensive in high-dimension. For $\boldsymbol{\beta} \in \mathbb{R}^d$, run time grows as:

Minimizing loss functions

Much Better idea. Some sort of guided search for a good of β.

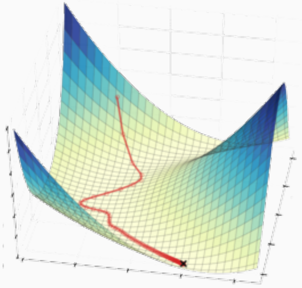
- Start with some $\boldsymbol{\beta}^{(0)}$, and at each step try to change $\boldsymbol{\beta}$ slightly to reduce $L(\boldsymbol{\beta})$.
- Hopefully find an approximate minimizer for $L(\boldsymbol{\beta})$ much more quickly than brute-force search.
- **Concrete goal:** Find $\boldsymbol{\beta}$ with

$$L(\boldsymbol{\beta}) < \min_{\boldsymbol{\beta}} L(\boldsymbol{\beta}) + \epsilon$$

for some small error term ϵ.

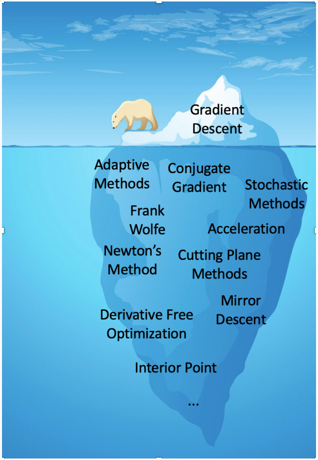
Gradient descent

Gradient descent: A greedy search algorithm for minimizing functions of multiple variables (including loss functions) that often works amazingly well.



The single most important computational tool in machine learning.
And it's remarkable simple + easy to implement.

Optimization algorithms



Just one method in a huge class of algorithms for numerical optimization. All of these methods are important in ML.

First order optimization

First order oracle model: Given a function L to minimize, assume we can:

- **Function oracle**: Evaluate $L(\boldsymbol{\beta})$ for any $\boldsymbol{\beta}$.
- **Gradient oracle**: Evaluate $\nabla L(\boldsymbol{\beta})$ for any $\boldsymbol{\beta}$.

These are very general assumptions. Gradient descent will not use <u>any other information</u> about the loss function L when trying to find a $\boldsymbol{\beta}$ which minimizes L.

Basic Gradient descent algorithm:

- Choose starting point $\beta^{(0)}$.
- For $i = 0, \ldots, T - 1$:
  - $\beta^{(i+1)} = \beta^{(i)} - \eta \nabla L(\beta^{(i)})$
- Return $\beta^{(T)}$.

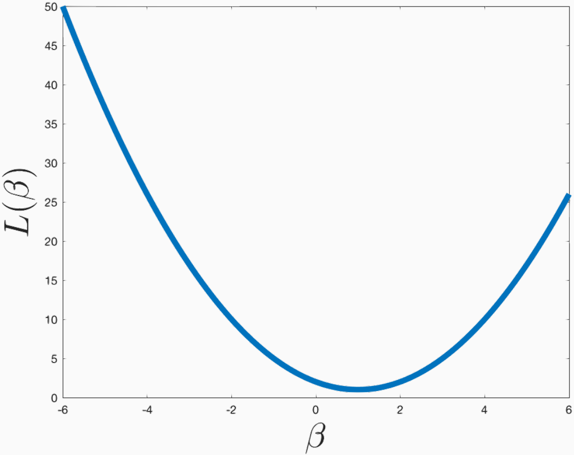
$\eta > 0$ is a step-size parameter. Also called the learning rate.

Why does this method work?

First observation: if we actually reach the minimizer β^* then we stop.

Intuition

Consider a 1-dimensional loss function. I.e. where β is just a single value. Our update step is $\beta^{(i+1)} = \beta^{(i)} - \eta L'(\beta^{(i)})$



Gradient descent in 1D

Mathematical way of thinking about it:

By definition, $L'(\beta) = \lim_{t \to 0} \frac{L(\beta+t)-L(\beta)}{t}$. So for small values of t, we expect that:

$$L(\beta + t) - L(\beta) \approx t \cdot L'(\beta).$$

We want $L(\beta + t)$ to be smaller than $L(\beta)$, so we want $t \cdot L'(\beta)$ to be negative.

This can be achieved by choosing $t = -\eta \cdot L'(\beta)$.

$$\beta^{(i+1)} = \beta^{(i)} - \eta L'(\beta^{(i)})$$

Directional derivatives

For high dimensional functions ($\boldsymbol{\beta} \in \mathbb{R}^d$), our update involves a vector $\mathbf{v} \in \mathbb{R}^d$. At each step:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \mathbf{v}.$$

Question: When $\mathbf{v}$ is small, what's an approximation for $L(\boldsymbol{\beta} + \mathbf{v}) - L(\boldsymbol{\beta})$?

$$L(\boldsymbol{\beta} + \mathbf{v}) - L(\boldsymbol{\beta}) \approx$$

Directional derivatives

We have

$$L(\boldsymbol{\beta} + \mathbf{v}) - L(\boldsymbol{\beta}) \approx \frac{\partial L}{\partial \beta_1} v_1 + \frac{\partial L}{\partial \beta_2} v_2 + \ldots + \frac{\partial L}{\partial \beta_d} v_d$$
$$= \langle \nabla L(\boldsymbol{\beta}), \mathbf{v} \rangle.$$

How should we choose v so that $L(\boldsymbol{\beta} + \mathbf{v}) < L(\boldsymbol{\beta})$?

