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

NYU Tandon School of Engineering, Prof. Christopher Musco

Lots of great ideas. Two popular ones:

- Using a different “temperature” for sampling.
- Backing off to a simple model (e.g., short *n*-gram) some of the time to add randomness.

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 (see notes on webpage):

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

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

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

LOGISTIC REGRESSION GRADIENT

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

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

- In contrast to what we saw when minimizing the squared loss for linear regression, there's no simple closed form expression for such a $\boldsymbol{\beta}$!
- 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 $L(\boldsymbol{\beta})$ when we can't explicitly compute where it's gradient is $\mathbf{0}$?

MINIMIZING LOSS FUNCTIONS

Much better idea. Use a guided search approach.

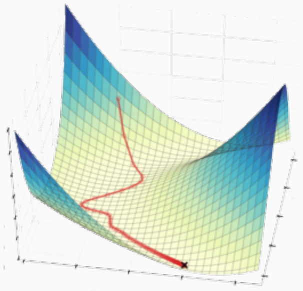
- Start with some $\beta^{(0)}$, and at each step try to change $\boldsymbol{\beta}$ slightly to reduce $L(\boldsymbol{\beta})$.
- Hopefully find an approximate minimizer for $L(\boldsymbol{\beta})$ much more quickly than brute-force search.
- **Concrete goal:** Find $\boldsymbol{\beta}$ with

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

for some small error term ϵ.

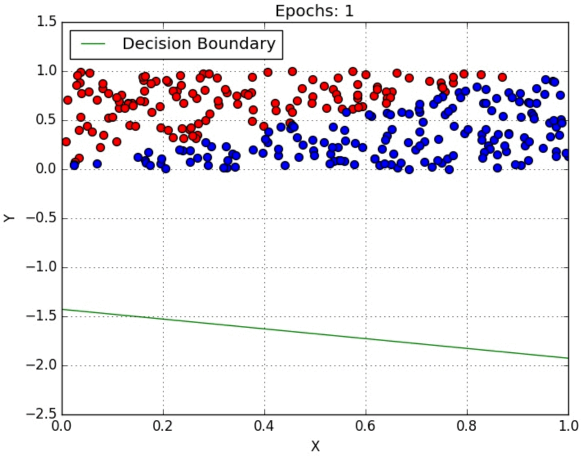
GRADIENT DESCENT

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

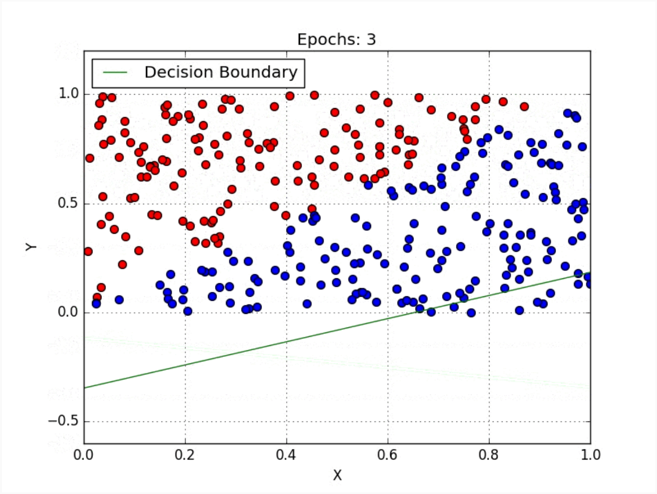


<u>The single most important computational tool in machine learning.</u> And it's remarkable simple + easy to implement.

