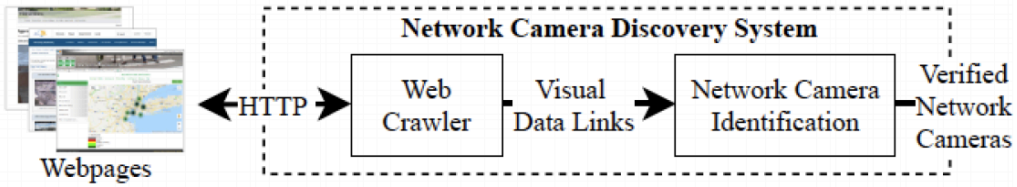
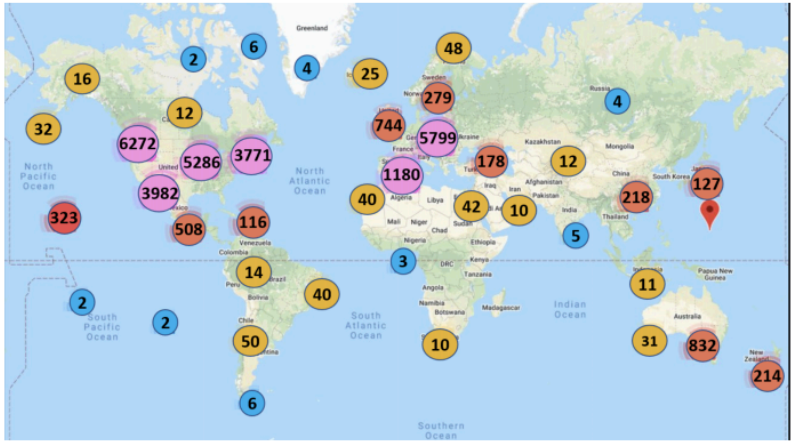
Linear Regression

Liang Zheng
Australian National University
liang.zheng@anu.edu.au

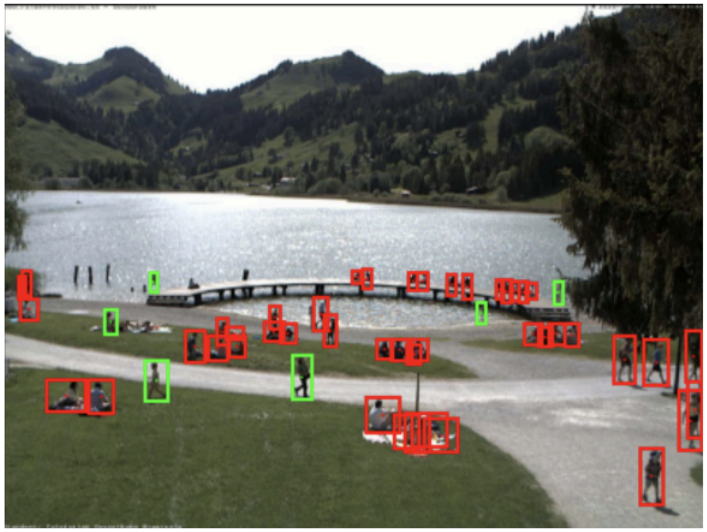
Analyzing Worldwide Social Distancing through Large-Scale Computer Vision, Ghodgaonkar et al., Arxiv 2020



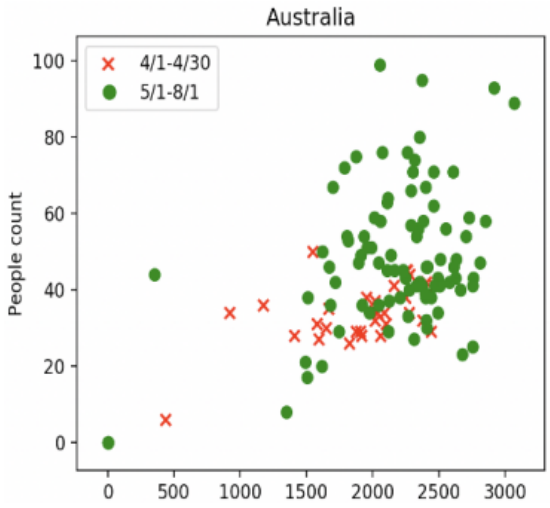
Live data stream crawling

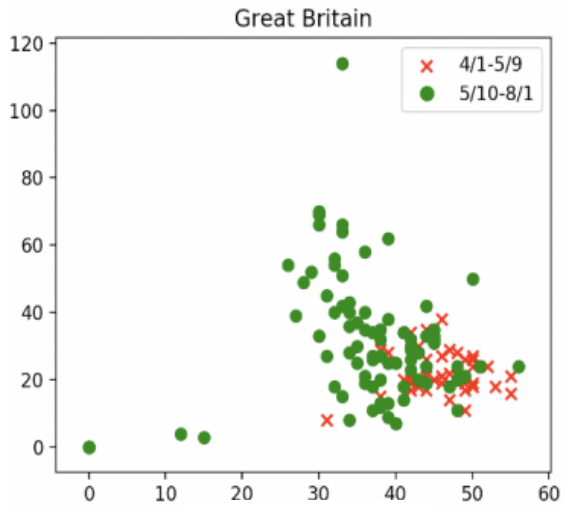


30,261 discovered cameras distributed by geographic location



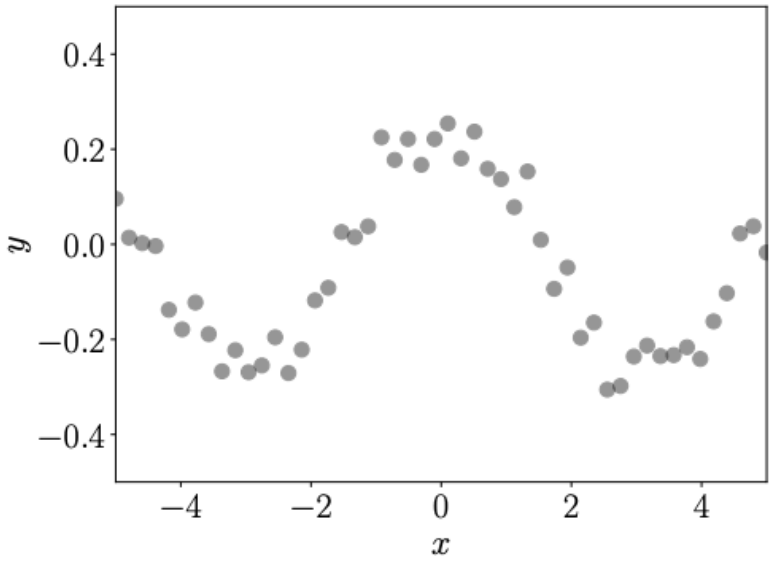
A park in Schwarzsee, Switzerland, on May 22, 2020



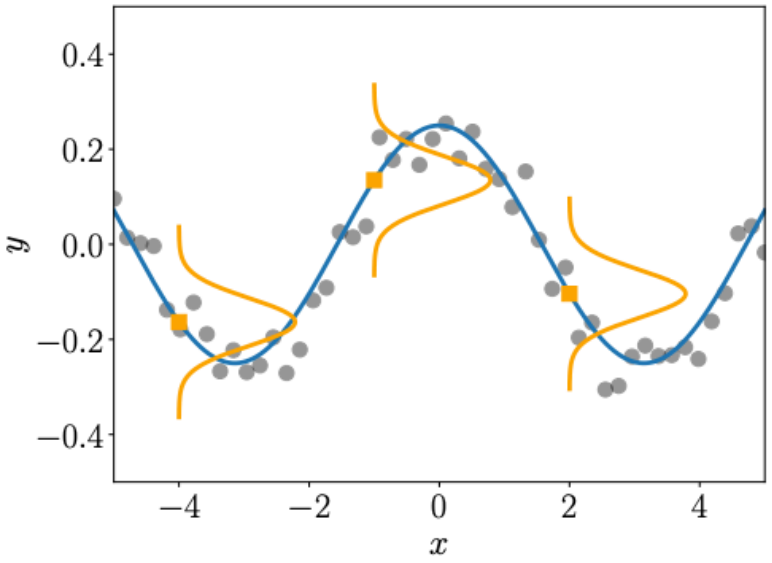


Regression problem

- Regression is a fundamental problem in machine learning



Regression problem: observed noisy function values from which we wish to infer the underlying function that generated the data.



