CSCI567 Machine Learning (Fall 2024)

Prof. Dani Yogatama

University of Southern California

September 6, 2024

Outline

1. Linear regression
2. Linear regression with nonlinear basis
3. Overfitting and preventing overfitting
4. A Detour of Numerical Optimization Methods

Regression

Predicting a continuous outcome variable using past observations

- Predicting future temperature (last lecture)
- Predicting the amount of rainfall
- Predicting the demand of a product
- Predicting the sale price of a house
- ...

Regression

Predicting a continuous outcome variable using past observations

- Predicting future temperature (last lecture)
- Predicting the amount of rainfall
- Predicting the demand of a product
- Predicting the sale price of a house
- ...

Key difference from classification

- continuous vs discrete
- measure *prediction errors* differently.
- lead to quite different learning algorithms.

Regression

Predicting a continuous outcome variable using past observations

- Predicting future temperature (last lecture)
- Predicting the amount of rainfall
- Predicting the demand of a product
- Predicting the sale price of a house
- ...

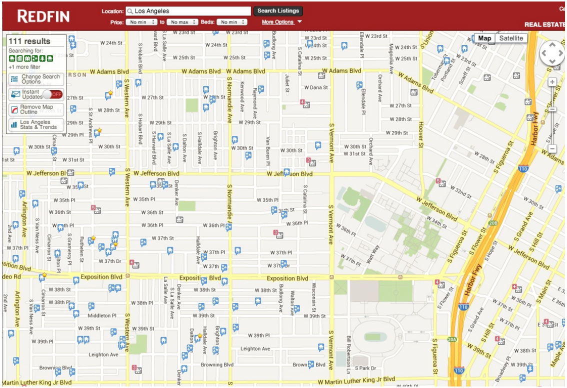
Key difference from classification

- continuous vs discrete
- measure *prediction errors* differently.
- lead to quite different learning algorithms.

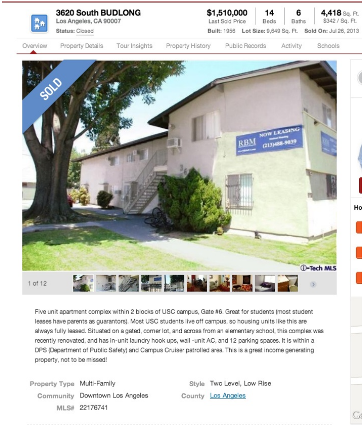
Linear Regression: regression with linear models

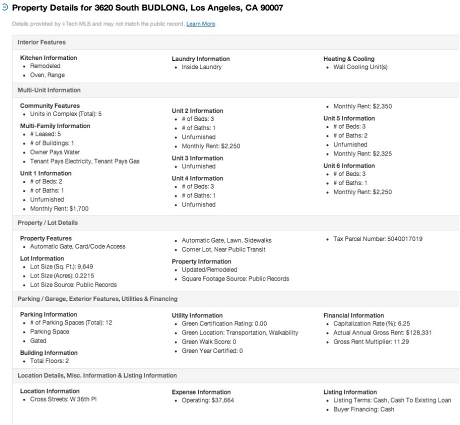
Ex: Predicting the sale price of a house

Retrieve historical sales records (training data)



Features used to predict





Property Details for 3620 South BUDLONG, Los Angeles, CA 90007

Details provided by i-Tech MLS and may not match the public record. Learn More.

Interior Features

Kitchen Information
- Remodeled
- Oven, Range

Laundry Information
- Inside Laundry

Heating & Cooling
- Wall Cooling Unit(s)

Multi-Unit Information

Community Features
- Units in Complex (Total): 5

Multi-Family Information
- # Leased: 5
- # of Buildings: 1
- Owner Pays Water
- Tenant Pays Electricity, Tenant Pays Gas

Unit 1 Information
- # of Beds: 2
- # of Baths: 1
- Unfurnished
- Monthly Rent: $1,700

Unit 2 Information
- # of Beds: 3
- # of Baths: 1
- Unfurnished
- Monthly Rent: $2,250

Unit 3 Information
- Unfurnished

Unit 4 Information
- # of Beds: 3
- # of Baths: 1
- Unfurnished
- Monthly Rent: $2,350

Unit 5 Information
- # of Beds: 3
- # of Baths: 2
- Unfurnished
- Monthly Rent: $2,325

Unit 6 Information
- # of Beds: 3
- # of Baths: 1
- Monthly Rent: $2,250

Property / Lot Details

Property Features
- Automatic Gate, Card/Code Access
- Automatic Gate, Lawn, Sidewalks
- Corner Lot, Near Public Transit

Lot Information
- Lot Size (Sq. Ft.): 9,649
- Lot Size (Acres): 0.2215
- Lot Size Source: Public Records

Property Information
- Updated/Remodeled
- Square Footage Source: Public Records
- Tax Parcel Number: 5040017019

Parking / Garage, Exterior Features, Utilities & Financing

Parking Information
- # of Parking Spaces (Total): 12
- Parking Space
- Gated

Building Information
- Total Floors: 2

Utility Information
- Green Certification Rating: 0.00
- Green Location: Transportation, Walkability
- Green Walk Score: 0
- Green Year Certified: 0

Financial Information
- Capitalization Rate (%): 6.25
- Actual Annual Gross Rent: $128,331
- Gross Rent Multiplier: 11.29

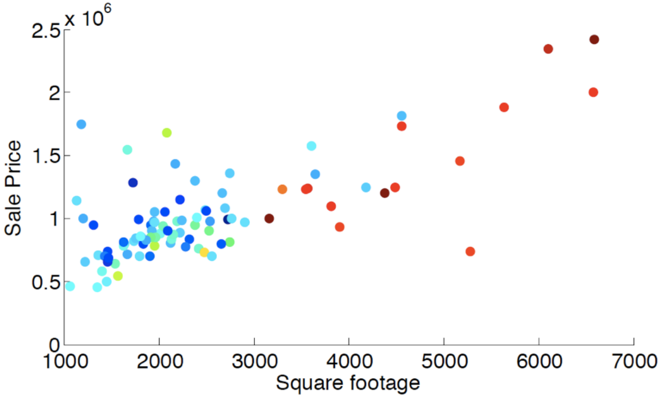
Location Details, Misc. Information & Listing Information

Location Information
- Cross Streets: W 36th Pl

Expense Information
- Operating: $37,664

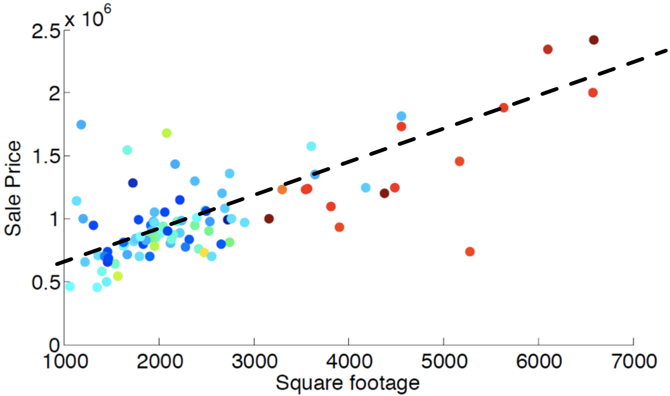
Listing Information
- Listing Terms: Cash, Cash To Existing Loan
- Buyer Financing: Cash

Correlation between square footage and sale price



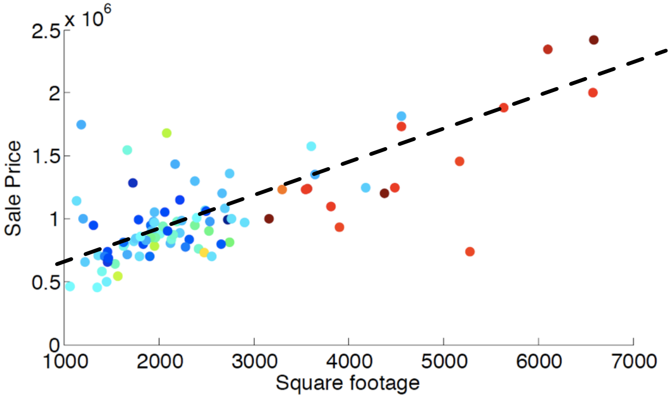
Possibly linear relationship

Sale price ≈ **price_per_sqft** × square_footage + **fixed_expense**



Possibly linear relationship

Sale price ≈ **price_per_sqft** × square_footage + **fixed_expense**
(*slope*) (*intercept*)



How to learn the unknown parameters?

How to measure error for one prediction?

- The classification error (0-1 loss, i.e. *right* or *wrong*) is *inappropriate* for continuous outcomes.

How to learn the unknown parameters?

How to measure error for one prediction?

- The classification error (0-1 loss, i.e. *right* or *wrong*) is *inappropriate* for continuous outcomes.
- We can look at
  - *absolute* error: | prediction - sale price |

How to learn the unknown parameters?

How to measure error for one prediction?

- The classification error (0-1 loss, i.e. *right* or *wrong*) is *inappropriate* for continuous outcomes.
- We can look at
  - *absolute* error: | prediction - sale price |
  - or *squared* error: (prediction - sale price)^2 (**most common**)

How to learn the unknown parameters?

How to measure error for one prediction?

- The classification error (0-1 loss, i.e. *right* or *wrong*) is *inappropriate* for continuous outcomes.
- We can look at
  - *absolute* error: | prediction - sale price |
  - or *squared* error: (prediction - sale price)^2 (**most common**)

Goal: pick the model (unknown parameters) that minimizes the average/total prediction error,

How to learn the unknown parameters?

How to measure error for one prediction?

- The classification error (0-1 loss, i.e. *right* or *wrong*) is *inappropriate* for continuous outcomes.
- We can look at
  - *absolute* error: | prediction - sale price |
  - or *squared* error: (prediction - sale price)^2 (**most common**)

Goal: pick the model (unknown parameters) that minimizes the average/total prediction error, but *on what set*?

How to learn the unknown parameters?

How to measure error for one prediction?

- The classification error (0-1 loss, i.e. *right* or *wrong*) is *inappropriate* for continuous outcomes.
- We can look at
  - *absolute* error: | prediction - sale price |
  - or *squared* error: (prediction - sale price)^2 (**most common**)

Goal: pick the model (unknown parameters) that minimizes the average/total prediction error, but *on what set*?

- test set, ideal but we *cannot use test set while training*

How to learn the unknown parameters?

How to measure error for one prediction?

- The classification error (0-1 loss, i.e. *right* or *wrong*) is *inappropriate* for continuous outcomes.
- We can look at
  - *absolute* error: | prediction - sale price |
  - or *squared* error: (prediction - sale price)^2 (**most common**)

Goal: pick the model (unknown parameters) that minimizes the average/total prediction error, but *on what set*?

- test set, ideal but we *cannot use test set while training*
- training set ✓

Example

Predicted price = **price_per_sqft** × square_footage + **fixed_expense**

one model: price_per_sqft = 0.3K, fixed_expense = 210K

sqft	sale price (K)	prediction (K)	squared error
2000	810	810	0
2100	907	840	67^2
1100	312	540	228^2
5500	2,600	1,860	740^2
$\cdots$	$\cdots$	$\cdots$	$\cdots$
Total			$0 + 67^2 + 228^2 + 740^2 + \cdots$

Adjust price_per_sqft and fixed_expense such that the total squared error is minimized.

