CS-UY 4563: Lecture 11
Finish-Up Gradient Descent, Midterm Review

NYU Tandon School of Engineering, Prof. Christopher Musco

GRADIENT DESCENT

- We want to choose $\vec{\beta}$ to minimize a loss function $L(\vec{\beta})$.
- Often we can compute $\nabla L(\vec{\beta})$ for any $\vec{\beta}$, but can't explicitly find a $\vec{\beta}^*$ for which $\nabla L(\vec{\beta}) = \vec{0}$.
- Instead, we iteratively search a near optimal $\vec{\beta}$.

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

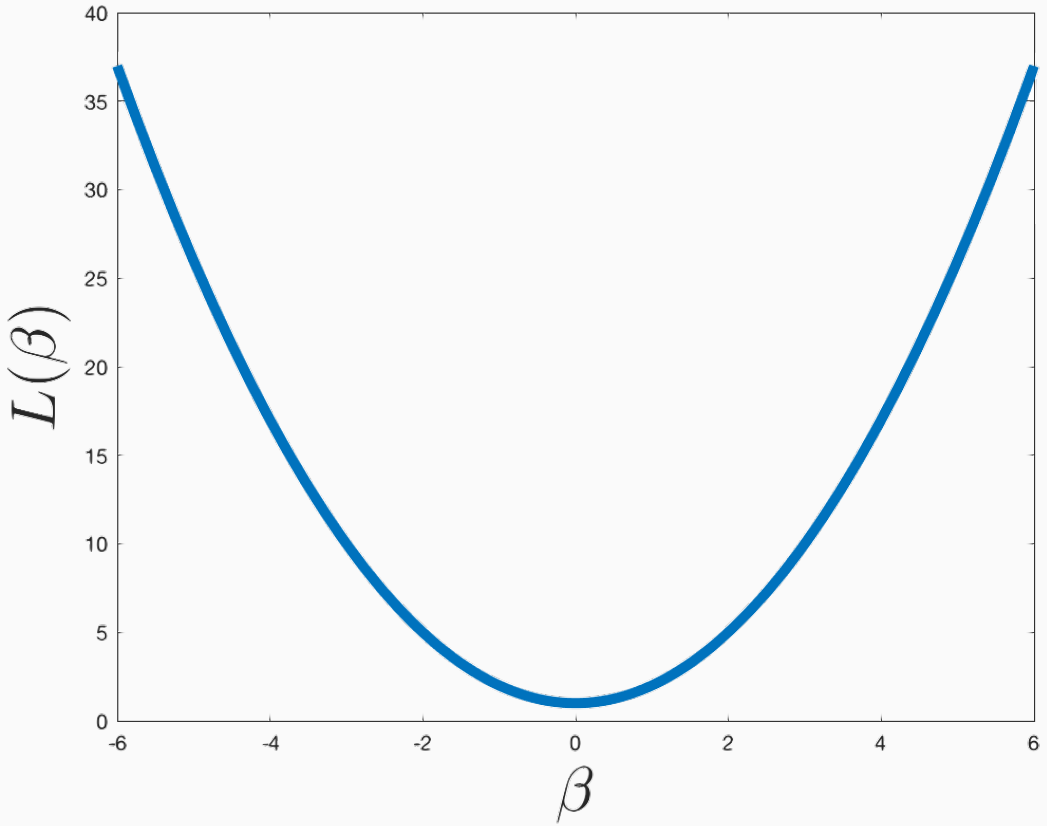
- Choose arbitrary starting point $\vec{\beta}^{(0)}$.
- For $i = 1, \ldots, T$:
  - $\vec{\beta}^{(i+1)} = \vec{\beta}^{(i)} - \eta \nabla L(\vec{\beta}^{(i)})$
- Return $\vec{\beta}^{(t)}$.

Or stop after $L(\beta^{(i)})$ stops decreasing.

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.

