Recitation 1
Statistical Learning Theory
Intro to Gradient Descent

Haoxu Huang

NYU CDS

Jan. 22, 2026

Statistical Learning Theory

Introduction

Section Lead TAs for this course:

- Ansh Sharma, Haoxu Huang, Patrick Shen, Yihui Hong, Karine

Office Hours:

- Ansh: Mon 11:30am - 12:30pm ET
- Haoxu: Tues 11:00am - 12:00pm ET
- Patrick: Wed 4:00pm - 5:00pm ET
- Yihui: Thurs 3:00pm - 4:00pm ET
- Karine: Fri 1:00pm - 2:00pm ET

Location:

- Room 242 in CDS (60 5th Ave, New York, NY 10011)

In data science, we generally need to **Make a Decision** on a problem.
To do this, we need to understand

- The setup of the problem
- The possible actions
- The effect of actions
- The evaluation of the results

How do we translate the problem into the language of DS/modeling?

Statistical Learning Theory

Formalization

The Spaces

$\mathcal{X}$: input space $\quad$ $\mathcal{Y}$: outcome space $\quad$ $\mathcal{A}$: action (decision) space

Prediction Function

A **prediction function** f gets an input $x \in \mathcal{X}$ and produces an action $a \in \mathcal{A}$:

$$f : \mathcal{X} \mapsto \mathcal{A}$$

Loss Function

A **loss function** $\ell(a, y)$ evaluates an action $a \in \mathcal{A}$ in the context of an outcome $y \in \mathcal{Y}$:

$$\ell : \mathcal{A} \times \mathcal{Y} \mapsto \mathbb{R}$$

Statistical Learning Theory

Risk Function

Given a loss function ℓ, how can we evaluate the "average performance" of a prediction function f?

- To do so, we need to first assume that there is a **data generating distribution** $P_{X,Y}$.
- Then the expected loss of f on $P_{X,Y}$ will reflect the notion of "average performance".

Definition

The **risk** of a prediction function $f : \mathcal{X} \mapsto \mathcal{A}$ is

$$R(f) = \mathbb{E}\ell(f(x), y)$$

It is the expected loss of f on a new sample (x, y) drawn from $P_{X,Y}$.

Statistical Learning Theory

Finding 'best' function

Definitions

Hypothesis Class

$\mathcal{F}$ is the family of functions we restrict our model to be. Example: Linear, quadratic, decision tree, two layer neural-net...

Bayes optimal predictor within $\mathcal{F}$

$f^*_{\mathcal{F}}$ is 'best' function one can obtain within $\mathcal{F}$.

Sample-optimal predictor

$\hat{f}_n$ is the 'best' function one can obtain using the data given.

Learned predictor

$\tilde{f}_n$ is the function actually obtained using the data given.

Definition

A **Bayes prediction function** $f^* : \mathcal{X} \mapsto \mathcal{Y}$ is a function that achieves the minimal risk among all possible functions:

$$f^* \in \arg\min_f R(f),$$

where the minimum is taken from all functions that maps from $\mathcal{X}$ to $\mathcal{A}$.

The risk of a Bayes function is called **Bayes risk**.

Statistical Learning Theory

Finding the Bayes Optimal Classifier

Goal:

In a multi-class classification problem with class $Y = \in \{0, 1, ..., n\}$ and features X, we want to find a classifier $h(X) = \hat{y}$ that minimizes the expected loss (risk).

$$\begin{aligned}
\mathbb{E}_Y(\mathbb{I}_{\{y \neq \hat{y}\}}) &= \mathbb{E}_X \mathbb{E}_{Y|X}(\mathbb{I}_{\{y \neq \hat{y}\}} | X = x) \quad \text{(Law of total expectation)} \\
&= \mathbb{E}[\Pr(Y = 1 | X = x) \mathbb{I}_{\{\hat{y} = 2,3,...,n\}}] \\
&+ \mathbb{E}[\Pr(Y = 2 | X = x) \mathbb{I}_{\{\hat{y} = 1,3,...,n\}}] + ... \\
&+ \mathbb{E}[\Pr(Y = n | X = x) \mathbb{I}_{\{\hat{y} = 1,2,3,...,n-1\}}]
\end{aligned}$$

Note:

This is minimized by simutaneously minimizing all terms of expectation using the classifier $h(x) = k$ with $\text{argmax}_k \Pr(Y = k | X = x)$ for each observation x.