Formal setup for linear regression

Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (features, covariates, context, predictors, etc)

Output: $y \in \mathbb{R}$ (responses, targets, outcomes, etc)

Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

Formal setup for linear regression

Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (features, covariates, context, predictors, etc)

Output: $y \in \mathbb{R}$ (responses, targets, outcomes, etc)

Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

Linear model: $f : \mathbb{R}^{\mathsf{D}} \rightarrow \mathbb{R}$, with $f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{D} w_d x_d$

Formal setup for linear regression

Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (features, covariates, context, predictors, etc)

Output: $y \in \mathbb{R}$ (responses, targets, outcomes, etc)

Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

Linear model: $f : \mathbb{R}^{\mathsf{D}} \rightarrow \mathbb{R}$, with $f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{D} w_d x_d = w_0 + \boldsymbol{w}^{\mathbf{T}} \boldsymbol{x}$ (superscript ^{T} stands for transpose),

Formal setup for linear regression

Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (features, covariates, context, predictors, etc)

Output: $y \in \mathbb{R}$ (responses, targets, outcomes, etc)

Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

Linear model: $f : \mathbb{R}^{\mathsf{D}} \rightarrow \mathbb{R}$, with $f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{D} w_d x_d = w_0 + \boldsymbol{w}^{\mathbf{T}}\boldsymbol{x}$
(superscript ^{T} stands for transpose), i.e. a *hyper-plane* parametrized by

- $\boldsymbol{w} = [w_1 \ w_2 \ \cdots \ w_{\mathsf{D}}]^{\mathrm{T}}$ (weights, weight vector, parameter vector, etc)
- bias w_0

Formal setup for linear regression

Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (features, covariates, context, predictors, etc)

Output: $y \in \mathbb{R}$ (responses, targets, outcomes, etc)

Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

Linear model: $f : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}$, with $f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{D} w_d x_d = w_0 + \boldsymbol{w}^{\mathbf{T}}\boldsymbol{x}$ (superscript ^{T} stands for transpose), i.e. a *hyper-plane* parametrized by

- $\boldsymbol{w} = [w_1 \ w_2 \ \cdots \ w_{\mathsf{D}}]^{\mathrm{T}}$ (weights, weight vector, parameter vector, etc)
- bias w_0

NOTE: for notation convenience, very often we

- append 1 to each x as the first feature: $\tilde{\boldsymbol{x}} = [1 \ x_1 \ x_2 \ \ldots \ x_{\mathsf{D}}]^{\mathrm{T}}$

Formal setup for linear regression

Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (features, covariates, context, predictors, etc)

Output: $y \in \mathbb{R}$ (responses, targets, outcomes, etc)

Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

Linear model: $f : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}$, with $f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{D} w_d x_d = w_0 + \boldsymbol{w}^{\mathbf{T}}\boldsymbol{x}$ (superscript ^{T} stands for transpose), i.e. a *hyper-plane* parametrized by

- $\boldsymbol{w} = [w_1 \ w_2 \ \cdots \ w_{\mathsf{D}}]^{\mathrm{T}}$ (weights, weight vector, parameter vector, etc)
- bias w_0

NOTE: for notation convenience, very often we

- append 1 to each x as the first feature: $\tilde{\boldsymbol{x}} = [1 \ x_1 \ x_2 \ \ldots \ x_{\mathsf{D}}]^{\mathrm{T}}$
- let $\tilde{\boldsymbol{w}} = [w_0 \ w_1 \ w_2 \ \cdots \ w_{\mathsf{D}}]^{\mathrm{T}}$, a concise representation of all $D+1$ parameters

Formal setup for linear regression

Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (features, covariates, context, predictors, etc)

Output: $y \in \mathbb{R}$ (responses, targets, outcomes, etc)

Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

Linear model: $f : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}$, with $f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{D} w_d x_d = w_0 + \boldsymbol{w}^{\mathbf{T}}\boldsymbol{x}$ (superscript ^{T} stands for transpose), i.e. a *hyper-plane* parametrized by

- $\boldsymbol{w} = [w_1 \; w_2 \; \cdots \; w_{\mathsf{D}}]^{\mathrm{T}}$ (weights, weight vector, parameter vector, etc)
- bias w_0

NOTE: for notation convenience, very often we

- append 1 to each x as the first feature: $\tilde{\boldsymbol{x}} = [1 \; x_1 \; x_2 \; \ldots \; x_{\mathsf{D}}]^{\mathrm{T}}$
- let $\tilde{\boldsymbol{w}} = [w_0 \; w_1 \; w_2 \; \cdots \; w_{\mathsf{D}}]^{\mathrm{T}}$, a concise representation of all $D+1$ parameters
- the model becomes simply $f(\boldsymbol{x}) = \tilde{\boldsymbol{w}}^{\mathbf{T}}\tilde{\boldsymbol{x}}$

Formal setup for linear regression

Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (features, covariates, context, predictors, etc)

Output: $y \in \mathbb{R}$ (responses, targets, outcomes, etc)

Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

Linear model: $f : \mathbb{R}^{\mathsf{D}} \rightarrow \mathbb{R}$, with $f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{D} w_d x_d = w_0 + \boldsymbol{w}^{\mathbf{T}}\boldsymbol{x}$ (superscript ^{T} stands for transpose), i.e. a *hyper-plane* parametrized by

- $\boldsymbol{w} = [w_1 \; w_2 \; \cdots \; w_{\mathsf{D}}]^{\mathrm{T}}$ (weights, weight vector, parameter vector, etc)
- bias w_0

NOTE: for notation convenience, very often we

- append 1 to each x as the first feature: $\tilde{\boldsymbol{x}} = [1 \; x_1 \; x_2 \; \ldots \; x_{\mathsf{D}}]^{\mathrm{T}}$
- let $\tilde{\boldsymbol{w}} = [w_0 \; w_1 \; w_2 \; \cdots \; w_{\mathsf{D}}]^{\mathrm{T}}$, a concise representation of all $D + 1$ parameters
- the model becomes simply $f(\boldsymbol{x}) = \tilde{\boldsymbol{w}}^{\mathbf{T}}\tilde{\boldsymbol{x}}$
- sometimes just use $\boldsymbol{w}, \boldsymbol{x}, \mathsf{D}$ for $\tilde{\boldsymbol{w}}, \tilde{\boldsymbol{x}}, \mathsf{D} + 1$!

Goal

Minimize total squared error

$$\sum_n (f(\boldsymbol{x}_n) - y_n)^2 = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)^2$$

Goal

Minimize total squared error

- **Residual Sum of Squares** (RSS), a function of $\tilde{\boldsymbol{w}}$

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (f(\boldsymbol{x}_n) - y_n)^2 = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)^2$$

Goal

Minimize total squared error

- **Residual Sum of Squares** (RSS), a function of $\tilde{\boldsymbol{w}}$

$$\text{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (f(\boldsymbol{x}_n) - y_n)^2 = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)^2$$

- find $\tilde{\boldsymbol{w}}^* = \operatorname*{argmin}_{\tilde{\boldsymbol{w}} \in \mathbb{R}^{\mathsf{D}+1}} \text{RSS}(\tilde{\boldsymbol{w}})$, i.e. **least squares solution** (more generally called **empirical risk minimizer**)

Goal

Minimize total squared error

- **Residual Sum of Squares** (RSS), a function of $\tilde{\boldsymbol{w}}$

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (f(\boldsymbol{x}_n) - y_n)^2 = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)^2$$

- find $\tilde{\boldsymbol{w}}^* = \operatorname*{argmin}_{\tilde{\boldsymbol{w}} \in \mathbb{R}^{\mathsf{D}+1}} \mathrm{RSS}(\tilde{\boldsymbol{w}})$, i.e. **least squares solution** (more generally called **empirical risk minimizer**)

- *reduce machine learning to optimization*

Goal

Minimize total squared error

- **Residual Sum of Squares** (RSS), a function of $\tilde{\boldsymbol{w}}$

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (f(\boldsymbol{x}_n) - y_n)^2 = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)^2$$

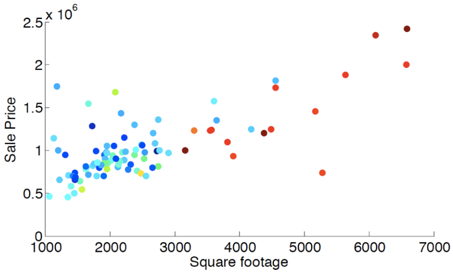
- find $\tilde{\boldsymbol{w}}^* = \operatorname*{argmin}_{\tilde{\boldsymbol{w}} \in \mathbb{R}^{\mathsf{D}+1}} \mathrm{RSS}(\tilde{\boldsymbol{w}})$, i.e. **least squares solution** (more generally called **empirical risk minimizer**)

- *reduce machine learning to optimization*

- in principle can apply any optimization algorithm, but linear regression admits a *closed-form solution*

Warm-up: $\mathsf{D} = 0$

Only one parameter w_0: constant prediction $f(x) = w_0$



f is a horizontal line, where should it be?

Warm-up: $D = 0$

Optimization objective becomes

$$\mathrm{RSS}(w_0) = \sum_n (w_0 - y_n)^2 \qquad \text{(it's a } \textit{quadratic } aw_0^2 + bw_0 + c\text{)}$$

Warm-up: $\mathsf{D} = 0$

Optimization objective becomes

$$\begin{aligned}\mathrm{RSS}(w_0) &= \sum_n (w_0 - y_n)^2 \qquad \text{(it's a } \textit{quadratic } aw_0^2 + bw_0 + c\text{)} \\ &= Nw_0^2 - 2\left(\sum_n y_n\right) w_0 + \text{cnt.}\end{aligned}$$

Warm-up: D $= 0$

Optimization objective becomes

$$\begin{aligned}
\mathrm{RSS}(w_0) &= \sum_n (w_0 - y_n)^2 \qquad \text{(it's a } \textit{quadratic } aw_0^2 + bw_0 + c\text{)} \\
&= Nw_0^2 - 2\left(\sum_n y_n\right) w_0 + \text{cnt.} \\
&= N\left(w_0 - \frac{1}{N}\sum_n y_n\right)^2 + \text{cnt.}
\end{aligned}$$

Warm-up: D $= 0$

Optimization objective becomes

$$\text{RSS}(w_0) = \sum_n (w_0 - y_n)^2 \qquad \text{(it's a } \textit{quadratic } aw_0^2 + bw_0 + c\text{)}$$

$$= Nw_0^2 - 2\left(\sum_n y_n\right) w_0 + \text{cnt.}$$

$$= N\left(w_0 - \frac{1}{N}\sum_n y_n\right)^2 + \text{cnt.}$$

It is clear that $w_0^* = \frac{1}{N}\sum_n y_n$, i.e. the **average**

Warm-up: D $= 0$

Optimization objective becomes

$$\begin{aligned} \text{RSS}(w_0) &= \sum_n (w_0 - y_n)^2 \qquad \text{(it's a } \textit{quadratic } aw_0^2 + bw_0 + c\text{)} \\ &= Nw_0^2 - 2\left(\sum_n y_n\right) w_0 + \text{cnt.} \\ &= N\left(w_0 - \frac{1}{N}\sum_n y_n\right)^2 + \text{cnt.} \end{aligned}$$

It is clear that $w_0^* = \frac{1}{N}\sum_n y_n$, i.e. the **average**

Exercise: what if we use absolute error instead of squared error?