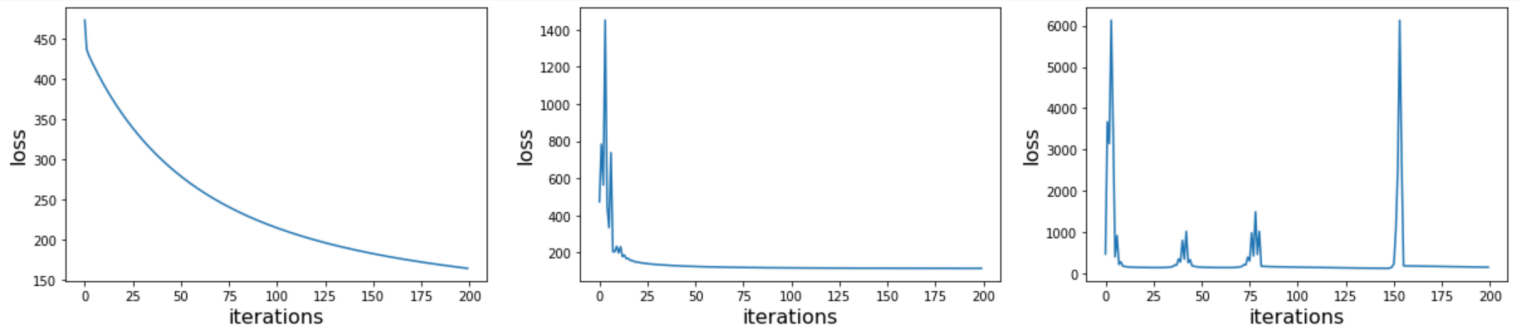
LEARNING RATE

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 mini-lab on gradient descent optimization after the midterm we're you'll get practice doing this.

EXPONENTIAL GRID SEARCH

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

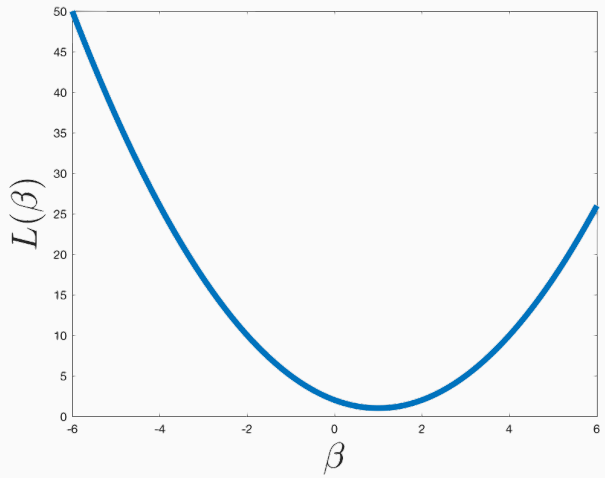
$$\eta = [2^{-5}, 2^{-4}, 2^{-3}, \ldots, 2^{9}, 2^{10}].$$

Or tune by hand based on the optimization curve.

BACKTRACKING LINE SEARCH/ARMIJO RULE

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

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



Approximation holds true for small η. When it does not, we might be overshooting.

