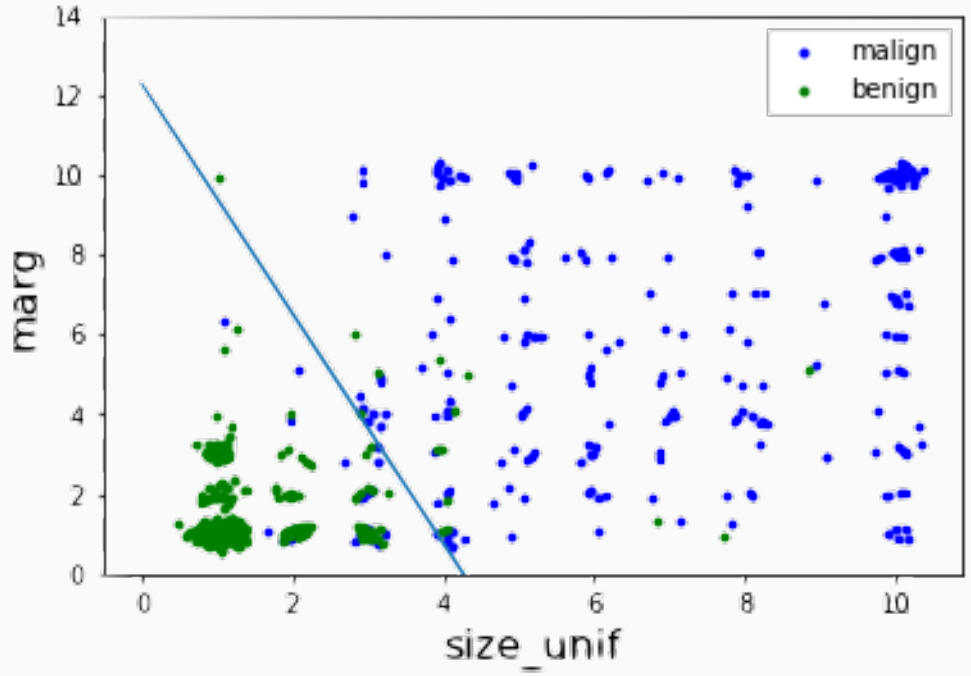
CS-GY 6923: Lecture 5
Logistic Regression + Gradient Descent

NYU Tandon School of Engineering, Prof. Christopher Musco

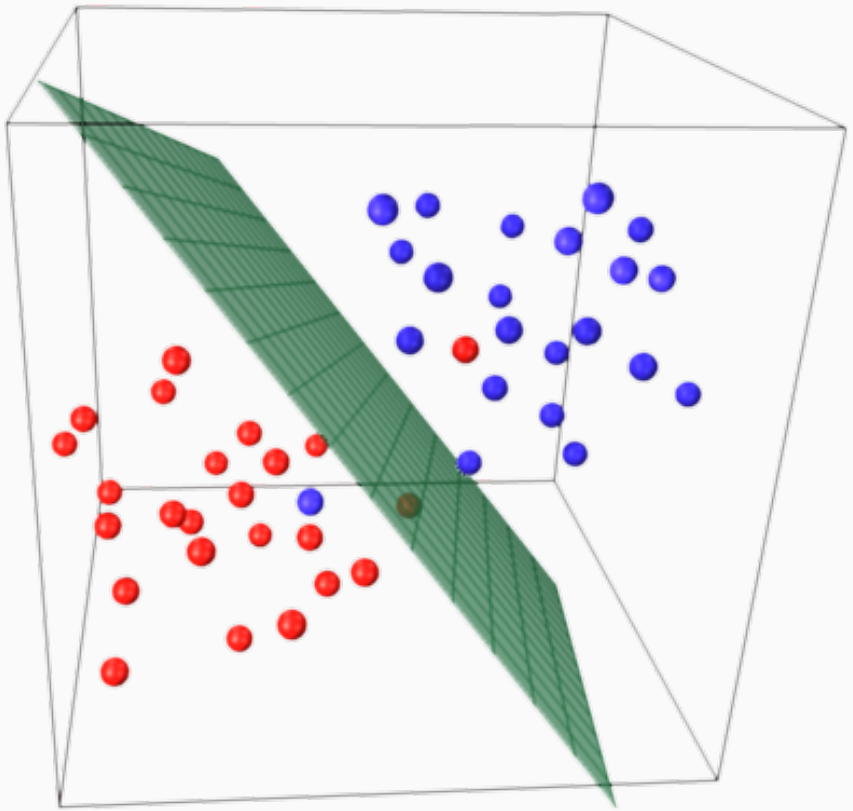
Standard approach for binary classification of real-valued data:

- Find parameter vector $\boldsymbol{\beta}$.
- For input data vector $\mathbf{x}$, predict 0 if $\boldsymbol{\beta}^T\mathbf{x} > \lambda$ and 1 $\boldsymbol{\beta}^T\mathbf{x} \leq \lambda$ for some threshold λ.[1]



[1] Can always assume $\lambda = 0$ if $\mathbf{x}$ has an intercept term.

In higher dimensions, we should think of data as being separated by a hyperplane, which has equation $\boldsymbol{\beta}^T \mathbf{x} = 0$.



Loss minimization approach:

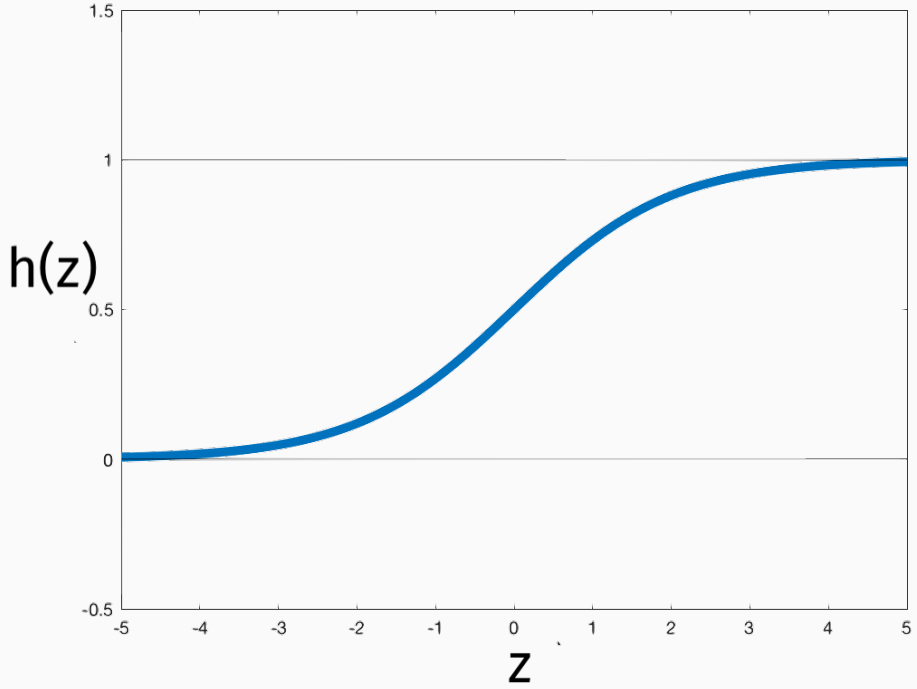
- Given training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$.
- Minimize "Logistic loss" aka "binary cross-entropy loss"

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

- Above $h(z)$ be the logistic/sigmoid function: $h(z) = \frac{1}{1+e^{-z}}$

Predict 1 if $\boldsymbol{\beta}^T \mathbf{x}_i \geq 0$, predict 0 otherwise.