Regression solution: possible function that could have generated the data (blue) with indication of the measurement noise of the function value at the corresponding inputs (orange distributions)

Linear Regression from a statistical point of view

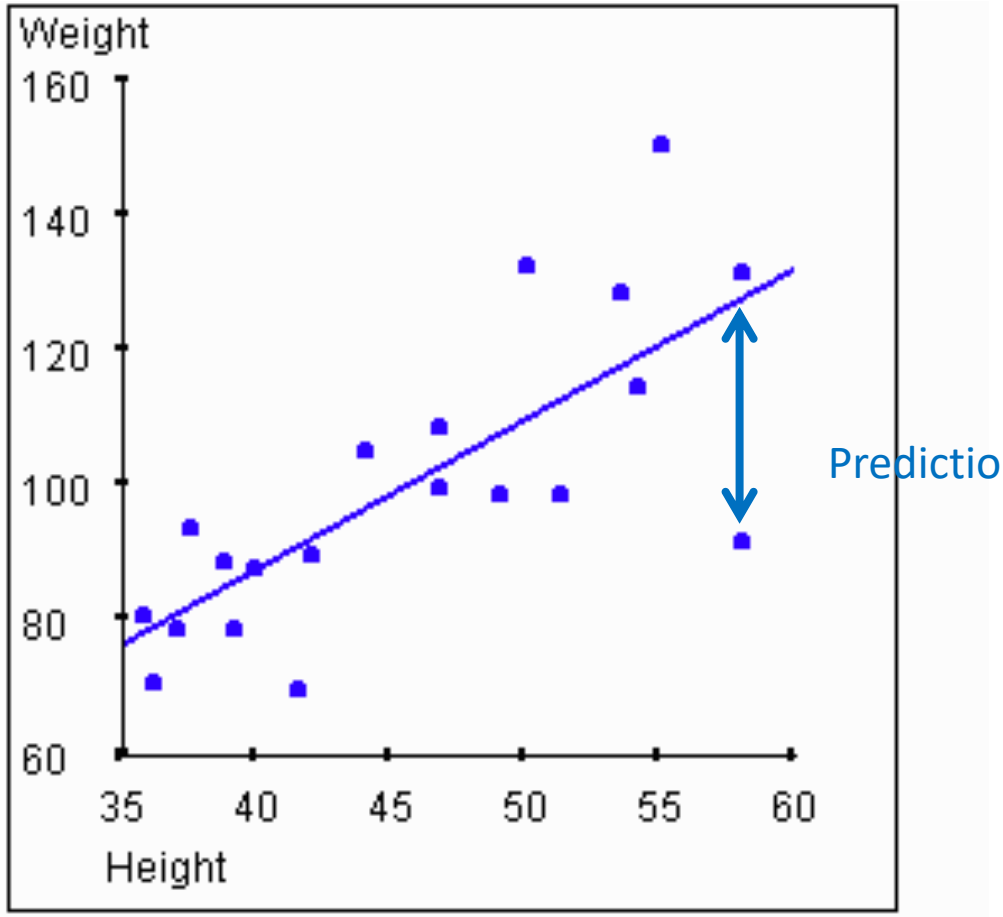
Problem Formulation

- Regression belongs to supervised learning

- **Task.** Find function $f: \mathbb{R}^D \rightarrow \mathbb{R}$ such that $y \approx f(\boldsymbol{x}; \boldsymbol{\theta})$
- **Experience.** Training set $\mathcal{D} \coloneqq \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, consisting of N inputs $\boldsymbol{x}_N \in \mathbb{R}^D$ and corresponding observations/targets $y_n \in \mathbb{R}, n = 1, \ldots, N$
- **Performance.** Estimated empirical risk on the test data

$$\mathbf{R}_{\text{emp}}(f, \boldsymbol{X}_{\text{test}}, \boldsymbol{y}_{\text{test}})$$

Working Example



Loss Function for Training (review)

- Under the i.i.d assumption, the empirical mean is a good estimate of the population mean.
- We can use the empirical mean of the loss on the training data
- Given a training set $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, we use the notation of an example matrix

$$\boldsymbol{X} \coloneqq [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^{\mathrm{T}} \in \mathbb{R}^{N \times D}$$

and a label vector

$$\boldsymbol{y} = [y_1, \ldots, y_N]^{\mathrm{T}} \in \mathbb{R}^N$$

- The average loss is given by

$$\mathbf{R}_{emp}(f, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, \hat{y}_n)$$

where $\hat{y}_n = f(\boldsymbol{x}_n, \boldsymbol{\theta})$. The above equation is called the empirical risk. The learning strategy is called empirical risk minimization.

Least Squares Linear Regression (revision)

- We use the squared loss function

$$\ell(y_n, \hat{y}_n) = (y_n - \hat{y}_n)^2$$

- The empirical risk becomes,

$$\mathbf{R}_{emp}(f, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, \hat{y}_n) = \frac{1}{N}\sum_{n=1}^{N} \big(y_n - f(\boldsymbol{x}_n, \boldsymbol{\theta})\big)^2$$

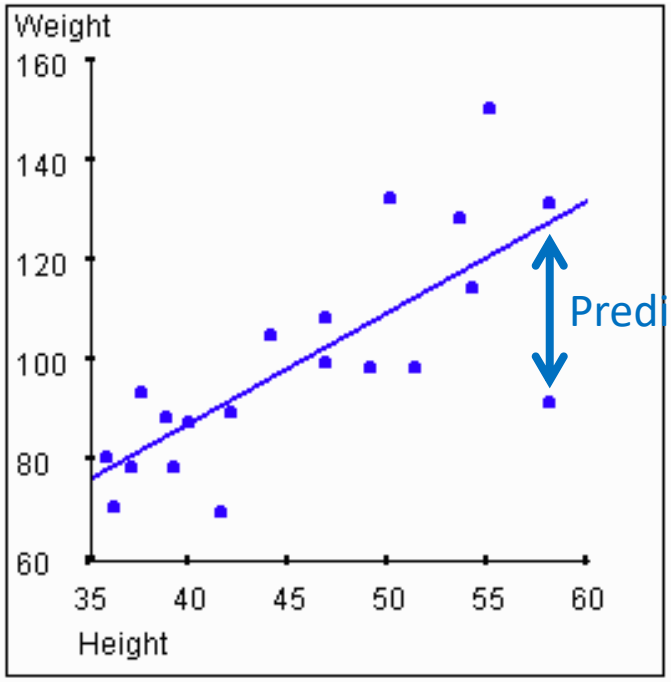
Using the linear predictor $f(\boldsymbol{x}_n, \boldsymbol{\theta}) = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_n$, we obtain the empirical risk as

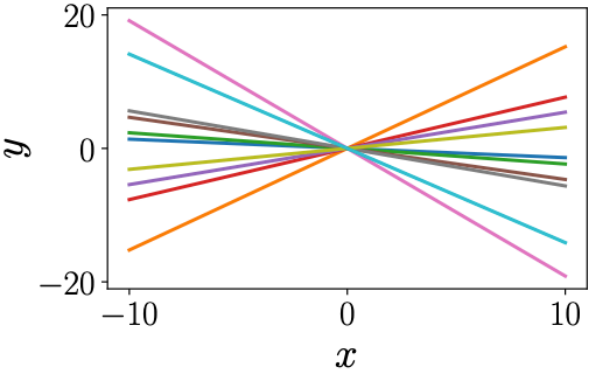
$$\mathbf{R}_{emp}(f, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{N}\sum_{n=1}^{N} \big(y_n - f(\boldsymbol{x}_n, \boldsymbol{\theta})\big)^2$$

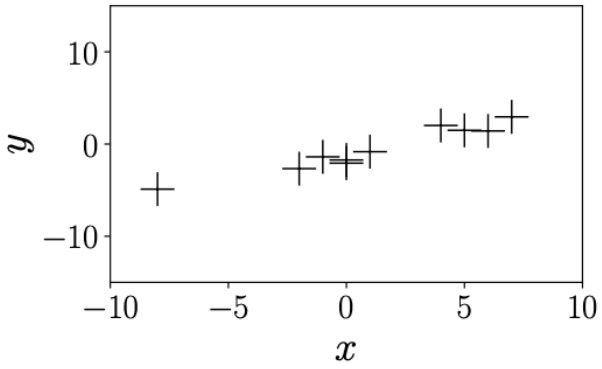
- This equation can be equivalently expressed in matrix form

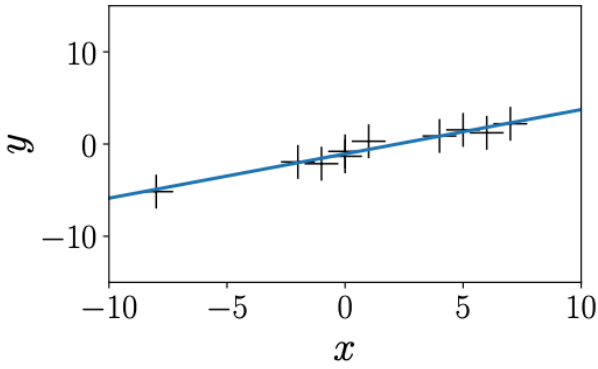
$$\mathcal{L}(\boldsymbol{\theta}) := \mathbf{R}_{emp}(f, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|^2 = \frac{1}{N}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})$$

Working Example









A closed-form analytic solution

- Loss function (mean square error)

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{X\theta}\|^2 = \frac{1}{N}(\boldsymbol{y} - \boldsymbol{X\theta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X\theta})$$

- We calculate the gradient of $\mathcal{L}$ with respect to the parameters $\boldsymbol{\theta}$ as

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} = \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left(\frac{1}{N}(\boldsymbol{y} - \boldsymbol{X\theta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X\theta})\right)$$

$$= \frac{1}{N}\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{X\theta} + \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X\theta}\right)$$

$$= \frac{1}{N}\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - 2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X\theta} + \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X\theta}\right)$$

$$= \frac{1}{N}\left(-2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X} + 2\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right) \in \mathbb{R}^{1\times D}$$

- The minimum is attained when the gradient is zero.

