Essential Math for Machine Learning

DEEP LEARNING 2023

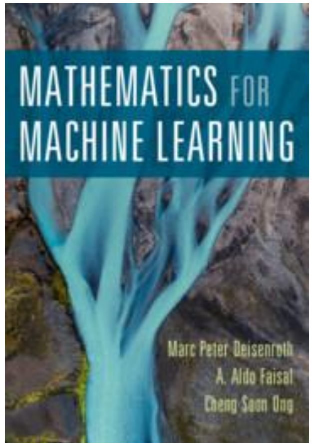
E. FATEMIZADEH

Introduction

- Need for Mathematics
  - Linear Algebra
  - Analytic Geometry
  - Matrix Decompositions
  - Vector Calculus
  - Probability and Distributions
  - Continuous Optimization

References

- Mathematics for Machine Learning, https://mml-book.github.io/
- Matrix Cookbook, https://www2.imm.dtu.dk/pubdb/pubs/3274-full.html



The Matrix Cookbook

[http://matrixcookbook.com]

Kaare Brandt Petersen
Michael Syskind Pedersen

VERSION: NOVEMBER 15, 2012

Linear Algebra

Contents

- Definitions
- Norms and Smoothed Norms
- Matrix Decomposition
- System of Linear Equations
- Useful Gradient

Matrix and Vectors

- General Definition:
  - Matrix: $A = [a_{ij}]_{m\times n}, \quad A \in \mathbb{R}^{m\times n}$
  - Transpose: $A^T = [a_{ji}]_{n\times m}, \quad A^T \in \mathbb{R}^{n\times m}$
  - Hermitian: $A^H = [a_{ji}^*]_{n\times m}, \quad A^H \in \mathbb{R}^{n\times m}$
  - Trace: $Tr(A) = \sum_{i=1}^{n} a_{ii}\,, \; A \in \mathbb{R}^{n\times n}$
  - Matrix Rank, $rank(A)$:
    - The number of linearly independent columns equals the number of linearly independent rows.

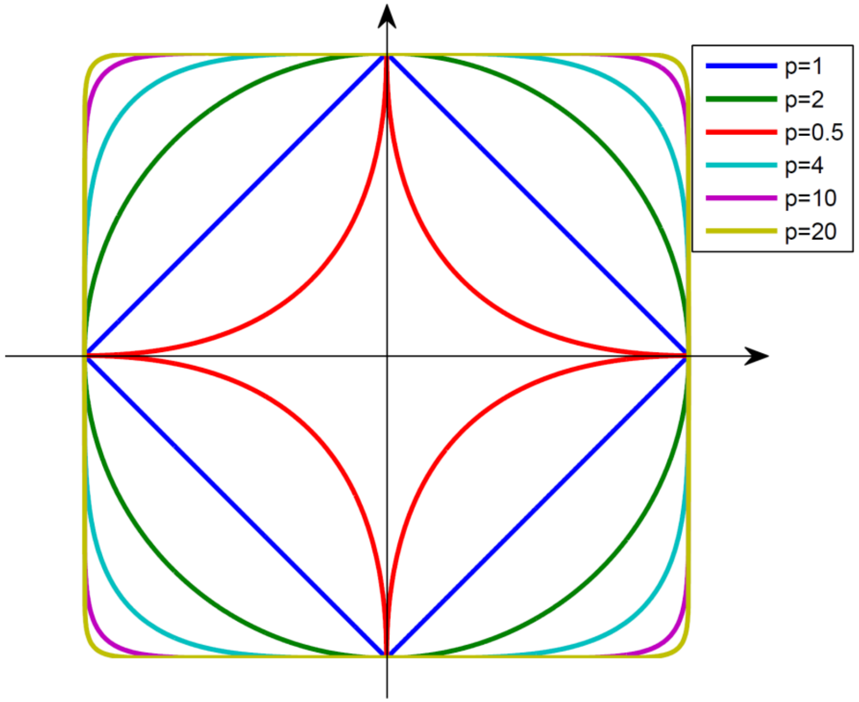
Matrix Norm

- Definition
- $\|A\|_1 = \underset{1\le j\le n}{Max} \sum_{i=1}^{m} \left|a_{ij}\right|$
- $\|A\|_\infty = \underset{1\le i\le m}{Max} \sum_{j=1}^{n} \left|a_{ij}\right|$
- $\|A\|_2^2 = \lambda_{max}(A^H A) = \sigma_{max}^2,$
  - λ_{max}: Largest eigenvalues of $A^H A$
  - σ_{max}: Largest singular value of A
- $\|A\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} \left|a_{ij}\right|^2 = Trace(A^H A),$ Frobenius Norm

Vector Norm

- L_p Norm Definitions:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$$



- L_2 Norm: $\|x\|_2 = (\sum_{i=1}^{n} |x_i|^2)^{\frac{1}{2}}$
- L_1 Norm: $\|x\|_1 = \sum_{i=1}^{n} |x_i|$
- L_0 Norm: $\|x\|_0$: # of non−zero entry
- L_∞ Norm: $\|x\|_\infty$: $max\{|x_i|\}_{i=1}^{n}$

Smoothed Norm

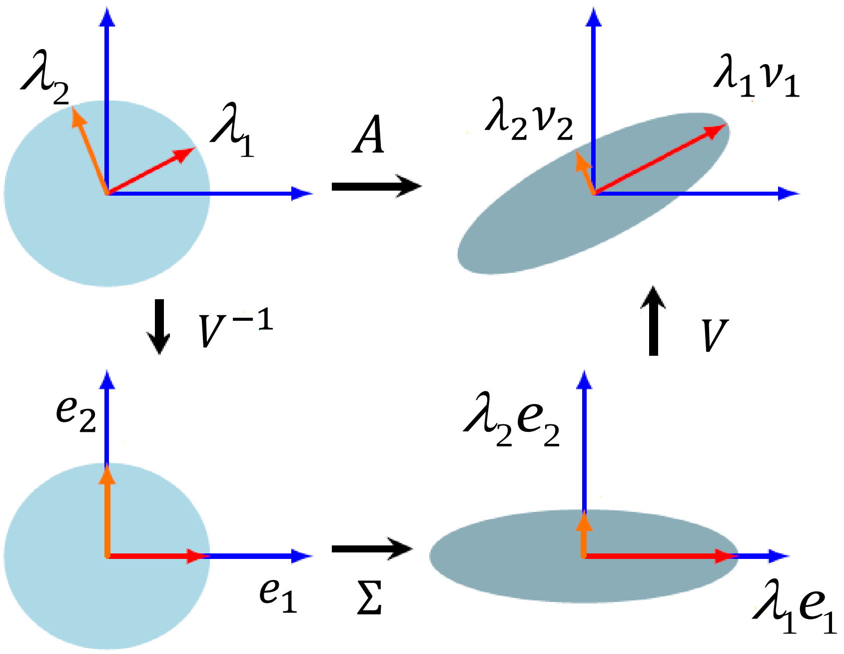
- Smooth L_0 Norm
  - $SL_0^{(1)}(x,\sigma) = n - \sum_{i=1}^{n} exp\left(-\frac{x_i^2}{2\varepsilon^2}\right),\ small\ \varepsilon$
  - $SL_0^{(2)}(x,\varepsilon) = \sum_{i=1}^{n} sin\left(arctan\left(\frac{|x_i|}{\varepsilon}\right)\right),\ small\ \varepsilon$
  - $SL_0^{(3)}(x,\varepsilon) = \sum_{i=1}^{n} \frac{x_i^2}{x_i^2+\varepsilon^2},\ small\ \varepsilon$
- Smooth L_∞ Norm
  - $SL_\infty^{(1)}(x,p) = \frac{ln\left(\sum_{i=1}^{n} exp(p|x_i|)\right)}{p},\ large\ positive\ p$
  - $SL_\infty^{(2)}(x,p) = \frac{\sum_{i=1}^{n}|x_i| exp(p|x_i|)}{\sum_{i=1}^{n} exp(p|x_i|)},\ large\ positive\ p$
  - $SL_\infty^{(3)}(x,p) = \left(\sum_{i=1}^{n}|x_i|^p\right)^{\frac{1}{p}},\ large\ positive\ p$