Let $h(z)$ be the logistic/sigmoid function: $h(z) = \frac{1}{1+e^{-z}}$



Can think of this function as mapping $\mathbf{x}^T\boldsymbol{\beta}$ to a probability that the true label is 1. If $\mathbf{x}^T\boldsymbol{\beta} \gg 0$ then the probability is close to 1, if $\mathbf{x}^T\boldsymbol{\beta} \ll 0$ then the probability is close to 0.

GREAT QUESTION

Great question by Azraf from last class: why not minimize

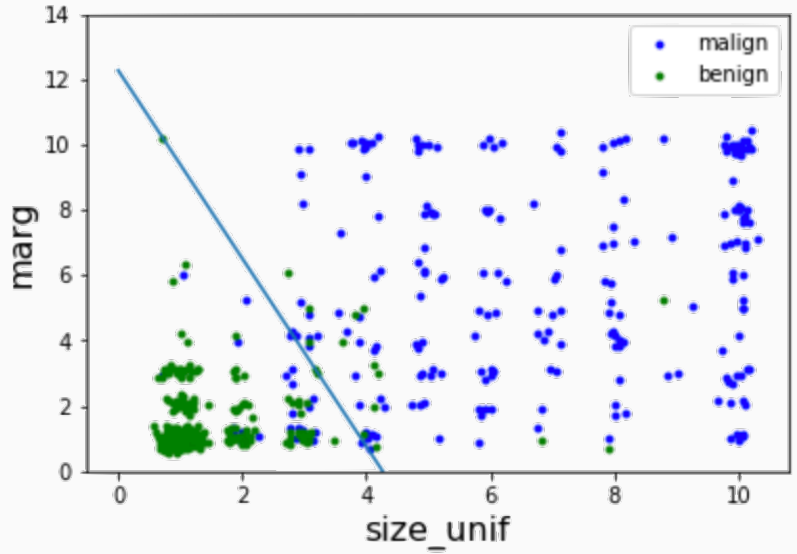
$$L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(y_i - h(\mathbf{x}^T\boldsymbol{\beta}\right)^2?$$

Answer: This is actually a pretty reasonable thing to do. An important issue however is that the loss here is not convex, which makes it hard to find the $\boldsymbol{\beta}$ that minimizes the loss.

Log-loss on the other hand is convex.

LOGISTIC LOSS

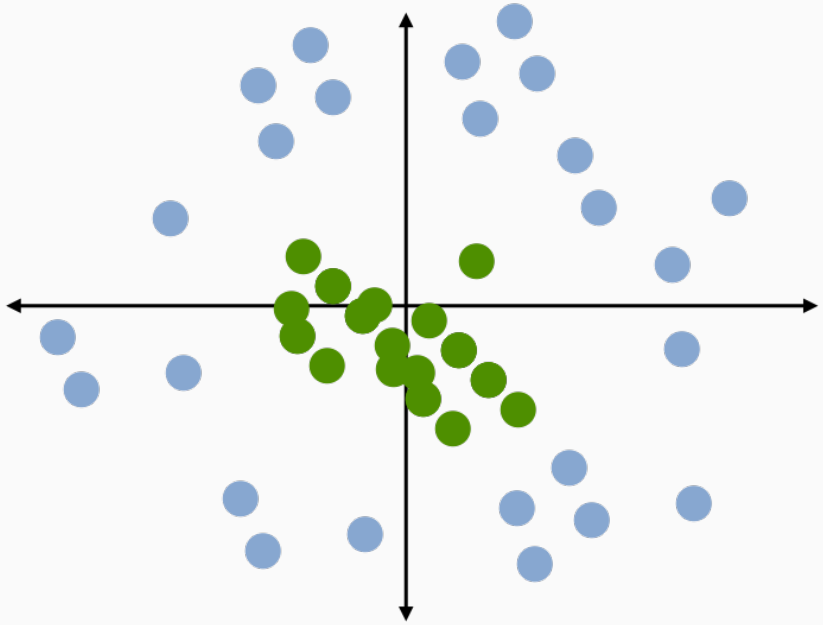
- Convex function in β, can be minimized using gradient descent.
- Works well in practice.
- Good Bayesian motivation.
- Easily combined with non-linear data transformations.



Fit using logistic regression/log loss.

NON-LINEAR TRANSFORMATIONS

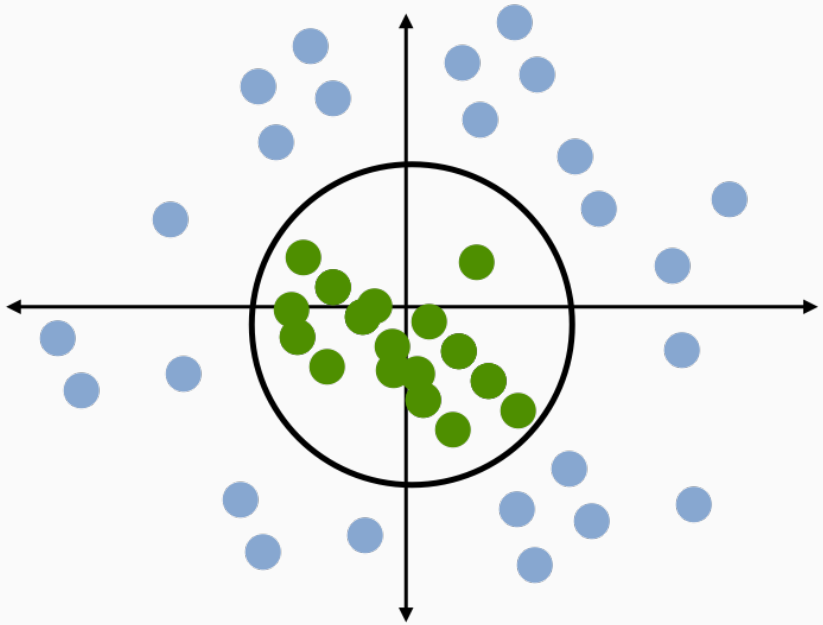
How would we learn a classifier for this data using logisitic regression?



This data is not linearly separable or even approximately linearly separable.

NON-LINEAR TRANSFORMATIONS

Transform each $\mathbf{x} = [x_1, x_2]$ to $\mathbf{x} = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]$

