Contents

1 Linear Algebra Tools

This chapter introduces inner product to give geometric meaning to vectors and vector spaces, enabling calculations of length, distance, and angles.

Definition (Symmetric Positive Definitive Matrix)**.** *A symmetric matrix $A \in \mathbb{R}^{n\times n}$ that satisfies*

$$\text{for every nonzero vector } x : x^T Ax > 0 \tag{1.1}$$

*is called **positive definite**. If only $\geq$ holds in 1.1, then A is called **positive semidefinite**.*

These properties helps in identifying positive definite matrices without having to check the definition explicitly:

1. The null space of A contains only the null vector;
2. The diagonal elements a_{ii} of A are positive;
3. The eigenvalues of A are real and positive.

1.1 Angles and Orthogonality

The angle ω between vectors x and y is computed as:

$$\cos\omega = \frac{\langle x, y\rangle}{\|x\|_2\|y\|_2}$$

Here, $\langle x, y\rangle$ denotes the inner product between x and y.
This angle indicated the vectors' similarity in orientation.

Definition (Orthogonal vectors)**.** *Two vectors are orthogonal if $\langle x, y\rangle = 0$. If additionally $\|x\| = 1 = \|y\|$, then x and y are orthonormal.*

Definition (Orthogonal matrix)**.** *A square matrix is an orthogonal matrix if and only if its columns are orthonormal so that*

$$AA^T = I = A^T A$$

which implies that

$$A^{-1} = A^T$$

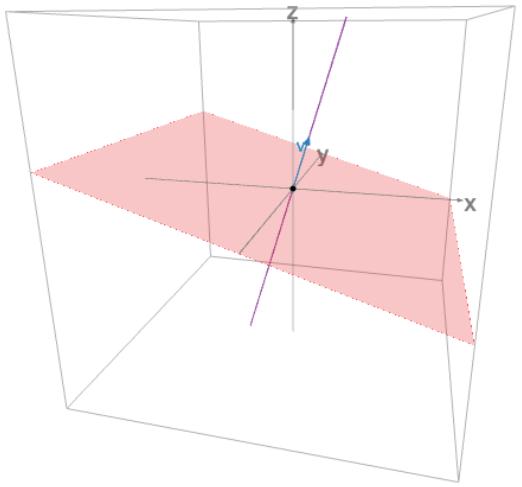
The length of a vector x is not changed when transforming it using an orthogonal matrix A.

$$\|Ax\|_2^2 = \|x\|_2^2$$

Moreover, the angle between any two vectors x, y is also unchanged when transforming both of them using an orthogonal matrix A.

Definition (Orthonormal Basis)**.** *In an n-dimensional vector space V with a basis set $\{b_1, \ldots, b_n\}$, if all the basis vectors are orthogonal to each other, the basis is called as an **orthogonal basis**. Additionally, if the length of each basis vector is 1, the basis is referred to as an **orthonormal basis**.*

We can also have vector spaces that are orthogonal to each other. Given a vector space V of dimension D, let's consider a subspace U of dimension M such that $U \subseteq V$. Then its **orthogonal complement** $U^\perp$ is a $D - M$ dimensional subspace V and contains all vectors in V that are orthogonal to every vector in U.



1.1.1 Orthogonal Projections

Projections are key linear transformations in machine learning and are particularly useful for handling high-dimensional data. Often, only a few dimensions in such data are essential for capturing the most relevant information. By projecting the original high-dimensional data onto a lower dimensional feature space, we can work more efficiently to learn about the dataset and extract significant patterns.

Definition (Projection)**.** *Let V be a vector space and $U \subseteq V$ a subspace of V. A linear mapping $\pi : V \to U$ is called **projection** if it satisfies $\pi^2 = \pi \circ \pi = \pi$.*

Given that linear mappings can be represented by transformation matrices, the above definition extends naturally to *projection matrices* P_π. These matrices exhibit the property that $P_\pi^2 = P_\pi$.
The projection $\pi_U(x)$ of a vector $x \in \mathbb{R}^n$ onto a subspace U is the closest point necessarily in U to x.

2 Matrix Decompositions

2.1 Eigenvalues and Eigenvectors

Eigenanalysis helps us understand linear transformations represented by a matrix A. Eigenvectors x are special vectors that only get scaled, not rotated, when multiplied by A. The scaling factor is the eigenvalue λ, which indicated how much x is stretched or shrunk. λ can also be zero.