Matrix Decomposition

- Eigendecomposition of square matrix $A \in \mathbb{R}^{n\times n}$ (Eigenvalues and Eigenvectors)
- $Av = \lambda v \Rightarrow (A - \lambda I)v = 0 \Rightarrow \{\lambda_i\}_{i=1}^{n},\ \{v_i\}_{i=1}^{n}$
- $V = [v_1|v_2|\cdots|v_n], \quad \sum = diag(\lambda_1,\ \lambda_2,\ \cdots, \lambda_n) \Rightarrow AV = V\sum$
- $AV = V\sum \Longrightarrow A = V\sum V^{-1},\ \sum = V^{-1}AV$ (Diagonalization)
- $trace(A) = \sum_{i=1}^{n} \lambda_i$
- $\det(A) = \prod_{i=1}^{n} \lambda_i$

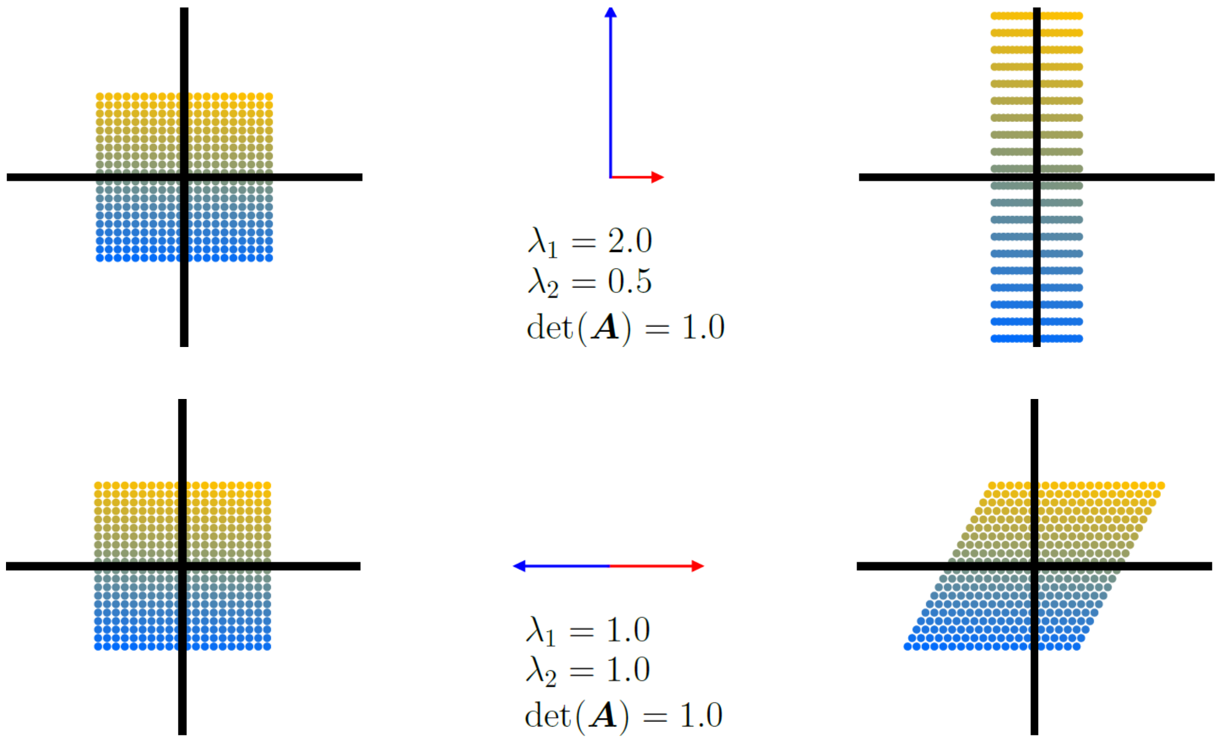
Eigendecomposition

- Eigen Decomposition:
- $A\color{red}{x} = V\Sigma V^{-1}\color{red}{x}$
- $\Sigma = V^{-1}AV$



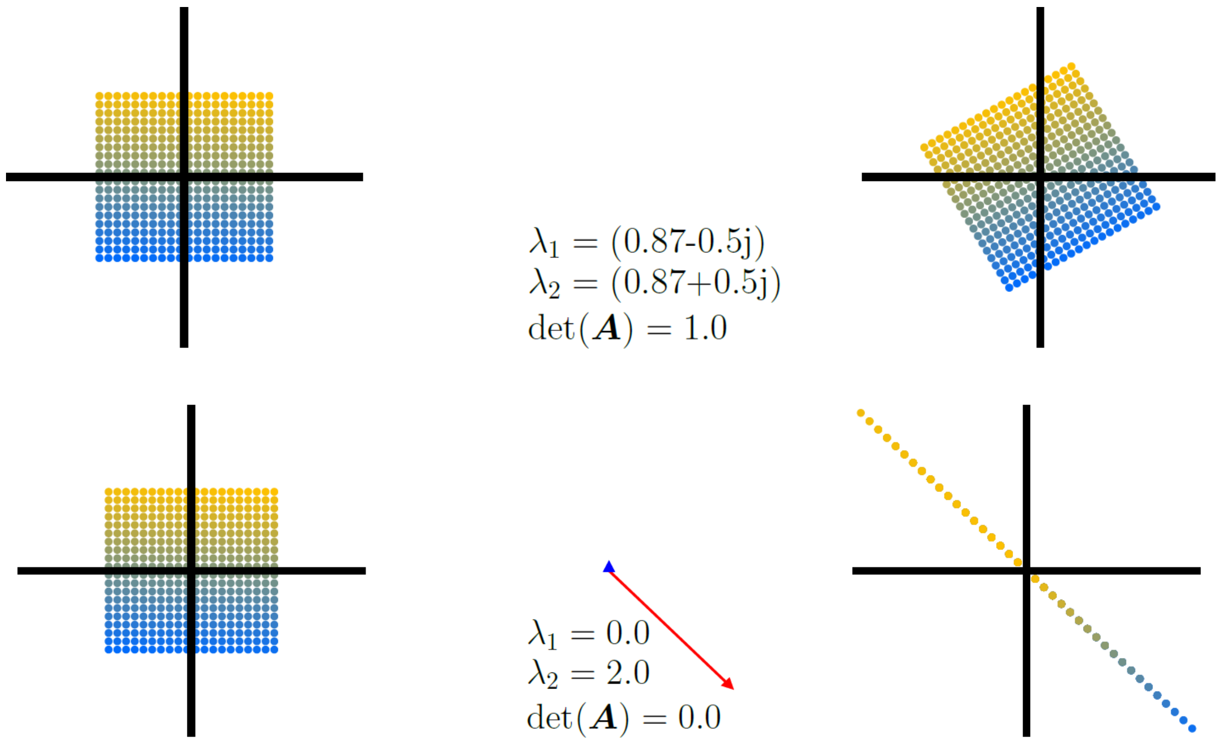
Eigendecomposition

- Projection Results ($Ax = V \Sigma V^{-1} x$):



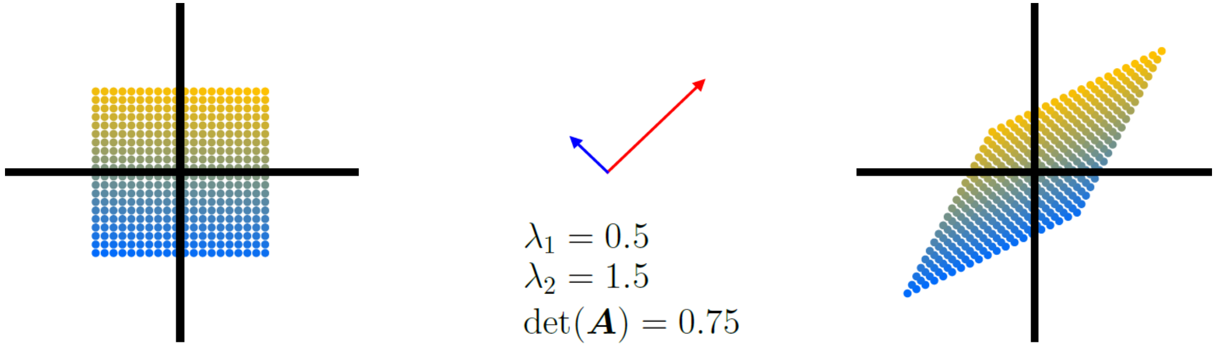
Eigendecomposition

- Projection Results ($Ax = V\Sigma V^{-1}x$):



Eigendecomposition

- Projection Results ($Ax = V\Sigma V^{-1}x$):



Hermitian Matrix ($A = A^H$)