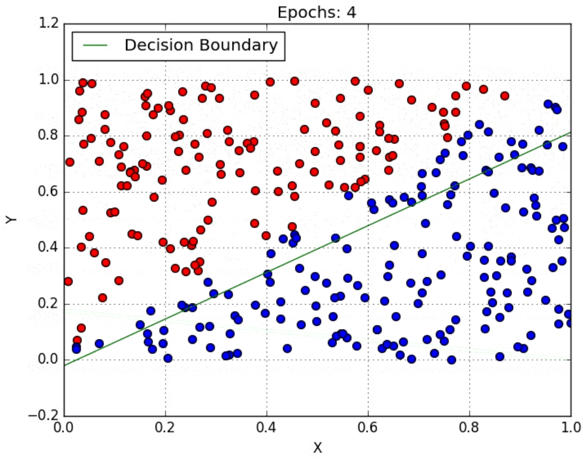
GRADIENT DESCENT VISUALIZATION



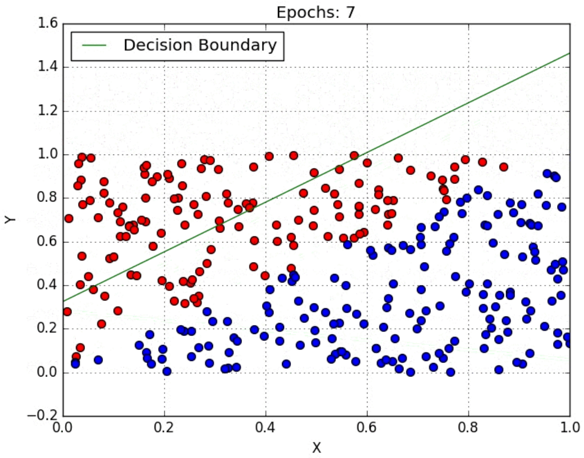
GRADIENT DESCENT VISUALIZATION



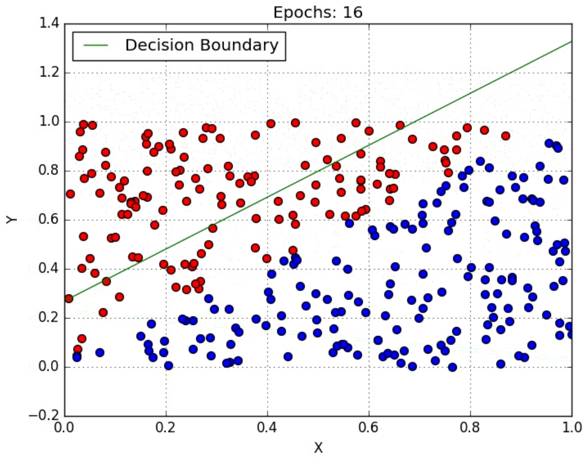
GRADIENT DESCENT VISUALIZATION



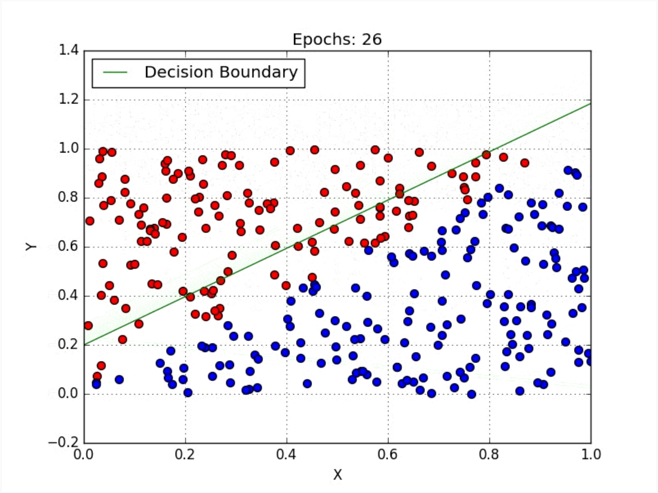
GRADIENT DESCENT VISUALIZATION



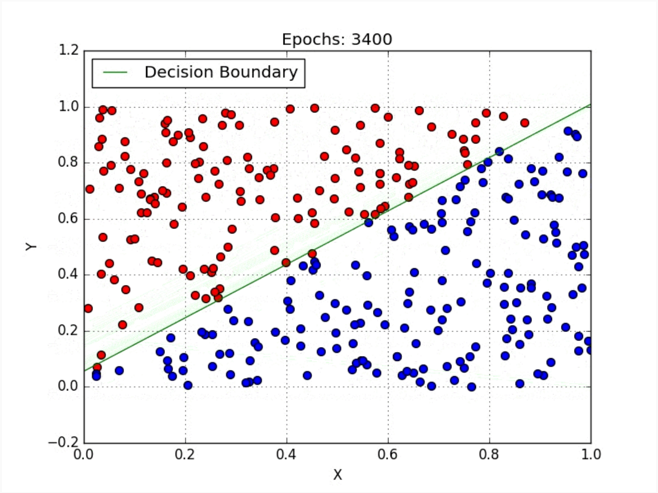
GRADIENT DESCENT VISUALIZATION



GRADIENT DESCENT VISUALIZATION



GRADIENT DESCENT VISUALIZATION



FIRST ORDER OPTIMIZATION

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

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

These are very general assumptions. Gradient descent will not use any other information about the loss function L when trying to find a β which minimizes L.

Basic Gradient descent algorithm:

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

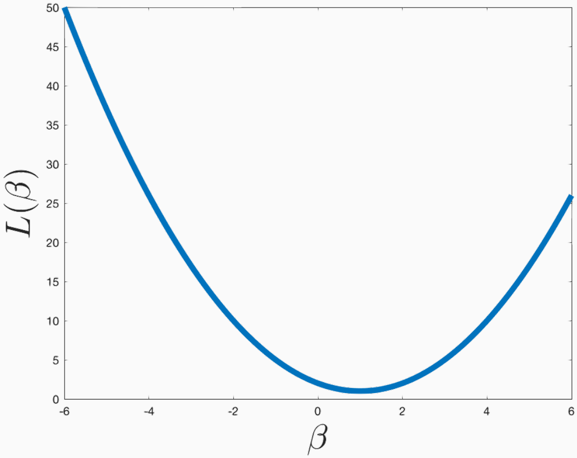
$\eta > 0$ is a step-size parameter. Also called the learning rate.