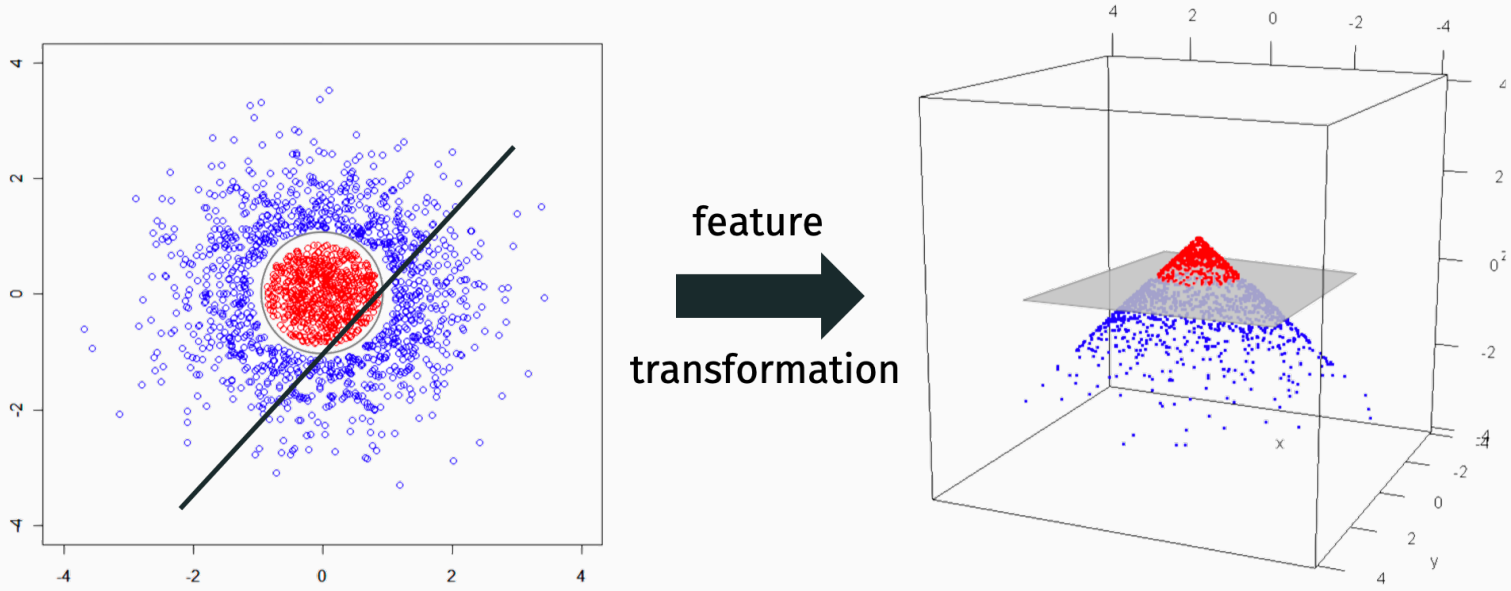


- Predict class 1 if $x_1^2 + x_2^2 < \lambda$.
- Predict class 0 if $x_1^2 + x_2^2 \geq \lambda$.

This is a linear classifier on our transformed data set. Logisitic regression might learn $\boldsymbol{\beta} = [0, 0, 0, 1, 1, 0]$.

NON-LINEAR TRANSFORMATIONS

View as mapping data to a higher dimensional space, where it is linearly separable.



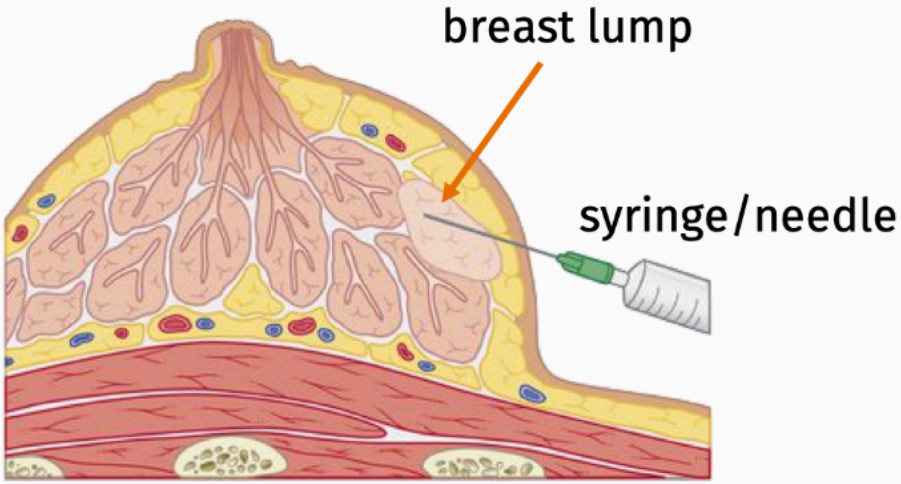
Lots more on this in future lecture!

ERROR IN CLASSIFICATION

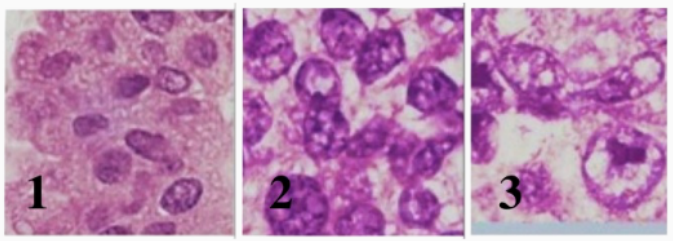
Once we have a classification algorithm, how do we judge its performance?

- **Simplest answer:** Error rate = fraction of data examples misclassified in test set.
- What are some issues with this approach?

Think back to motivating problem of breast cancer detection.



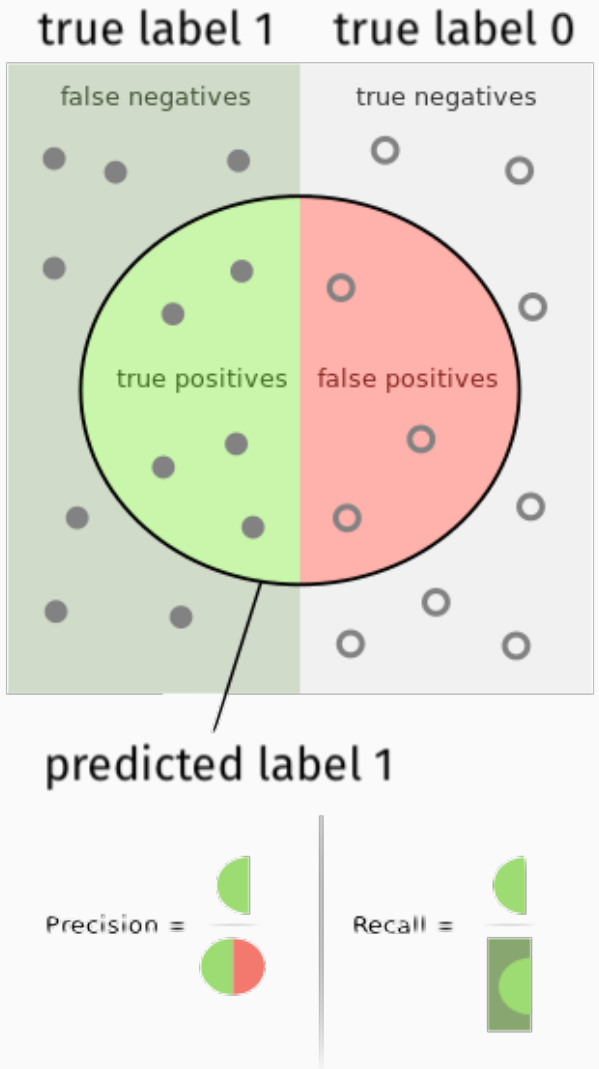
cross section



ERROR IN CLASSIFICATION

- **Precision:** Fraction of positively labeled examples (label 1) which are correct.
- **Recall:** Fraction of true positives that we labeled correctly with label 1.

Question: Which should we optimize for medical diagnosis?



Logistic regression workflow:

- Select $\boldsymbol{\beta}$ via training and compute $h(\boldsymbol{\beta}^T \mathbf{x}_i) = \frac{1}{1+e^{-\langle \mathbf{x}_i, \boldsymbol{\beta} \rangle}}$ for all $\mathbf{x}_i$.
- Predict $y_i = 0$ if $h(\boldsymbol{\beta}^T \mathbf{x}_i) \leq \lambda$, $y_i = 1$ if $h(\boldsymbol{\beta}^T \mathbf{x}_i) > \lambda$.
- Default value of λ is $1/2$. Increasing λ improves precision. Decreasing λ improves recall.

