

# Statistical Machine Learning: Exercise 1

Yi Cui, 2758172  
Lingwei Liu, 2659255



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Sommersemester 2020

## Aufgabe 1: Linear Algebra Refresher

1a)

A colleague of yours suggests that for matrices, addition and multiplication works similarly as for scalars, in particular, that the commutative, distributive and associative properties hold. Are these statements correct? Prove or disprove them analytically for both operations and all three properties considering three matrices A, B, C of size  $n \times n$ .

answer:

$$\underline{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & \ddots & & a_{2n} \\ \vdots & & & \vdots \\ a_{n1} & \cdots & & a_{nn} \end{bmatrix} \quad \underline{B} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & \ddots & & b_{2n} \\ \vdots & & & \vdots \\ b_{n1} & \cdots & & b_{nn} \end{bmatrix} \quad \underline{C} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & \ddots & & c_{2n} \\ \vdots & & & \vdots \\ c_{n1} & \cdots & & c_{nn} \end{bmatrix}$$

associative:

$$\begin{aligned} (\underline{A} \cdot \underline{B}) \underline{C} &= \begin{bmatrix} \sum_{i=1}^n a_{1i} b_{i1} & \cdots & \sum_{i=1}^n a_{1i} b_{in} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n a_{ni} b_{i1} & \cdots & \sum_{i=1}^n a_{ni} b_{in} \end{bmatrix} \cdot \begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{j=1}^n \left( \sum_{i=1}^n a_{1i} b_{ij} \right) \cdot c_{j1} & \cdots & \sum_{j=1}^n \left( \sum_{i=1}^n a_{1i} b_{ij} \right) \cdot c_{jn} \\ \vdots & \ddots & \vdots \\ \sum_{j=1}^n \left( \sum_{i=1}^n a_{ni} b_{ij} \right) \cdot c_{j1} & \cdots & \sum_{j=1}^n \left( \sum_{i=1}^n a_{ni} b_{ij} \right) \cdot c_{jn} \end{bmatrix} \end{aligned}$$

$$\begin{aligned}\underline{A}(\underline{B} \cdot \underline{C}) &= \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} \sum_{i=1}^n b_{1i}c_{i1} & \cdots & \sum_{i=1}^n b_{1i}c_{in} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n b_{ni}c_{i1} & \cdots & \sum_{i=1}^n b_{ni}c_{in} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{j=1}^n a_{1j} \cdot \left(\sum_{i=1}^n b_{ji}c_{i1}\right) & \cdots & \sum_{j=1}^n a_{1j} \cdot \left(\sum_{i=1}^n b_{ji}c_{in}\right) \\ \vdots & \ddots & \vdots \\ \sum_{j=1}^n a_{nj} \cdot \left(\sum_{i=1}^n b_{ji}c_{i1}\right) & \cdots & \sum_{j=1}^n a_{nj} \cdot \left(\sum_{i=1}^n b_{ji}c_{in}\right) \end{bmatrix}\end{aligned}$$

element analysis:

$$\sum_{j=1}^n \left(\sum_{i=1}^n a_{1i}b_{ij}\right) \cdot c_{j1} \neq \sum_{j=1}^n a_{1j} \cdot \left(\sum_{i=1}^n b_{ji}c_{i1}\right)$$

$$(\underline{A} \cdot \underline{B})\underline{C} \neq \underline{A}(\underline{B} \cdot \underline{C})$$

$\Rightarrow$  associative disproved

distribute:

$$\begin{aligned}(\underline{A} + \underline{B}) \cdot \underline{C} &= \begin{bmatrix} a_{11} + b_{11} & \cdots & a_{1n} + b_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} + b_{n1} & \cdots & a_{nn} + b_{nn} \end{bmatrix} \cdot \begin{bmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{i=1}^n (a_{1i} + b_{1i}) \cdot c_{i1} & \cdots & \sum_{i=1}^n (a_{1i} + b_{1i}) \cdot c_{in} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n (a_{ni} + b_{ni}) \cdot c_{i1} & \cdots & \sum_{i=1}^n (a_{ni} + b_{ni}) \cdot c_{in} \end{bmatrix} \\ &= \begin{bmatrix} \sum_{i=1}^n a_{1i} \cdot c_{i1} & \cdots & \sum_{i=1}^n a_{1i} \cdot c_{in} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n a_{ni} \cdot c_{i1} & \cdots & \sum_{i=1}^n a_{ni} \cdot c_{in} \end{bmatrix} + \begin{bmatrix} \sum_{i=1}^n b_{1i} \cdot c_{i1} & \cdots & \sum_{i=1}^n b_{1i} \cdot c_{in} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n b_{ni} \cdot c_{i1} & \cdots & \sum_{i=1}^n b_{ni} \cdot c_{in} \end{bmatrix} \\ &= \underline{A} \cdot \underline{C} + \underline{B} \cdot \underline{C}\end{aligned}$$

$\Rightarrow$  distribute proved

commutative:

$$\underline{A} \cdot \underline{B} = \begin{bmatrix} \sum_{i=1}^n a_{1i}b_{i1} & \cdots & \sum_{i=1}^n a_{1i}b_{in} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n a_{ni}b_{i1} & \cdots & \sum_{i=1}^n a_{ni}b_{in} \end{bmatrix}$$

$$\underline{B} \cdot \underline{A} = \begin{bmatrix} \sum_{i=1}^n b_{1i}a_{i1} & \cdots & \sum_{i=1}^n b_{1i}a_{in} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^n b_{ni}a_{i1} & \cdots & \sum_{i=1}^n b_{ni}a_{in} \end{bmatrix}$$

element analysis:

$$\sum_{i=1}^n b_{1i}a_{i1} \neq \sum_{i=1}^n b_{ni}a_{i1}$$

$$\underline{A} \cdot \underline{B} \neq \underline{B} \cdot \underline{A}$$

⇒ commutative disproved

1b)

