Mathematics Toolbox for Machine Learning

Panos Parpas, April 2023

Introductions

- Panos Parpas
- Reader in Computational and Applied Mathematics
- Department Computing, Imperial College London
- panos.parpas@fitchlearning.com
- www.doc.ic.ac.uk/~pp500

Previously

- Research Fellow, Institute for Data, Systems, and Society, Massachusetts Institute of Technology (2009-2011)
- Global Model and Analytics Group, Credit Suisse (2007-2009)
- PhD in Stochastic Optimization (Imperial College London, 2005)
- CQF Alumni (2007)

Contents and Aims

Aims:

- Introduce a mathematical toolbox as used in ML
- Demonstrate mathematics with different ML models

A. Linear Algebra

1. Matrices and vectors
2. Measuring similarity
3. Eigenvectors/Eigenvalues
4. Matrix Factorisations

B. Calculus

C. Optimisation

1. Classes of Optimization Models/Algorithms
2. Gradient Descent (Stochastic, Acceleration, Momentum)
3. Optimality Conditions (constrained case)

Exercises with solutions available on the platform

Linear Algebra: Vectors, Matrices and Tensors

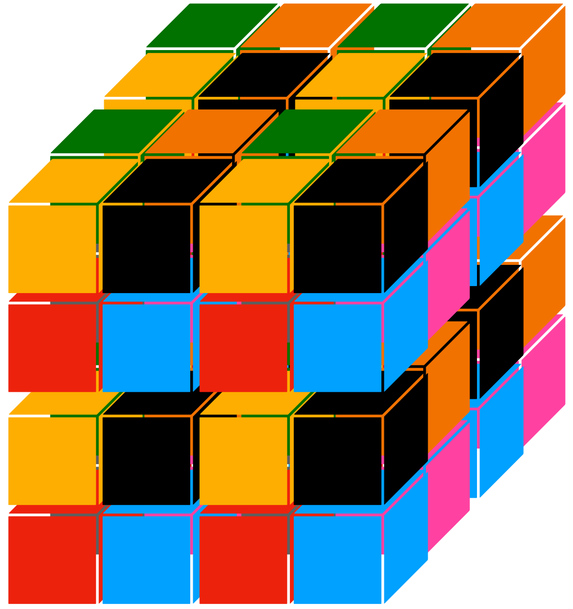
Vectors, matrices and tensors of real numbers can represent *most* of the objects encountered in ML

$\in \mathbb{R}^4$

Vector

$\in \mathbb{R}^{4\times4}$

Matrix



$\in \mathbb{R}^{4\times4\times4}$

Tensor

Vectors and Matrices as Functions

Vectors and Matrices as Functions

Vector [0.1,-1.2,3.14] is also a function:

$$1 \mapsto 0.1$$
$$2 \mapsto -1.2$$
$$3 \mapsto 3.14$$

Vectors and Matrices as Functions

Vector [0.1,-1.2,3.14] is also a function:

$$1 \mapsto 0.1$$
$$2 \mapsto -1.2$$
$$3 \mapsto 3.14$$

Discrete probability distributions e.g. fair coin

$$H \mapsto 1/2$$
$$T \mapsto 1/2$$

Vectors and Matrices as Functions

Vector [0.1,-1.2,3.14] is also a function:

$$1 \mapsto 0.1$$
$$2 \mapsto -1.2$$
$$3 \mapsto 3.14$$

Discrete probability distributions e.g. fair coin

$$H \mapsto 1/2$$
$$T \mapsto 1/2$$

Images as vectors:

$$\{1,...,...n\} \times \{1,...,...n\} \mapsto \mathbb{R}^3$$
$$[120,145] \mapsto [10,30,14]$$

Image Size (400,600)

Vectors and Matrices as Functions

Vector [0.1,-1.2,3.14] is also a function:

$$1 \mapsto 0.1$$
$$2 \mapsto -1.2$$
$$3 \mapsto 3.14$$

Discrete probability distributions e.g. fair coin

$$H \mapsto 1/2$$
$$T \mapsto 1/2$$

Images as vectors:

$$\{1,...,...n\} \times \{1,...,...n\} \mapsto \mathbb{R}^3$$
$$[120,145] \mapsto [10,30,14]$$

Image Size (400,600)

Bag of words model (Natural Language Processing)

```
John likes to watch movies. Mary likes movies too.
```

$$\text{John} \mapsto 1$$
$$\text{likes} \mapsto 2$$
$$\text{to} \mapsto 1$$
$$\text{watch} \mapsto 1$$
$$\text{movies} \mapsto 2$$
$$\text{Mary} \mapsto 1$$
$$\text{too} \mapsto 1$$

Inner Products Measure Distances

Inner (dot) product in $\mathbb{R}^d$: $\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i = x^\top y$

Length (norm) in $\mathbb{R}^d$: $\|x\| = \sqrt{\langle x, x \rangle}$

Angle between vectors $\cos(\theta) = \dfrac{\langle x, y \rangle}{\|x\| \|y\|}$

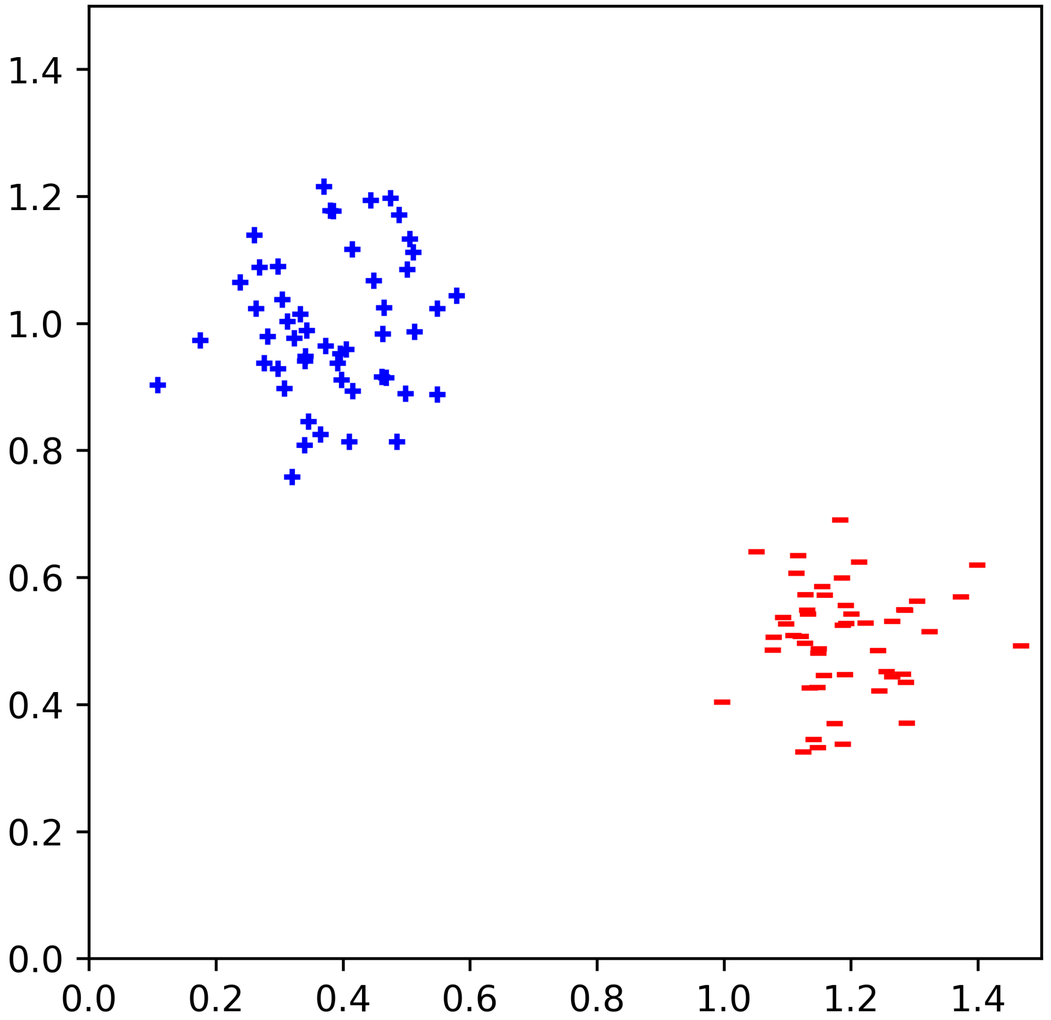
Inner Products Measure Distances

Inner (dot) product in $\mathbb{R}^d$: $\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i = x^\top y$

Length (norm) in $\mathbb{R}^d$: $\|x\| = \sqrt{\langle x, x \rangle}$

Angle between vectors $\cos(\theta) = \dfrac{\langle x, y \rangle}{\|x\| \|y\|}$

Data: $(x_1, y_1), \ldots, (x_m, y_m) \in \mathbb{R}^2 \times \pm 1$



Inner Products Measure Distances

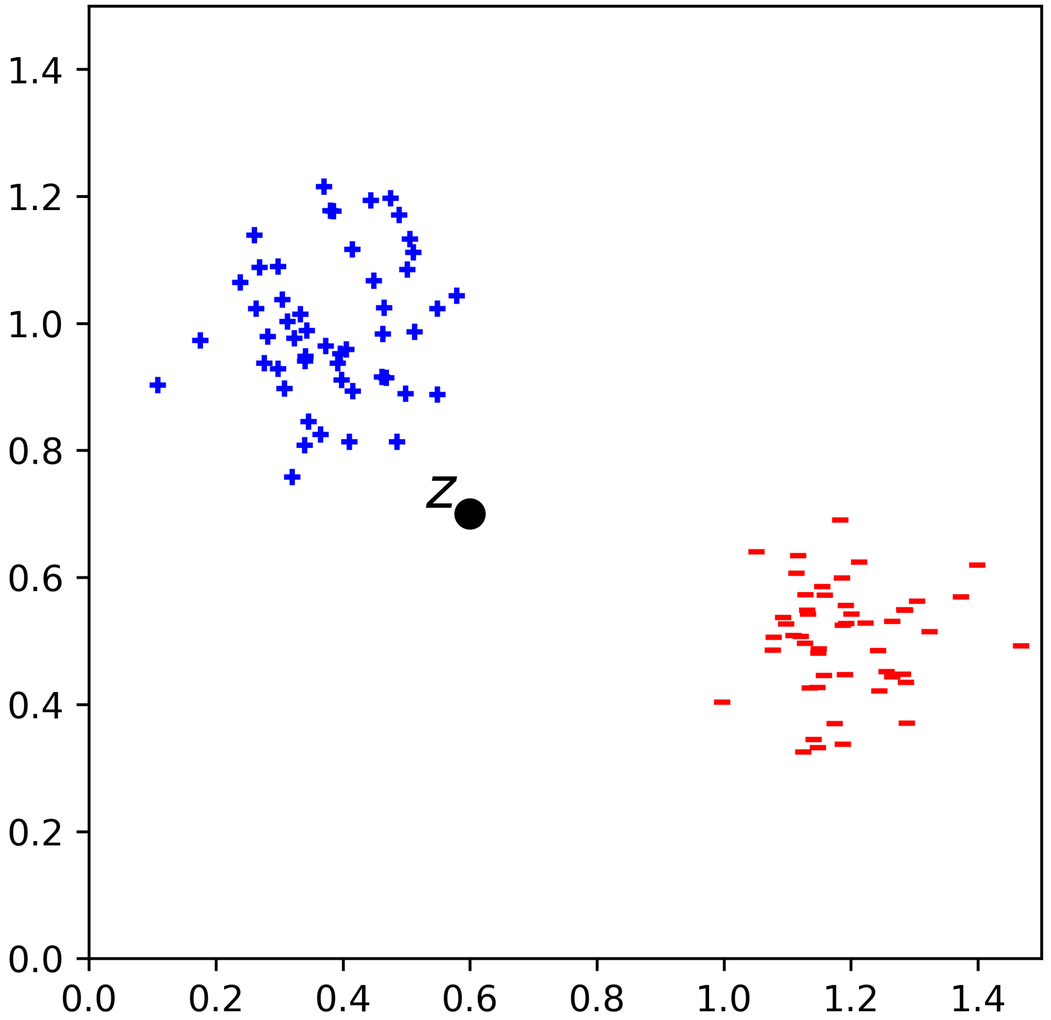
Inner (dot) product in $\mathbb{R}^d$: $\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i = x^\top y$

Length (norm) in $\mathbb{R}^d$: $\|x\| = \sqrt{\langle x, x \rangle}$

Angle between vectors $\cos(\theta) = \dfrac{\langle x, y \rangle}{\|x\| \|y\|}$

Data: $(x_1, y_1), \ldots, (x_m, y_m) \in \mathbb{R}^2 \times \pm 1$

New point Z: Is it a + or a - ?



Inner Products Measure Distances

Inner (dot) product in $\mathbb{R}^d$: $\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i = x^\top y$

Length (norm) in $\mathbb{R}^d$: $\|x\| = \sqrt{\langle x, x \rangle}$

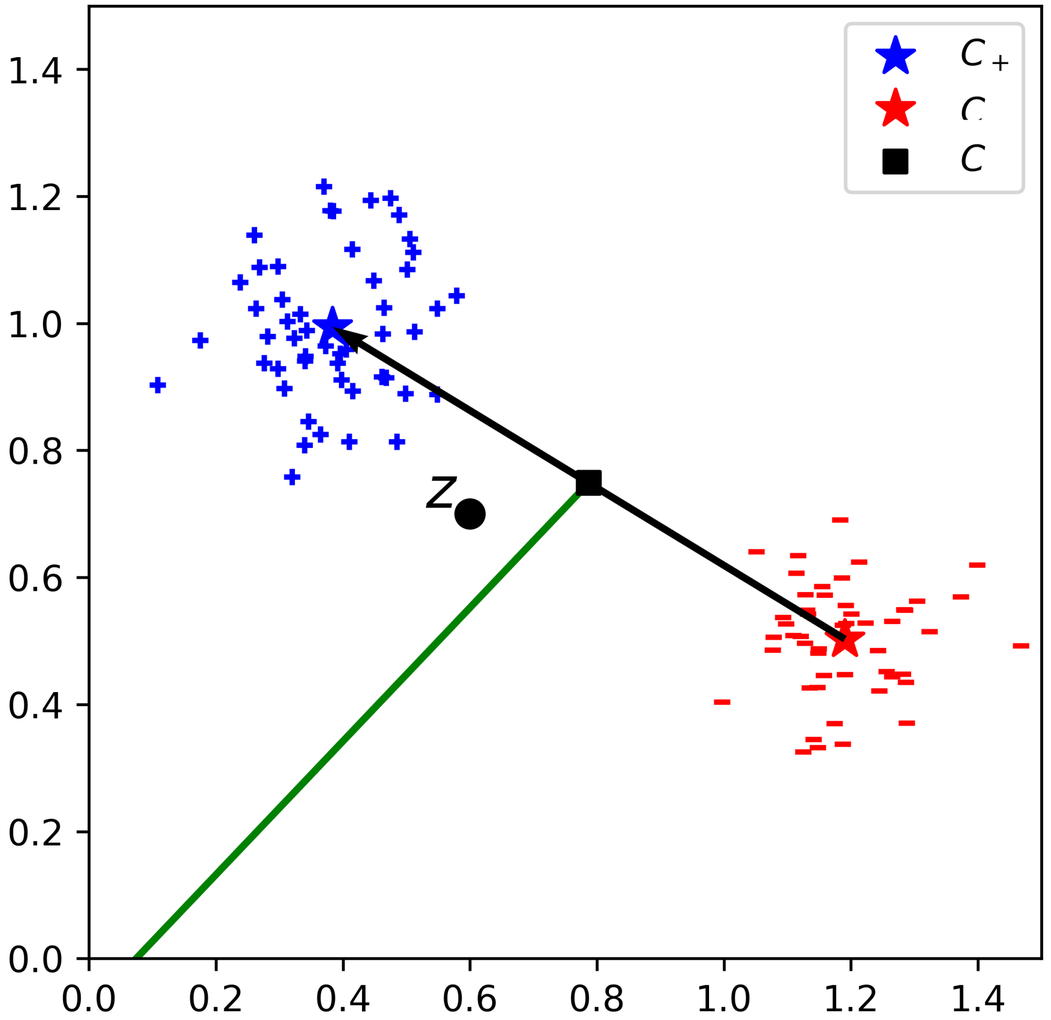
Angle between vectors $\cos(\theta) = \dfrac{\langle x, y \rangle}{\|x\| \|y\|}$

Data: $(x_1, y_1), \ldots, (x_m, y_m) \in \mathbb{R}^2 \times \pm 1$

New point Z: Is it a + or a - ?

$$C_+ = \frac{1}{m_+} \sum_{i=1}^{m_+} x_i, C_- = \frac{1}{m_-} \sum_{i=1}^{m_-} x_i$$

$$C = \frac{1}{2}(C_+ + C_-), w = C_+ - C_-$$



$\cos(\theta) = \langle z - C, w \rangle / (\|z - C\| \|w\|)$ or look at the sign of $\langle z - C, w \rangle$

Inner Products Measure Distances

Inner (dot) product in $\mathbb{R}^d$: $\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i = x^\top y$

Length (norm) in $\mathbb{R}^d$: $\|x\| = \sqrt{\langle x, x \rangle}$

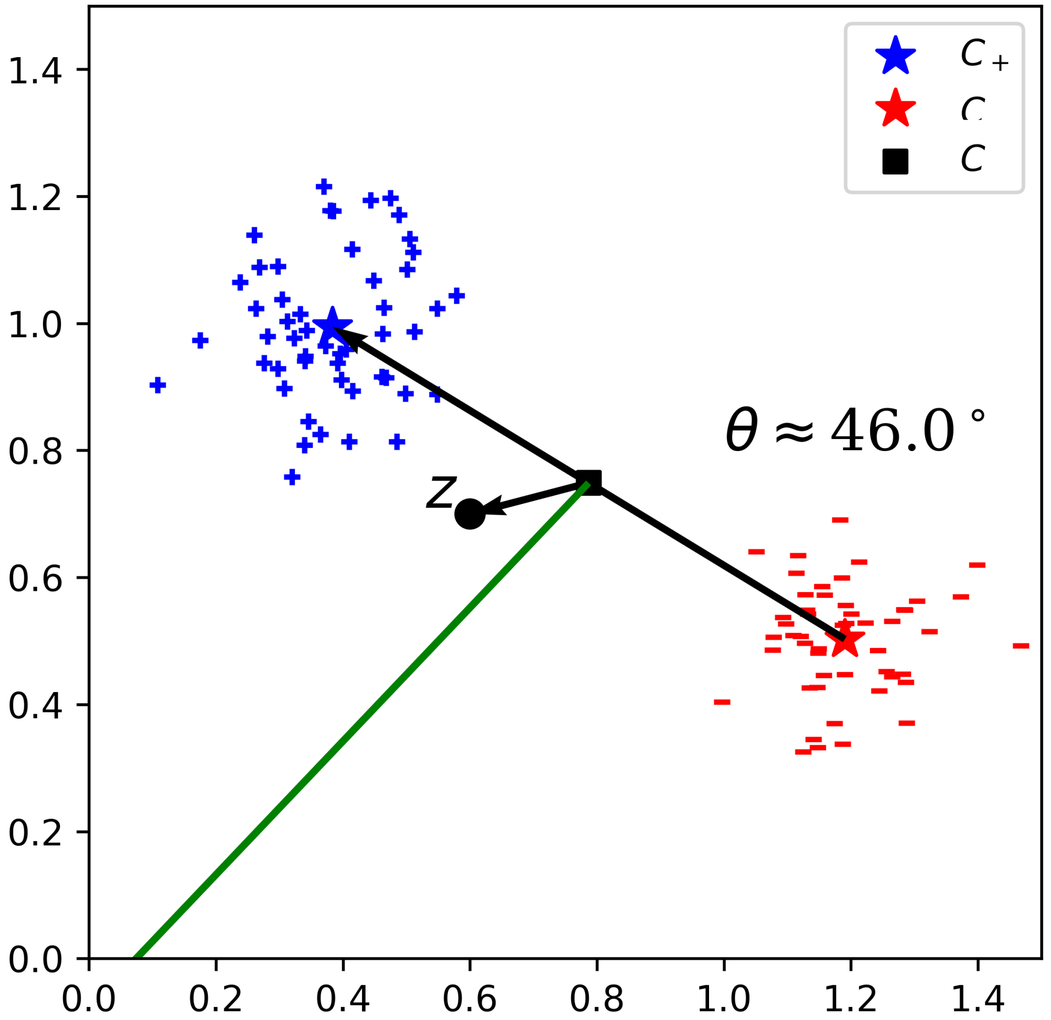
Angle between vectors $\cos(\theta) = \dfrac{\langle x, y \rangle}{\|x\| \|y\|}$

Data: $(x_1, y_1), \ldots, (x_m, y_m) \in \mathbb{R}^2 \times \pm 1$

New point Z: Is it a + or a - ?

$$C_+ = \frac{1}{m_+} \sum_{i=1}^{m_+} x_i, C_- = \frac{1}{m_-} \sum_{i=1}^{m_-} x_i$$

$$C = \frac{1}{2}(C_+ + C_-), w = C_+ - C_-$$



$\cos(\theta) = \langle z - C, w \rangle / (\|z - C\| \|w\|)$ or look at the sign of $\langle z - C, w \rangle$

Inner Products Measure Distances

Inner (dot) product in $\mathbb{R}^d$: $\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i = x^\top y$

Length (norm) in $\mathbb{R}^d$: $\|x\| = \sqrt{\langle x, x \rangle}$

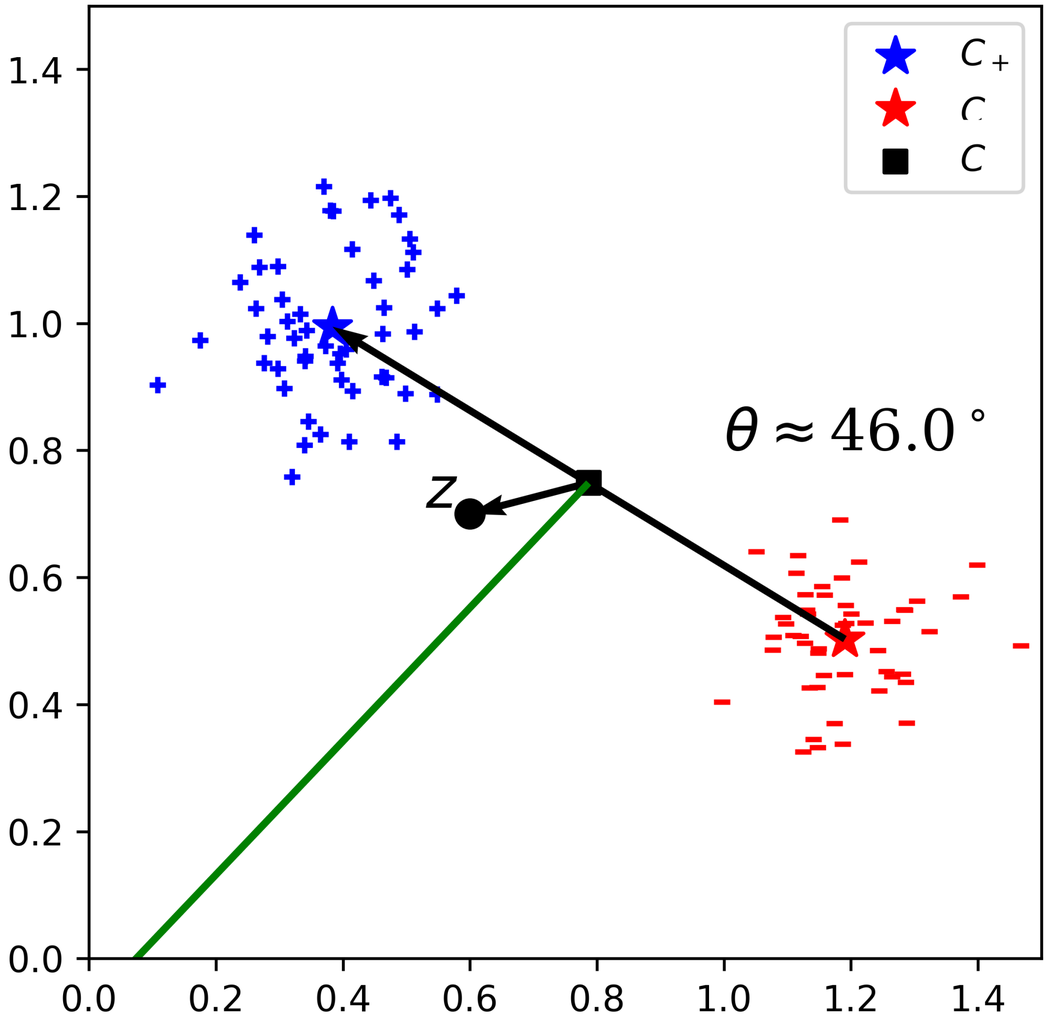
Angle between vectors $\cos(\theta) = \dfrac{\langle x, y \rangle}{\|x\| \|y\|}$

Data: $(x_1, y_1), \ldots, (x_m, y_m) \in \mathbb{R}^2 \times \pm 1$

New point Z: Is it a + or a - ?

$$C_+ = \frac{1}{m_+} \sum_{i=1}^{m_+} x_i, C_- = \frac{1}{m_-} \sum_{i=1}^{m_-} x_i$$

$$C = \frac{1}{2}(C_+ + C_-), w = C_+ - C_-$$



$\cos(\theta) = \langle z - C, w \rangle / (\|z - C\| \|w\|)$ or look at the sign of $\langle z - C, w \rangle$

Inner Products Measure Distances

Inner (dot) product in $\mathbb{R}^d$: $\langle x, y \rangle = \sum_{i=1}^{d} x_i y_i = x^\top y$

Length (norm) in $\mathbb{R}^d$: $\|x\| = \sqrt{\langle x, x \rangle}$

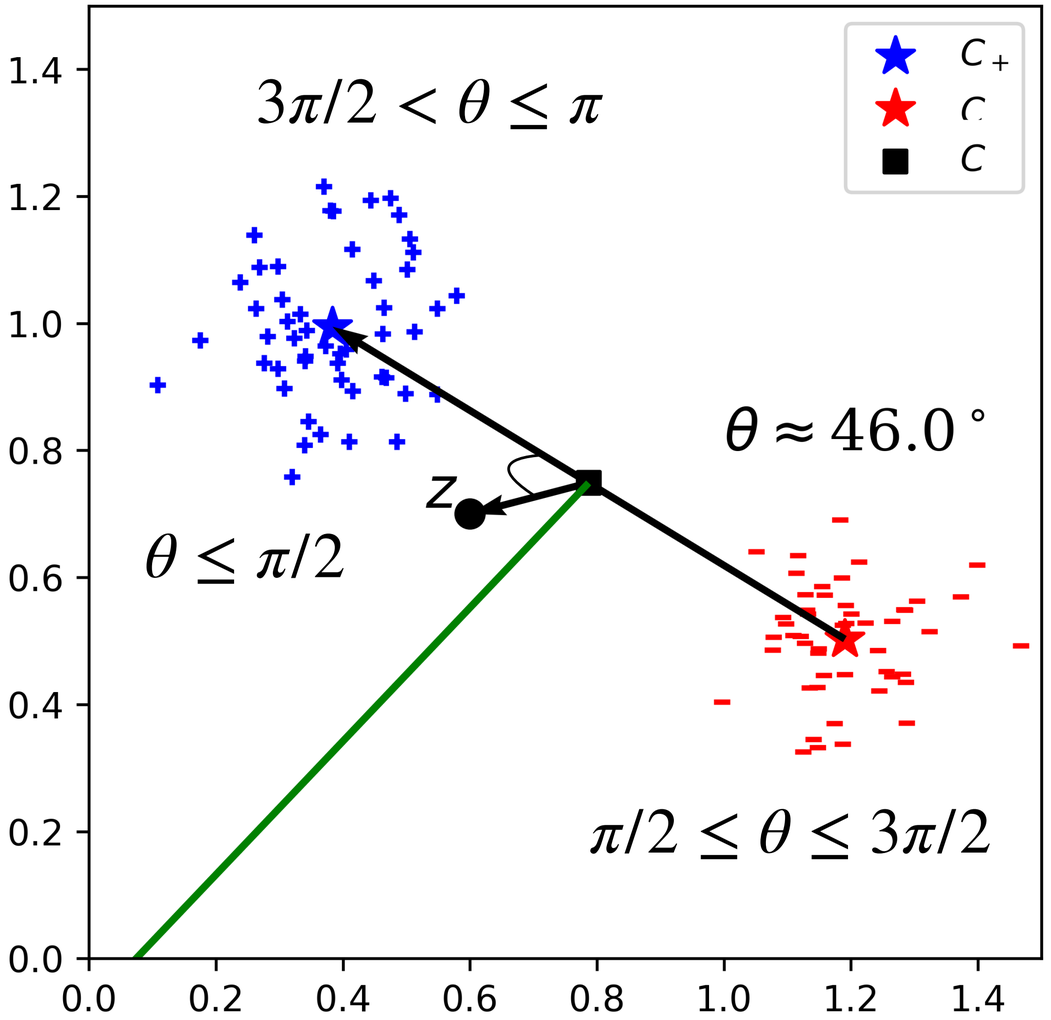
Angle between vectors $\cos(\theta) = \dfrac{\langle x, y \rangle}{\|x\| \|y\|}$

Data: $(x_1, y_1), \ldots, (x_m, y_m) \in \mathbb{R}^2 \times \pm 1$

New point Z: Is it a + or a - ?

$$C_+ = \frac{1}{m_+} \sum_{i=1}^{m_+} x_i, C_- = \frac{1}{m_-} \sum_{i=1}^{m_-} x_i$$

$$C = \frac{1}{2}(C_+ + C_-), w = C_+ - C_-$$



$\cos(\theta) = \langle z - C, w \rangle / (\|z - C\| \|w\|)$ or look at the sign of $\langle z - C, w \rangle$