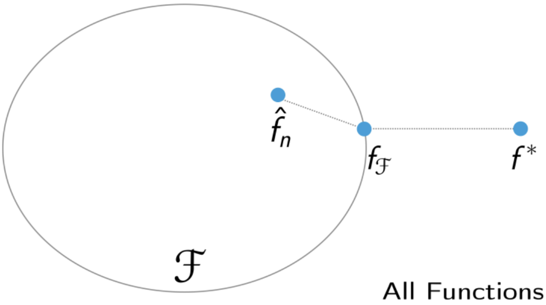
Statistical Learning Theory

Finding the Bayes Optimal Classifier

- $P(Y = 1|x) + P(Y = 2|x) + ... + P(Y = n|x) = 1$ by complement rule of probability.
- The probability of error is $P(Error|x) = 1 - P(Correct|x)$
- To minimize the error, we need to maxmize the probability of being correct.
- Consequently, for any given observation x, the strategy that minimizes risk is simply: Pick the class k that has the highest posterior probability (i.e. $\text{argmax}_k \text{Pr}(Y = k|X = x)$)

Statistical Learning Theory

Error Decomposition



Approximation Error

- Caused by the choice of family of functions or capacity of the model. ($\epsilon_{approx} = R(f_{\mathcal{F}}) - R(f^*)$)
- Expand the capacity of the model.

Estimation Error

- Caused by finite number of data. ($\epsilon_{est} = (R(\hat{f}_n) - R(f_{\mathcal{F}}))$)
- Obtain more data/add regularizer

Optimization Error

- Caused by not able to find the best parameters. ($\epsilon_{opt} = R(\tilde{f}) - R(\hat{f}_n)$)
- Try different optimization algorithms, learning rates, etc.

Decomposition of "Excess Risk" (how much worse our final model is compared to the perfect Bayes optimal)

$$
\begin{aligned}
R(\tilde{f}_n) - R(f^*) &= \epsilon_{\text{approx}} + \epsilon_{\text{est}} + \epsilon_{\text{opt}} \\
&= (R(f_{\mathcal{F}}) - R(f^*)) \\
&\quad + (R(\hat{f}_n) - R(f_{\mathcal{F}})) \\
&\quad + (R(\tilde{f}_n) - R(\hat{f}_n))
\end{aligned}
$$

Statistical Learning Theory

Gradient Descent

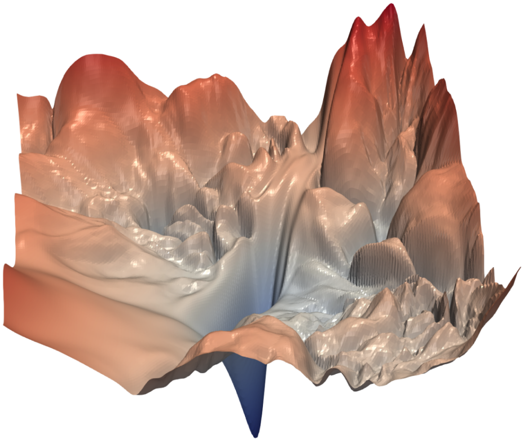
Motivation:
Our goal is the find $\hat{f}_n$, the best possible model from given data
Naive approach: Take gradient of loss function, solve for parameters that gives you 0.

- Computationally intractable
- Impossible to compute due to complex function structure

The optimal parameters for LR: $\hat{\beta} = (X^\top X)^{-1} X^\top Y$
When X's dimension reaches the millions, the inverse is essentially intractable.
But we do not need $\hat{f}_n$, a close $\tilde{f}_n$ is good enough for decision making.
Therefore, instead of solving for the best parameters, we just need to approximate it well enough.

Statistical Learning Theory

Loss Landscape for Classical Neural Networks



Hao Li, et.al. Visualizing the Loss Landscape of Neural Nets. NeurIPS 2018.

Idea:

- Given any starting parameters, the gradient indicates the direction of local maximal change.
- If we obtain new parameters by moving old parameter along its gradient, the new ones will give smaller loss (if we are careful).
- We can repeat this procedure until we are happy with the result.

Statistical Learning Theory

Directional Gradient Recap

- We say a function $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable at $x \in \mathbb{R}^n$ if

$$\lim_{v \to 0} \frac{f(x+v) - f(x) - g^T v}{\|v\|_2} = 0,$$

  for some gradient vector $g \in \mathbb{R}^n$ and displacement vector $v \in \mathbb{R}^n$
- If it exists, this g is unique and is called the gradient of f at x with notation

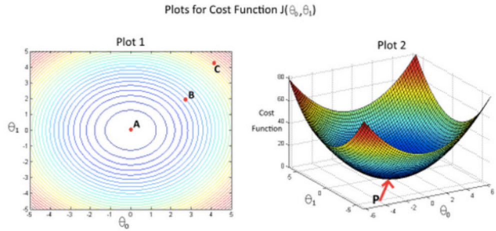
$$g = \nabla f(x)$$

- It can be shown that

$$\nabla f(x) = \begin{pmatrix} \partial_{x_1} f(x) \\ \vdots \\ \partial_{x_n} f(x) \end{pmatrix}$$