- All $\{\lambda_i\}_{i=1}^n$ are real
- $v_i^H v_j = \delta(i-j), \delta(s) = \begin{cases} 1, & s = 0 \\ 0, & s \neq 0 \end{cases}, \ orthonormality$
- $A = V\Sigma V^H, \ VV^H = I$
- For real Hermitian Matrix:
- $\{v_i\}_{i=1}^n$ are real
- $A = V\Sigma V^H = \sum_{i=1}^n \lambda_i \, v_i v_i^H, \ \ A^{-1} = V\Sigma^{-1}V^H = \sum_{i=1}^n \frac{1}{\lambda_i} v_i v_i^H$

Hermitian Matrix ($A = A^H$)

- For any Hermitian matrix V is unitary: ($V^{-1} = V^H$)

$$V = [v_1|v_2|\cdots|v_n] \Longrightarrow VV^H = V^HV = I$$

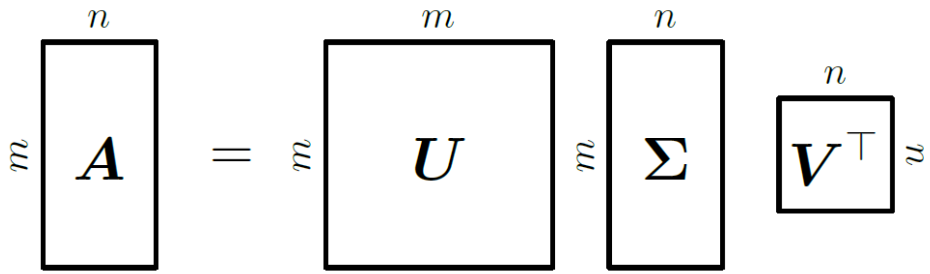
Positive Definite Matrix

- For Hermitian matrix, is pdm if:

$$x^H A x > 0, \qquad \forall x$$

Singular Value Decomposition (SVD)

- For any matrix $A \in \mathbb{R}^{m\times n}$
- $A = U\Sigma V^H$
- $UU^H = I_{m\times m}$, left singular vector (AA^H)
- $VV^H = I_{n\times n}$, right singular vector ($A^H A$)
- Σ: Rectangular diagonal ($m \times n$) nonnegative real (*decreasing ordered*)
- Matrix Approximation:



$$A = U\Sigma V^H = \sum_{i=1}^{rank(A)} \lambda_i u_i v_i^T \Rightarrow \tilde{A} \cong \sum_{i=1}^{k<rank(A)} \lambda_i u_i v_i^T$$

Singular Value Decomposition (SVD)

- Example:

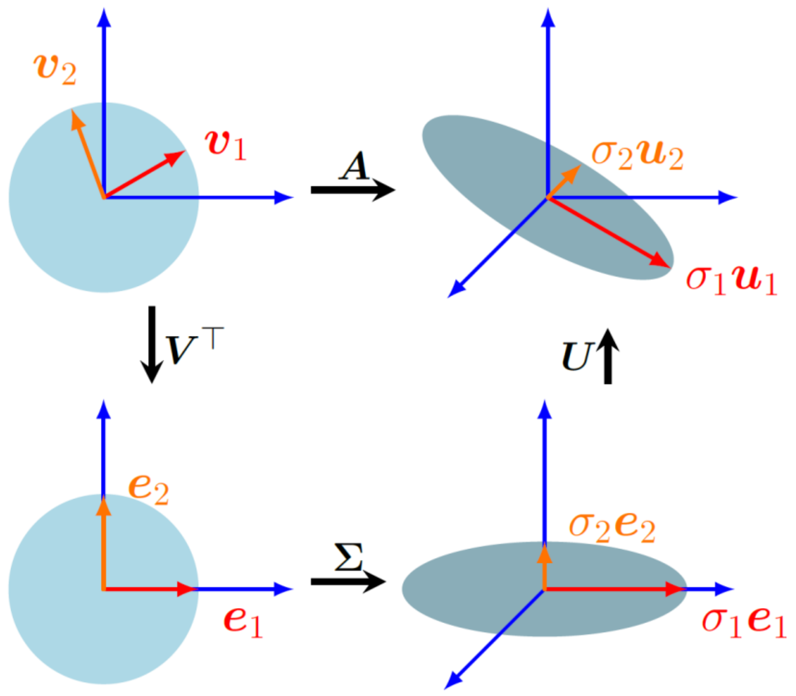
$$\begin{bmatrix} 5 & 4 & 1 \\ 5 & 5 & 0 \\ 0 & 0 & 5 \\ 1 & 0 & 4 \end{bmatrix} = \begin{bmatrix} -0.6710 & 0.0236 & 0.4647 & -0.5774 \\ -0.7197 & 0.2054 & -0.4759 & 0.4619 \\ -0.0939 & -0.7705 & -0.5268 & -0.3464 \\ -0.1515 & -0.6030 & 0.5293 & -0.5774 \end{bmatrix}$$

$$\begin{bmatrix} 9.6438 & 0 & 0 \\ 0 & 6.3639 & 0 \\ 0 & 0 & 0.7056 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -0.7367 & -0.6515 & -0.1811 \\ 0.0852 & 0.1762 & -0.9807 \\ 0.6708 & -0.7379 & -0.0743 \end{bmatrix}$$

Singular Value Decomposition (SVD)

- Projection results ($Ax = U\Sigma V^H x$):



System of Linear Equations

- There are 3 situations, while solving $A_{m\times n}x_{n\times 1} = b_{m\times 1}$
  - Determined $(m = n)$

$$x = A^{-1}b$$

  - Overdetermines $(m > n)$, more equations than unknowns (no exact solution)

$$x^* = \underset{x}{\operatorname{argmin}} \|Ax - b\|_2^2 \Rightarrow x = (A^H A)^{-1} A^H b$$

  - Underdetermined $(m < n)$, more equations than unknowns (too many exact solution)

$$x^* = \underset{x}{\operatorname{argmin}} \|x\|_2^2, \qquad s.t. Ax = b \Rightarrow x = A^H (AA^H)^{-1} b$$

Useful gradient

- Useful Identities for Computing Gradients (X: Matrix, x: vector), from *The Matrix cookbook*
  - $\frac{\partial x^T a}{\partial x} = \frac{\partial a^T x}{\partial x} = a$
  - $\frac{\partial x^T B x}{\partial x} = (B + B^T)x$
  - $\frac{\partial}{\partial s}(x - As)^T W(x - As) = -2A^T W(x - As)$, W is symmetric
  - $\frac{\partial a^T X b}{\partial X} = ab^T$
  - $\frac{\partial a^T X^T b}{\partial X} = ba^T$

Useful gradient

- Useful Identities for Computing Gradients (X: Matrix, x: vector), from *The Matrix cookbook*
  - $\frac{\partial a^T X a}{\partial X} = \frac{\partial a^T X^T a}{\partial X} = aa^T$
  - $\frac{\partial b^T X^T X c}{\partial X} = X(bc^T + cb^T)$
  - $\frac{\partial}{\partial A}(x - As)^T W(x - As) = -2W(x - As)s^T$, W is symmetric
  - $\frac{\partial}{\partial X} \| X \|_F^2 = \frac{\partial}{\partial X} Tr(XX^H) = 2X$

Random Vectors

Contents

- Definitions
- Covariance Matrix and Properties
- Whitening
- Principal Components Analysis (PCA)

Random Vectors

- Consider random vector: $\boldsymbol{x} = (x_1 \quad x_2 \quad \dots \quad x_n)^T$
- Mean vector: $\boldsymbol{m_x} = E\{\boldsymbol{x}\}$
- Autocorrelation Matrix: $R_{xx} = E\{\boldsymbol{x}\boldsymbol{x}^H\} = [r_{ij}]$
- Covariance Matrix: $C_{xx} = E\{(\boldsymbol{x} - \boldsymbol{m_x})(\boldsymbol{x} - \boldsymbol{m_x})^H\} = R_{xx} - \boldsymbol{m_x}\boldsymbol{m_x^H} = [\sigma_{ij}]$
- Cross-Correlation Matrix: $R_{xy} = E\{\boldsymbol{x}\boldsymbol{y}^H\}$
- Cross-Covariance Matrix: $C_{xy} = E\left\{(\boldsymbol{x} - \boldsymbol{m_x})\left(\boldsymbol{y} - \boldsymbol{m_y}\right)^H\right\} = R_{xy} - \boldsymbol{m_x}\boldsymbol{m_y^H}$
- Orthogonal random vector: $R_{xy} = 0$
- Uncorrelated random vector: $C_{xy} = 0$