This is very heuristic. There are other methods for handling "class imbalance" or fine tuning precision or recall. Techniques include weighting the loss function to care more about false negatives, or subsampling the larger class.

Possible logistic regression workflow:

- Learn $\boldsymbol{\beta}$ and compute $\boldsymbol{\beta}^T \mathbf{x}_i$ for all $\vec{x}_i$.
- Predict $y_i = 0$ if $\boldsymbol{\beta}^T \mathbf{x}_i \leq \lambda$, $y_i = 1$ if $\boldsymbol{\beta}^T \mathbf{x}_i > \lambda$.
- Default value of λ is 0. Increasing λ improves precision. Decreasing λ improves recall.

This is very heuristic. There are other methods for handling "class imbalance" which can often lead to good overall error, but poor precision or recall. Techniques include weighting the loss function to care more about false negatives, or subsampling the larger class.

MULTI-CLASS

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

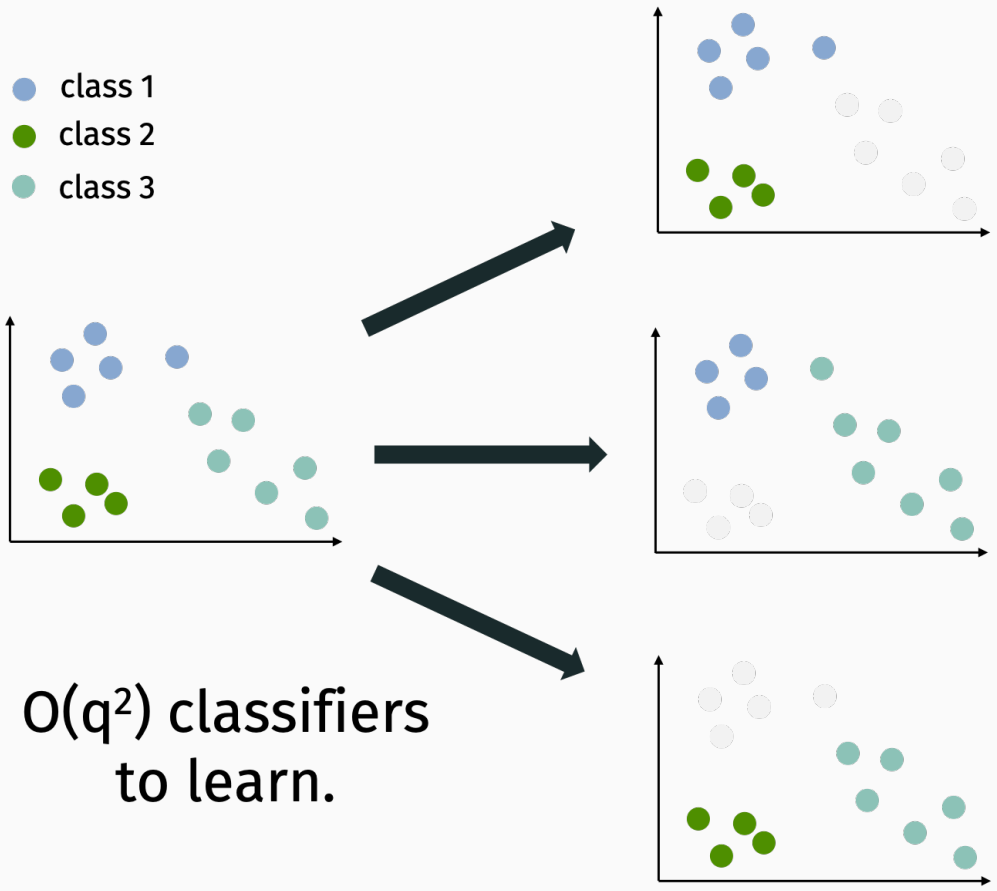
Two common options for reducing multi-class problems to binary problems:

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

ONE VS. REST

- For q classes train q classifiers. Obtain parameters $\beta_1, \ldots, \beta_q$.
- Assign y to class i if $\beta_i^T \mathbf{x}$ is 1. Could be ambiguous!
- **Better:** Assign y to class i with maximum value of $h(\beta_i^T \mathbf{x})$.

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.

But one-vs.-one can be super expensive! E.g when $q = 100$ or $q = 1000$.

More common modern alternative: If we have q classes, train a single model with q parameter vectors $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_q$, and predict class $i = \arg\max_i \boldsymbol{\beta}_i^T \mathbf{x}$.

Same idea as one-vs.-rest, but we treat $[\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_q]$ as a single length qd parameter vector which we to optimize to minimize a single joint loss function. We do not train the parameter vectors separately.

What's a good loss function?

Softmax function:

$$\begin{bmatrix} \boldsymbol{\beta}_1^T \mathbf{x} \\ \vdots \\ \boldsymbol{\beta}_q^T \mathbf{x} \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} e^{\boldsymbol{\beta}_1^T \mathbf{x}} / \sum_{i=1}^{q} e^{\boldsymbol{\beta}_i^T \mathbf{x}} \\ \vdots \\ e^{\boldsymbol{\beta}_q^T \mathbf{x}} / \sum_{i=1}^{q} e^{\boldsymbol{\beta}_i^T \mathbf{x}} \end{bmatrix}$$

Softmax takes in a vector of numbers and converts it to a vector of probabilities:

$$\begin{bmatrix} -10 & 4 & 1 & 0 & -5 \end{bmatrix} \rightarrow \begin{bmatrix} .00 & .94 & .04 & .02 & -.00 \end{bmatrix}$$

Multi-class cross-entropy:

$$L(\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_q) = -\sum_{i:y_i=1} \log \frac{e^{\boldsymbol{\beta}_1^T \mathbf{x}}}{\sum_{j=1}^q e^{\boldsymbol{\beta}_j^T \mathbf{x}}} - \sum_{i:y_i=2} \log \frac{e^{\boldsymbol{\beta}_2^T \mathbf{x}}}{\sum_{j=1}^q e^{\boldsymbol{\beta}_j^T \mathbf{x}}} - \ldots - \sum_{i:y_i=k} \log \frac{e^{\boldsymbol{\beta}_k^T \mathbf{x}}}{\sum_{j=1}^q e^{\boldsymbol{\beta}_j^T \mathbf{x}}}$$

$$= -\sum_{i=1}^n \sum_{\ell=1}^q \mathbb{1}[y_i = \ell] \cdot \log \frac{e^{\boldsymbol{\beta}_\ell^T \mathbf{x}}}{\sum_{j=1}^q e^{\boldsymbol{\beta}_j^T \mathbf{x}}}$$

Binary cross-entropy:

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

$$= -\sum_{i:y_i=1} \log(h(\boldsymbol{\beta}^T \mathbf{x}_i)) - \sum_{i:y_i=0} \log(1 - h(\boldsymbol{\beta}^T \mathbf{x}_i))$$

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.
- Useful to see whole matrix to visualize where errors occur.