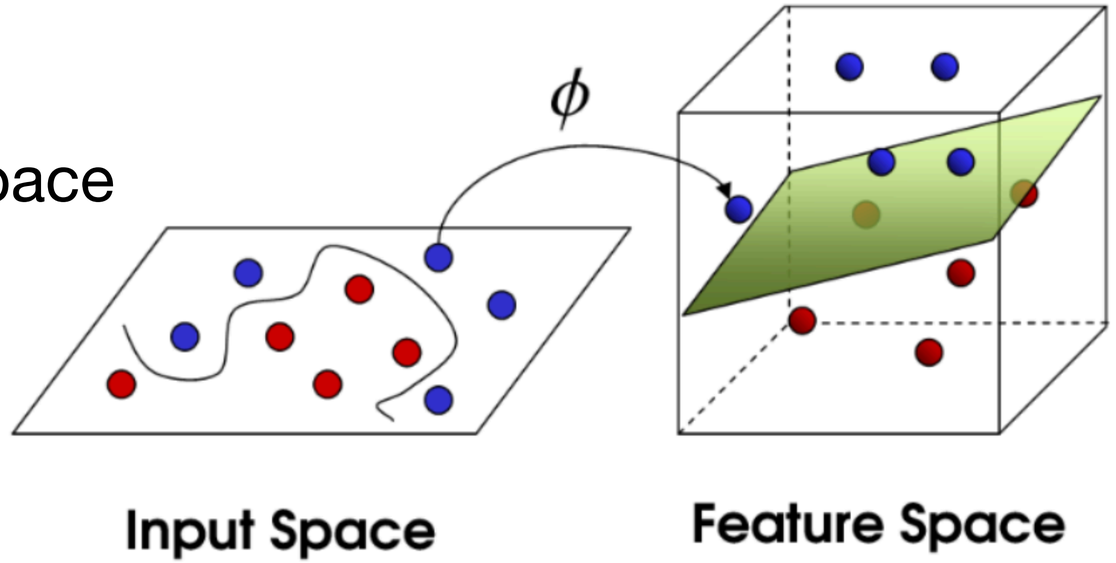
Inner product and non-linearities

Inner products can be generalised to deal with non-linearly separable data

1. Embed the data in an appropriate inner product space

non-linear map $\Phi : \mathcal{X} \mapsto \mathcal{H}$

Input Space Feature Space



2. Compute distances using inner products:

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle$$

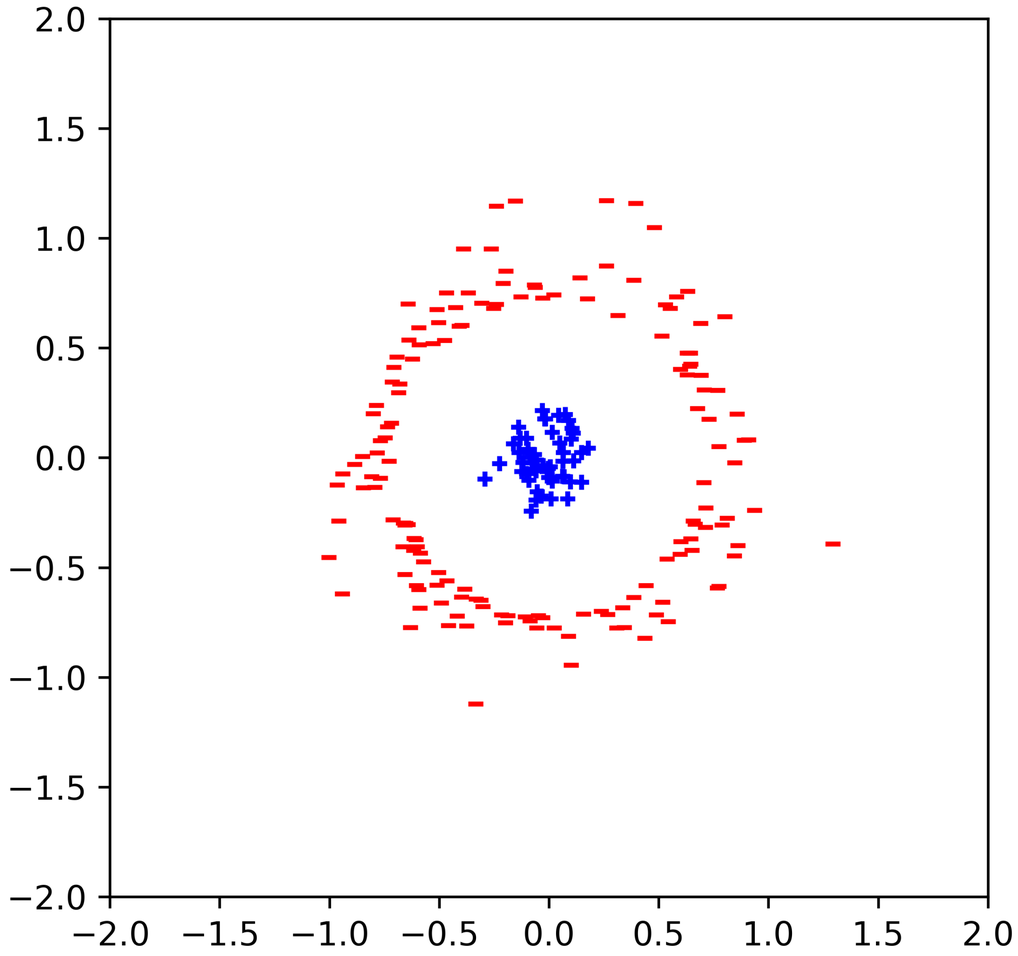
Kernel

Example: Data Lifting & Kernel Trick

Feature map:

$$\Phi : \mathbb{R}^2 \mapsto \mathbb{R}^3$$

$$\Phi(x) = [x_1^2 \; x_2^2 \; \sqrt{2}x_1x_2]^\top$$



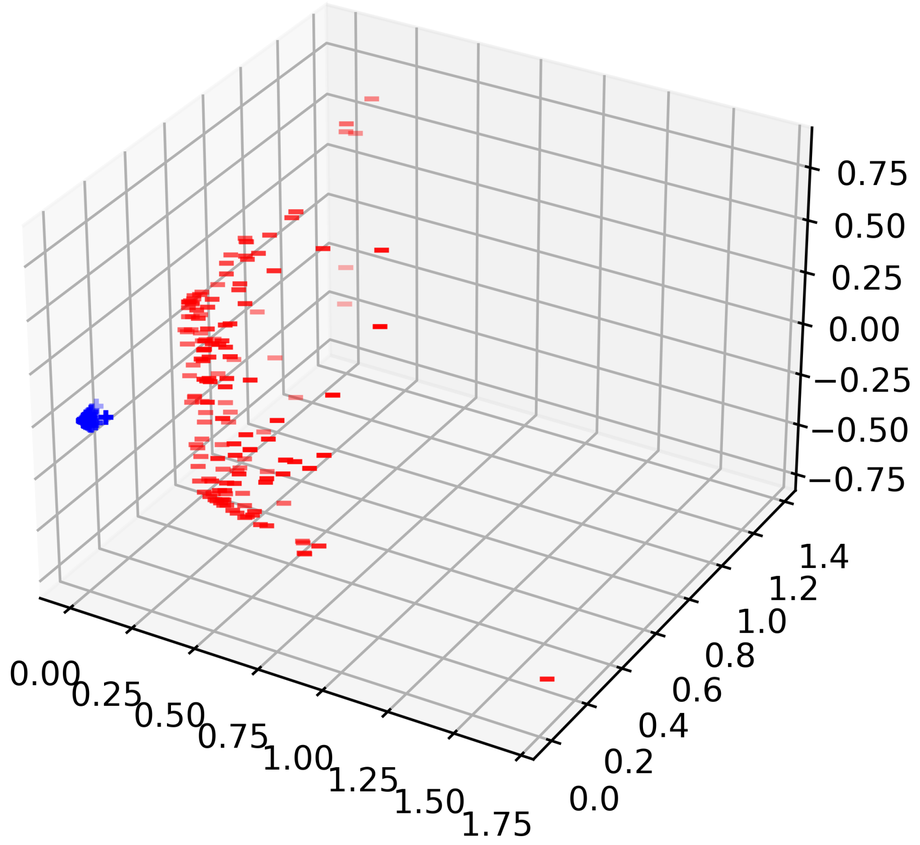
Non-linearly separable data in hypothesis space

Example: Data Lifting & Kernel Trick

Feature map:

$$\Phi : \mathbb{R}^2 \mapsto \mathbb{R}^3$$

$$\Phi(x) = [x_1^2 \; x_2^2 \; \sqrt{2}x_1x_2]^\top$$



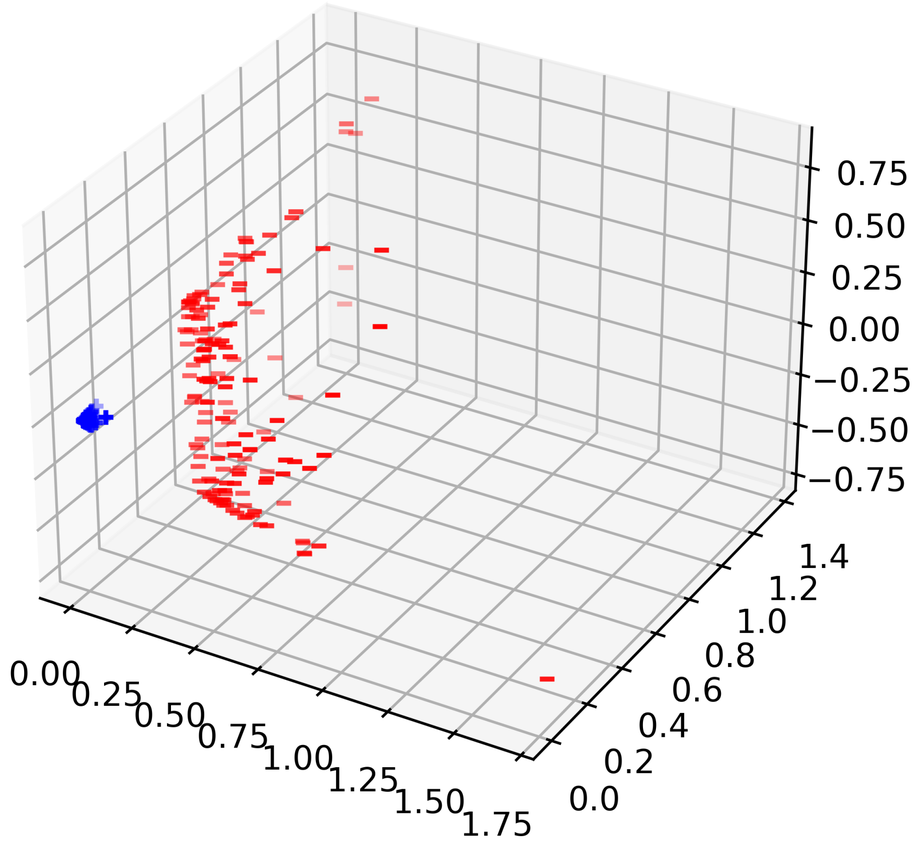
Data is linearly separable in feature space

Example: Data Lifting & Kernel Trick

Feature map:

$$\Phi : \mathbb{R}^2 \mapsto \mathbb{R}^3$$
$$\Phi(x) = [x_1^2 \; x_2^2 \; \sqrt{2}x_1x_2]^\top$$

Do we always need to compute Φ?



Data is linearly separable in feature space

Example: Data Lifting & Kernel Trick

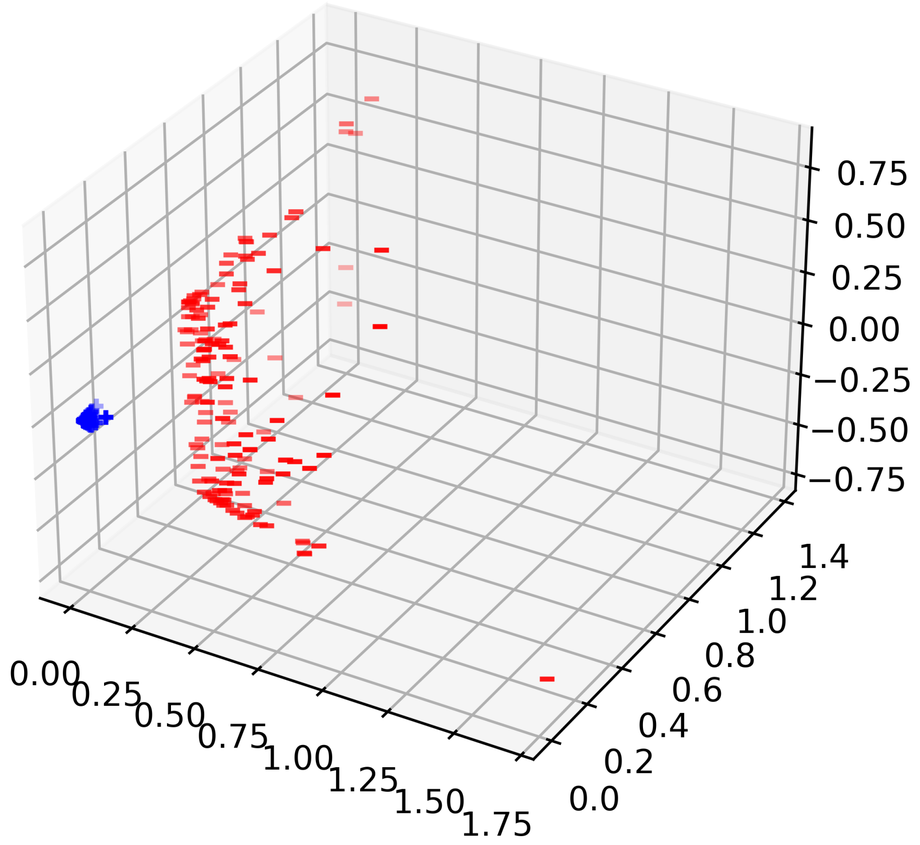
Feature map:

$$\Phi : \mathbb{R}^2 \mapsto \mathbb{R}^3$$
$$\Phi(x) = [x_1^2 \; x_2^2 \; \sqrt{2}x_1x_2]^\top$$

Do we always need to compute Φ?

For this example we note:

$$\langle \Phi(x), \Phi(y) \rangle = \langle x, y \rangle^2 = k(x, y)$$



Data is linearly separable in feature space

Example: Data Lifting & Kernel Trick

Feature map:

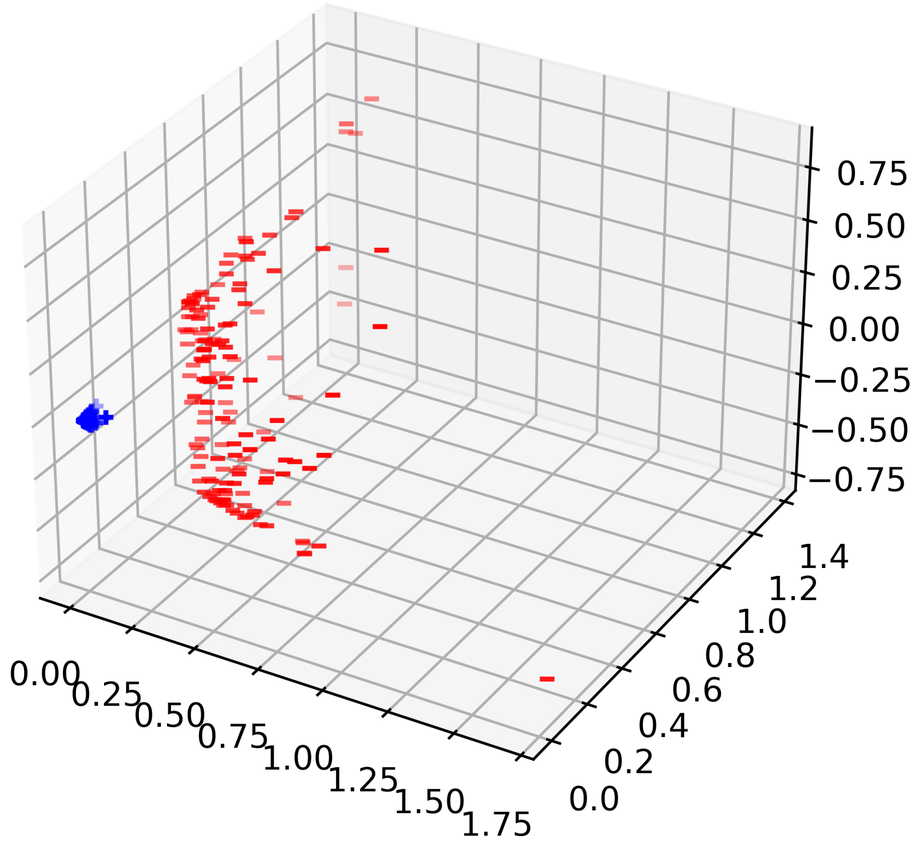
$$\Phi : \mathbb{R}^2 \mapsto \mathbb{R}^3$$
$$\Phi(x) = [x_1^2 \; x_2^2 \; \sqrt{2}x_1x_2]^\top$$

Do we always need to compute Φ?

For this example we note:

$$\langle \Phi(x), \Phi(y) \rangle = \langle x, y \rangle^2 = k(x, y)$$

“Kernel trick”: avoid expensive operations in higher dimensions by finding an appropriate kernel



Data is linearly separable in feature space

Example: Data Lifting & Kernel Trick

Feature map:

$$\Phi : \mathbb{R}^2 \mapsto \mathbb{R}^3$$
$$\Phi(x) = [x_1^2 \; x_2^2 \; \sqrt{2}x_1x_2]^\top$$

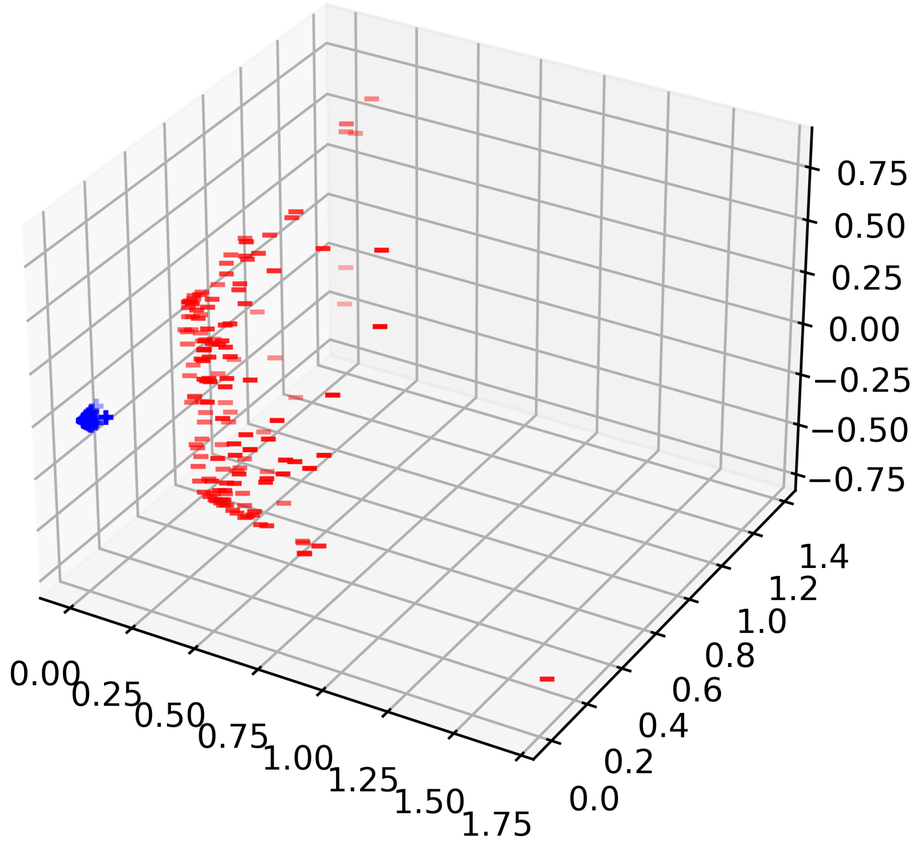
Do we always need to compute Φ?

For this example we note:

$$\langle \Phi(x), \Phi(y) \rangle = \langle x, y \rangle^2 = k(x, y)$$

“Kernel trick”: avoid expensive operations in higher dimensions by finding an appropriate kernel

Is it always possible to do this?
Yes! (Mercer’s Theorem)



Data is linearly separable in feature space

Mercer’s and Cover’s Theorems

(Mercer's Theorem) A symmetric function can be written as the inner product

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle$$

if and only if k is positive semidefinite i.e.

$$\int k(x, y) g(x) g(y) dx dy \geq 0$$

Cover’s Theorem: As the dimension of the feature space increase, so does the probability of having linearly separable data.

Practical implication: increase the dimension of the feature space to make the data linearly separable and use the kernel trick to manage computation

Applications: support vector machines, kernel PCA, Bayesian Kernel Methods, Regularised Kernel Methods, and so much more!

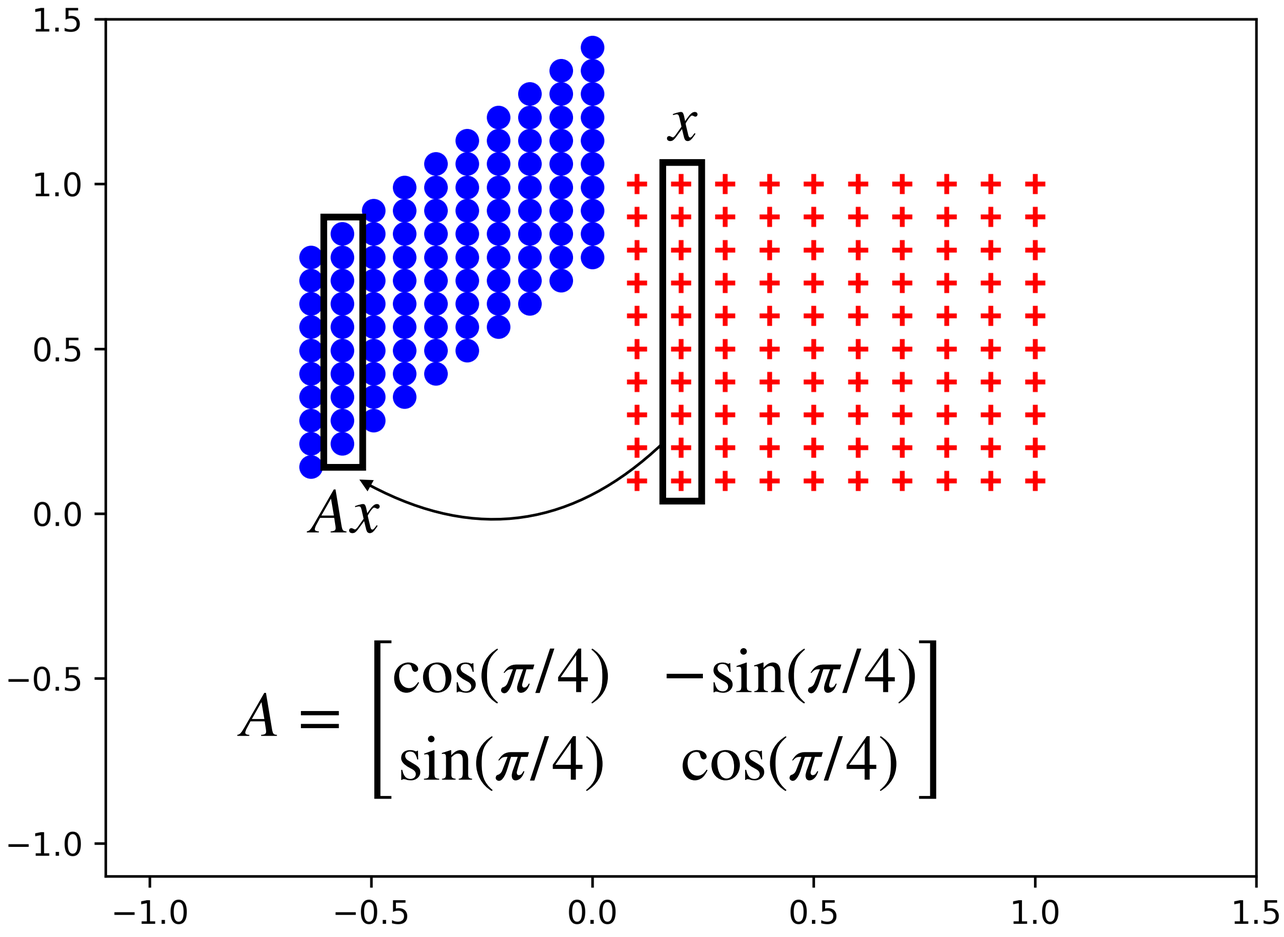
Matrix Decompositions/Factorisations

Tools:

- SVD (Singular Value Decomposition)
- Cholesky Decomposition
- Eigenvalue Decomposition

Applications:

- Dimensionality reduction
- Data scaling
- Simulation
- Preconditioning
- Recommender systems (netflix)
- Pagerank (Google)



$A \in \mathbb{R}^{N \times d}$ performs linear transformations on vectors in $\mathbb{R}^d$

Eigenvalues/Eigenvectors

Let $A \in \mathbb{R}^{n \times n}$ be a square matrix then $\lambda \in \mathbb{R}$ is an eigenvalue and $x \in \mathbb{R}^n \backslash 0$ the associated eigenvector:

$$Ax = \lambda x$$

Useful facts:

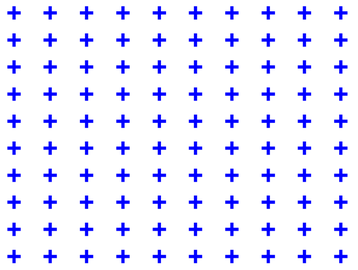
- The number of non-zero eigenvalues == number of linearly independent row/columns/rank
- Trace of matrix: $\text{trace}(A) = \sum_{i=1}^{n} A_{ii} = \sum_{i=1}^{n} \lambda_i$
- Determinant of matrix (measures volume)

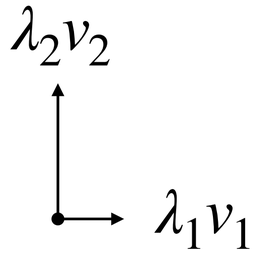
$$\det(A) = \prod_{i=1}^{n} A_{ii} = \prod_{i=1}^{n} \lambda_i$$

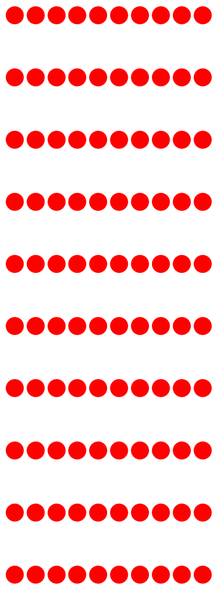
Intuition: Eigenvalues/Eigen Vectors

$$A_1 = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 2 \end{bmatrix}$$

$$\lambda_1 = \frac{1}{2}, \lambda_2 = 2, v_1 = [1,0]^\top v_2 = [0,1]^\top .$$







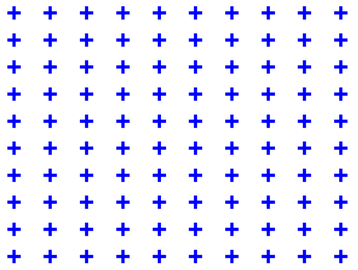
Eigenvectors are canonical- i.e. mapping changes each direction

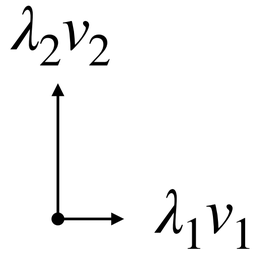
Area preserving. i.e determinant is the same

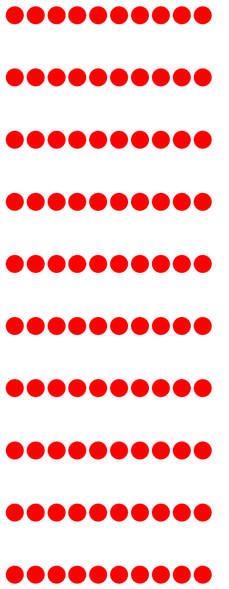
Intuition: Eigenvalues/Eigen Vectors

$$A_1 = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 2 \end{bmatrix}$$

$$\lambda_1 = \frac{1}{2}, \lambda_2 = 2, v_1 = [1,0]^\top v_2 = [0,1]^\top .$$



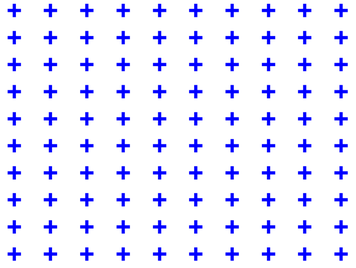


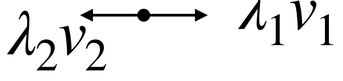


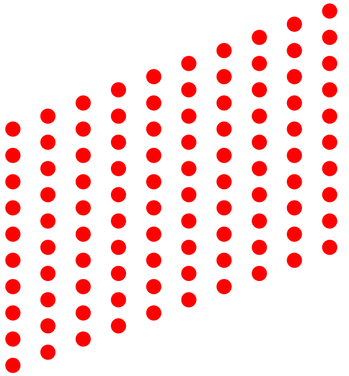
Eigenvectors are canonical- i.e. mapping changes each direction
Area preserving. i.e determinant is the same

$$A_2 = \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & 1 \end{bmatrix}$$

$$\lambda_1 = 1, \lambda_2 = 1, v_1 = [1,0]^\top v_2 = [-1,0]^\top .$$



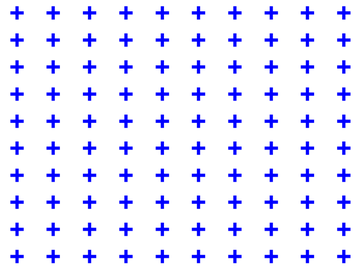


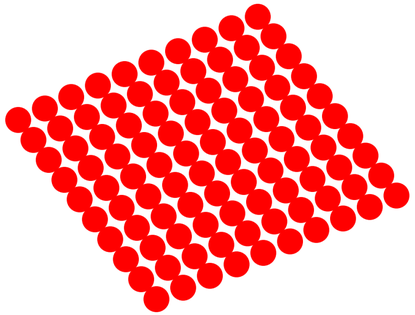


Eigenvectors are co-linear i.e. mapping changes only one direction
(Area preserving)

Intuition: Eigenvalues/Eigen Vectors

$$A_3 = \begin{bmatrix} \cos(\frac{\pi}{6}) & -\sin(\frac{\pi}{6}) \\ \sin(\frac{\pi}{6}) & \cos(\frac{\pi}{6}) \end{bmatrix}$$



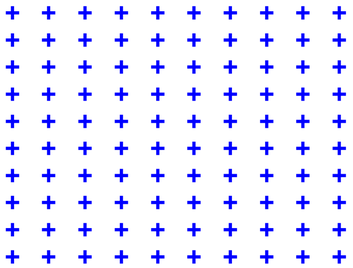


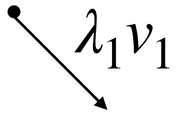
$\lambda_1 = 0.87 + 0.5j, \lambda_2 = 0.87 - 0.5j$

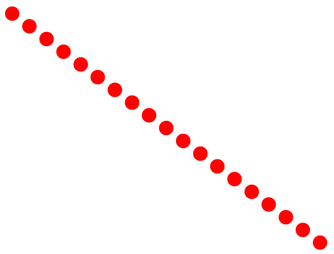
$v_1 = [0.71, -0.71j]^\top v_2 = [0.71, 0.7j]^\top .$

Eigenvalues are complex (rotation)

$$A_4 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$





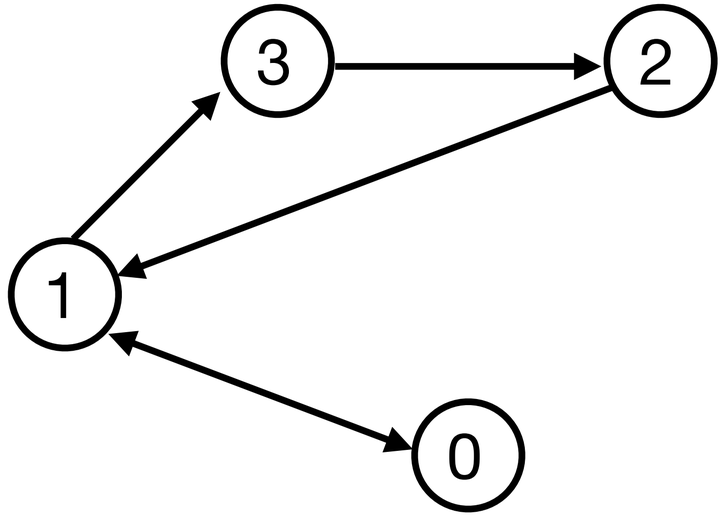


$\lambda_1 = 2, \lambda_2 = 0$

$v_1 = [0.71, -0.71]^\top v_2 = [0.71, 0.71]^\top .$

A zero eigenvalue means the direction in v_2 collapses (the matrix has linearly dependent rows). The determinant becomes 0, matrix/transformation is not invertible.

Computing Eigenvalues: Power Iteration



Graph view of the “web”

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array} = A$$

Columns of **adjacency matrix** are out-going edges from node

Link from node 3 to 2

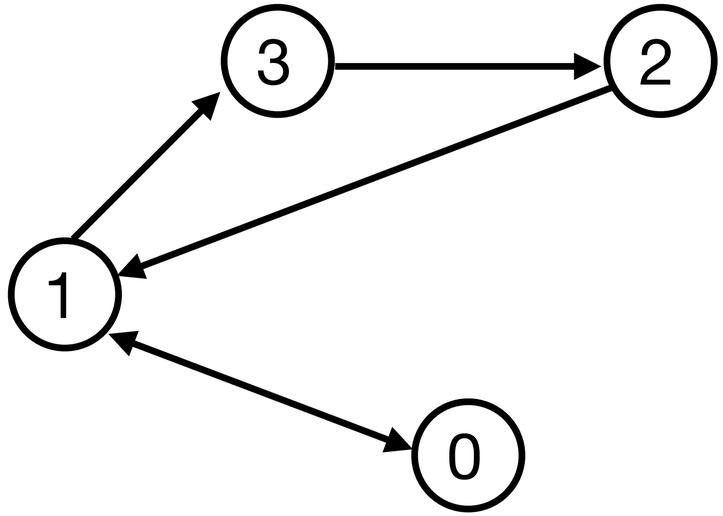
$$D = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Degree matrix: diagonal matrix of sum of rows

$$M = AD^{-1} = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1/2 & 0 & 0 \end{bmatrix}$$

Transition matrix: column stochastic matrix for a random surfer

Computing Eigenvalues: Power Iteration



Graph view of the “web”

$$M = \begin{bmatrix} 0 & 1/2 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1/2 & 0 & 0 \end{bmatrix}$$

Transition matrix: column stochastic matrix for a random surfer e.g. if on page 1 then 0.5 probability to go to 0 or 3

Power Iteration: $v_{t+1} = Mv_t$

After a few iterations v converges to:

$$[0.2, 0.4, 0.2, 0.2]^\top$$

Cholesky Decomposition*

A symmetric positive definite matrix can be factorized into a product $A = LL^{\top}$ where L is a lower triangular matrix with positive diagonal elements

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ l_{n1} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & \cdots & l_{n1} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & l_{nn} \end{bmatrix}$$

L is called the Cholesky factor of A and is unique.