Covariance Matrix Properties

- Symmetries:

$$R_{xx} = R_{xx}^H, \qquad C_{xx} = C_{xx}^H, \qquad R_{xy} = R_{yx}^H, \qquad C_{xy} = C_{yx}^H$$

- Eigenvalues and Eigenvectors ($AV = V\sum$):
  - For real vectors $\{v_i\}_{i=1}^n$ are real
  - $\{\lambda_i\}_{i=1}^n$ are non-negative
- C_{xx} is *nnpd* matrix
- An useful decomposition:

$$C_{xx} = LL^H, \qquad L = V\Sigma^{0.5}$$

Whitening

- White stochastic Vectors:

$$\boldsymbol{m}_w = \boldsymbol{0}$$

$$R_{ww} = C_{ww} = diag(\sigma^2 I)$$

- Whitening:

$$\boldsymbol{w} = A\boldsymbol{x} + b \Longrightarrow b = -A\boldsymbol{m}_x \Longrightarrow \boldsymbol{w} = A(\boldsymbol{x} - \boldsymbol{m}_x)$$

$$A = L^{-1} = \Gamma, C_{xx} = LL^H \text{ (}\textit{Whitener}\text{ Matrix)}$$

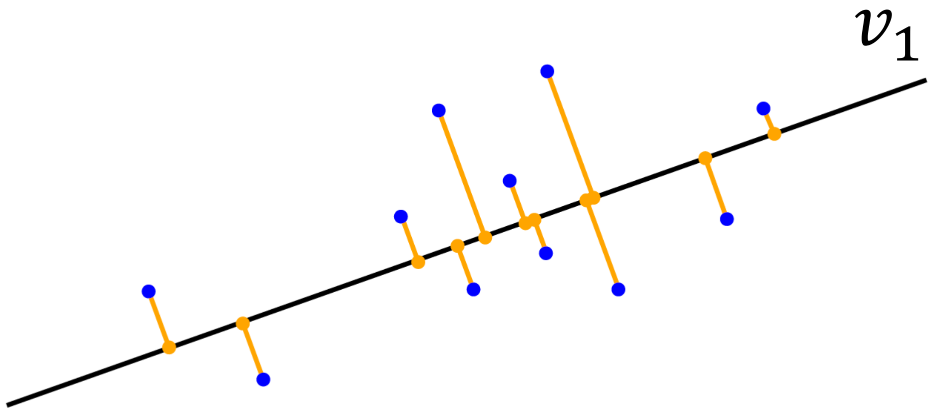
$$\boldsymbol{x} = L\boldsymbol{w} + \boldsymbol{m}_x \text{ (}\textit{Innovation}\text{ Matrix)}$$

Principal Component Analysis

- Algorithm:
  - Data: $\{\boldsymbol{x}_i\}_{i=1}^{N}, \boldsymbol{x}_i \in \mathbb{R}^D$
  - $\boldsymbol{m_x} = E\{\boldsymbol{x}\} \cong \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i$
  - $C_{xx} = E\{(\boldsymbol{x}-\boldsymbol{m_x})(\boldsymbol{x}-\boldsymbol{m_x})^T\} \cong \frac{1}{N-1}\sum_{i=1}^{N}(\boldsymbol{x}_i-\boldsymbol{m_x})(\boldsymbol{x}_i-\boldsymbol{m_x})^H$
  - Eigen decomposition: $C_{xx}v = \lambda v$
  - $A^T = [v_1|v_2|\cdots|v_D],\ \ \lambda_1 \geq \lambda_2 \cdots \geq \lambda_D \geq 0 \Rightarrow A^T = A^{-1}$

Principal Component Analysis

- Direction with Maximal Variance (largest eigenvalue):



Principal Component Analysis

- Whitening:

$$\boldsymbol{z} = A(\boldsymbol{x} - \boldsymbol{m}_x) \Rightarrow \boldsymbol{m}_z = \boldsymbol{0}, \quad \mathbf{C}_z = A\mathbf{C}_{xx}A^{\mathbf{T}} = diag([\lambda_1 \quad \lambda_2 \quad \cdots \quad \lambda_D])$$

- Complete (Error Free) Synthesis:

$$\boldsymbol{x} = A^T\boldsymbol{z} + \boldsymbol{m}_x = [v_1|v_2|\cdots|v_D]\boldsymbol{z} + \boldsymbol{m}_x = \sum_{i=1}^{D} z_i v_i + \boldsymbol{m}_x$$

- Optimal Lossy Synthesis (Note: $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_D \geq 0$):

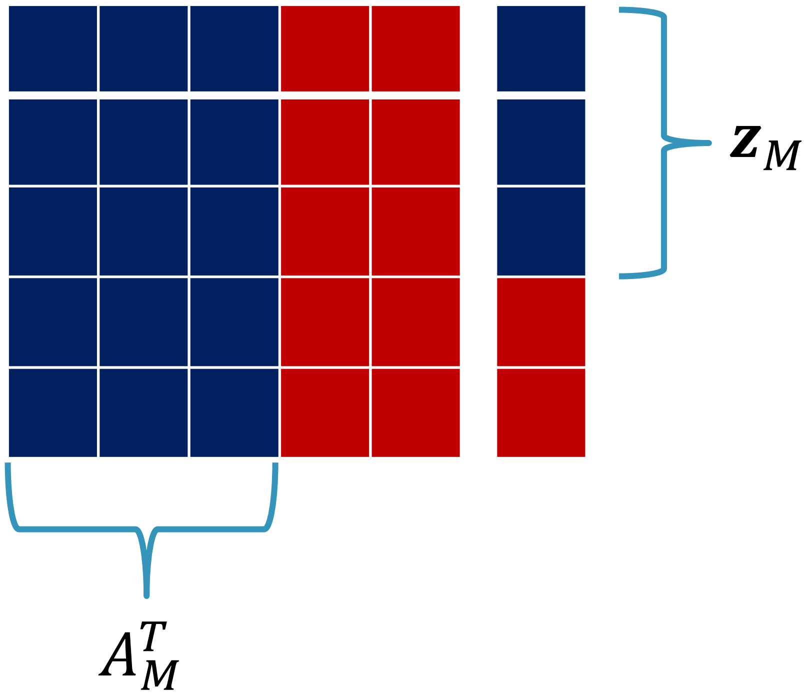
$$\widetilde{\boldsymbol{x}} = \sum_{i=1}^{M} z_i v_i + \boldsymbol{m}_x = [v_1|v_2|\cdots|v_M]\boldsymbol{z}_M + \boldsymbol{m}_x = A_M^T\boldsymbol{z}_M + \boldsymbol{m}_x$$

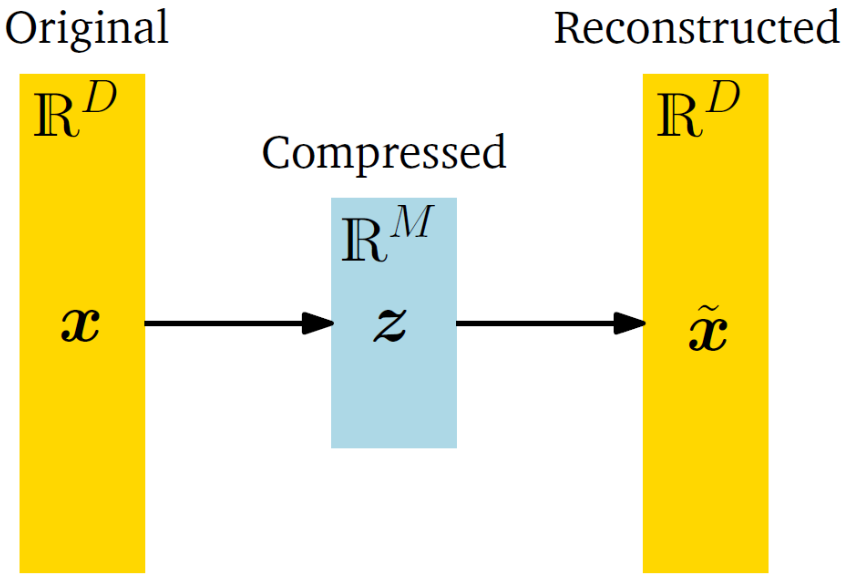
$$e_{rms} = E\{\|\boldsymbol{x} - \widetilde{\boldsymbol{x}}\|^2\} = \sum_{j=1}^{D} \lambda_j - \sum_{j=1}^{M} \lambda_j = \sum_{j=M+1}^{D} \lambda_j$$