Definition (Eigenvalue and Eigenvector)**.** *Let $A \in \mathbb{R}^{n\times n}$ be a square matrix. Then $\lambda \in \mathbb{R}$ is an **eigenvalue** of A and nonzero vector x is the corresponding **eigenvector** of A if*

$$Ax = \lambda x \tag{2.1}$$

*We call 2.1 the **eigenvalue equation**.*

The following statements are equivalent:

- λ is an eigenvalue of $A \in \mathbb{R}^{n\times n}$.
- A nonzero vector x exists such that $Ax = \lambda x$ or, equivalently, $(A - \lambda I_n)x = 0$ for $x \neq 0$.
- Then $A - \lambda I$ is a **singular matrix** and its determinant is **zero**.

Each eigenvector x has one unique eigenvalue λ, but each λ can have multiple eigenvectors.

Definition (Eigenspace and Eigenspectrum)**.** *For $A \in \mathbb{R}^{n\times n}$, the set of all eigenvectors of A associated with an eigenvalue λ spans a subspace of $\mathbb{R}^n$, which is called the **eigenspace** of A with respect to λ and is denoted by E_λ. The set of all eigenvalues of A is called the **eigenspectrum** of A.*

Definition. *Let λ_i be an eigenvalue of a square matrix A. Then the **geometric multiplicity** of λ_i is the number of linearly independent eigenvectors associated with λ_i. In other words, it is the dimensionality of the eigenspace spanned by the eigenvectors associated with λ_i.*

Theorem. *The eigenvectors $x_1, \ldots, x_n$ of a matrix $A \in \mathbb{R}^{n\times n}$ with n distinct eigenvalues $\lambda_1, \ldots, \lambda_n$ are linearly independent.*

This theorem states that eigenvectors of a matrix with n distinct eigenvalues form a basis of $\mathbb{R}^n$.

3 Vector calculus

Firstly, we'll explore partial derivatives and gradients, focusing on functions that take a vector as input and produce a single real number as output. These functions are formally represented as $f : \mathbb{R}^n \to \mathbb{R}$.
Subsequently, we will extend these ideas to functions that not only take a vector as input but also produce a vector as output. These functions can be written as $f : \mathbb{R}^n \to \mathbb{R}^m$.

3.1 Gradients of Real-Valued Functions

When we deal with a function that depends on multiple variables, such as $f(x) = f(x_1, x_2)$, we use the **gradient** to represent its derivative. The gradient is a vector composed of **partial derivates** of the function. To compute each partial derivates, we differentiate the function with respect to one variable while keeping all other variables constant.

$$\nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix} \in \mathbb{R}^{1\times n} \tag{3.1}$$

where n is the number of variables.

Basic Rules of Partial Differentiation

Product rule:
$$\frac{\partial}{\partial x}(f(x)g(x)) = \frac{\partial f}{\partial x}g(x) + f(x)\frac{\partial g}{\partial x}$$

Sum rule:
$$\frac{\partial}{\partial x}(f(x) + g(x)) = \frac{\partial f}{\partial x} + \frac{\partial g}{\partial x}$$

Chain rule:
$$\frac{\partial}{\partial x}(g \circ f)(x) = \frac{\partial}{\partial x}\left(g(f(x))\right) = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x}$$

In the context of the chain rule, consider f as implicitly a composition $f \circ g$.
If a function $f(x_1, x_2)$ is a function of x_1 and x_2, where $x_1(t)$ and $x_2(t)$ are themselves functions of a single variable t, the chain rule yields the partial derivates

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1(t)}{\partial t} \\ \frac{\partial x_2(t)}{\partial t} \end{bmatrix} = \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial t}{\partial x_2}\frac{\partial x_2}{\partial t}$$

Example

Consider $f(x_1, x_2) = x_1^2 + 2x_2$, where $x_1 = \sin t$ and $x_2 = \cos t$, then

$$\text{with} \quad \frac{\partial f}{\partial x_1} = 2x_1, \quad \frac{\partial f}{\partial x_2} = 2$$

$$\begin{aligned} \frac{\mathrm{d}f}{\mathrm{d}t} &= 2\sin t\frac{\partial \sin t}{\partial t} + 2\frac{\partial \cos t}{\partial t} \\ &= 2\sin t\cos t - 2\sin t \end{aligned}$$