Given the following matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 6 \\ 0 & 0 & 1 \end{bmatrix}$$

analytically compute its inverse  $A^{-1}$  and illustrate the steps. What algorithm did you use? If we change the matrix to

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 6 \\ 1 & 0 & 0 \end{bmatrix}$$

is it still invertible? Why or why not?

answer:

method 1: elementray transformation

$$\begin{aligned} [\underline{A} \quad \underline{I}] &\Rightarrow [\underline{I} \quad \underline{A}^{-1}] \\ \left[ \begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 1 & 4 & 6 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] &\Rightarrow \left[ \begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \\ \Rightarrow \left[ \begin{array}{ccc|ccc} 0 & 1 & 0 & -\frac{1}{2} & \frac{1}{2} & \frac{3}{2} \\ 1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] &\Rightarrow \left[ \begin{array}{ccc|ccc} 1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 1 & 0 & -\frac{1}{2} & \frac{1}{2} & \frac{3}{2} \\ 0 & 0 & 1 & 0 & 0 & 1 \end{array} \right] \end{aligned}$$

method 2: use adjoint matrix

$$\underline{A}^{-1} = \frac{1}{|A|} \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{n1} & \cdots & A_{nn} \end{bmatrix} \Rightarrow \text{adjoint}$$

where  $A_{ij}$  is algebraic cofactor

$$= \frac{1}{\det(\underline{A})} \cdot \begin{bmatrix} 4 & -2 & 0 \\ -1 & 1 & -3 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ -\frac{1}{2} & \frac{1}{2} & \frac{3}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

with  $\det(\underline{A}) = |A| = 2$

If  $\underline{A}$  is changed to  $\begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 6 \\ 1 & 0 & 0 \end{bmatrix}$

$\det(\underline{A}) = 0 \implies A$  is a singular matrix  $\implies$  irreversible

1c)

Write the definition of the right and left Moore-Penrose pseudoinverse of a generic matrix  $A \in \mathbb{R}^{n \times m}$ . Given  $A \in \mathbb{R}^{n \times m}$ , which one does exist? Write down the equation for computing it, specifying the dimensionality of the matrices in the intermediate steps.

Answer:

for  $A \in \mathbb{R}^{n \times m}$

left Moore-Penrose pseudoinverse:  $\underline{A}_{left}^f = (\underline{A}^T \cdot \underline{A})^{-1} \cdot \underline{A}^T$

right Moore-Penrose pseudoinverse:  $\underline{A}_{right}^f = \underline{A}^T \cdot (\underline{A} \cdot \underline{A}^T)^{-1}$

dimensional analysis:

if  $A \in \mathbb{R}^{2 \times 3}$

$$\begin{aligned} \underline{A}_{left}^f &= (\underline{A}^T \cdot \underline{A})^{-1} \cdot \underline{A}^T \\ &\Rightarrow (3 \times 2) \cdot (2 \times 3) \cdot (3 \times 2) \Rightarrow (3 \times 2) \\ \underline{A}_{right}^f &= \underline{A}^T \cdot (\underline{A} \cdot \underline{A}^T)^{-1} \\ &\Rightarrow (3 \times 2) \cdot [(2 \times 3) \cdot (3 \times 2)] \Rightarrow (3 \times 2) \\ &\Rightarrow \underline{A}^f \in \mathbb{R}^{2 \times 3} \end{aligned}$$

1d)

What are eigenvectors and eigenvalues of a matrix  $A$ ? Briefly explain why they are important in machine learning.

answer:

eigenvector: represent the orientation of each principal components

eigenvalue: represent the weight of each principal components

both of them represent the component distribution of all features (PCA perspective)

**Aufgabe 2: Statistics Refresher**

2a)

Let  $\Omega$  be a finite set and  $P: \Omega \Rightarrow \mathbb{R}$  a probability measure that (by definition) satisfies  $P(\omega) \geq 0$  for all  $\omega \in \Omega$  and  $\sum_{\omega \in \Omega} P(\omega) = 1$ . Let  $f: \Omega \in \mathbb{R}$  be an arbitrary function on  $\Omega$ .

1) Write the definition of expectation and variance of  $f$  and discuss if they are linear operators.

2) You are given a set of three dice {A,B,C}. The following table contains results obtained from rolling these three standard, six-sided dice 18 times each. Each row corresponds to one die, each column shows how often a number was rolled on the respective die.

	1	2	3	4	5	6
A	4	1	6	2	1	4
B	5	6	1	1	4	1
C	3	3	4	2	3	3

Estimate the expectation and the variance for each die using unbiased estimators. (Show your computations).

