Timothy Chung

COMP70015
Mathematics for Machine Learning

AUTUMN 2024

Contents

Part I
Fundamental Concepts in Mathematics for Machine Learning

0
Preliminaries

0.1 Course Resources

Course Resources **Reference 0.1.1**

- The textbook, *Mathematics for Machine Learning*, to be referenced as Deisenroth et al. (2020), can be found here
- The textbook, *Machine Learning a Probabilistic Perspective*, to be referenced as Murphy (2012), can be found here
- The textbook, *Pattern Recognition and Machine Learning*, to be referenced as Bishop (2006), can be found here

Additional Resources **Reference 0.1.2**

- **Linear Algebra:** Chapter 2 (page 29) *Bengio et al. (2017)* link
- **Probability Theory Review:** Chapter 2 (page 27) *Murphy (2012). Machine learning: a probabilistic perspective.* MIT Press.
- **Calculus Review:** Chapter 5 (page 139) *Deisenroth et al. (2020). Mathematics for machine learning.* Cambridge University Press.

0.2 Course Notation

Real values are denoted with lower-case Roman letters, e.g. x, y, z. As convention, we will use a bold Roman letter or a Greek symbol to denote a vector, e.g. $\boldsymbol{x}$, $\boldsymbol{\theta}$. Do not confuse this with random variables, which are denoted using bold, non-italic Roman letters, e.g. $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$. Matrices are represented by upper-case Roman letters such as X or random matrices like $\mathbf{X}$. Greek letters will always have their type specified, except for $\boldsymbol{\theta}$, which will be used without loss of generality to represent model parameters.

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

0.3 Linear Algebra

A core prerequisite for this class is linear algebra. Please use this section and references herein to brush up on necessary concepts.

0.3.1 Vectors

A vector is defined by a list of numbers. It is most useful to geometrically interpret vectors as points in space.

Notation Type	Example
Real Values	x, y, z
Vectors	$\boldsymbol{x}$, $\boldsymbol{\theta}$ (where $\boldsymbol{\theta}$ represents parameters)
Random Variables	$\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ Example: $\mathbf{z} \sim \text{Unif}(0, 1)$
Matrices	$X = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix}$
Random Matrices	$\mathbf{X}$ Example: $\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Table 1: Summary of Course Notation with Examples

$$\boldsymbol{x} := \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{1}$$

0.3.2 Vectors Operations

We have several vector operations:

- **Scalar Multiplication**:

$$\alpha \boldsymbol{x} = \begin{bmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_n \end{bmatrix} \tag{2}$$

- **Vector Addition**:

$$\boldsymbol{x} + \boldsymbol{y} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{bmatrix} \tag{3}$$

- **Dot Product**:

$$\boldsymbol{x}^\top \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i \tag{4}$$

Symbol	Meaning
$\mathbf{v}^\top$	Transpose of $\mathbf{v}$
$\mathbf{u}^\top \mathbf{v}$	Inner (scalar) product
$\mathbf{u}\mathbf{v}^\top$	Outer product (n × n matrix)
$\mathbf{u} \odot \mathbf{v}$	Hadamard/Elementwise product, $[u_1 v_1, \ldots, u_n v_n]$
$\dim(\mathbf{v})$	Dimensionality of $\mathbf{v}$ (namely n)
$\text{diag}(\mathbf{v})$	Diagonal $n \times n$ matrix made from vector $\mathbf{v}$
1 or $\mathbf{1}_n$	Vector of ones (of length n)
0 or $\mathbf{0}_n$	Vector of zeros (of length n)
$\|\mathbf{v}\|$ or $\|\mathbf{v}\|_2$	Euclidean or ℓ_2 norm, $\sqrt{\sum_{i=1}^{n} v_i^2}$
$\|\mathbf{v}\|_1$	ℓ_1 norm, $\sum_{i=1}^{n} \lvert v_i \rvert$
$\mathbf{e}_i$	ith-basis vector (1 in dimension i and 0 elsewhere)

Table 2: Standard Notation for Vectors

0.3.3 Matrices

Matrices define linear transforms. Geometrically, it is best to interpret them as transformations whose columns are the new basis vectors.

$$A := \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{bmatrix} \quad A \in \mathbb{R}^{m \times n} \tag{5}$$

Non-square matrices can be thought of as transformations from $\mathbb{R}^m$ dimentional space to $\mathbb{R}^n$ dimensional space.

0.3.4 Matrix Operations

We define several matrix operations as follows:

- **Scalar Multiplication**:
$$cA = cA_{i,j} \tag{6}$$
- **Hadamard Product (Element-wise Multiplication)**:
$$A \circ B = A_{i,j} B_{i,j} \tag{7}$$
- **Matrix Subtraction**:
$$A - B = A_{i,j} - B_{i,j} \tag{8}$$
- **Matrix Product**:
$$C_{i,j} = \sum_k A_{i,k} B_{k,j} \tag{9}$$
- **Trace of a Matrix**:
$$\mathrm{Tr}(A) = \sum_i A_{i,i} \tag{10}$$
- **Kronecker Product**:
$$A \otimes B = \begin{pmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{pmatrix} \tag{11}$$
  The Kronecker product $A \otimes B$ results in a block matrix, where each element a_{ij} of matrix A is multiplied by the entire matrix B.
- **Positive Definite Matrix** ($S \succ 0$):
$$\mathbf{x}^\top S \mathbf{x} > 0 \quad \forall \mathbf{x} \in \mathbb{R}^n, \mathbf{x} \neq \mathbf{0} \tag{12}$$
  A matrix S is positive definite if for all non-zero vectors $\mathbf{x}$, the quadratic form $\mathbf{x}^\top S \mathbf{x}$ is strictly greater than 0. Positive definite matrices are often used to indicate that a matrix represents a convex function.