Warm-up: D $= 1$

Optimization objective becomes

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (w_0 + w_1 x_n - y_n)^2$$

Warm-up: D $= 1$

Optimization objective becomes

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (w_0 + w_1 x_n - y_n)^2$$

General approach: find *stationary points*, i.e., points with *zero gradient*

$$\begin{cases} \frac{\partial \mathrm{RSS}(\tilde{\boldsymbol{w}})}{\partial w_0} = 0 \\ \frac{\partial \mathrm{RSS}(\tilde{\boldsymbol{w}})}{\partial w_1} = 0 \end{cases} \Rightarrow \begin{array}{ll} \sum_n (w_0 + w_1 x_n - y_n) & = 0 \\ \sum_n (w_0 + w_1 x_n - y_n) x_n & = 0 \end{array}$$

Warm-up: D $= 1$

Optimization objective becomes

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (w_0 + w_1 x_n - y_n)^2$$

General approach: find *stationary points*, i.e., points with *zero gradient*

$$\begin{cases} \frac{\partial \mathrm{RSS}(\tilde{\boldsymbol{w}})}{\partial w_0} = 0 \\ \frac{\partial \mathrm{RSS}(\tilde{\boldsymbol{w}})}{\partial w_1} = 0 \end{cases} \Rightarrow \begin{array}{ll} \sum_n (w_0 + w_1 x_n - y_n) & = 0 \\ \sum_n (w_0 + w_1 x_n - y_n) x_n & = 0 \end{array}$$

$$\Rightarrow \begin{array}{ll} N w_0 + w_1 \sum_n x_n & = \sum_n y_n \\ w_0 \sum_n x_n + w_1 \sum_n x_n^2 & = \sum_n y_n x_n \end{array}$$ (**a linear system**)

Warm-up: D $= 1$

Optimization objective becomes

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (w_0 + w_1 x_n - y_n)^2$$

General approach: find *stationary points*, i.e., points with *zero gradient*

$$\begin{cases} \frac{\partial \mathrm{RSS}(\tilde{\boldsymbol{w}})}{\partial w_0} = 0 \\ \frac{\partial \mathrm{RSS}(\tilde{\boldsymbol{w}})}{\partial w_1} = 0 \end{cases} \Rightarrow \begin{array}{ll} \sum_n (w_0 + w_1 x_n - y_n) & = 0 \\ \sum_n (w_0 + w_1 x_n - y_n) x_n & = 0 \end{array}$$

$$\Rightarrow \begin{array}{ll} N w_0 + w_1 \sum_n x_n & = \sum_n y_n \\ w_0 \sum_n x_n + w_1 \sum_n x_n^2 & = \sum_n y_n x_n \end{array} \qquad \text{(a linear system)}$$

$$\Rightarrow \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} = \begin{pmatrix} \sum_n y_n \\ \sum_n x_n y_n \end{pmatrix}$$

Least square solution for D = 1

$$\Rightarrow \begin{pmatrix} w_0^* \\ w_1^* \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum_n y_n \\ \sum_n x_n y_n \end{pmatrix}$$

(assuming the matrix is invertible)

Least square solution for D $= 1$

$$\Rightarrow \begin{pmatrix} w_0^* \\ w_1^* \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum_n y_n \\ \sum_n x_n y_n \end{pmatrix}$$

(assuming the matrix is invertible)

Are stationary points minimizers?

Least square solution for D = 1

$$\Rightarrow \begin{pmatrix} w_0^* \\ w_1^* \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum_n y_n \\ \sum_n x_n y_n \end{pmatrix}$$

(assuming the matrix is invertible)

Are stationary points minimizers?

- yes for **convex** objectives (RSS is convex in $\tilde{\boldsymbol{w}}$)

Least square solution for D = 1

$$\Rightarrow \begin{pmatrix} w_0^* \\ w_1^* \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum_n y_n \\ \sum_n x_n y_n \end{pmatrix}$$

(assuming the matrix is invertible)

Are stationary points minimizers?

- yes for **convex** objectives (RSS is convex in $\tilde{\boldsymbol{w}}$)
- not true in general

General least square solution

Objective

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)^2$$

General least square solution

Objective

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)^2$$

Again, find stationary points (**multivariate calculus**)

$$\nabla \mathrm{RSS}(\tilde{\boldsymbol{w}}) = 2 \sum_n \tilde{\boldsymbol{x}}_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)$$

General least square solution

Objective

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)^2$$

Again, find stationary points (**multivariate calculus**)

$$\nabla \mathrm{RSS}(\tilde{\boldsymbol{w}}) = 2 \sum_n \tilde{\boldsymbol{x}}_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n) \propto \left(\sum_n \tilde{\boldsymbol{x}}_n \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \right) \tilde{\boldsymbol{w}} - \sum_n \tilde{\boldsymbol{x}}_n y_n$$

General least square solution

Objective

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)^2$$

Again, find stationary points (**multivariate calculus**)

$$\begin{aligned}\nabla \mathrm{RSS}(\tilde{\boldsymbol{w}}) &= 2 \sum_n \tilde{\boldsymbol{x}}_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n) \propto \left(\sum_n \tilde{\boldsymbol{x}}_n \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \right) \tilde{\boldsymbol{w}} - \sum_n \tilde{\boldsymbol{x}}_n y_n \\ &= (\tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}}) \tilde{\boldsymbol{w}} - \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y}\end{aligned}$$

where

$$\tilde{\boldsymbol{X}} = \begin{pmatrix} \tilde{\boldsymbol{x}}_1^{\mathrm{T}} \\ \tilde{\boldsymbol{x}}_2^{\mathrm{T}} \\ \vdots \\ \tilde{\boldsymbol{x}}_{\mathsf{N}}^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{\mathsf{N} \times (D+1)}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{\mathsf{N}} \end{pmatrix} \in \mathbb{R}^{\mathsf{N}}$$

General least square solution

Objective

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)^2$$

Again, find stationary points (**multivariate calculus**)

$$\begin{aligned} \nabla \mathrm{RSS}(\tilde{\boldsymbol{w}}) &= 2 \sum_n \tilde{\boldsymbol{x}}_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n) \propto \left(\sum_n \tilde{\boldsymbol{x}}_n \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \right) \tilde{\boldsymbol{w}} - \sum_n \tilde{\boldsymbol{x}}_n y_n \\ &= (\tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}}) \tilde{\boldsymbol{w}} - \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y} = \boldsymbol{0} \end{aligned}$$

where

$$\tilde{\boldsymbol{X}} = \begin{pmatrix} \tilde{\boldsymbol{x}}_1^{\mathrm{T}} \\ \tilde{\boldsymbol{x}}_2^{\mathrm{T}} \\ \vdots \\ \tilde{\boldsymbol{x}}_{\mathsf{N}}^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{\mathsf{N} \times (D+1)}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{\mathsf{N}} \end{pmatrix} \in \mathbb{R}^{\mathsf{N}}$$

General least square solution

$$(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})\tilde{\boldsymbol{w}} - \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{0} \quad \Rightarrow \quad \tilde{\boldsymbol{w}}^* = (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

assuming $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ (**covariance matrix**) is invertible for now.

General least square solution

$$(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})\tilde{\boldsymbol{w}} - \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{0} \quad \Rightarrow \quad \tilde{\boldsymbol{w}}^* = (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

assuming $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ (**covariance matrix**) is invertible for now.

Again by convexity $\tilde{\boldsymbol{w}}^*$ is the minimizer of RSS.

General least square solution

$$(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})\tilde{\boldsymbol{w}} - \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{0} \quad \Rightarrow \quad \tilde{\boldsymbol{w}}^* = (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

assuming $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ (**covariance matrix**) is invertible for now.

Again by convexity $\tilde{\boldsymbol{w}}^*$ is the minimizer of RSS.

Verify the solution when $\mathsf{D} = 1$:

$$\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_{\mathsf{N}} \end{pmatrix} \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdots & \cdots \\ 1 & x_{\mathsf{N}} \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}$$

General least square solution

$$(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})\tilde{\boldsymbol{w}} - \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{0} \quad \Rightarrow \quad \tilde{\boldsymbol{w}}^* = (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

assuming $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ (**covariance matrix**) is invertible for now.

Again by convexity $\tilde{\boldsymbol{w}}^*$ is the minimizer of RSS.

Verify the solution when $\mathsf{D} = 1$**:**

$$\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_{\mathsf{N}} \end{pmatrix} \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdots & \cdots \\ 1 & x_{\mathsf{N}} \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}$$

when $\mathsf{D} = 0$**:** $(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1} = \frac{1}{N}$, $\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} = \sum_n y_n$

Another approach

RSS is a **quadratic**, so let's complete the square:

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n - y_n)^2 = \|\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\|_2^2$$

Another approach

RSS is a **quadratic**, so let's complete the square:

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n - y_n)^2 = \|\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\|_2^2$$

$$= \left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\right)^{\mathrm{T}} \left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\right)$$

Another approach

RSS is a **quadratic**, so let's complete the square:

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n - y_n)^2 = \|\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\|_2^2$$

$$= \left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\right)^{\mathrm{T}} \left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\right)$$

$$= \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}} - \boldsymbol{y}^{\mathrm{T}} \tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}} - \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y} + \text{cnt.}$$

Another approach

RSS is a **quadratic**, so let's complete the square:

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n - y_n)^2 = \|\tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}} - \boldsymbol{y}\|_2^2$$

$$= \left(\tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}} - \boldsymbol{y}\right)^{\mathrm{T}} \left(\tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}} - \boldsymbol{y}\right)$$

$$= \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}} - \boldsymbol{y}^{\mathrm{T}} \tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}} - \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y} + \text{cnt.}$$

$$= \left(\tilde{\boldsymbol{w}} - (\tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}})^{-1} \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y}\right)^{\mathrm{T}} \left(\tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}}\right) \left(\tilde{\boldsymbol{w}} - (\tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}})^{-1} \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y}\right) + \text{cnt.}$$

Another approach

RSS is a **quadratic**, so let's complete the square:

$$\begin{aligned}
\mathrm{RSS}(\tilde{\boldsymbol{w}}) &= \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{x}}_n - y_n)^2 = \|\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\|_2^2 \\
&= \left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\right)^{\mathrm{T}}\left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\right) \\
&= \tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} + \text{cnt.} \\
&= \left(\tilde{\boldsymbol{w}} - (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right)^{\mathrm{T}}\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)\left(\tilde{\boldsymbol{w}} - (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right) + \text{cnt.}
\end{aligned}$$

Note: $\boldsymbol{u}^{\mathrm{T}}\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)\boldsymbol{u} = \left(\tilde{\boldsymbol{X}}\boldsymbol{u}\right)^{\mathrm{T}}\tilde{\boldsymbol{X}}\boldsymbol{u} = \|\tilde{\boldsymbol{X}}\boldsymbol{u}\|_2^2 \geq 0$ and is 0 if $\boldsymbol{u} = 0$.

Another approach

RSS is a **quadratic**, so let's complete the square:

$$\begin{aligned}
\mathrm{RSS}(\tilde{\boldsymbol{w}}) &= \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{x}}_n - y_n)^2 = \|\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\|_2^2 \\
&= \left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\right)^{\mathrm{T}}\left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\right) \\
&= \tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} + \text{cnt.} \\
&= \left(\tilde{\boldsymbol{w}} - (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right)^{\mathrm{T}}\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)\left(\tilde{\boldsymbol{w}} - (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right) + \text{cnt.}
\end{aligned}$$

Note: $\boldsymbol{u}^{\mathrm{T}}\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)\boldsymbol{u} = \left(\tilde{\boldsymbol{X}}\boldsymbol{u}\right)^{\mathrm{T}}\tilde{\boldsymbol{X}}\boldsymbol{u} = \|\tilde{\boldsymbol{X}}\boldsymbol{u}\|_2^2 \geq 0$ and is 0 if $\boldsymbol{u} = 0$.

So $\tilde{\boldsymbol{w}}^* = (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$ is the minimizer.

Computational complexity

Bottleneck of computing

$$\tilde{\boldsymbol{w}}^* = \left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

is to invert the matrix $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} \in \mathbb{R}^{(\mathsf{D}+1)\times(\mathsf{D}+1)}$

- naively need $O(\mathsf{D}^3)$ time

Computational complexity

Bottleneck of computing

$$\tilde{\boldsymbol{w}}^* = \left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

is to invert the matrix $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} \in \mathbb{R}^{(\mathsf{D}+1)\times(\mathsf{D}+1)}$

- naively need $O(\mathsf{D}^3)$ time
- there are many faster approaches (such as conjugate gradient)

What if $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ is not invertible

What does that imply?

What if $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ is not invertible

What does that imply?

Recall $\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)\boldsymbol{w}^* = \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$.

What if $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ is not invertible

What does that imply?

Recall $\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)\boldsymbol{w}^* = \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$. If $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ not invertible, this equation has

- no solution

What if $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ is not invertible

What does that imply?

Recall $\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)\boldsymbol{w}^* = \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$. If $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ not invertible, this equation has

- no solution
- or infinitely many solutions

What if $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ is not invertible

What does that imply?