OPTIMIZATION

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 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:

```
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:

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

- In contrast to what we saw when minimizing the squared loss for linear regression, there's no simple closed form expression for such a $\boldsymbol{\beta}$!
- 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 $\mathbf{0}$?

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:

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

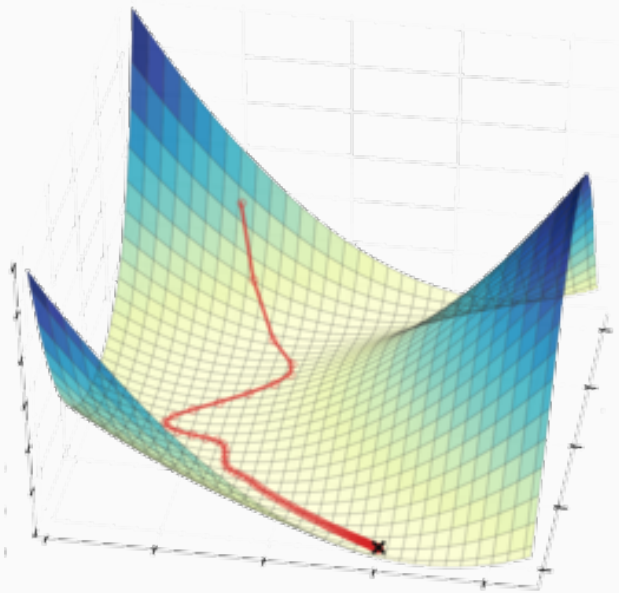
- Start with some $\boldsymbol{\beta}^{(0)}$, and at each step try to change $\boldsymbol{\beta}$ slightly to reduce $L(\boldsymbol{\beta})$.
- Hopefully find an approximate minimizer for $L(\boldsymbol{\beta})$ much more quickly than brute-force search.
- **Concrete goal:** Find $\boldsymbol{\beta}$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

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

  for some small error term ϵ.

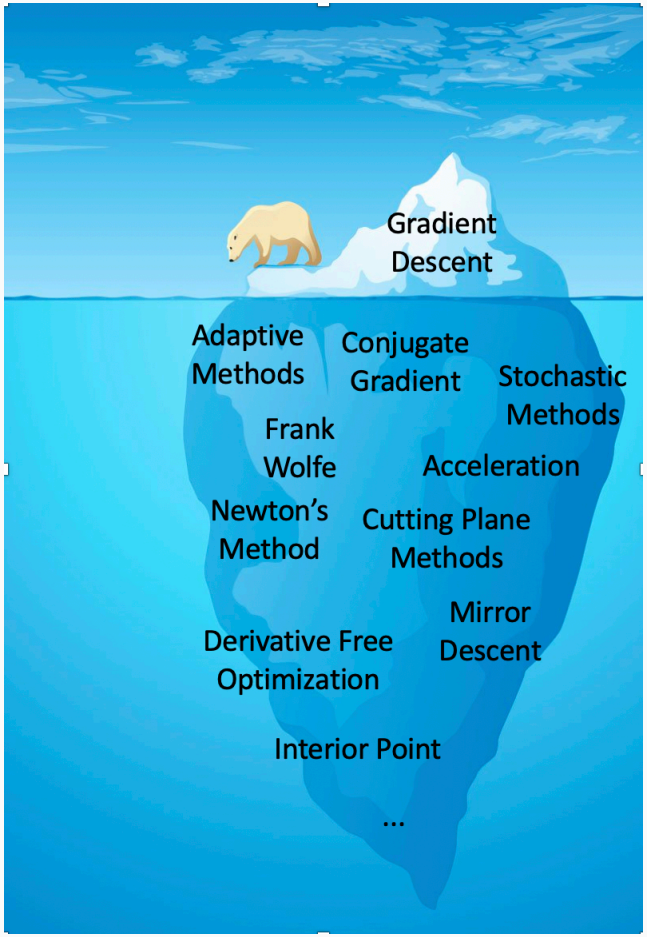
GRADIENT DESCENT

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 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:

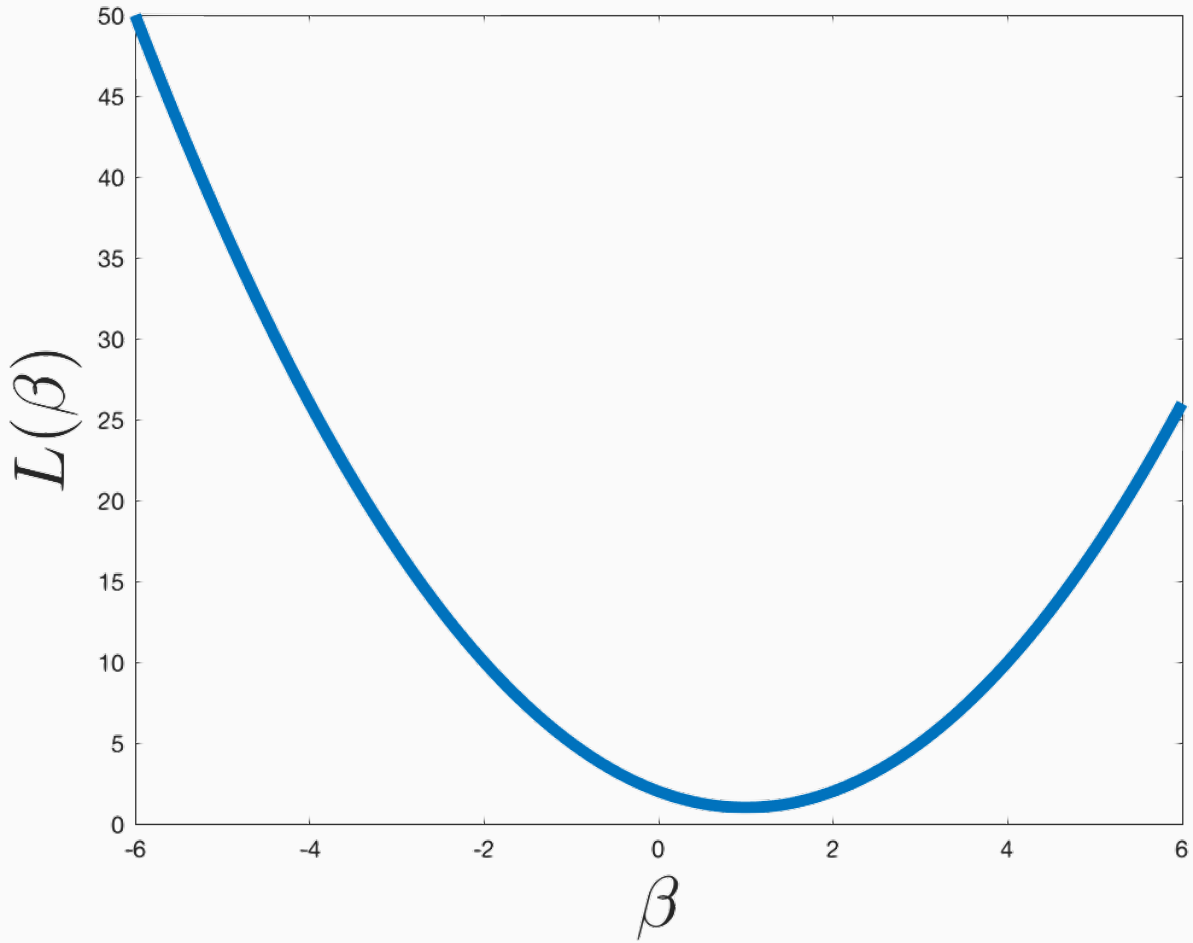
- Choose starting point $\boldsymbol{\beta}^{(0)}$.
- For $i = 1, \ldots, T$:
  - $\boldsymbol{\beta}^{(i+1)} = \boldsymbol{\beta}^{(i)} - \eta \nabla L(\boldsymbol{\beta}^{(i)})$
- Return $\boldsymbol{\beta}^{(t)}$.

η is a step-size parameter. Also called the learning rate.

Why does this method work?

First observation: if we actually reach the minimizer $\boldsymbol{\beta}^*$ then we stop.

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



Mathematical way of thinking about it:

By definition, $L'(\beta) = \lim_{\eta \to 0} \frac{L(\beta+t)-L(\beta)}{t}$. 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 $L(\boldsymbol{\beta} + \mathbf{v}) - L(\boldsymbol{\beta})$?

