CS-UY 4563: Lecture 10
Gradient Descent

NYU Tandon School of Engineering, Prof. Christopher Musco

LOGISTICS

- Homework due Wednesday, lab Thursday.
- My office hours moved to **4-6pm** on Wednesday.
- Midterm exam next Monday.
    - I will post a list of topics and sample questions shortly.
    - Questions will be similar to written homework, but shorter.
- Wednesday's class is going to be a review day. We've already learned a lot!
    - We will go through sample problems, and problems that were sticking points on the homework.

A FEW COMMENTS ON CLASSIFICATION

Last class we learned about linear classification and logistic regression which is a specific model/loss function combo which works particularly well for finding good linear classifiers.

And for learning non-linear classifiers when combined with feature transformations!

Point mentioned at end of last class:

- In classification problem, minimizing error rate doesn't always make the most sense.
- Sometimes false positives have a different (real world) cost than false negatives.
- Instead consider metrics like precision and recall.

MULTICLASS

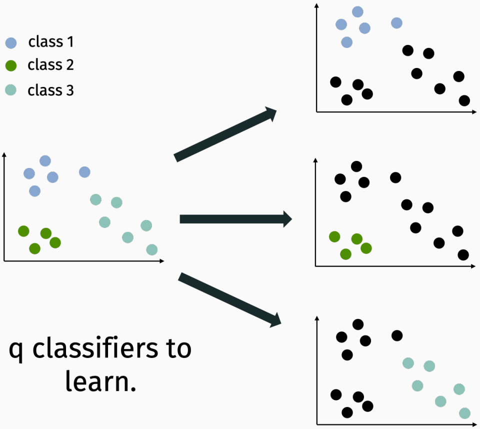
What about when $y \in \{1, \ldots, q\}$ instead of $y \in \{0, 1\}$

Two options for multiclass data:

- One-vs.-all (most common, also called one-vs.-rest)
- One-vs.-one (slower, but can be more effective)

In both cases, we convert to multiple binary classification problems.

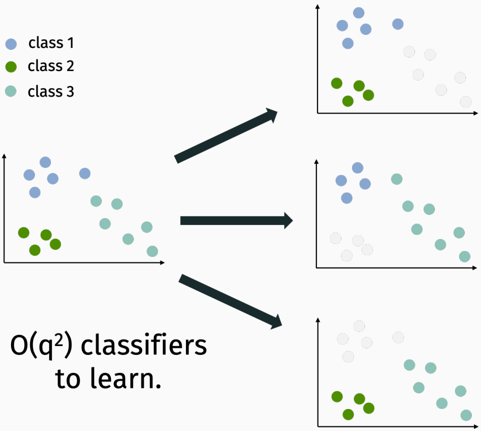
ONE VS. REST



q classifiers to learn.

- For q classes train q classifiers. Obtain parameters $\vec{\beta}_1, \dots, \vec{\beta}_q$.
- Assign y to class i with maximum $\langle \vec{\beta}_i, \vec{x} \rangle$.

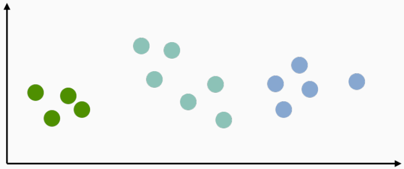
ONE VS. REST



- For q classes train $\frac{q(q-1)}{2}$ classifiers.
- Assign y to class which i which wins in the most number of head-to-head comparisons.

ONE VS. ONE

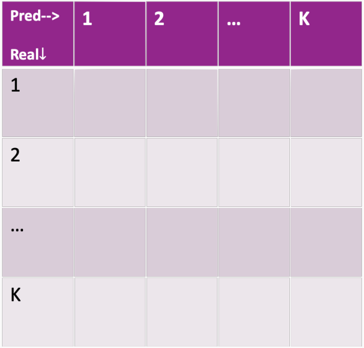
Hard case for one-vs.-all.