3) Estimate the KL-divergence between each die's distribution and the uniform distribution. According to your results, which one of them is closest to a fair, uniform die?

answer:

for  $\Omega$  is a finite set

Expectation:

$$E[f(\omega)] = \sum_{\omega \in \Omega} f(\omega) \cdot P(\omega)$$

Variance:

$$\begin{aligned} V[f(\omega)] &= \sum_{\omega \in \Omega} P(\omega) \cdot [f(\omega) - E(\omega)]^2 \\ &= E[[f(\omega) - E[f(\omega)]]^2] \end{aligned}$$

For Expectation: the rules of expectation prove it:

$$\begin{aligned} E[aX] &= aE[X] \\ E[x + y] &= E[X] + E[Y] \\ E\left[\sum_i a_i x_i\right] &= \sum_i a_i E[x_i] \end{aligned}$$

Variance is essentially an expectation operation

$$\begin{aligned} V[f(\omega)] &= E[[f(\omega) - E[f(\omega)]]^2] \\ &= E[f(\omega)^2 - 2E[f(\omega)] \cdot E[f(\omega)] + (E[f(\omega)])^2] \\ &= E[f(\omega)^2] - (E[f(\omega)])^2 \\ &= \sum_{\omega \in \Omega} P(\omega) \cdot (f(\omega))^2 - E[f(\omega)]^2 \end{aligned}$$

2)

For diece A set  $P(A) = 1 \Rightarrow P(x, A) = P(x|A) \cdot P(A) = P(x|A) = P_A(x)$

$$\begin{aligned}\bar{x}_A &= \sum_{i=1}^6 x_i \cdot P(x_i, A) = 1 \cdot 2/9 + 2 \cdot 1/18 + 3 \cdot 1/3 + 4 \cdot 1/9 + 5 \cdot 1/18 + 6 \cdot 2/9 \\ &= 61/18 \approx 3.3889 \\ \text{unbiased } V[(x, A)] &= \frac{1}{N-1} \cdot \sum_{i=1}^{N=18} (X_i - \bar{X}_A)^2 \quad (\text{Bessel's Correction}) \\ &= \frac{1}{17} \cdot (4 \cdot (1 - \frac{61}{18})^2 + 1 \cdot (2 - \frac{61}{18})^2 + 6 \cdot (3 - \frac{61}{18})^2 + 2 \cdot (4 - \frac{61}{18})^2 \\ &\quad + 1 \cdot (5 - \frac{61}{18})^2 + 4 \cdot (6 - \frac{61}{18})^2) \\ &= \frac{1013}{306} \approx 3.3105\end{aligned}$$

For diece B set  $P(B) = 1 \Rightarrow P(x, B) = P(x|B) \cdot P(B) = P(x|B) = P_B(x)$

$$\begin{aligned}\bar{x}_B &= \sum_{i=1}^6 x_i \cdot P(x_i, B) = 1 \cdot 5/18 + 2 \cdot 1/3 + 3 \cdot 1/18 + 4 \cdot 1/18 + 5 \cdot 2/9 + 6 \cdot 1/18 \\ &= 25/9 \approx 2.7778 \\ \text{unbiased } V[(x, B)] &= \frac{1}{N-1} \cdot \sum_{i=1}^{N=18} (X_i - \bar{X}_B)^2 \\ &= \frac{1}{17} \cdot (5 \cdot (1 - \frac{25}{9})^2 + 6 \cdot (2 - \frac{25}{9})^2 + 1 \cdot (3 - \frac{25}{9})^2 + 1 \cdot (4 - \frac{25}{9})^2 \\ &\quad + 4 \cdot (5 - \frac{25}{9})^2 + 1 \cdot (6 - \frac{25}{9})^2) \\ &= \frac{460}{153} \approx 3.0065\end{aligned}$$

For diece C set  $P(C) = 1 \Rightarrow P(x, C) = P(x|C) \cdot P(C) = P(x|C) = P_C(x)$

$$\begin{aligned}\bar{x}_C &= \sum_{i=1}^6 x_i \cdot P(x_i, C) = 1 \cdot 1/6 + 2 \cdot 1/6 + 3 \cdot 2/9 + 4 \cdot 1/9 + 5 \cdot 1/6 + 6 \cdot 1/6 \\ &= 31/9 \approx 3.4444 \\ \text{unbiased } V[(x, C)] &= \frac{1}{N-1} \cdot \sum_{i=1}^{N=18} (X_i - \bar{X}_C)^2 \\ &= \frac{1}{17} \cdot (3 \cdot (1 - \frac{31}{9})^2 + 3 \cdot (2 - \frac{31}{9})^2 + 4 \cdot (3 - \frac{31}{9})^2 + 2 \cdot (4 - \frac{31}{9})^2 \\ &\quad + 3 \cdot (5 - \frac{31}{9})^2 + 3 \cdot (6 - \frac{31}{9})^2) \\ &= \frac{472}{153} \approx 3.0850\end{aligned}$$

3)

Dice A :

$$\begin{aligned}
 KL &= \sum_{x=1}^6 P_A(x) \cdot \ln \frac{P_A(x)}{Q(x)} \\
 &= \frac{2}{9} \cdot \ln\left(\frac{2}{9}/\frac{1}{6}\right) + \frac{1}{18} \cdot \ln\left(\frac{1}{18}/\frac{1}{6}\right) + \frac{1}{3} \cdot \ln\left(\frac{1}{3}/\frac{1}{6}\right) + \frac{1}{9} \cdot \ln\left(\frac{1}{9}/\frac{1}{6}\right) + \frac{1}{18} \cdot \ln\left(\frac{1}{18}/\frac{1}{6}\right) + \frac{2}{9} \cdot \ln\left(\frac{2}{9}/\frac{1}{6}\right) \\
 &= 0.1918
 \end{aligned}$$