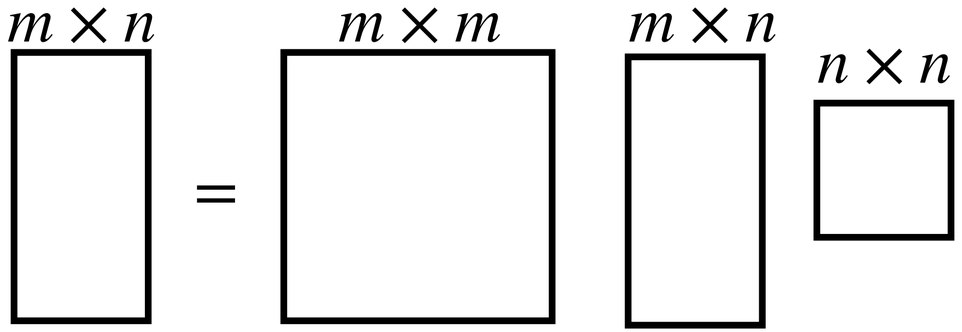
Applications:

Factorisation of Gaussian random variables
Solving linear systems of equations

* A square root for matrices. A special case of LU decomposition which applies to all square matrices

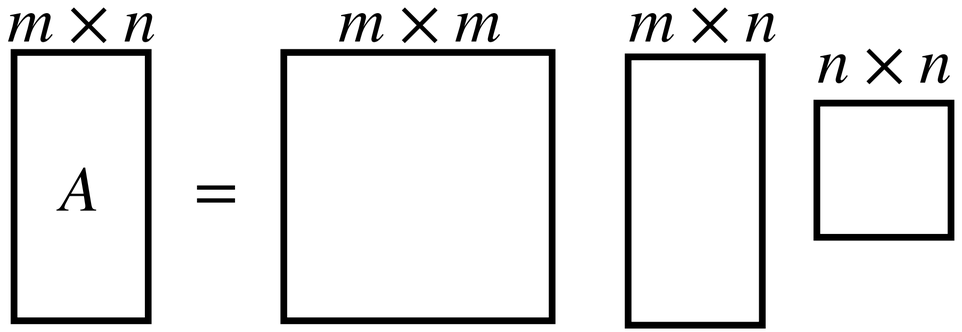
Singular Value Decomposition (SVD)

For a matrix $A \in \mathbb{R}^{m \times n}$ rank $r = \min\{m, n\}$ then SVD is of the form $A = USV^\top$



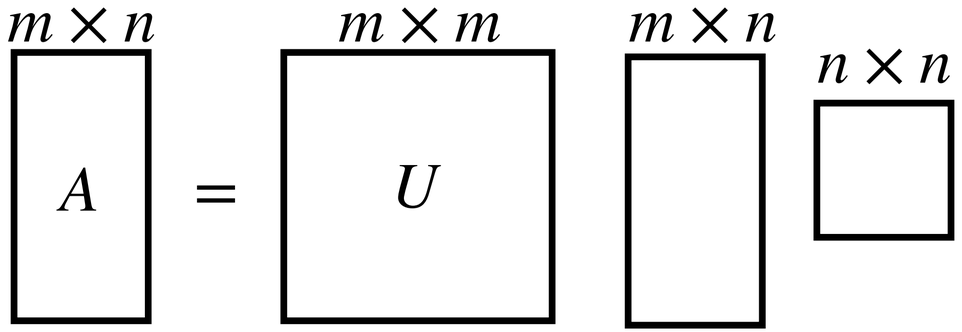
Singular Value Decomposition (SVD)

For a matrix $A \in \mathbb{R}^{m \times n}$ rank $r = \min\{m, n\}$ then SVD is of the form $A = USV^{\top}$



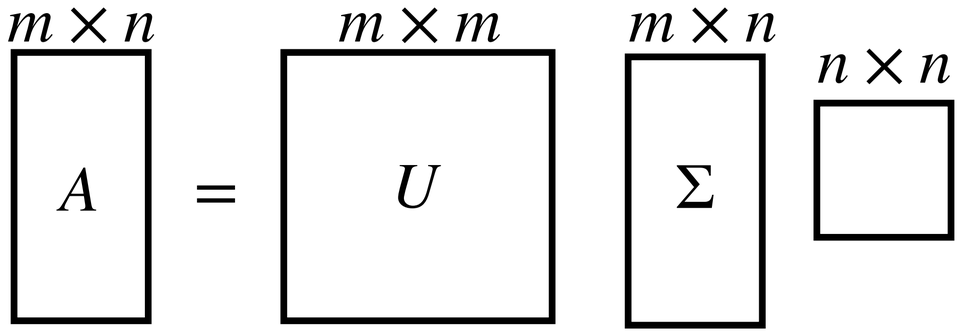
Singular Value Decomposition (SVD)

For a matrix $A \in \mathbb{R}^{m \times n}$ rank $r = \min\{m, n\}$ then SVD is of the form $A = USV^\top$



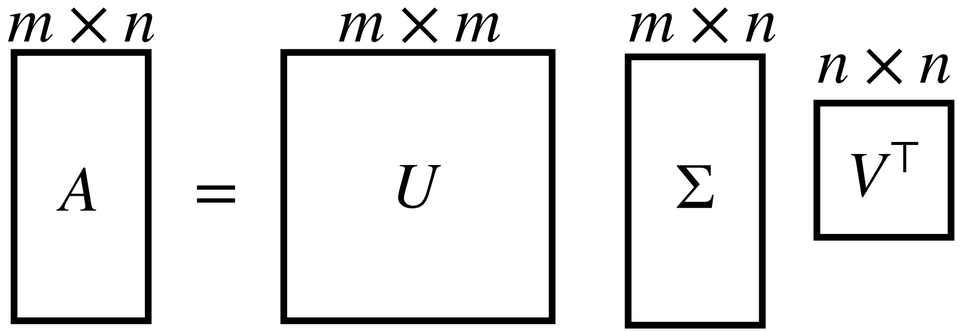
Singular Value Decomposition (SVD)

For a matrix $A \in \mathbb{R}^{m \times n}$ rank $r = \min\{m, n\}$ then SVD is of the form $A = USV^\top$



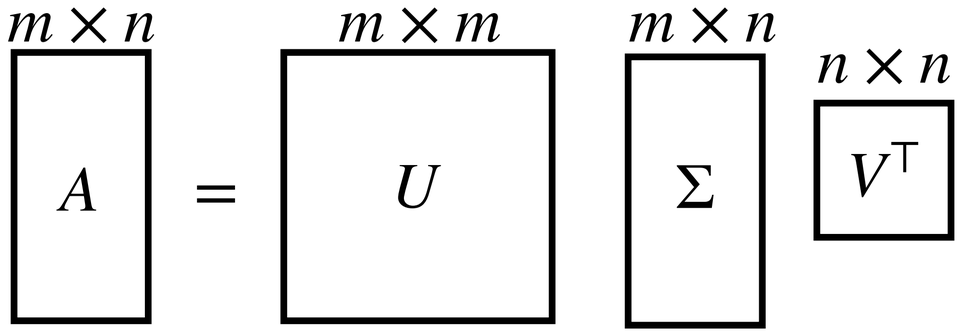
Singular Value Decomposition (SVD)

For a matrix $A \in \mathbb{R}^{m \times n}$ rank $r = \min\{m, n\}$ then SVD is of the form $A = USV^\top$



Singular Value Decomposition (SVD)

For a matrix $A \in \mathbb{R}^{m \times n}$ rank $r = \min\{m, n\}$ then SVD is of the form $A = USV^\top$



where $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices and

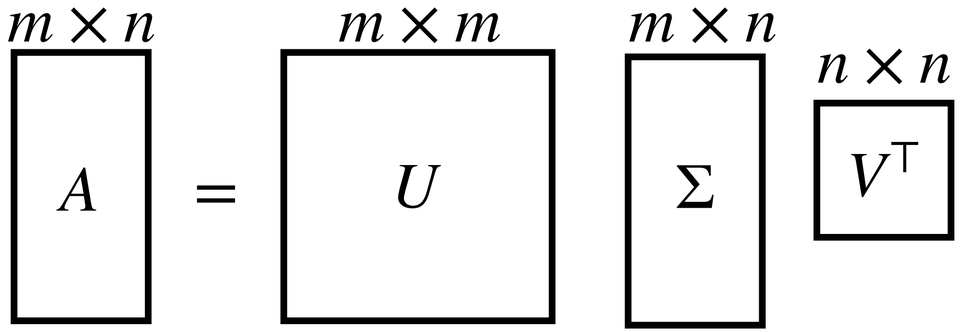
Σ has entries $\Sigma_{ii} = \sigma_i \geq 0$ and $\Sigma_{ij} = 0$ if $i \neq j$.

The columns of $U(V)$ are called the left (right) singular vectors of A.

The σ_i are called the singular values of A.

Singular Value Decomposition (SVD)

For a matrix $A \in \mathbb{R}^{m \times n}$ rank $r = \min\{m, n\}$ then SVD is of the form $A = USV^\top$



where $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices and

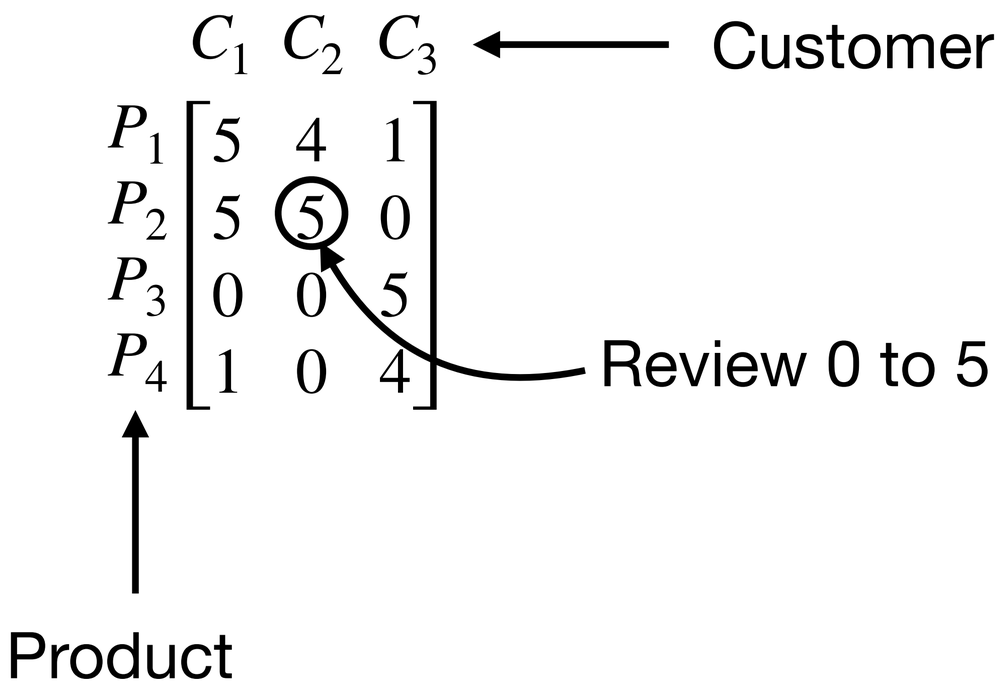
Σ has entries $\Sigma_{ii} = \sigma_i \geq 0$ and $\Sigma_{ij} = 0$ if $i \neq j$.

The columns of $U(V)$ are called the left (right) singular vectors of A.

The σ_i are called the singular values of A.

- SVD can be applied to **any** matrix (not just symmetric matrices), other definitions of SVD are sometimes used but equivalent to the one above
- **Extremely** useful in ML and is for dimensionality reduction (Principal Component Analysis), solving linear systems (linear regression), noise removal, outlier detection, feature engineering, etc…

SVD in Recommender Systems

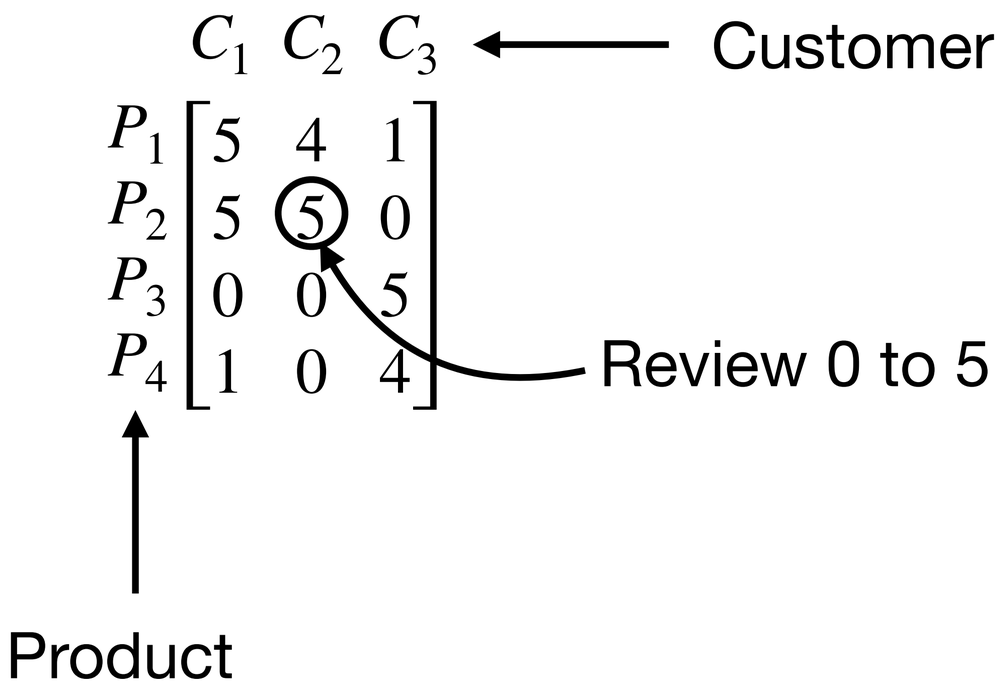


Some observations:

C_1 and C_2 appear to like similar products

P_3 and P_4 are liked by a single person

SVD in Recommender Systems



Some observations:

C_1 and C_2 appear to like similar products

P_3 and P_4 are liked by a single person

How do you automate these observations?

SVD captures the relationships between customers and products

SVD in Recommender Systems

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array}\begin{array}{c} \begin{array}{ccc} C_1 & C_2 & C_3 \end{array} \\ \begin{bmatrix} 5 & 4 & 1 \\ 5 & 5 & 0 \\ 0 & 0 & 5 \\ 1 & 0 & 4 \end{bmatrix} \end{array} = \begin{bmatrix} -0.6710 & 0.0236 & 0.4647 & -0.5774 \\ -0.7197 & 0.2054 & -0.4759 & 0.4619 \\ -0.0939 & -0.7705 & -0.5268 & -0.3464 \\ -0.1515 & -0.6030 & 0.5293 & -0.5774 \end{bmatrix}$$

Stereotypical Products

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

Singular values ‘mix’ singular vectors to reproduce data

Stereotypical Customers

$$\begin{bmatrix} -0.7367 & -0.6516 & -0.1811 \\ 0.0852 & 0.1762 & -0.9807 \\ 0.6708 & -0.7379 & -0.0743 \end{bmatrix}$$

SVD in Recommender Systems

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array}\begin{array}{c} \begin{array}{ccc} C_1 & C_2 & C_3 \end{array} \\ \begin{bmatrix} 5 & 4 & 1 \\ 5 & 5 & 0 \\ 0 & 0 & 5 \\ 1 & 0 & 4 \end{bmatrix} \end{array} = \begin{bmatrix} -0.6710 & 0.0236 & 0.4647 & -0.5774 \\ -0.7197 & 0.2054 & -0.4759 & 0.4619 \\ -0.0939 & -0.7705 & -0.5268 & -0.3464 \\ -0.1515 & -0.6030 & 0.5293 & -0.5774 \end{bmatrix}$$

Stereotypical Products
U

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

Singular values ‘mix’ singular vectors to reproduce data

Stereotypical Customers

$$\begin{bmatrix} -0.7367 & -0.6516 & -0.1811 \\ 0.0852 & 0.1762 & -0.9807 \\ 0.6708 & -0.7379 & -0.0743 \end{bmatrix}$$

SVD in Recommender Systems

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array}\begin{array}{c} \begin{array}{ccc} C_1 & C_2 & C_3 \end{array} \\ \begin{bmatrix} 5 & 4 & 1 \\ 5 & 5 & 0 \\ 0 & 0 & 5 \\ 1 & 0 & 4 \end{bmatrix} \end{array} = \begin{bmatrix} -0.6710 & 0.0236 & 0.4647 & -0.5774 \\ -0.7197 & 0.2054 & -0.4759 & 0.4619 \\ -0.0939 & -0.7705 & -0.5268 & -0.3464 \\ -0.1515 & -0.6030 & 0.5293 & -0.5774 \end{bmatrix}$$

Stereotypical Products

U

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

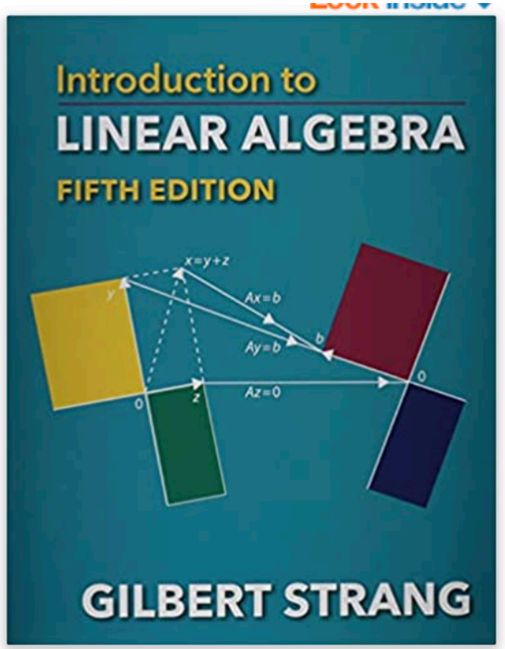
Singular values ‘mix’ singular vectors to reproduce data

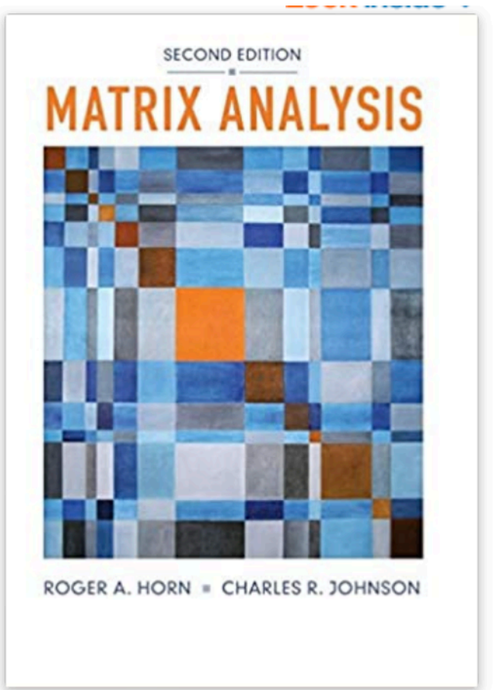
Stereotypical Customers

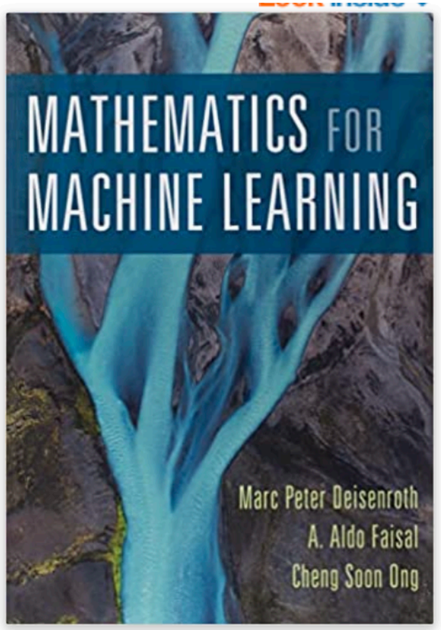
$$\begin{bmatrix} -0.7367 & -0.6516 & -0.1811 \\ 0.0852 & 0.1762 & -0.9807 \\ 0.6708 & -0.7379 & -0.0743 \end{bmatrix} V^\top$$

Linear Algebra Summary

- In ML we represent data as vectors
- Necessary to go beyond simple geometric vectors (but computation often reduces to this)
- Inner products measure distances
- If the space is “non-linear” kernels can help
- Matrix factorisations are used to give insights and reduce dimensions







Calculus Background

Gradient of $L : \mathbb{R}^d \mapsto \mathbb{R}$

$$\nabla L(w) = \begin{bmatrix} \frac{\partial L(w)}{\partial w_1} & \cdots & \frac{\partial L(w)}{\partial w_d} \end{bmatrix}$$

Hessian of $L : \mathbb{R}^d \mapsto \mathbb{R}$

$$\nabla^2 L(w) = \begin{bmatrix} \frac{\partial^2 L(w)}{\partial w_1^2} & \cdots & \frac{\partial^2 L(w)}{\partial w_1 \partial w_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 L(w)}{\partial w_1 \partial w_d} & \cdots & \frac{\partial^2 L(w)}{\partial w_d \partial w_d} \end{bmatrix}$$

We use the convention that the gradient is a column vector i.e $\nabla L \in \mathbb{R}^{1 \times d}$

Gradient of $f(\mathbf{x}) = x_1^2 + x_1x_2 + x_3x_2 + x_3^2$

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x1}\frac{\partial f(\mathbf{x})}{\partial x2}\frac{\partial f(\mathbf{x})}{\partial x3}\right] = \left[2x_1 + x_2x_1 + x_3x_2 + 2x_3\right]$$

Hessian of $f(\mathbf{x}) = x_1^2 + x_1x_2 + x_3x_2 + x_3^2$

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x}))}{\partial x_1 \partial x_2} & \frac{\partial^2 f(\mathbf{x}))}{\partial x_1 \partial x_3} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 x_2} & \frac{\partial^2 f(\mathbf{x}))}{\partial x_2^2} & \frac{\partial^2 f(\mathbf{x}))}{\partial x_2 \partial x_3} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_1 x_3} & \frac{\partial^2 f(\mathbf{x}))}{\partial x_2 x_3} & \frac{\partial^2 f(\mathbf{x}))}{\partial x_3^2} \end{bmatrix} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Differentiation Rules (1d)

- Sum Rule

$$\big(f(x)+g(x)\big)' = f'(x)+g'(x) = \frac{df(x)}{dx} + \frac{dg(x)}{dx}$$

- Product Rule

$$\big(f(x)g(x)\big)' = f'(x)g(x)+f(x)g'(x) = \frac{df(x)}{dx}g(x) + f(x)\frac{dg(x)}{dx}$$

- Chain Rule

$$(g\circ f)'(x) = \big(g(f(x))\big)' = g'(f(x))f'(x) = \frac{dg(f(x))}{df}\frac{df(x)}{dx}$$

- Quotient Rule

$$\left(\frac{f(x)}{g(x)}\right)' = \frac{f(x)'g(x)-f(x)g(x)'}{(g(x))^2} = \frac{\frac{df}{dx}g(x)-f(x)\frac{dg}{dx}}{(g(x))^2}$$

Chain Rule Example

$$(g \circ f)'(x) = \big(g(f(x))\big)' = g'(f(x))f'(x) = \frac{dg}{df}\frac{df}{dx}$$

$$\begin{aligned} g(z) &= 6z + 3 \\ z = f(x) &= -2x + 5 \\ (g \circ f)'(x) &= \underbrace{(6)}_{dg/df}\underbrace{(-2)}_{df/dx} \\ &= -12 \end{aligned}$$

$$\begin{aligned} g(z) &= \tanh(z) \\ z = f(x) &= x^n \\ (g \circ f)'(x) &= \underbrace{(1 - \tanh^2(x^n))}_{dg/df}\underbrace{nx^{n-1}}_{df/dx} \end{aligned}$$

Vector Differentiation — Index Notation

Consider the multivariate Linear Regression problem

$$L(\boldsymbol{\theta}) = \sum_{n=1}^{N} (y_n - \boldsymbol{\phi}(x_n)^\top \boldsymbol{\theta})^2 = ||\mathbf{y} - \Phi(X)\boldsymbol{\theta}||^2$$

Suppose $\theta \in \mathbb{R}^d$, what is $\nabla L(\theta)$?

We could write all the expressions in index notation

Then *carefully* differentiate every term

Most ML algorithms/software rely on *multivariate* chain rule

But let's do the slow way first!

Index Notation — Examples

$$|\mathbf{x}|^2 = \sum_i x_i^2$$

$$\mathbf{x}^\mathsf{T}\mathbf{y} = \sum_i x_i y_i$$

$$\mathbf{y} = f(\mathbf{x})\mathbf{x} \implies y_i = f(\mathbf{x})x_i$$

$$\boldsymbol{A}\mathbf{x} = ?$$

$$[\boldsymbol{A}\mathbf{x}]_j = \sum_j A_{ij}x_j$$

$$\boldsymbol{A}\mathbf{x} = \left[\sum_j A_{ij}x_j\right]_i$$

Index Notation — Linear Regression

$$L(\boldsymbol{\theta}) = \sum_{n=1}^{N} (y_n - \boldsymbol{\phi}(x_n)^\top \boldsymbol{\theta})^2 = ||\mathbf{y} - \Phi(X)\boldsymbol{\theta}||^2$$

$$= \sum_{n=1}^{N} \left(y_n - \sum_{m=1}^{M} \phi_m(x_n)\theta_m \right)^2$$

$$[\nabla L]_i = \frac{\partial L}{\partial \theta_i} = \frac{\partial}{\partial \theta_i} \sum_{n=1}^{N} \left(y_n - \sum_{m=1}^{M} \phi_m(x_n)\theta_m \right)^2$$

$$= \sum_{n} 2 \left(y_n - \sum_{m=1}^{M} \phi_m(x_n)\theta_m \right) \frac{\partial}{\partial \theta_i} \left(y_n - \sum_{k=1}^{M} \phi_k(x_n)\theta_k \right)$$

Index Notation — Linear Regression

$$\frac{\partial L}{\partial \theta_i} = -2 \sum_n \left(y_n - \sum_{m=1}^{M} \phi_m(x_n)\theta_m \right) \sum_k \phi_k(x_n) \frac{\partial \theta_k}{\partial \theta_i}$$

$$\frac{\partial L}{\partial \theta_i} = -2 \sum_n \left(y_n - \sum_{m=1}^{M} \phi_m(x_n)\theta_m \right) \sum_k \phi_k(x_n) \delta_{ki}$$

$$= -2 \sum_n \left(y_n - \sum_{m=1}^{M} \phi_m(x_n)\theta_m \right) \phi_i(x_n)$$

That was a lot of work! But there is an easier way, which turns out to generalise well to other types of models e.g. deep neural networks

Linear Regression — Multivariate Chain Rule

$$L(\boldsymbol{\theta}) = \sum_{n=1}^{N} (y_n - \boldsymbol{\phi}(x_n)^\top \boldsymbol{\theta})^2 = ||\mathbf{y} - \Phi(X)\boldsymbol{\theta}||^2$$

Instead of seeing this as a multivariate function we could write it as the composition of a scalar function with a multivariate function:

$f(g(\boldsymbol{\theta}))$ with $f : \mathbb{R}^D \mapsto \mathbb{R}$ and $g : \mathbb{R}^E \mapsto \mathbb{R}^D$

If we have a chain rule like the one below, computations would simplify a lot!

$$\frac{\partial f(g(\boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial \boldsymbol{\theta}}$$

But how do we compute a derivative wrt another vector?

Differentiation wrt a scalar

Revisit the definition of derivative

$$x : \mathbb{R} \mapsto \mathbb{R}^d$$

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{\mathbf{x}(t + \Delta t) - \mathbf{x}(t)}{\Delta t}$$

Compute each element using index notation:

$$\left[\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}\right]_i = \lim_{\Delta t \to 0} \frac{x_i(t + \Delta t) - x_i(t)}{\Delta t}$$

Using our convention: $\frac{d\mathbf{x}}{dt} \in \mathbb{R}^{d \times 1}$

Multivariate Chain Rule w.r.t. scalar

There is a multivariate chain rule and using our convention it looks like an inner product!

Index notation:

$$\frac{df(\mathbf{g}(t))}{dt} = \sum_{i=1}^{D} \frac{\partial f}{\partial g_i} \frac{\partial g_i}{\partial t}$$

Vector form:

$$\frac{df(\mathbf{g}(t))}{dt} = \frac{\partial f}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial t}$$

Row

Column

Multivariate Chain Rule w.r.t. scalar — Example

Consider $f : \mathbb{R}^2 \to \mathbb{R}, \quad \boldsymbol{x} : \mathbb{R} \to \mathbb{R}^2$

$$f(\boldsymbol{x}) = f(x_1, x_2) = x_1^2 + 2x_2\,,$$

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}$$

What are the dimensions of $\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}$ and $\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}$?

1×2 and 2×1

Compute the gradient $\frac{\mathrm{d}f}{\mathrm{d}t}$ using the chain rule:

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{bmatrix} = \begin{bmatrix} 2\sin t & 2 \end{bmatrix} \begin{bmatrix} \cos t \\ -\sin t \end{bmatrix}$$

$$= 2\sin t \cos t - 2\sin t = 2\sin t(\cos t - 1)$$

Derivative of a vector wrt vector

We saw the chain rule if we were differentiating w.r.t. a scalar:

$$\frac{\mathrm{d}f(\boldsymbol{g}(t))}{\mathrm{d}t} = \sum_{i=1}^{D} \frac{\partial f}{\partial g_i} \frac{\mathrm{d}g_i}{\mathrm{d}t} = \underbrace{\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{g}}}_{\text{row}} \cdot \underbrace{\frac{\mathrm{d}\boldsymbol{g}}{\mathrm{d}t}}_{\text{column}} \qquad \boldsymbol{g}(t) \in \mathbb{R}^D$$

What happens if we differentiate w.r.t. a vector?