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, $L'(\beta) = \lim_{\Delta \to 0} \frac{L(\beta+\Delta)-L(\beta)}{\Delta}$. So for small values of Δ, we expect that:

$$L(\beta + \Delta) - L(\beta) \approx \Delta \cdot L'(\beta).$$

We want $L(\beta + \Delta)$ to be <u>smaller</u> than $L(\beta)$, so we want $\Delta \cdot L'(\beta)$ to be negative.

This can be achieved by choosing $\Delta = -L'(\beta)$, or really $\Delta = -\eta \cdot L'(\beta)$ for positive step size η.

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

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: $D_{\mathbf{v}} L(\boldsymbol{\beta}) = \lim_{\Delta \to 0} \frac{L(\boldsymbol{\beta} + \Delta \mathbf{v}) - L(\boldsymbol{\beta})}{\Delta}$. We have that $D_{\mathbf{v}} L(\boldsymbol{\beta}) = \langle \nabla L(\boldsymbol{\beta}), \mathbf{v} \rangle$.

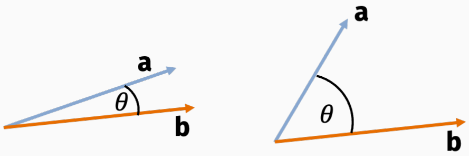
STEEPEST DESCENT

Claim (Gradient descent = Steepest descent[1])