---

[0]Formally, you might remember that we can define the **directional derivative** of a multivariate function: $D_{\mathbf{v}} L(\boldsymbol{\beta}) = \lim_{t \to 0} \frac{L(\boldsymbol{\beta} + t\mathbf{v}) - L(\boldsymbol{\beta})}{t}$.

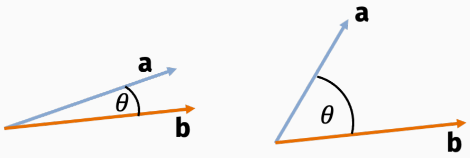
Steepest descent

Claim (Gradient descent = Steepest descent[1])

$$\frac{-\nabla L(\beta)}{\|\nabla L(\beta)\|_2} = \arg\min_{\mathbf{v}, \|\mathbf{v}\|_2 = 1} \langle \nabla L(\beta), \mathbf{v} \rangle$$

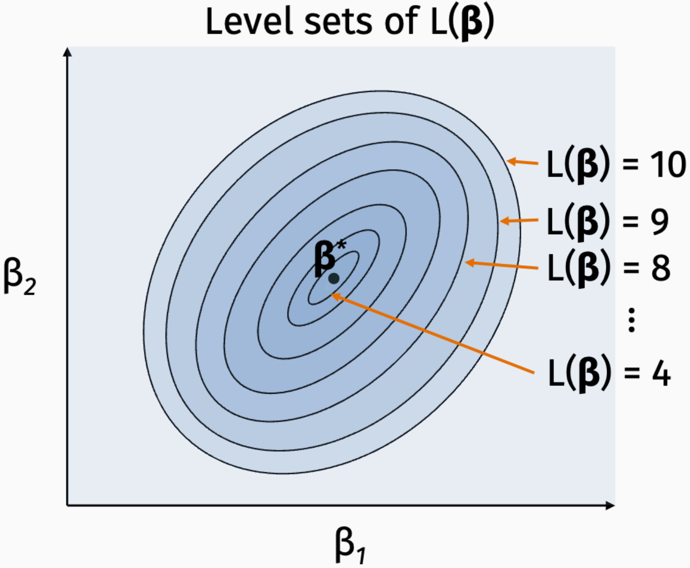
Recall: For two vectors $\mathbf{a}, \mathbf{b}$,

$$\langle \mathbf{a}, \mathbf{b} \rangle = \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 \cdot \cos(\theta)$$



[1] We could have restricted $\mathbf{v}$ using a different norm. E.g. $\|\mathbf{v}\|_1 \leq 1$ or $\|\mathbf{v}\|_\infty = 1$. These choices lead to variants of generalized steepest descent..

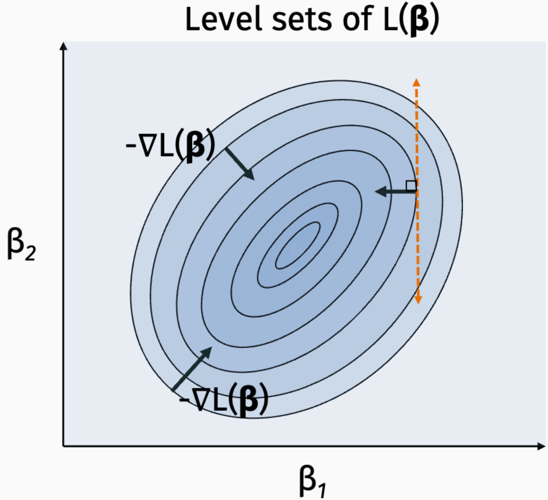
Visualizing in 2D



Steepest descent

Claim (Gradient descent = Steepest descent)

$$\frac{-\nabla L(\beta)}{\|\nabla L(\beta)\|_2} = \arg\min_{\mathbf{v}, \|\mathbf{v}\|_2 = 1} \langle \nabla L(\beta), \mathbf{v} \rangle$$



Gradient descent

Basic Gradient descent (GD) algorithm:

- Choose starting point $\boldsymbol{\beta}^{(0)}$.
- For $i = 0, \ldots, T-1$:
  - $\boldsymbol{\beta}^{(i+1)} = \boldsymbol{\beta}^{(i)} - \eta \nabla L(\boldsymbol{\beta}^{(i)})$
- Return $\boldsymbol{\beta}^{(T)}$.

- **Theoretical questions:** Does gradient descent always converge to the minimum of the loss function L? Can you prove how quickly?
- **Practical questions:** How to choose η? Any other modifications needed for good practical performance?

Basic claim

- For sufficiently small η, every step of GD either
  1. Decreases the function value.
  2. Gets stuck because the gradient term equals 0

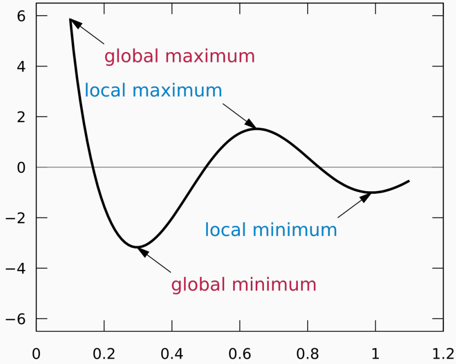
Claim

For sufficiently small η and a sufficiently large number of iterations T, gradient descent will converge to a **local minimum** *or* **stationary point** *of the loss function $\tilde{\boldsymbol{\beta}}^*$. I.e. with*

$$\nabla L(\tilde{\boldsymbol{\beta}}^*) = \mathbf{0}.$$

Basic claim

You can have stationary points that are not minima (local maxima, saddle points). In practice, always converge to local minimum.