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

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

:

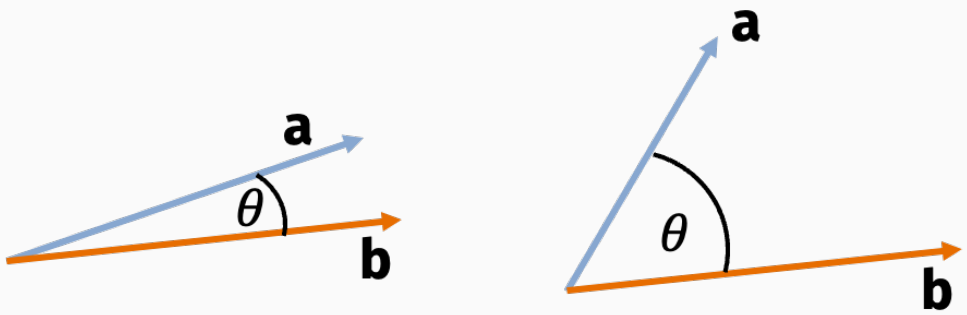
[1] Formally, you might remember that we can define the **directional derivative** of a multivariate function: $D_{\mathbf{v}} L(\boldsymbol{\beta}) = \lim_{t \to 0} \frac{L(\boldsymbol{\beta} + t\mathbf{v}) - L(\boldsymbol{\beta})}{t}$.

Claim (Gradient descent = Steepest descent[2])

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

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

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



[2] 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..

Claim (Gradient descent = Steepest descent)

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

Basic Gradient descent (GD) algorithm:

- 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 $\boldsymbol{\beta}^{(t)}$.

- **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. Get's stuck because the gradient term equals 0

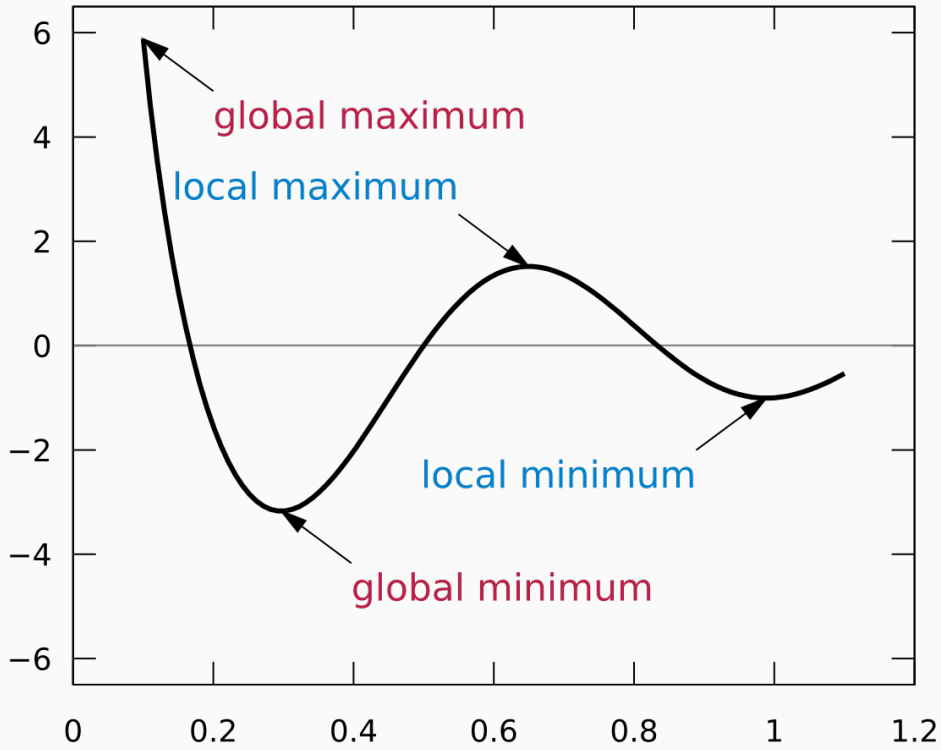
Claim

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 $\tilde{\beta}^*$. I.e. with

$$\|\nabla L(\tilde{\beta}^*)\|_2 = 0.$$

BASIC CLAIM

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



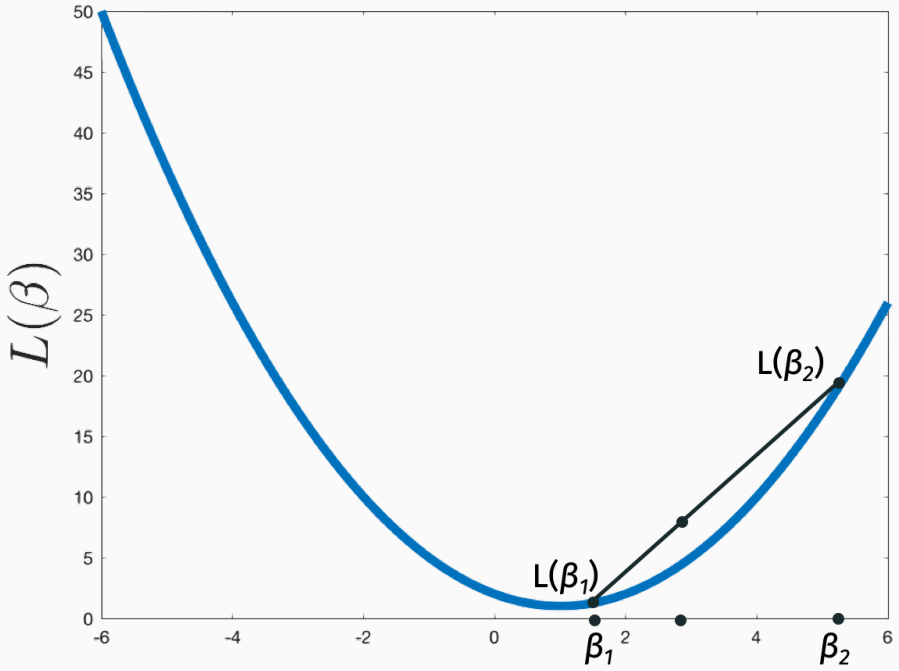
Very unlikely to land precisely on another stationary point and get stuck. Non-minimal stationary points are “unstable”.

We can say something more for a broad class of functions!

Definition (Convex)

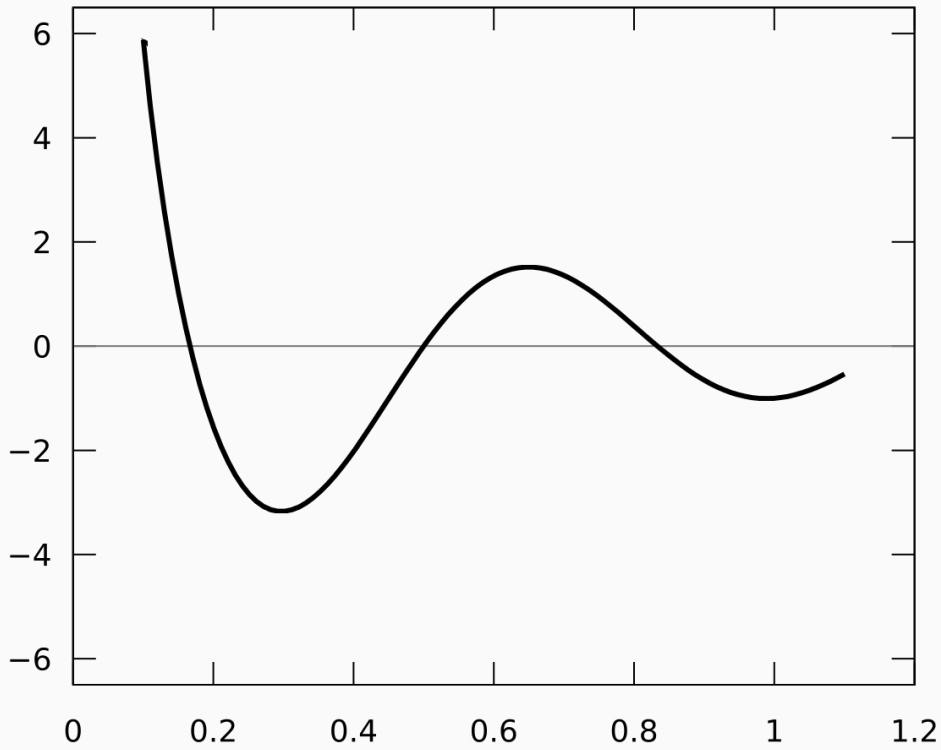
A function L is convex iff for any $\beta_1, \beta_2, \lambda \in [0, 1]$:

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



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.

Claim (Convex Function Minimizers.)

Every stationary point of a differentiable convex function is a global minimum of the function.