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

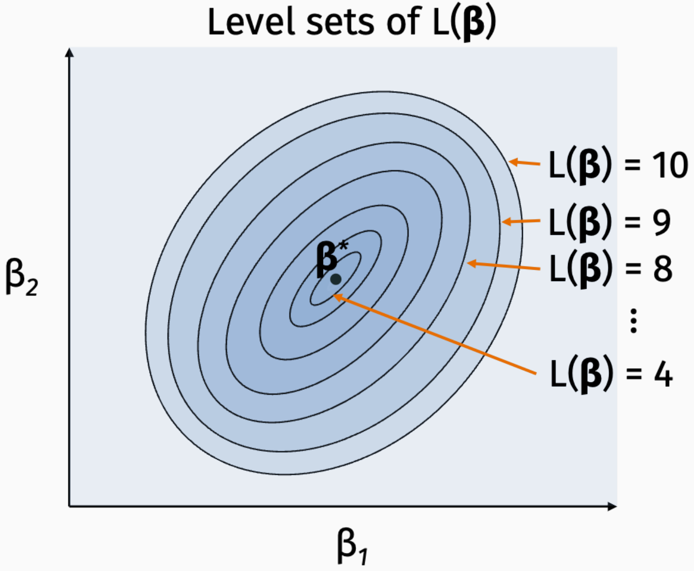
Recall: For two vectors $\mathbf{a}, \mathbf{b}$,

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



[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..

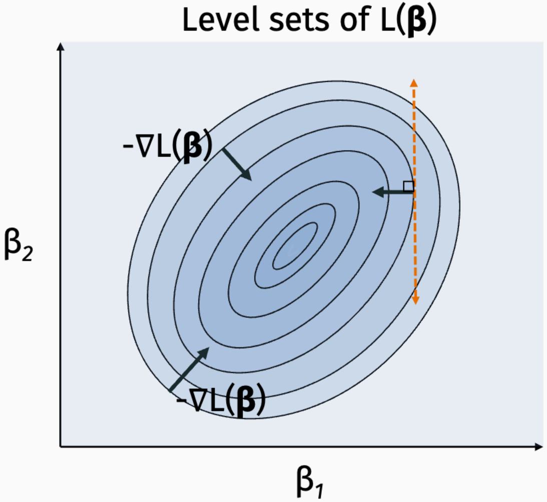
VISUALIZING IN 2D


Level sets of $L(\boldsymbol{\beta})$
β_2
β_1
$\boldsymbol{\beta}^*$
$L(\boldsymbol{\beta}) = 10$
$L(\boldsymbol{\beta}) = 9$
$L(\boldsymbol{\beta}) = 8$
$\vdots$
$L(\boldsymbol{\beta}) = 4$

STEEPEST DESCENT

Claim (Gradient descent = Steepest descent)

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



Basic Gradient descent (GD) algorithm:

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

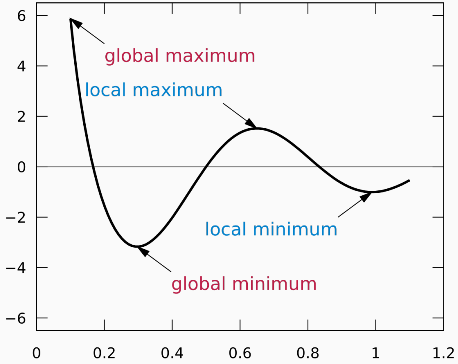
Claim

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.



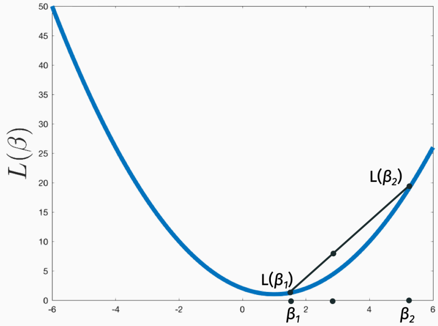
Very unlikely to land precisely on another stationary point and get stuck. Non-minimal stationary points are “unstable”.

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

Definition (Convex)

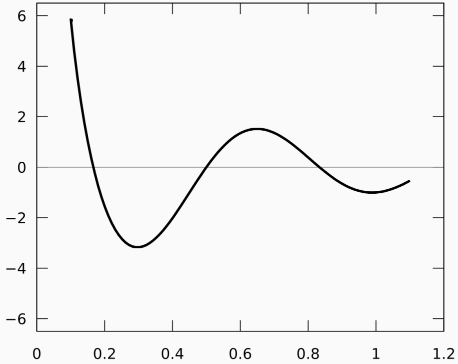
A function L is convex iff for any $\beta_1, \beta_2, \lambda \in [0,1]$:

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



CONVEX FUNCTION

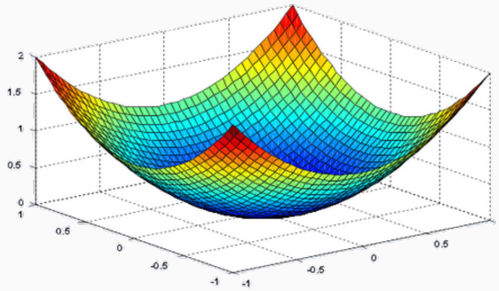
In words: A function is convex if a line between any two points on the function lies above the function. Captures the notion that a function looks like a bowl.



This function **is not** convex.

CONVEX FUNCTION

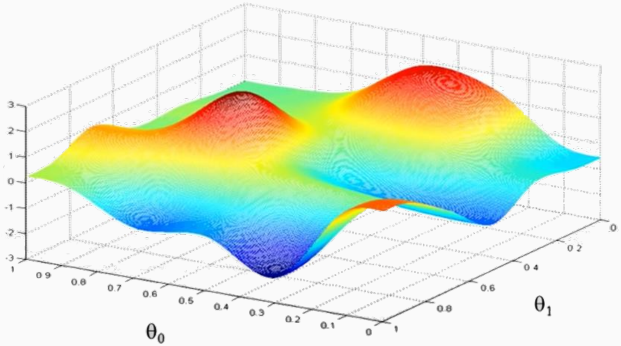
In words: A function is convex if a line between any two points on the function lies above the function. Captures the notion that a function looks like a bowl.



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.

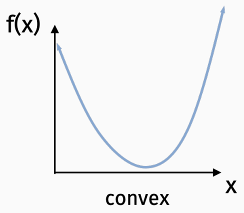


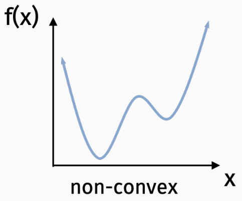
This function **is not** convex.

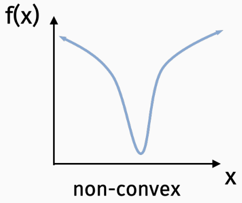
What functions are convex?

- 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.

Vary in how "bad" the non-convexity is. For example, some matrix factorization problems are non-convex but still only have global minima.

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

Analog for higher dimensional functions is clunky. Need to prove that the Hessian matrix, $\mathbf{H} \in \mathbb{R}^{d\times d}$, is positive semi-definite.

$$\mathbf{H}_{ij} = \frac{\partial^2 L}{\partial \beta_i \partial \beta_j}$$

CONVEXITY OF LEAST SQUARES REGRESSION LOSS

Prove that $L(\boldsymbol{\beta}) = \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2$ 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:

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

Right hand side:

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

Need to show:

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

CONVEXITY OF LEAST SQUARES REGRESSION LOSS

Vector version of AM-GM:

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

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

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

RATE OF CONVERGENCE FOR CONVEX FUNCTIONS

Claim: For any convex function $L(\boldsymbol{\beta})$, gradient descent with sufficiently small step size η converges to the global minimum $\boldsymbol{\beta}^*$ of L.

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

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)

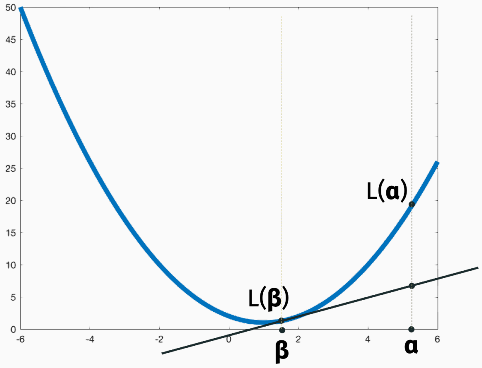
If $T \geq \frac{R^2 G^2}{\epsilon^2}$, then $L(\hat{\beta}) \leq L(\beta^*) + \epsilon$.