Very unlikely to land precisely on another stationary point and get stuck. Non-minimal stationary points are "unstable".

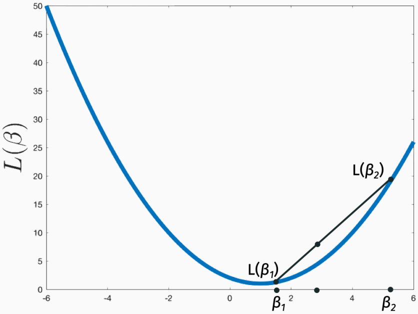
Convex function

For a broad class of functions, GD converges to global minima.

Definition (Convex)

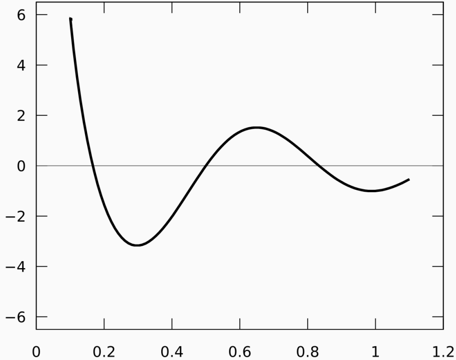
A function L is convex iff for any $\boldsymbol{\beta_1}, \boldsymbol{\beta_2}, \lambda \in [0,1]$:

$$(1-\lambda)\cdot L(\boldsymbol{\beta_1}) + \lambda \cdot L(\boldsymbol{\beta_2}) \geq L((1-\lambda)\cdot \boldsymbol{\beta_1} + \lambda \cdot \boldsymbol{\beta_2})$$



Convex function

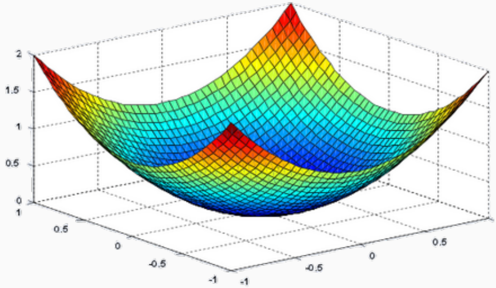
In words: A function is convex if a line between any two points on the function lies above the function. Captures the notion that a function looks like a bowl.



This function **is not** convex.

Convex function

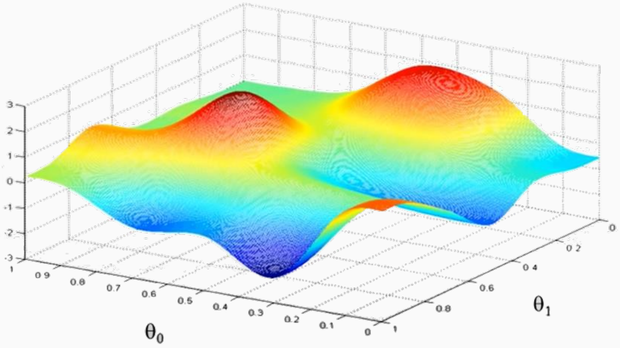
In words: A function is convex if a line between any two points on the function lies above the function. Captures the notion that a function looks like a bowl.



This function **is** convex.

Convex function

In words: A function is convex if a line between any two points on the function lies above the function. Captures the notion that a function looks like a bowl.



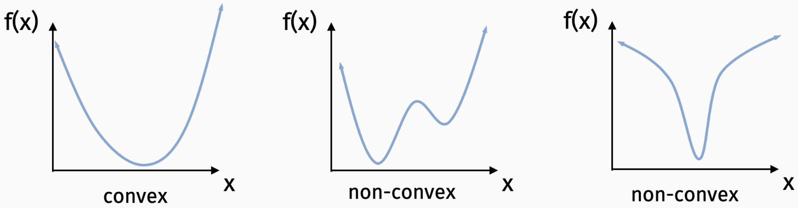
This function **is not** convex.

Convergence of gradient descent

What functions are convex?

- Least squares loss for linear regression.
- ℓ_1 loss for linear regression.
- Either of these with and ℓ_1 or ℓ_2 regularization penalty.
- Logistic regression! Logistic regression with regularization.
- Many other models in machine leaning.

Non-convex



What functions in machine learning are not convex? Loss functions involving neural networks, matrix completion problems, mixture models, many more.

Convexity warm up

Prove that $L(\beta) = \beta^2$ is convex.

To show: For any $\beta_1, \beta_2, \lambda \in [0, 1]$,
$\lambda L(\beta_1) + (1 - \lambda)L(\beta_2) \geq L(\lambda \cdot \beta_1 + (1 - \lambda) \cdot \beta_2)$

AM-GM Inequality:

Convexity warm up

Prove that $L(\beta) = \beta^2$ is convex.

To show: For any $\beta_1, \beta_2, \lambda \in [0, 1]$,
$\lambda L(\beta_1) + (1 - \lambda)L(\beta_2) \geq L(\lambda \cdot \beta_1 + (1 - \lambda) \cdot \beta_2)$

AM-GM Inequality:

Convexity warm up

Trick for twice differentiable single variable functions: $L(\beta)$ is convex if and only if $L''(\beta) \geq 0$ for all β.