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} = \boldsymbol{0}^{\mathrm{T}} \Leftrightarrow \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}$$

$$\Leftrightarrow \quad \boldsymbol{\theta}^{\mathrm{T}} = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}$$

$$\Leftrightarrow \quad \boldsymbol{\theta} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

Linear Regression from a probability point of view

9.1 Problem formulation

- A linear regression problem with likelihood function

$$p(y|\boldsymbol{x},\boldsymbol{\theta}) = \mathcal{N}\left(y|\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta},\sigma^2\right)$$

$$\Leftrightarrow y = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta} + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0,\sigma^2)$$

- The class of functions described by $y = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta} + \varepsilon$ are straight lines that pass through the origin.
- The likelihood in $p(y|\boldsymbol{x},\boldsymbol{\theta}) = \mathcal{N}\left(y|\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta},\sigma^2\right)$ is the probability density function of y evaluated at $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta}$.
- The only source of uncertainty originates from the observation noise ε (we assume ε is known)

9.2 Parameter estimation

- Given a training set $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ where $\boldsymbol{x}_n \in \mathbb{R}^D$ and corresponding observations/targets $y_n \in \mathbb{R}$
- y_i and y_j are conditionally independent given their inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$
- we use the notation of set of training input

$$\mathcal{X} := \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$$

  and set of corresponding targets

$$\mathcal{Y} := \{y_1, \ldots, y_N\}$$

- The likelihood factorizes

$$p(\mathcal{Y}|\mathcal{X}, \boldsymbol{\theta}) = p(y_1, \ldots, y_N|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{\theta})$$

$$\prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}) = \prod_{n=1}^{N} \mathcal{N}\left(y_n|\boldsymbol{x}_n{}^{\mathrm{T}}\boldsymbol{\theta}, \sigma^2\right)$$

9.2.1 Maximum Likelihood Estimation

- Intuitively, maximizing the likelihood means maximizing the predictive distribution of the training data given the model parameters. We obtain the maximum likelihood parameters as

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p(\mathcal{Y}|\mathcal{X}, \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta})$$

- Instead of maximizing the likelihood directly, we apply the log-transformation to the likelihood function and minimize the negative log-likelihood,

$$-\log p(\mathcal{Y}|\mathcal{X}, \boldsymbol{\theta}) = -\log \prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta})$$

- In the linear regression model $y = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta} + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)$, the likelihood is Gaussian (due to the noise term ε), such that we arrive at

$$\log p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}) = -\frac{1}{2\sigma^2}\left(y_n - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta}\right)^2 + \mathrm{const}$$

Maximum Likelihood Estimation

- Negative log-likelihood:

$$-\log p(\mathcal{Y}|\mathcal{X}, \boldsymbol{\theta}) = -\log \prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta})$$

- We have

$$\log p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}) = -\frac{1}{2\sigma^2}\left(y_n - \boldsymbol{x}_n{}^{\mathrm{T}}\boldsymbol{\theta}\right)^2 + \text{const}$$

- By substitution and discarding const terms, we have

$$\mathcal{L}(\boldsymbol{\theta}) \coloneqq \frac{1}{2\sigma^2}\sum_{n=1}^{N}\left(y_n - \boldsymbol{x}_n{}^{\mathrm{T}}\boldsymbol{\theta}\right)^2 = \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) = \frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|^2$$

where $\boldsymbol{X}$ is the example feature matrix

$$\boldsymbol{X} \coloneqq [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^{\mathrm{T}} \in \mathbb{R}^{N \times D}$$

and $\boldsymbol{y}$ the label vector

$$\boldsymbol{y} = [y_1, \ldots, y_N]^{\mathrm{T}} \in \mathbb{R}^N$$

Linear regression from Stats and Prob views

- From the Stats view:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{X\theta}\|^2 = \frac{1}{N}(\boldsymbol{y} - \boldsymbol{X\theta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X\theta})$$

- From the Prob view:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X\theta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X\theta}) = \frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{X\theta}\|^2$$

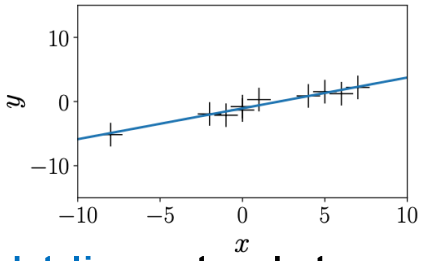
- Both N and σ are known, so these two loss functions are equivalent and have the same closed-form solution

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

Issues.

1. Need $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ to be invertible
   - Feature vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ must span $\mathbb{R}^D$
   - Must have more data than features, $\mathrm{N} \geq D$
   - Use regularization if $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ is not invertible
2. What if $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} \in \mathbb{R}^{d \times d}$ is a large matrix?
   - Takes long time to invert
   - Use stochastic gradient descent if $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ is too large

Linear regression with features



- So far, we have discussed linear regression which fits straight lines to data.
- However straight lines are not sufficiently expressive when fitting more complex data
- Fortunately, “linear regression” only refers to “linear in the parameters”
- we can perform an arbitrary nonlinear transformation $\phi(\boldsymbol{x})$ of the inputs $\boldsymbol{x}$ and then linearly combine the components of this transformation.

$$p(y|\boldsymbol{x},\boldsymbol{\theta}) = \mathcal{N}\left(y\middle|\phi^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{\theta},\sigma^2\right)$$

$$\Leftrightarrow y = \phi^T(\boldsymbol{x})\boldsymbol{\theta} + \epsilon = \sum_{K=0}^{K-1} \theta_k \phi_k(\boldsymbol{x}) + \epsilon$$

where $\phi: \mathbb{R}^D \to \mathbb{R}^K$ is a (nonlinear) transformation of the inputs $\boldsymbol{x}$ and $\phi_k: \mathbb{R}^D \to \mathbb{R}$ is the kth component of the feature vector ϕ. Note that the model parameters $\boldsymbol{\theta}$ still appear only linearly

Example - Polynomial Regression

- We are concerned with a regression problem $y = \phi^{\mathrm{T}}(x)\boldsymbol{\theta} + \epsilon$, where $x \in \mathbb{R}$ and $\boldsymbol{\theta} \in \mathbb{R}^K$. A transformation that is often used in this context is

$$\phi(x) = \begin{bmatrix} \phi_0(x) \\ \phi_1(x) \\ \vdots \\ \phi_{K-1}(x) \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \\ \vdots \\ x^{K-1} \end{bmatrix} \in \mathbb{R}^K$$

- We create a K-dimensional feature from a 1-dimensional input
- With these features, we can model polynomials of degree $\leq K-1$ within the framework of linear regression: A polynomial of degree $K-1$ is

$$f(x) = \sum_{k=0}^{K-1} \theta_k \, x^k = \phi^{\mathrm{T}}(x)\boldsymbol{\theta}$$

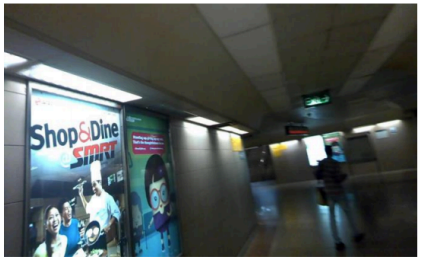
where $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_{K-1}]^{\mathrm{T}} \in \mathbb{R}^K$ contains the (linear) parameters.

Check your understanding

- Linear regression belongs to supervised learning.
- If we use a polynomial feature expression, linear regression is not longer *linear*.
- Linear regression is *linear* in that it uses a linear modeling of the input feature.
- If we have limited data but very high dimensional data, we will likely have overfitting, but it is still possible to use the equation $\boldsymbol{\theta} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$ to calculate the optimal parameters.
- You are running a restaurant and want to use linear regression to estimate your daily income. Your prior knowledge tells you that the income is cyclical with different seasons. Which feature is best for your model?
  - Polynomials of days
  - Cosine function of days
  - Exponential function of days
  - Days

Synthetic-to-Real Unsupervised Domain Adaptation for Scene Text Detection in the Wild. Wu et al, Arxiv 2020

ICDAR2013



ICDAR2015

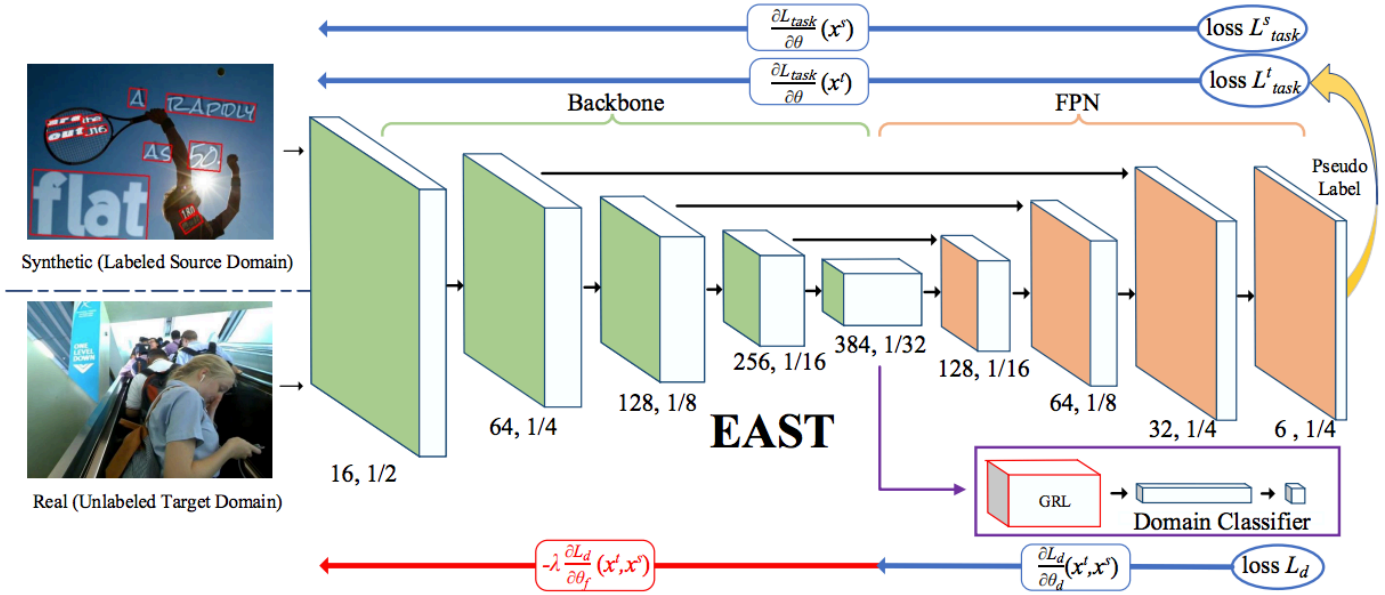
ICDAR2017 MLI

SynthText

VISD

UnrealText

gradient reversal layer (GRL)