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

Definition (Alternative Convexity Definition)

A function L is convex if and only if for any $\boldsymbol{\beta}, \boldsymbol{\alpha}$:

$$L(\boldsymbol{\alpha}) - L(\boldsymbol{\beta}) \leq \nabla L(\boldsymbol{\beta})^T(\boldsymbol{\alpha} - \boldsymbol{\beta})$$



GRADIENT DESCENT ANALYSIS

Claim (GD Convergence Bound)

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

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

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

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

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

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

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

GRADIENT DESCENT ANALYSIS

Claim (GD Convergence Bound)

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

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

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

Telescoping sum:

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

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

Final step:

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

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

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

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

setting learning rate/step size

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

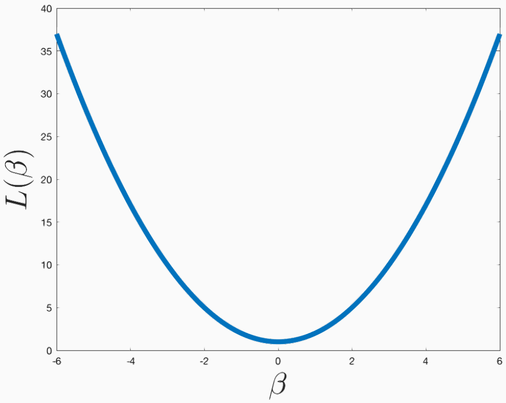
- Choose arbitrary starting point $\boldsymbol{\beta}^{(0)}$.
- For $i = 1, \ldots, T$:
    - $\boldsymbol{\beta}^{(i+1)} = \boldsymbol{\beta}^{(i)} - \eta \nabla L(\boldsymbol{\beta}^{(i)})$
- Return $\boldsymbol{\beta}^{(T)}$.

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.



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 $\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}^{(i)}} L(\boldsymbol{\beta})$.

This result tells us exactly how to set the learning rate η.

SETTING LEARNING RATE

But...

- We don't usually know R or G in advance. We might not even know T.
- Even if we did, setting $\eta = \frac{R}{G\sqrt{T}}$ tends to be a very conservative in practice. The choice 100% leads to convergence, but usually to fairly slow convergence.
- What if L is not convex?

FIRST APPROACH: EXPONENTIAL GRID SEARCH

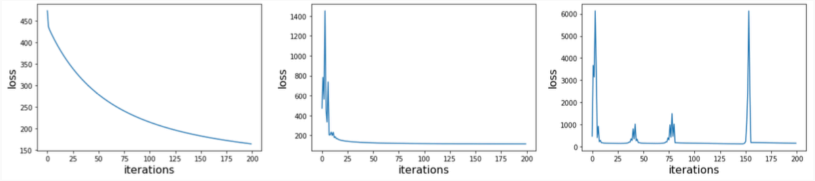
Just as in regularization, search over a grid of possible parameters:

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

Can manually check if we are converging too slow or undershooting by plotting the optimization curve.

LEARNING RATE

Plot’s of loss vs. number of iterations for three difference choices of step size.



BACKTRACKING LINE SEARCH/ARMIJO RULE

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

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

Approximation holds for small η. If it holds, maybe we could get away with a larger η. If it doesn't, we should probably reduce η.

BACKTRACKING LINE SEARCH/ARMIJO RULE

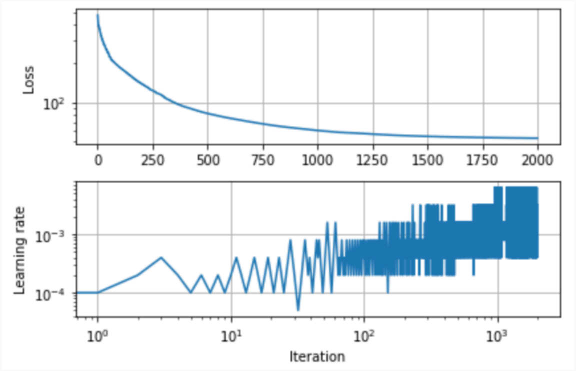
Gradient descent with backtracking line search:

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

Always decreases objective value, works very well in practice.