Recall $\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)\boldsymbol{w}^* = \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$. If $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ not invertible, this equation has

- no solution ($\Rightarrow$ RSS has no minimizer? ✗)
- or infinitely many solutions ($\Rightarrow$ infinitely many minimizers ✓)

What if $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ is not invertible

Why would that happen?

What if $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ is not invertible

Why would that happen?

One situation: $\mathsf{N} < \mathsf{D} + 1$, i.e. not enough data to estimate all parameters.

What if $\tilde{X}^{\mathrm{T}}\tilde{X}$ is not invertible

Why would that happen?

One situation: $\mathsf{N} < \mathsf{D} + 1$, i.e. not enough data to estimate all parameters.

Example: $\mathsf{D} = \mathsf{N} = 1$

sqft	sale price
1000	500K

What if $\tilde{X}^{\mathrm{T}}\tilde{X}$ is not invertible

Why would that happen?

One situation: $\mathsf{N} < \mathsf{D} + 1$, i.e. not enough data to estimate all parameters.

Example: $\mathsf{D} = \mathsf{N} = 1$

sqft	sale price
1000	500K

Any line passing this single point is a minimizer of RSS.

How about the following?

$\mathsf{D} = 1, \mathsf{N} = 2$

sqft	sale price
1000	500K
1000	600K

How about the following?

$D = 1, N = 2$

sqft	sale price
1000	500K
1000	600K

Any line passing **the average** is a minimizer of RSS.

How about the following?

$\mathsf{D} = 1, \mathsf{N} = 2$

sqft	sale price
1000	500K
1000	600K

Any line passing **the average** is a minimizer of RSS.

$\mathsf{D} = 2, \mathsf{N} = 3$**?**

sqft	#bedroom	sale price
1000	2	500K
1500	3	700K
2000	4	800K

How about the following?

$D = 1, N = 2$

sqft	sale price
1000	500K
1000	600K

Any line passing **the average** is a minimizer of RSS.

$D = 2, N = 3$**?**

sqft	#bedroom	sale price
1000	2	500K
1500	3	700K
2000	4	800K

Again *infinitely many minimizers*.

How to resolve this issue?

Intuition: what does inverting $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ do?

eigendecomposition: $$\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_{\mathsf{D}} & 0 \\ 0 & \cdots & 0 & \lambda_{\mathsf{D}+1} \end{bmatrix} \boldsymbol{U}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_{\mathsf{D}+1} \geq 0$ are **eigenvalues**.

How to resolve this issue?

Intuition: what does inverting $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ do?

$$\textbf{eigendecomposition}: \quad \tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_{\mathsf{D}} & 0 \\ 0 & \cdots & 0 & \lambda_{\mathsf{D}+1} \end{bmatrix} \boldsymbol{U}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_{\mathsf{D}+1} \geq 0$ are **eigenvalues**.

$$\textbf{inverse}: \quad (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \frac{1}{\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_{\mathsf{D}}} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{\mathsf{D}+1}} \end{bmatrix} \boldsymbol{U}$$

i.e. just invert the eigenvalues

How to solve this problem?

Non-invertible $\Rightarrow$ some eigenvalues are 0.

How to solve this problem?

Non-invertible $\Rightarrow$ some eigenvalues are 0.

One natural fix: add something positive

$$\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} + \lambda \boldsymbol{I} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \lambda_1 + \lambda & 0 & \cdots & 0 \\ 0 & \lambda_2 + \lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_{\mathsf{D}} + \lambda & 0 \\ 0 & \cdots & 0 & \lambda_{\mathsf{D}+1} + \lambda \end{bmatrix} \boldsymbol{U}$$

where $\lambda > 0$ and $\boldsymbol{I}$ is the identity matrix.

How to solve this problem?

Non-invertible $\Rightarrow$ some eigenvalues are 0.

One natural fix: add something positive

$$\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} + \lambda\boldsymbol{I} = \boldsymbol{U}^{\mathrm{T}}\begin{bmatrix} \lambda_1+\lambda & 0 & \cdots & 0 \\ 0 & \lambda_2+\lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_{\mathsf{D}}+\lambda & 0 \\ 0 & \cdots & 0 & \lambda_{\mathsf{D}+1}+\lambda \end{bmatrix}\boldsymbol{U}$$

where $\lambda > 0$ and $\boldsymbol{I}$ is the identity matrix. Now it is invertible:

$$(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} + \lambda\boldsymbol{I})^{-1} = \boldsymbol{U}^{\mathrm{T}}\begin{bmatrix} \frac{1}{\lambda_1+\lambda} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2+\lambda} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_{\mathsf{D}}+\lambda} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{\mathsf{D}+1}+\lambda} \end{bmatrix}\boldsymbol{U}$$

Fix the problem

The solution becomes

$$\tilde{\boldsymbol{w}}^* = \left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} + \lambda \boldsymbol{I}\right)^{-1} \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

Fix the problem

The solution becomes

$$\tilde{\boldsymbol{w}}^* = \left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} + \lambda \boldsymbol{I}\right)^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

- not a minimizer of the original RSS

Fix the problem

The solution becomes

$$\tilde{\boldsymbol{w}}^* = \left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} + \lambda \boldsymbol{I}\right)^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

- not a minimizer of the original RSS
- more than an arbitrary hack (as we will see soon)

Fix the problem

The solution becomes

$$\tilde{\boldsymbol{w}}^* = \left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} + \lambda \boldsymbol{I}\right)^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

- not a minimizer of the original RSS
- more than an arbitrary hack (as we will see soon)

λ is a *hyper-parameter*, can be tuned by cross-validation.

Comparison to NNC

Non-parametric versus Parametric

- **Non-parametric methods**: the size of the model *grows* with the size of the training set.
  - e.g. NNC, the training set itself needs to be kept in order to predict. Thus, the size of the model is the size of the training set.

Comparison to NNC

Non-parametric versus Parametric

- **Non-parametric methods**: the size of the model *grows* with the size of the training set.
  - e.g. NNC, the training set itself needs to be kept in order to predict. Thus, the size of the model is the size of the training set.
- **Parametric methods**: the size of the model does *not grow* with the size of the training set N.
  - e.g. linear regression, $D+1$ parameters, independent of N.

Outline

1 Linear regression

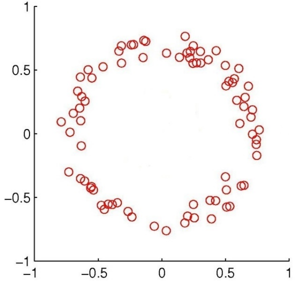
2 Linear regression with nonlinear basis

3 Overfitting and preventing overfitting

4 A Detour of Numerical Optimization Methods

What if linear model is not a good fit?

Example: a straight line is a bad fit for the following data



Solution: nonlinearly transformed features

1. Use a nonlinear mapping

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^D \to \boldsymbol{z} \in \mathbb{R}^M$$

to transform the data to a more complicated feature space

Solution: nonlinearly transformed features

1. Use a nonlinear mapping

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^D \to \boldsymbol{z} \in \mathbb{R}^M$$

to transform the data to a more complicated feature space

2. Then apply linear regression (hope: linear model is a better fit for the new feature space).

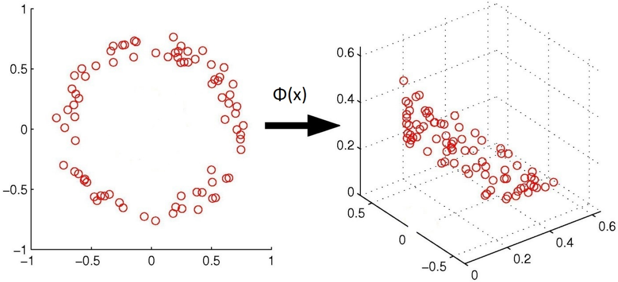
Solution: nonlinearly transformed features

1. Use a nonlinear mapping

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^D \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

to transform the data to a more complicated feature space

2. Then apply linear regression (hope: linear model is a better fit for the new feature space).



Regression with nonlinear basis

Model: $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$ where $\boldsymbol{w} \in \mathbb{R}^M$

Regression with nonlinear basis

Model: $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$ where $\boldsymbol{w} \in \mathbb{R}^M$

Objective:

$$\mathrm{RSS}(\boldsymbol{w}) = \sum_n \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) - y_n\right)^2$$

Regression with nonlinear basis

Model: $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$ where $\boldsymbol{w} \in \mathbb{R}^M$

Objective:

$$\mathrm{RSS}(\boldsymbol{w}) = \sum_n \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_n) - y_n\right)^2$$

Similar least square solution:

$$\boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y} \quad \text{where} \quad \boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}} \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{N\times M}$$

Example

Polynomial basis functions for $\mathrm{D} = 1$

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \quad \Rightarrow \quad f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

Example

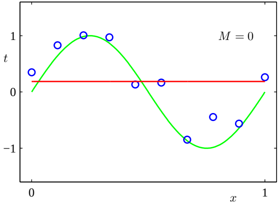
Polynomial basis functions for $\mathrm{D} = 1$

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \quad \Rightarrow \quad f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

Learning a linear model in the new space
= learning an *M-degree polynomial model* in the original space

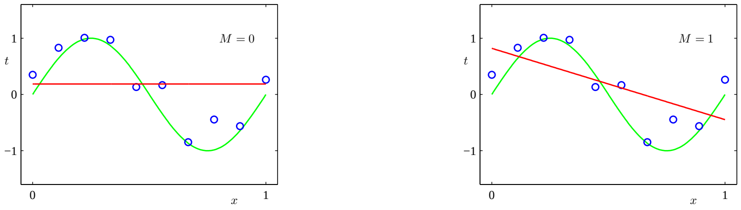
Example

Fitting a noisy sine function with a polynomial ($M = 0, 1,$ or 3):



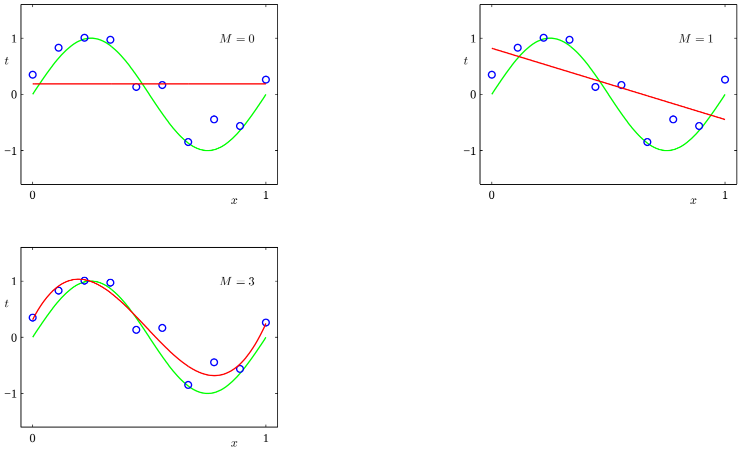
Example

Fitting a noisy sine function with a polynomial ($M = 0, 1,$ or 3):



Example

Fitting a noisy sine function with a polynomial ($M = 0, 1,$ or 3):



Why nonlinear?

Can I use a fancy **linear feature map**?

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} x_1 - x_2 \\ 3x_4 - x_3 \\ 2x_1 + x_4 + x_5 \\ \vdots \end{bmatrix} = \boldsymbol{A}\boldsymbol{x} \quad \text{for some } \boldsymbol{A} \in \mathbb{R}^{\mathsf{M}\times\mathsf{D}}$$

Why nonlinear?

Can I use a fancy **linear feature map**?

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} x_1 - x_2 \\ 3x_4 - x_3 \\ 2x_1 + x_4 + x_5 \\ \vdots \end{bmatrix} = \boldsymbol{A}\boldsymbol{x} \quad \text{for some } \boldsymbol{A} \in \mathbb{R}^{\mathsf{M}\times\mathsf{D}}$$

No, it basically *does nothing* since

$$\min_{\boldsymbol{w}\in\mathbb{R}^{\mathsf{M}}} \sum_n \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}_n - y_n\right)^2 = \min_{\boldsymbol{w}'\in\mathrm{Im}(\boldsymbol{A}^{\mathrm{T}})\subset\mathbb{R}^{\mathsf{D}}} \sum_n \left(\boldsymbol{w}'^{\mathrm{T}}\boldsymbol{x}_n - y_n\right)^2$$

Why nonlinear?