- One-vs.-one would be a better choice here.
- Also tends to work better when there is class in balance.

error in (multiclass) classification

Confusion matrix for k classes:



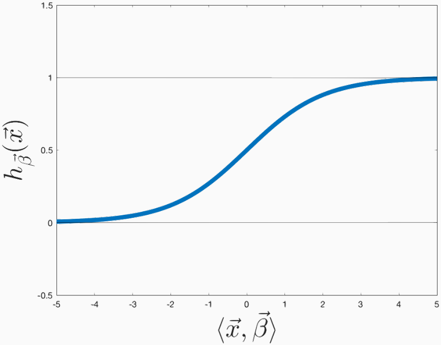
Pred--> Real↓	1	2	...	K
1				
2				
...				
K				

- Entry i, j is the fraction of class i items classified as class j.
- Overall accuracy is the average of the diagonals.
- Useful to see whole matrix to visualize where errors occur.

LOGISTIC REGRESSION

Let $h_{\vec{\beta}}(\vec{x})$ be the logistic function:

$$h_{\vec{\beta}}(\vec{x}) = \frac{1}{1 + e^{-\langle \vec{\beta}, \vec{x} \rangle}}$$



LOGISTIC REGRESSION

- **Model**: Let $h_{\vec{\beta}}(\vec{x}) = \frac{1}{1+e^{-\langle \vec{\beta}, \vec{x} \rangle}}$

$$f_{\vec{\beta}}(\vec{x}) = \mathbb{1}\left[h_{\vec{\beta}}(\vec{x}) > 1/2\right]$$

- **Loss function**: “Logistic loss” aka “Cross-entropy loss”

$$L(\vec{\beta}) = -\sum_{i=1}^{n} y_i \log(h_{\vec{\beta}}(\vec{x})) + (1 - y_i) \log(1 - h_{\vec{\beta}}(\vec{x}))$$

How do we find $\vec{\beta}$ which minimizes $L(\vec{\beta})$?

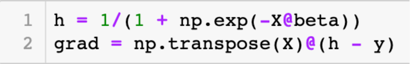
LOGISTIC REGRESSION GRADIENT

$$L(\vec{\beta}) = -\sum_{i=1}^{n} y_i \log(h_{\vec{\beta}}(\vec{x})) + (1 - y_i)\log(1 - h_{\vec{\beta}}(\vec{x}))$$

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

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

where $h_{\vec{\beta}}(\mathbf{X}) = \frac{1}{1+e^{-\mathbf{X}\vec{\beta}}}$. 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 $\vec{\beta}$ minimizing $L(\vec{\beta})$ we need to find a $\vec{\beta}$ where:

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

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

MINIMIZING LOSS FUNCTIONS

First idea. Brute-force search. Test our many possible values for $\vec{\beta}$ and just see which gives the smallest value of $L(\vec{\beta})$.

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

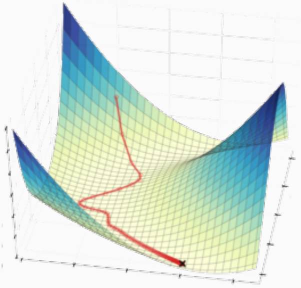
MINIMIZING LOSS FUNCTIONS

Much Better idea. Some sort of guided search for a good of $\vec{\beta}$.

- Start with some $\vec{\beta}_0$, and at each step try to change $\vec{\beta}$ slightly to reduce $L(\vec{\beta})$.
- Hopefully find an approximate minimizer for $L(\vec{\beta})$ much more quickly than brute-force search.
- **Concrete goal:** Find $\vec{\beta}$ with $L(\vec{\beta}) < \min_{\vec{\beta}} L(\vec{\beta}) + \epsilon$ for some small error term ϵ.

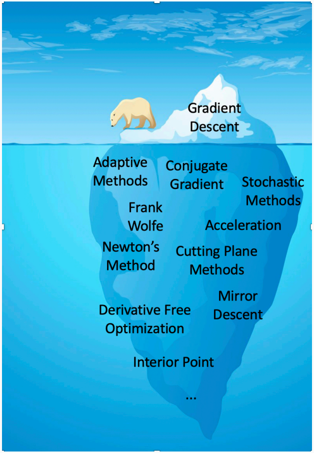
GRADIENT DESCENT

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: take my class in the fall (CS-GY 9223I) if you want to learn more.