Dimensional Notation:

$$A \in \mathbb{R}^{n \times m} : \mathbb{R}^n \to \mathbb{R}^m \tag{13}$$
$$B \in \mathbb{R}^{m \times k} : \mathbb{R}^m \to \mathbb{R}^k \tag{14}$$
$$AB \in \mathbb{R}^{n \times k} : \mathbb{R}^n \to \mathbb{R}^k \tag{15}$$

Note 0.3.1 Matrix Multiplication as Dot Products of the Row and Column

This matrix multiplication is equivalent to the dot product between row i of matrix A and column j of matrix B:

$$\mathbf{x}^\top \mathbf{y} = \sum_{i=1}^{n} x_i y_i \tag{16}$$

Symbol	Meaning
$\mathbf{X}_{:,j}$	j-th column of matrix
$\mathbf{X}_{i,:}$	i-th row of matrix (treated as a column vector)
$\mathbf{X}_{ij}$	Element (i, j) of matrix
$\mathbf{X}^\top$	Transpose of a matrix
$\text{diag}(\mathbf{S})$	Diagonal vector extracted from square matrix
$\mathbf{I}$ or $\mathbf{I}_n$	Identity matrix of size $n \times n$
$\mathbf{X} \odot \mathbf{Y}$	Hadamard/Elementwise product
$\mathbf{X} \otimes \mathbf{Z}$	Kronecker product
$\mathbf{S} \succ 0$	True iff $\mathbf{S}$ is a positive definite matrix
$\text{tr}(\mathbf{S})$	Trace of a square matrix
$\det(\mathbf{S})$	Determinant of a square matrix
$\lvert\mathbf{S}\rvert$	Determinant of a square matrix

Table 3: Standard Notation for Matrices

0.4 Calculus

In this section, we cover basic multivariate calculus and key identities. These are considered prerequisite material, so students should spend time reviewing them if needed. For a more in-depth refresher, refer to Chapter 5 of Deisenroth et al. (2020).

Unqualified, a **function** is a mapping from the reals to the reals: $f : \mathbb{R} \to \mathbb{R}$. The mapping of a value x from the domain is denoted $f(x)$. The derivative is defined as:

$$f'(x) = \frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

assuming the limit exists. This is also known as the forward finite difference quotient.

Leibniz and **Lagrange** notations are often used for derivatives, as shown below:

Notation Type	Representation
Leibniz Notation	$\frac{dy}{dx}$, $\frac{d}{dx}f$
Lagrange Notation	$f'(x)$, $f'(a)$ (for derivative at a)

Table 4: Comparison of Leibniz and Lagrange Notations

The notation $\frac{dy}{dx}\big|_a$ is used for the derivative of f at a, while $f'(a)$ is used in Lagrange notation.

0.5 Multivariate Calculus

For vector-valued functions $f : \mathbb{R}^n \to \mathbb{R}$, we define the partial derivative:

$$\frac{df}{dx_i} = \lim_{h \to 0} \frac{f(x + he_i) - f(x)}{h}$$

where e_i is the i-th basis vector. This extends the definition of the derivative to each variable.

The gradient $\mathbf{g}$, or ∇f, is defined as the collection of all partial derivatives into a row vector:

$$\mathbf{g} = \nabla f = \left[\frac{df}{dx_0}, \frac{df}{dx_1}, \ldots, \frac{df}{dx_n}\right]$$

The gradient maps vectors to vectors: $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n$, which we call a **vector field**. The derivative of a vector field $f : \mathbb{R}^n \to \mathbb{R}^m$ is a matrix called the **Jacobian**, given by:

$$\nabla f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

where f_i represents the i-th output of the function f. This matrix captures all partial derivatives and forms the foundation for calculating Jacobians of vector-valued functions.

0.6 Probability and Statistics

x is a sample from a random variable X.
$P(X = x)$ refers to probability that the random variable X takes on the value x.

0.6.1 Probability Density Function (PDF)

The Probability Density Function (PDF) of a continuous random variable X is a function $f_X(x)$ that describes the relative likelihood for this random variable to take on a given value. The PDF has the following properties:

- $f_X(x) \geq 0$ for all x.
- $\int_{-\infty}^{\infty} f_X(x)\, dx = 1$.

Mathematically, the PDF is defined such that the probability that X lies within a particular interval $[a, b]$ is given by:

$$P(a \leq X \leq b) = \int_a^b f_X(x)\, dx.$$

The most common example of a PDF is the normal distribution, which is characterised by two parameters: the mean μ and the variance σ^2. The PDF of a normal distribution is given by:

$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

Example PDF: Normal Distribution

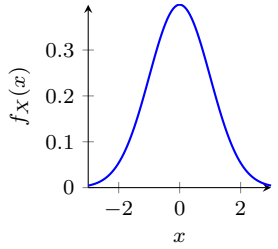


Figure 1: Probability Density Function of a standard normal distribution

0.6.2 Cumulative Distribution Function (CDF)

The Cumulative Distribution Function (CDF) of a continuous random variable X is a function $F_X(x)$ that describes the probability that X will take a value less than or equal to x. It is defined as:

$$F_X(x) = P(X \leq x) = \int_{-\infty}^{x} f_X(t)\, dt.$$

The CDF has the following properties:

- $0 \leq F_X(x) \leq 1$ for all x.
- $F_X(x)$ is a non-decreasing function.
- $\lim_{x \to -\infty} F_X(x) = 0$.
- $\lim_{x \to \infty} F_X(x) = 1$.