Dice B :

$$\begin{aligned}
 KL &= \sum_{x=1}^6 P_B(x) \cdot \ln \frac{P_B(x)}{Q(x)} \\
 &= \frac{5}{18} \cdot \ln\left(\frac{5}{18}/\frac{1}{6}\right) + \frac{1}{3} \cdot \ln\left(\frac{1}{3}/\frac{1}{6}\right) + \frac{1}{18} \cdot \ln\left(\frac{1}{18}/\frac{1}{6}\right) + \frac{1}{18} \cdot \ln\left(\frac{1}{18}/\frac{1}{6}\right) + \frac{2}{9} \cdot \ln\left(\frac{2}{9}/\frac{1}{6}\right) + \frac{1}{18} \cdot \ln\left(\frac{1}{18}/\frac{1}{6}\right) \\
 &= 0.2538
 \end{aligned}$$

Dice C :

$$\begin{aligned}
 KL &= \sum_{x=1}^6 P_C(x) \cdot \ln \frac{P_C(x)}{Q(x)} \\
 &= \frac{1}{6} \cdot \ln\left(\frac{1}{6}/\frac{1}{6}\right) + \frac{1}{6} \cdot \ln\left(\frac{1}{6}/\frac{1}{6}\right) + \frac{2}{9} \cdot \ln\left(\frac{2}{9}/\frac{1}{6}\right) + \frac{1}{9} \cdot \ln\left(\frac{1}{9}/\frac{1}{6}\right) + \frac{1}{6} \cdot \ln\left(\frac{1}{6}/\frac{1}{6}\right) + \frac{1}{6} \cdot \ln\left(\frac{1}{6}/\frac{1}{6}\right) \\
 &= 0.0189
 \end{aligned}$$

Dice B &gt; Dice A &gt; Dice C

⇒ Dice C is closest to a fair uniform die



2b)

Consider the following three statements: a) A person with a cold has back-pain 30 % of the time. b) 3% of the world population has a cold. c) 10% of those who do not have a cold still have back-pain. 1) Identify random variables from the statements above and define a symbol for each of them. 2) Define the domain of each random variable. 3) Represent the three statements above with your random variables. 4) If you suffer from back-pain, what are the chances that you suffer from a cold? (Show all the intermediate steps.)

answer:

1)

A:{ a Person has a cold }      B:{ a Person has back-pain }

2)

C:{ World Population }      D:{ healthy People who do not have cold }

3)

a) { A person with a cold has back-pain 30% of the time }

$$\Rightarrow P(B|A) = 30\%$$

b) { 3% of the world population has a cold }

$$\Rightarrow P(A|C) = 3\% \quad \text{with } P(C) = 1 \Rightarrow P(A) = 3\%$$

c) { 10% of those who do not have a cold still have back-pain }

$$\Rightarrow P(B|D) = 10\% \quad \text{with } P(D) = 1 - P(A|C) = 97\%$$

4)

$$\begin{aligned} P(B) &= P(B, A) + P(B, D) \\ &= P(B|A) \cdot P(A) + P(B|D) \cdot P(D) \\ &= 30\% \cdot 3\% + 10\% \cdot 97\% = 10.6\% \end{aligned}$$

Bayes - theoren  $\Rightarrow$ 

$$\begin{aligned} P(A|B) &= \frac{P(B|A) \cdot P(A)}{P(B)} \\ &= \frac{30\% \cdot 3\%}{10.6\%} = 8.49\% \end{aligned}$$

2c)

You are a brilliant scientist trying to create a cure to stop the dangerous virus C. To produce the needed molecules, you need to use the bacterium FC with a specific mutation HC-SML. Sadly, the mutation is unstable and each day, the bacterium has a 45% chance of replicating with the mutation. Luckily, the bacterium also has a 2.3% chance to develop the mutation when it previously did not have it.

1) We describe the current state at time  $t$  with vector  $\vec{s}_t = (m \quad \tilde{m})^T$  where  $m$  is the chance your vial currently contains the bacterium with the mutation and  $\tilde{m}$  describes the probability it doesn't. Formalize the problem of computing the next state  $\vec{s}_{t+1}$  using matrix notation (this is called a Markov chain). To simplify the problem, we assume that the whole culture either has the mutation or it does not, percentages of mutated bacteria do not have to be modeled.

2) You have a small culture with the mutated bacterium and feel close to saving humanity. You decide to make a simulation to predict the outcome of breeding the bacterium for several days. What is the probability that the mutation will still be present in your culture after 20 generations? Code a simulation in Python, run it, include a snippet of your code, and attach a plot with the probability after each generation. Will your brilliant plan succeed?