Linear regression with features

- We consider training inputs $\boldsymbol{x}_n \in \mathbb{R}^D$ and targets $y_n \in \mathbb{R}, n = 1, \ldots, N$ and define the feature matrix (design matrix) as

$$\boldsymbol{\Phi} := \begin{bmatrix} \phi^{\mathrm{T}}(\boldsymbol{x}_1) \\ \vdots \\ \phi^{\mathrm{T}}(\boldsymbol{x}_N) \end{bmatrix} = \begin{bmatrix} \phi_0(\boldsymbol{x}_1) & \cdots & \phi_{K-1}(\boldsymbol{x}_1) \\ \phi_0(\boldsymbol{x}_2) & \cdots & \phi_{K-1}(\boldsymbol{x}_2) \\ \vdots & & \vdots \\ \phi_0(\boldsymbol{x}_N) & \cdots & \phi_{K-1}(\boldsymbol{x}_N) \end{bmatrix} \in \mathbb{R}^{N \times K}$$

  where $\Phi_{ij} = \phi_j(\boldsymbol{x}_i)$ and $\phi_j : \mathbb{R}^D \to \mathbb{R}$.

- With this feature matrix $\boldsymbol{\Phi}$, the negative log-likelihood for the linear regression model can be written as

$$-\log p(\mathcal{Y} | \boldsymbol{\mathcal{X}}, \boldsymbol{\theta}) = \frac{1}{2\sigma^2} (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^{\mathrm{T}} (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}) + \text{const.}$$

Linear regression with features

- With this feature matrix $\boldsymbol{\Phi}$, the negative log-likelihood for the linear regression model can be written as

$$-\log \mathrm{p}(\mathcal{Y}|\boldsymbol{\mathcal{X}}, \boldsymbol{\theta}) = \frac{1}{2\sigma^2}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^{\mathrm{T}}(\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta}) + \text{const} \quad (1)$$

- Recall we have

$$\mathcal{L}(\boldsymbol{\theta}) \coloneqq \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - \boldsymbol{x}_n{}^T\boldsymbol{\theta})^2 = \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) = \frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|^2 \quad (2)$$

  where $\mathbf{X}$ is the matrix of the data points.

- Comparing (1) and (2), we immediately see we just need to replace $\mathbf{X}$ with $\boldsymbol{\Phi}$.
- The solution to (2) was

$$\boldsymbol{\theta} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

- we arrive immediately at the closed form solution to the Linear regression with features problem

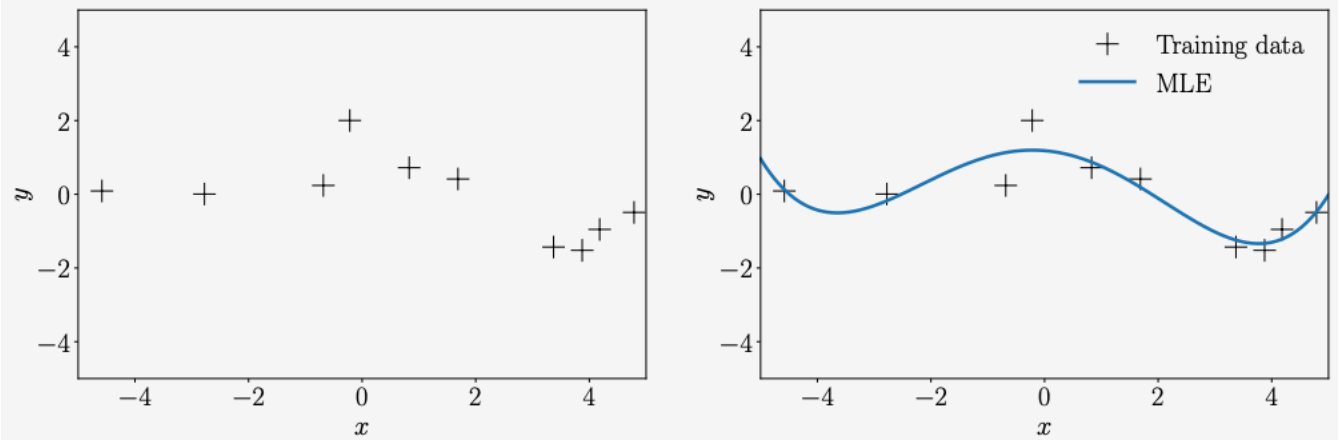
$$\boldsymbol{\theta} = \left(\boldsymbol{\Phi}^{\mathbf{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathbf{T}}\mathrm{y}$$

Example - Feature Matrix for Second-order Polynomials

- For a second-order polynomial and N training points $x_n \in \mathbb{R}, n = 1, \ldots, N$, the feature matrix is

$$\boldsymbol{\Phi} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix}$$

- Another example:



- The dataset consists of $N = 20$ data pairs (x_n, y_n), where $x_n \sim \mathcal{U}[-5,5]$, and $y_n = -\sin(x_n/5) + \cos(x_n) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, 0.2^2)$
- In this figure, we fit a polynomial of degree $K = 4$ using maximum likelihood estimation.

9.2.2 Overfitting in Linear Regression

- Loss function (mean square error)

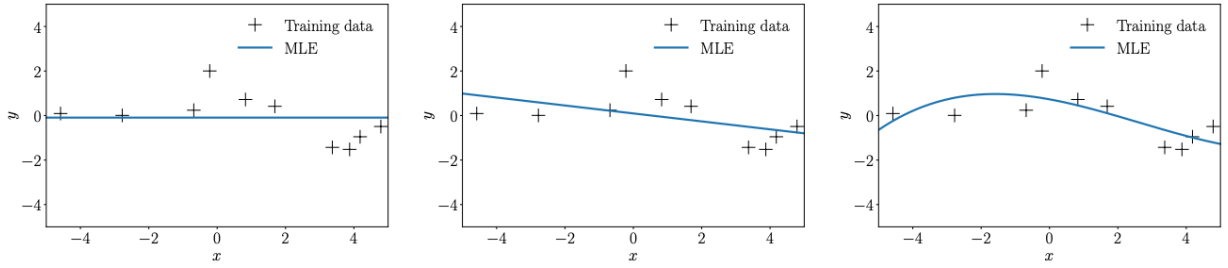
$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{\Phi\theta}\|^2 = \frac{1}{N}\sum_{n=1}^{N}(y_n - \phi^T(\boldsymbol{x}_n)\boldsymbol{\theta})^2$$

- Root mean square error

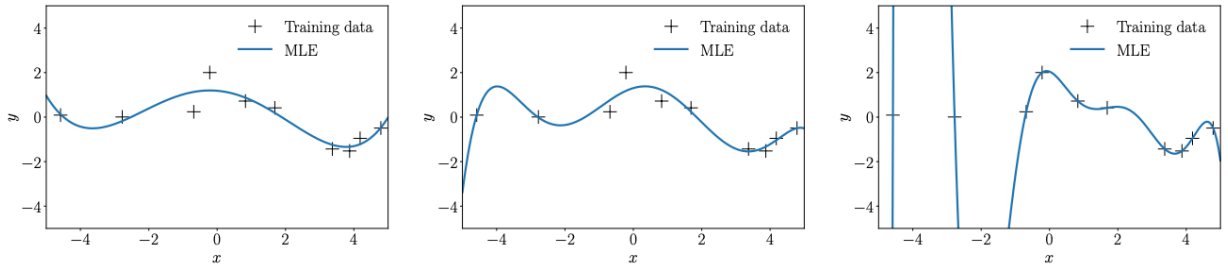
$$\mathcal{L}(\boldsymbol{\theta}) = \sqrt{\frac{1}{N}\|\boldsymbol{y} - \boldsymbol{\Phi\theta}\|^2} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n - \phi^T(\boldsymbol{x}_n)\boldsymbol{\theta})^2}$$

- We have N data pairs and want to fit a polynomial model to the data
- We want to determine the polynomial degree M that yields a low training loss and generalizes well (low test error).

- To determine the best value of M (polynomial degree), we can perform a brute-force search and enumerate all (reasonable) values of M.