Principal Component Analysis

- Dimensionality Reduction with PCA

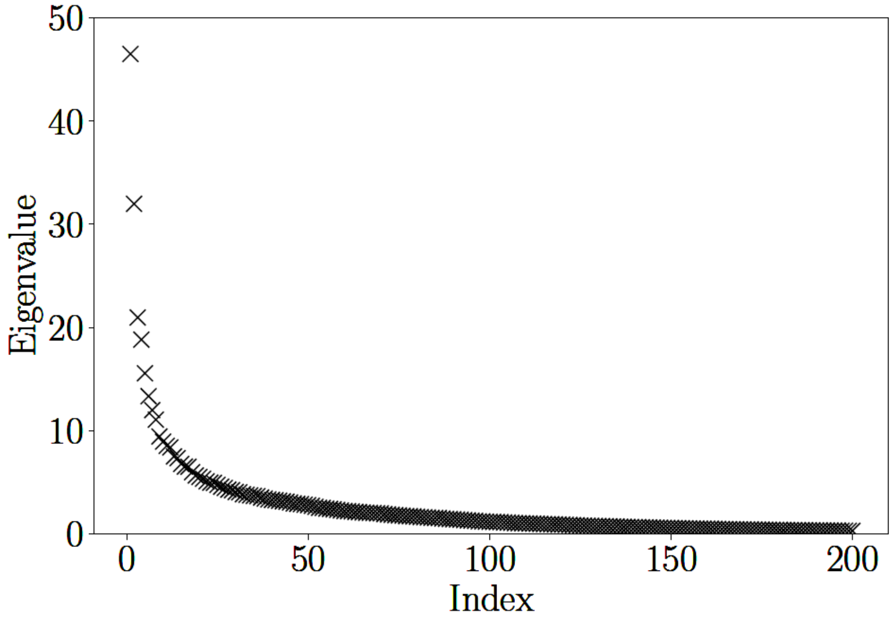
$$\tilde{\boldsymbol{x}} = A_M^T \boldsymbol{z}_M + \boldsymbol{m}_x$$

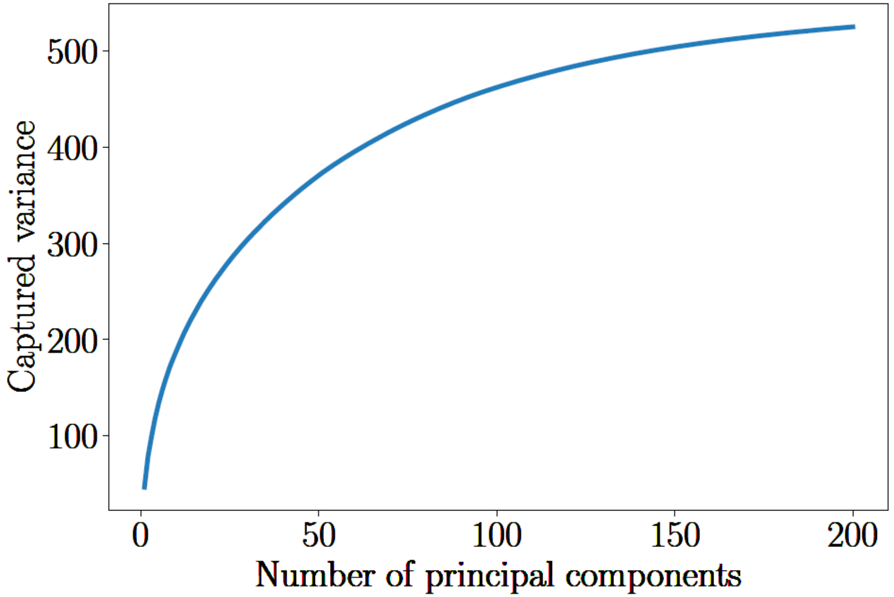




Principal Component Analysis

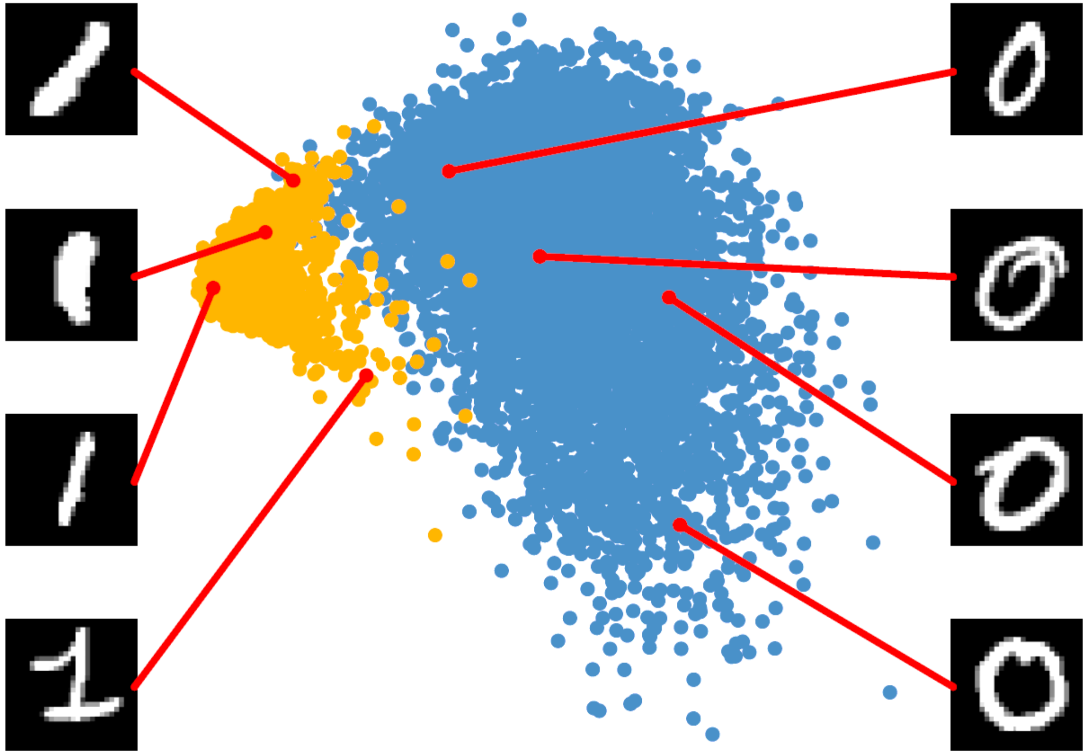
- Example:
  - MNIST “8”