If a function $f(x_1, x_2)$ is a function of x_1 and x_2, where $x_1(s, t)$ and $x_2(s, t)$ are themselves functions of two variables s and t, the chain rule yields the partial derivates

$$\frac{\mathrm{d}f}{\mathrm{d}(s,t)} = \begin{bmatrix} \frac{\partial f}{\partial s} & \frac{\partial f}{\partial t} \end{bmatrix}$$

where

$$\begin{aligned} \frac{\partial f}{\partial s} &= \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial s} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial s} \\ \frac{\partial f}{\partial t} &= \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial t} + \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial t} \end{aligned}$$

Another way to obtain these two partial derivatives is to represent the previous formula as a row vector containing the partial derivatives of f with respect to x_1 and x_2. This row vector is then multiplied by a matrix composed of the partial derivatives of x_1 and x_2 with respect to s and t. When you perform this multiplication, you get the exact same result as above.

$$\begin{bmatrix} \frac{\partial f}{\partial s} & \frac{\partial f}{\partial t} \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial s} & \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial s} & \frac{\partial x_2}{\partial t} \end{bmatrix}$$

Example

Given the following functions:
$g : \mathbb{R}^2 \to \mathbb{R}^2 \quad g(s,t) = (\sin(t)s, \cos(s)t)$
$f : \mathbb{R}^2 \to \mathbb{R} \quad f(x_1, x_2) = x_1^2 + 2x_2$
$f \circ g : \mathbb{R}^2 \to \mathbb{R}$
Compute $\nabla_{(s,t)}(f \circ g)$ and evaluate $\nabla_{(s,t)}(f \circ g)(0,0)$.

$$= \begin{bmatrix} 2s\sin(t) & 2 \end{bmatrix} \begin{bmatrix} \sin(t) & s\cos(t) \\ -t\sin(s) & \cos(s) \end{bmatrix}$$
$$= \begin{bmatrix} 2s\sin^2(t) - 2t\sin(s) \\ 2s^2\sin(t)\cos(t) + 2\cos t \end{bmatrix} = (0, 2)$$

3.2 Gradients of Vector-Valued Functions

We can express a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$ as a column vector of m real-valued functions $f_i : \mathbb{R}^n \to \mathbb{R}$. Given an input vector $x = [x_1, \ldots, x_n]^T \in \mathbb{R}^n$, the output is defined as:

$$f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix} \in \mathbb{R}^m$$

Definition (Jacobian)**.** *By contrast, in Equation 3.1, each partial derivative $\frac{\partial f}{\partial x_i}$ is a column vector.*

$$J = \nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix}$$
$$= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$
$$J(i,j) = \frac{\partial f_i}{\partial x_j}$$

*The collection of all first-order partial derivatives of a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$ is called the **Jacobian**. The Jacobian J is an $m \times n$ matrix.*

Example

$$f : \mathbb{R}^2 \to \mathbb{R}^3, \quad f : (x_1, x_2) = \begin{pmatrix} x_1 + x_2 \\ 2x_1^2 - x_2 \\ -x_1 x_2 \end{pmatrix}$$
$$J(f) : \mathbb{R}^2 \to \mathbb{R}^{3\times 2}, \quad J(i,j) = \begin{bmatrix} 1 & 1 \\ 4x_1 & -1 \\ -x_2 & -x_1 \end{bmatrix}, \quad J(1,1) = \begin{bmatrix} 1 & 1 \\ 4 & -1 \\ -1 & -1 \end{bmatrix}$$

Example

Let us consider the linear model

$$y = \Phi\theta$$

where $\theta \in \mathbb{R}^D$ is a parameter vector, $\Phi \in \mathbb{R}^{N\times D}$ are input features, and $y \in \mathbb{R}^N$ are the corresponding observations. We define the functions

$$e : \mathbb{R}^D \to \mathbb{R}^N, \quad e(\theta) = y - \Phi\theta$$
$$L : \mathbb{R}^N \to \mathbb{R}, \quad L(e) = \|e\|_2^2, \quad L(\theta) = \|y - \Phi\theta\|_2^2$$

This is called a **least-squares loss** function.
We want to find $\frac{\partial L}{\partial \theta}$, which is derivative of the loss function with respect to the parameters θ. This will allow us to find the optimal θ that minimizes the loss function $L(\theta)$.
The chain rule allows us to compute the gradient as

$$\frac{\partial L}{\partial e} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial \theta}$$