FIRST ORDER OPTIMIZATION

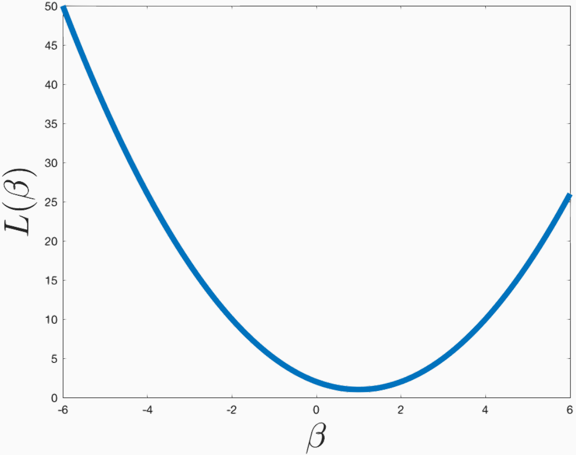
First order oracle model: Given a function L to minimize, assume we can:

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

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

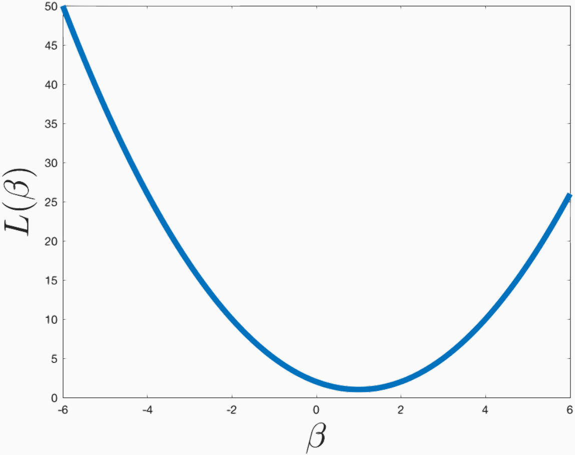
INTUITION

Consider a 1-dimensional loss function. I.e. where β has just one entry:



INTUITION

Consider a 1-dimensional loss function. I.e. where β has just one entry:



GRADIENT DESCENT IN 1D

Recap:

- Consider an algorithm which incrementally adjusts β. I.e. at each step $\beta \leftarrow \beta + \eta$ for some small η. Our goal is to "make progress" towards minimizing L, which means we want $L(\beta + \eta) < L(\beta)$.
- For a 1D function, $\nabla L(\beta) = L'(\beta)$.
- So, for small η, $L(\beta + \eta) - L(\beta) \approx \nabla L(\beta) \cdot \eta$.
- Want right hand side $\nabla L(\beta) \cdot \eta$ to be $\underline{\text{negative}}$.
- So choose η to be positive if $\nabla L(\beta)$ is negative, and negative if $\nabla L(\beta)$ is positive.

This is Gradient Descent (in 1D)!

DIRECTIONAL DERIVATIVES

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

$$\vec{\beta} \leftarrow \vec{\beta} + \vec{v}.$$

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

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

DIRECTIONAL DERIVATIVES

$$\begin{aligned} L(\vec{\beta} + \vec{v}) - L(\vec{\beta}) &\approx \frac{\partial L}{\partial \beta_1} v_1 + \frac{\partial L}{\partial \beta_2} v_2 + \ldots + \frac{\partial L}{\partial \beta_d} v_d \\ &= \langle \nabla L(\vec{\beta}), \vec{v} \rangle. \end{aligned}$$

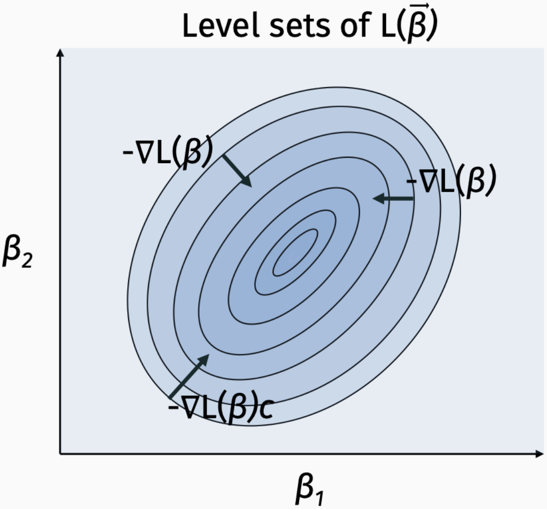
How should we choose $\vec{v}$ so that $L(\vec{\beta} + \vec{v}) < L(\vec{\beta})$?