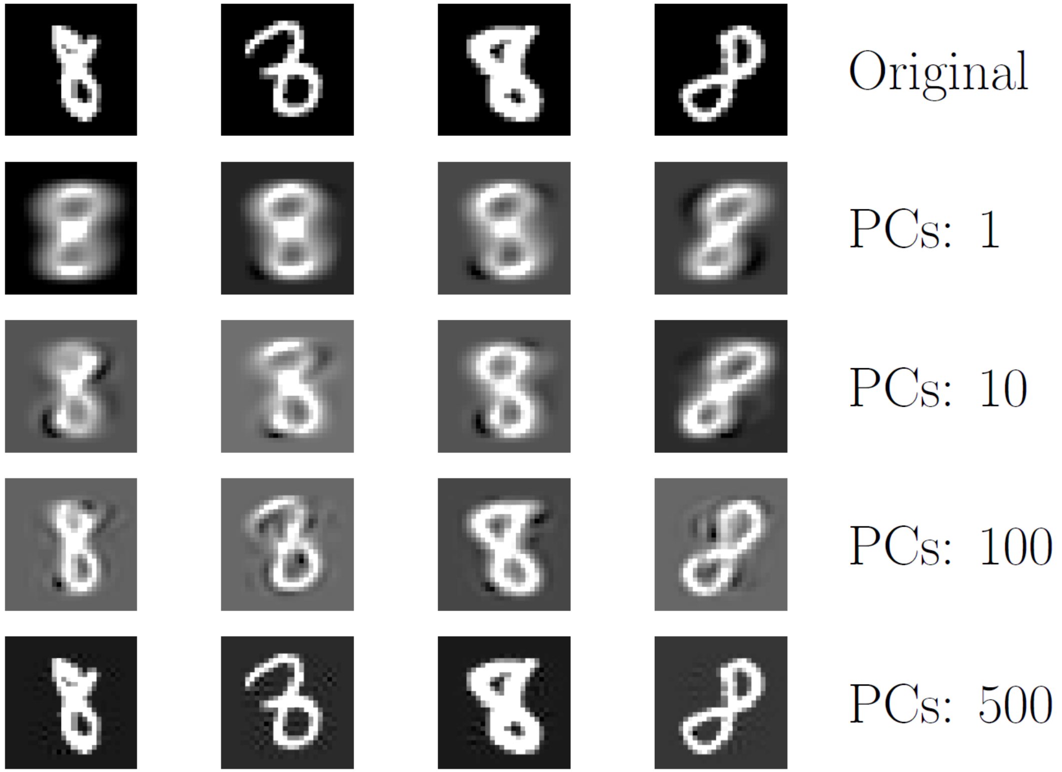
Principal Component Analysis

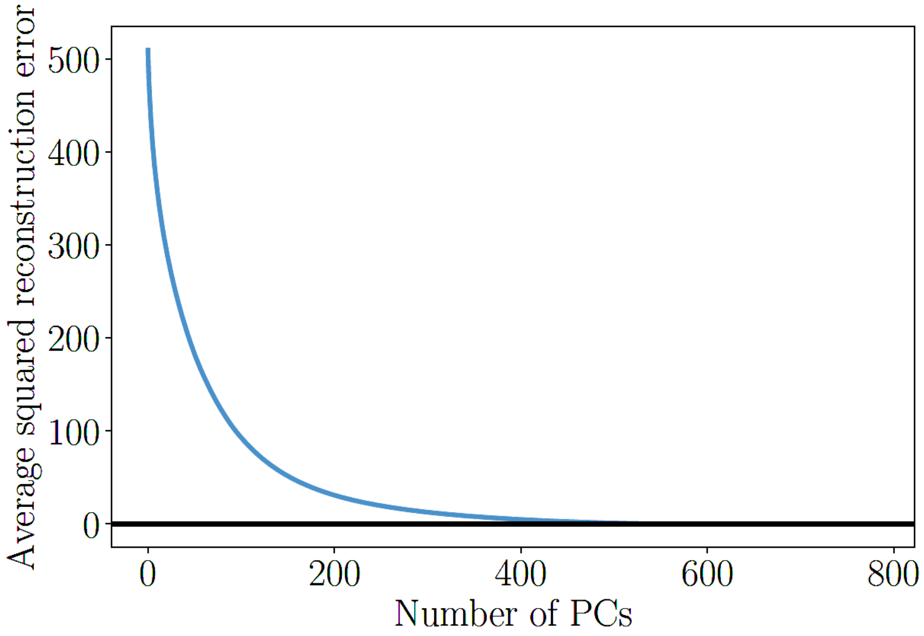
- MNIST digits embedding (first two components)



Principal Component Analysis

- MNIST Reconstruction (D=28×28=784, N=60000, M=1, 10, 100, 500)





Optimization

Contents

- Definitions
- Machine Learning Problems
- Gradient Descent (GD)
- Learning rate and initial guess effect
- Stochastic Gradient Descend (SGD)

Types of Optimization Problems

- Two types of optimization problems
- Unconstraint Problems:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{x};\boldsymbol{\theta})$$

- Constraint Problems:

$$\begin{aligned} &\min_{\boldsymbol{\theta}} J(\boldsymbol{x};\boldsymbol{\theta}) \\ \mathit{subject\ to}:\ & g_i(\boldsymbol{x};\boldsymbol{\theta}) = 0 \quad i = 1, 2, \dots, p \\ & h_k(\boldsymbol{x};\boldsymbol{\theta}) \geq 0 \quad k = 1, 2, \dots, m \end{aligned}$$

- We deal with first problem:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{x};\boldsymbol{\theta}) = \frac{\partial J(\boldsymbol{x};\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbf{0}$$

- A difficult/impossible to solve exactly

Optimization in Machine Learning

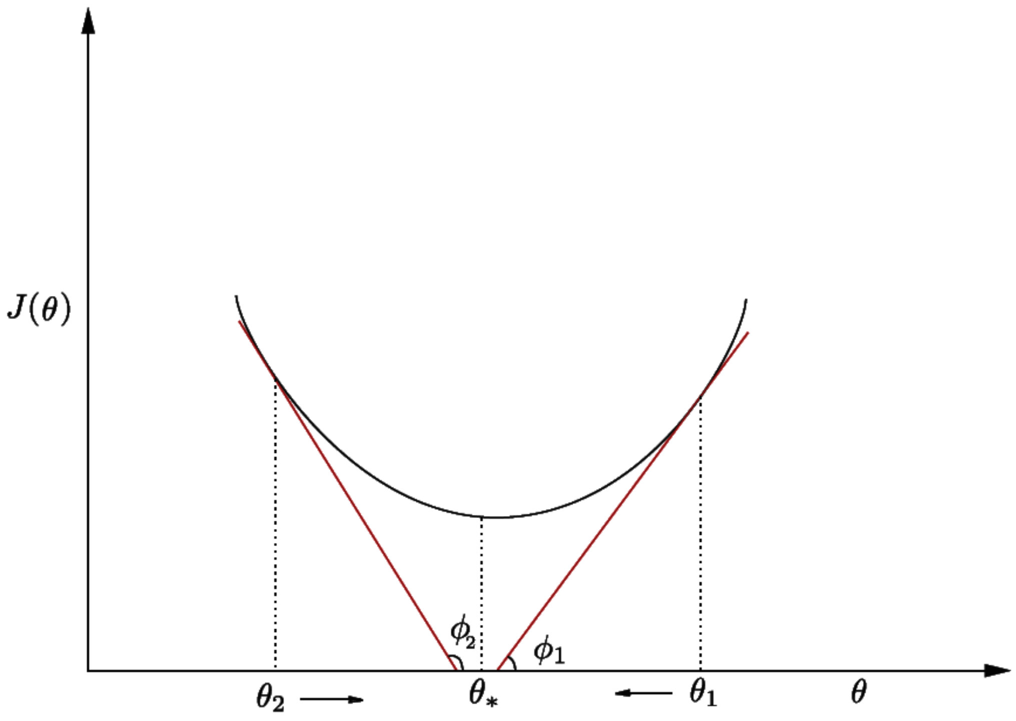
- Most common problem in machine learning:

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} Loss(f(\boldsymbol{x}_i; \boldsymbol{\theta}), \boldsymbol{y}_i)$$

- $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$: machine input and *desired* output (training set)
- $\boldsymbol{\theta}$: machine parameters
- $f(\boldsymbol{x}_i; \boldsymbol{\theta})$: machine actual output in response to $\boldsymbol{x}_i$

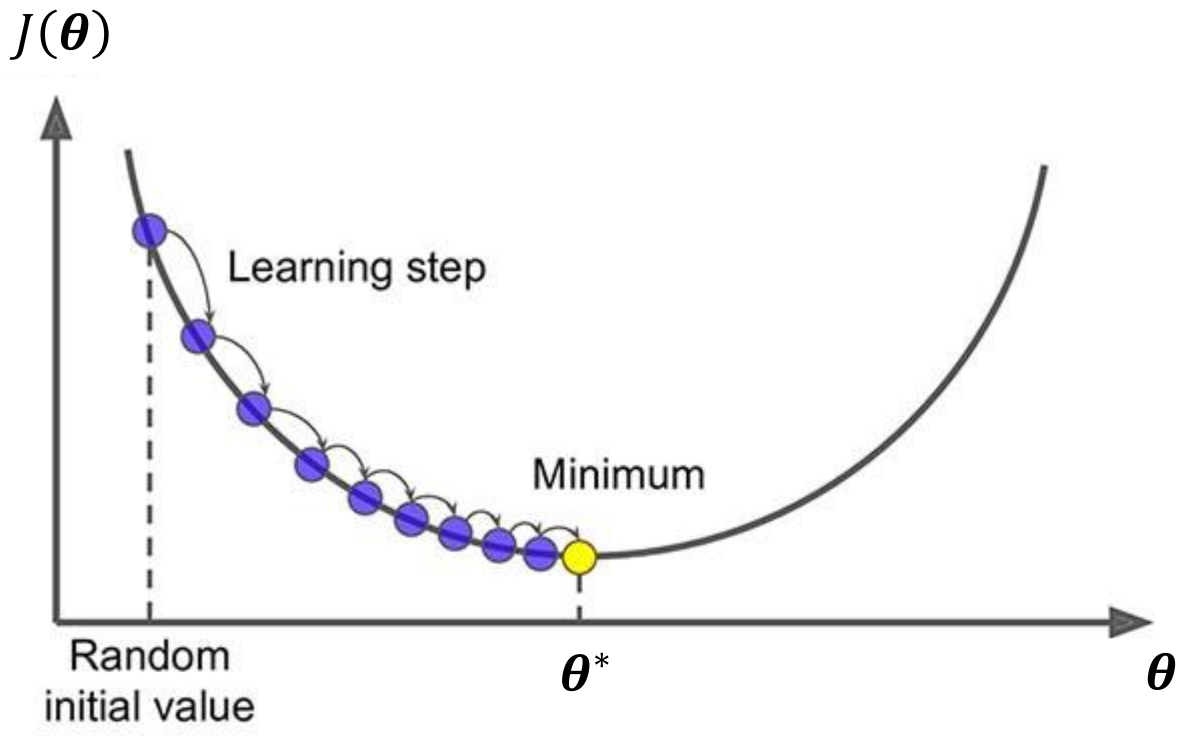
Gradient Descent (Steepest Descent)