We know that $\|e\|_2^2 = e^T e$ and so

$$\frac{\partial L}{\partial e} = 2e^T \in \mathbb{R}^{1\times N}$$

Furthermore, we obtain

$$\frac{\partial e}{\partial \theta} = -\Phi \in \mathbb{R}^{N\times D}$$

such that our desired derivative is

$$\nabla L_\theta = -2e^T\Phi = -2\underbrace{(y^T - \theta^T\Phi^T)}_{1\times N}\underbrace{\Phi}_{N\times D} \in \mathbb{R}^{1\times D}$$

3.3 Backpropagation and Automatic Differentiation

In machine learning, finding optimal model parameters often involves performing gradient descent. This requires computing the gradient of a learning objective with respect to the model's parameters. Calculating the gradient explicitly can be impractical due to the complexity and length of the resulting derivative equations. To address this, the **backpropagation** algorithm was introduced in 1962 as an efficient way to compute these gradients, particularly for neural networks.

In neural networks, the output y is computed through a multi-layered function composition $y = (f_K \circ f_{K-1} \circ \cdots f_1)(x)$. Here, x are the inputs (e.g., images), y are the observations (e.g., class labels). Each functions $f_i, i = 1, \ldots, K$, has its own parameters. Specifically, in the i^{th} layer, the function is given $f_i(x_{i-1}) = \sigma(A_{i-1} + b_{i-1})$, where x_{i-1} is the output from layer $i-1$ and σ is an activation function.

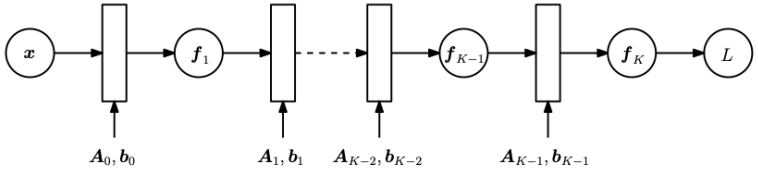


Figure 5.2 Forward pass in a multi-layer neural network to compute the loss L as a function of the inputs $\boldsymbol{x}$ and the parameters $\boldsymbol{A}_i,\ \boldsymbol{b}_i$.

In order to train a neural network, we aim to minimize a loss function L with respect to all parameters A_j, b_j for $j = 0, \ldots, K-1$. Specifically, we're interested in optimizing these parameters to minimize the squared loss given by

$$L(\theta) = \|y - f_K(\theta, x)\|^2$$

where $\theta = \{A_0, b_0, \ldots, A_{K-1}, b_{K-1}\}$.

To minimize $L(\theta)$ we need to compute its gradients of L to the parameter set θ. This involes calculating the partial derivatives of L with respect to the parameters $\theta_j = \{A_j, b_j\}$ for each layer $j = 0, \ldots, K-1$. The chain rule allows us to determine the partial derivatives as

$$\begin{aligned}
\frac{\partial L}{\partial \theta_{K-1}} &= \frac{\partial L}{\partial f_K}\frac{\partial f_K}{\partial \theta_{K-1}} \\
\frac{\partial L}{\partial \theta_{K-2}} &= \frac{\partial L}{\partial f_K}\boxed{\frac{\partial f_K}{\partial f_{K-1}}\frac{\partial f_{K-1}}{\partial \theta_{K-2}}} \\
\frac{\partial L}{\partial \theta_{K-3}} &= \frac{\partial L}{\partial f_K}\frac{\partial f_K}{\partial f_{K-1}}\boxed{\frac{\partial f_{K-1}}{\partial f_{K-2}}\frac{\partial f_{K-2}}{\partial \theta_{K-3}}} \\
\frac{\partial L}{\partial \theta_i} &= \frac{\partial L}{\partial f_K}\frac{\partial f_K}{\partial f_{K-1}}\cdots\boxed{\frac{\partial f_{i+2}}{\partial f_{i+1}}\frac{\partial f_{i+1}}{\partial \theta_i}}
\end{aligned}$$

The red terms are partial derivatives of the output of a layer with respect to its inputs, whereas the blue terms are partial derivatives of the output of a layer with respect to its parameters.

The key insight of backpropagation is to reuse previously computed derivatives to avoid redundant calculations. When we've computed the partial derivatives $\frac{\partial L}{\partial \theta_{i+1}}$, we can reuse them to efficiently calculate the partial derivatives $\frac{\partial L}{\partial \theta_i}$.