3) After starting another project, you forget about your culture for a while and remember it weeks later. You wonder whether there is still a chance that the mutation is present. Run your simulation for as long as it is necessary to obtain a stable probability. How likely is it that the culture contains the mutation in the long run and after how many timesteps do the ratios stop to change significantly? In addition, find a mathematical expression for a stable probability. Present the equation and the solution. Does the probability actually converge to this solution?

answer:

$$1) \vec{s}_t = \begin{pmatrix} m \\ \tilde{m} \end{pmatrix} \text{ with probability matrix : } \underline{T} = \begin{bmatrix} 45\% & 2.3\% \\ 55\% & 97.7\% \end{bmatrix}$$

$$\Rightarrow \vec{s}_{t+1} = \underline{T} \cdot \vec{s}_t = \begin{bmatrix} 45\% & 2.3\% \\ 55\% & 97.7\% \end{bmatrix} \begin{pmatrix} m \\ \tilde{m} \end{pmatrix} = \begin{pmatrix} 45\%m + 2.3\%\tilde{m} \\ 55\%m + 97.7\%\tilde{m} \end{pmatrix}$$

2)

```
# -*- coding: utf-8 -*-
```

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def markov_chain(initial=np.array([1e-5, 1-1e-5]), epoch=20):
    """
    markov_chain to compute bacterium with specific mutation
    :param initial: [array], initial bacterium (default value:np.array([1e-5, 1-1e-5]) )
    :param epoch:[int], iteration times
    :return:
    mutation [list], the record of evaluation history
    """
    # assign probability matrix
    T = np.array([[.45, .023], [.55, .977]])

    # initial mutation hist list
    mutation = []

    # assign initial value
    mutation.append(initial[0])

    for i in range(epoch):
```

```

    initial = np.dot(T,initial)
    mutation.append(initial[0])

    return mutation

def plot_mutation(mutation):
    """
    plot evaluation history of bacterium with mutation
    :param mutation: , the record of evaluation history
    :return:
    """
    # assign plot data in dict form
    data = {"mutation_proportion": mutation,
            "iteration": np.linspace(0, len(mutation), len(mutation), dtype=np.int16)}

    # initial figure
    plt.figure()
    plt.plot("iteration", "mutation_proportion", data=data)
    plt.grid()
    plt.xlabel("iteration")
    plt.ylabel("mutation_proportion")
    plt.title('evaluation in iteration')

    plt.show()

if __name__ == '__main__':
    m = markov_chain(initial=np.array([1, 0]), epoch=20)
    plot_mutation(m)

    m2 = markov_chain(initial=np.array([1, 0]), epoch=40)
    plot_mutation(m2)