Backtracking line search/Armijo rule

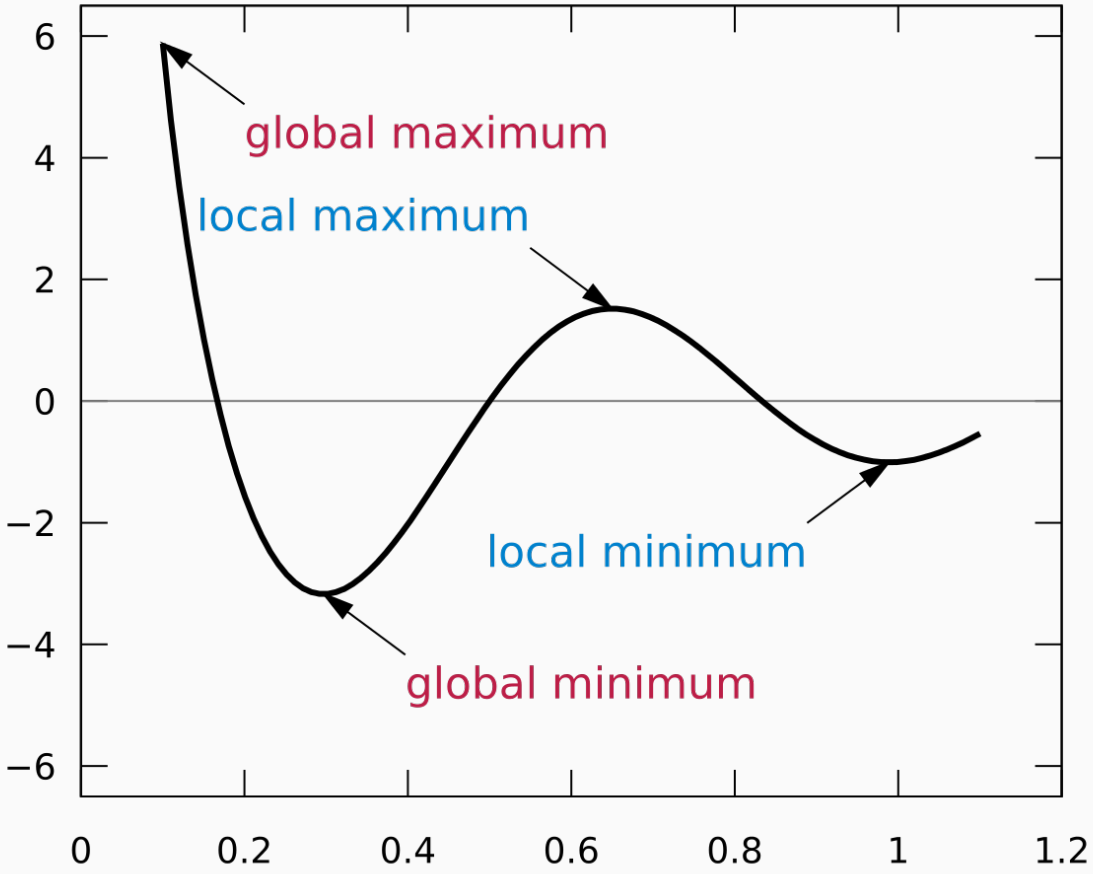
Gradient descent with backtracking line search:

- Choose arbitrary starting point $\vec{\beta}$.
- Choose starting step size η.
- Choose $\tau, c < 1$ (typically both $c = 1/2$ and $\tau = 1/2$)
- For $i = 1, \ldots, T$:
  - $\vec{\beta}^{(new)} = \vec{\beta} - \eta \nabla L(\vec{\beta})$
  - If $L(\vec{\beta}^{(new)}) \leq L(\vec{\beta}) - c\eta \|\nabla L(\vec{\beta})\|_2^2$
    - $\vec{\beta} \leftarrow \vec{\beta}^{(new)}$
    - $\eta \leftarrow \tau^{-1}\eta$
  - Else
    - $\eta \leftarrow \tau\eta$

Always decreases objective value, works very well in practice.