Convexity of least squares regression loss

Prove that $L(\beta) = \|\mathbf{X}\beta - \mathbf{y}\|_2^2$ is convex. I.e. that:

$$\|\mathbf{X}(\lambda\boldsymbol{\beta}_1 + (1-\lambda)\boldsymbol{\beta}_1) - \mathbf{y}\|_2^2 \leq \lambda\|\mathbf{X}\boldsymbol{\beta}_1 - \mathbf{y}\|_2^2 + (1-\lambda)\|\mathbf{X}\boldsymbol{\beta}_2 - \mathbf{y}\|_2^2$$

Left hand side:

$$\|\mathbf{X}(\lambda\boldsymbol{\beta}_1 + (1-\lambda)\boldsymbol{\beta}_1) - \mathbf{y}\|_2^2 = \lambda^2\boldsymbol{\beta}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_1 + 2\lambda(1-\lambda)\boldsymbol{\beta}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_2 + (1-\lambda)^2\boldsymbol{\beta}_2^T\mathbf{X}^T\mathbf{X}$$
$$+ \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T(\lambda\mathbf{X}\boldsymbol{\beta}_1 + (1-\lambda)\lambda\mathbf{X}\boldsymbol{\beta}_2)$$

Right hand side:

$$\lambda\|\mathbf{X}\boldsymbol{\beta}_1 - \mathbf{y}\|_2^2 + (1-\lambda)\|\mathbf{X}\boldsymbol{\beta}_2 - \mathbf{y}\|_2^2 = \lambda\boldsymbol{\beta}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_1 + \lambda\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T(\lambda\mathbf{X}\boldsymbol{\beta}_1) + (1-\lambda)\boldsymbol{\beta}_2^T$$
$$+ (1-\lambda)\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T((1-\lambda)\mathbf{X}\boldsymbol{\beta}_2)$$

Need to show:

$$\lambda^2\boldsymbol{\beta}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_1 + 2\lambda(1-\lambda)\boldsymbol{\beta}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_2 + (1-\lambda)^2\boldsymbol{\beta}_2^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_2 \leq \lambda\boldsymbol{\beta}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_1 + (1-\lambda)\boldsymbol{\beta}_2^T$$

Convexity of least squares regression loss

Vector version of AM-GM:

$$\|\mathbf{a} - \mathbf{b}\|_2^2 = \mathbf{a}^T\mathbf{a} - 2\mathbf{a}^T\mathbf{b} + \mathbf{b}^T\mathbf{b} \geq 0$$
$$2\mathbf{a}^T\mathbf{b} \leq \mathbf{a}^T\mathbf{a} + \mathbf{b}^T\mathbf{b}$$

$$\begin{aligned}\lambda^2\boldsymbol{\beta}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_1 + 2\lambda(1-\lambda)\boldsymbol{\beta}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_2 + (1-\lambda)^2\boldsymbol{\beta}_2^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_2 \\ \leq \lambda^2\boldsymbol{\beta}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_1 + \lambda(1-\lambda)(\boldsymbol{\beta}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_1 + \boldsymbol{\beta}_2^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_2) + (1-\lambda)^2\boldsymbol{\beta}_2^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \\ = \lambda\boldsymbol{\beta}_1^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_1 + (1-\lambda)\boldsymbol{\beta}_2^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}_2\end{aligned}$$

Good exercise: Prove that $L(\boldsymbol{\beta}) = \alpha\|\boldsymbol{\beta}\|_2^2$ is convex.

Convexity of least squares regression loss

Question: Is there a shorter way of proving $L(\boldsymbol{\beta}) = \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2$ is convex? We know it is twice differentiable.

- $\nabla L(\boldsymbol{\beta}) = 2\mathbf{X}^T(\mathbf{X}\boldsymbol{\beta} - \mathbf{y})$
- $\nabla^2 L(\boldsymbol{\beta}) = ?$

Convexity of least squares regression loss

Question: Is there a shorter way of proving $L(\boldsymbol{\beta}) = \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2$ is convex? We know it is twice differentiable.

- $\nabla L(\boldsymbol{\beta}) = 2\mathbf{X}^T(\mathbf{X}\boldsymbol{\beta} - \mathbf{y})$
- $\nabla^2 L(\boldsymbol{\beta}) = 2\mathbf{X}^T\mathbf{X}$

In general, for a scalar-valued multivariate function, all the second order partial derivatives form a matrix called **Hessian matrix**.

$$\begin{bmatrix} \frac{\partial^2 L}{\partial \beta_1^2} & \frac{\partial^2 L}{\partial \beta_1 \partial \beta_2} & \cdots & \frac{\partial^2 L}{\partial \beta_1 \partial \beta_d} \\ \frac{\partial^2 L}{\partial \beta_2 \partial \beta_1} & \frac{\partial^2 L}{\partial \beta_2^2} & \cdots & \frac{\partial^2 L}{\partial \beta_2 \partial \beta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 L}{\partial \beta_n \partial \beta_1} & \frac{\partial^2 L}{\partial \beta_n \partial \beta_2} & \cdots & \frac{\partial^2 L}{\partial \beta_n^2} \end{bmatrix}$$

Convexity using Hessian

Theorem

If f is twice differentiable, then it is convex if and only if $\nabla^2 f(\boldsymbol{x})$ is positive semi-definite.