steepest descent

Claim (Gradient descent = Steepest descent[1])

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

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


Level sets of $L(\vec{\beta})$
$-\nabla L(\beta)$
$-\nabla L(\beta)$
$-\nabla L(\beta)c$
β_2
β_1

Gradient algorithm:

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

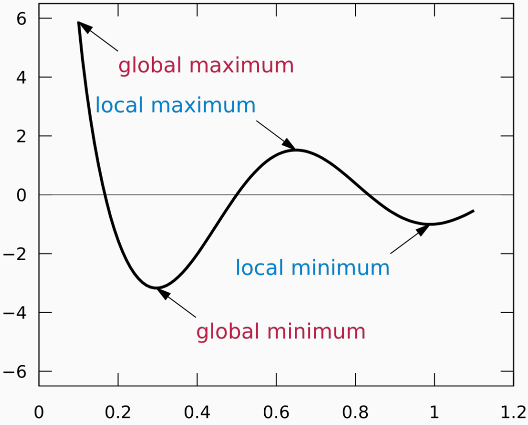
η is a step-size parameter. Also called the learning rate. Needs to be chosen sufficiently small for gradient descent to converge, but too small will slow down the algorithm. Often “tuned” by trying out many different values.

CONVERGENCE OF GRADIENT DESCENT

Does gradient descent converge for all loss functions L?

CONVERGENCE OF GRADIENT DESCENT

In general GD only converges to a local minimum.

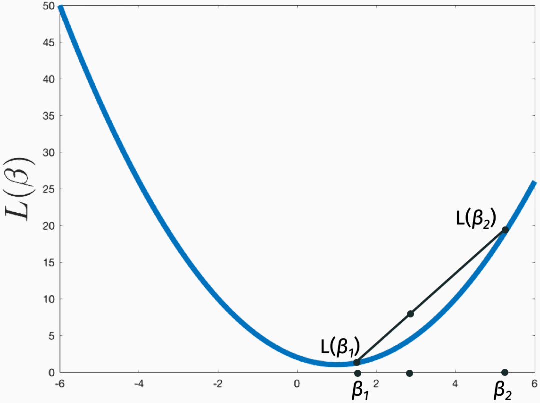


CONVEX FUNCTION

Definition (Convex)

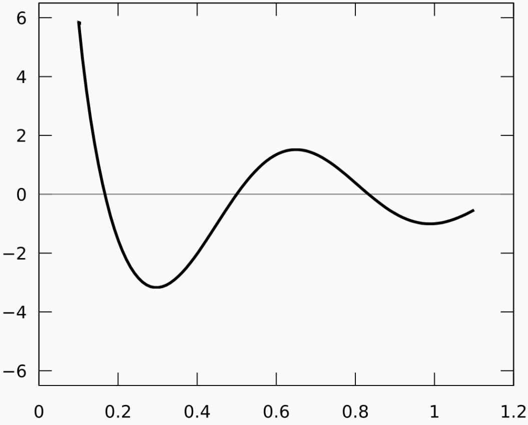
A function L is convex iff for any $\vec{\beta}_1, \vec{\beta}_2, \lambda \in [0, 1]$:

$$(1-\lambda) \cdot L(\vec{\beta}_1) + \lambda \cdot L(\vec{\beta}_2) \geq L\left((1-\lambda) \cdot \vec{\beta}_1 + \lambda \cdot \vec{\beta}_2\right)$$



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.



CONVEX FUNCTION

Claim (Convex Function Minimizers.)

Every local minimum of a convex function is also a global minimum.

Claim (GD Convergence for Convex Functions.)

For sufficiently small step-size η, Gradient Descent converges to the global minimum of any convex function L.

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!

This is not a coincidence: often it makes sense to reformulate your problem so that the loss function is convex, simply so you can minimize it with GD.

CONVERGENCE OF GRADIENT DESCENT

Thing we will talk about after the midterm + spring break:

- Even though GD always converges for a convex function, it's rate of convergence can vary widely. There are lots of methods to speed up the algorithm.
- To implement GD, we need to compute $\nabla L(\vec{\beta})$ at every iteration. Typically pretty cheap, but not always for huge datasets. We will see an alternative approach called stochastic gradient descent (SGD) to address this issue.
- What happens when we apply GD to non-convex functions?