CONVERGENCE OF GRADIENT DESCENT

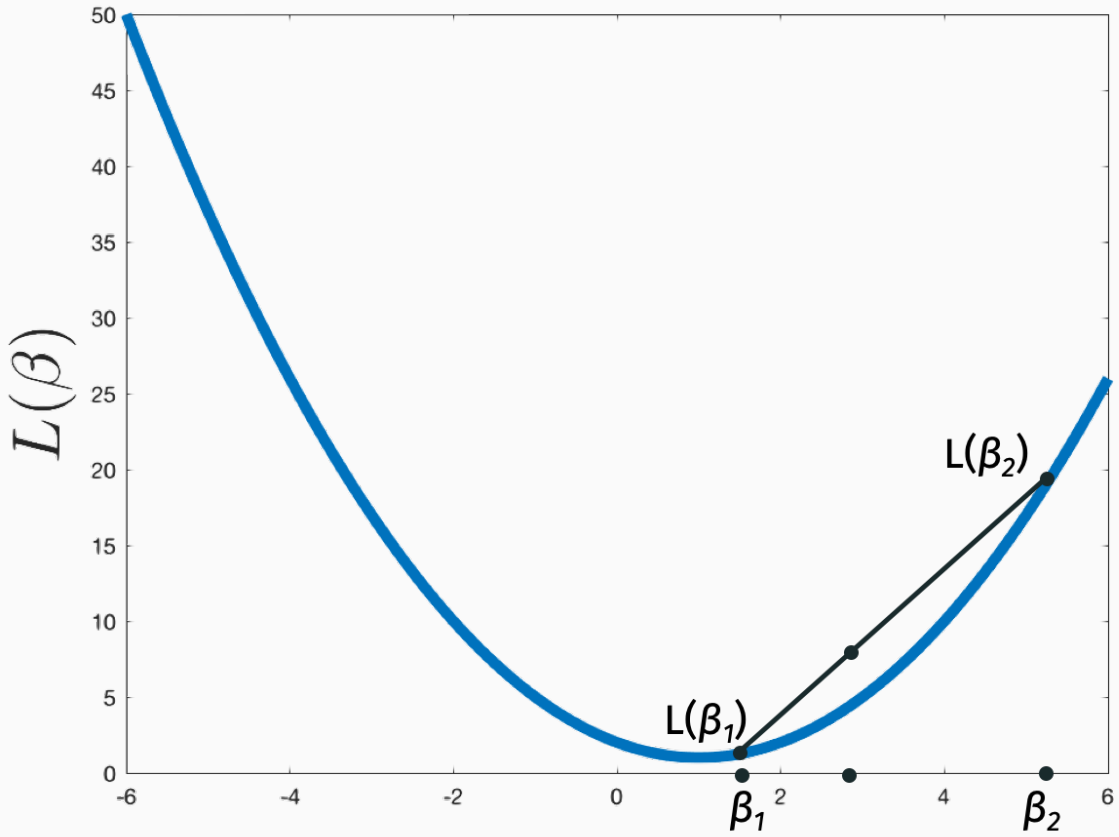
In general GD only converges to a local minimum.



Definition (Convex)