$\implies$ As before, we just stack the derivatives w.r.t. each of the inputs.

$$\frac{\partial f(\boldsymbol{g}(\mathbf{x}))}{\partial x_j} = \sum_{i=1}^{D} \frac{\partial f}{\partial g_i} \frac{\partial g_i}{\partial x_j} \qquad \boldsymbol{g}(\mathbf{x}) \in \mathbb{R}^D$$

This is a matrix multiplication! Can write in vector form:

$$\frac{\mathrm{d}f(\boldsymbol{g}(\mathbf{x}))}{\mathrm{d}\mathbf{x}} = \underbrace{\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{g}}}_{\text{row}} \cdot \underbrace{\frac{\mathrm{d}\boldsymbol{g}}{\mathrm{d}\mathbf{x}}}_{\text{matrix}}$$

Derivative of a vector wrt vector

$\mathbf{f} : \mathbb{R}^N \mapsto \mathbb{R}^M$

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} f_1(x_1, \ldots, x_N) \\ \vdots \\ f_M(x_1, \ldots, x_N) \end{bmatrix}$$

- Jacobian matrix (collection of all partial derivatives)

$$\begin{bmatrix} \frac{dy_1}{d\boldsymbol{x}} \\ \vdots \\ \frac{dy_M}{d\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix} \in \mathbb{R}^{M \times N}$$

Dimensions of Derivative of a vector wrt vector

In general: A function $\boldsymbol{f} : \mathbb{R}^{N} \rightarrow \mathbb{R}^{M}$ has a gradient that is an $M \times N$-matrix with

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \in \mathbb{R}^{M \times N}, \qquad \mathrm{d}\boldsymbol{f}[m,n] = \frac{\partial f_m}{\partial x_n}$$

Gradient dimension: # target dimensions $\times$ # input dimensions

Example

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}, \qquad \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M, \quad \boldsymbol{A} \in \mathbb{R}^{M\times N}, \quad \boldsymbol{x} \in \mathbb{R}^N$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + & \cdots & + A_{1N}x_N \\ \vdots & \vdots & \vdots \\ A_{M1}x_1 + A_{M2}x_2 + & \cdots & + A_{MN}x_N \end{bmatrix}$$

- Compute the gradient $\frac{\mathrm{d}f}{\mathrm{d}x}$
  - Gradient:

$$f_i(\boldsymbol{x}) = \sum_{k=1}^{N} A_{ik}x_k \qquad \Longrightarrow \quad \frac{\partial f_i}{\partial x_j} = \sum_k A_{ik}\frac{\partial x_k}{\partial x_j} = \sum_k A_{ik}\delta_{kj} = A_{ij}$$

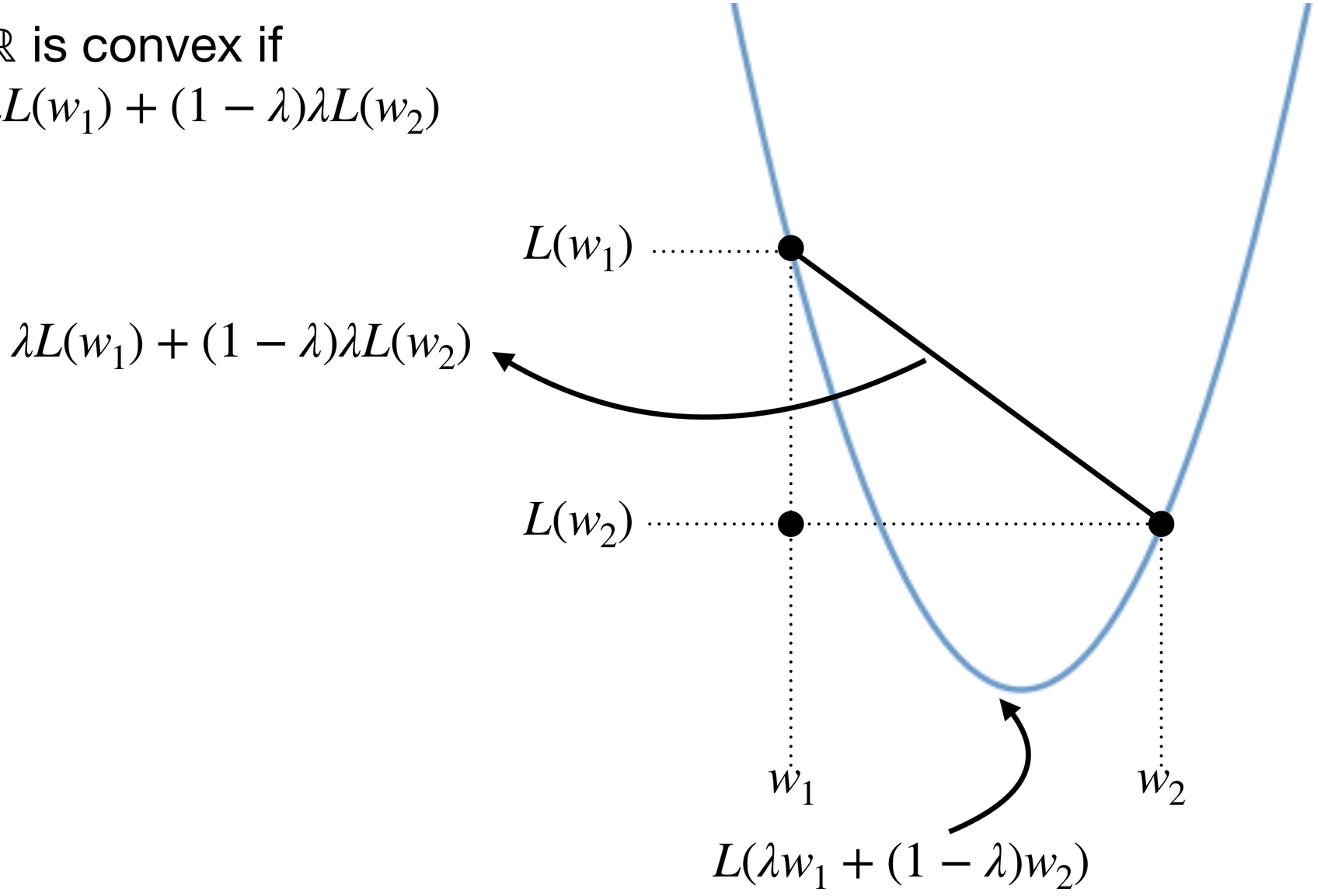
$$\Longrightarrow \quad \frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & & \vdots \\ A_{M1} & \cdots & A_{MN} \end{bmatrix} = \boldsymbol{A} \in \mathbb{R}^{M\times N}$$

Convex Functions

A function $L : \mathbb{R}^d \mapsto \mathbb{R}$ is convex if

$$L(\lambda w_1 + (1-\lambda)w_2) \leq \lambda L(w_1) + (1-\lambda)\lambda L(w_2)$$

$$0 \leq \lambda \leq 1$$

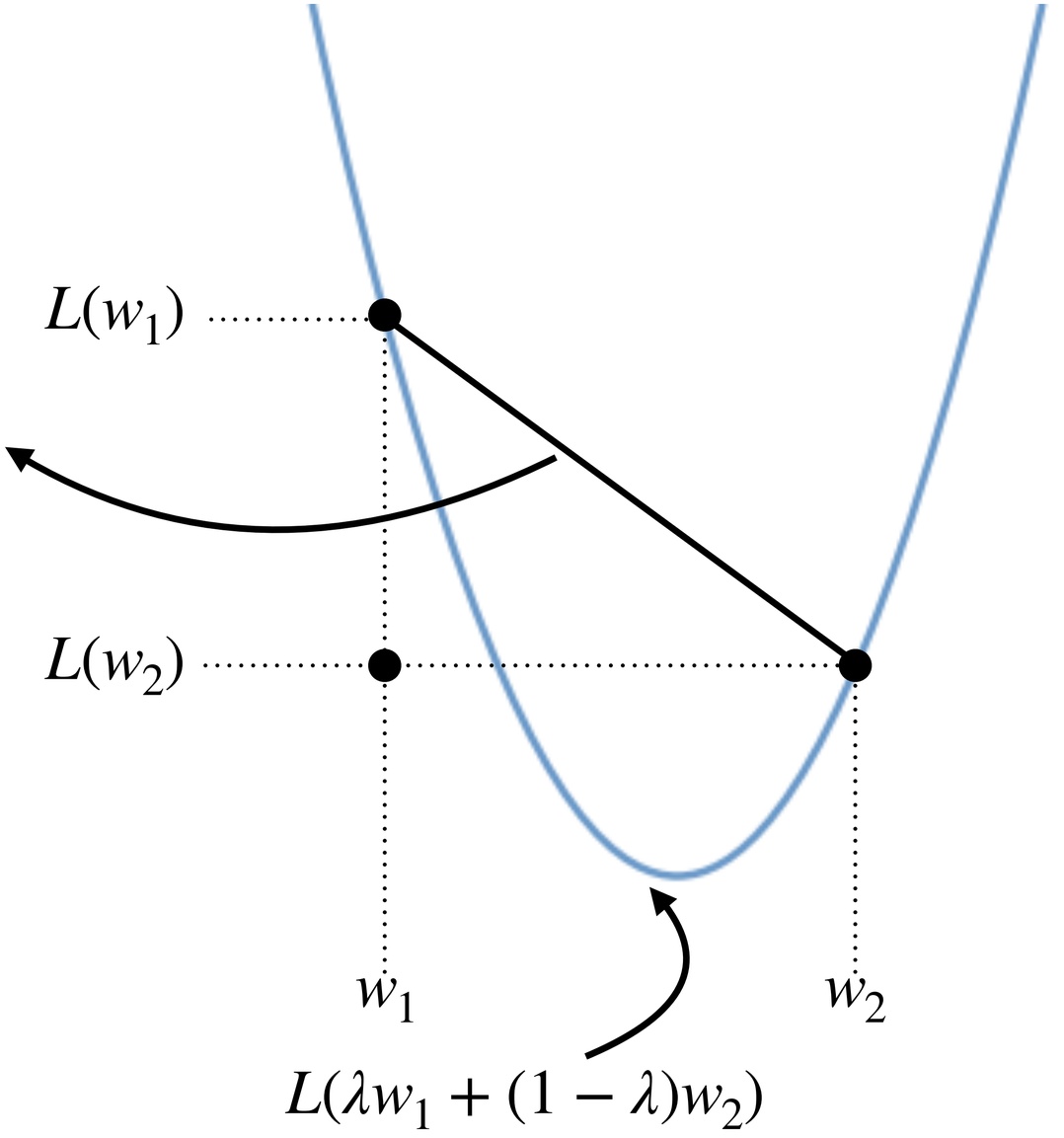


Convex Functions

A function $L : \mathbb{R}^d \mapsto \mathbb{R}$ is convex if

$$L(\lambda w_1 + (1-\lambda)w_2) \leq \lambda L(w_1) + (1-\lambda)\lambda L(w_2)$$

$$0 \leq \lambda \leq 1$$



Useful test for convexity:

$\nabla^2 L(w)$ is positive definite

Convex functions have **unique** minimum

Examples in ML: regression, SVMs are convex, NNs typically are not

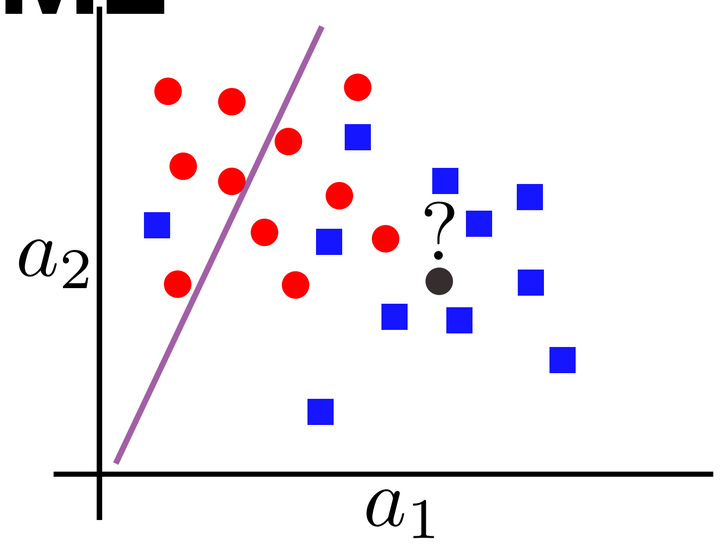
Optimization is everywhere in ML

Typical ML models have parameters to be tuned e.g.

$$\min_{w} \|Aw - b\|^2 + \|w\|_2^2$$

Data **fidelity** term

Regularisation term



Even complex ML models are an optimisation problem, e.g. (DNNs have a complicated objective function but are just a huge optimisation problem)

Other **applications** of optimisation:

- **Reinforcement learning** (branch of dynamic optimisation under uncertainty)
- **Variational Inference/Sampling** (e.g. Bayesian optimisation)
- Mathematical **foundations** of ML (e.g. decision trees, more on this in the next lecture!)

Optimization is everywhere in ML

ML projects “typical” steps :

- **Understand** the data (e.g. feature engineering, dimensionality reduction)
- Model **development** (e.g. choosing the architecture of a NN)
- **Optimising** the model

Fortunately, most ML libraries have state-of-the-art optimisation algorithms

Unlike linear algebra algorithms, they require more care to use

Knowledge of optimisation is necessary to **develop novel ML models and solve them**

Classes of Optimization Models

- **Unconstrained Optimization:** $\min_{w \in \mathbb{R}^d} L(w)$ ← Select parameters over all d-dim vectors

  (most ML models are unconstrained, e.g. linear regression, NNs)

- **Constrained Optimization** $\min_{w \in \mathscr{W}} L(w)$ ← Select parameters from a set

  (e.g. support vector machines)

  - **Convex Optimization** (convex constraints and loss function, easier to solve and interpret, unique solution, e.g. linear regression, SVMs)
  - **Non-convex Optimization** (hard to solve, multiple solutions, e.g. RL, NNs (e.g. neural nets)
  - **Discrete Optimization** (particular subclass of constrained non-convex optimisation where the set $\mathscr{W}$ is discrete, e.g.$w \in \{\pm 1\}$ (**very difficult** to solve, e.g. interpretable ML, verifiability of ML models)

Optimization Terminology

$$\min_{w \in \mathbb{R}^d} L(w)$$

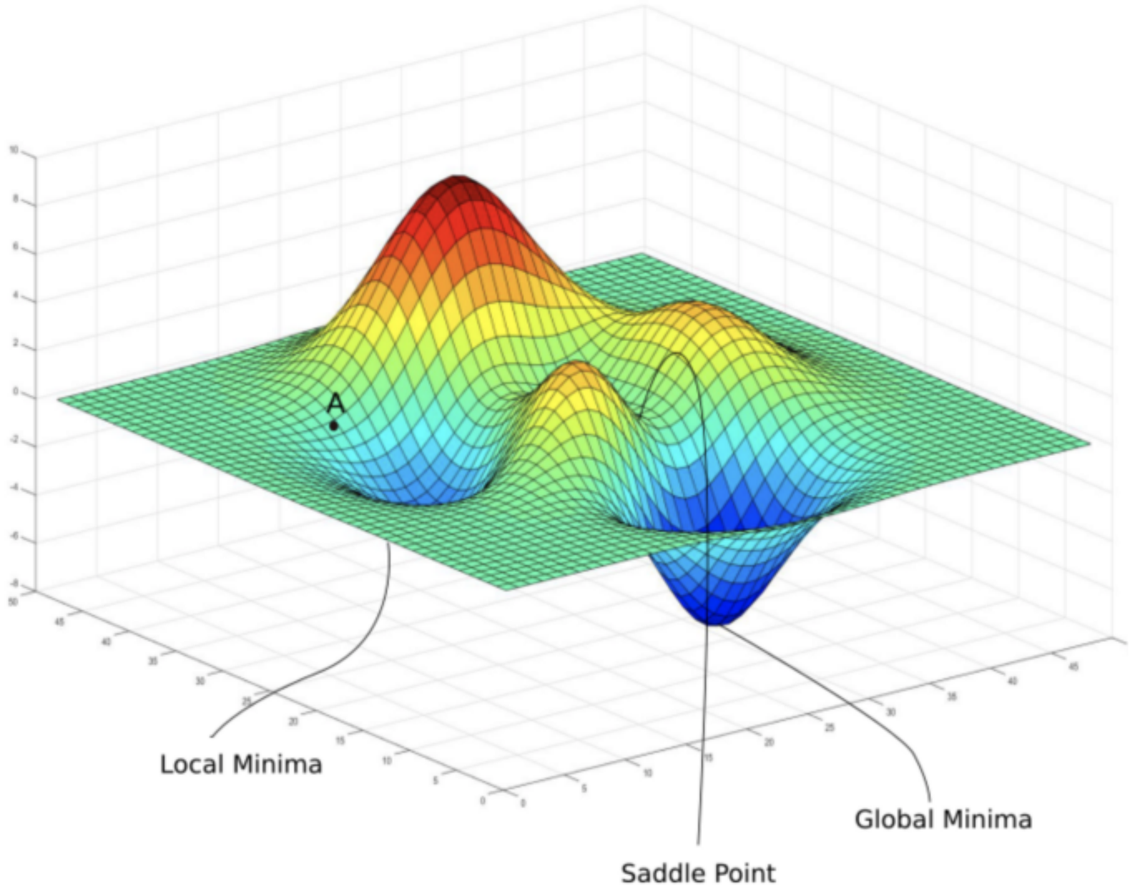
A point $w^\star \in \mathbb{R}^d$

Satisfies first order conditions: $\nabla L(w^\star) = 0$ (critical/stationary points)

If $\nabla^2 L(w^\star) \succ 0$ (local minimum)

If $\nabla^2 L(w^\star) \prec 0$ (local maximum)

otherwise a saddle point



Example Linear Regression in 1d

$$f(x) = a \cdot x \qquad L(a) = \sum_{n=1}^{N} (f(x_n) - y_n)^2$$

$$\frac{\mathrm{d}L}{\mathrm{d}a} = \sum_{n=1}^{N} 2(ax_n - y_n)x_n = \sum_{n=1}^{N} 2ax_n^2 - 2x_n y_n = 0$$

$$2a \sum_n x_n^2 = \sum_n 2x_n y_n$$

$$a = \frac{\sum_n x_n y_n}{\sum_n x_n^2}$$

$$\frac{\mathrm{d}^2 L}{\mathrm{d}a^2} = \sum_{n=1}^{N} 2x_n^2 \geqslant 0$$

Classes of Optimization Solvers

- **Zero Order** Methods (black-box, derivative-free optimization) (require function values only, **very slow** (e.g. Nelder-Mead)
- **First Order** Methods (require gradients, for smooth problems converge linearly, scales to very large problems (e.g. gradient descent)
- **Second Order** Methods (require gradients, Hessians, typically converge to minima, very fast, do not scale well (e.g. Newton method)

Classes of Optimization Solvers

- **Zero Order** Methods (black-box, derivative-free optimization) (require function values only, **very slow** (e.g. Nelder-Mead)
- **First Order** Methods (require gradients, for smooth problems converge linearly, scales to very large problems (e.g. gradient descent)
- **Second Order** Methods (require gradients, Hessians, typically converge to minima, very fast, do not scale well (e.g. Newton method)

Other methods

- **Simplex** method (special algorithm for linear optimization)
- **Interior Point Method** (polynomial time algorithm for linear, quadratic and other convex problems).
- **Projection and proximal** methods (special algorithms for problems with certain constraint sets)
- **Discrete Optimization** Methods (e.g. branch and bound)

Classes of Optimization Solvers

- **Zero Order** Methods (black-box, derivative-free optimization) (require function values only, **very slow** (e.g. Nelder-Mead)
- **First Order** Methods (require gradients, for smooth problems converge linearly, scales to very large problems (e.g. gradient descent)
- **Second Order** Methods (require gradients, Hessians, typically converge to minima, very fast, do not scale well (e.g. Newton method)

Other methods

- **Simplex** method (special algorithm for linear optimization)
- **Interior Point Method** (polynomial time algorithm for linear, quadratic and other convex problems).
- **Projection and proximal** methods (special algorithms for problems with certain constraint sets)
- **Discrete Optimization** Methods (e.g. branch and bound)

In ML we typically use stochastic variants of all the above (more later)

Optimization Methods rely on local approximations

$$w^\star \in \arg\min_{w \in \mathbb{R}^d} L(w)$$

1. Given an initial point w

2. Select d (**direction**) to improve solution e.g.

$$L(w+d) < L(w) \text{ or } \|w^\star - (w+d)\| < \|w^\star - w\|$$

3. Update solution (**learning**) $w \leftarrow w + \tau\, d$

Positive scalar (step-size, learning rate)

Optimization Methods rely on local approximations

$$w^\star \in \arg\min_{w \in \mathbb{R}^d} L(w)$$

1. Given an initial point w

2. Select d (**direction**) to improve solution e.g.

$$L(w+d) < L(w) \text{ or } \|w^\star - (w+d)\| < \|w^\star - w\|$$

3. Update solution (**learning**) $w \leftarrow w + \tau d$

Positive scalar (step-size, learning rate)

Repeat the process until happy with the solution, **but how is d chosen?**

Local Approximations with Gradient Descent

How do we ensure that $L(w_{k+1}) < L(w_k)$?

GD makes a local quadratic approximation:

$$\hat{L}(w) \triangleq L(w_k) + \langle \nabla L(w_k), w - w_k \rangle + \frac{1}{2\tau}\|w - w_k\|_2^2$$

Approx. only uses first order information

$$w_{k+1} = \arg\min \hat{L}(w_k)$$

since $\nabla \hat{L}(w_{k+1}) = 0$ if $w_{k+1} = w_k - \tau \nabla L(w_k)$

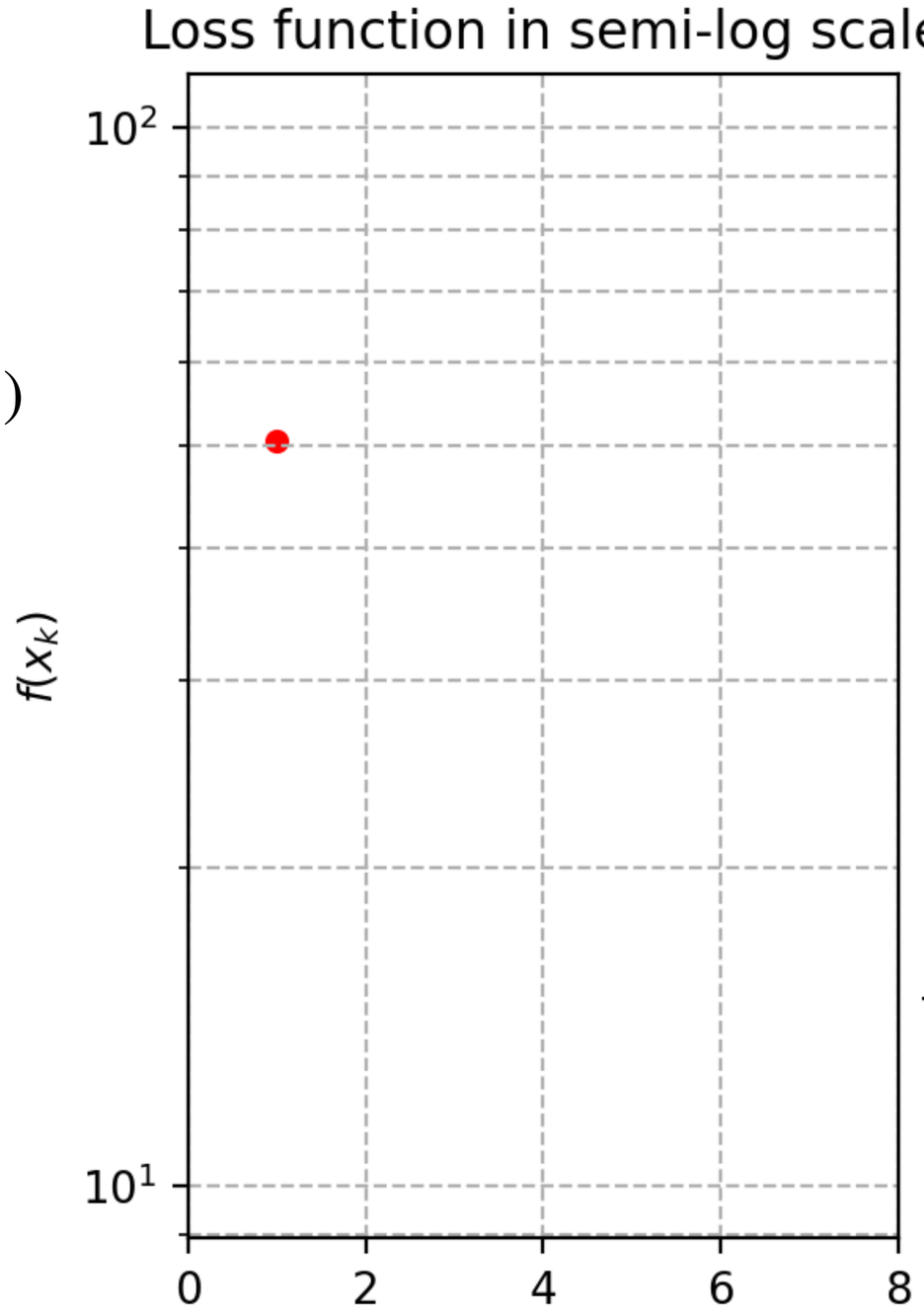
Can be shown that for τ small enough GD will converge to a stationary point