(a) $M = 0$ (b) $M = 1$ (c) $M = 3$



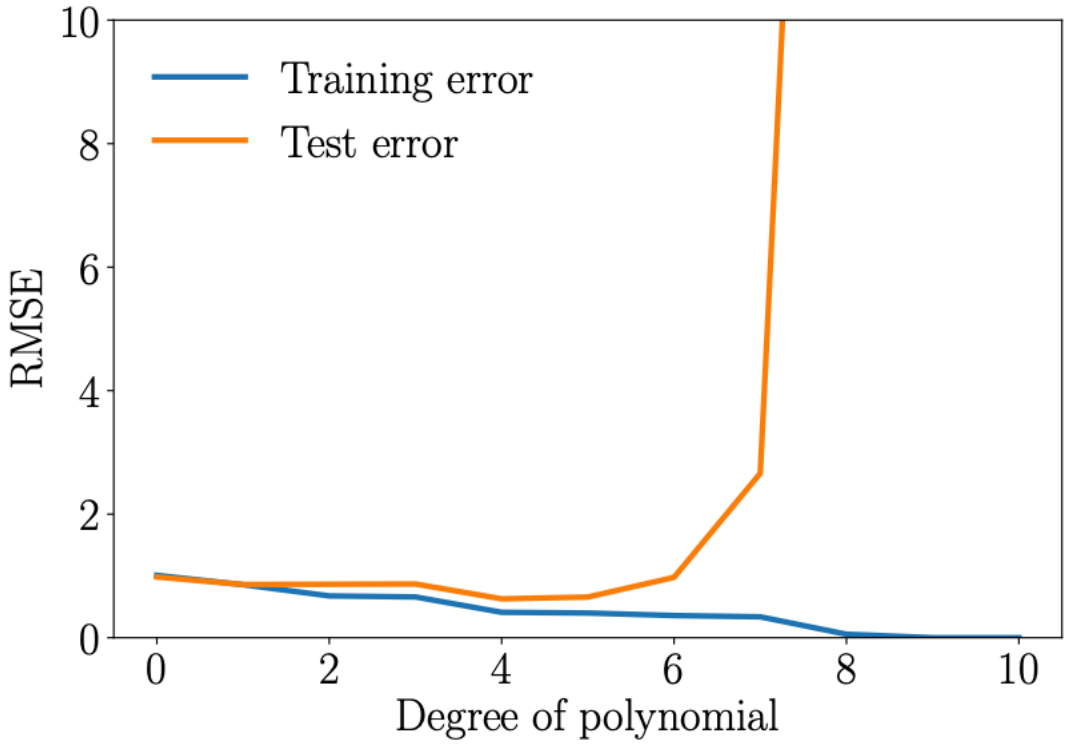
(d) $M = 4$ (e) $M = 6$ (f) $M = 9$

$N = 10$ Data points

- When $M = 0,1$, polynomials fit data poorly
- When $M = 3,4$, the fits look plausible and smoothly interpolate the data
- When $M = 6,9$, polynomials fit data better and better. In the extreme case of $M = N - 1 = 9$, the function will pass through every single data point. These high-degree polynomials oscillate wildly and are a poor representation of the underlying function that generated the data, such that we suffer from overfitting.

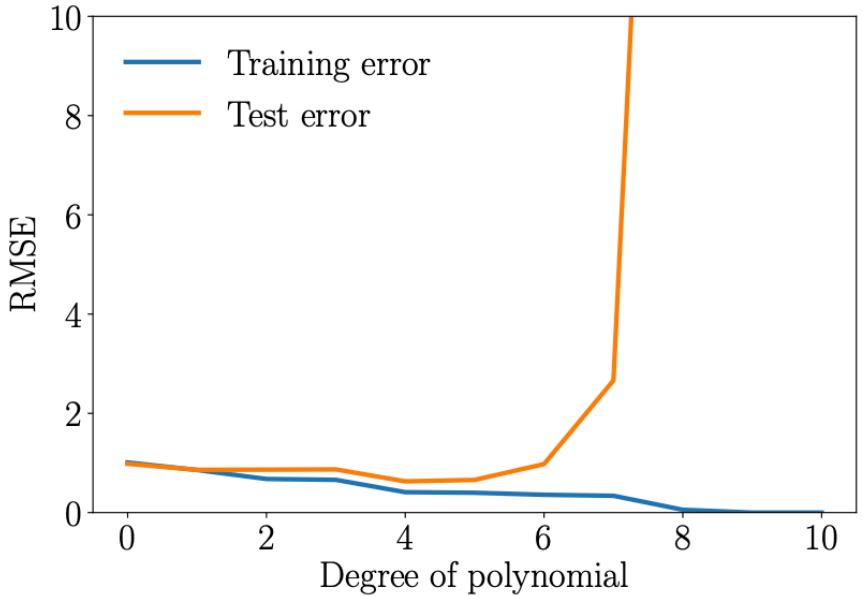
9.2.2 Overfitting in Linear Regression

- Evaluate whether a model generalizes well



A test set of 200 data points generated by the same procedure as the training set
For each choice of M, calculate the RMSE value.

- Evaluate whether a model generalizes well



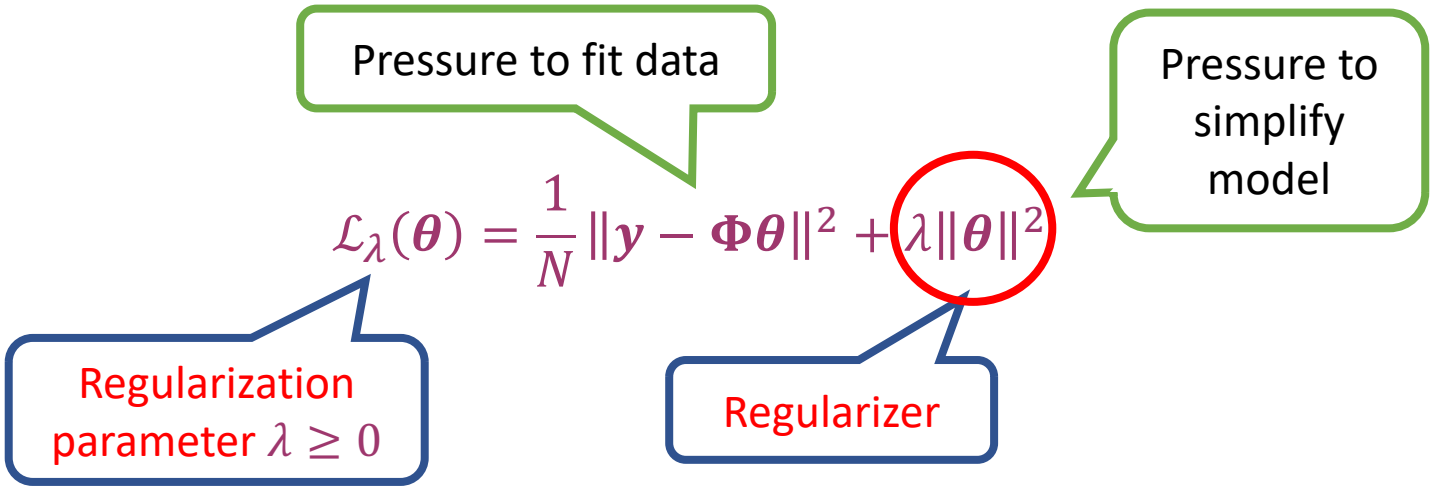
A test set of 200 data points generated by the same procedure as the training set

For each choice of M, calculate the RMSE value.

- Initially the test error decreases
- For fourth-order polynomials, the test error is relatively low and stays relatively constant up to degree 5
- From degree 6 onward the test error increases significantly, and high-order polynomials have very bad generalization properties.
- The training error never increases when the degree of the polynomial increases
- the best generalization (smallest test error) is obtained for $M = 4$.

9.2.4 Regularized Least Squares

- Overfitting occurs because the model is too complex ($\boldsymbol{\theta}$ has too many non-zero entries).
- We want to penalize the amplitude of parameters by regularization
- In regularized least squares, we consider the loss function



- First term: data-fit term; second term: regularizer
- We can use any p-norm $\|\cdot\|_p$
- smaller p leads to sparser solutions, i.e., many parameter values $\theta_d = 0$.

9.2.4 Regularized Least Squares

- Loss function

$$\mathcal{L}_\lambda(\boldsymbol{\theta}) = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|^2 + \lambda\|\boldsymbol{\theta}\|^2$$