Example CDF: Sigmoid Approximation

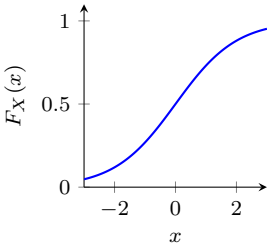


Figure 2: Cumulative Distribution Function

0.7 Moments

Moments help describe probability distributions. The **first moment** is the mean (or centre of mass), the **second moment** is the variance (or spread), and the **third moment** is the skewness. Higher moments are also useful but are less common in practice.
The **expectation** of a distribution, or its first moment, is given by:

$$\mu = \mathbb{E}[x] = \int_{-\infty}^{\infty} p(x = x)dx$$

where $p(x)$ is the probability density or mass function.
The general form of the n-th moment is:

$$m_n = \frac{\mathbb{E}[(x-\mu)^n]}{\mathbb{E}[(x-\mu)^2]^{n/2}}$$

Moment	Name	Formula	Integration Formula	Description
1st	Mean	$\mu = \mathbb{E}[x]$	$\mu = \int_{-\infty}^{\infty} xp(x)dx$	Centre of mass, the average value of the random variable.
2nd	Variance	$\sigma^2 = \mathbb{E}[(x-\mu)^2]$	$\sigma^2 = \int_{-\infty}^{\infty}(x-\mu)^2p(x)dx$	Spread of the distribution around the mean.
3rd	Skewness	$\frac{\mathbb{E}[(x-\mu)^3]}{\sigma^3}$	$\int_{-\infty}^{\infty}\frac{(x-\mu)^3}{\sigma^3}p(x)dx$	Asymmetry or tilt of the distribution.
4th	Kurtosis	$\frac{\mathbb{E}[(x-\mu)^4]}{\sigma^4}$	$\int_{-\infty}^{\infty}\frac{(x-\mu)^4}{\sigma^4}p(x)dx$	Measure of the tail thickness or sharpness of the peak.

0.8 Independent and Identically Distributed (i.i.d.) Random Variables

0.8.1 Independence of Random Variables

Independence refers to the idea that the values or outcomes of one observation in a dataset do not depend on or influence the values of any other observation.
For the set of random variables $X_1, X_2, \ldots X_n$, for the collection

$$\{x_1 \sim X_1, x_2 \sim X_2, \ldots, x_N \sim X_N\}$$

we often assume independence, for any subset of observations $\{X_{i1}, X_{i2}, \ldots, X_{ik}\}$ where $i_1, i_2, \ldots, i_k \in \{1, 2, \ldots, N\}$, the joint distribution factorises:

$$P(X_{i_1} = x_{i_1}, \ldots, X_{i_k} = x_{i_k}) = P(X_{i_1} = x_{i_1}) \cdot \ldots \cdot P(X_{i_k} = x_{i_k}) \tag{17}$$

The joint distribution of the subset is the product of the marginal distributions of the individual random variables.

Independence — **Definition 0.8.1**

The occurrence of any event does not affect the occurrence of others. This is a key assumption in many ML models that we will see the usefulness of in later chapters.

Why do we model samples as distinct random variables?

1. By treating each sample as a distinct variable, we assume samples are i.i.d, allowing every sample to contribute independently to the likelihood of oberving the data given the model prarameters – so every sample provides unique information to estimate the parameters of the underlying distribution. If we treated all samples as a single random variable, we would lose the granularity of information, leading to a poorer estimate.
2. The assumption of i.i.d follows many results in probability and statistics, such as the Central Limit Theorem, which states that the sum of a large number of i.i.d. random variables is approximately normally distributed. It is also an assumed requirement for a model to generalise. The assumption simplifies the analysis and derivation process, allowing us to use techniques like maximum likelihood estimation (MLE) and empirical risk minimization (ERM).

0.8.2 Identically Distributed Random Variables

The "identically distributed" part of i.i.d. refers to the idea that all random variables in the sample follow the same probability distribution. That is, they share the same probability density function (pdf) or probability mass function (pmf), depending on whether the data is continuous or discrete. If the set of random variables $\{X_1, X_2, \ldots, X_n\}$ are identically distributed, then:

$$f_{X_1}(x) = f_{X_2}(x) = \ldots = f_{X_n}(x) = f_X(x) \tag{18}$$

$$F_{X_1}(x) = F_{X_2}(x) = \ldots = F_{X_n}(x) = F_X(x) \tag{19}$$

This implies that the cumulative distribution function (CDF) is the same for all these random variables.

Identically Distributed **Definition 0.8.2**

The random variables in a sample are drawn from the same probability distribution.

0.9 Einstein/Pythonic Index Notation

Most readers will have covered a lot of the linear algebra basics before, if so, this is the most important section to read!

Refer to this video on Einstein summation convention for more information.

As reasoning about matrix and vector products can sometimes be cumbersome, it is often useful to write out the operations we perform in index notation. The matrix product $C = AB$ can be written as: $C_{i,j} = \sum_k A_{i,k} B_{k,j}$. It can also be useful to adopt a more "pythonic" index notation where we consider the system $Ax = b$ and write the first entry of the vector b as:

$$A_{1,:}x = b_1 \tag{20}$$

which indicates that the first value of the result is simply the dot product of the first row of matrix A with the vector x.
Then, we can write the entire system as:

$$C_{i,j} = \sum A_{i,:} B_{:,j} \tag{21}$$

Generally in tensor calculus, a lot of expressions involve summing over particular indices.

$$\sum_{i=1}^{3} a_i x_i = a_1 x_1 + a_2 x_2 + a_3 x_3 \tag{22}$$

Fun fact

Yes, this was introduced by Albert Einstein in 1916!