The immediate implication is that for any convex loss function, gradient descent converges to $\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}} L(\boldsymbol{\beta})$

Claim (GD Convergence for Convex Functions.)

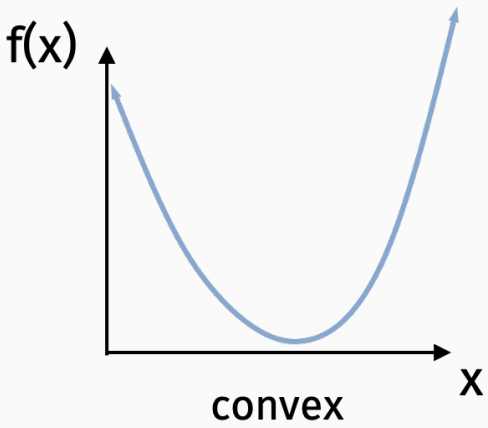
For sufficiently small step-size η, Gradient Descent converges to the global minimum of any convex function L.

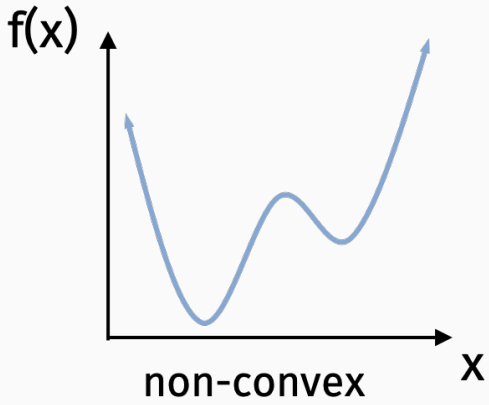
What functions are convex?

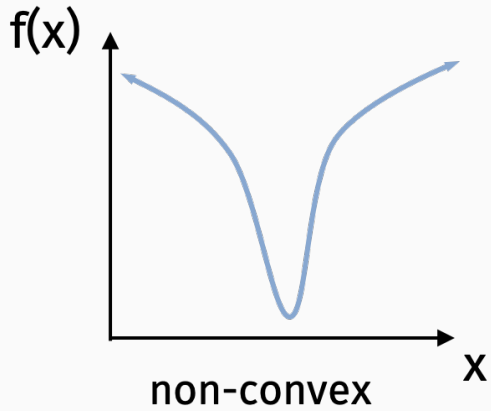
- Least squares loss for linear regression.
- ℓ_1 loss for linear regression.
- Either of these with and ℓ_1 or ℓ_2 regularization penalty.
- Logistic regression! Logistic regression with regularization.
- Many other models in machine leaning.

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.

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 OF LEAST SQUARES REGRESSION LOSS

Prove that $L(\boldsymbol{\beta}) = \|\mathbf{X}\boldsymbol{\beta} - \mathbf{y}\|_2^2$ is convex. For now just consider $\lambda = \frac{1}{2}$ case. The general λ case is similar, but messier.

[2] **Useful identity:** $(a+b)^2 \leq 2(a^2+b^2)$

CONVEXITY OF LEAST SQUARES REGRESSION LOSS

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

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

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!).

Next slides: A canonical gradient descent analysis that every computer scientist should know.

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

Claim (GD Convergence Bound)

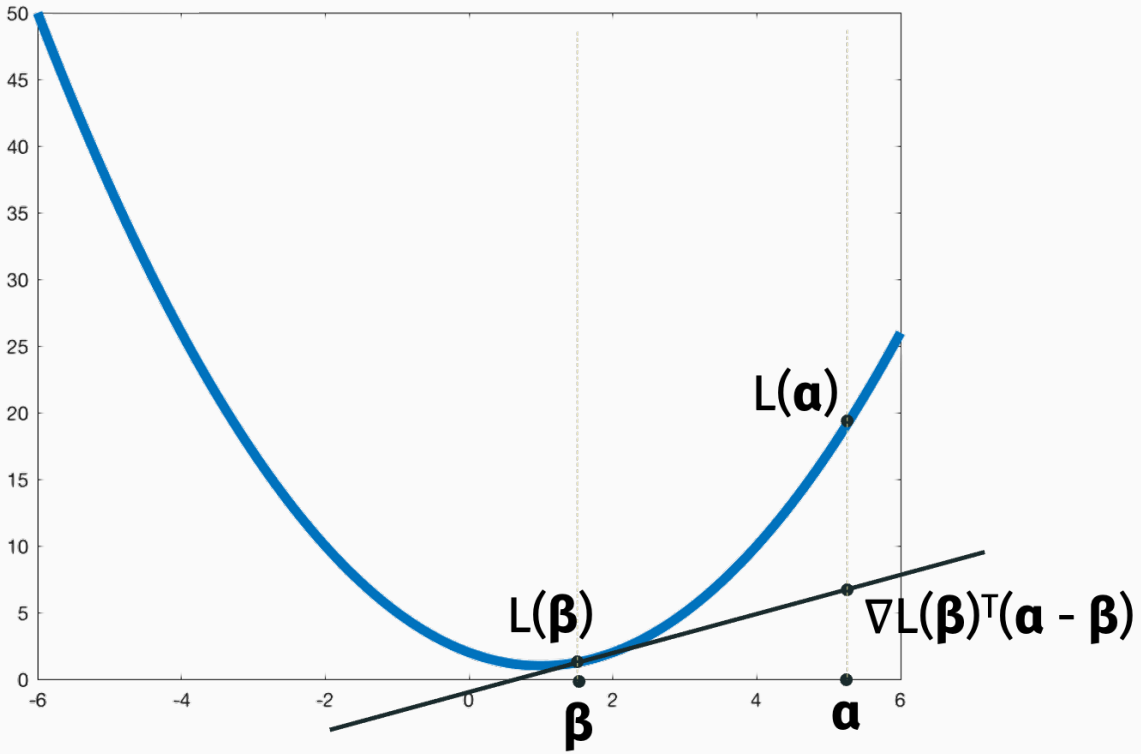
If $T \geq \frac{R^2 G^2}{\epsilon^2}$, *then* $L(\hat{\boldsymbol{\beta}}) \leq L(\boldsymbol{\beta}^*) + \epsilon$.

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 $1/G$.

Definition (Convex)

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

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



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