Can I use a fancy **linear feature map**?

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} x_1 - x_2 \\ 3x_4 - x_3 \\ 2x_1 + x_4 + x_5 \\ \vdots \end{bmatrix} = \boldsymbol{A}\boldsymbol{x} \quad \text{for some } \boldsymbol{A} \in \mathbb{R}^{\mathsf{M}\times\mathsf{D}}$$

No, it basically *does nothing* since

$$\min_{\boldsymbol{w}\in\mathbb{R}^{\mathsf{M}}} \sum_n \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}_n - y_n\right)^2 = \min_{\boldsymbol{w}'\in \mathrm{Im}(\boldsymbol{A}^{\mathrm{T}})\subset\mathbb{R}^{\mathsf{D}}} \sum_n \left(\boldsymbol{w}'^{\mathrm{T}}\boldsymbol{x}_n - y_n\right)^2$$

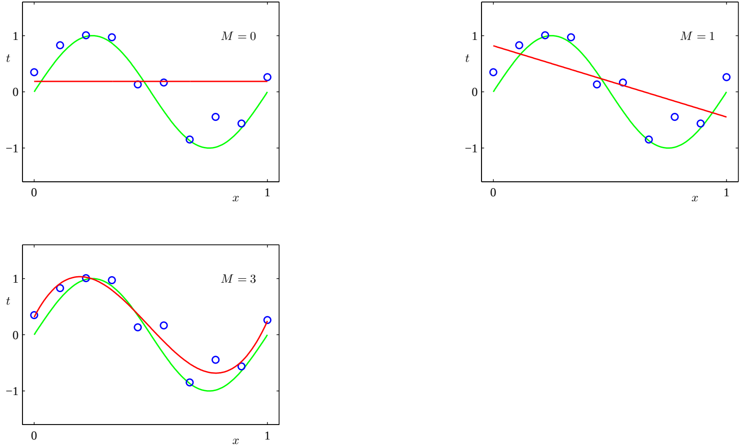
We will see more nonlinear mappings soon.

Outline

1. Linear regression
2. Linear regression with nonlinear basis
3. Overfitting and preventing overfitting
4. A Detour of Numerical Optimization Methods

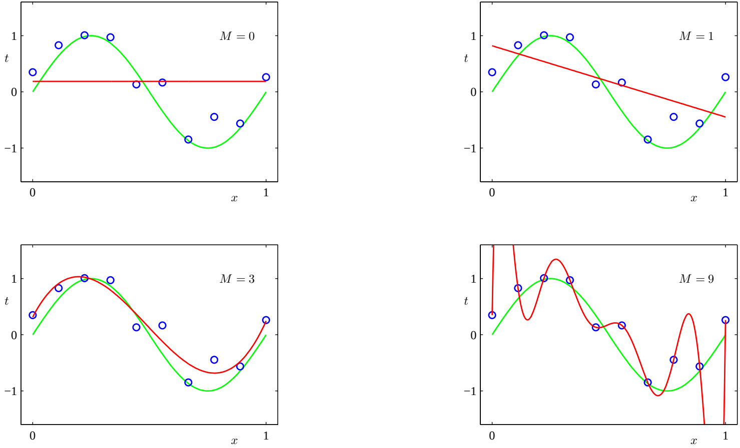
Should we use a very complicated mapping?

Ex: fitting a noisy sine function with a polynomial:



Should we use a very complicated mapping?

Ex: fitting a noisy sine function with a polynomial:



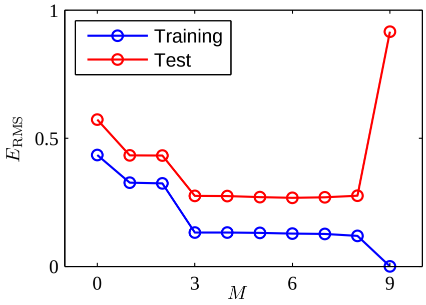
Underfitting and Overfitting

$M \leq 2$ is *underfitting* the data

- large training error
- large test error

$M \geq 9$ is *overfitting* the data

- small training error
- **large test error**



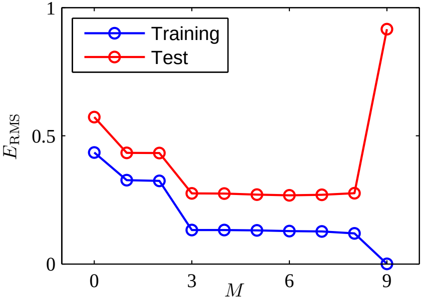
Underfitting and Overfitting

$M \leq 2$ is *underfitting* the data

- large training error
- large test error

$M \geq 9$ is *overfitting* the data

- small training error
- **large test error**



More complicated models $\Rightarrow$ larger gap between training and test error

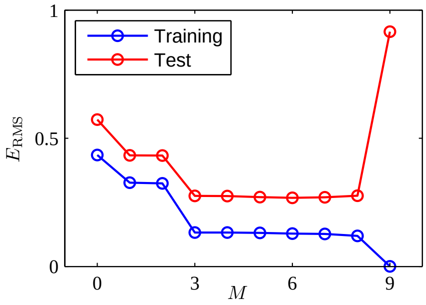
Underfitting and Overfitting

$M \leq 2$ is *underfitting* the data

- large training error
- large test error

$M \geq 9$ is *overfitting* the data

- small training error
- **large test error**

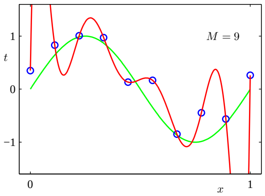


More complicated models $\Rightarrow$ larger gap between training and test error

How to prevent overfitting?

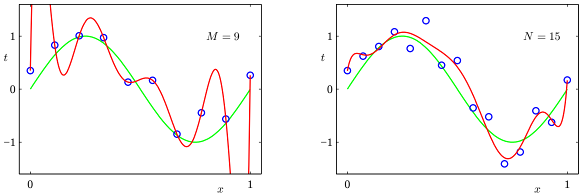
Method 1: use more training data

The more, the merrier



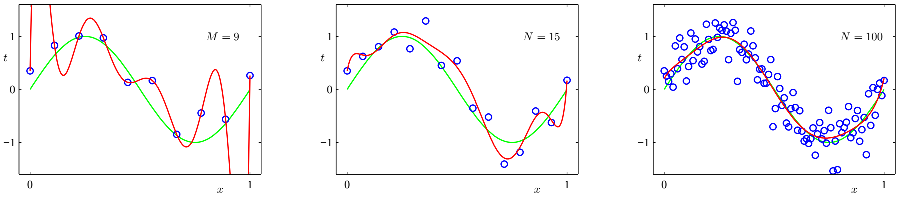
Method 1: use more training data

The more, the merrier



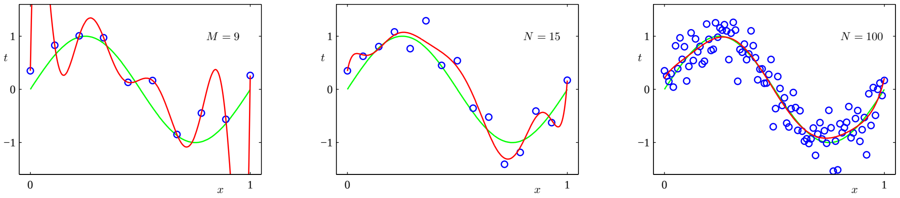
Method 1: use more training data

The more, the merrier



Method 1: use more training data

The more, the merrier



More data ⇒ smaller gap between training and test error

Method 2: control the model complexity

For polynomial basis, the **degree** M clearly controls the complexity

- use cross-validation to pick hyper-parameter M

Method 2: control the model complexity

For polynomial basis, the **degree** M clearly controls the complexity

- use cross-validation to pick hyper-parameter M

When M or in general Φ is fixed, are there still other ways to control complexity?

Magnitude of weights

Least square solution for the polynomial example:

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

Magnitude of weights

Least square solution for the polynomial example:

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

Intuitively, **large weights $\Rightarrow$ more complex model**

How to make $\boldsymbol{w}$ small?

Regularized linear regression: new objective

$$\mathcal{E}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

Goal: find $\boldsymbol{w}^* = \operatorname{argmin}_w \mathcal{E}(\boldsymbol{w})$

How to make $\boldsymbol{w}$ small?

Regularized linear regression: new objective

$$\mathcal{E}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

Goal: find $\boldsymbol{w}^* = \mathrm{argmin}_w \, \mathcal{E}(\boldsymbol{w})$

- $R : \mathbb{R}^{\mathsf{D}} \rightarrow \mathbb{R}^+$ is the *regularizer*
  - measure how complex the model $\boldsymbol{w}$ is, penalize complex models
  - common choices: $\|\boldsymbol{w}\|_2^2$, $\|\boldsymbol{w}\|_1$, etc.

How to make $\boldsymbol{w}$ small?

Regularized linear regression: new objective

$$\mathcal{E}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

Goal: find $\boldsymbol{w}^* = \operatorname{argmin}_w \mathcal{E}(\boldsymbol{w})$

- $R : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}^+$ is the *regularizer*
  - measure how complex the model $\boldsymbol{w}$ is, penalize complex models
  - common choices: $\|\boldsymbol{w}\|_2^2$, $\|\boldsymbol{w}\|_1$, etc.
- $\lambda > 0$ is the *regularization coefficient*
  - $\lambda = 0$, no regularization
  - $\lambda \to +\infty$, $\boldsymbol{w} \to \operatorname{argmin}_w R(\boldsymbol{w})$
  - i.e. control **trade-off** between training error and complexity

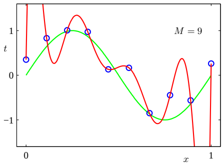
The effect of λ

when we increase regularization coefficient λ

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
w_0	0.35	0.35	0.13
w_1	232.37	4.74	-0.05
w_2	-5321.83	-0.77	-0.06
w_3	48568.31	-31.97	-0.06
w_4	-231639.30	-3.89	-0.03
w_5	640042.26	55.28	-0.02
w_6	-1061800.52	41.32	-0.01
w_7	1042400.18	-45.95	-0.00
w_8	-557682.99	-91.53	0.00
w_9	125201.43	72.68	0.01

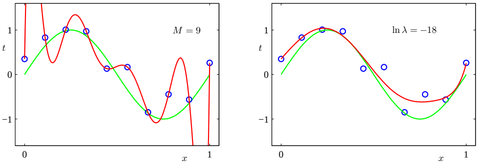
The trade-off

when we increase regularization coefficient λ



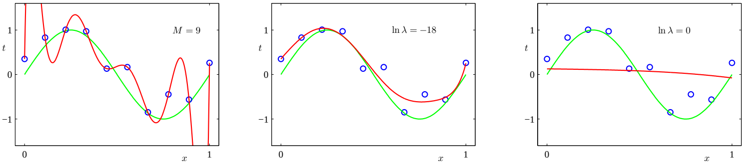
The trade-off

when we increase regularization coefficient λ



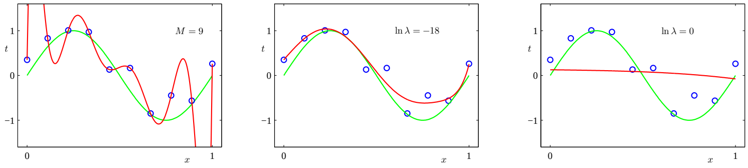
The trade-off

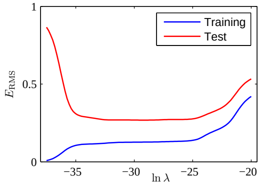
when we increase regularization coefficient λ



The trade-off

when we increase regularization coefficient λ





How to solve the new objective?

Simple for $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$:

$$\mathcal{E}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$$

How to solve the new objective?