Non-Examinable **0.9.1**

Einstein Notation consists of two different kinds of indices: the Dummy Index and the Free Index. In **Einstein notation**, we can write this as:

$$a_i x_i \quad (i = 1, 2, 3) \tag{23}$$

This brings us to our first rule (and several others):

Rule 1: Any twice-repeated index in a single term is implicitly summed over. Typically, this is from index 1 to 3 because most calculations are done in 3D space. For example, if we had:

$$a_{ij} b_j = a_{i1} b_1 + a_{i2} b_2 + a_{i3} b_3 \tag{24}$$

We can then express this in the Einstein notation as:

$$a_{ij}b_j = a_{i\alpha}b_\alpha \quad \alpha \in \{1, 2, 3\} \tag{25}$$

This will begin to make more sense as we come up with a better way to describe indices that appear once, and twice-repeated indices.

Rule 2: The definitions of indices:

- We let j be the dummy index, because it is repeated only twice. One can thus replace j with any other index or letter, it is just a placeholder (thus called a dummy index). Although more rigorously:
  - One can replace any dummy index with a letter/index that is not already used in the expression.
  - This letter must be over the same range as the original dummy index, so in the case of replacing j, it must be over the range 1 to 3.

$$a_{ij}b_j = a_{i1}b_1 + a_{i2}b_2 + a_{i3}b_3 \tag{26}$$
$$= a_{i\alpha}b_\alpha \quad \alpha \in \{1, 2, 3\} \tag{27}$$

- i is the free index, which can take on any value that j takes on, but it is not summed over and can only take on one value at a time $i \in \{1, 2, 3\}$.
- The free index occurs only **once** in the expression and **cannot be replaced by another free index,**

$$a_{ij}b_j \neq a_{kj}b_j \tag{28}$$

- To help avoid confusion, one tip is to use roman letters (i, j, k) for free indices, and greek letters (λ, μ, ρ) for dummy indices.

Rule 3: No index may occur 3 or more times in a given term.

- $a_{ij}b_{ij}$ ✓
- $a_{ii}b_{ij}$ ×
- $a_{ij}b_{ij}$ ×
- $a_{ij}b_j + a_{ji}b_j$ ✓

In the last example, we are adding **multiple terms**, so the index occurence rule only applies by term. So j is a dummy index for both terms since it occurs twice (per term).

Rule 4: In an equation involving Einstein notation, the free indices on the left-hand side must match the right-hand side.

- $x_i = a_{ij}b_j$ ✓
  - i is a free index on both the LHS and RHS.
- $a_i = A_{xi}B_{xk}x_j + C_{ik}u_k$ ✓
  - i is the free index on both the LHS and RHS.
- $x_i = A_{ij}$ ×
  - i is a free index on both the LHS and RHS, but j is a free index on the RHS that is not on the LHS.
- $x_j = A_{ik}u_k$ ×
  - LHS free index: j.

  - RHS free index: i.
- $x_i = A_{ik}u_k + c_j \quad \times$
  - LHS free index: i.
  - RHS free indices: i and j.

Relating Einstein Notation to Pythonic Notation: We had:

$$C_{i,j} = \sum_k A_{i,k}B_{k,j} \tag{29}$$

Since k is our dummy variable (it is summed over and doesn't appear in the final result), we can replace it with colon (:) to indicate we are working with all elements along that dimension. So we have

$$C_{i,j} = A_{i,:}B_{:,j} \tag{30}$$

In Python, this is:

```
C[i, j] = np.dot(A[i, :], B[:, j])
```

1
Formalising ML Problem Settings

1.1 Machine Learning from the Mathematical Perspective

What is machine learning?

Professor

Machine learning is the study of algorithms which automatically perform a task by processing an example dataset.

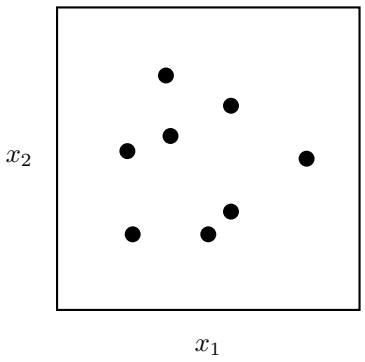
ChatGPT

Machine learning is a computational approach that allows systems to automatically learn and improve from experience without being explicitly programmed.

The professor himself much prefers his definition, which focuses on the word **algorithms**, **task**, and **dataset**.
The issue with the GPT definition is that the word 'computational' is too broad and vague, and the notion of 'learn' and 'improving from experience' is too colloqiually used in the actual nature of machine learning.

1.2 Datasets

Data is the keystone of machine learning, used to infer and learn good models. We denote the input or feature space as $x \in \mathbb{R}^n$ and outputs as $y \in \mathbb{R}^m$.
There are several ways we can look at data:



1.2.1 Set Perspective

- Data is viewed as a collection of N 2-tuples: $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, where each input-output pair is part of the dataset.
- This perspective is order/permutation invariant, focusing on the set of samples rather than the order in which they appear.
- The size of the set measures the sample complexity.