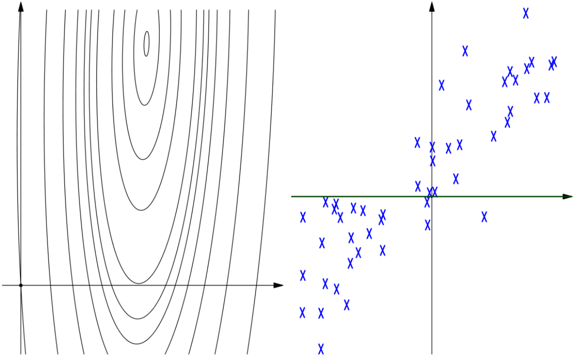
Imagine we are solving a simple linear regression problem: $y = \theta_0 + \theta_1 x$ with loss function:

$$J(\theta_0, \theta_1) = \sum_n (y_i - (\theta_0 + \theta_1 x_i))^2$$



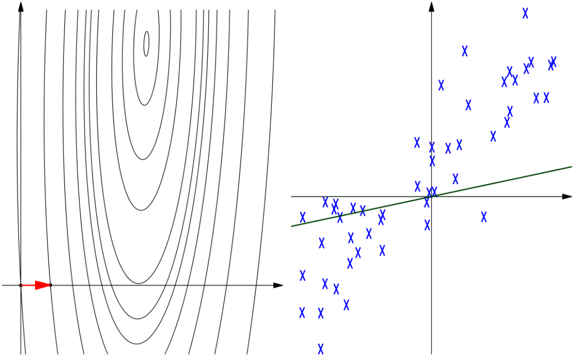
Statistical Learning Theory

Negative Gradient Steps



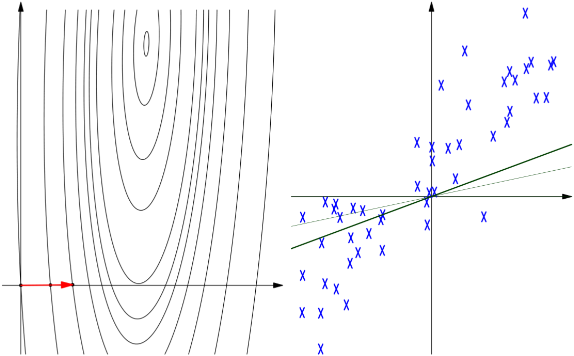
Statistical Learning Theory

Negative Gradient Steps



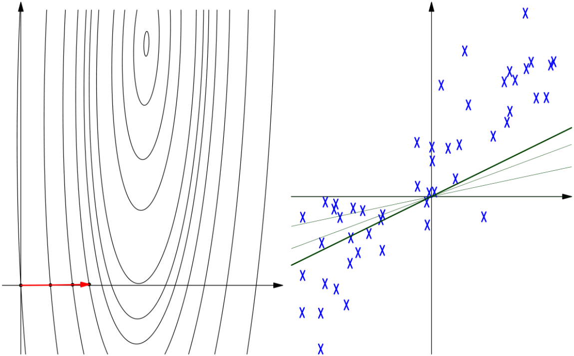
Statistical Learning Theory

Negative Gradient Steps



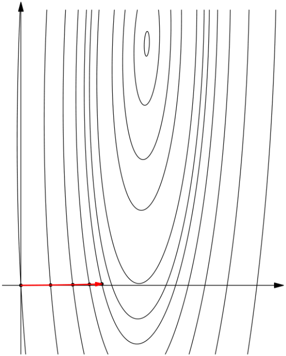
Statistical Learning Theory

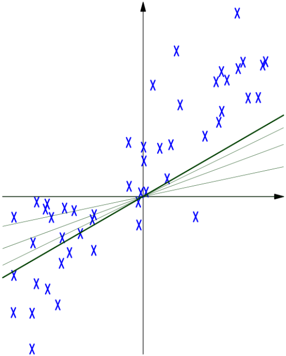
Negative Gradient Steps



Statistical Learning Theory

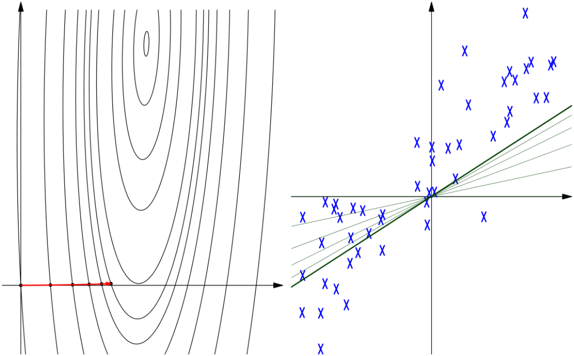
Negative Gradient Steps





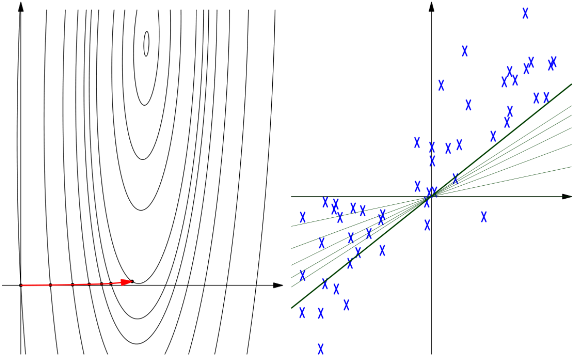
Statistical Learning Theory

Negative Gradient Steps



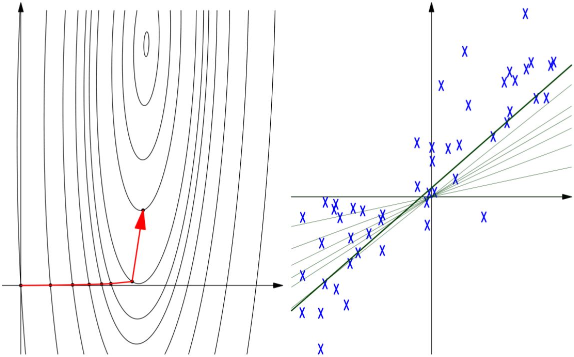
Statistical Learning Theory

Negative Gradient Steps



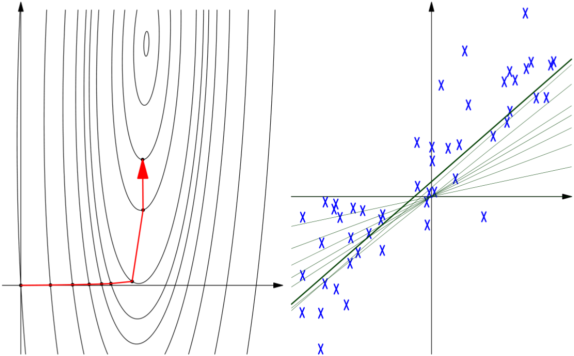
Statistical Learning Theory

Negative Gradient Steps



Statistical Learning Theory

Negative Gradient Steps



Gradient Descent

Goal: find $\theta^* = \arg\min_\theta J(\theta)$

$\theta_0 :=$ [initial condition] (can be randomly chosen)
$i := 0$
while not [termination condition] **do**
  compute $\nabla J(\theta_i)$
  $\alpha :=$ [choose learning rate at iteration i]
  $\theta_{i+1} := \theta_i - \alpha \nabla J(\theta_i)$