Simple for $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$:

$$\mathcal{E}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$$

$$\nabla\mathcal{E}(\boldsymbol{w}) = 2(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}) + 2\lambda\boldsymbol{w} = 0$$

How to solve the new objective?

Simple for $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$:

$$\mathcal{E}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$$

$$\nabla\mathcal{E}(\boldsymbol{w}) = 2(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}) + 2\lambda\boldsymbol{w} = 0$$
$$\Rightarrow \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)\boldsymbol{w} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

How to solve the new objective?

Simple for $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$:

$$\mathcal{E}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{\Phi w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$$

$$\nabla\mathcal{E}(\boldsymbol{w}) = 2(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi w} - \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}) + 2\lambda\boldsymbol{w} = 0$$

$$\Rightarrow \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)\boldsymbol{w} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

$$\Rightarrow \boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

How to solve the new objective?

Simple for $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$:

$$\mathcal{E}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$$

$$\nabla\mathcal{E}(\boldsymbol{w}) = 2(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}) + 2\lambda\boldsymbol{w} = 0$$
$$\Rightarrow \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)\boldsymbol{w} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$
$$\Rightarrow \boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

Note the same form as in the fix when $\boldsymbol{X}^T\boldsymbol{X}$ is not invertible!

How to solve the new objective?

Simple for $R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$:

$$\mathcal{E}(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$$

$$\begin{aligned} \nabla\mathcal{E}(\boldsymbol{w}) &= 2(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}) + 2\lambda\boldsymbol{w} = 0 \\ &\Rightarrow \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)\boldsymbol{w} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y} \\ &\Rightarrow \boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y} \end{aligned}$$

Note the same form as in the fix when $\boldsymbol{X}^T\boldsymbol{X}$ *is not invertible!*

For other regularizers, as long as it's **convex**, standard optimization algorithms can be applied.

Equivalent form

Regularization is also sometimes formulated as

$$\operatorname*{argmin}_{\boldsymbol{w}} \mathrm{RSS}(w) \quad \textbf{subject to } R(\boldsymbol{w}) \leq \beta$$

where β is some hyper-parameter.

Equivalent form

Regularization is also sometimes formulated as

$$\operatorname*{argmin}_{\boldsymbol{w}} \mathrm{RSS}(w) \quad \textbf{subject to } R(\boldsymbol{w}) \leq \beta$$

where β is some hyper-parameter.

Finding the solution becomes a *constrained optimization problem*.

Equivalent form

Regularization is also sometimes formulated as

$$\underset{\boldsymbol{w}}{\operatorname{argmin}}\, \mathrm{RSS}(w) \quad \textbf{subject to } R(\boldsymbol{w}) \leq \beta$$

where β is some hyper-parameter.

Finding the solution becomes a *constrained optimization problem*.

Choosing either λ or β can be done by cross-validation.

Summary

$$\boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

Summary

$$\boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

Important to understand the derivation than remembering the formula

Summary

$$\boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

Important to understand the derivation than remembering the formula

Overfitting: small training error but large test error

Summary

$$\boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$$

Important to understand the derivation than remembering the formula

Overfitting: small training error but large test error

Preventing Overfitting: more data + regularization

Recall the question

Typical steps of developing a machine learning system:

- Collect data, split into training, development, and test sets.
- *Train a model with a machine learning algorithm.* Most often we apply cross-validation to tune hyper-parameters.
- Evaluate using the test data and report performance.
- Use the model to predict future/make decisions.

How to do the *red part* exactly?

General idea to derive ML algorithms

1. Pick a set of **models** $\mathcal{F}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{D}}\}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Phi}(\boldsymbol{x}) \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{M}}\}$

General idea to derive ML algorithms

1. Pick a set of **models** $\mathcal{F}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{D}}\}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Phi}(\boldsymbol{x}) \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{M}}\}$
2. Define **error/loss** $L(y', y)$

General idea to derive ML algorithms

1. Pick a set of **models** $\mathcal{F}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{D}}\}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Phi}(\boldsymbol{x}) \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{M}}\}$

2. Define **error/loss** $L(y', y)$

3. Find **empirical risk minimizer (ERM)**:

$$\boldsymbol{f}^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{n=1}^{N} L(f(x_n), y_n)$$

General idea to derive ML algorithms

1. Pick a set of **models** $\mathcal{F}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{D}}\}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Phi}(\boldsymbol{x}) \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{M}}\}$
2. Define **error/loss** $L(y', y)$
3. Find **empirical risk minimizer (ERM)**:

$$\boldsymbol{f}^* = \operatorname*{argmin}_{f \in \mathcal{F}} \sum_{n=1}^{N} L(f(x_n), y_n)$$

or **regularized empirical risk minimizer**:

$$\boldsymbol{f}^* = \operatorname*{argmin}_{f \in \mathcal{F}} \sum_{n=1}^{N} L(f(x_n), y_n) + \lambda R(f)$$

General idea to derive ML algorithms

1. Pick a set of **models** $\mathcal{F}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{D}}\}$
   - e.g. $\mathcal{F} = \{f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Phi}(\boldsymbol{x}) \mid \boldsymbol{w} \in \mathbb{R}^{\mathsf{M}}\}$

2. Define **error/loss** $L(y', y)$

3. Find **empirical risk minimizer (ERM)**:

$$\boldsymbol{f}^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{n=1}^{N} L(f(x_n), y_n)$$

or **regularized empirical risk minimizer**:

$$\boldsymbol{f}^* = \underset{f \in \mathcal{F}}{\operatorname{argmin}} \sum_{n=1}^{N} L(f(x_n), y_n) + \lambda R(f)$$

ML becomes optimization

Outline

1. Linear regression

2. Linear regression with nonlinear basis

3. Overfitting and preventing overfitting

4. A Detour of Numerical Optimization Methods

Numerical optimization

Problem setup

- Given: a function $F(\boldsymbol{w})$
- Goal: minimize $F(\boldsymbol{w})$ (approximately)

First-order optimization methods

Two simple yet extremely popular methods

- **Gradient Descent (GD)**: simple and fundamental
- **Stochastic Gradient Descent (SGD)**: faster, effective for large-scale problems

First-order optimization methods

Two simple yet extremely popular methods

- **Gradient Descent (GD)**: simple and fundamental
- **Stochastic Gradient Descent (SGD)**: faster, effective for large-scale problems

Gradient is sometimes referred to as *first-order* information of a function. Therefore, these methods are called *first-order methods*.

Gradient Descent (GD)

GD: keep moving in the *negative gradient direction*

Gradient Descent (GD)

GD: keep moving in the *negative gradient direction*

Start from some $\boldsymbol{w}^{(0)}$. For $t = 0, 1, 2, \ldots$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)})$$

where $\eta > 0$ is called step size or learning rate

Gradient Descent (GD)

GD: keep moving in the *negative gradient direction*

Start from some $\boldsymbol{w}^{(0)}$. For $t = 0, 1, 2, \ldots$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)})$$

where $\eta > 0$ is called step size or learning rate

- in theory η should be set in terms of some parameters of F

Gradient Descent (GD)

GD: keep moving in the *negative gradient direction*

Start from some $\boldsymbol{w}^{(0)}$. For $t = 0, 1, 2, \ldots$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)})$$

where $\eta > 0$ is called step size or learning rate

- in theory η should be set in terms of some parameters of F
- in practice we just try several small values

Gradient Descent (GD)

GD: keep moving in the *negative gradient direction*

Start from some $\boldsymbol{w}^{(0)}$. For $t = 0, 1, 2, \ldots$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)})$$

where $\eta > 0$ is called step size or learning rate

- in theory η should be set in terms of some parameters of F
- in practice we just try several small values
- might need to be **changing** over iterations (think $F(w) = |w|$)

Gradient Descent (GD)

GD: keep moving in the *negative gradient direction*

Start from some $\boldsymbol{w}^{(0)}$. For $t = 0, 1, 2, \ldots$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)})$$

where $\eta > 0$ is called step size or learning rate

- in theory η should be set in terms of some parameters of F
- in practice we just try several small values
- might need to be **changing** over iterations (think $F(w) = |w|$)
- adaptive and automatic step size tuning is an active research area

An example

Example: $F(\boldsymbol{w}) = 0.5(w_1^2 - w_2)^2 + 0.5(w_1 - 1)^2$.

An example

Example: $F(\boldsymbol{w}) = 0.5(w_1^2 - w_2)^2 + 0.5(w_1 - 1)^2$. Gradient is

$$\frac{\partial F}{\partial w_1} = 2(w_1^2 - w_2)w_1 + w_1 - 1 \qquad \frac{\partial F}{\partial w_2} = -(w_1^2 - w_2)$$

An example

Example: $F(\boldsymbol{w}) = 0.5(w_1^2 - w_2)^2 + 0.5(w_1 - 1)^2$. Gradient is

$$\frac{\partial F}{\partial w_1} = 2(w_1^2 - w_2)w_1 + w_1 - 1 \qquad \frac{\partial F}{\partial w_2} = -(w_1^2 - w_2)$$

GD:

- Initialize $w_1^{(0)}$ and $w_2^{(0)}$ (to be 0 or *randomly*), $t = 0$

An example

Example: $F(\boldsymbol{w}) = 0.5(w_1^2 - w_2)^2 + 0.5(w_1 - 1)^2$. Gradient is

$$\frac{\partial F}{\partial w_1} = 2(w_1^2 - w_2)w_1 + w_1 - 1 \qquad \frac{\partial F}{\partial w_2} = -(w_1^2 - w_2)$$

GD:

- Initialize $w_1^{(0)}$ and $w_2^{(0)}$ (to be 0 or *randomly*), $t = 0$
- do

$$w_1^{(t+1)} \leftarrow w_1^{(t)} - \eta \left[2({w_1^{(t)}}^2 - w_2^{(t)})w_1^{(t)} + w_1^{(t)} - 1 \right]$$

$$w_2^{(t+1)} \leftarrow w_2^{(t)} - \eta \left[-({w_1^{(t)}}^2 - w_2^{(t)}) \right]$$

$$t \leftarrow t + 1$$

An example

Example: $F(\boldsymbol{w}) = 0.5(w_1^2 - w_2)^2 + 0.5(w_1 - 1)^2$. Gradient is

$$\frac{\partial F}{\partial w_1} = 2(w_1^2 - w_2)w_1 + w_1 - 1 \qquad \frac{\partial F}{\partial w_2} = -(w_1^2 - w_2)$$

GD:

- Initialize $w_1^{(0)}$ and $w_2^{(0)}$ (to be 0 or *randomly*), $t = 0$
- do

$$w_1^{(t+1)} \leftarrow w_1^{(t)} - \eta \left[2({w_1^{(t)}}^2 - w_2^{(t)})w_1^{(t)} + w_1^{(t)} - 1\right]$$
$$w_2^{(t+1)} \leftarrow w_2^{(t)} - \eta \left[-({w_1^{(t)}}^2 - w_2^{(t)})\right]$$
$$t \leftarrow t + 1$$

- until $F(\boldsymbol{w}^{(t)})$ **does not change much** or t **reaches a fixed number**

Why GD?

Intuition: by first-order **Taylor approximation**

$$F(\boldsymbol{w}) \approx F(\boldsymbol{w}^{(t)}) + \nabla F(\boldsymbol{w}^{(t)})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

Why GD?

Intuition: by first-order **Taylor approximation**

$$F(\boldsymbol{w}) \approx F(\boldsymbol{w}^{(t)}) + \nabla F(\boldsymbol{w}^{(t)})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

GD ensures

$$F(\boldsymbol{w}^{(t+1)}) \approx F(\boldsymbol{w}^{(t)}) - \eta \|\nabla F(\boldsymbol{w}^{(t)})\|_2^2 \leq F(\boldsymbol{w}^{(t)})$$

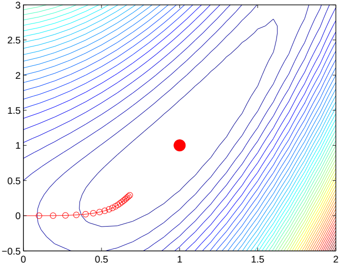
Why GD?