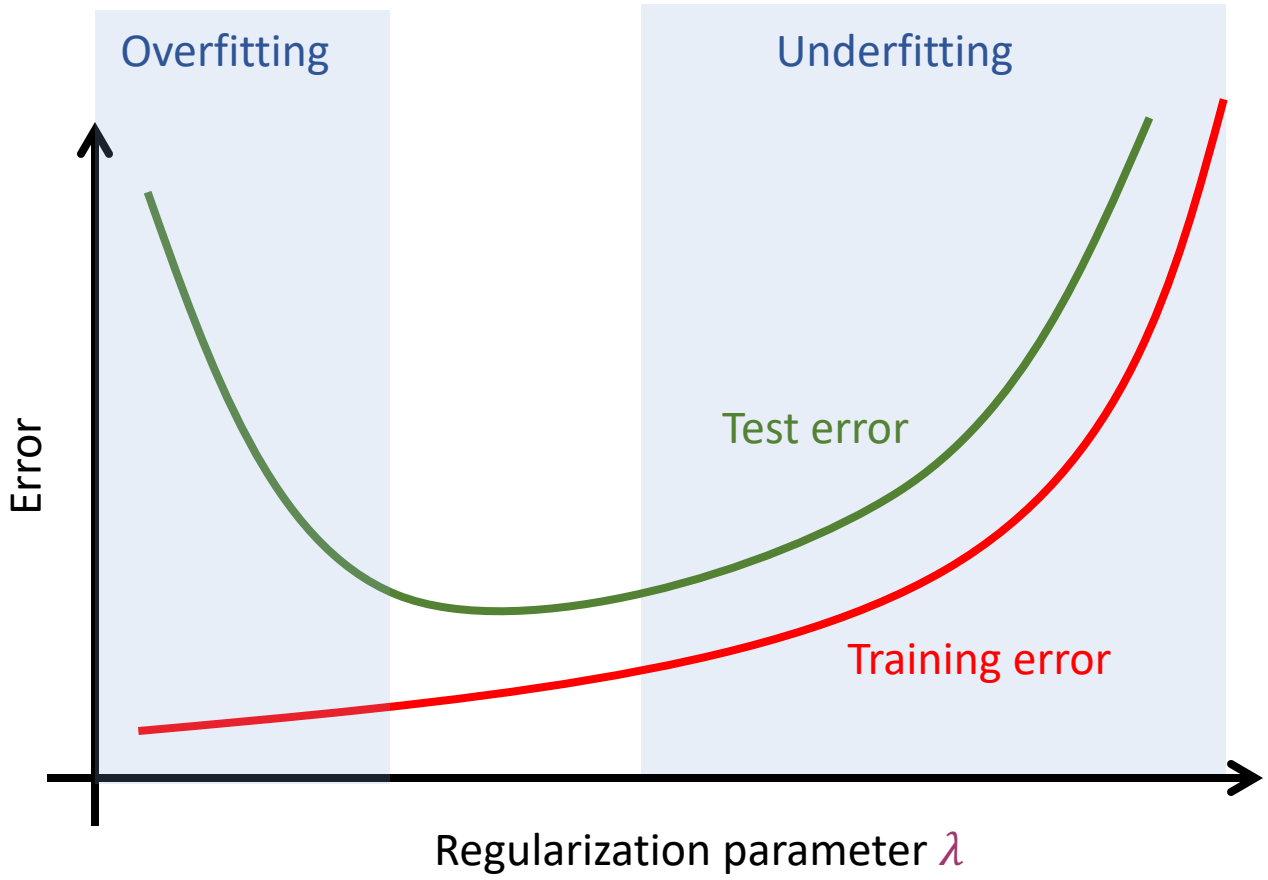
- We calculate the gradient of $\mathcal{L}$ with respect to the parameters $\boldsymbol{\theta}$ as

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} &= \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left(\frac{1}{N}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}\|^2\right)\\ &= \frac{1}{N}\left(-2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X} + 2\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right) + \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}(\lambda\|\boldsymbol{\theta}\|^2)\\ &= \frac{1}{N}\left(-2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X} + 2\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right) + 2\lambda\boldsymbol{\theta}^{\mathrm{T}} \in \mathbb{R}^{1\times D}\end{aligned}$$

- The minimum is attained when the gradient is zero.

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} = \boldsymbol{0}^{\mathrm{T}} \;&\Leftrightarrow\; 2\lambda\boldsymbol{\theta}^{\mathrm{T}} + \frac{1}{N}2\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} = \frac{1}{N}2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\\ &\Leftrightarrow\quad \boldsymbol{\theta}^{\mathrm{T}} = \boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\left(N\lambda\boldsymbol{I} + \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\\ &\Leftrightarrow\quad \boldsymbol{\theta} = \left(N\lambda\boldsymbol{I} + \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}\end{aligned}$$

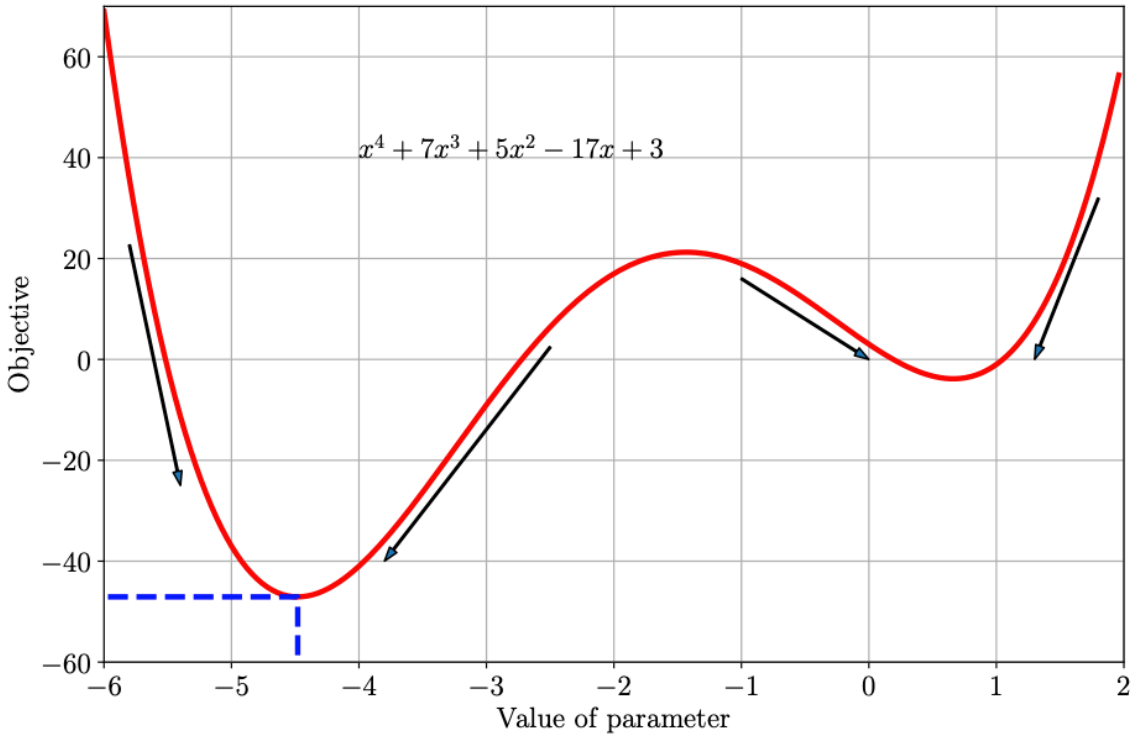
Effect of Regularization



7. Continuous Optimization – analytic solution

- The function below has a global minimum global minimum around $x = -4.5$, where the function value is about -47
- There exists another local minimum around $x = 0.7$
- We can solve for all the stationary points of a function by calculating its derivative and setting it to zero

$$\ell(x) = x^4 + 7x^3 + 5x^2 - 17x + 3$$



7. Continuous Optimization – analytic solution

- Considering the following polynomial

$$\ell(x) = x^4 + 7x^3 + 5x^2 - 17x + 3$$

- We calculate the gradient

$$\frac{d\ell(x)}{dx} = 4x^3 + 21x^2 + 10x - 17$$

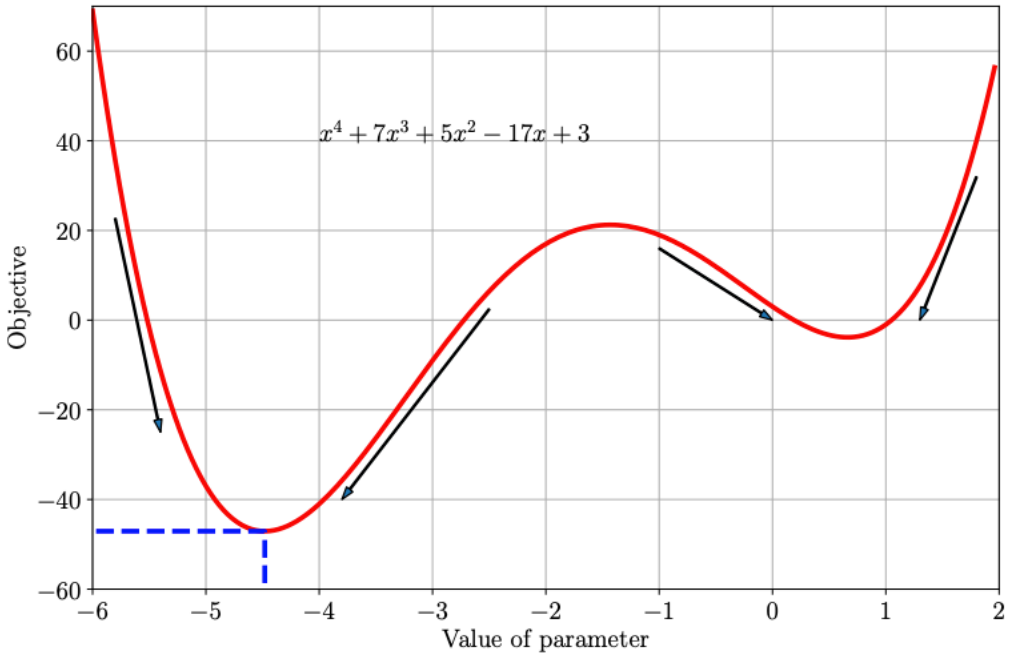
- A cubic equation, it has in general three solutions when set to zero
  - Two are minimums ($x \approx -4.5, 0.7$)
  - One is maximum ($x \approx -1.4$)
- To determine whether it is minimum or maximum, we calculate the second derivative

$$\frac{\mathrm{d}^2\ell(x)}{\mathrm{d}^2x} = 12x^2 + 42x + 10$$

- Substitute our visually estimated values of $x = -4.5, -1.4, 0.7$
- We observe that $x \approx -1.4$ is a maximum, because $\frac{\mathrm{d}^2\ell(x)}{\mathrm{d}^2x} < 0$
- $x \approx 4.5, 0.7$ are two minimums, because $\frac{\mathrm{d}^2\ell(x)}{\mathrm{d}^2x} > 0$

7. Continuous Optimization – Motivation for gradient descent

- In general, we are unable to find analytic solutions
- We need to start at some value e.g., $x_0 = -10$, and follow the gradient
- The gradient points in the direction of steepest ascent of f.
- When we start from $x_0 = -10$, we know we should go right, but don't know how far we should go (step size)
- When starting from $x_0 > -1$, we might find the local minimum.