- This is denoted as $\nabla^2 f(\boldsymbol{x}) \succeq 0$
- Formally, it means for every $\boldsymbol{x} \in Dom(f)$ we have $\boldsymbol{x}^T \nabla^2 f(\boldsymbol{x})\, \boldsymbol{x} \geq 0$.
- Verify that $\nabla^2 L(\beta) = 2\mathbf{X}^T\mathbf{X}$ is positive semi-definite.

$$\nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Rate of convergence for convex functions

We care about how fast gradient descent and related methods converge, not just that they do converge.

- Bounding iteration complexity requires placing some assumptions on $L(\beta)$.
- Stronger assumptions lead to better bounds on the convergence.

Understanding these assumptions can help us design faster variants of gradient descent (there are many!).

Rate of convergence for convex functions

We care about how fast gradient descent and related methods converge, not just that they do converge.

- Bounding iteration complexity requires placing some assumptions on $L(\beta)$.
- Stronger assumptions lead to better bounds on the convergence.

Understanding these assumptions can help us design faster variants of gradient descent (there are many!).

Convergence analysis for convex functions

Assume:

- L is convex.
- Lipschitz function: for all $\boldsymbol{\beta}$, $\|\nabla L(\boldsymbol{\beta})\|_2 \leq G$.
- Starting radius: $\|\boldsymbol{\beta}^* - \boldsymbol{\beta}^{(0)}\|_2 \leq R$.

Gradient descent:

- Choose number of steps T.
- Starting point $\boldsymbol{\beta}^{(0)}$. E.g. $\boldsymbol{\beta}^{(0)} = \mathbf{0}$.
- $\eta = \frac{R}{G\sqrt{T}}$
- For $i = 0, \ldots, T$:
  - $\boldsymbol{\beta}^{(i+1)} = \boldsymbol{\beta}^{(i)} - \eta \nabla L(\boldsymbol{\beta}^{(i)})$
- Return $\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}^{(i)}} L(\boldsymbol{\beta})$.

Gradient descent analysis

Claim (GD Convergence Bound)

If $T \geq \frac{R^2 G^2}{\epsilon^2}$, *then* $L(\hat{\boldsymbol{\beta}}) \leq L(\boldsymbol{\beta}^*) + \epsilon$.

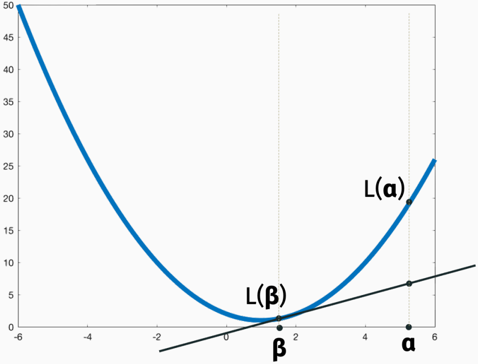
Proof is made tricky by the fact that $L(\boldsymbol{\beta}^{(i)})$ does not improve monotonically. We can "overshoot" the minimum. This is why the step size needs to depend on $1/G$.

Gradient descent

Definition (Alternative Convexity Definition)

A function L is convex if and only if for any β, α:

$$L(\alpha) - L(\beta) \geq \nabla L(\beta)^T(\alpha - \beta)$$



Gradient descent analysis

Claim (GD Convergence Bound)

If $T \geq \frac{R^2 G^2}{\epsilon^2}$ and $\eta = \frac{R}{G\sqrt{T}}$, then $L(\hat{\beta}) \leq L(\beta^*) + \epsilon$.

Claim 1: For all $i = 0, \ldots, T$,

$$L(\beta^{(i)}) - L(\beta^*) \leq \frac{\|\beta^{(i)} - \beta^*\|_2^2 - \|\beta^{(i+1)} - \beta^*\|_2^2}{2\eta} + \frac{\eta G^2}{2}$$

"If you are far away, you make progress towards the optimum".

Claim 1(a): For all $i = 0, \ldots, T$,

$$\nabla L(\beta^{(i)})^T(\beta^{(i)} - \beta^*) \leq \frac{\|\beta^{(i)} - \beta^*\|_2^2 - \|\beta^{(i+1)} - \beta^*\|_2^2}{2\eta} + \frac{\eta G^2}{2}$$

Claim 1 follows from Claim 1(a) by our new definition of convexity.

Gradient descent analysis

Claim (GD Convergence Bound)

If $T \geq \frac{R^2 G^2}{\epsilon^2}$ *and* $\eta = \frac{R}{G\sqrt{T}}$, *then* $L(\hat{\boldsymbol{\beta}}) \leq L(\boldsymbol{\beta}^*) + \epsilon$.

Claim 1(a): For all $i = 0, \ldots, T$, [2]

$$\nabla L(\boldsymbol{\beta}^{(i)})^T(\boldsymbol{\beta}^{(i)} - \boldsymbol{\beta}^*) \leq \frac{\|\boldsymbol{\beta}^{(i)} - \boldsymbol{\beta}^*\|_2^2 - \|\boldsymbol{\beta}^{(i+1)} - \boldsymbol{\beta}^*\|_2^2}{2\eta} + \frac{\eta G^2}{2}$$

[2] Recall that $\|\mathbf{x} - \mathbf{y}\|_2^2 = \|\mathbf{x}\|_2^2 - 2\mathbf{x}^T\mathbf{y} + \|\mathbf{y}\|_2^2$.