- Sets are often useful for reasoning about feature/label spaces in an abstract way.
- Example of dataset representation:

$$\{(\boldsymbol{x}^{(i)})_{i=1}^{N}\}$$

This represents a dataset consisting of a collection of N input vectors $\boldsymbol{x}^{(i)}$. Each $\boldsymbol{x}^{(i)}$ is a feature vector or a sample, without any corresponding labels or outputs. This kind of dataset is typically used in unsupervised learning, where we only have inputs and are not concerned with outputs (e.g., clustering or dimensionality reduction).

$$\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})_{i=1}^{N}\}$$

This represents a set of N input-output pairs, where $\boldsymbol{x}^{(i)}$ is the input feature vector and $\boldsymbol{y}^{(i)}$ is the corresponding label or output. This is the standard representation for datasets in supervised learning, where the goal is to learn a function that maps inputs to outputs (e.g., regression or classification tasks).

$$\{(\boldsymbol{s}^{(i)}, \boldsymbol{a}^{(i)}, \boldsymbol{s}^{(i+1)})_{i=1}^{N}\}$$

This set represents a sequence of N tuples, where $\boldsymbol{s}^{(i)}$ is the state at time step i, $\boldsymbol{a}^{(i)}$ is the action taken in that state, and $\boldsymbol{s}^{(i+1)}$ is the resulting next state. This kind of dataset is commonly used in reinforcement learning (RL), where the goal is to learn a policy that dictates which actions to take in various states to maximize a reward signal over time.

1.2.2 Probability Perspective

- Data is viewed as samples where each sample is a 2-tuple $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ drawn from a joint distribution $P(\mathbf{X}, \mathbf{Y})$, where the relationship between input $\mathbf{X}$ and output $\mathbf{Y}$ is of primary interest.
- We are interested in the probability of an output given we have observed a particular input: $P(\mathbf{Y} = \boldsymbol{y} | \mathbf{X} = \boldsymbol{y})$.
- This perspective is fundamental in probabilistic machine learning and provides tools for modelling uncertainty and reasoning about observations.
- Common assumptions like i.i.d. (independent and identically distributed) are often applied:

$$P(x_1, x_2, \ldots, x_k) = P(x_1) \cdot P(x_2) \ldots P(x_k)$$

- Example of dataset representation with probabilistic modeling:

$$\boldsymbol{x} \sim P(\mathbf{X})$$

This represents an input vector $\boldsymbol{x}$ sampled from a probability distribution $P(\mathbf{X})$. In this case, $\mathbf{X}$ refers to the input space, and each input $\boldsymbol{x}$ is drawn according to its distribution.

$$(\boldsymbol{x}, \boldsymbol{y}) \sim P(\mathbf{X}, \mathbf{Y})$$

This represents an input-output pair $(\boldsymbol{x}, \boldsymbol{y})$ sampled from the joint distribution $P(\mathbf{X}, \mathbf{Y})$, where $\mathbf{X}$ refers to the input space, and $\mathbf{Y}$ refers to the output space. The goal here is to model the relationship between inputs $\boldsymbol{x}$ and corresponding outputs $\boldsymbol{y}$, as is typical in supervised learning.

$$\boldsymbol{s}^{(i+1)} \sim P(\boldsymbol{s}^{(i+1)} \mid \boldsymbol{s}^{(i)}, \boldsymbol{a}^{(i)})$$

This represents the next state $\boldsymbol{s}^{(i+1)}$ sampled from the conditional probability distribution given the current state $\boldsymbol{s}^{(i)}$ and action $\boldsymbol{a}^{(i)}$. This kind of representation is typical in reinforcement learning (RL), where we model transitions between states based on actions taken by the agent.

1.2.3 Linear Algebra/Statistics Perspective

- A bit misleading to call it a 'linear algebra perspective' since it's used here in almost al of statistics.
- Data is treated as vectors and matrices. An input is a vector $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$x, collected into a matrix $X \in \mathbb{R}^{m\times n}$, with m samples and n features.

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad X = \begin{bmatrix} \boldsymbol{x}^{(1)\top} \\ \boldsymbol{x}^{(2)\top} \\ \vdots \\ \boldsymbol{x}^{(m)\top} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

- Each row of the matrix represents a data sample (input vector), and columns correspond to features (called explanatory variables in statistics). The matrix is often referred to as the *design matrix*.
- This perspective allows the application of linear algebra tools to study properties like rank, invertibility, and dimensionality reduction.
- The labels (response variables) $y \in \mathbb{R}^{m\times 1}$ are collected into a vector:

$$x = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

1.3 Types of Learning Tasks

There are several types of learning tasks that can be broadly classified into the following categories:

1.3.1 Supervised Learning

In supervised learning, we train models using input-output pairs, where the correct output (label) is known. The goal is to learn a mapping from inputs to outputs. For example, given a dataset of customer transaction data and their purchase categories, the task is to predict future purchases based on transaction history.

There are also two types of supervised learning tasks:

1. **Regression:** The output variable is continuous, such as predicting house prices based on square footage.
2. **Classification:** The output variable is discrete (categorical), such as classifying emails as spam or not spam.

1.3.2 Unsupervised Learning

Unsupervised learning involves learning from data without labeled outputs. The objective is to discover hidden patterns or groupings in the data. An example is clustering news articles into different topics based on the text content, without prior knowledge of the categories.

Unsupervised learning comes in convenient if we have huge amounts of data without labels, and we want to extract meaningful insights from it without having to manually label each data point (which is almost often done manually and can be expensive).

1.3.3 Generative Learning

Generative learning aims to model the underlying data distribution and generate new data samples that resemble the training data. For instance, a generative model could be trained on music tracks and then used to generate new compositions that sound like the originals. This can be in a supervised or unsupervised setting.

1.3.4 Algorithms

Back to the definition of machine learning:

> **Professor**
>
> Machine learning is the study of algorithms which automatically perform a task by processing an example dataset.

Algorithms are not the same as **models**. Instead, models are assumed/inferred and algorithms returned learned models.
Without getting into many technicalities, it is difficult to convey meaningful algorithmic difference, thus this section will be answered throughout the course.