  $i := i + 1$
end while
return θ_i

explain how to take gradient/calculus etc. It will be present in next class Objective: only calculating and using gradient information $\nabla_\theta L(\theta)$ on deciding where to move on loss surface.

Pros: Computational Efficiency, Low Memory Footprint, More Scalable

Cons: Slow Convergence, Struggle with Curvature, Can stuck at Saddle Points

Example Algorithms: SGD, SGD + momentum, Adam, AdamW, RMSProp

Gradient Descent

Second-order Gradient Descent

Objective: calculating and using both gradient information $\nabla_\theta L(\theta)$ and curvature information $\nabla^2_\theta L(\theta)$ (i.e. Hessian approx) on deciding where to move on loss surface.
(e.g. $\theta_{i+1} = \theta_i - \alpha(\nabla^2 L(\theta_i))^{-1} \nabla L(\theta_i)$)

Pros: Fast Convergence, Escapes Saddle Points, Curvature awareness

Cons: Computational Expense (inverting the Hessian matrix is $O(n^3)$), Memory Explosion (storing full hessian matrix)

Example Algorithms: Newton's method, Newton Schulz, Quasi-Newton (L-BFGS), Muon

Things to review

Calculus

- Gradients, taking (partial) derivatives

Linear Algebra

- Matrix computation, matrix derivatives
- Example: compute $\frac{\partial x^T A x}{\partial x}$, where A is a matrix and x is a vector

Suggested textbooks on Course Website
(`https://nyu-dsga-1003.github.io/sp26/`)