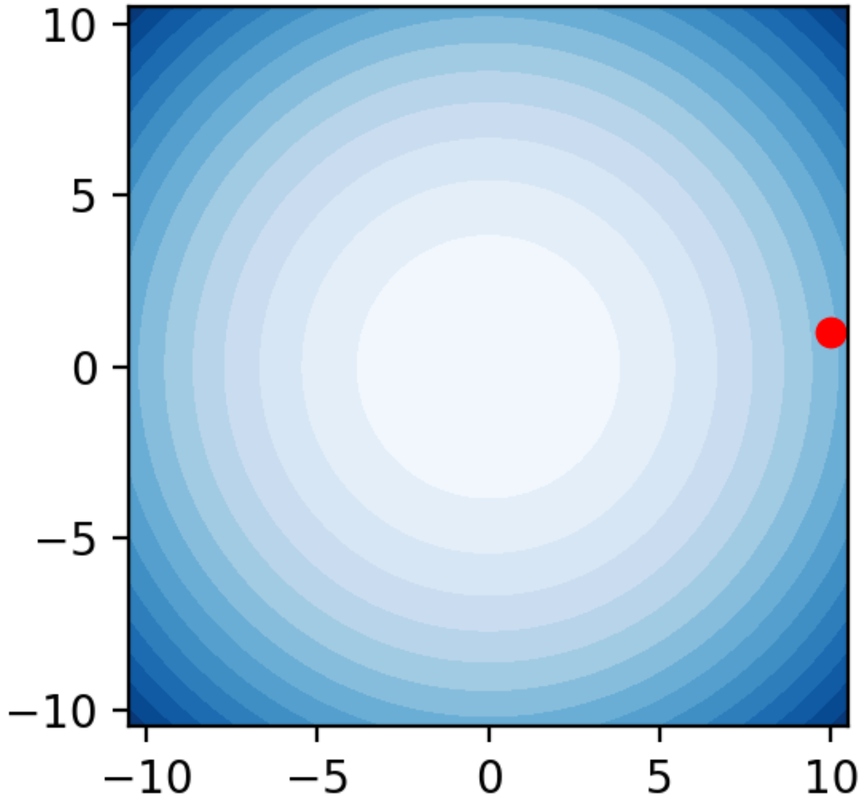
When GD works well

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 1$, initial point $(10,1)$

G.D. with $\tau = 0.5$



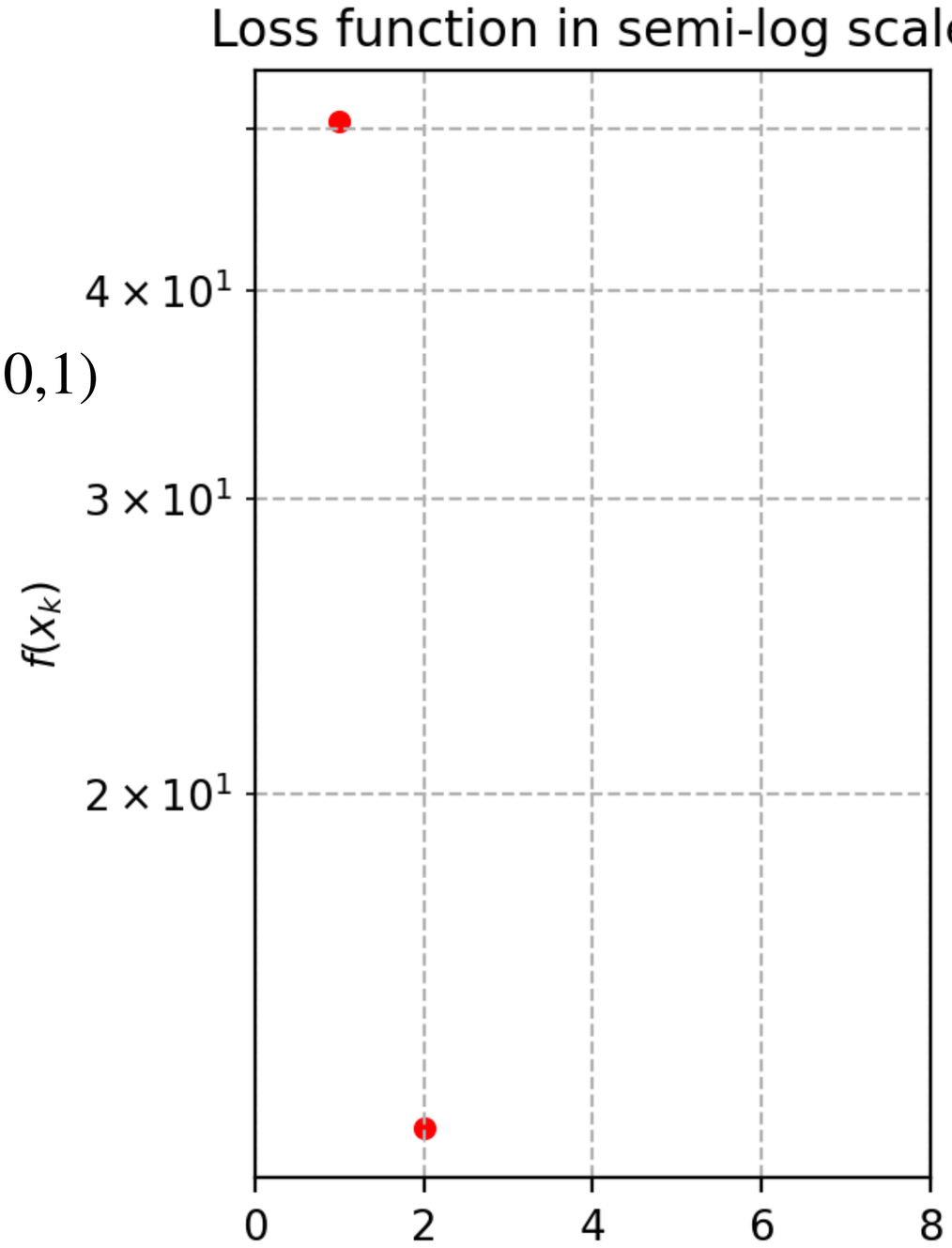


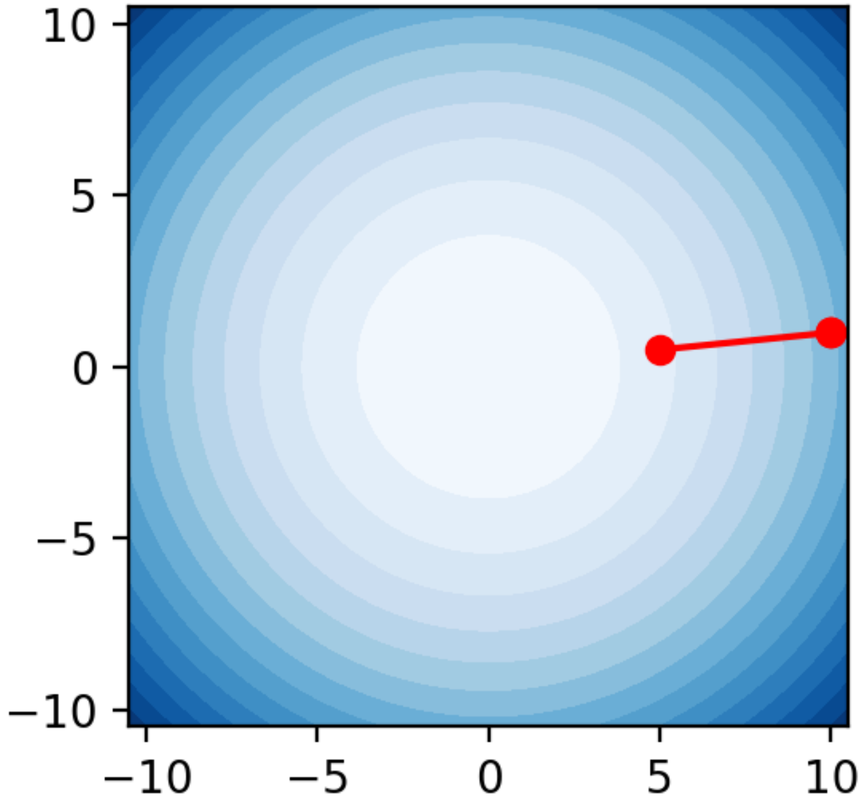
When GD works well

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 1$, initial point $(10,1)$

G.D. with $\tau = 0.5$



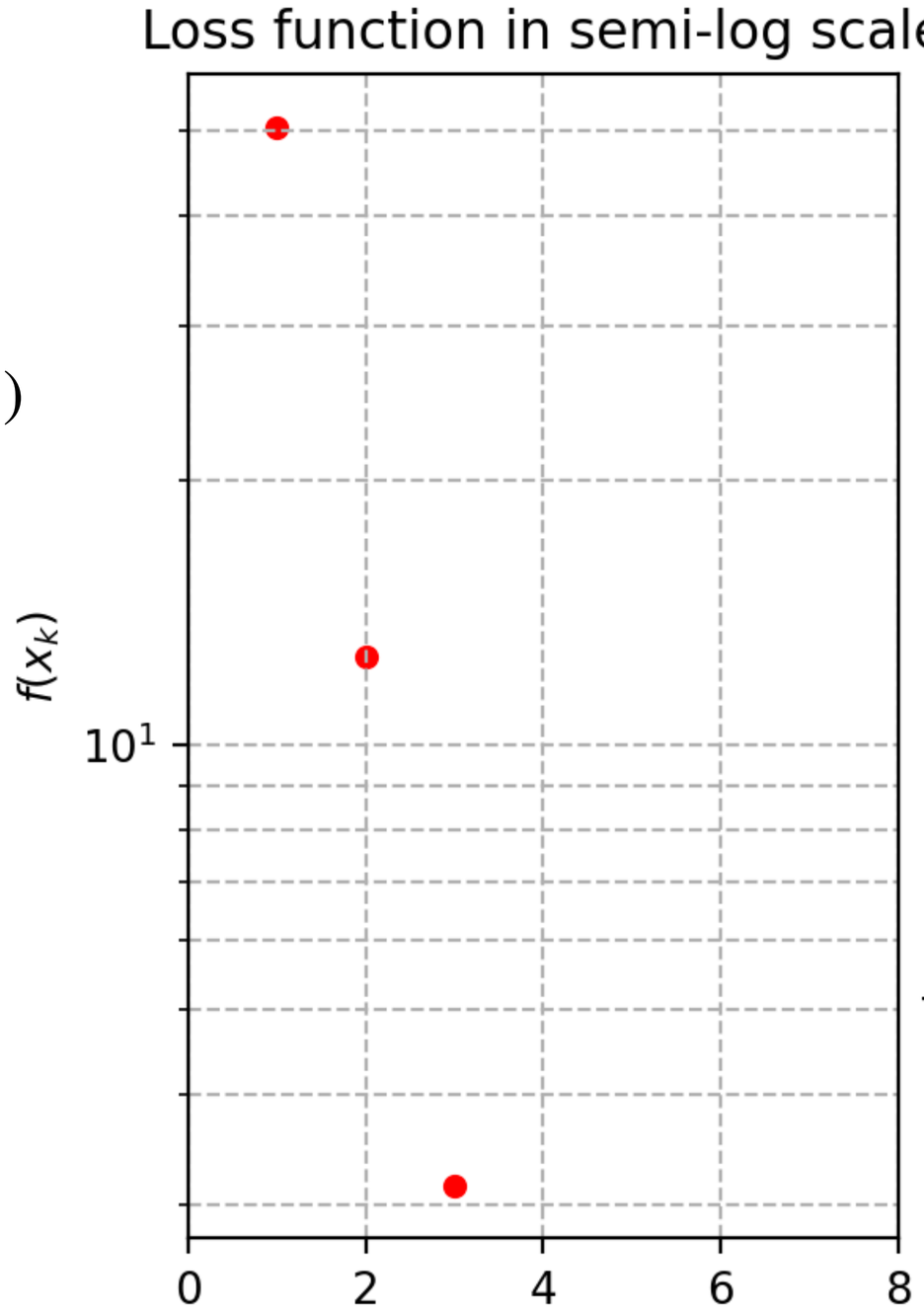


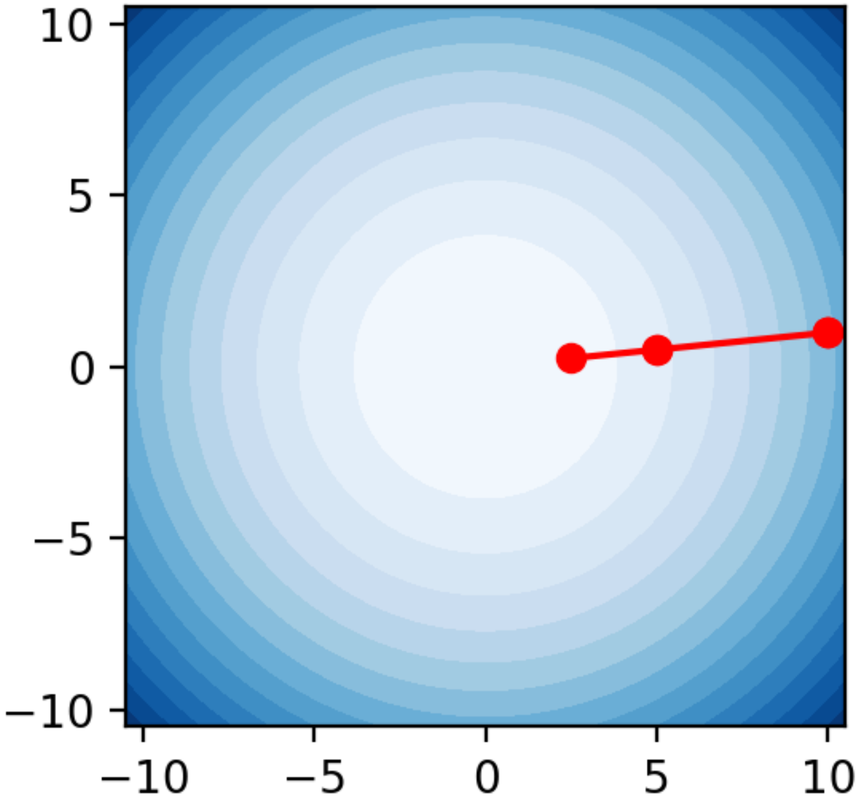
When GD works well

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 1$, initial point $(10,1)$

G.D. with $\tau = 0.5$



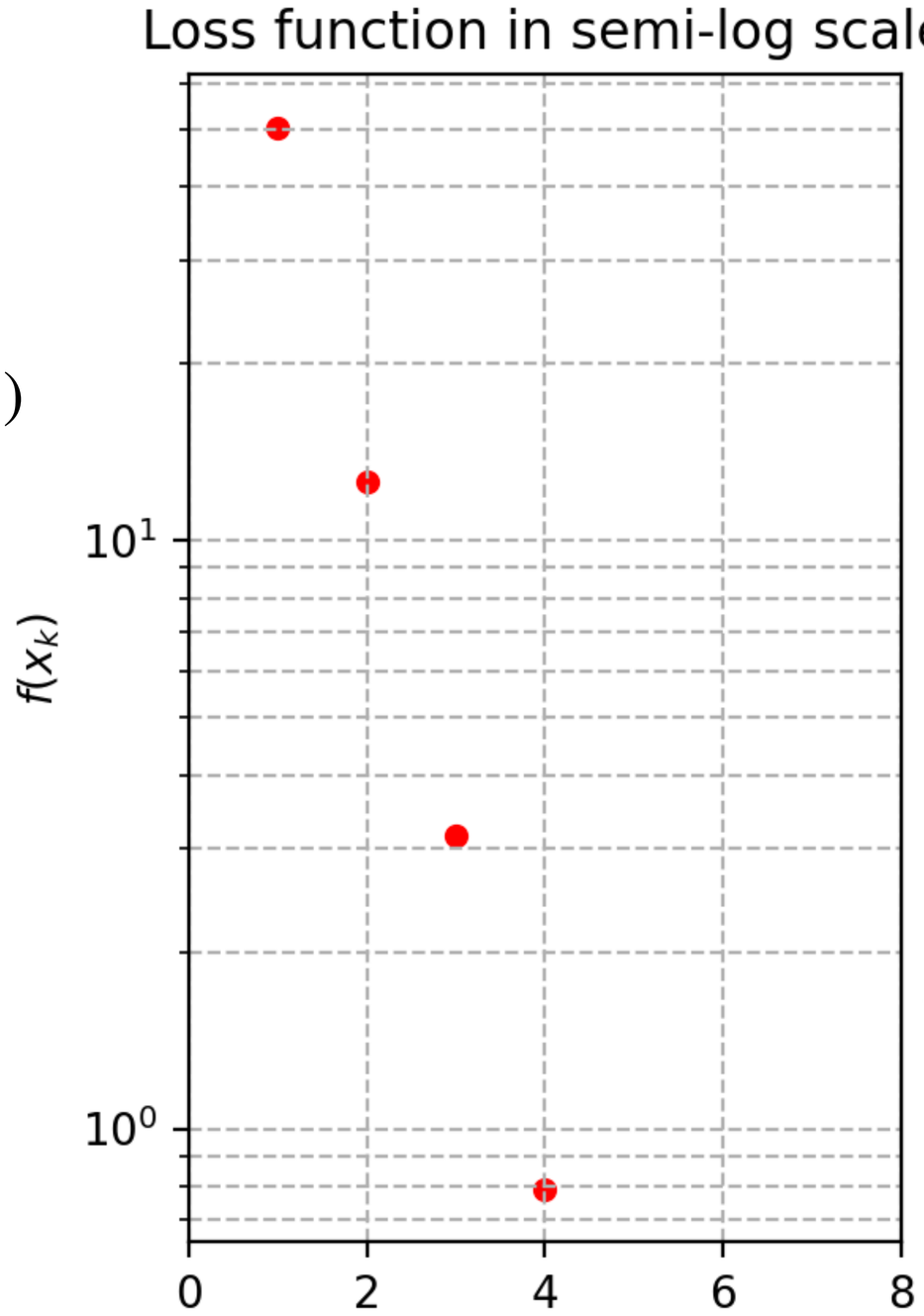


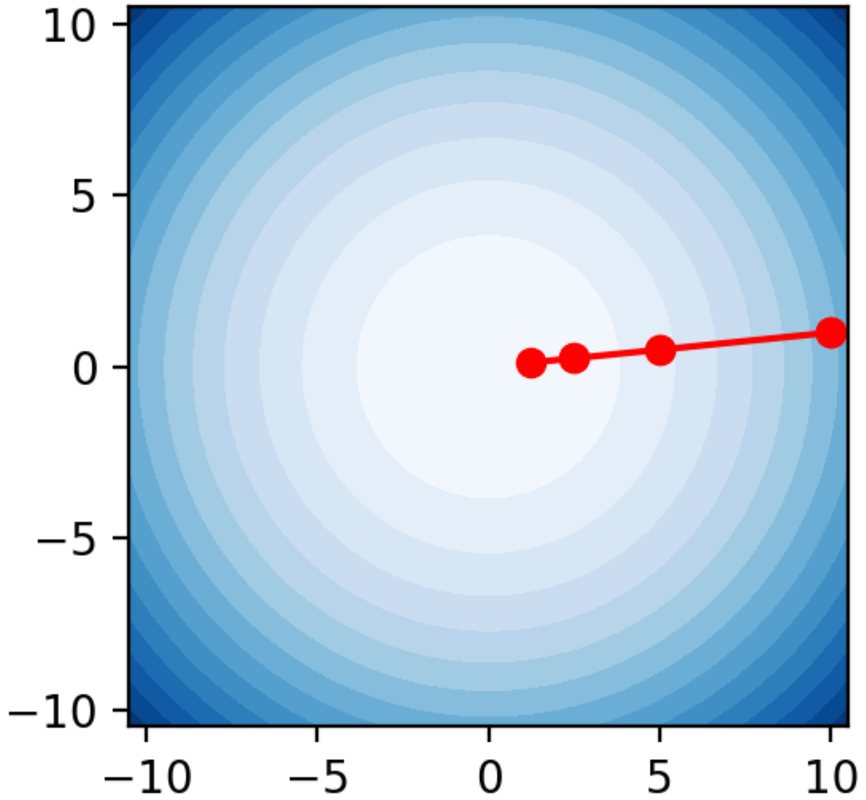
When GD works well

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 1$, initial point $(10,1)$

G.D. with $\tau = 0.5$



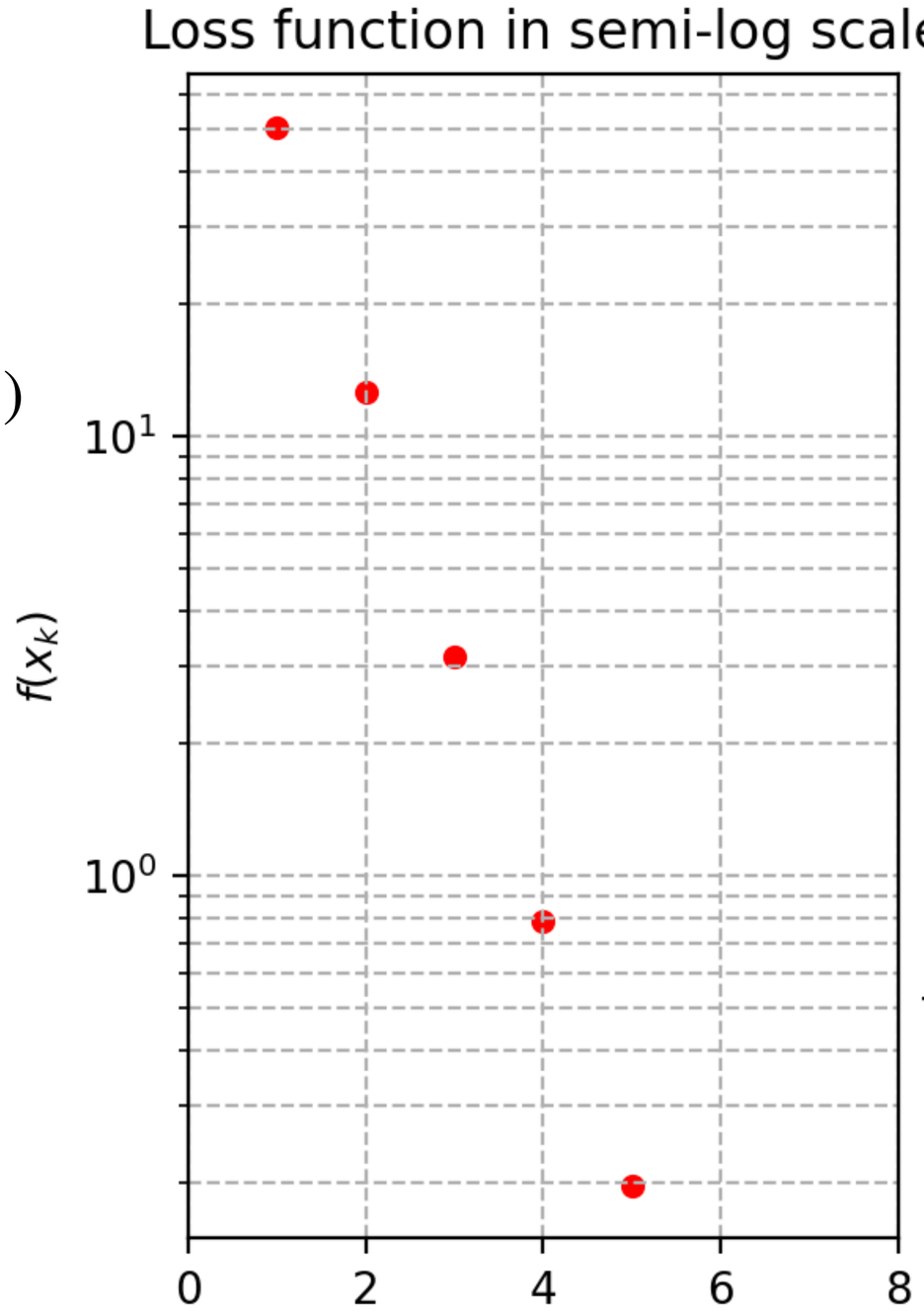


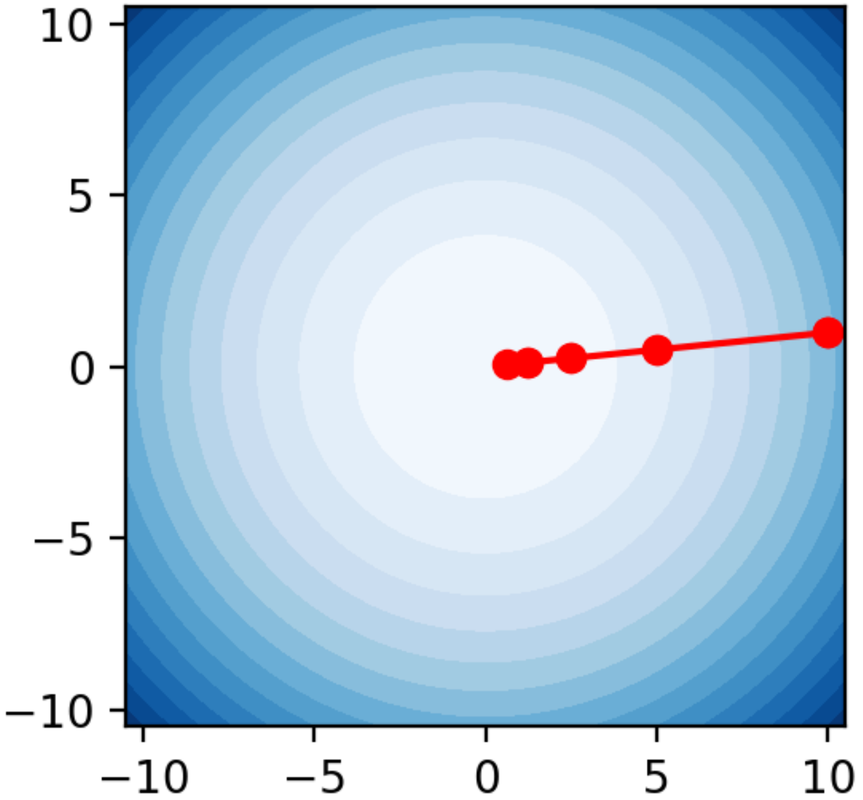
When GD works well

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 1$, initial point $(10,1)$

G.D. with $\tau = 0.5$



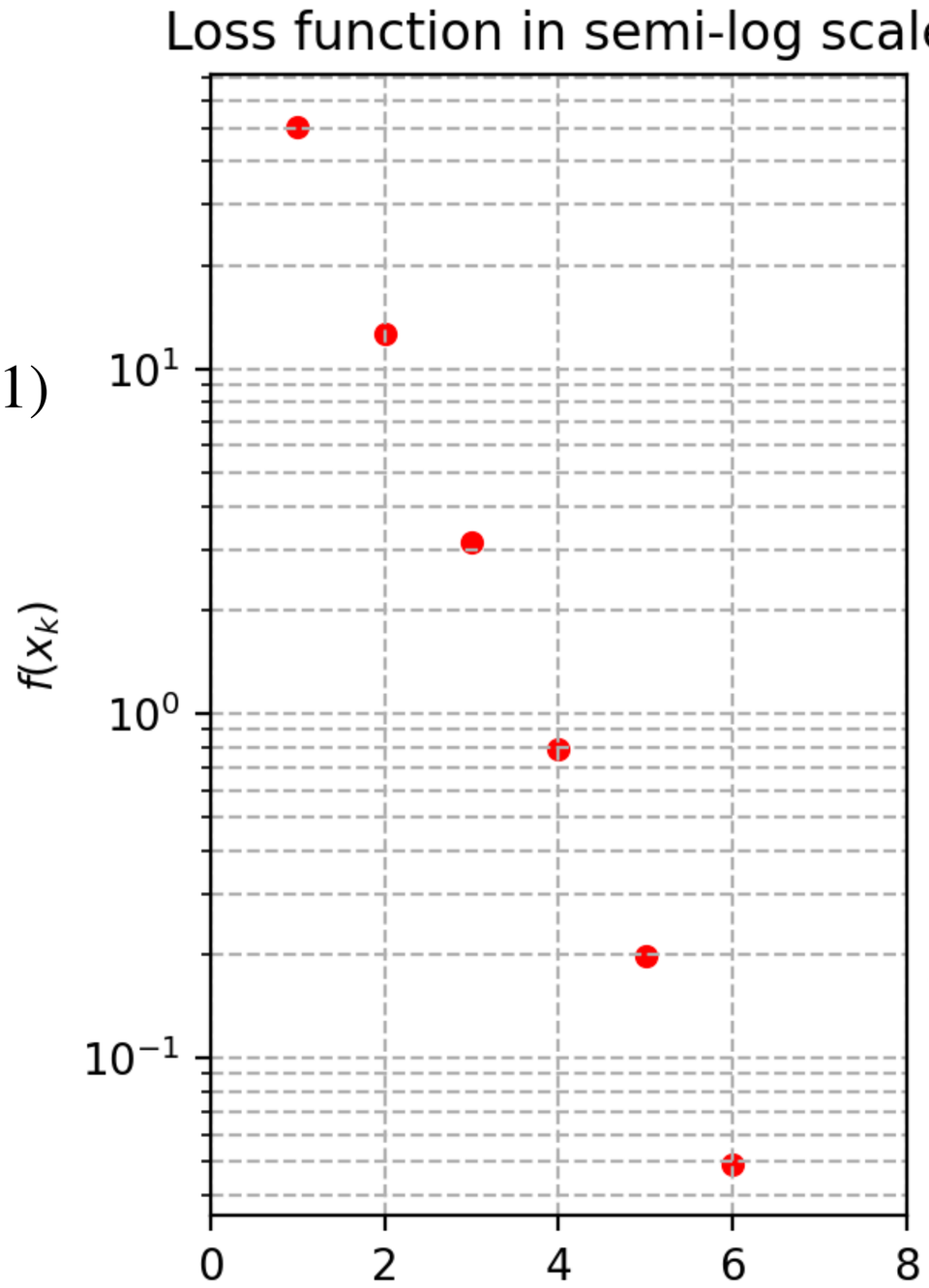


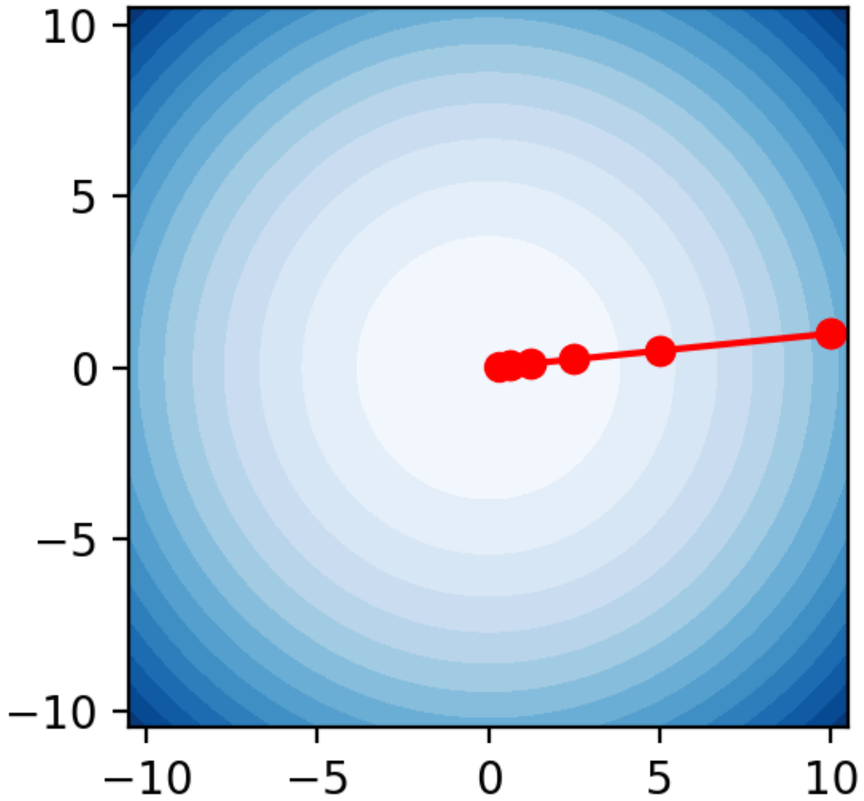
When GD works well

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 1$, initial point $(10,1)$

G.D. with $\tau = 0.5$



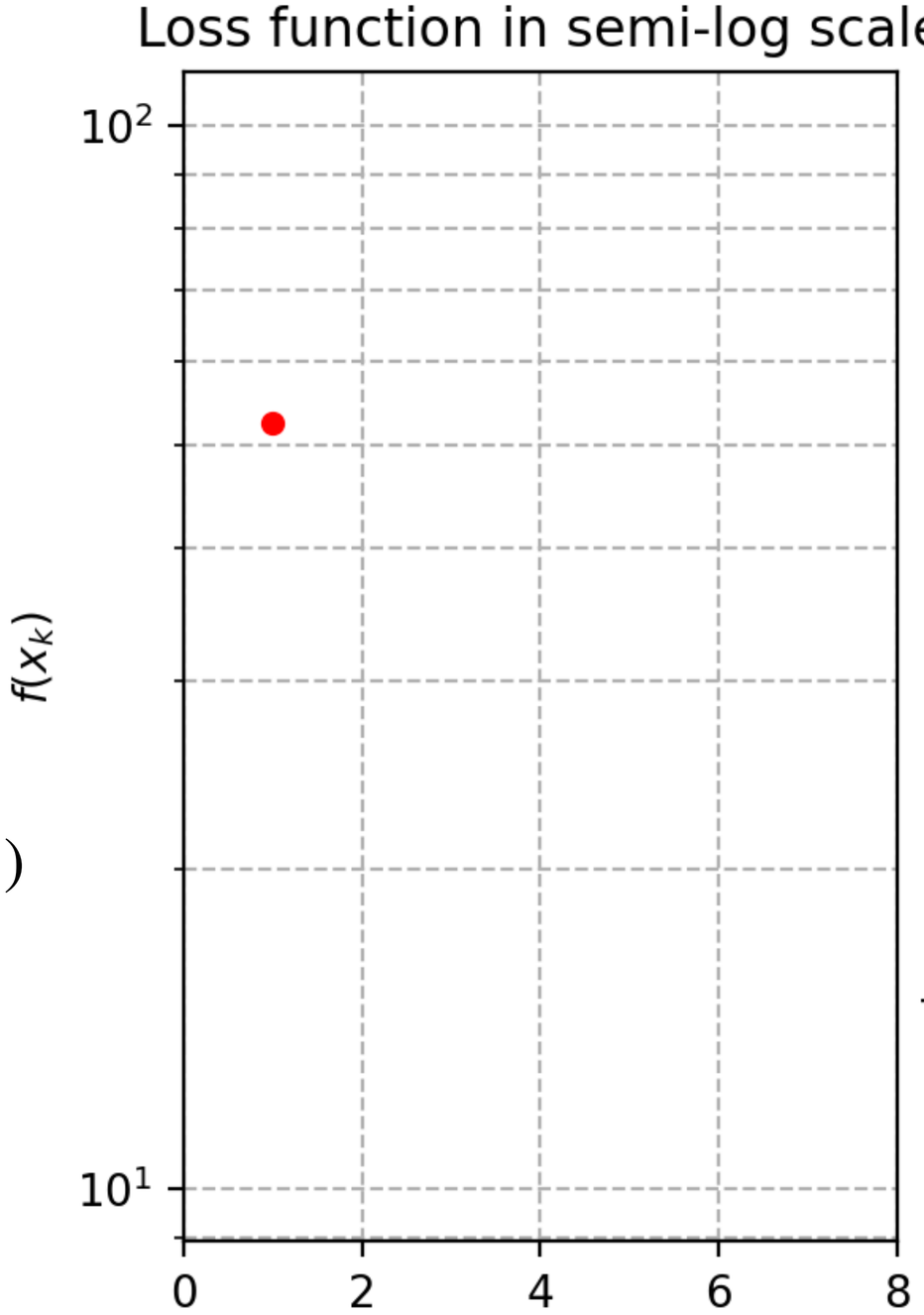


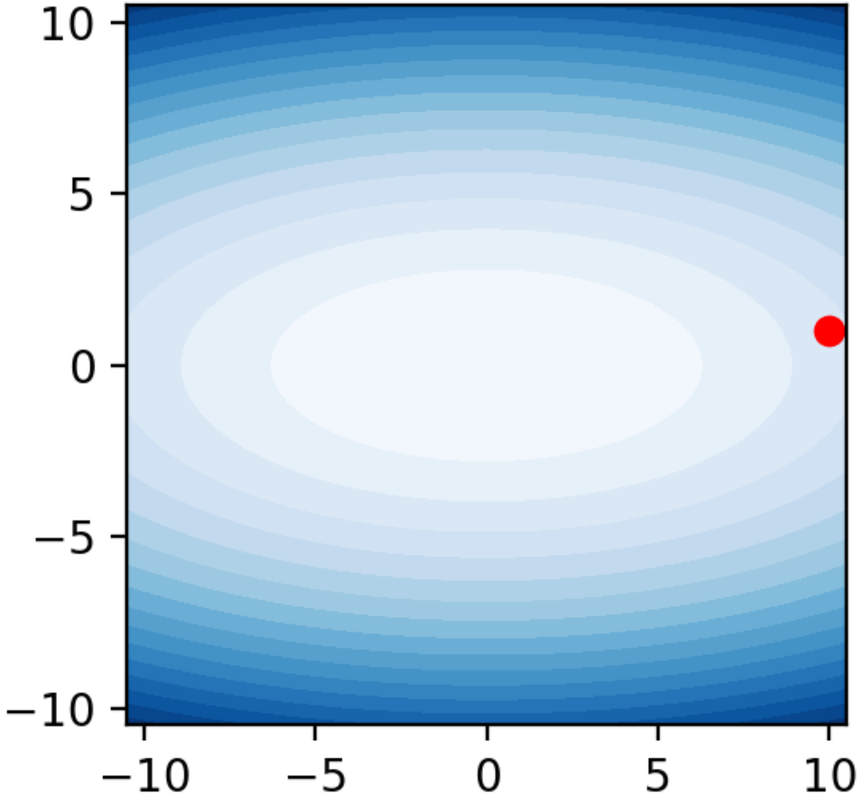
GD can fail for ill-conditioned problems