7.1 Optimization Using Gradient Descent

- Given $f : \mathbb{R}^d \to \mathbb{R}$, we consider the problem of solving for the minimum of f

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

- Gradient descent is a first-order optimization algorithm.
- To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.
- Gradient descent exploits the fact that $f(\boldsymbol{x}_0)$ decreases fastest if one moves from $\boldsymbol{x}_0$ in the direction of the negative gradient $-\left(\nabla f(\boldsymbol{x}_0)\right)^{\mathrm{T}}$ of f at $\boldsymbol{x}_0$.
- Observation: if

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - \gamma \left(\nabla f(\boldsymbol{x}_0)\right)^{\mathrm{T}}$$

- For a small step size $\gamma \geq 0$, then $f(\boldsymbol{x}_1) \leq f(\boldsymbol{x}_0)$

7.1 Optimization Using Gradient Descent

- A simple gradient descent algorithm:
- If we want to find a local optimum $f(\boldsymbol{x}_*)$ of a function $f : \mathbb{R}^n \to \mathbb{R}, \boldsymbol{x} \mapsto f(\boldsymbol{x})$, we start with an initial guess $\boldsymbol{x}_0$ (parameter we want to optimize), and then iterate according to

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \gamma_i \left(\nabla f(\boldsymbol{x}_i)\right)^{\mathrm{T}}$$

- For suitable step-size γ_i, the sequence $f(\boldsymbol{x}_0) \geq f(\boldsymbol{x}_1) \geq \cdots$ converges to a local minimum.

Example – gradient descent

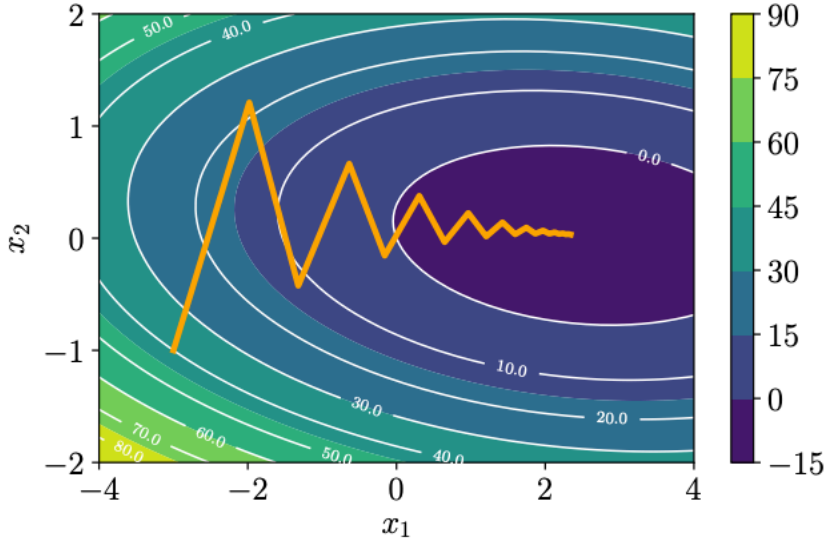
- Consider a quadratic function in two dimensions

$$f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^{\mathrm{T}}\begin{bmatrix} 2 & 1 \\ 1 & 20 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 5 \\ 3 \end{bmatrix}^{\mathrm{T}}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

  with gradient

$$\nabla f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^{\mathrm{T}}\begin{bmatrix} 2 & 1 \\ 1 & 20 \end{bmatrix} - \begin{bmatrix} 5 \\ 3 \end{bmatrix}^{\mathrm{T}}$$

- Starting at the initial location $\boldsymbol{x}_0 = [-3, -1]^{\mathrm{T}}$, we apply gradient descent (iteratively) to obtain a sequence of estimates that converge to the minimum value
- The gradient of $\boldsymbol{x}_0$ points north and east
- leading to $\boldsymbol{x}_1 = [-1.98, 1.21]^{\mathrm{T}}$
- We can then have $\boldsymbol{x}_2 = [-1.32, -0.42]^{\mathrm{T}}$



Example - Solving a Linear Equation System

- When we solve linear equations of the form $\boldsymbol{Ax} = \boldsymbol{b}$, in practice we aim to approximately find $\boldsymbol{x}_*$ that minimizes the squares error

$$\mathcal{L}(\boldsymbol{x}) = \|\boldsymbol{Ax} - \boldsymbol{b}\|^2 = (\boldsymbol{Ax} - \boldsymbol{b})^{\mathrm{T}}(\boldsymbol{Ax} - \boldsymbol{b})$$

- The gradient of $\mathcal{L}(\boldsymbol{x})$ with respect to $\boldsymbol{x}$ is

$$\nabla_{\boldsymbol{x}} = -2\boldsymbol{b}^{\mathrm{T}}\boldsymbol{A} + 2\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$$

- We can use this gradient directly in a gradient descent algorithm

7.1.3 Stochastic Gradient Descent

- Drawback of gradient descent
- Computing the gradient can be very time consuming, why?
- Given N data points $1,2,\dots,N$, the loss function is a sum of losses $\mathcal{L}_n$ incurred by each example n. That is,

$$\mathcal{L}_N(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{\theta})$$

- Example (regression)

$$\mathcal{L}_N(\boldsymbol{\theta}) = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{\Phi\theta}\|^2 = \frac{1}{N}\sum_{n=1}^{N}(y_n - \phi^T(\boldsymbol{x}_n)\boldsymbol{\theta})^2$$

where $\boldsymbol{x}_n \in \mathbb{R}^D$ are the training samples, y_n are the labels, and $\boldsymbol{\theta}$ are the parameters of the linear regression model we want to optimize.

- In standard gradient descent, optimization is performed using the full training set, by updating the vector of parameters according to

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \gamma_i\left(\nabla\mathcal{L}(\boldsymbol{\theta}_i)\right)^{\mathrm{T}} = \boldsymbol{\theta}_i - \gamma_i\frac{1}{N}\sum_{n=1}^{N}\nabla\mathcal{L}_n(\boldsymbol{\theta})^T$$

7.1.3 Stochastic Gradient Descent

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \gamma_i \left(\nabla \mathcal{L}(\boldsymbol{\theta}_i)\right)^{\mathrm{T}} = \boldsymbol{\theta}_i - \gamma_i \frac{1}{N} \sum_{n=1}^{N} \nabla \mathcal{L}_n(\boldsymbol{\theta})^T$$

- Evaluating the sum gradient may require expensive evaluations of the gradients from all individual functions $\mathcal{L}_n$
- Stochastic Gradient Descent
- Estimate the gradient by averaging over a smaller minibatch (subset of the training data).

$$\nabla \mathcal{L}(\boldsymbol{\theta}) \approx \nabla \mathcal{L}_M(\boldsymbol{\theta}) = \frac{1}{M} \sum_{n \in \mathcal{B}_M} \nabla \mathcal{L}_n(\boldsymbol{\theta})^T$$

where $M \ll N$, and $\mathcal{B}_M$ is a subset of the permutated indices of the samples in the minibatch. $|\mathcal{B}_M| = M$.

- For example, the whole dataset has $N = 4$ samples indexed with $n = 1,2,3,4$. The minibatch contains $M = 2$ samples. $\mathcal{B}_M$ could be $\{1,3\},\{2,4\},\ldots$

Stochastic Gradient Descent

1. Initialize $\boldsymbol{\theta}$ randomly.
2. Select minibatch $\mathcal{B}_M$ of data from the training set at random.

$$\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} - \gamma_i \nabla \mathcal{L}_M(\boldsymbol{\theta})$$

3. Repeat Step (2) until convergence.

Gradient Descent

1. Initialize $\boldsymbol{\theta}$ randomly.
2. Update (use all the training set calculate the gradient)

$$\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} - \gamma_i \nabla \mathcal{L}_N(\boldsymbol{\theta})$$

3. Repeat Step (2) until convergence.

7.1.1 Step-size

- Choosing a good step-size (learning rate) is important in gradient descent
- If the step-size is too small, gradient descent can be slow
- If the step-size is chosen too large, gradient descent can overshoot, fail to converge, or even diverge