Intuition: by first-order **Taylor approximation**

$$F(\boldsymbol{w}) \approx F(\boldsymbol{w}^{(t)}) + \nabla F(\boldsymbol{w}^{(t)})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

GD ensures

$$F(\boldsymbol{w}^{(t+1)}) \approx F(\boldsymbol{w}^{(t)}) - \eta \|\nabla F(\boldsymbol{w}^{(t)})\|_2^2 \leq F(\boldsymbol{w}^{(t)})$$



reasonable η decreases function value

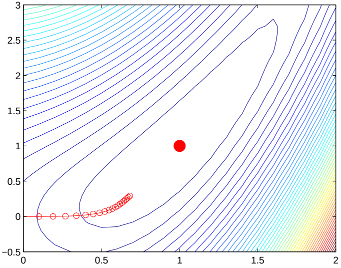
Why GD?

Intuition: by first-order **Taylor approximation**

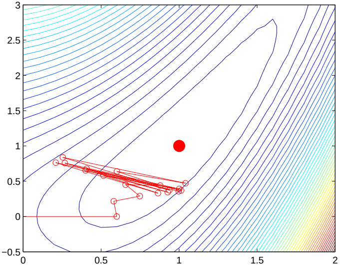
$$F(\boldsymbol{w}) \approx F(\boldsymbol{w}^{(t)}) + \nabla F(\boldsymbol{w}^{(t)})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

GD ensures

$$F(\boldsymbol{w}^{(t+1)}) \approx F(\boldsymbol{w}^{(t)}) - \eta \|\nabla F(\boldsymbol{w}^{(t)})\|_2^2 \leq F(\boldsymbol{w}^{(t)})$$



reasonable η decreases function value



but large η is unstable

Stochastic Gradient Descent (SGD)

GD: keep moving in the negative gradient direction

SGD: keep moving in some *noisy* negative gradient direction

Stochastic Gradient Descent (SGD)

GD: keep moving in the negative gradient direction

SGD: keep moving in some *noisy* negative gradient direction

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \tilde{\nabla} F(\boldsymbol{w}^{(t)})$$

where $\tilde{\nabla} F(\boldsymbol{w}^{(t)})$ is a random variable (called **stochastic gradient**) s.t.

$$\mathbb{E}\left[\tilde{\nabla} F(\boldsymbol{w}^{(t)})\right] = \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(unbiasedness)}$$

Stochastic Gradient Descent (SGD)

GD: keep moving in the negative gradient direction

SGD: keep moving in some *noisy* negative gradient direction

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \tilde{\nabla} F(\boldsymbol{w}^{(t)})$$

where $\tilde{\nabla} F(\boldsymbol{w}^{(t)})$ is a random variable (called **stochastic gradient**) s.t.

$$\mathbb{E}\left[\tilde{\nabla} F(\boldsymbol{w}^{(t)})\right] = \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(unbiasedness)}$$

Key point: it could be *much faster to obtain a stochastic gradient!*
(examples coming soon)

Convergence guarantees — convex objectives

Many for both GD and SGD on **convex objectives**.

Convergence guarantees — convex objectives

Many for both GD and SGD on **convex objectives**.

They tell you how many iterations t (in terms of ϵ) needed to achieve

$$F(\boldsymbol{w}^{(t)}) - F(\boldsymbol{w}^*) \leq \epsilon$$

Convergence guarantees — convex objectives

Many for both GD and SGD on **convex objectives**.

They tell you how many iterations t (in terms of ϵ) needed to achieve

$$F(\boldsymbol{w}^{(t)}) - F(\boldsymbol{w}^*) \leq \epsilon$$

- usually SGD needs more iterations

Convergence guarantees — convex objectives

Many for both GD and SGD on **convex objectives**.

They tell you how many iterations t (in terms of ϵ) needed to achieve

$$F(\boldsymbol{w}^{(t)}) - F(\boldsymbol{w}^*) \leq \epsilon$$

- usually SGD needs more iterations
- but then again each iteration takes less time

Convergence guarantees — nonconvex objectives

Even for *nonconvex objectives*, some guarantees exist: e.g. how many iterations t (in terms of ϵ) needed to achieve

$$\|\nabla F(\boldsymbol{w}^{(t)})\| \leq \epsilon$$

Convergence guarantees — nonconvex objectives

Even for *nonconvex objectives*, some guarantees exist: e.g. how many iterations t (in terms of ϵ) needed to achieve

$$\|\nabla F(\boldsymbol{w}^{(t)})\| \leq \epsilon$$

- that is, how close $\boldsymbol{w}^{(t)}$ is as an **approximate stationary point**

Convergence guarantees — nonconvex objectives

Even for *nonconvex objectives*, some guarantees exist: e.g. how many iterations t (in terms of ϵ) needed to achieve

$$\|\nabla F(\boldsymbol{w}^{(t)})\| \leq \epsilon$$

- that is, how close $\boldsymbol{w}^{(t)}$ is as an **approximate stationary point**
- for convex objectives, stationary point $\Rightarrow$ global minimizer

Convergence guarantees — nonconvex objectives

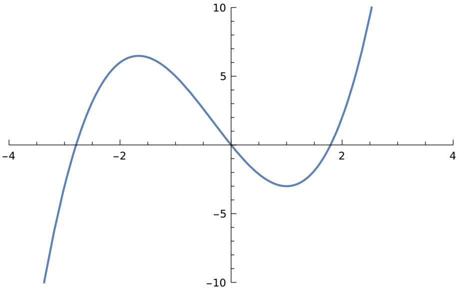
Even for *nonconvex objectives*, some guarantees exist: e.g. how many iterations t (in terms of ϵ) needed to achieve

$$\|\nabla F(\boldsymbol{w}^{(t)})\| \leq \epsilon$$

- that is, how close $\boldsymbol{w}^{(t)}$ is as an **approximate stationary point**
- for convex objectives, stationary point $\Rightarrow$ global minimizer
- for nonconvex objectives, *what does it mean?*

Convergence guarantees — nonconvex objectives

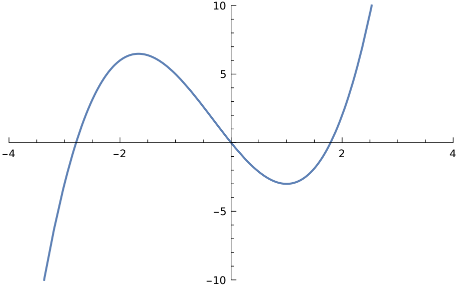
A stationary point can be a **local minimizer**



$$f(w) = w^3 + w^2 - 5w$$

Convergence guarantees — nonconvex objectives

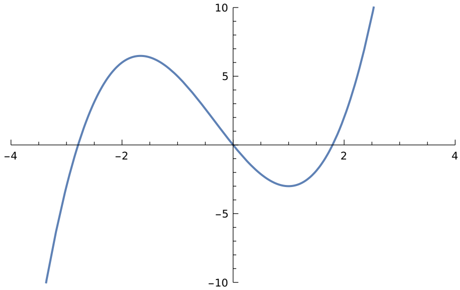
A stationary point can be a **local minimizer** or even a **local/global maximizer**



$$f(w) = w^3 + w^2 - 5w$$

Convergence guarantees — nonconvex objectives

A stationary point can be a **local minimizer** or even a **local/global maximizer** (but the latter is not an issue for GD/SGD).



$$f(w) = w^3 + w^2 - 5w$$

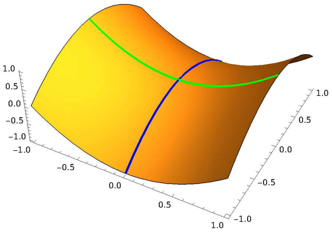
Convergence guarantees — nonconvex objectives

A stationary point can also be *neither a local minimizer nor a local maximizer!*

Convergence guarantees — nonconvex objectives

A stationary point can also be *neither a local minimizer nor a local maximizer!*

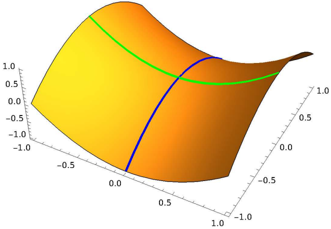
- $f(\boldsymbol{w}) = w_1^2 - w_2^2$



Convergence guarantees — nonconvex objectives

A stationary point can also be *neither a local minimizer nor a local maximizer!*

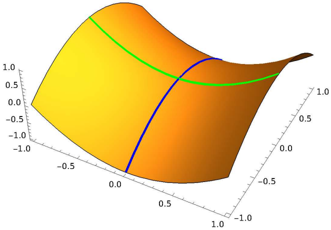
- $f(\boldsymbol{w}) = w_1^2 - w_2^2$
- $\nabla f(\boldsymbol{w}) = (2w_1, -2w_2)$



Convergence guarantees — nonconvex objectives

A stationary point can also be *neither a local minimizer nor a local maximizer!*

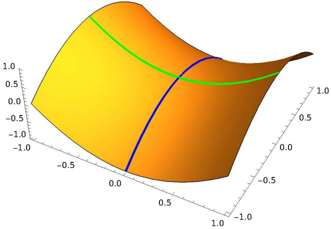
- $f(\boldsymbol{w}) = w_1^2 - w_2^2$
- $\nabla f(\boldsymbol{w}) = (2w_1, -2w_2)$
- so $\boldsymbol{w} = (0, 0)$ is stationary



Convergence guarantees — nonconvex objectives

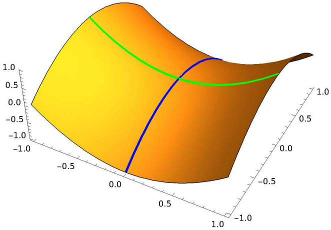
A stationary point can also be *neither a local minimizer nor a local maximizer!*

- $f(\boldsymbol{w}) = w_1^2 - w_2^2$
- $\nabla f(\boldsymbol{w}) = (2w_1, -2w_2)$
- so $\boldsymbol{w} = (0,0)$ is stationary
- local max for blue direction $(w_1 = 0)$



Convergence guarantees — nonconvex objectives

A stationary point can also be *neither a local minimizer nor a local maximizer!*

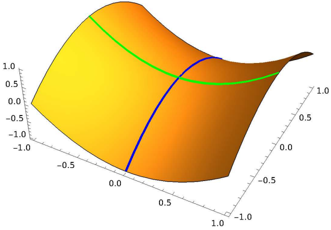


- $f(\boldsymbol{w}) = w_1^2 - w_2^2$
- $\nabla f(\boldsymbol{w}) = (2w_1, -2w_2)$
- so $\boldsymbol{w} = (0,0)$ is stationary
- local max for blue direction $(w_1 = 0)$
- local min for green direction $(w_2 = 0)$

Convergence guarantees — nonconvex objectives

A stationary point can also be *neither a local minimizer nor a local maximizer!* This is called a **saddle point**.

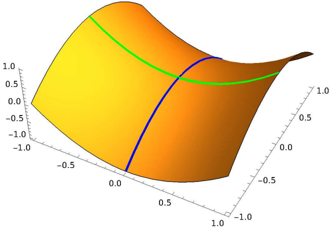
- $f(\boldsymbol{w}) = w_1^2 - w_2^2$
- $\nabla f(\boldsymbol{w}) = (2w_1, -2w_2)$
- so $\boldsymbol{w} = (0, 0)$ is stationary
- local max for blue direction $(w_1 = 0)$
- local min for green direction $(w_2 = 0)$



Convergence guarantees — nonconvex objectives

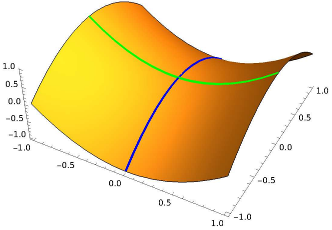
A stationary point can also be *neither a local minimizer nor a local maximizer!* This is called a **saddle point**.