    for i in range(1, len(m)):
        if m[i] / m[i - 1] < 1.01:
            index = i
    print(f"after {index} iteration change not significant")

```

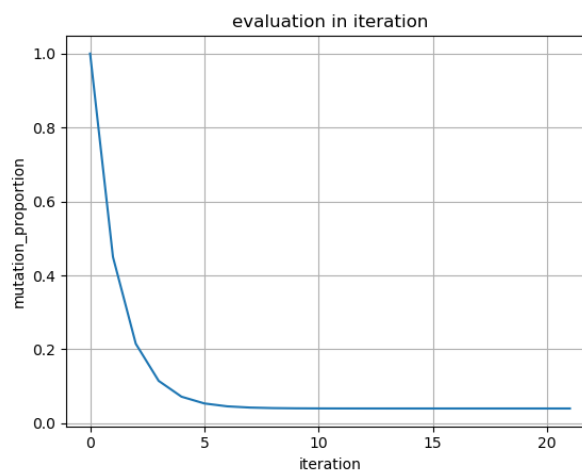


Abbildung 1: Aufgabe2c 2

3)

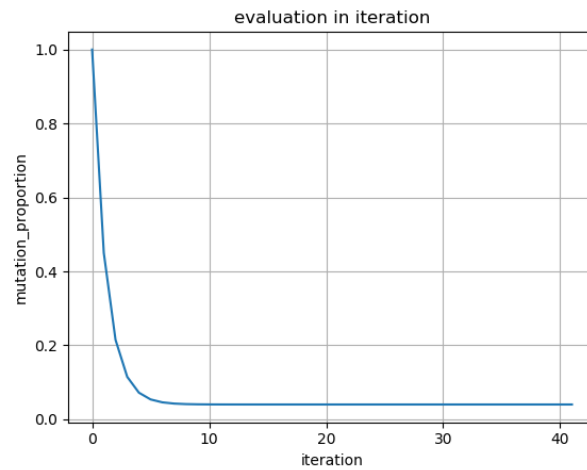


Abbildung 2: Aufgabe2c 3

approximate at 5th step will culture contain stop to change mutation

if

$$45\%m + 2.6\%\tilde{m} = m \quad \text{with } \tilde{m} = (1 - m)$$

$$\Rightarrow m \approx 4.01\%$$

⇒ Evaluation proves get stable probability

**Aufgabe 3: Optimization and Information Theory**

3a)

You work for a telecommunication company that uses a system to transmit four different symbols  $S_1, S_2, S_3, S_4$  over a channel. In the current system, each symbol has a probability to occur according to the following table:

	$S_1$	$S_2$	$S_3$	$S_4$
$p_i$	0.02	0.67	0.23	0.08

How many bits of information can be transmitted on average per symbol under this distribution? Compute the entropy. In general, what is the maximum number of bits per symbols that can be transmitted using a set of four symbols? Which distribution over the symbols is required to achieve this?

answer:

$$h(p_i) = -\log_2 P_i$$

$$h(p_{S_1}) = 5.6439 \quad h(p_{S_2}) = 0.5778 \quad h(p_{S_3}) = 2.1203 \quad h(p_{S_4}) = 3.6439$$

$$\text{average information : } H_{(p)} = E[h_{(p)}] = \sum_{i=1}^n p_i h_{(p_i)} = 1.2792 \text{ bit}$$

$$\text{In general: } \max: \log_2(4) = 2 \text{ bit}$$

Uniform distribution is required to achieve this

3b)

After an upgrade of the system described above, your boss asks you to change the probabilities of transmission in order to maximize the entropy. However, the new system still has the following constraint:

$$6 = \sum_{i=1}^4 2p_i \cdot i$$

- 1) Formulate this as a constrained optimization problem. Do you need to include additional constraints besides the one above?
- 2) Write down the Lagrangian function of the problem. Use one Lagrangian multiplier per constraint.
- 3) Compute the partial derivatives of the Lagrangian above for each multiplier and the objective variable. How challenging is it to solve it analytically?
- 4) Formulate the dual function of this constrained optimization problem. Solve it analytically.
- 5) Name one technique for numerically solving these problems and briefly describe it.

answer:

1)

$$\text{additional condition : } 1 = \sum_{i=1}^4 p_i \quad \text{with } 1 \geq p_i \geq 0 \quad i = 1, 2, 3 \text{ or } 4$$

2)

Cost function : (here Entropie with unit "nat" not unit "bit")

$$\begin{aligned} \max_p \quad & H(p) = - \sum_{i=1}^4 p_i \ln(p_i) \\ \text{s.t.} \quad & t_1(p) = \sum_{i=1}^4 2p_i \cdot i - 6 = 0 \\ & t_2(p) = \sum_{i=1}^4 p_i - 1 = 0 \end{aligned}$$

$$\text{Lagrangian function} \quad L(\underline{p}, \lambda) = - \sum_{i=1}^4 p_i \ln(p_i) + \lambda_1 \left( \sum_{i=1}^4 2p_i \cdot i - 6 \right) + \lambda_2 \left( \sum_{i=1}^4 p_i - 1 \right)$$

3)

$$\begin{aligned} \frac{\partial L}{\partial \lambda_1} &= \sum_{i=1}^4 2p_i \cdot i - 6 \\ \frac{\partial L}{\partial \lambda_2} &= \sum_{i=1}^4 p_i - 1 \Rightarrow \text{monotone increasing} \end{aligned}$$

$\underline{p}$  is correlative with  $\frac{\partial L}{\partial \lambda_1} \Rightarrow$  very hard to analytically solve

4)

Dual Formulation

$$\begin{aligned} \max_p \quad & H(p) = - \sum_{i=1}^4 p_i \ln(p_i) \\ \text{s.t.} \quad & t_1(p) = \sum_{i=1}^4 2p_i \cdot i - 6 = 0 \\ & t_2(p) = \sum_{i=1}^4 p_i - 1 = 0 \end{aligned}$$

$$1) \quad L(\underline{p}, \lambda) = - \sum_{i=1}^4 p_i \ln(p_i) + \lambda_1 \left( \sum_{i=1}^4 2p_i \cdot i - 6 \right) + \lambda_2 \left( \sum_{i=1}^4 p_i - 1 \right)$$

$$\begin{aligned} 2) \quad & \frac{\partial L}{\partial p_i} = -\ln(p_i) - 1 + \lambda_1 \cdot 2i + \lambda_2 = 0 \\ & \Rightarrow \ln(p_i) = -1 + \lambda_1 \cdot 2i + \lambda_2 \\ & p_i^* = e^{(-1 + \lambda_1 \cdot 2i + \lambda_2)} \end{aligned}$$

$$3) \quad \text{replace } p_i^* \text{ in } L(\underline{p}, \lambda)$$

$$\begin{aligned} \Rightarrow G(\lambda) &= - \sum_{i=1}^4 (-1 + \lambda_1 \cdot 2i + \lambda_2) \cdot e^{(-1 + \lambda_1 \cdot 2i + \lambda_2)} \\ &\quad + \left( \sum_{i=1}^4 2i\lambda_1 \cdot e^{(-1 + \lambda_1 \cdot 2i + \lambda_2)} - 6\lambda_1 \right) + \left( \sum_{i=1}^4 \lambda_2 \cdot e^{(-1 + \lambda_1 \cdot 2i + \lambda_2)} - \lambda_2 \right) \\ &= \sum_{i=1}^4 e^{(-1 + \lambda_1 \cdot 2i + \lambda_2)} - 6\lambda_1 - \lambda_2 \end{aligned}$$

assume that  $G(x)$  is a convex function

$$\begin{cases} \frac{\partial G}{\partial \lambda_1} = \sum_{i=1}^4 2i \cdot e^{(-1+\lambda_1 \cdot 2i+\lambda_2)} - 6 = 0 \\ \frac{\partial G}{\partial \lambda_2} = \sum_{i=1}^4 e^{(-1+\lambda_1 \cdot 2i+\lambda_2)} - 1 = 0 \end{cases}$$

$$\Rightarrow \lambda_1 = \text{solve}_1 \quad \lambda_2 = \text{solve}_2$$

```
|
solve_1 =

log(6*exp(1)*root(32*z^3*exp(3) + 32*z^2*exp(2) + 10*z*exp(1) - 1, z, 1) + 8*exp(2)*root(32*z^3*exp(3) + 32*z^2*exp(2) + 10*z*exp(1) - 1, z, 1)^2 + 1)/2

solve_2 =

log(root(32*z^3*exp(3) + 32*z^2*exp(2) + 10*z*exp(1) - 1, z, 1))