Gradient descent analysis

Claim (GD Convergence Bound)

If $T \geq \frac{R^2 G^2}{\epsilon^2}$ and $\eta = \frac{R}{G\sqrt{T}}$, then $L(\hat{\beta}) \leq L(\beta^*) + \epsilon$.

Claim 1: For all $i = 0, \ldots, T$,

$$L(\beta^{(i)}) - L(\beta^*) \leq \frac{\|\beta^{(i)} - \beta^*\|_2^2 - \|\beta^{(i+1)} - \beta^*\|_2^2}{2\eta} + \frac{\eta G^2}{2}$$

Telescoping sum:

$$\begin{aligned}
\sum_{i=0}^{T-1} \left[L(\beta^{(i)}) - L(\beta^*)\right] \leq\ & \frac{\|\beta^{(0)} - \beta^*\|_2^2 - \|\beta^{(1)} - \beta^*\|_2^2}{2\eta} + \frac{\eta G^2}{2} \\
& + \frac{\|\beta^{(1)} - \beta^*\|_2^2 - \|\beta^{(2)} - \beta^*\|_2^2}{2\eta} + \frac{\eta G^2}{2} \\
& + \frac{\|\beta^{(2)} - \beta^*\|_2^2 - \|\beta^{(3)} - \beta^*\|_2^2}{2\eta} + \frac{\eta G^2}{2} \\
& \vdots \\
& + \frac{\|\beta^{(T-1)} - \beta^*\|_2^2 - \|\beta^{(T)} - \beta^*\|_2^2}{2\eta} + \frac{\eta G^2}{2}
\end{aligned}$$

Gradient descent analysis

Claim (GD Convergence Bound)

If $T \geq \frac{R^2 G^2}{\epsilon^2}$ *and* $\eta = \frac{R}{G\sqrt{T}}$*, then* $L(\hat{\boldsymbol{\beta}}) \leq L(\boldsymbol{\beta}^*) + \epsilon$.

Telescoping sum:

$$\sum_{i=0}^{T-1} \left[L(\boldsymbol{\beta}^{(i)}) - L(\boldsymbol{\beta}^*) \right] \leq \frac{\|\boldsymbol{\beta}^{(0)} - \boldsymbol{\beta}^*\|_2^2 - \|\boldsymbol{\beta}^{(T)} - \boldsymbol{\beta}^*\|_2^2}{2\eta} + \frac{T\eta G^2}{2}$$

$$\frac{1}{T} \sum_{i=0}^{T-1} \left[L(\boldsymbol{\beta}^{(i)}) - L(\boldsymbol{\beta}^*) \right] \leq \frac{R^2}{2T\eta} + \frac{\eta G^2}{2}$$

Gradient descent analysis

Claim (GD Convergence Bound)

If $T \geq \frac{R^2 G^2}{\epsilon^2}$ *and* $\eta = \frac{R}{G\sqrt{T}}$, *then* $L(\hat{\boldsymbol{\beta}}) \leq L(\boldsymbol{\beta}^*) + \epsilon$.

Final step:

$$\frac{1}{T}\sum_{i=0}^{T-1}\left[L(\boldsymbol{\beta}^{(i)}) - L(\boldsymbol{\beta}^*)\right] \leq \epsilon$$

$$\left[\frac{1}{T}\sum_{i=0}^{T-1} L(\boldsymbol{\beta}^{(i)})\right] - L(\boldsymbol{\beta}^*) \leq \epsilon$$

We always have that $\min_i L(\boldsymbol{\beta}^{(i)}) \leq \frac{1}{T}\sum_{i=0}^{T-1} L(\boldsymbol{\beta}^{(i)})$, so this is what we return:

$$L(\hat{\boldsymbol{\beta}}) = \min_{i \in 1,\ldots,T} L(\boldsymbol{\beta}^{(i)}) \leq L(\boldsymbol{\beta}^*) + \epsilon.$$

Setting learning rate/step size

Gradient descent algorithm for minimizing $L(\beta)$:

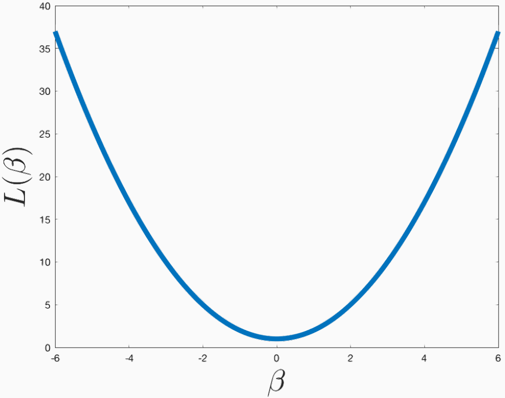
- Choose arbitrary starting point $\boldsymbol{\beta}^{(0)}$.
- For $i = 1, \ldots, T$:
  - $\boldsymbol{\beta}^{(i+1)} = \boldsymbol{\beta}^{(i)} - \eta \nabla L(\boldsymbol{\beta}^{(i)})$
- Return $\boldsymbol{\beta}^{(t)}$.

In practice we don't set the step-size/learning rate parameter $\eta = \frac{R}{G\sqrt{T}}$, since we typically don't know these parameters. The above analysis can also be loose for many functions.

η needs to be chosen sufficiently small for gradient descent to converge, but too small will slow down the algorithm.