BACKTRACKING LINE SEARCH/ARMIJO RULE

Gradient descent with backtracking line search:



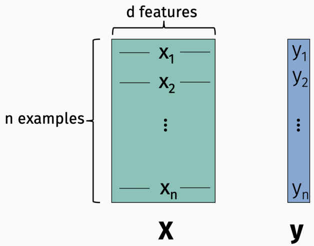
Always decreases objective value, works very well in practice. We will see this in a lab.

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.

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



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

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

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

- 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. This will also be the case for neural networks.

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

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 from $\{1, 2, \ldots, n\}$.
    - 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 <u>in expectation.</u>

Cost of computing $\mathbf{g}$ is <u>independent</u> of n!

COMPLEXITY OF STOCHASTIC GRADIENT DESCENT

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

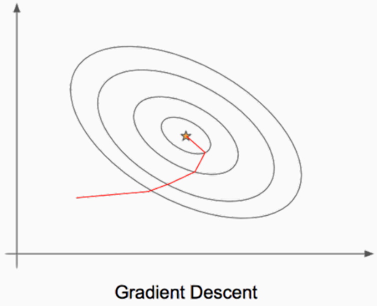
$$L(\boldsymbol{\beta}) = \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2 = \sum_{j=1}^{n} (y_j - \boldsymbol{\beta}^T \mathbf{x}_j)^2$$

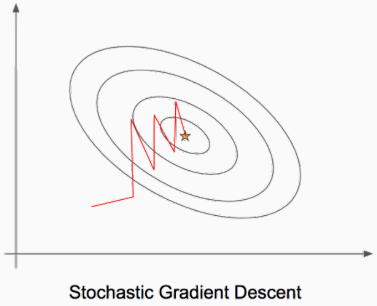
- Runtime of one SGD step:

STOCHASTIC GRADIENT DESCENT

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

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

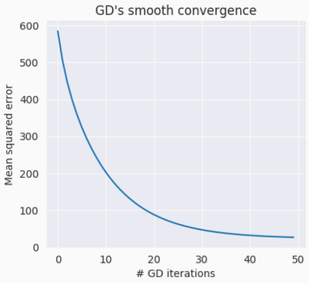


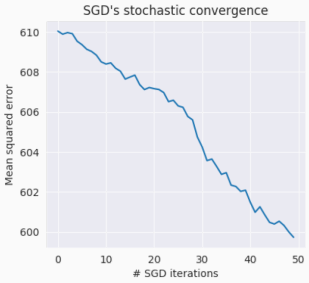


STOCHASTIC GRADIENT DESCENT

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,

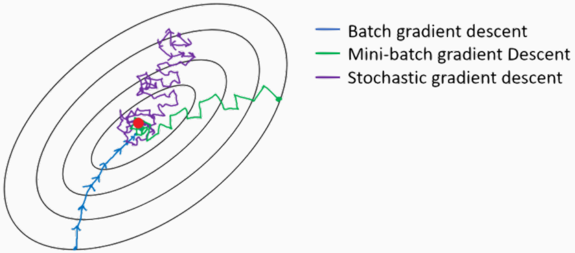
$$\mathbb{E}\left[\frac{n}{s}\sum_{i=1}^{s}\nabla L_{j_i}(\beta)\right] = \nabla L(\beta).$$

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:

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

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

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). And each $\eta_1, \ldots, \eta_p$ is chosen adaptively.

MIDTERM

- 1.5 hours long, but should take 1 hour. Here in the classroom.
- Will have a short lecture after exam/break.
- You can bring in a single, 2-sided cheat sheet with terms, definitions, etc.
- Mix of short answer questions (true/false, matching, etc.) and questions similar to the homework but easier.
- Covers everything through <u>last class</u>. Don't need to know gradient descent or optimization.