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 5$, initial point $(10,1)$

G.D. with $\tau = 0.4$



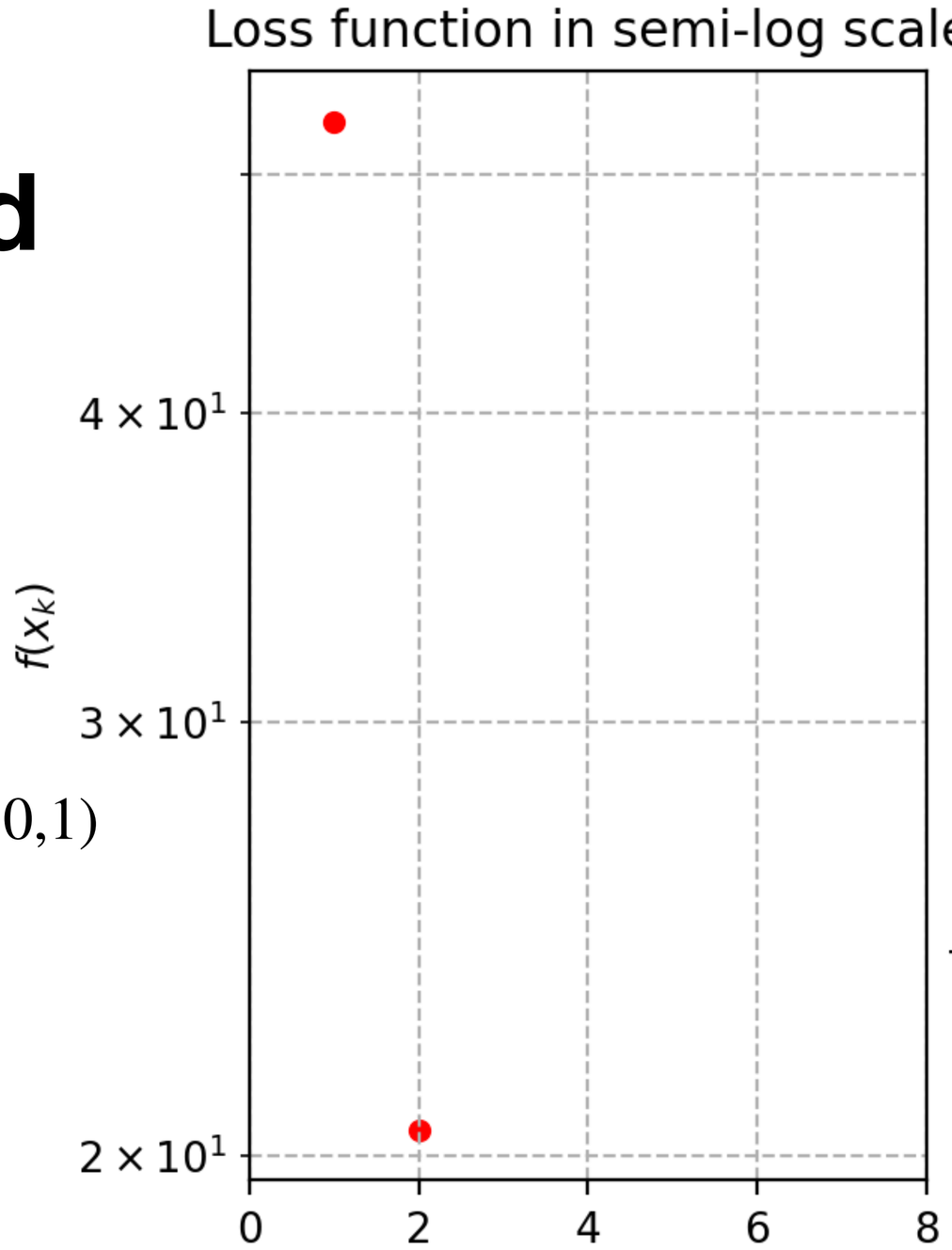


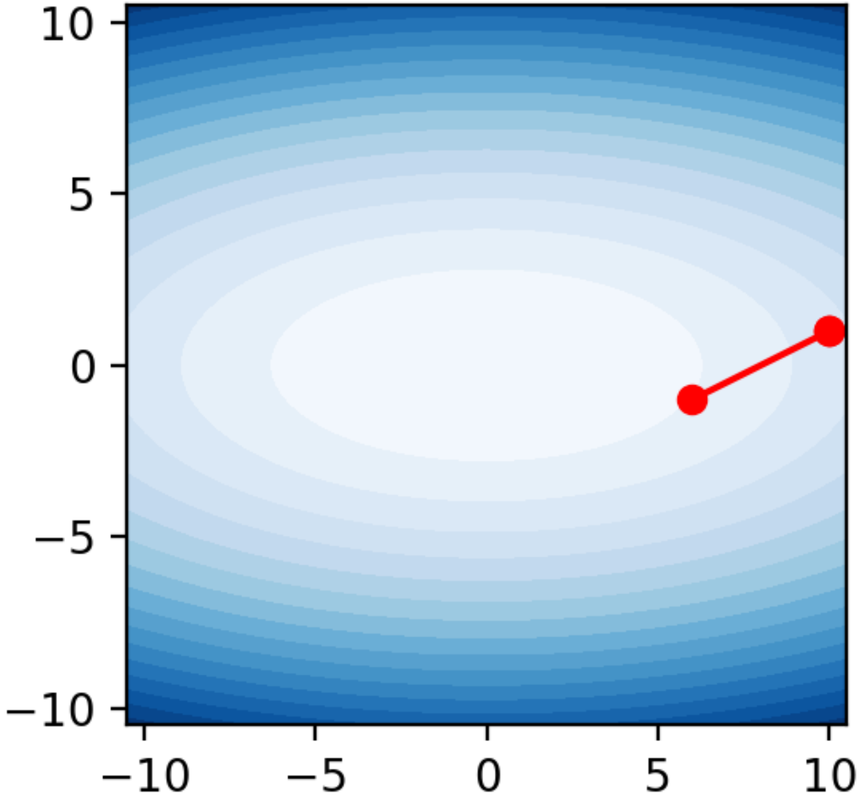
GD can fail for ill-conditioned problems

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 5$, initial point $(10,1)$

G.D. with $\tau = 0.4$



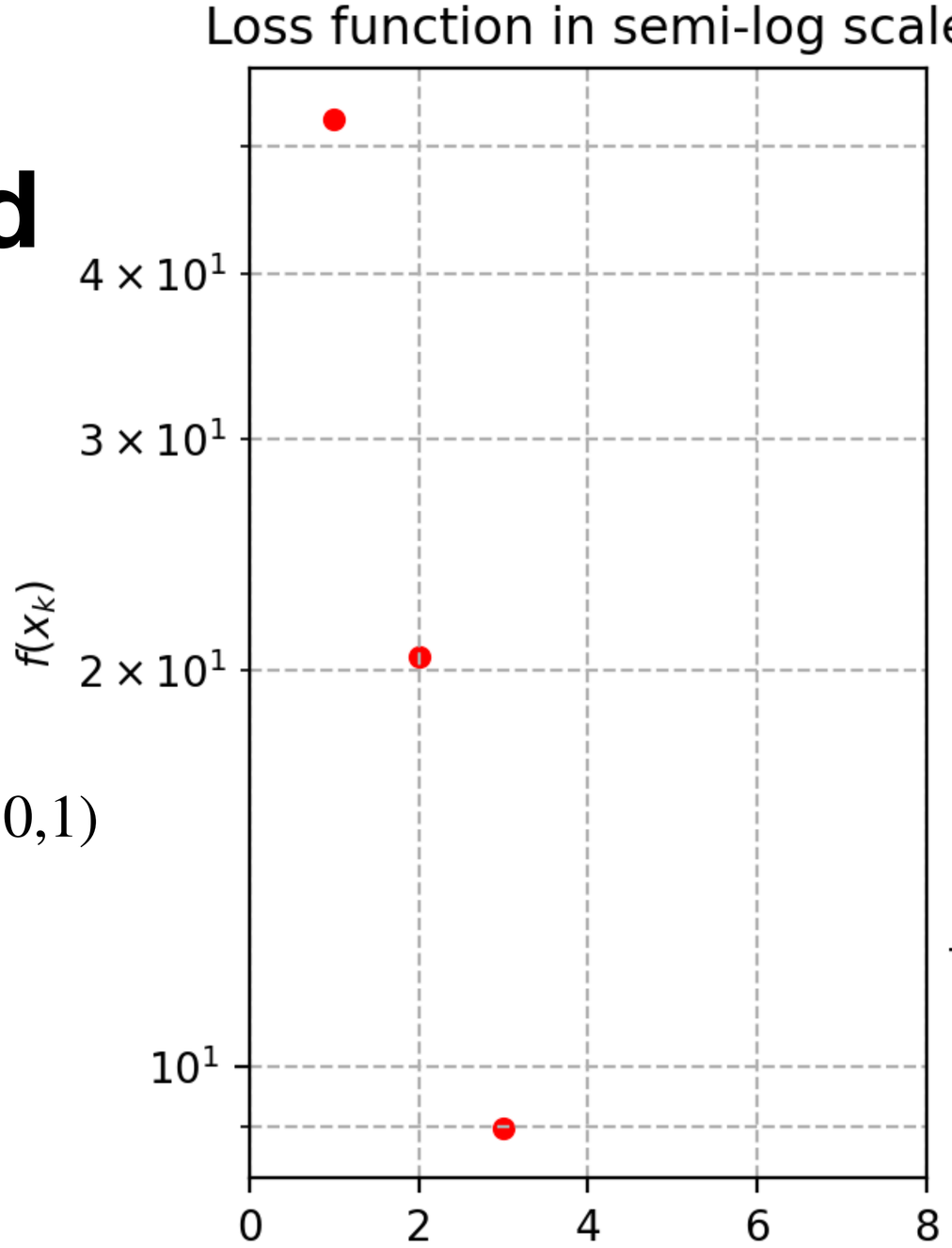


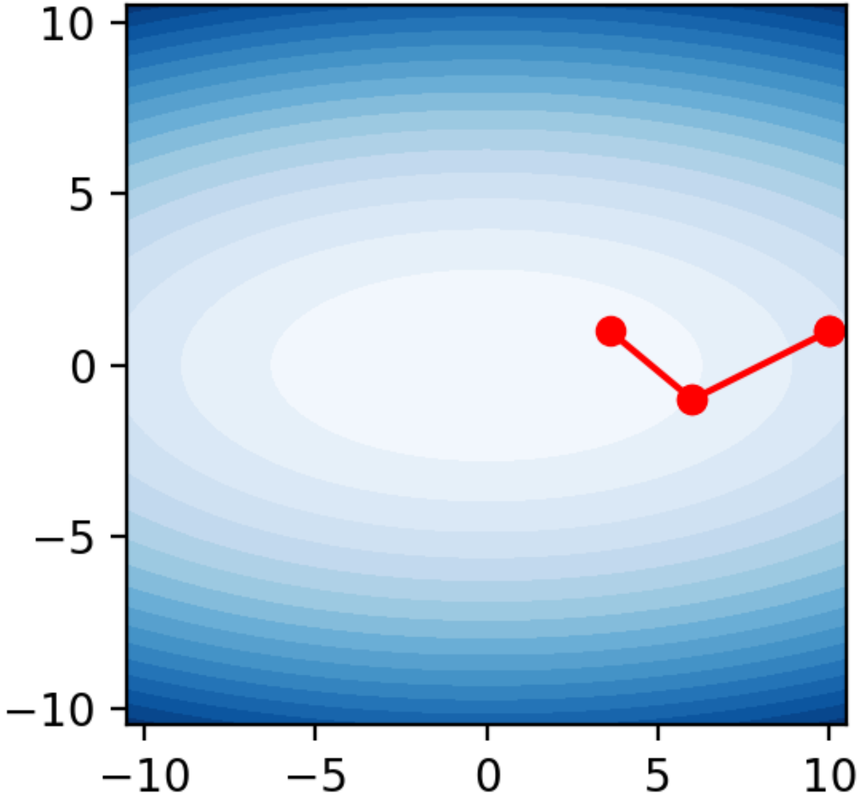
GD can fail for ill-conditioned problems

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 5$, initial point $(10,1)$

G.D. with $\tau = 0.4$



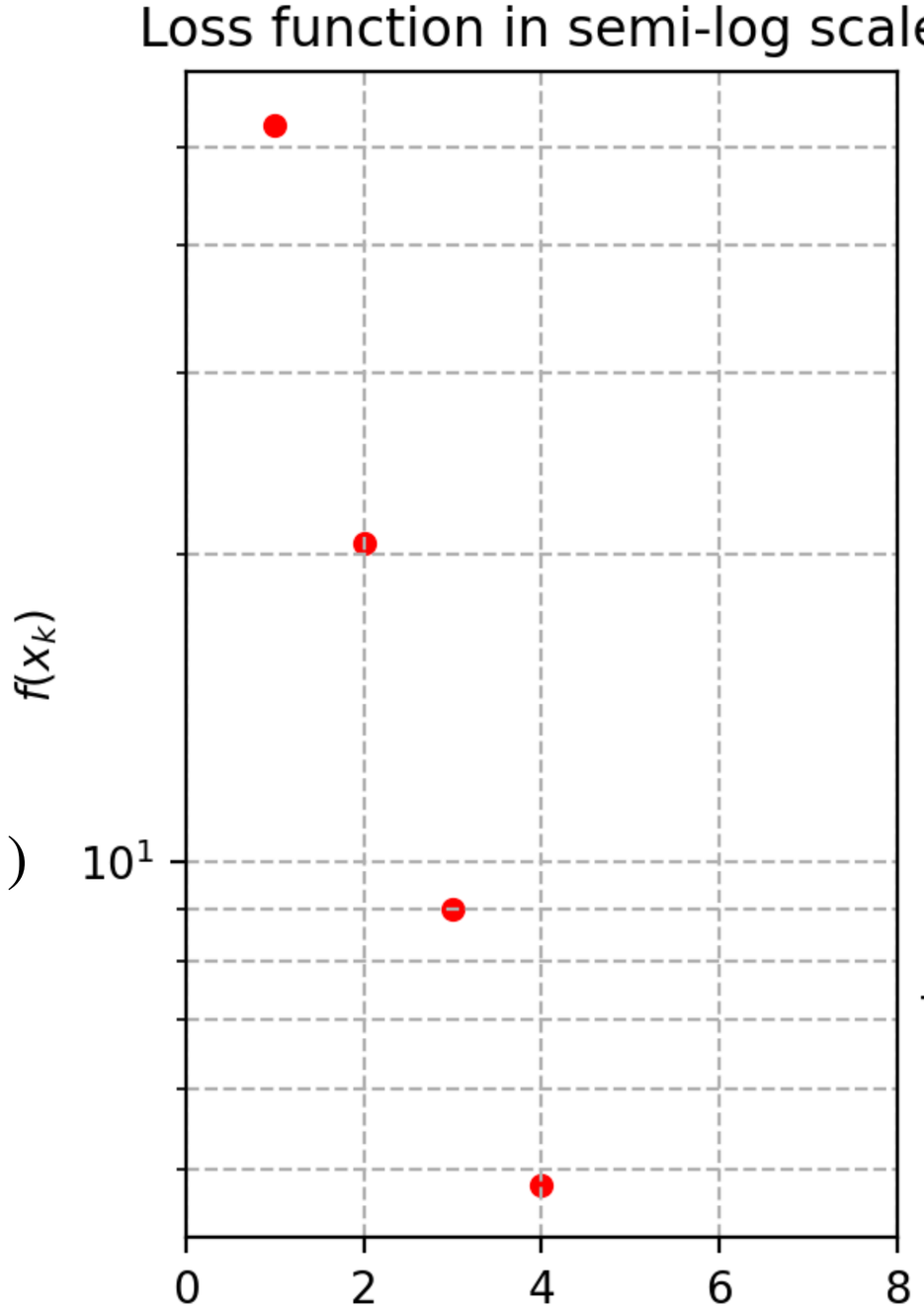


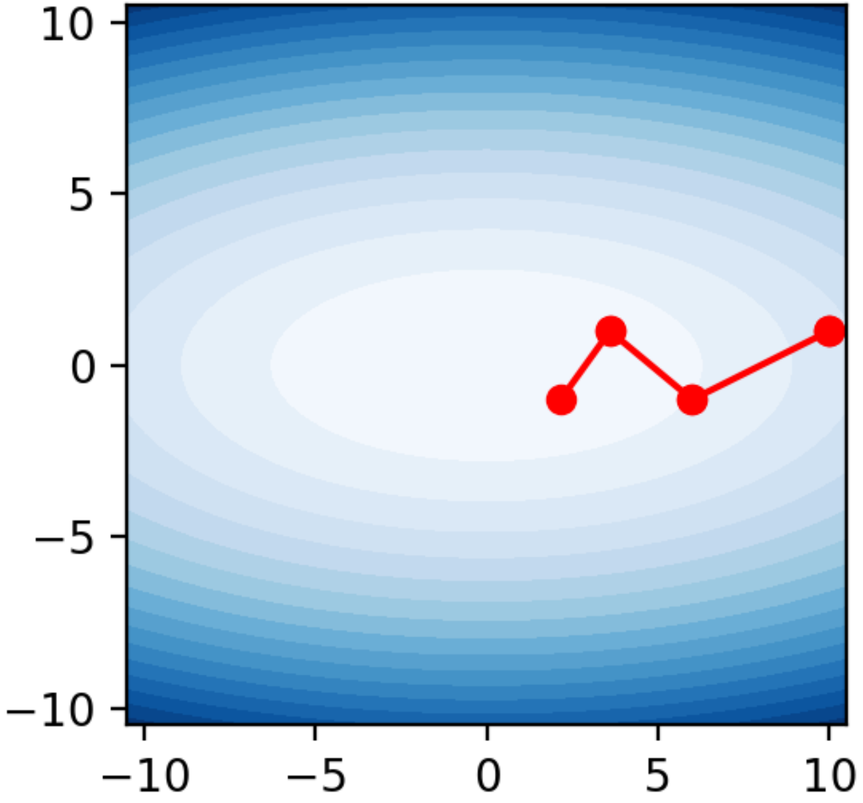
GD can fail for ill-conditioned problems

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 5$, initial point $(10,1)$

G.D. with $\tau = 0.4$



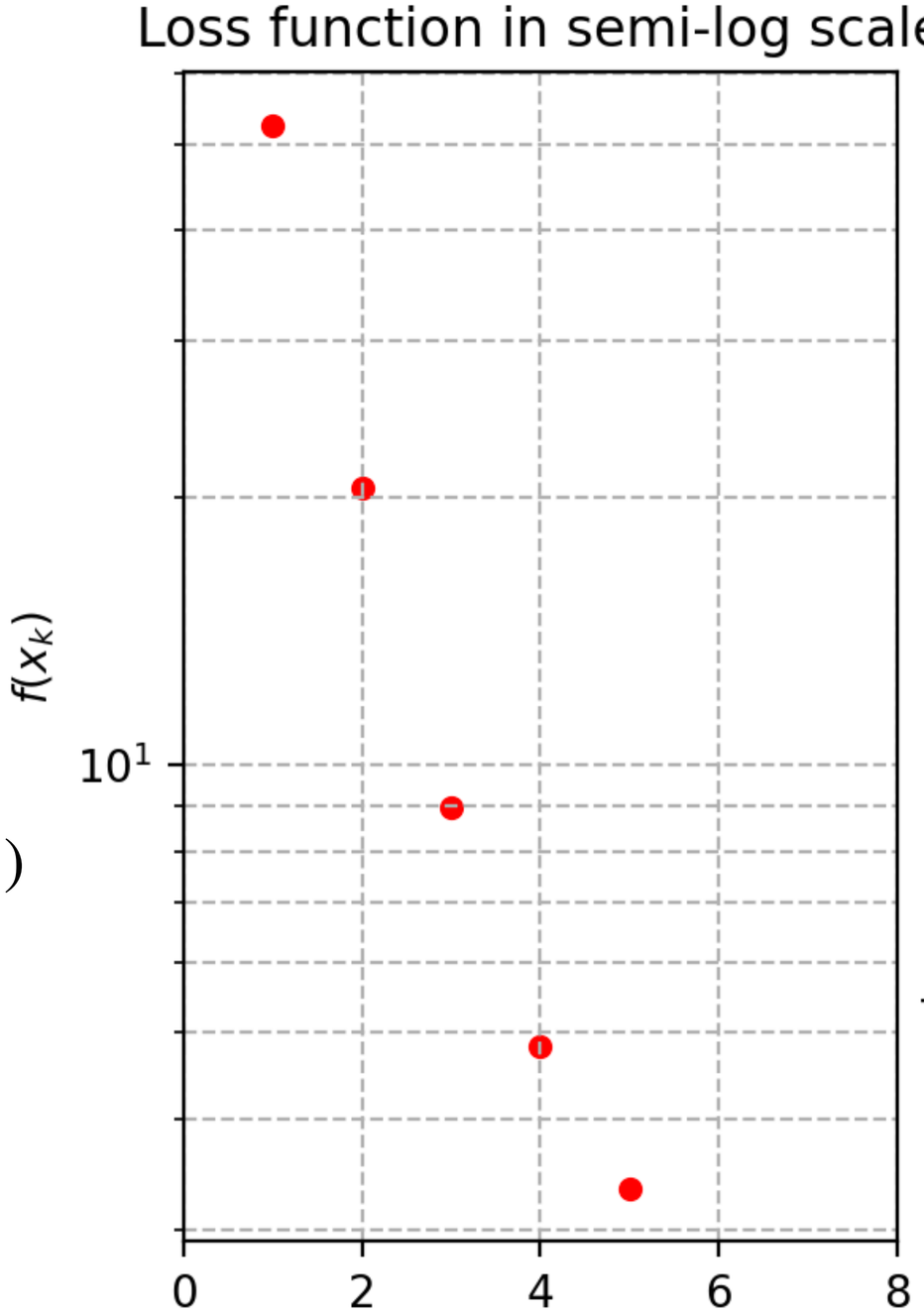


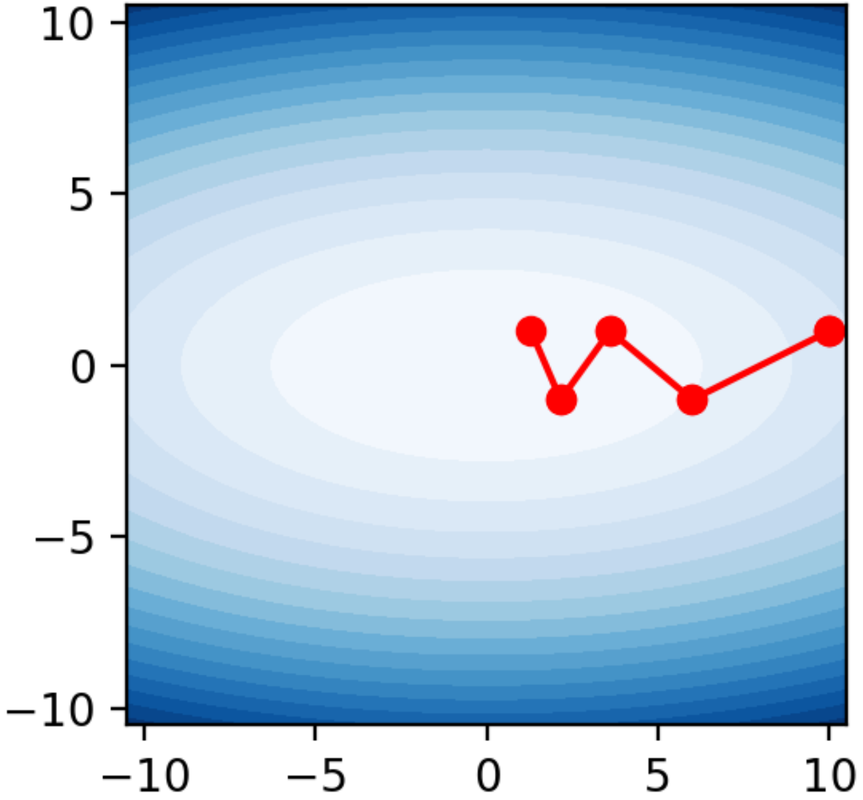
GD can fail for ill-conditioned problems

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 5$, initial point $(10,1)$

G.D. with $\tau = 0.4$



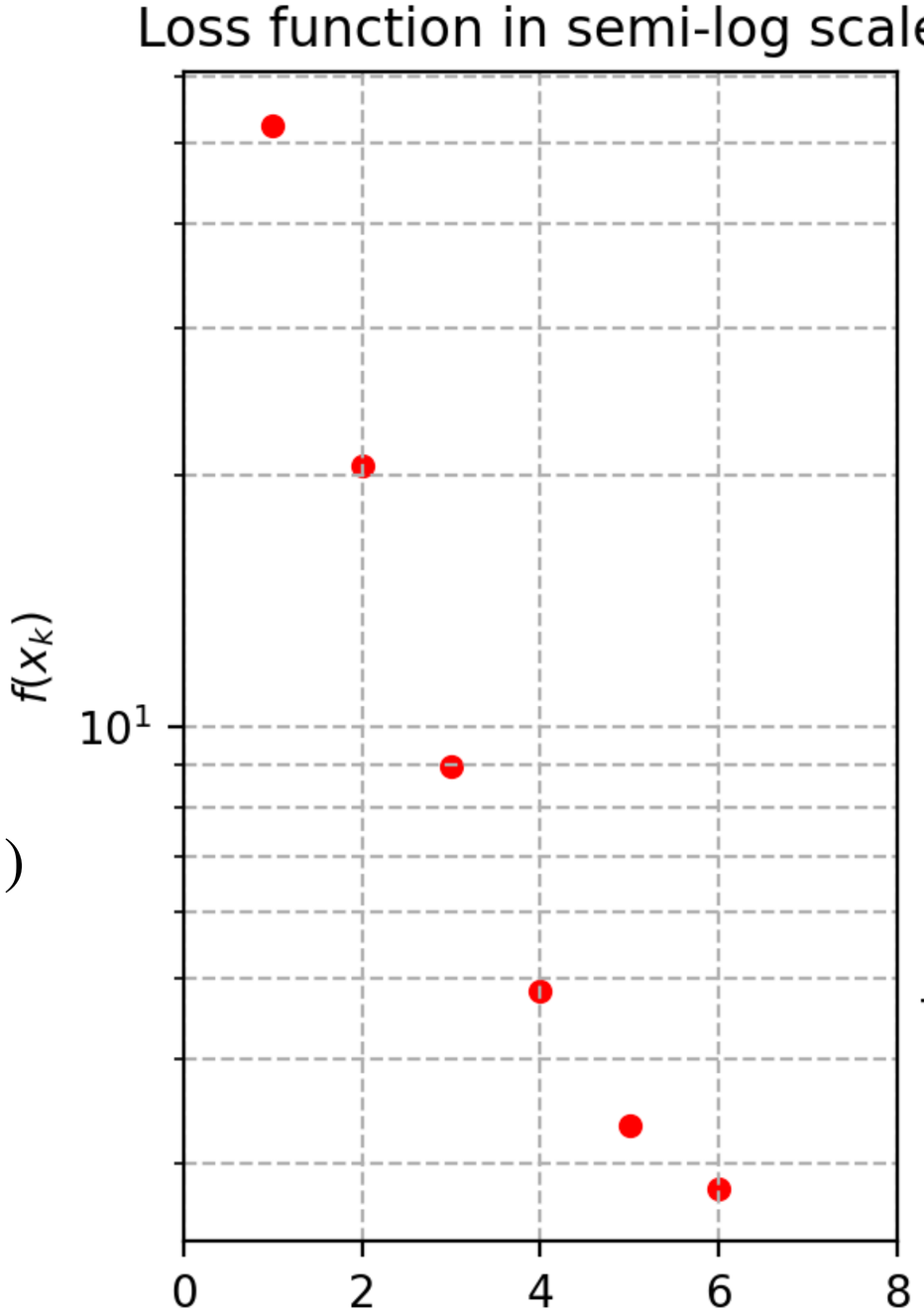


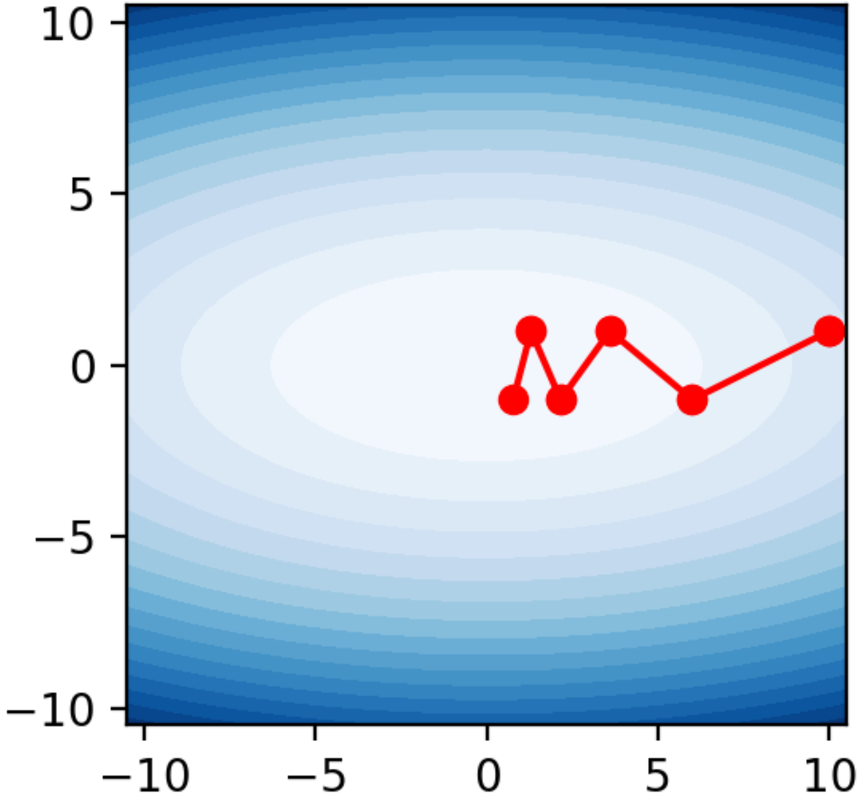
GD can fail for ill-conditioned problems

$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 5$, initial point $(10,1)$

G.D. with $\tau = 0.4$





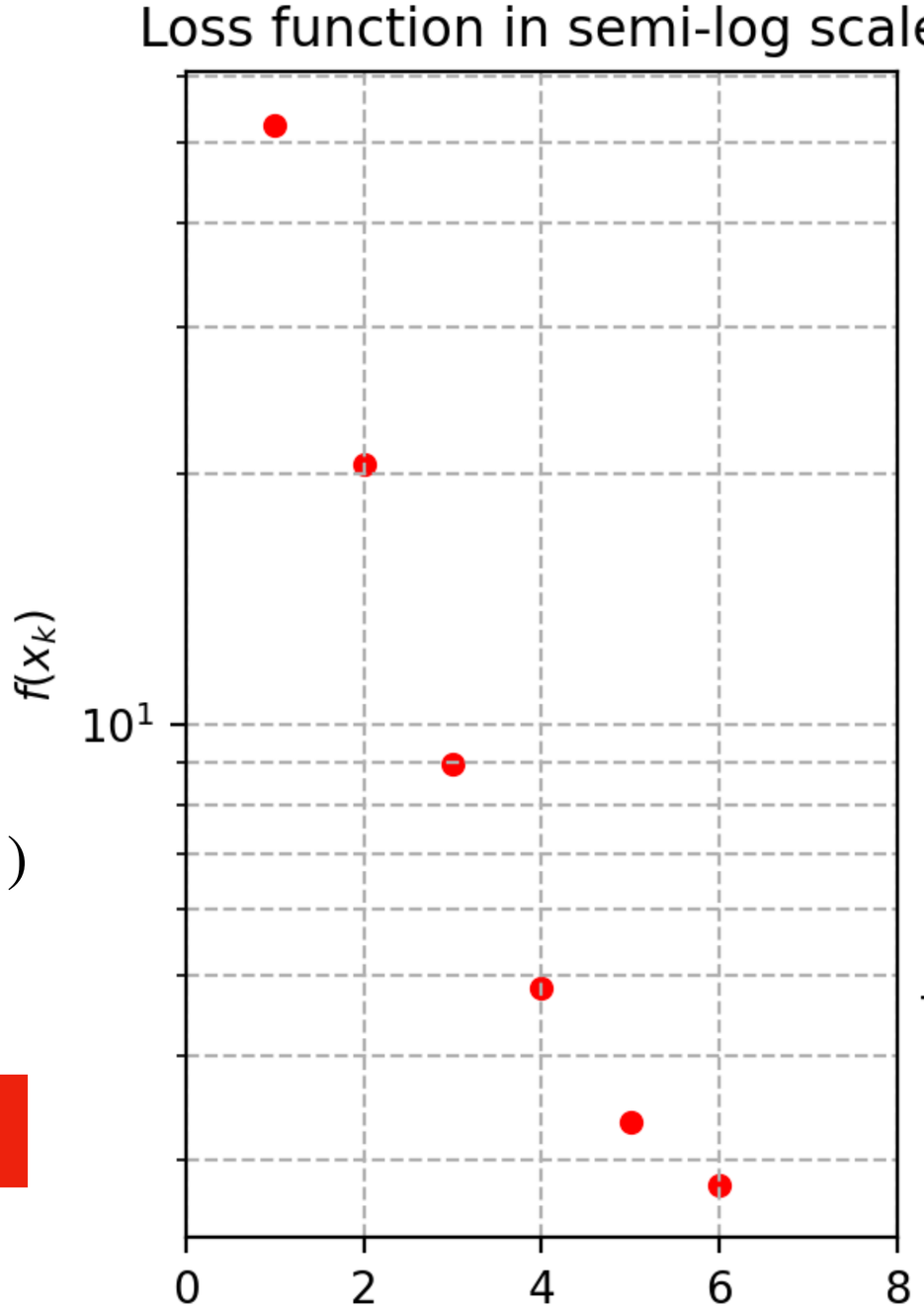
GD can fail for ill-conditioned problems

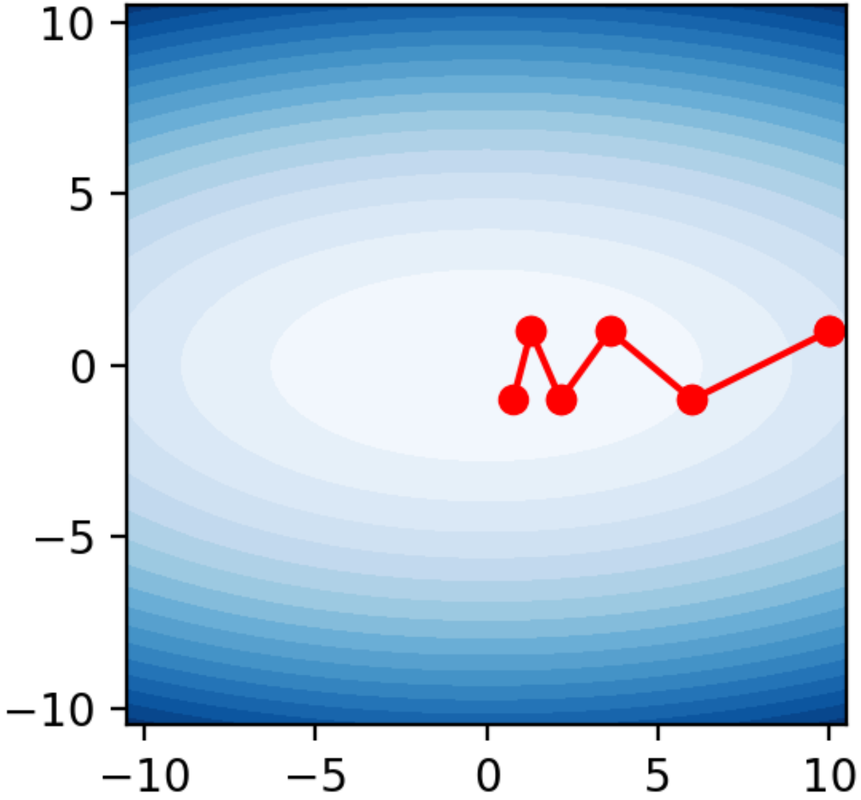
$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 5$, initial point $(10,1)$

G.D. with $\tau = 0.4$

Key insight for ML scale your data!