It turns out that backpropagation is a special case of a set of techniques known as **automatic differentiation**. Automatic differentiation numerically evaluate the exact (up to machine precision) gradient of a function by working with intermediate variables and applying the chain rule.

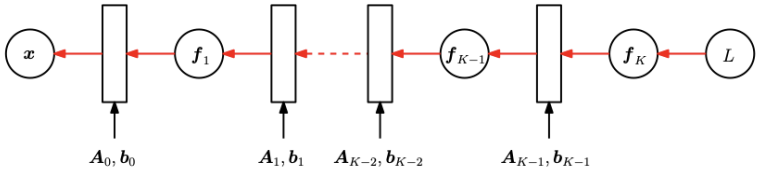


Figure 5.2 Backward pass in a multi-layer neural network to compute the gradients of the loss function.

Example

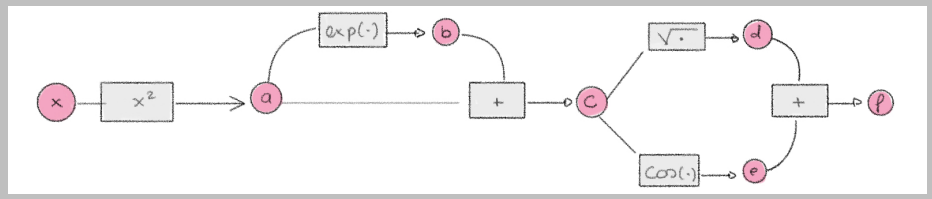
Consider the real-valued function

$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos(x^2 + \exp(x^2))$$

Another way to attach this would be to just define some *intermediate variables*. Say

$$\begin{aligned}
a &= x^2 \\
b &= \exp(a) \\
c &= a + b \\
d &= \sqrt{c} \\
e &= \cos(c) \\
f &= d + e
\end{aligned}$$

The set of equations that include intermediate variables can be thought of as a computational graph



By looking at the computation graph, we can compute $\frac{\partial f}{\partial x}$ by working backward from the end of the graph and obtain the derivative of each variable, making the use of the derivatives of the children of that variable

$$\begin{aligned}
\frac{\partial f}{\partial d} &= \frac{\partial f}{\partial e} = 1 \\
\frac{\partial f}{\partial c} &= \frac{\partial f}{\partial d}\frac{\partial d}{\partial c} + \frac{\partial f}{\partial e}\frac{\partial e}{\partial c} \\
\frac{\partial f}{\partial b} &= \frac{\partial f}{\partial c}\frac{\partial c}{\partial b} \\
\frac{\partial f}{\partial a} &= \frac{\partial f}{\partial b}\frac{\partial b}{\partial a} + \frac{\partial f}{\partial c}\frac{\partial c}{\partial a} \\
\frac{\partial f}{\partial x} &= \frac{\partial f}{\partial a}\frac{\partial a}{\partial x}
\end{aligned}$$

We observe that the computation required for calculating the derivative is of similar complexity as the computation of the function itself (forward pass).

Automatic differentiation is a formalization of last Example. Let $x_1, \ldots, x_d$ be the input variables to the function, $x_{d+1}, \ldots, x_{D-1}$ be the intermediate variables, and x_D the output variable. Then the computation graph can be expressed as follows:

$$\text{For } i = d+1, \ldots, D: \quad x_i = g_i(x_{Pa}(x_i)) \tag{3.2}$$

where the $g_i(\cdot)$ are elementary functions and $x_{Pa}(x_i)$ are the parent nodes of the variable x_i in the graph.
Recall that by definition $f = x_D$ and hence

$$\frac{\partial f}{\partial x_D} = 1$$

For other variables x_i, we apply the chain rule

$$\frac{\partial f}{\partial x_i} = \sum_{x_j : x_i \in Pa(x_j)} \frac{\partial f}{\partial x_j}\frac{\partial x_j}{\partial x_i} = \sum_{x_j : x_i \in Pa(x_j)} \frac{\partial f}{\partial x_j}\frac{\partial g_j}{\partial x_i} \tag{3.3}$$

where $Pa(x_j)$ is the set of parent nodes of x_j in the computation graph. Equation 3.2 is the forward pass, whereas 3.3 is the backward pass.
The automatic differentiation approach works whenever we have a function that can be expressed as a computation graph, where the elementary functions are differentiable.

4 Continuous Optimization