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

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.

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 $i = 0, \ldots, T$,

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

Claim (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: 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:

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

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

Telescoping sum:

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

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

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:

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

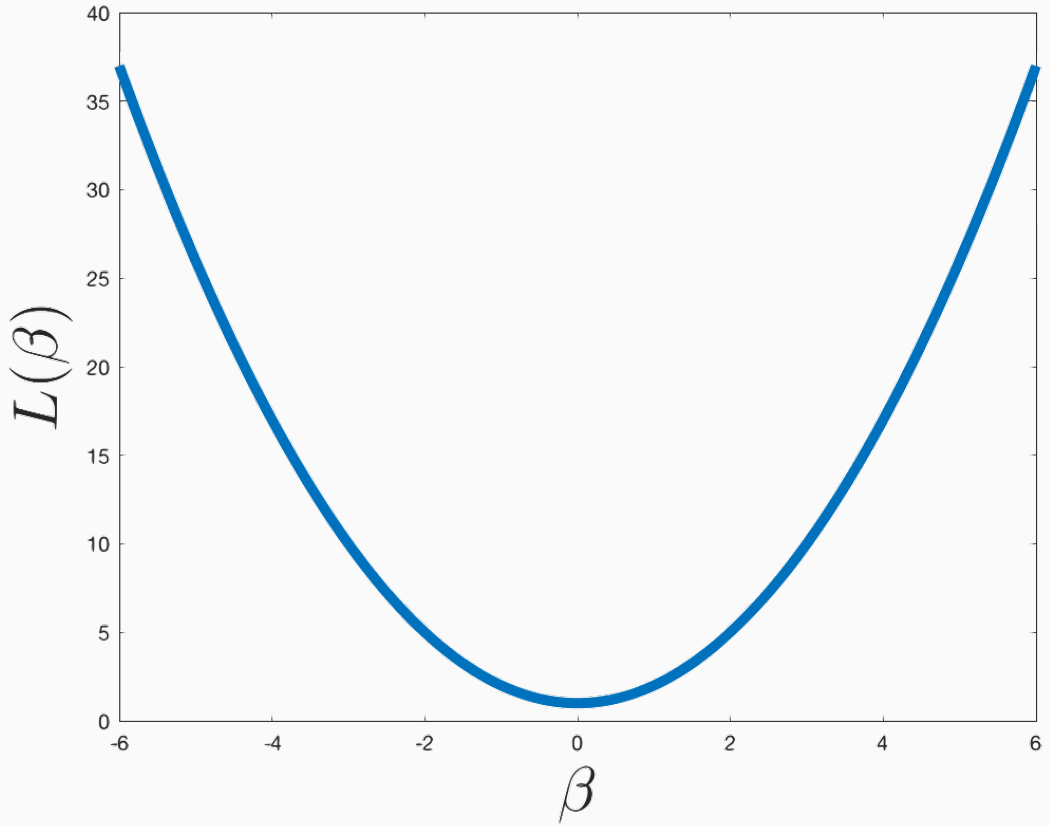
Gradient descent algorithm for minimizing $L(\vec{\beta})$:

- 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 $\vec{\beta}^{(t)}$.

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

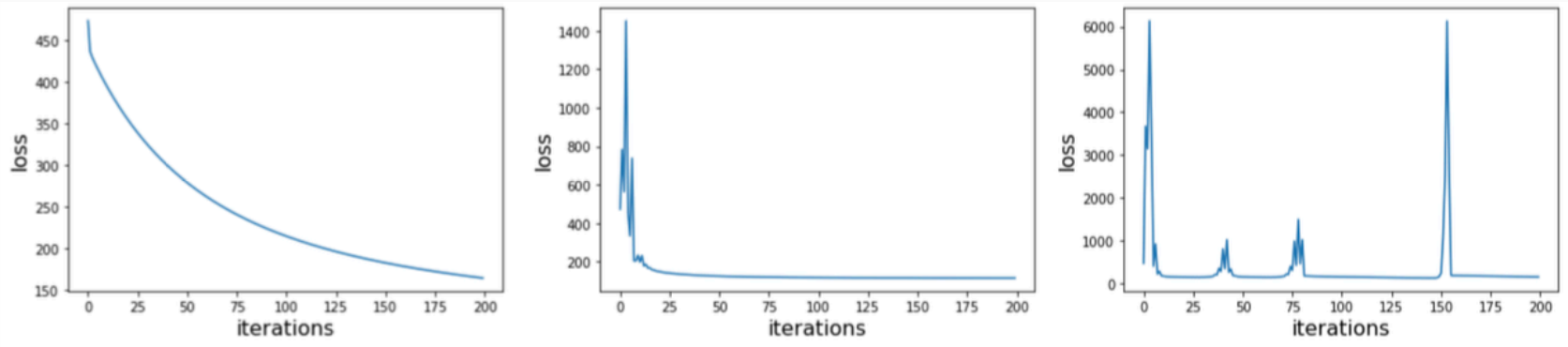
η needs to be chosen sufficiently small for gradient descent to converge, but too small will slow down the algorithm.

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

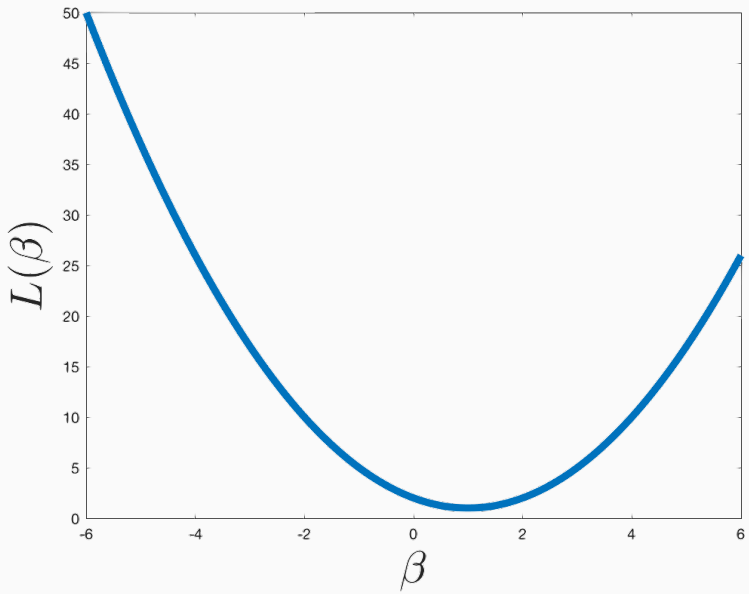
Just as in regularization, search over a grid of possible parameters:

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

Or tune by hand based on the optimization curve.

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

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



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 $\boldsymbol{\beta}$.
- 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 $L(\boldsymbol{\beta}^{(new)}) \leq L(\boldsymbol{\beta}) - c\eta \|\nabla L(\boldsymbol{\beta})\|_2^2$
    - $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}^{(new)}$
    - $\eta \leftarrow \tau^{-1}\eta$
  - Else
    - $\eta \leftarrow \tau\eta$

Always decreases objective value, works very well in practice.