- Gradient Descent (GD) is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function (or global minimum of convex function).
- $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i \nabla J\left(\boldsymbol{\theta}^{(i-1)}\right)$
- $\theta^* = \underset{\theta}{\operatorname{argmin}}\, J(\theta)$
- i: $iteration\ step$
- μ_i: $step\ size\ (learning\ rate)$



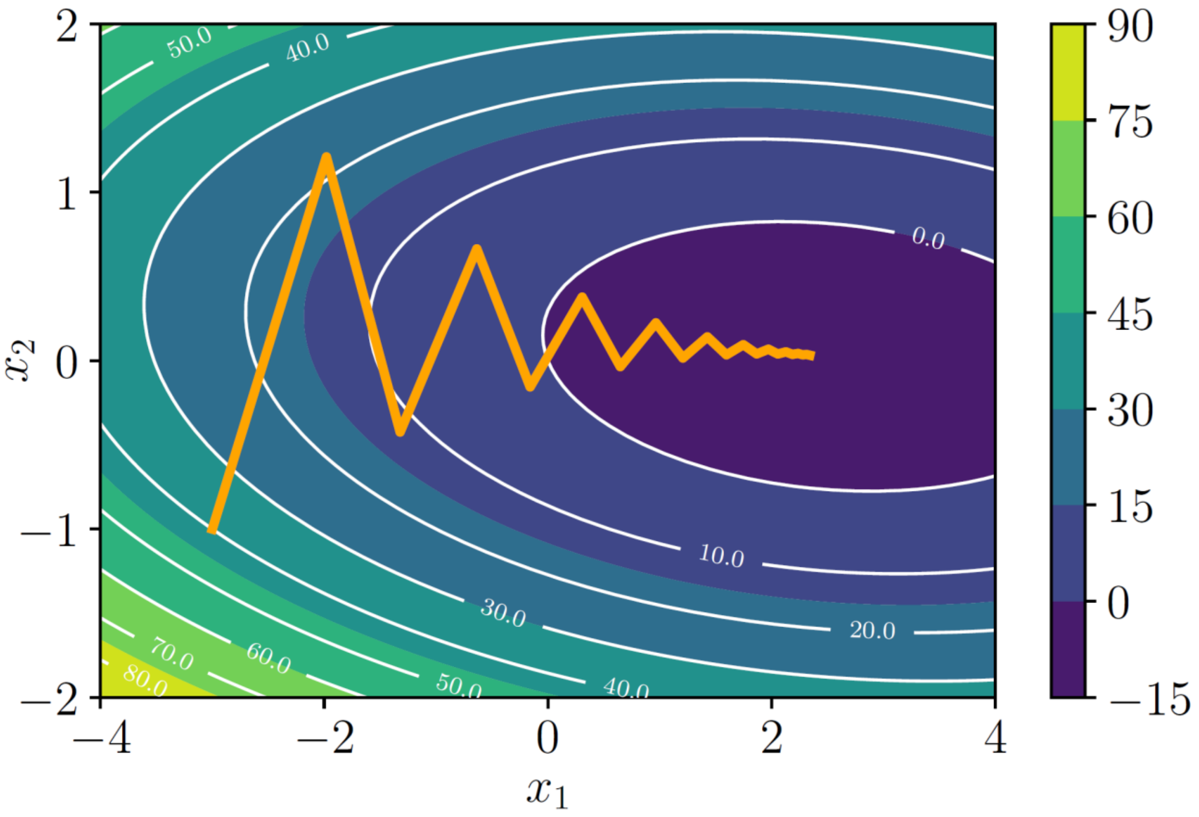
Gradient Descent (Steepest Descent)

- How it works



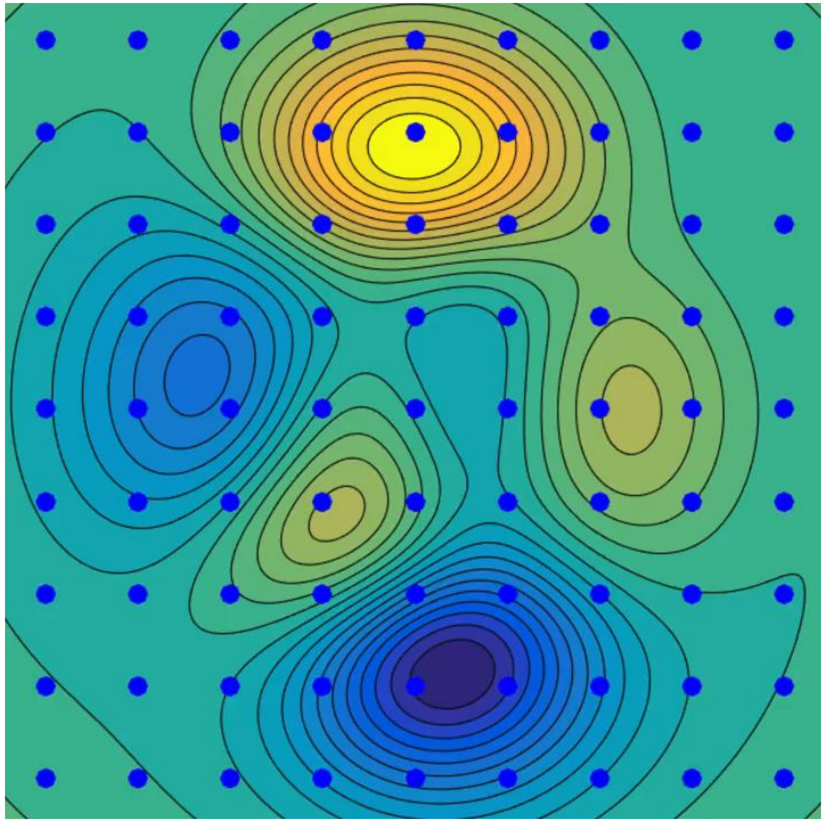
Gradient Descent (Steepest Descent)

- How it works



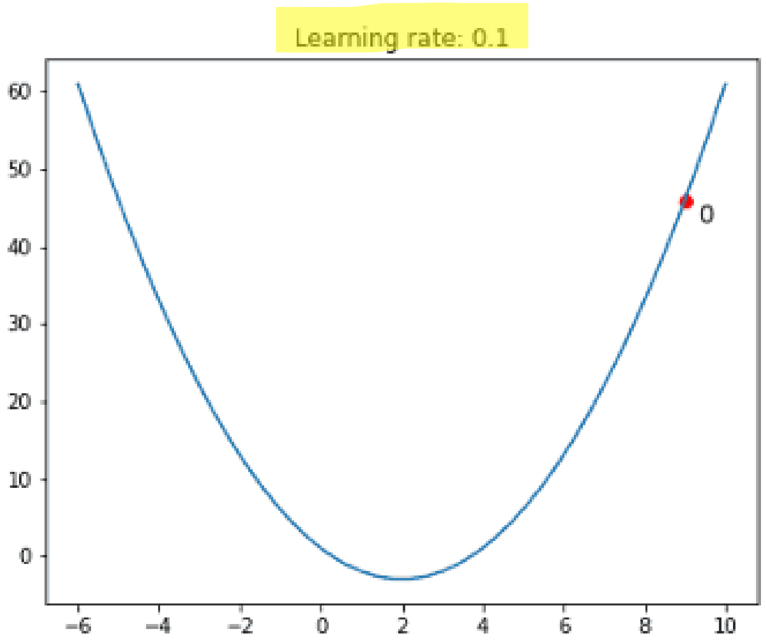
Gradient Descent (Steepest Descent)