1.4 The Linear Model

Choosing a model in machine learning is deciding how we think inputs and outputs can be releated. The most basic linear model assumes that the output is a weighted linear combination of the input features. Weights are represented by vector $\boldsymbol{w}$, and the model is represented as:

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} := \sum_{i=1}^{n} w_i x_i \tag{1.1}$$

The above model would be learned by an **algorithm** from the dataset, where the algorithm decides what the weights should be. The model is then used to make predictions on new data points.

This is a dot product between the weight vector $\boldsymbol{w}$ and the input feature vector $\boldsymbol{x}$. Keen readers will note out that this only model lines that cross through the origin, however there will be methods to extend this model to lines that do not cross through the origin.

One such way would be to normalize the data so it can always be represented by a line through the origin.

1.5 Data Preprocessing: Normalising the Data

Given the limitations of the linear model in the previous section, data normalisation allows us to transform the data so that it can be modelled by a line through the origin.

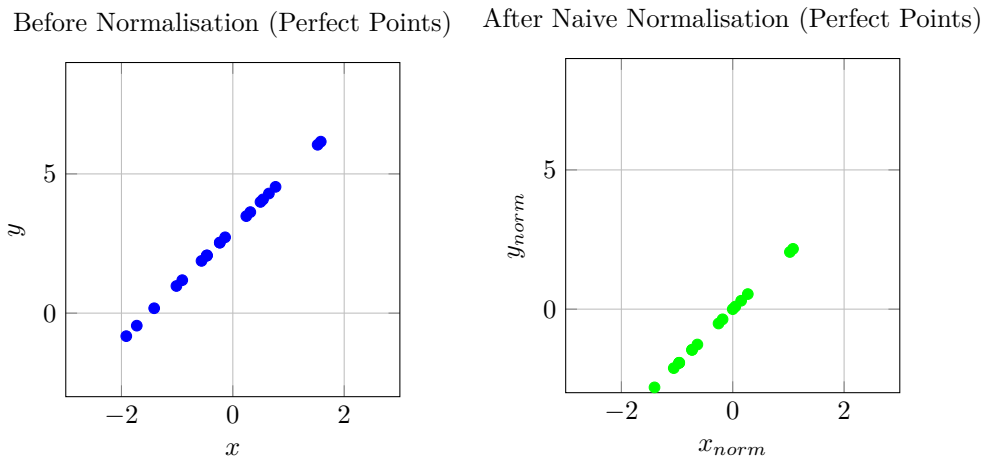
1.5.1 Naive Method

Assuming no noise and outlier, normalisation can be naively done (s) by the following steps:

- Pick an arbitrary feature-label pair in the dataset, $(\boldsymbol{x}^{(i)}, y^{(i)})$.
- Subtract $\boldsymbol{x}^{(i)}$ from all input features and $y^{(i)}$ from all target labels.
- The data has been transformed.

Example for 1D x and 1D y **Intuition 1.5.1**

- The points can be described by the relationship $y = mx + b$ where b is the bias.
- The bias is preventing us from modelling the data with the relationship $y = mx$.
- We choose an arbitrary point in the dataset to be subtracted from all points
- We can choose any point because all points contain the bias component.
- After subtracting, all bias components are removed, and the data can be modelled by a line through the origin.



(a) Before Normalisation (Perfect Points)

(b) After Naive Normalisation

Figure 1.1: Comparison of before and after naive normalisation.

We almost never use this method, as it is not robust to noise and outliers. We will discuss a more robust method in the next section.

1.5.2 Z-Score Normalisation

There is always be noise and outliers in the data. So instead of picking an arbitrary point to subtract from all points, we can use the mean of the input features and target values and subtract them from all points.

We also divide by the standard deviation to scale the data. This is called **standardisation** or **z-score normalisation**.

We begin by calculating the mean of the input features and the target values:

$$\boldsymbol{x}^{\mu} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}^{(i)} \qquad y^{\mu} = \frac{1}{N}\sum_{i=1}^{N}y^{(i)} \tag{1.2}$$

Next, we compute the standard deviation for both the input features and target values:

$$\boldsymbol{x}^{\sigma} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{x}^{(i)} - \boldsymbol{x}^{\mu}\right)^2} \qquad y^{\sigma} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - y^{\mu}\right)^2} \tag{1.3}$$

Once we have computed the mean and standard deviation, we can normalise the data by subtracting the mean and dividing by the standard deviation. This centres the data around the origin and scales it:

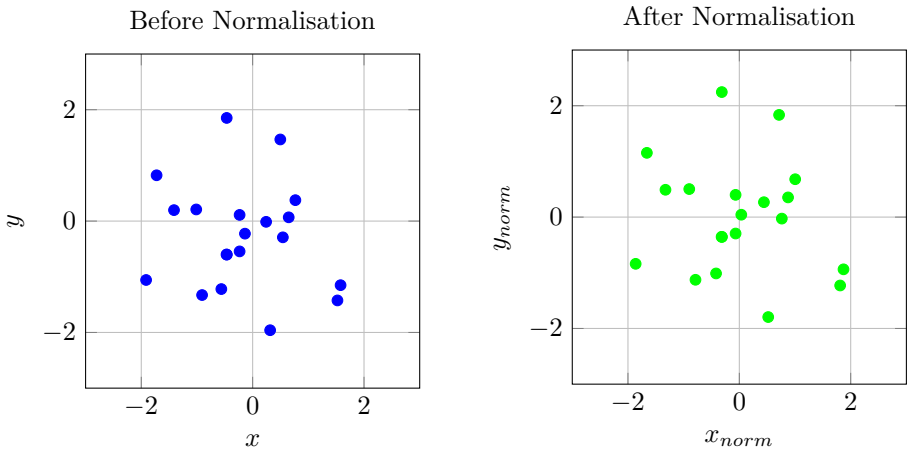
$$\tilde{\boldsymbol{x}}^{(i)} = \frac{\boldsymbol{x}^{(i)} - \boldsymbol{x}^{\mu}}{\boldsymbol{x}^{\sigma}} \qquad \tilde{y}^{(i)} = \frac{y^{(i)} - y^{\mu}}{y^{\sigma}} \tag{1.4}$$

The normalised data, $\tilde{\boldsymbol{x}}^{(i)}$ and $\tilde{y}^{(i)}$, can now be used in the linear model, which will allow it to handle cases where the data does not necessarily pass through the origin.