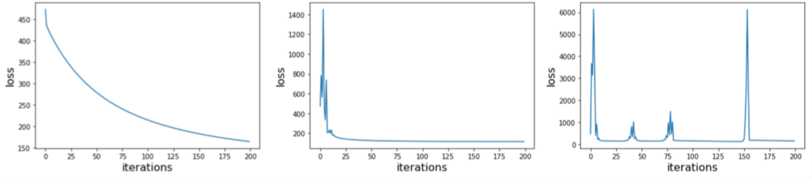
Learning rate

Precision in choosing the learning rate η is not super important, but we do need to get it to the right order of magnitude.



Learning rate

“Overshooting” can be a problem if you choose the step-size too high.



Often a good idea to plot the entire optimization curve for diagnosing what’s going on.

We will have a lab on gradient descent optimization where you’ll get practice doing this.

Exponential grid search

Just as in regularization, search over a grid of possible parameters:

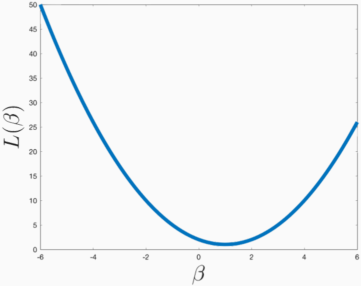
$$\eta = [2^{-5}, 2^{-4}, 2^{-3}, \ldots, 2^{9}, 2^{10}].$$

Or tune by hand based on the optimization curve.

Backtracking line search/armijo rule

Recall: If we set $\beta^{(i+1)} \leftarrow \beta^{(i)} - \eta \nabla L(\beta^{(i)})$ then:

$$L(\beta^{(i+1)}) \approx L(\beta^{(i)}) - \eta \left\langle \nabla L(\beta^{(i)}), \nabla L(\beta^{(i)}) \right\rangle$$
$$= L(\beta^{(i)}) - \eta \|\nabla L(\beta^{(i)})\|_2^2.$$



Approximation holds true for small η. If it holds, error monotonically decreases.

Backtracking line search/armijo rule

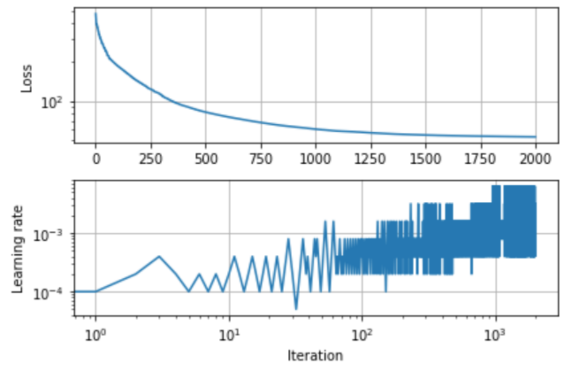
Gradient descent with backtracking line search:

- Choose arbitrary starting point $\boldsymbol{\beta}$.
- Choose starting step size η.
- Choose $\tau, c < 1$ (typically both $c = 1/2$ and $\tau = 1/2$)
- For $i = 1, \ldots, T$:
  - $\boldsymbol{\beta}^{(new)} = \boldsymbol{\beta} - \eta \nabla L(\boldsymbol{\beta})$
  - If $L(\boldsymbol{\beta}^{(new)}) \leq L(\boldsymbol{\beta}) - c\eta \nabla L(\boldsymbol{\beta})$
    - $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}^{(new)}$
    - $\eta \leftarrow \tau^{-1}\eta$
  - Else
    - $\eta \leftarrow \tau\eta$

Always decreases objective value, works very well in practice.

Backtracking line search/armijo rule

Gradient descent with backtracking line search:



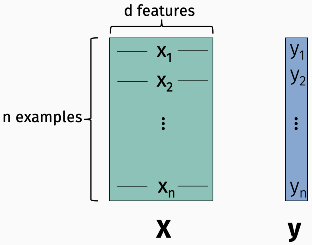
Always decreases objective value, works very well in practice.

Complexity of gradient descent

Complexity of computing the gradient will depend on you loss function.

Example 1: Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be a data matrix.

$$L(\beta) = \|\mathbf{X}\beta - \mathbf{y}\|_2^2 \qquad \nabla L(\beta) = 2\mathbf{X}^T(\mathbf{X}\beta - \mathbf{y})$$



- **Runtime of closed form solution** $\beta^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$:
- **Runtime of one GD step:**

Complexity of gradient descent

Complexity of computing the gradient will depend on you loss function.

Example 1: Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be a data matrix.

$$L(\beta) = -\sum_{i=1}^{n} y_i \log(h(\beta^T \mathbf{x}_i)) + (1 - y_i) \log(1 - h(\beta^T \mathbf{x}_i))$$

$$\nabla L(\beta) = \mathbf{X}^T (h(\mathbf{X}\beta) - \mathbf{y})$$

- **No closed form solution.**
- **Runtime of one GD step:**

Complexity of gradient descent

Frequently the complexity is $O(nd)$ if you have n data-points and d parameters in your model.

Not bad, but the dependence on n can be a lot! n might be on the order of thousands, or millions.

Training neural networks

Stochastic Gradient Descent (SGD).

- Powerful randomized variant of gradient descent used to train machine learning models when n is large and thus computing a full gradient is expensive.

Applies to any loss with finite sum structure:

$$L(\boldsymbol{\beta}) = \sum_{j=1}^{n} \ell(\boldsymbol{\beta}, \mathbf{x}_j, y_j)$$

Stochastic gradient descent

Let $L_j(\boldsymbol{\beta})$ denote $\ell(\boldsymbol{\beta}, \mathbf{x}_j, y_j)$.