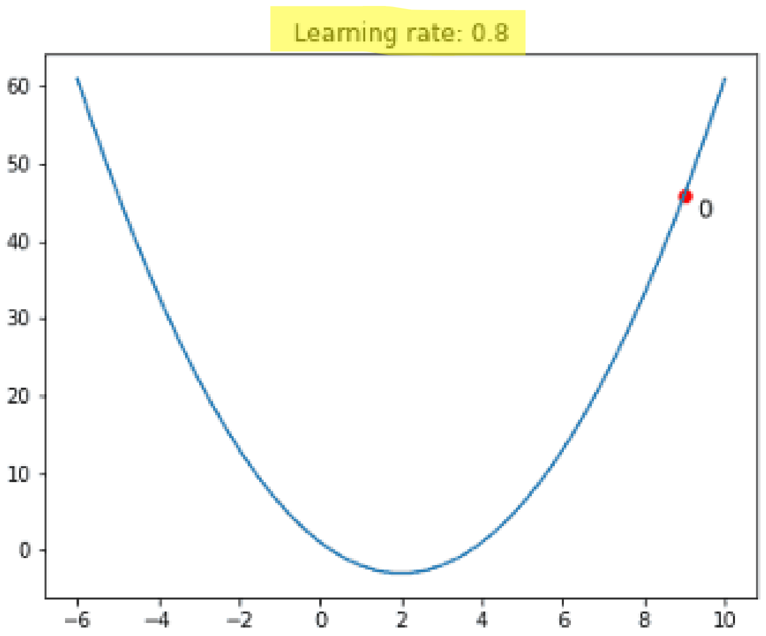
- How it works



Step-size (learning rate) effect (1)

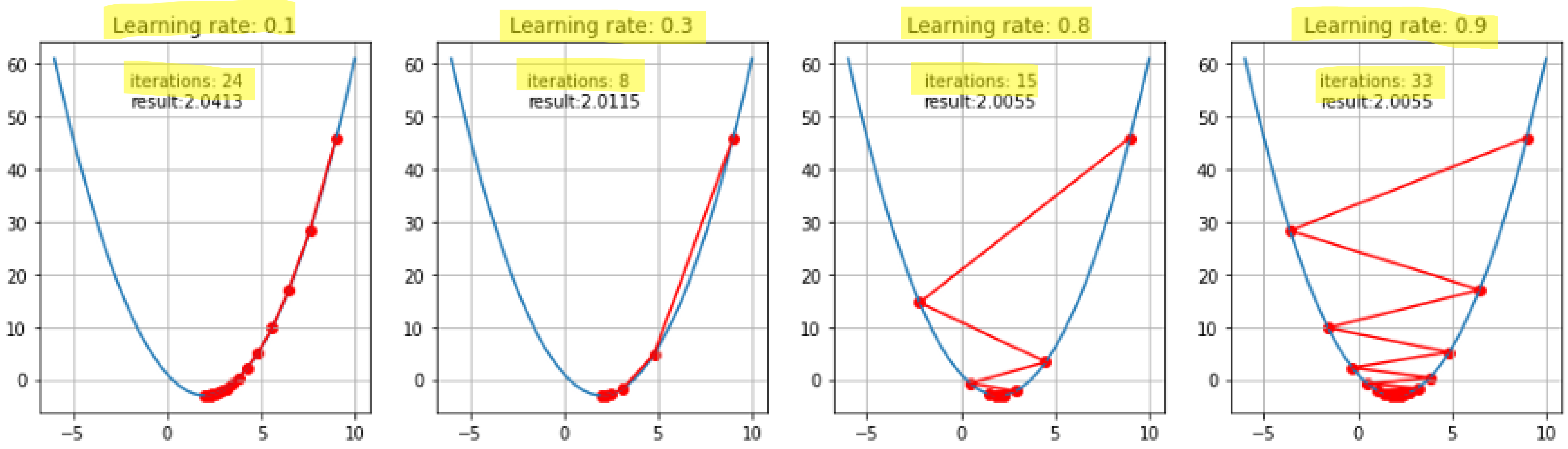
- Consider a simple convex problem:





Step-size (learning rate) effect (2)

- Consider a simple convex problem:



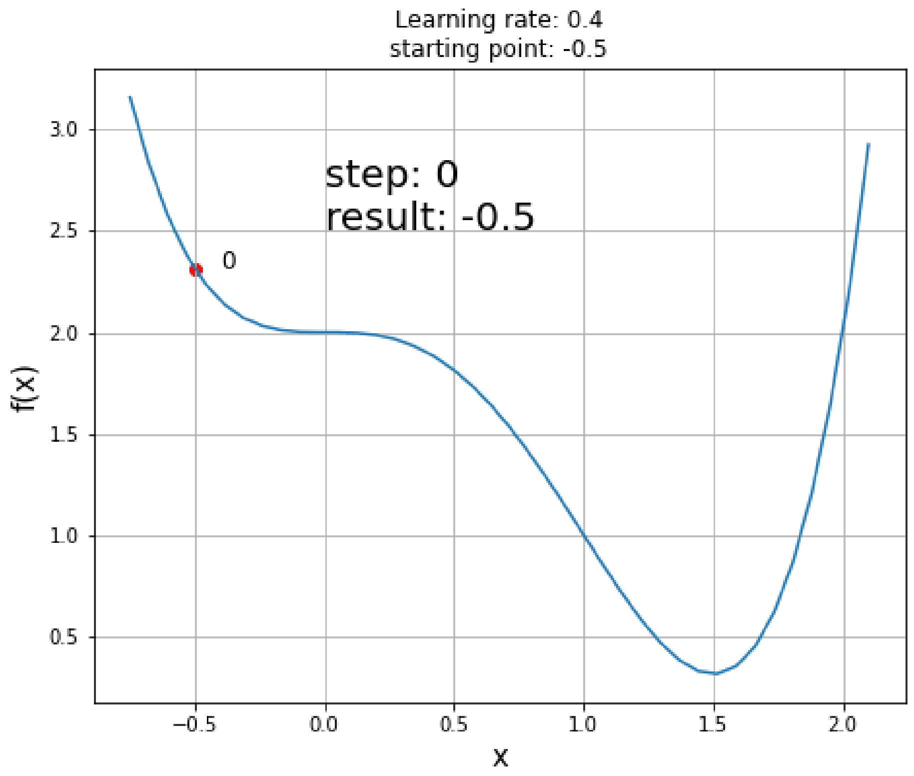
- Look at number of iterations as learning rate increase!

Initial guess and learning rate effect

- Consider a non-convex (difficult) problem:

Saddle point and learning rate effect

- Consider a non-convex (difficult) problem:



Learning rate variation

- There are several variations:

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i \nabla J\left(\boldsymbol{\theta}^{(i-1)}\right)$$

- $\mu_i = \mu_0, (10^{-6} < \mu_i < 1), \propto 10^{-2}$
- $\mu_i = \frac{\mu_0}{i}$
- $\mu_i = \frac{\mu_0}{1+i/T}$
- $\mu_i = \mu_0 \frac{\left|\nabla J\left(\boldsymbol{\theta}^{(i-1)}\right)\right|}{1+\left|\nabla J\left(\boldsymbol{\theta}^{(i-1)}\right)\right|^2}$
- …

Stochastic Gradient Descent (SGD)

- Recall the standard (*batch-mode*) Gradient Descent:

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} Loss(f(\boldsymbol{x}_i; \boldsymbol{\theta}), \boldsymbol{y}_i)$$

- SGD replaces the actual gradient (calculated from the entire training dataset by stochastic gradient using *randomly* selected subset of the training dataset (minibatch).

SGD implementation(s):

- Online GD (Sample/pattern mode):

- Hyperparameter selection (here learning rate), μ_0
- Random initialization, $\boldsymbol{\theta}^{(0)}$
- Repeat until a convergence criteria satisfied
  - Randomly shuffle samples in the training set
  - for all samples (1 to N)
    - $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i \nabla Loss\left(f\left(\boldsymbol{x}_i; \boldsymbol{\theta}^{(i-1)}\right), \boldsymbol{y}_i\right)$

SGD implementation(s):

- Mini-batch SGD:

- Hyperparameter selection (here learning rate and minibatch size), μ_0, m
- Random initialization, $\boldsymbol{\theta}^{(0)}$
- Repeat until a convergence criteria satisfied
  - Randomly pick a *mini-batch* of size m from the training set
  - $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i \frac{1}{m} \nabla \sum_{k=1}^{m} Loss\left(f\left(\boldsymbol{x}_k; \boldsymbol{\theta}^{(i-1)}\right), \boldsymbol{y}_k\right)$

Any Question