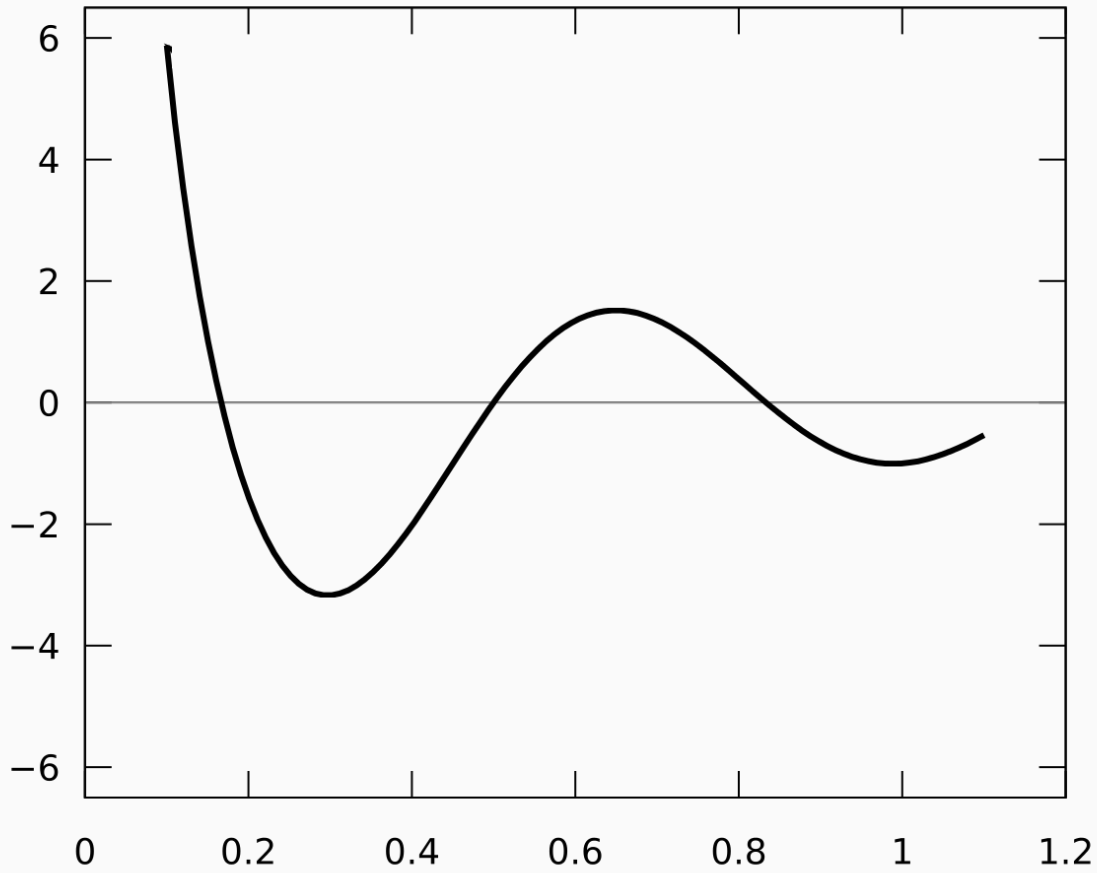
A function L is convex iff for any $\vec{\beta}_1, \vec{\beta}_2, \lambda \in [0, 1]$:

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



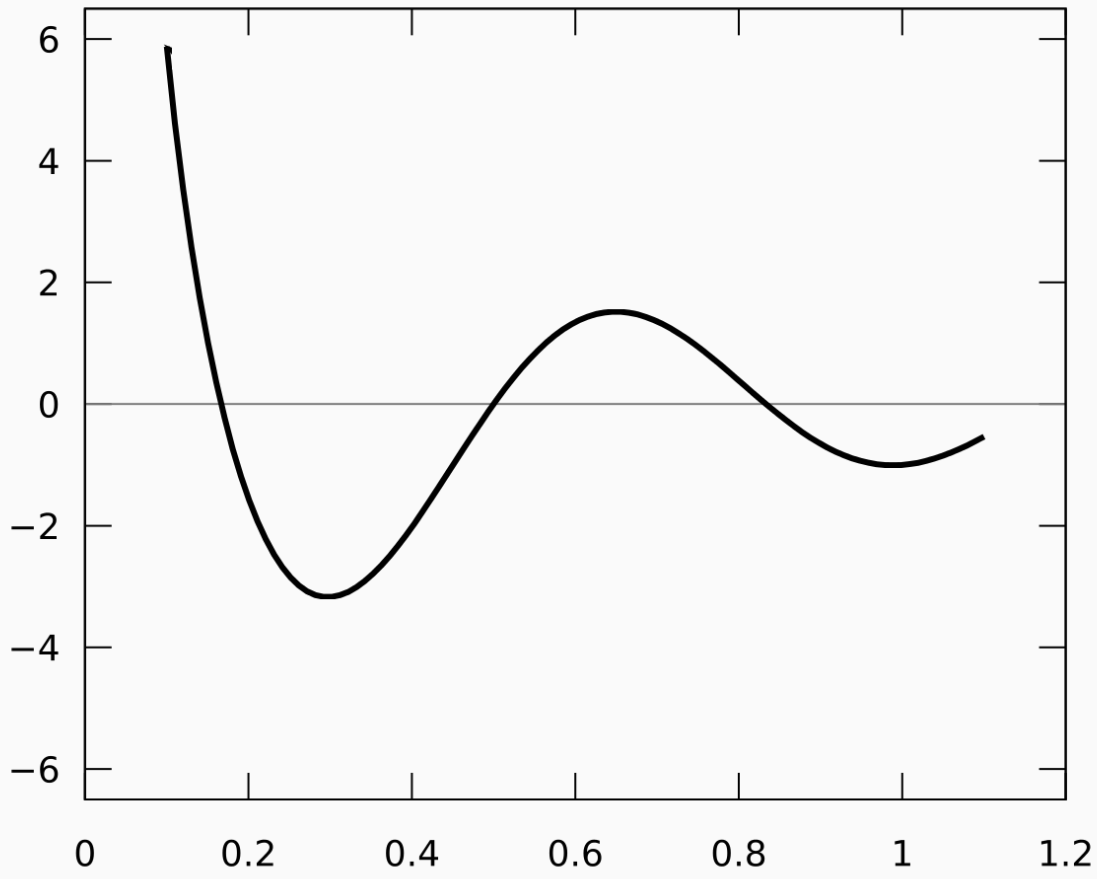
CONVEX FUNCTION

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.



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 an approximate 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.

MIDTERM REVIEW

Problem 2: Thinking About Data Transformations (10pts)

You are trying to fit a multiple linear regression model for a given data set. You have already transformed your data by appending a column of all ones, which resulted in a final data matrix:

$$X = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ 1 & x_{2,1} & x_{2,2} & \dots & x_{2,d} \\ \vdots & \vdots & & \vdots & \\ 1 & x_{n,1} & x_{n,2} & \dots & x_{n,d} \end{bmatrix}$$

However, your model does not seem to be working well. It obtains poor loss in both training and test.

(a) A friend suggests that you should try mean centering your data columns. In other words, for each i, compute the column mean $\bar{x}_i = \frac{1}{n}\sum_{j=1}^{n} x_{j,i}$ and subtract $\bar{x}_i$ from every entry in column i. Note that we won't mean center the first column, as doing so would set the 1s to 0s. Using Python broadcasting you might mean center by running:

```
T = X[:,1:]
X[:,1:] = T - np.mean(T, axis=0)
```

Do you expect your friend's suggestion to improve the performance of the linear model. Will it help in all cases? Some cases? No cases?

MEAN CENTERING HAS NO EFFECT ON LINEAR REGRESSION

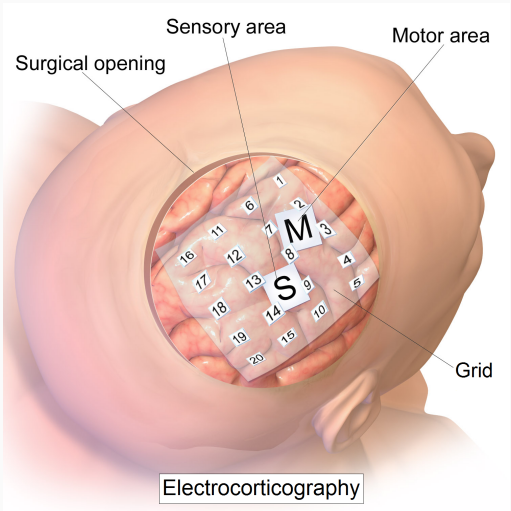
MEAN CENTERING HAS NO EFFECT ON LINEAR REGRESSION

COLUMN SCALING HAS NO EFFECT ON LINEAR REGRESSION

COLUMN SCALING HAS NO EFFECT ON LINEAR REGRESSION

Electrocorticography ECoG lab:

- Implant grid of electrodes on surface of monkey's brain to measure electrical activity in different regions.



- Predict hand motion based on 53 ECoG measurements.
- **Model order:** predict movement at time t using brain signals at time $t, t-1, \ldots, t-q$ for varying values of q.

DATA TRANSFORMATIONS FOR TEMPORAL DATA

DATA TRANSFORMATIONS FOR TEMPORAL DATA