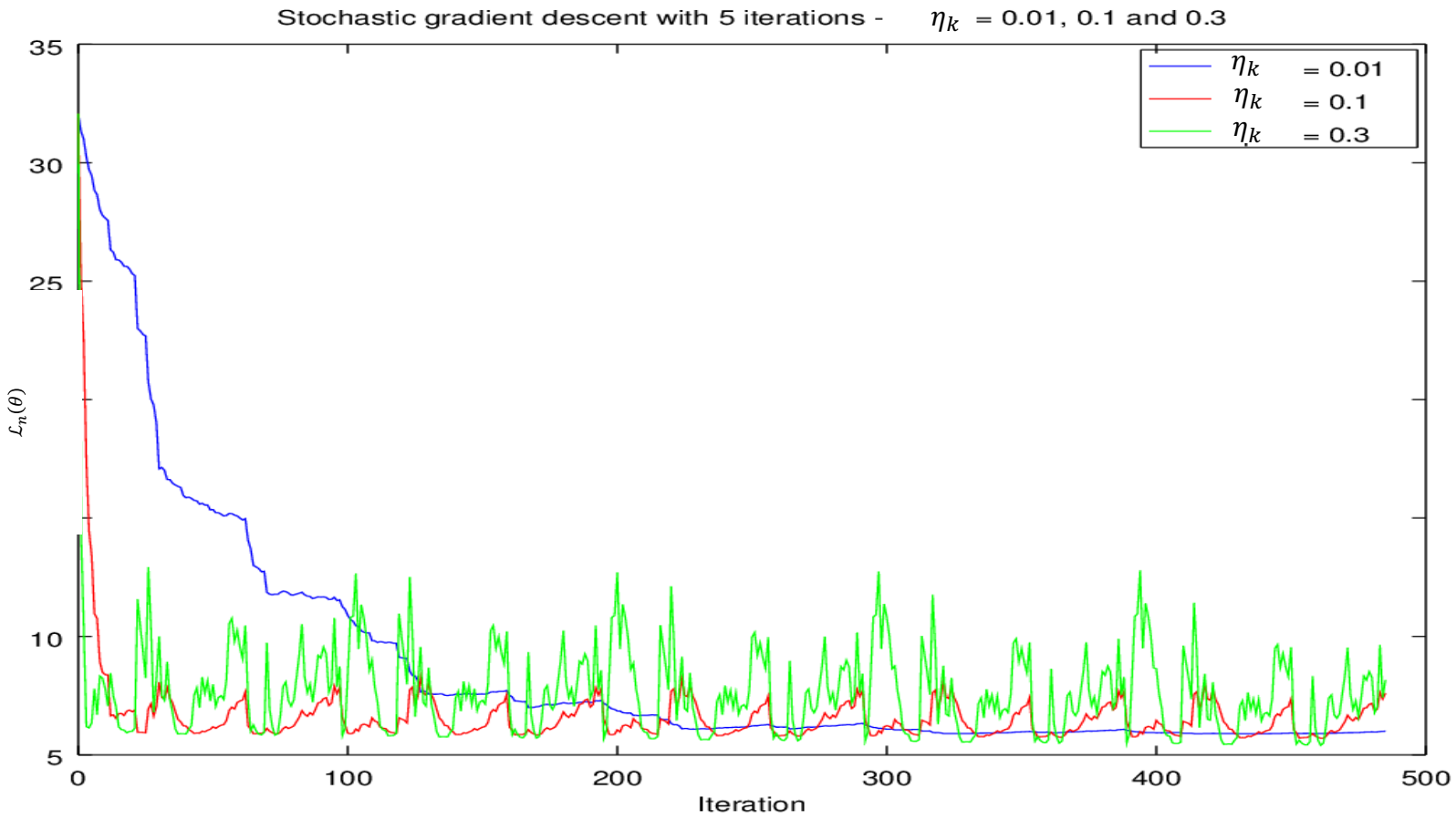
- There are several heuristics to adapt the step size
- When the function value increases after a gradient step, the step-size was too large. Undo the step and decrease the step-size
- When the function value decreases the step could have been larger. Try to increase the step-size.
- Typically, we choose a learning rate that starts big and ends small, e.g. $\gamma_i = 1/(i+1)$

7.1.1 Step-size

- There has been quite a few papers studying the step size/learning rate.
- [1] Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour
- [2] Large Batch Training of Convolutional Networks
- [3] A Closer Look at Deep Learning Heuristics: Learning Rate Restarts, Warmup and Distillation
- [4] Deep Residual Learning for Image Recognition

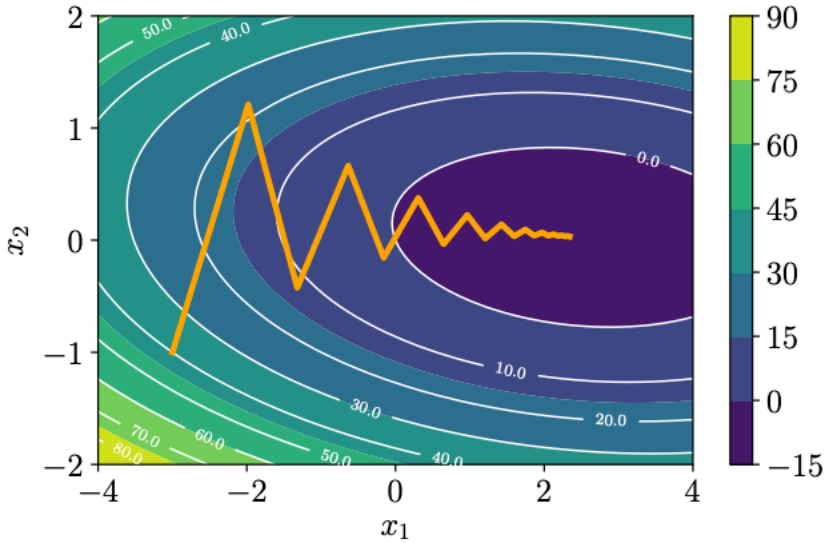
- A recent practice in deep learning is to start training with a small learning rate, switch to a large learning rate, and then decrease it gradually.

Impact of step size



7.1.2 Gradient Descent With Momentum

- The convergence of gradient descent may be very slow if the curvature of the optimization surface is such that there are regions that are poorly scaled
- The proposed method to improve convergence is to give gradient descent some memory
- Gradient descent with momentum is a method that introduces an additional term to remember what happened in the previous iteration.
- This memory dampens oscillations and smooths out the gradient updates
- The idea is to have a gradient update with memory to implement a moving average



7.1.2 Gradient Descent With Momentum

- The momentum-based method remembers the update $\Delta \boldsymbol{x}_i$ at each iteration i and determines the next update as a linear combination of the current and previous gradients

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \gamma_i \left(\nabla f(\boldsymbol{x}_i)\right)^{\mathrm{T}} + \alpha \Delta \boldsymbol{x}_i$$
$$\Delta \boldsymbol{x}_i = \boldsymbol{x}_i - \boldsymbol{x}_{i-1} = \gamma_{i-1} \left(\nabla f(\boldsymbol{x}_{i-1})\right)^{\mathrm{T}}$$

- Where $\alpha \in [-1,1]$.
- Sometimes we only know the gradient approximately.
- In such cases, the momentum term is useful since it averages out different noisy estimates of the gradient.

Linear regression with gradient descent

- Loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{X\theta}\|^2 = \frac{1}{N}(\boldsymbol{y} - \boldsymbol{X\theta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X\theta})$$

- Gradient

$$\nabla\mathcal{L}_{\boldsymbol{N}} = \frac{1}{N}\left(-2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X} + 2\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)$$

- Gradient descent

$$\theta \longleftarrow \theta - \gamma_i\left[\frac{2}{N}\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} - \frac{2}{N}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\right]$$

- Stochastic gradient descent

$$\theta \longleftarrow \theta - \gamma_i\left[\frac{2}{M}\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} - \frac{2}{M}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\right]$$

- Why not use the exact solution? $\boldsymbol{\theta} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}$

  $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} \in \mathbb{R}^{D\times D}$ could be a large matrix

  - Takes long time to invert

Regularized Linear regression with gradient descent

- Loss function

$$\mathcal{L}_\lambda(\boldsymbol{\theta}) = \frac{1}{N}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|^2 + \lambda\|\boldsymbol{\theta}\|^2$$

- Gradient

$$\nabla\mathcal{L}_{\boldsymbol{N}} = \frac{1}{N}\left(-2\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X} + 2\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right) + 2\lambda\boldsymbol{\theta}^{\mathrm{T}}$$

- Gradient descent

$$\boldsymbol{\theta} \longleftarrow (1 - 2\gamma_i\lambda)\boldsymbol{\theta} - \gamma_i\left[\frac{2}{N}\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} - \frac{2}{N}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\right]$$

- Stochastic gradient descent

$$\boldsymbol{\theta} \longleftarrow (1 - 2\gamma_i\lambda)\boldsymbol{\theta} - \gamma_i\left[\frac{2}{M}\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} - \frac{2}{M}\boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}\right]$$

Check your understanding

- When you increase the step size, the training time will be shortened.
- The step size is a hyperparameter.
- “Using a small step size all the time” vs “First use a big step size, then use a small step size”: both methods give you similar optimization results, when starting from the same random seed.
- We can use gradient descent to solve k-means.
- SGD gives you a more precise optimization result than standard GD.
- In both GD and SGD, the loss function will decrease with each iteration.
- In linear regression, the training loss $\frac{1}{N}\|\boldsymbol{y}-\boldsymbol{\Phi\theta}\|^2$ is usually greater when using regularization than not using regularization.
- In $\mathcal{L}_\lambda(\boldsymbol{\theta})=\frac{1}{N}\|\boldsymbol{y}-\boldsymbol{\Phi\theta}\|^2+\lambda\|\boldsymbol{\theta}\|^2$, we can instead put λ in front of $\frac{1}{N}\|\boldsymbol{y}-\boldsymbol{\Phi\theta}\|^2$, which will give us similar optimization results.