- $f(\boldsymbol{w}) = w_1^2 - w_2^2$
- $\nabla f(\boldsymbol{w}) = (2w_1, -2w_2)$
- so $\boldsymbol{w} = (0,0)$ is stationary
- local max for blue direction $(w_1 = 0)$
- local min for green direction $(w_2 = 0)$
- but GD gets stuck at $(0,0)$ only if initialized along the green direction



Convergence guarantees — nonconvex objectives

A stationary point can also be *neither a local minimizer nor a local maximizer!* This is called a **saddle point**.



- $f(\boldsymbol{w}) = w_1^2 - w_2^2$
- $\nabla f(\boldsymbol{w}) = (2w_1, -2w_2)$
- so $\boldsymbol{w} = (0, 0)$ is stationary
- local max for blue direction $(w_1 = 0)$
- local min for green direction $(w_2 = 0)$
- but GD gets stuck at $(0, 0)$ only if initialized along the green direction
- so not a real issue especially *when initialized randomly*

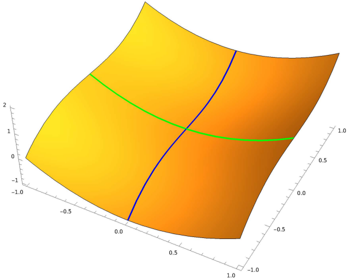
Convergence guarantees — nonconvex objectives

But not all saddle points look like a “saddle”...

Convergence guarantees — nonconvex objectives

But not all saddle points look like a "saddle"...

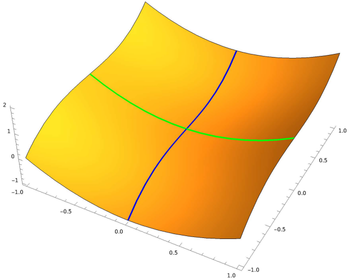
- $f(\boldsymbol{w}) = w_1^2 + w_2^3$



Convergence guarantees — nonconvex objectives

But not all saddle points look like a "saddle"...

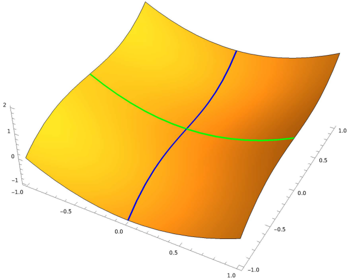
- $f(\boldsymbol{w}) = w_1^2 + w_2^3$
- $\nabla f(\boldsymbol{w}) = (2w_1, 3w_2^2)$



Convergence guarantees — nonconvex objectives

But not all saddle points look like a "saddle"...

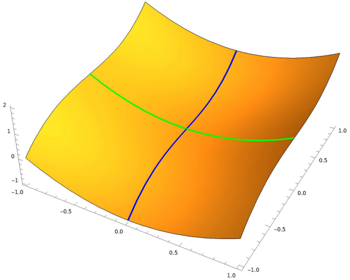
- $f(\boldsymbol{w}) = w_1^2 + w_2^3$
- $\nabla f(\boldsymbol{w}) = (2w_1, 3w_2^2)$
- so $\boldsymbol{w} = (0, 0)$ is stationary



Convergence guarantees — nonconvex objectives

But not all saddle points look like a "saddle"...

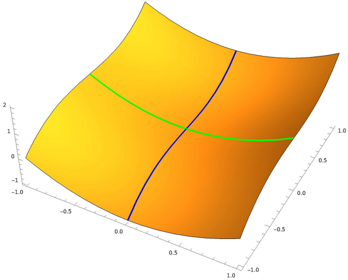
- $f(\boldsymbol{w}) = w_1^2 + w_2^3$
- $\nabla f(\boldsymbol{w}) = (2w_1, 3w_2^2)$
- so $\boldsymbol{w} = (0, 0)$ is stationary
- not local min/max for blue direction ($w_1 = 0$)



Convergence guarantees — nonconvex objectives

But not all saddle points look like a "saddle"...

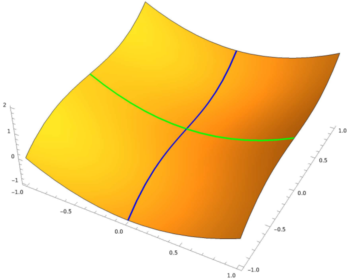
- $f(\boldsymbol{w}) = w_1^2 + w_2^3$
- $\nabla f(\boldsymbol{w}) = (2w_1, 3w_2^2)$
- so $\boldsymbol{w} = (0, 0)$ is stationary
- not local min/max for blue direction $(w_1 = 0)$
- GD gets stuck at $(0, 0)$ for *any initial point with $w_2 \geq 0$ and small η*



Convergence guarantees — nonconvex objectives

But not all saddle points look like a "saddle"...

- $f(\boldsymbol{w}) = w_1^2 + w_2^3$
- $\nabla f(\boldsymbol{w}) = (2w_1, 3w_2^2)$
- so $\boldsymbol{w} = (0, 0)$ is stationary
- not local min/max for blue direction $(w_1 = 0)$
- GD gets stuck at $(0, 0)$ for *any initial point with $w_2 \geq 0$ and small η*



Even worse, distinguishing local min and saddle point is generally *NP-hard*.

Convergence guarantees

Summary:

- GD/SGD converges to a stationary point

Convergence guarantees

Summary:

- GD/SGD converges to a stationary point
- for convex objectives, this is all we need

Convergence guarantees

Summary:

- GD/SGD converges to a stationary point
- for convex objectives, this is all we need
- for nonconvex objectives, can get stuck at local minimizers or "bad" saddle points (random initialization escapes "good" saddle points)

Convergence guarantees

Summary:

- GD/SGD converges to a stationary point
- for convex objectives, this is all we need
- for nonconvex objectives, can get stuck at local minimizers or "bad" saddle points (random initialization escapes "good" saddle points)
- recent research shows that *many problems have no "bad" saddle points or even "bad" local minimizers*

Convergence guarantees

Summary:

- GD/SGD converges to a stationary point
- for convex objectives, this is all we need
- for nonconvex objectives, can get stuck at local minimizers or "bad" saddle points (random initialization escapes "good" saddle points)
- recent research shows that *many problems have no "bad" saddle points or even "bad" local minimizers*
- justify the practical effectiveness of GD/SGD (default method to try)

Second-order methods

Recall the intuition of GD: we look at first-order **Taylor approximation**

$$F(\boldsymbol{w}) \approx F(\boldsymbol{w}^{(t)}) + \nabla F(\boldsymbol{w}^{(t)})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

Second-order methods

Recall the intuition of GD: we look at first-order **Taylor approximation**

$$F(\boldsymbol{w}) \approx F(\boldsymbol{w}^{(t)}) + \nabla F(\boldsymbol{w}^{(t)})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

What if we look at *second-order* Taylor approximation?

$$F(\boldsymbol{w}) \approx F(\boldsymbol{w}^{(t)}) + \nabla F(\boldsymbol{w}^{(t)})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}}\boldsymbol{H}_t(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

Second-order methods

Recall the intuition of GD: we look at first-order **Taylor approximation**

$$F(\boldsymbol{w}) \approx F(\boldsymbol{w}^{(t)}) + \nabla F(\boldsymbol{w}^{(t)})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

What if we look at *second-order* Taylor approximation?

$$F(\boldsymbol{w}) \approx F(\boldsymbol{w}^{(t)}) + \nabla F(\boldsymbol{w}^{(t)})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}}\boldsymbol{H}_t(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

where $\boldsymbol{H}_t = \nabla^2 F(\boldsymbol{w}^{(t)}) \in \mathbb{R}^{\mathsf{D}\times\mathsf{D}}$ is the *Hessian* of F at $\boldsymbol{w}^{(t)}$, i.e.,

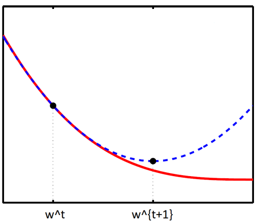
$$H_{t,ij} = \frac{\partial^2 F(\boldsymbol{w})}{\partial w_i \partial w_j}\Big|_{\boldsymbol{w}=\boldsymbol{w}^{(t)}}$$

(think "second derivative" when $D = 1$)

Newton method

If we minimize the second-order approximation (via "complete the square")

$$F(\boldsymbol{w})$$
$$\approx F(\boldsymbol{w}^{(t)}) + \nabla F(\boldsymbol{w}^{(t)})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}}\boldsymbol{H}_t(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$
$$= \frac{1}{2}\left(\boldsymbol{w} - \boldsymbol{w}^{(t)} + \boldsymbol{H}_t^{-1}\nabla F(\boldsymbol{w}^{(t)})\right)^{\mathrm{T}} \boldsymbol{H}_t \left(\boldsymbol{w} - \boldsymbol{w}^{(t)} + \boldsymbol{H}_t^{-1}\nabla F(\boldsymbol{w}^{(t)})\right) + \text{cnt.}$$



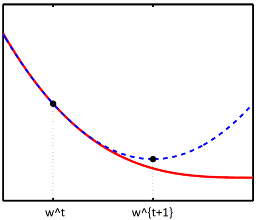
Newton method

If we minimize the second-order approximation (via "complete the square")

$$F(\boldsymbol{w})$$
$$\approx F(\boldsymbol{w}^{(t)}) + \nabla F(\boldsymbol{w}^{(t)})^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}}\boldsymbol{H}_t(\boldsymbol{w} - \boldsymbol{w}^{(t)})$$
$$= \frac{1}{2}\left(\boldsymbol{w} - \boldsymbol{w}^{(t)} + \boldsymbol{H}_t^{-1}\nabla F(\boldsymbol{w}^{(t)})\right)^{\mathrm{T}}\boldsymbol{H}_t\left(\boldsymbol{w} - \boldsymbol{w}^{(t)} + \boldsymbol{H}_t^{-1}\nabla F(\boldsymbol{w}^{(t)})\right) + \text{cnt.}$$

for convex F (so H_t is *positive semidefinite*)
we obtain **Newton method**:

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \boldsymbol{H}_t^{-1}\nabla F(\boldsymbol{w}^{(t)})$$



Comparing GD and Newton

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(GD)}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \boldsymbol{H}_t^{-1} \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(Newton)}$$

Both are iterative optimization procedures,

Comparing GD and Newton

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)}) \quad \text{(GD)}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \boldsymbol{H}_t^{-1} \nabla F(\boldsymbol{w}^{(t)}) \quad \text{(Newton)}$$

Both are iterative optimization procedures, but Newton method

- has no learning rate η (so **no tuning needed!**)

Comparing GD and Newton

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(GD)}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \boldsymbol{H}_t^{-1} \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(Newton)}$$

Both are iterative optimization procedures, but Newton method

- has no learning rate η (so **no tuning needed!**)
- converges **super fast** in terms of #iterations (for convex objectives)

Comparing GD and Newton

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(GD)}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \boldsymbol{H}_t^{-1} \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(Newton)}$$

Both are iterative optimization procedures, but Newton method

- has no learning rate η (so **no tuning needed!**)
- converges **super fast** in terms of #iterations (for convex objectives)
  - e.g. how many iterations needed when applied to a quadratic?

Comparing GD and Newton

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(GD)}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \boldsymbol{H}_t^{-1} \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(Newton)}$$

Both are iterative optimization procedures, but Newton method

- has no learning rate η (so **no tuning needed!**)
- converges **super fast** in terms of #iterations (for convex objectives)
  - e.g. how many iterations needed when applied to a quadratic?
- computing Hessian in each iteration is *very slow* though

Comparing GD and Newton

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(GD)}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \boldsymbol{H}_t^{-1} \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(Newton)}$$

Both are iterative optimization procedures, but Newton method

- has no learning rate η (so **no tuning needed!**)
- converges **super fast** in terms of #iterations (for convex objectives)
  - e.g. how many iterations needed when applied to a quadratic?
- computing Hessian in each iteration is *very slow* though
- does not really make sense for *nonconvex objectives* (but generally Hessian can be useful for escaping saddle points)