fx >>
```

### Abbildung 3: Aufgabe3b 4

5)

Gradient descent is based on the observation that if the multi-variable function  $F(\mathbf{x})$  is defined and differentiable in a neighborhood of a point  $\mathbf{a}$ , then  $F(\mathbf{x})$  decreases fastest if one goes from  $\mathbf{a}$  in the direction of the negative gradient of  $F$  at  $\mathbf{a}$ ,  $-\nabla F(\mathbf{a})$ . It follows that, if

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

for  $\gamma \in \mathbb{R}_+$  small enough, then  $F(\mathbf{a}_n) \geq F(\mathbf{a}_{n+1})$ . In other words, the term  $\gamma \nabla F(\mathbf{a})$  is subtracted from  $\mathbf{a}$  because we want to move against the gradient, toward the local minimum. With this observation in mind, one starts with a guess  $\mathbf{x}_0$  for a local minimum of  $F$ , and considers the sequence  $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots$  such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \quad n \geq 0.$$

We have a monotonic sequence

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \dots,$$

so hopefully the sequence  $(\mathbf{x}_n)$  converges to the desired local minimum. Note that the value of the step size  $\gamma$  is allowed to change at every iteration. With certain assumptions on the function  $F$  (for example,  $F$  convex and  $\nabla F$  Lipschitz) and particular choices of  $\gamma$  (e.g., chosen either via a line search that satisfies the Wolfe conditions, or the Barzilai–Borwein method shown as following),

$$\gamma_n = \frac{|(\mathbf{x}_n - \mathbf{x}_{n-1})^T [\nabla F(\mathbf{x}_n) - \nabla F(\mathbf{x}_{n-1})]|}{\|\nabla F(\mathbf{x}_n) - \nabla F(\mathbf{x}_{n-1})\|^2}$$

convergence to a local minimum can be guaranteed. When the function  $F$  is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

This process is illustrated in the adjacent picture. Here  $F$  is assumed to be defined on the plane, and that its graph has a bowl shape. The blue curves are the contour lines, that is, the regions on which the value of  $F$  is constant. A red arrow originating at a point shows the direction of the negative gradient at that point. Note that the (negative) gradient at a point is orthogonal to the contour line going through that point. We see that gradient descent leads us to the bottom of

the bowl, that is, to the point where the value of the function  $F$  is minimal.

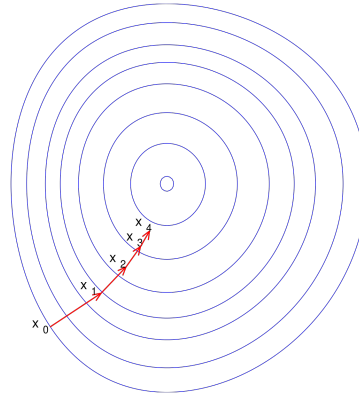


Abbildung 4: Gradient descent

3c)

Rosenbrock's function is a benchmark function for optimization and defined as:

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

To solve the corresponding optimization problem, we would like to find

$$\arg_x \min f(x)$$

In Python, write a simple gradient descent algorithm and run it for 10,000 steps on Rosenbrock's function with  $n = 20$ . To do so, determine the derivative of the function and write a function which evaluates it. Attach a snippet of your gradient descent algorithm, discuss the effects of the learning rate and attach a plot of your learning curve with your best learning rate.

answer:

```
# -*- coding: utf-8 -*-
```

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def benchmark_function(array_x=np.random.random((20, 1))):
    """
    evaluate Rosenbrock's function under array_x
    :param array_x: [array], multidimensional variable (default value: np.random.random((20,
    1)))
    :return:
    cost [float], the value of Rosenbrock's function
    """
    # initial list of intermediate variables
    value_list = []

    # compute
    for i in range(array_x.shape[0]-1):
```



```

    f_x = 100*(array_x[i+1] - array_x[i]**2)**2 + (array_x[i] - 1)**2
    value_list.append(f_x)

    return sum(value_list)

def gradient_descent(array_x):
    """
    compute gradient in Rosenbrock's function
    :param array_x: [array], input array
    :return:
    [array], the gradient matrix, size(n x 1)
    """
    # initial list of intermediate variables
    row_list = []

    # compute
    for i in range(array_x.shape[0]):
        # initial for each for
        row = []
        for j in range(array_x.shape[0] - 1):
            if i == j:
                d_fx = 200*(array_x[i+1] - array_x[i]**2)*(-2*array_x[i]) + 2*array_x[i]
                row.append(float(d_fx))
            elif i == j+1:
                d_fx = 200*(array_x[i] - array_x[i-1]**2)
                row.append(float(d_fx))
            else:
                row.append(0)
        row_list.append(row)

    # convert list to array, size(n-1 x n)
    gradient_matrix = np.array(row_list)

    # merge matrix in vector
    gradient_vector = np.sum(gradient_matrix, axis=1,
                             dtype='float64').reshape(array_x.shape[0], 1)

    return gradient_vector

def var_updating(array_x, alpha=1e-3, epoch=500):
    """
    update variable with gradient descent in iteration
    :param array_x: [array], initial variable, size(n x 1)
    :param alpha: [float], learning rate (default value: 1e-3)
    :param epoch: [int] , time of iteration
    :return:
    """
    # initial cost record
    cost_hist = [benchmark_function(array_x)]

    # initial iteration record
    iteration = 0

    for i in range(epoch):
        # compute gradient
        gradient_vector = gradient_descent(array_x)

        # update array