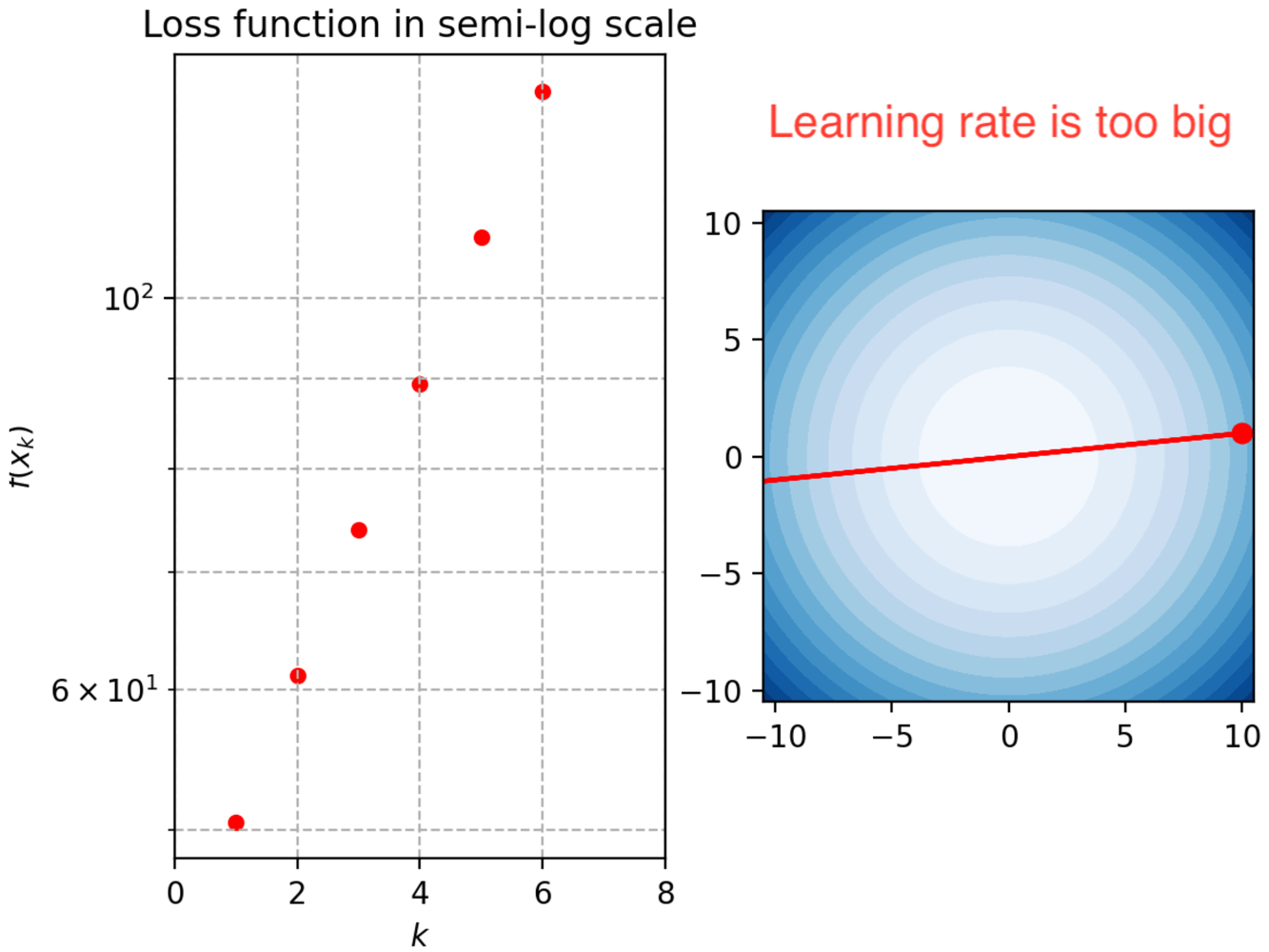




Selecting the learning rate

For deterministic problems the learning rate is chosen by **line-search**

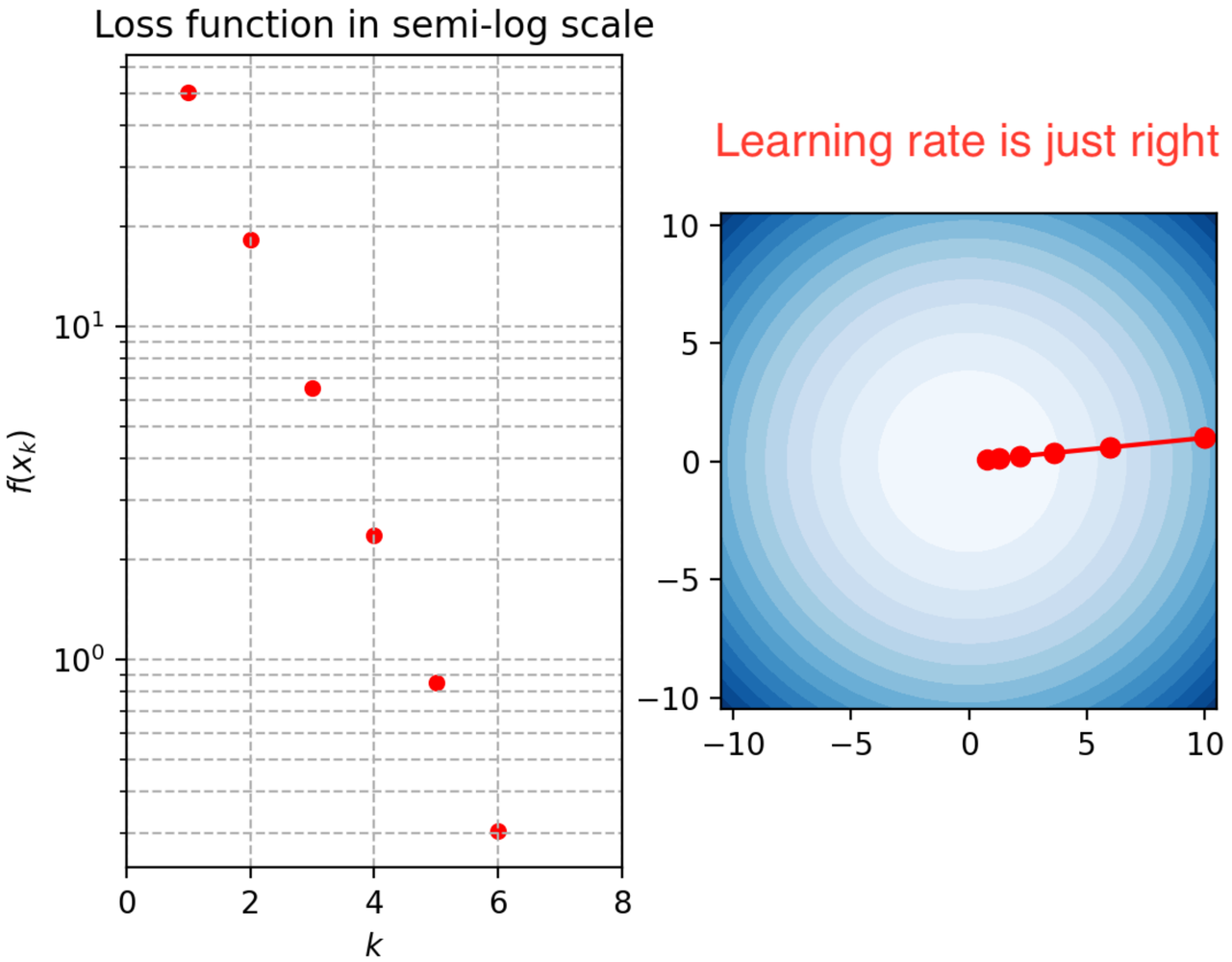
In ML gradients are stochastic and choosing the learning rate is set by **trial and error**!



Selecting the learning rate

For deterministic problems the learning rate is chosen by **line-search**

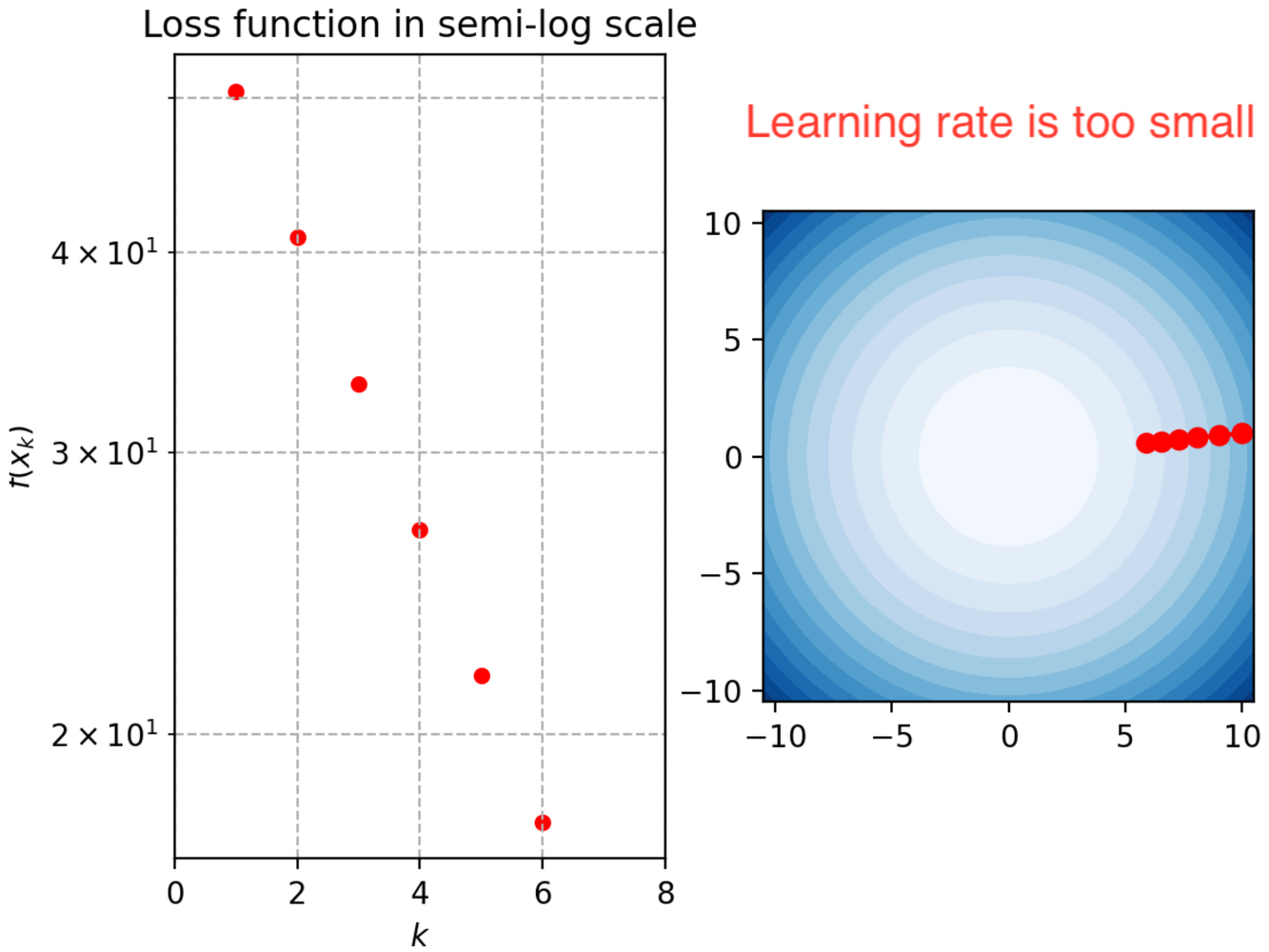
In ML gradients are stochastic and choosing the learning rate is set by **trial and error**!



Selecting the learning rate

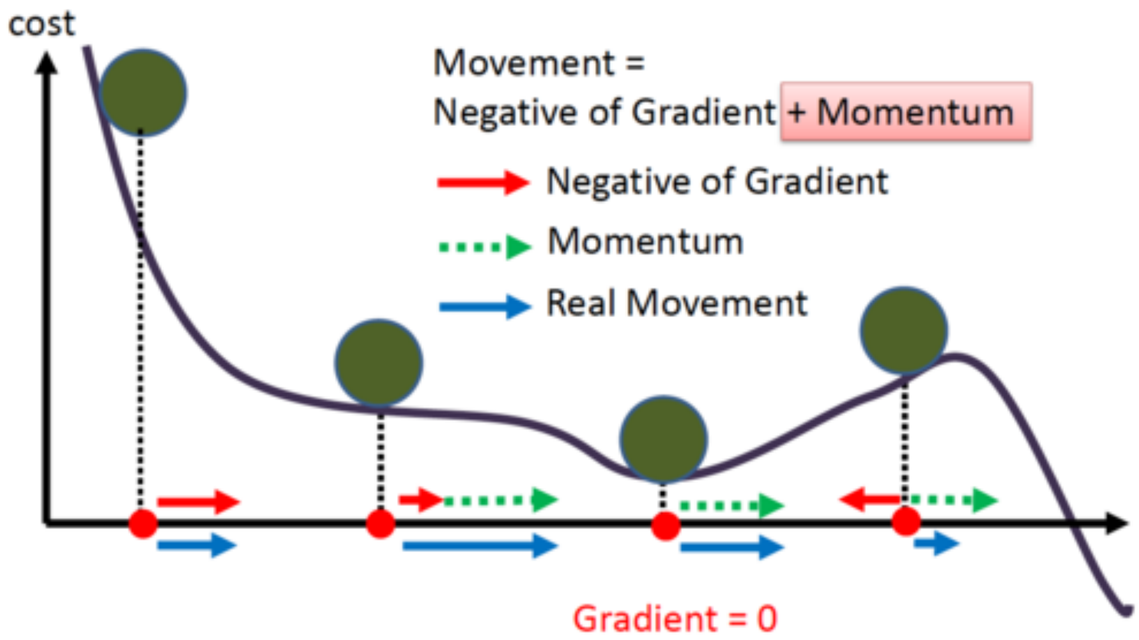
For deterministic problems the learning rate is chosen by **line-search**

In ML gradients are stochastic and choosing the learning rate is set by **trial and error**!



Gradient Descent with Momentum (Acceleration)

GD works by following the negative gradient (direction of steepest descent)



Possible issues:

- Ill conditioning (loss function changes more in some dimensions than others)
- Flat regions (gradient is small, but not zero)
- Shallow minima

Adding a **momentum term** to GD (may) help with these problems

Gradient Descent with Momentum Motivation

$$\boxed{\min_{x \in \mathbb{R}^d} f(x)} \tag{1}$$

The Gradient Descent (GD) algorithm:

$$\boxed{\dot{x} = -\nabla f(x),}$$

The fixed points of GD coincide with the stationary points of f (since $\nabla f(x^\star) = 0$

GD not the only dynamical system with this property.

Consider the second order dynamical system:

$$\mu \ddot{x} = -\nabla f(x) - b\dot{x}, \tag{2}$$

where $\mu \geq 0$ & $b \geq 0$

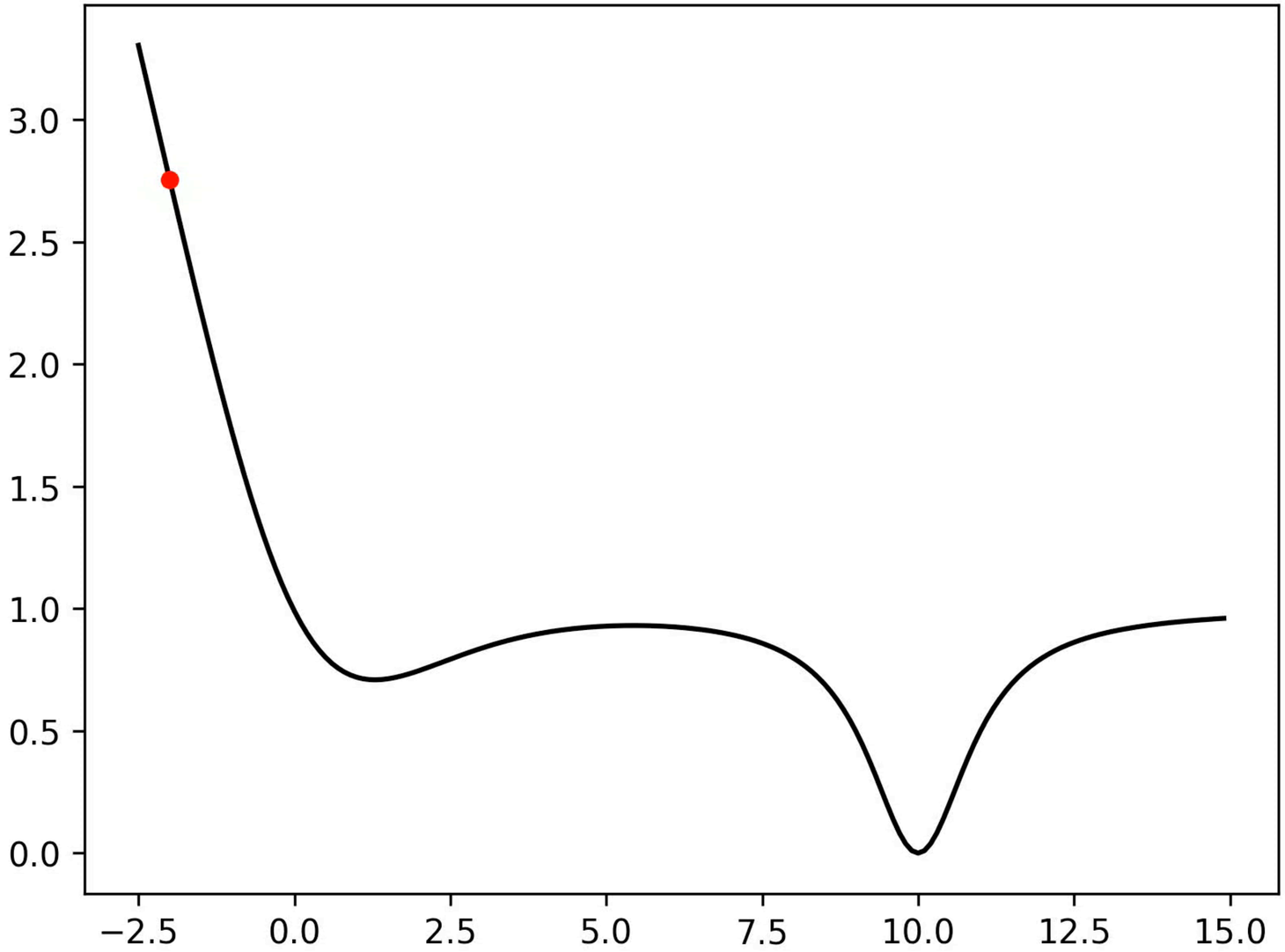
If we allow $\mu \to 0$ we have that the fixed points of (2) coincide with the stationary points of f.

Gradient Descent Near a local minimum

$$\dot{x} = -\nabla f(x)$$

Note how the algorithm slows down near the minimum

(The gradient is small)

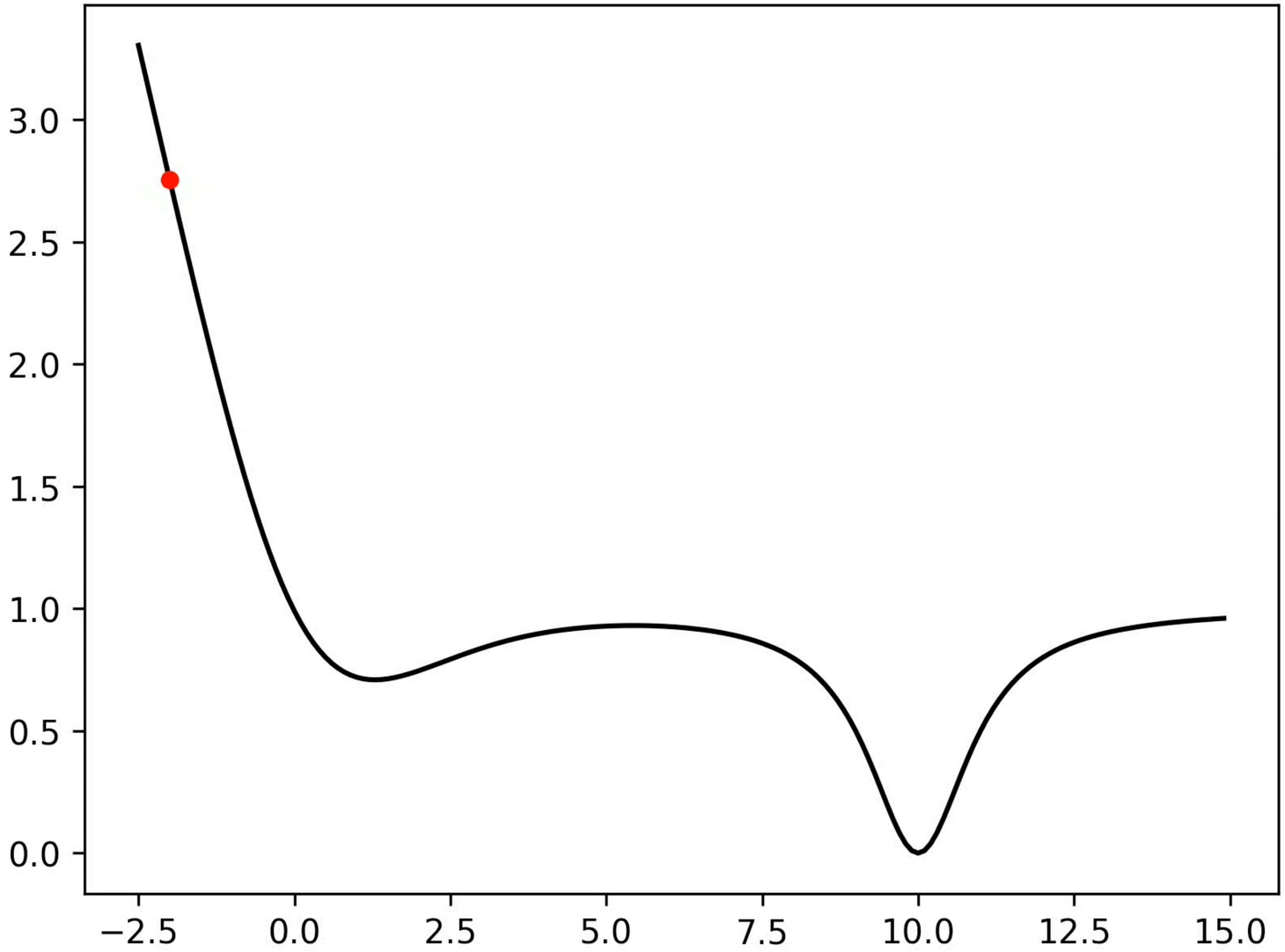


Gradient Descent Near a local minimum

$$\dot{x} = -\nabla f(x)$$

Note how the algorithm slows down near the minimum

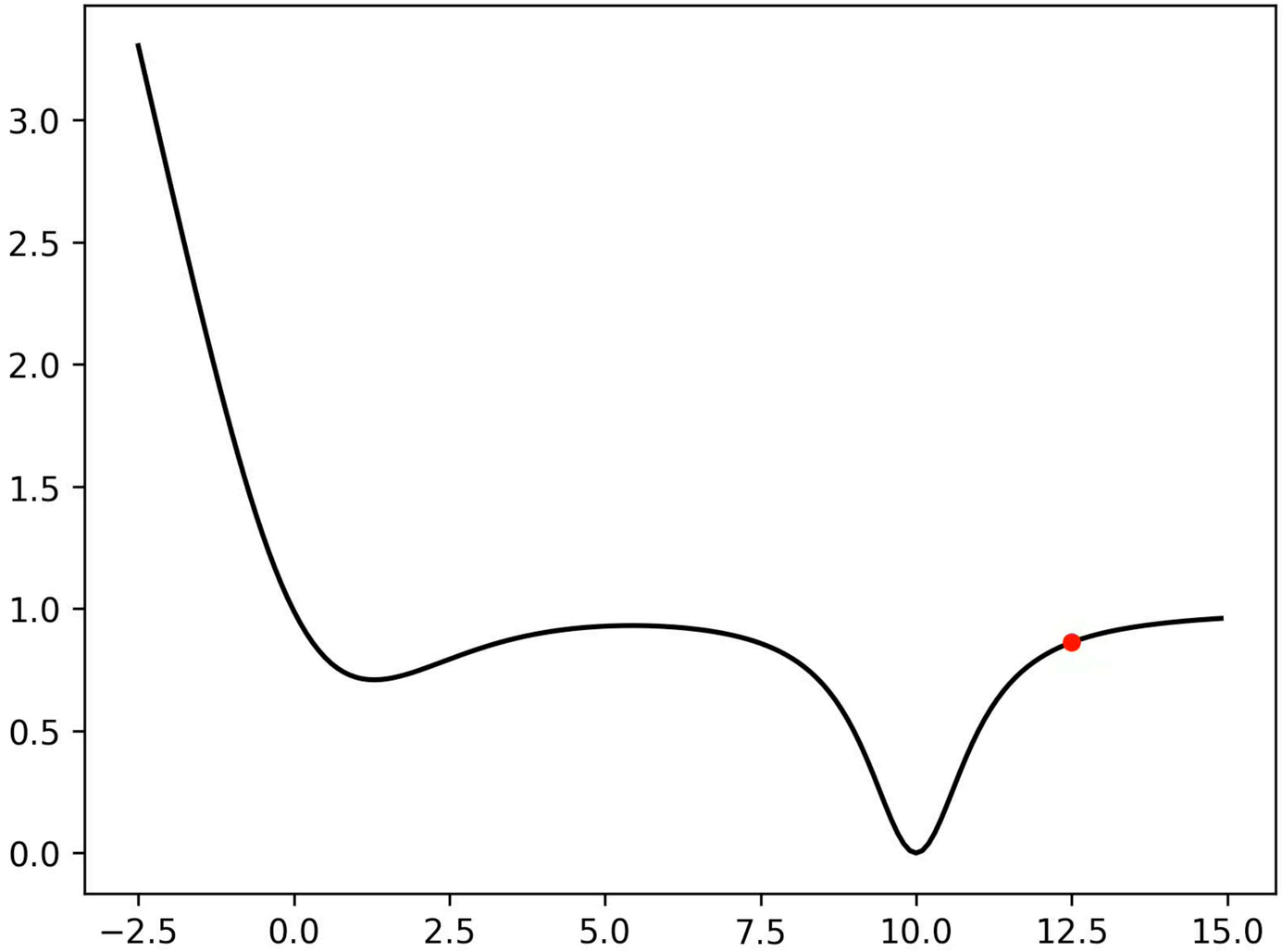
(The gradient is small)



Gradient Descent Around Flat Regions

$$\dot{x} = -\nabla f(x)$$

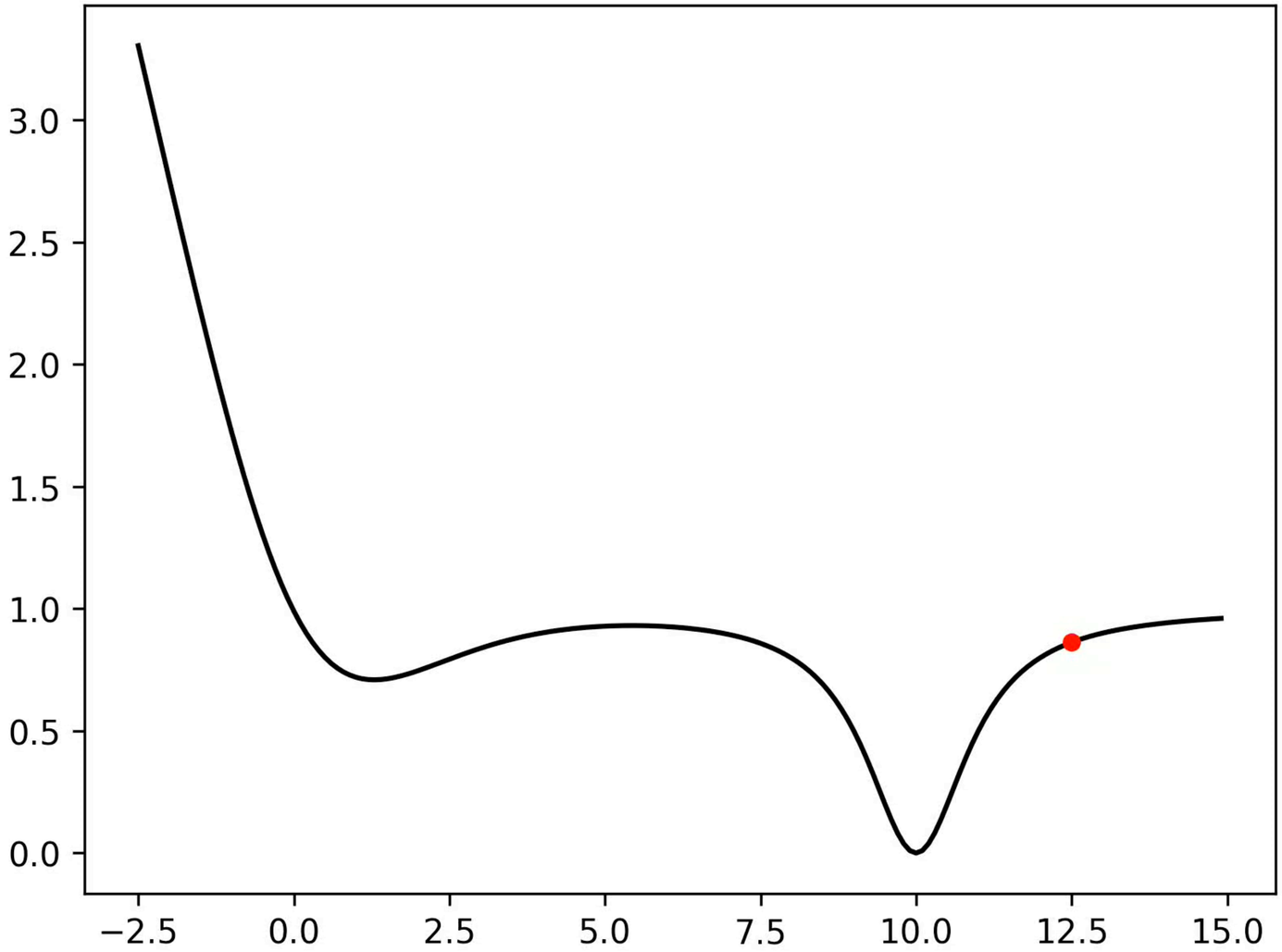
Note how the algorithm slows because the function is flat (second derivative is small)