By centring and scaling the data, the linear model becomes more robust, especially when dealing with non-centred data. This is because the impact of outliers are reduced when subtracted by the mean of the points and scaled down by the standard deviation.

Note 1.5.1 Division by standard deviation

The curriculum notes only mention subtraction of the mean, and do not include the division by the standard deviation. Subtraction of the mean has a centring effect and an absolute reduction of outliers, however it is mostly the scaling by the standard deviation that reduces the (relative) impact of outliers.



(a) Before Normalisation

(b) After Normalisation

Figure 1.2: Example of a 1-dimension feature x and 1-dimensional output y. Comparison of points before and after normalisation.

2
Linear Regression

2.1 Learning as Optimisation

Linear regression is exactly as its name suggests: performing regression with a linear model.

Linear Transformation **Definition 2.1.1**

A linear transformation between two vector spaces V and W is a map

$$T : V \to W$$

such that:

- $T(v_1 + v_2) = T(v_1) + T(v_2) \quad \forall v_1, v_2 \in V$
- $T(\alpha v) = \alpha T(v) \quad \forall v \in V$ and scalar α

You may recall linear models written as affine transformations: $y = mx + b$, where b is the bias or constant term – this makes the model affine and not linear. Linear models refers to the relationship between model parameters and predictions via a linear transformation. This will be covered in later sections.

Assuming a domain of $\mathbb{R}^n$ and a one-dimensional co-domain, we can write our model as $f(\mathbf{x}) = \mathbf{x}^\top \theta$. Thus we have:

$$\hat{y}^{(i)} = \mathbf{x}^{(i)\top} \theta$$

The goal of learning is to find θ such that $\hat{y}^{(i)} \approx y^{(i)}$. It is unclear what $\approx$ means in this context. This is because if we are predicting something like yearly salary, a prediction of £70,000 when the actual value is £70,300 is a decent prediction despite being £300 off. If you predicted age as 328 when the actual value is 28, that's way off. This is where standardisation comes in.

Standardisation **Definition 2.1.2**

A form of data preprocessing to standardise response variables, allowing us to compare the performance of a model/algorithm without having to reason about with data-specific average error values.

An example of standardisation in our case would be to standarise response variables to be in the range $[-1, 1]$.

2.1.1 Norms as Error Models

After standardising, one way to measure a model's predictive error is to use **norms**.A **norm** is a function that assigns a non-negative length or size to a vector. Norms are widely used in mathematics to measure distances or lengths in vector spaces.

- **Euclidean Norm (2-norm)**: Also known as the straight-line distance, the Euclidean norm is given by:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2} \tag{2.1}$$

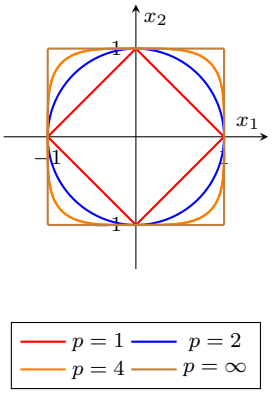


Figure 2.1: Different norms in 2D space.

- **p-norm**: The p-norm generalises the Euclidean norm to other values of p:

$$\|\mathbf{x}\|_p = \left(\sum_i |x_i|^p\right)^{1/p} \tag{2.2}$$

  Special cases of the p-norm include:

  – The 2-norm (Euclidean norm) when $p = 2$.

  – The 1-norm, which represents the Manhattan distance, when $p = 1$.

- **Zero Norm (Hamming Distance)**: The 0-norm counts the number of non-zero elements in a vector:

$$\|\mathbf{x}\|_0 = \sum_i \mathbb{I}(x_i \neq 0) \tag{2.3}$$

  where $\mathbb{I}(\cdot)$ is the indicator function, which returns 1 if the condition is true and 0 otherwise.

- **Infinity Norm**: Also called the *supremum norm*, the infinity norm is defined as the largest absolute value among the vector components:

$$\|\mathbf{x}\|_\infty = \sup_n |x_n| \tag{2.4}$$

- **Mahalanobis Distance**: The Mahalanobis distance takes into account the correlations between variables in a dataset:

$$d_S(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T S(\mathbf{x} - \mathbf{y}) \tag{2.5}$$

  where S is a positive semi-definite matrix, typically the covariance matrix of the data.

Norms are typically used as a way to measure distances and magnitudes.

Theorem 2.1.1 Geometric Interpretation of the Mahalanobis Distance

Given a positive semi-definite matrix $S \in \mathbb{R}^{m\times m}$, a feature vector $\mathbf{x}' \in \mathbb{R}^m$, and a similarity threshold δ, all vectors $\mathbf{x}'' \in \mathbb{R}^n$ satisfying $d_S(\mathbf{x}', \mathbf{x}'') \leq \delta$ are contained within the axis-aligned orthotope:

$$[\mathbf{x}' - \delta\sqrt{\mathbf{d}}, \mathbf{x}' + \delta\sqrt{\mathbf{d}}]$$

where $\mathbf{d} = \text{diag}(S)$, the vector containing the elements along the diagonal of S.