```

```

    array_x = array_x - alpha * gradient_vector

    # compute cost
    cost = benchmark_function(array_x)

    cost_hist.append(cost)

    # assign necessary iteration
    if i > 10 and iteration == 0 and \
        (cost_hist[i-1]-cost_hist[i])/cost_hist[i] <= 1e-80:
        iteration = i
        continue

    return cost_hist, iteration

def plot_cost(cost_hist, alpha=1e-3, iteration=500):
    """
    plot evaluation history of bacterium with mutation
    :param cost_hist: , the record of evaluation history
    :param alpha: [float], learning rate (default value: 1e-3)
    :return:
    """
    # assign plot data in dict form
    data = {"cost": cost_hist,
            "iteration": np.linspace(0, len(cost_hist), len(cost_hist), dtype=np.int16)}

    # initial figure
    plt.plot("iteration", "cost", data=data, label=f'alpha = {alpha}, iteration = {iteration}')
    plt.grid()
    plt.xlabel("iteration")
    plt.ylabel("Rosenbrock's function")
    plt.title("learning curve")

    return plt

if __name__ == '__main__':
    # set random seed
    np.random.seed(233)

    # set search space
    candidate_alpha = np.logspace(-10, -1, 10)

    plt.figure(1)

    for alpha in candidate_alpha:
        array_x = np.random.random((20, 1))
        hist, iteration = var_updating(array_x)
        plt = plot_cost(hist, alpha=alpha, iteration=iteration)

    plt.legend()
    plt.grid()
    plt.show()

```

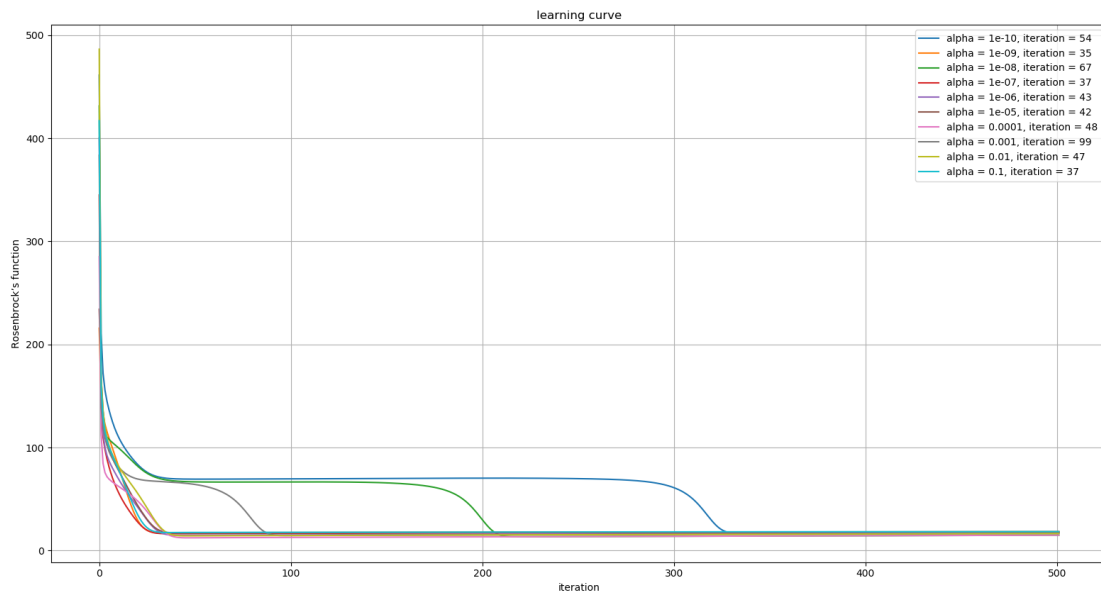


Abbildung 5: Aufgabe3c

3d)

Throughout this class we will see that gradient descent is one of the most used optimization techniques in machine learning. This task asks you to get a more in-depth understanding of the topic by conducting some research by yourself.

1) There are several variants of gradient descent, namely vanilla (also known as batch), stochastic and mini-batch. Each variant differs in how much data we use to compute the gradient of the objective function. Discuss the differences among them, pointing out pros and cons of each one. 2) Many gradient descent optimization algorithms use the so-called momentum to improve convergence. What is it? Is it always useful?

answer:

1)

Vanilla means standard, usual, or unmodified version of something. Vanilla gradient descent means the basic gradient descent algorithm without any bells or whistles.

There are many variants on gradient descent. In usual gradient descent (also known as batch gradient descent or vanilla gradient descent), the gradient is computed as the average of the gradient of each datapoint.

$$\nabla f = \frac{1}{n} \sum_i \nabla \text{loss}(x_i)$$

pros:

great for convex, or relatively smooth error manifolds

cons:

too computationally expensive for not that much of a gain

In stochastic gradient descent with a batch size of one, we might estimate the gradient as

$$\nabla f \approx \nabla \text{loss}(x^*)$$

, where  $x^*$  is randomly sampled from our entire dataset. It is a variant of normal gradient descent, so it wouldn't be vanilla gradient descent

pros:

computationally a lot faster

cons:

single samples are really noisy

The trade off between vanilla gradient descend and stochastic gradient descend is that is called mini-batch gradient descend. In this method, the objective function is computed over a small batch of samples. The size of batch is much smaller than the size of samples in the training set.

pros:

small enough to avoid some of the poor local minima, but large enough that it doesn't avoid the global minima or better-performing local minima

cons:

not easy to find a suitable batch size

2)

Gradient Descent with Momentum considers the past gradients to smooth out the update. It computes an exponentially weighted average of your gradients, and then use that gradient to update the weights instead. It works faster than the standard gradient descent algorithm.

It is not always useful