Gradient Descent Around Flat Regions

$$\dot{x} = -\nabla f(x)$$

Note how the algorithm slows because the function is flat (second derivative is small)

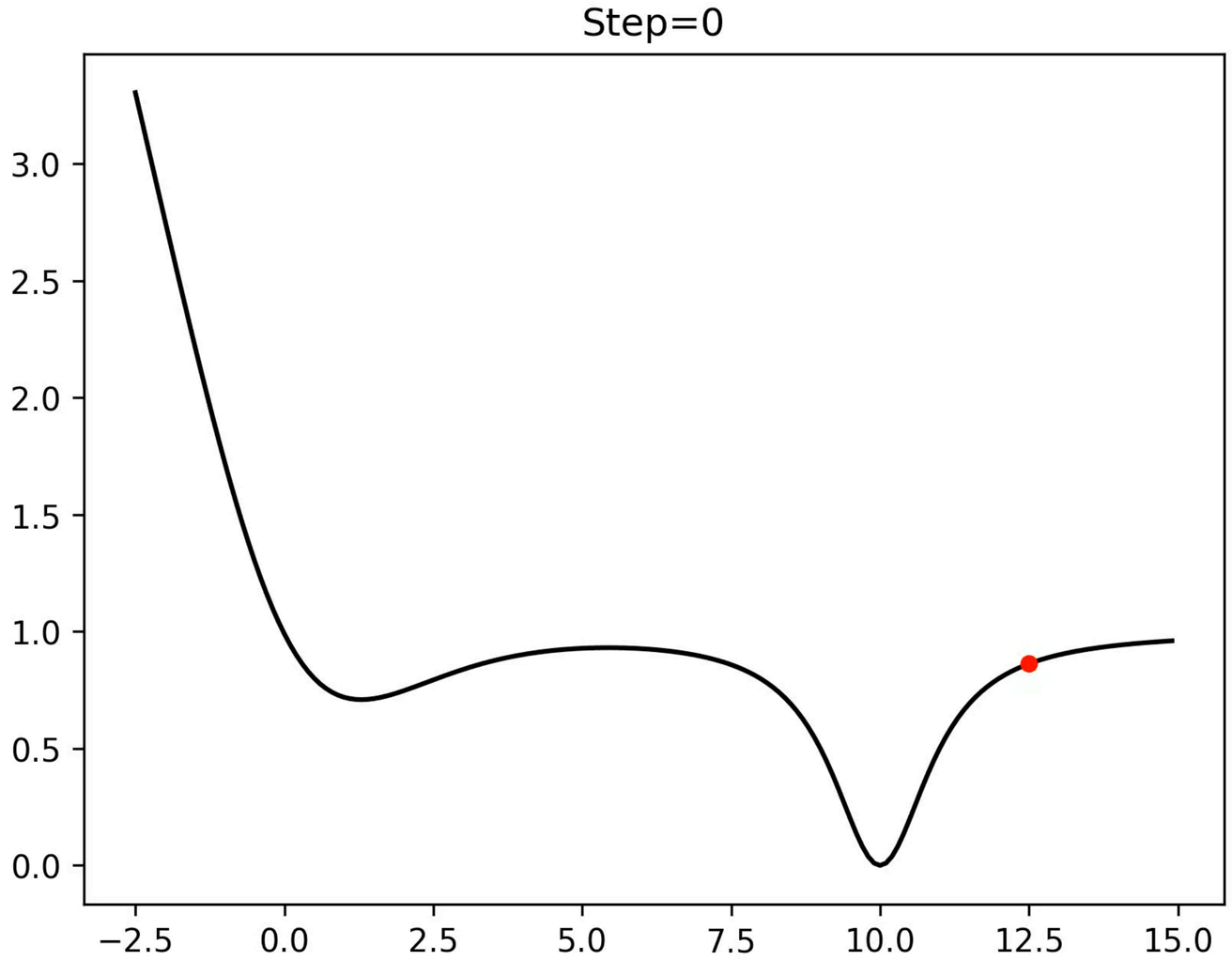


Gradient Descent with Momentum

$$\mu\ddot{x} + \dot{x} = -\nabla f(x)$$

Animation is with fixed momentum

(How most practitioners use it, but theory requires the momentum to decrease)

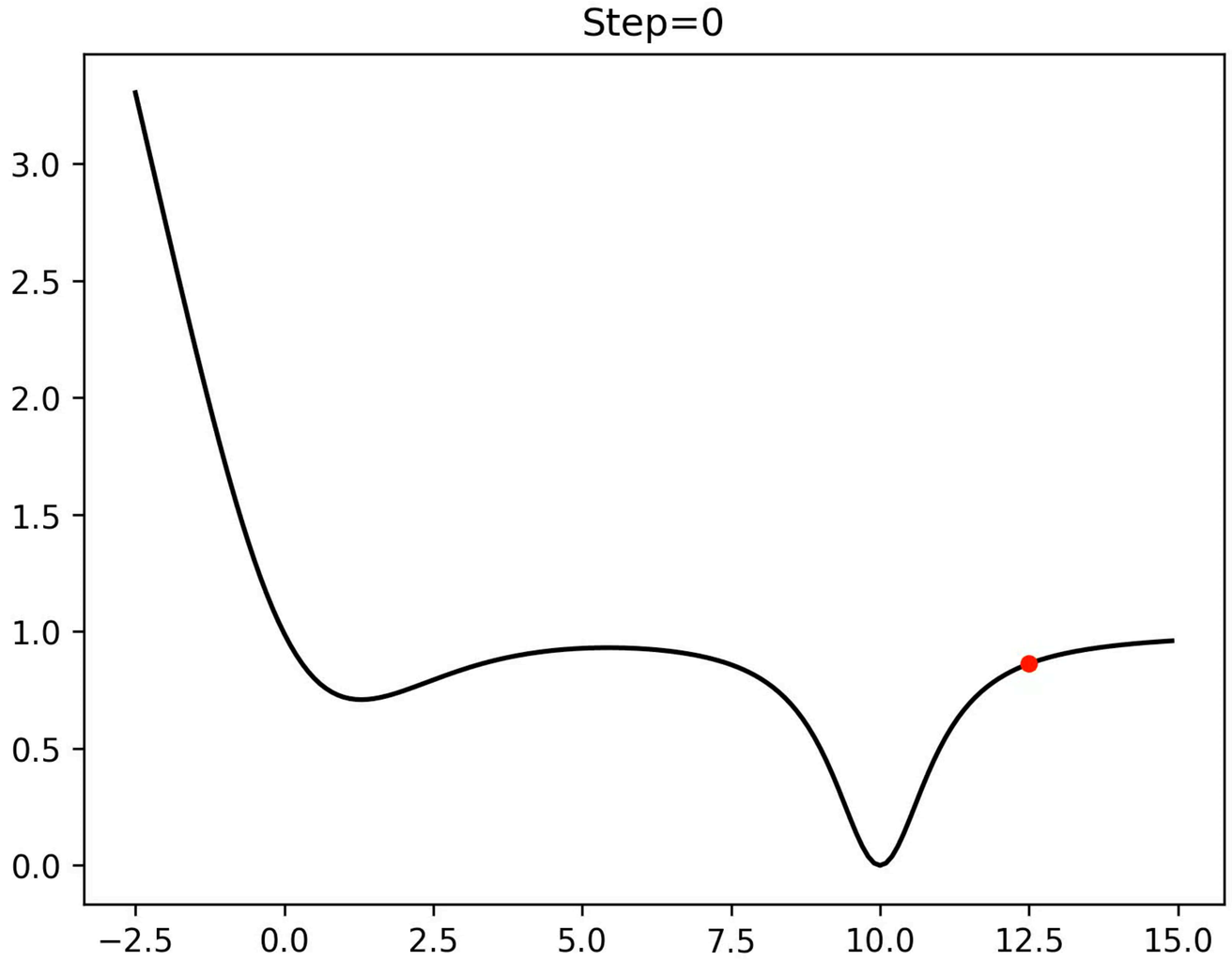


Gradient Descent with Momentum

$$\mu\ddot{x} + \dot{x} = -\nabla f(x)$$

Animation is with fixed momentum

(How most practitioners use it, but theory requires the momentum to decrease)



Gradient Descent with Momentum (Acceleration)

$$z_{k+1} = \beta z_k + \nabla L(w_k)$$
$$w_{k+1} = w_k - \tau z_{k+1}$$

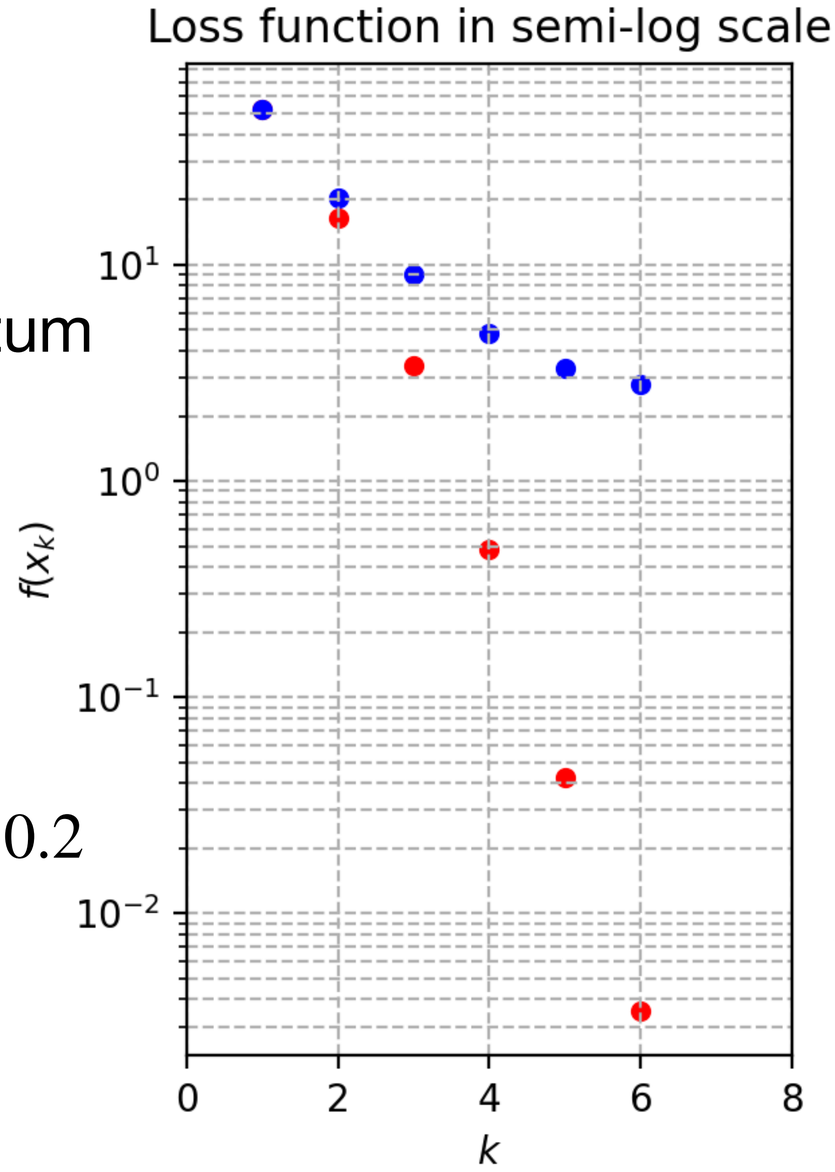
$\beta \geq 0$ controls momentum

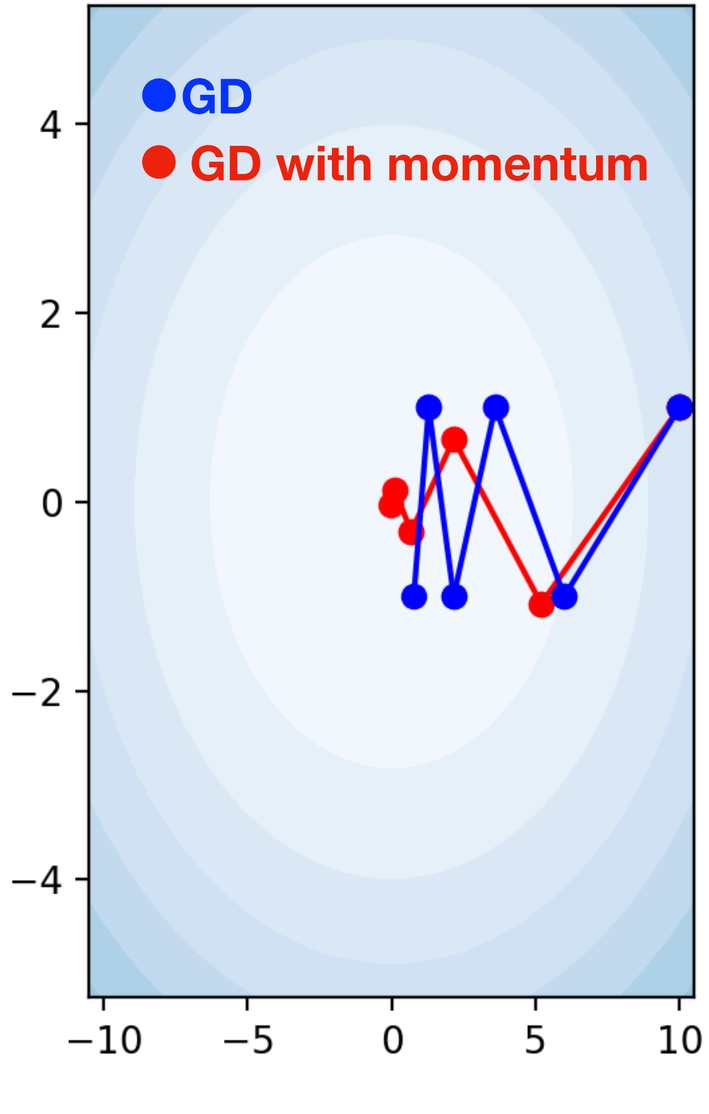
$$\min \frac{1}{2}(x_1^2 + \gamma x_2^2),$$

$\gamma = 5$, initial point $(10,1)$

G.D. with $\tau = 0.4$

G.D. with momentum $\tau = 0.4, \beta = 0.2$





Stochastic Gradient Descent

$$\min_{w \in \mathbb{R}^d} L(w) = \frac{1}{N} \sum_{i=1}^{N} l(w, \xi_i)$$

Loss functions in ML are a sum of functions

Loss associated with ith example ξ_i

''True" gradient: $\nabla L(w) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(w, \xi_i)$

Stochastic gradient: $\nabla \hat{L}(w) = \frac{1}{|B|} \sum_{i \in B} \nabla L(w, \xi_i)$

Where B is a subset of $\{1, \ldots, N\}$ without replacement of size $M \leq N$

M is the size of the batch

$\mathbb{E} \nabla \hat{L}(w_i) = \nabla L(w_i)$ i.e. stochastic gradient is unbiased

Why use a Stochastic Gradient?

Typical Objective function in ML (logistic regression):

$$\min_{w \in \mathbb{R}^d} L(w) = \frac{1}{N} \sum_{i=1}^{N} \underbrace{(-y_i x_i^\top w + \log(1 + e^{x_i^\top w}))}_{L(w, \xi_i)}$$

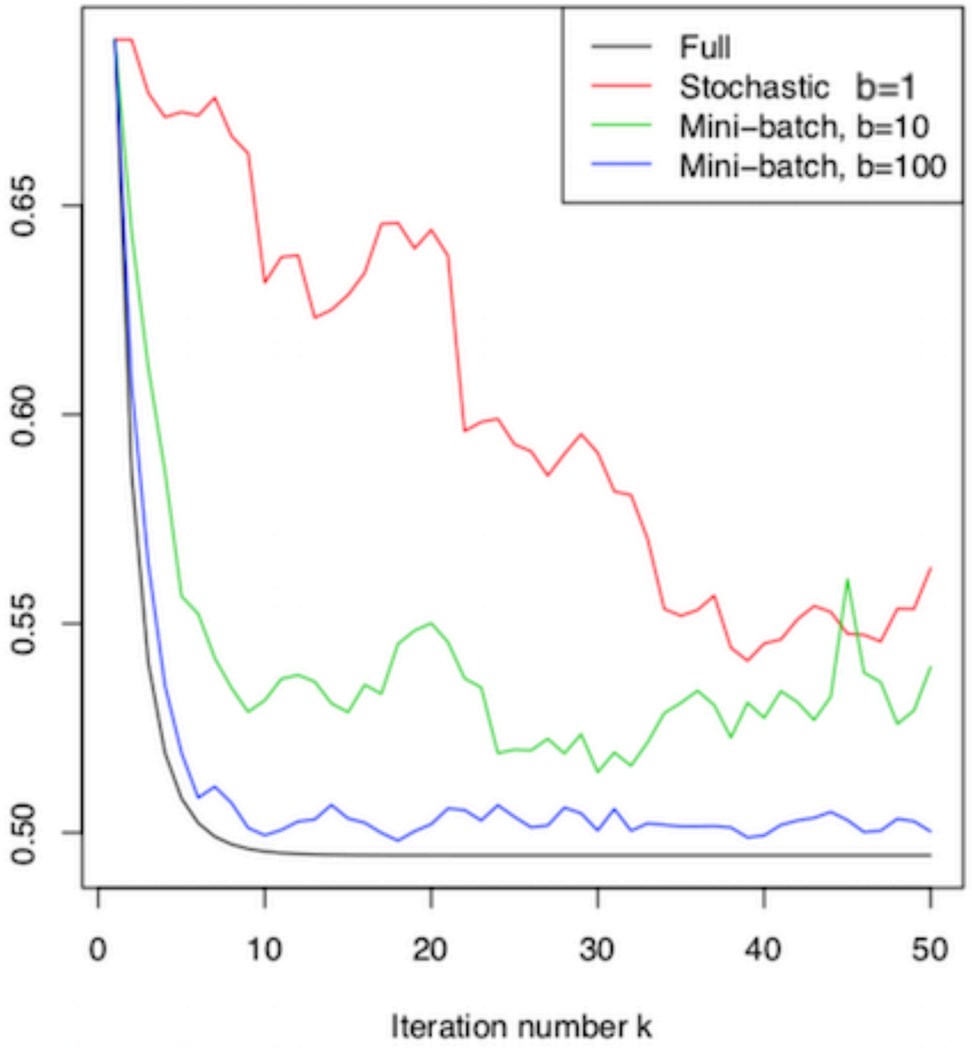
where $\xi_i = (y_i, x_i) \in \{\pm 1\} \times \mathbb{R}^d$ are labels and data respectively

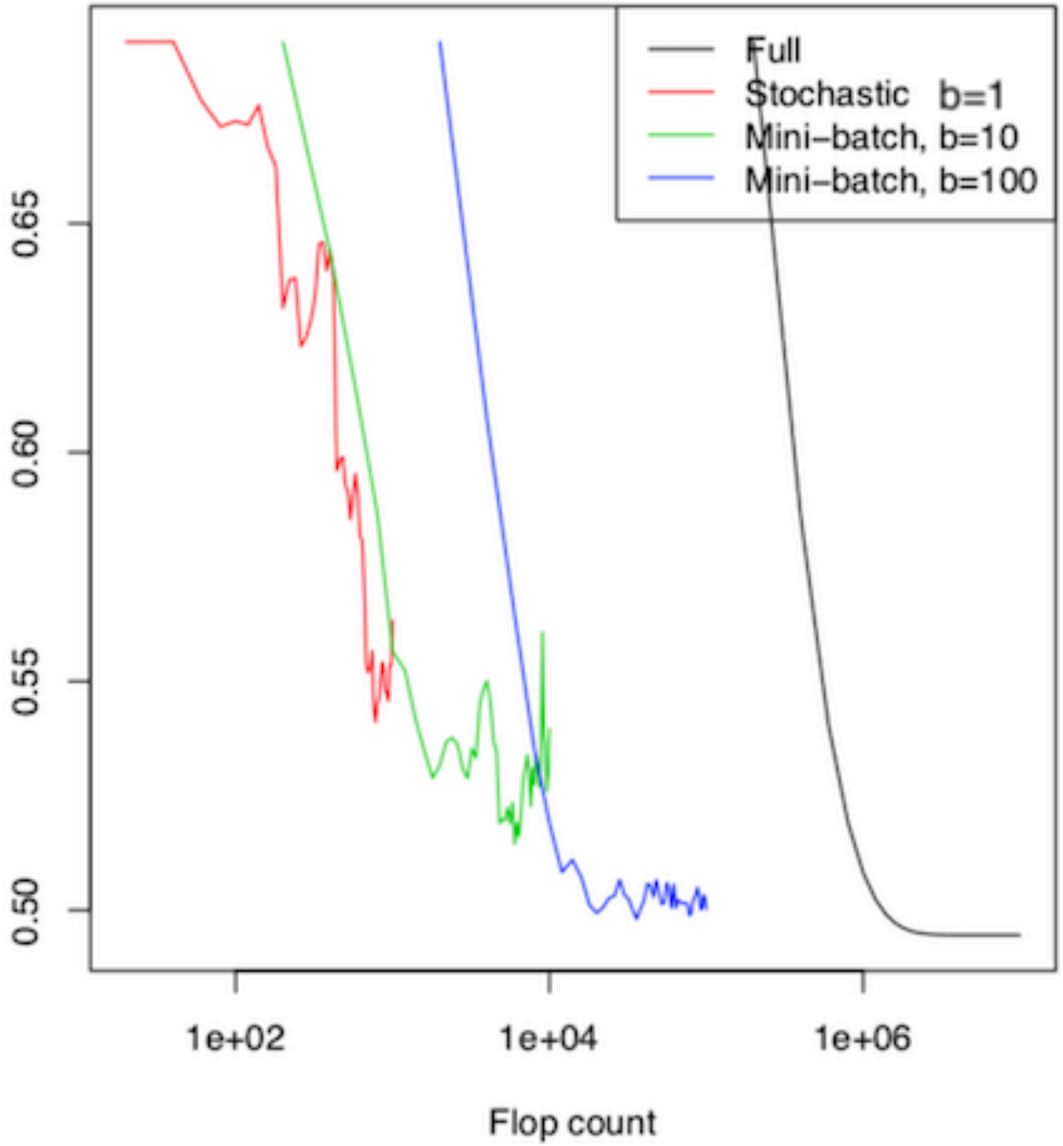
''True" gradient: $\nabla L(w) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(w, \xi_i)$ comp. cost $\approx N \times d$

b-Batch Stochastic gradient: $\nabla \widehat{L}(w) = \frac{1}{b} \sum_{i \in B} \nabla L(w, \xi_i)$ comp. cost $\approx b \times d$

For huge problems each iteration is much cheaper.
But is the algorithm faster?

Why use a Stochastic Gradient?





Why use a Stochastic Gradient?

Advantages of SG:

- Can easily adjust b to ensure gradients fit in available CPU/GPU constraints
- Noise due to mini-batch may help escape shallow local minima (theory is weak but some numerical evidence to support this)
- Since the exact minimum is less relevant in ML mini-batching is popular

Disadvantages of SG:

- Possibly high variance
- Setting the learning rate can be difficult

Constrained Optimization

Constrained Optimization

$$L^\star = \min_w L(w) \qquad L : \mathbb{R}^d \mapsto \mathbb{R}$$

$$g_i(w) = 0, \quad i = 1, \ldots, M \qquad g_i : \mathbb{R}^d \mapsto \mathbb{R}$$

Equality Constrained Problem

Lagrangian function: $\mathscr{L}(w, \lambda) = L(w) + \sum_{i=1}^{M} \lambda_i g_i(w)$

where $\lambda_i \in \mathbb{R}$ are the Lagrange multipliers

Constrained Optimization

$$L^\star = \min_w L(w) \qquad L : \mathbb{R}^d \mapsto \mathbb{R}$$
$$g_i(w) = 0, \quad i = 1,\ldots,M \qquad g_i : \mathbb{R}^d \mapsto \mathbb{R}$$

Equality Constrained Problem

Lagrangian function: $\mathscr{L}(w,\lambda) = L(w) + \sum_{i=1}^{M} \lambda_i g_i(w)$

where $\lambda_i \in \mathbb{R}$ are the Lagrange multipliers

$$L^\star = \min_w L(w) \qquad L : \mathbb{R}^d \mapsto \mathbb{R}$$
$$g_i(w) = 0, \quad i = 1,\ldots,M \qquad g_i : \mathbb{R}^d \mapsto \mathbb{R}$$
$$h_i(w) \le 0, \quad i = 1,\ldots,P \qquad h_i : \mathbb{R}^d \mapsto \mathbb{R}$$

General Constrained Problem

Lagrangian function:

$$\mathscr{L}(w,\lambda,\mu) = L(w) + \sum_{i=1}^{M} \lambda_i g_i(w) + \sum_{i=1}^{P} \mu_i h_i(w)$$

where $\lambda_i \in \mathbb{R}, \mu_i \ge 0$ are the Lagrange multipliers

Lagrangian Duality

$$\mathscr{L}(w, \lambda, \mu) = L(w) + \sum_{i=1}^{M} \lambda_i g_i(w) + \sum_{i=1}^{P} \mu_i h_i(w)$$

Lagrangian

$$Q(\lambda, \mu) = \min_{w} \mathscr{L}(w, \lambda, \mu)$$

Dual Function

$$Q^\star = \max_{\lambda_i \in \mathbb{R}, \mu_i \geq 0} Q(\lambda, \mu)$$

Dual Problem

Weak duality: $F^\star \geq Q^\star$

Holds even if the problem is non-convex, the dual problem is always concave

Strong duality: $F^\star = Q^\star$

Holds if the problem is convex, convex loss and constraints

Optimality Conditions (KKT)

$$L^\star = \min_w L(w)$$
$$g_i(w) = 0, \quad i = 1,\ldots,M$$
$$h_i(w) \leq 0, \quad i = 1,\ldots,P$$

First-order necessary conditions:

$$\nabla_w \mathscr{L}(w,\lambda) = \nabla_w L(w) + \sum_{i=1}^{M} \lambda_i \nabla_w g_i(w) = 0$$

$$g_i(w) = 0 \qquad i = 1,\ldots,M$$
$$h_i(w) \leq 0 \qquad i = 1,\ldots,P$$
$$\lambda_i h_i(w) = 0,\ \lambda_i \geq 0 \qquad i = 1,\ldots,P$$

Support Vector Machines

$$\min_w \frac{1}{2}||w||_2^2$$

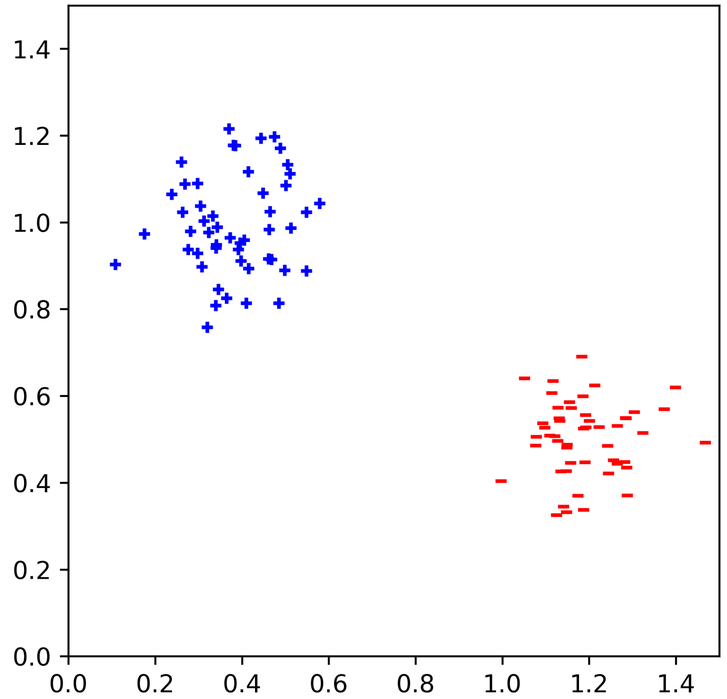
s.t. $y_i w^\top x_i \geq 1 \qquad i = 1,\ldots,N$

$\max_{\lambda_i \geq 0} L(\lambda)$:Dual Problem

$$L(\lambda) = \min_w \frac{1}{2}\|w\|_2^2 + \sum_{i=1}^{N} \lambda_i (1 - y_i w^\top x_i)$$

From Optimality conditions $w = \sum_{i=1}^{N} \lambda_i y_i x_i$ i.e $L(\lambda) = \sum_{i=1}^{N} -\frac{1}{2} \sum_{i,j}^{N} \lambda_i \lambda_j y_i y_j x_i^\top x_j$

$$\max_{\lambda_i \geq 0} L(\lambda) = \sum_{i=1}^{N} -\frac{1}{2} \sum_{i,j}^{N} \lambda_i \lambda_j y_i y_j x_i^\top x_j$$



Dual problem only contains inner products (use kernel trick to increase feature space without additional cost)

Optimisation in ML — Remarks

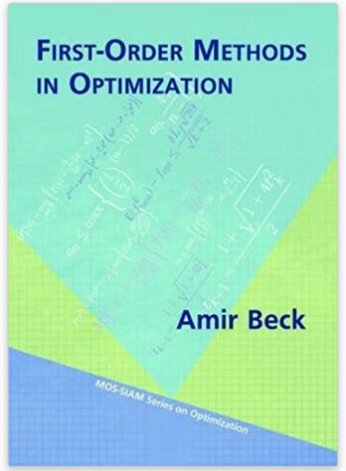
Optimisation is everywhere in ML.

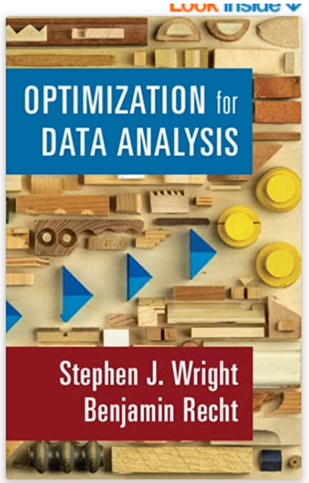
Both the model and the optimisation algorithm are critical to the success of ML projects, especially if the model is non-convex and has noise.

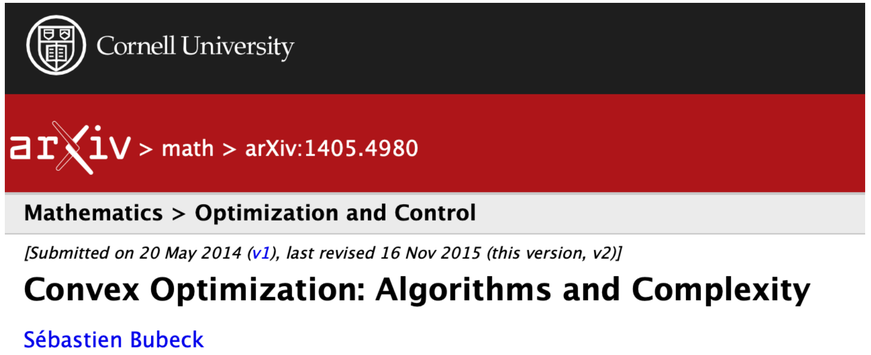
Acceleration may help, and choosing a learning rate can be time-consuming.

Scaling data is essential before calling an optimisation algorithm.

The **theory of optimisation is vital** for a deeper understanding of ML






Cornell University

arXiv > math > arXiv:1405.4980

Mathematics > Optimization and Control

[Submitted on 20 May 2014 (v1), last revised 16 Nov 2015 (this version, v2)]

Convex Optimization: Algorithms and Complexity

Sébastien Bubeck

Contents and Aims

Aims:

- Introduce a mathematical toolbox as used in ML
- Demonstrate mathematics with different ML models

A. **Linear Algebra**
    1. Matrices and vectors
    2. Measuring similarity
    3. Eigenvectors/Eigenvalues
    4. Matrix Factorisations
B. **Calculus**
C. **Optimisation**
    1. Classes of Optimization Models/Algorithms
    2. Gradient Descent (Stochastic, Acceleration, Momentum)
    3. Optimality Conditions (constrained case)