Claim: If $j \in 1, \ldots, n$ is chosen uniformly at random. Then:

$$\mathbb{E}\left[n \cdot \nabla L_j(\boldsymbol{\beta})\right] = \nabla L(\boldsymbol{\beta}).$$

$\nabla L_j(\boldsymbol{\beta})$ is called a **stochastic gradient**.

Stochastic gradient descent

SGD iteration:

- Initialize $\beta^{(0)}$.
- For $i = 0, \ldots, T-1$:
  - Choose j uniformly at random.
  - Compute stochastic gradient $\mathbf{g} = \nabla L_j(\beta^{(i)})$.
  - Update $\beta^{(t+1)} = \beta^{(t)} - \eta \cdot n\mathbf{g}$

Move in direction of steepest descent in expectation.

Cost of computing $\boldsymbol{g}$ is independent of n!

Complexity of stochastic gradient descent

Example: Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be a data matrix.

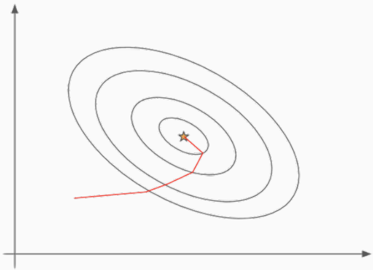
$$L(\beta) = \|\mathbf{X}\beta - \mathbf{y}\|_2^2 = \sum_{j=1}^{n} (y_j - \beta^T \mathbf{x}_j)^2$$

- **Runtime of one SGD step:**

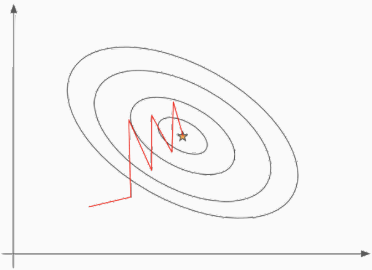
Stochastic gradient descent

Gradient descent: Fewer iterations to converge, higher cost per iteration.

Stochastic Gradient descent: More iterations to converge, lower cost per iteration.



Gradient Descent

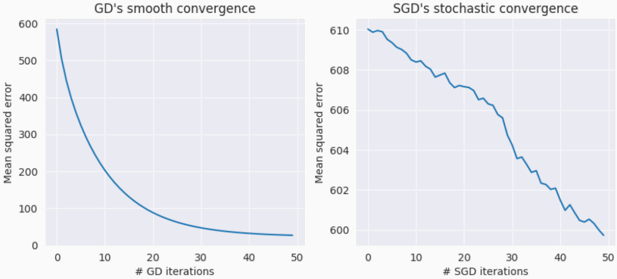


Stochastic Gradient Descent

Stochastic gradient descent

Gradient descent: Fewer iterations to converge, higher cost per iteration.

Stochastic Gradient descent: More iterations to converge, lower cost per iteration.



Stochastic gradient descent in practice

Typical implementation: **Shuffled Gradient Descent.**

Instead of choosing j independently at random for each iteration, randomly permute (shuffle) data and set $j = 1, \ldots, n$. After every n iterations, reshuffle data and repeat.

- Relatively similar convergence behavior to standard SGD.
- **Important term:** one **epoch** denotes one pass over all training examples: $j = 1, \ldots, j = n$.
- Convergence rates for training ML models are often discussed in terms of epochs instead of iterations.

Stochastic gradient descent in practice

Practical Modification: **Mini-batch Gradient Descent.**

Observe that for any batch size s,

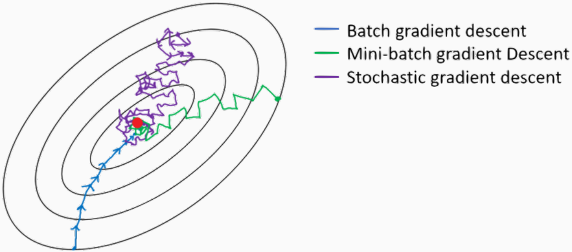
$$\mathbb{E}\left[\frac{n}{s}\sum_{i=1}^{s}\nabla L_{j_i}(\beta)\right] = \nabla L(\beta).$$

if $j_1, \ldots, j_s$ are chosen independently and uniformly at random from $1, \ldots, n$.

Instead of computing a full stochastic gradient, compute the average gradient of a small random set (a mini-batch) of training data examples.

Question: Why might we want to do this?

Mini-batch gradient descent



- Overall faster convergence (fewer iterations needed).

Stochastic gradient descent in practice

Practical Mod. 2: **Per-parameter adaptive learning rate.**

Let $\mathbf{g} = \begin{bmatrix} g_1 \\ \vdots \\ g_p \end{bmatrix}$ be a stochastic or batch stochastic gradient. Our typical parameter update looks like:

$$\beta^{(t+1)} = \beta^{(t)} - \eta \mathbf{g}.$$

We've already seen a simple method for adaptively choosing the learning rate/step size η.

Stochastic gradient descent in practice

Practical Mod. 2: **Per-parameter adaptive learning rate.**

In practice, ML lost functions can often be optimized much faster by using "adaptive gradient methods" like Adagrad, Adadelta, RMSProp, and ADAM. These methods make updates of the form:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \begin{bmatrix} \eta_1 \cdot g_1 \\ \vdots \\ \eta_d \cdot g_d \end{bmatrix}$$

So we have a separate learning rate for each entry in the gradient (e.g. parameter in the model); each $\eta_1, \ldots, \eta_p$ is chosen adaptively.