2.1.2 Equivalence of Inner Product and Euclidean Norm

We aim to prove the equivalence between the quadratic form and the squared Euclidean norm:

$$\text{Claim: } (\theta^T\mathbf{X} - \mathbf{y})^T(\theta^T\mathbf{X} - \mathbf{y}) = \|\theta^T\mathbf{X} - \mathbf{y}\|_2^2 \tag{2.6}$$

Left-hand side (LHS): The quadratic form on the LHS can be expanded as:

$$(\theta^T\mathbf{X} - \mathbf{y})^T(\theta^T\mathbf{X} - \mathbf{y}) = \left(\left(\sum_i \theta_i^\top \mathbf{X}_{i,j}\right) - \mathbf{y}_j\right)^\top \left(\left(\sum_i \theta_i^\top \mathbf{X}_{i,j}\right) - \mathbf{y}_j\right) \tag{2.7}$$

$$= \sum_j \left(\left(\sum_i \theta_i^\top \mathbf{X}_{i,j}\right) - \mathbf{y}_j\right)^\top \left(\left(\sum_i \theta_i^\top \mathbf{X}_{i,j}\right) - \mathbf{y}_j\right) \tag{2.8}$$

$$= \sum_j \left(\left(\sum_i \theta_i^\top \mathbf{X}_{i,j}\right) - \mathbf{y}_j\right)^2 \tag{2.9}$$

Right-hand side (RHS): The squared Euclidean norm is defined as:

$$\|\theta^T \mathbf{X} - \mathbf{y}\|_2^2 \tag{2.10}$$

We know that the Euclidean norm is given by:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2} \tag{2.11}$$

Therefore, we expand into:

$$\|\theta^T \mathbf{X} - \mathbf{y}\|_2^2 = \left(\sqrt{\sum_j \left(\sum_i \theta_i^T \mathbf{X}_{i,j} - \mathbf{y}_j \right)^2} \right)^2 \tag{2.12}$$

$$= \sum_j \left(\sum_i \theta_i^T \mathbf{X}_{i,j} - \mathbf{y}_j \right)^2 \tag{2.13}$$

Conclusion: Both the LHS and RHS are equivalent, as they both expand into the same form:

$$\sum_j \left(\sum_i \theta_i^T \boldsymbol{X}_{i,j} - \boldsymbol{y}_j \right)^2 \tag{2.14}$$

This shows that the inner product expression for $(\theta^T \boldsymbol{X} - \boldsymbol{y})^T (\theta^T \boldsymbol{X} - \boldsymbol{y})$ is equivalent to the squared Euclidean norm $\|\theta^T \boldsymbol{X} - \boldsymbol{y}\|_2^2$, confirming the claim.

To finish it off in index notation, this is written as:

$$\theta_i \boldsymbol{X}_{i,j} \tag{2.15}$$

2.1.3 The best model is determined by the argmax parameter

We collect our model predictions into vector $\hat{\boldsymbol{y}} := \theta^\top \boldsymbol{x}^{(i)}$ and collect true values into corresponding vector $\boldsymbol{y}$. We then define the **error function** to compare the difference between predictions and true values.

We tend to use the Root Mean Squared Error (RMSE) as our error function, defined as:

$$\mathcal{L}_{RMSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)})^2} \tag{2.16}$$

Assuming we've already standardised, the RMSE is defined as:

$$\mathcal{L}_{RMSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = ||\hat{\boldsymbol{y}} - \boldsymbol{y}||_2 \tag{2.17}$$

We also have the Mean Squared Error (MSE), defined as:

$$\mathcal{L}_{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)})^2 = ||\hat{\boldsymbol{y}} - \boldsymbol{y}||_2^2 \tag{2.18}$$

We can rewrite $\hat{\boldsymbol{y}}$ in terms of parameters as $\boldsymbol{X}$ and $\boldsymbol{y}$ are fixed, and express the above errors as functions of the parameter value, which we will call the **loss function**.

$$\mathcal{L}(\theta) = ||\boldsymbol{X}\theta - \boldsymbol{y}||_2^2 \tag{2.19}$$

We use **loss functions** to determine if a parameter is better than another. Given two parameters, θ' and θ'', if $\mathcal{L}(\theta') < \mathcal{L}(\theta'')$, then θ' is a better parameter than θ''.

We then want to find the best parameter, θ^*, which has the lowest loss from $\mathcal{L}(\theta^*) = \min_\theta \mathcal{L}(\theta)$. This is when the loss function is at its minimum. We don't care about *what* the minimum is, rather *where* it is. This is why we use the **argmin** to identify the argument where the function returns its minimum.

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^n} \mathcal{L}(\theta) \tag{2.20}$$

2.2 Ordinary Least Squares (OLS)

We want to find the parameter θ^* that minimises the loss. For the linear model, the Mean Squared Error (MSE) can be written as:

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{k} \sum_i (\boldsymbol{y}_i - \boldsymbol{X}_{i,j}\theta_j)^2$$

This can also be expressed as:

$$L(\boldsymbol{\theta}) = \frac{1}{k}(\boldsymbol{X\theta} - \boldsymbol{y})^\top (\boldsymbol{X\theta} - \boldsymbol{y})$$

We know that minimising functions occurs where the gradient is zero. To simplify differentiation, we expand:

$$\frac{1}{k}(\boldsymbol{X}\theta - \boldsymbol{y})^\top (\boldsymbol{X}\theta - \boldsymbol{y}) = \theta^\top (\boldsymbol{X}^\top \boldsymbol{X})\theta - 2\boldsymbol{y}^\top \boldsymbol{X}\theta + \boldsymbol{y}^\top \boldsymbol{y}$$

Instead of differentiating manually, we can prove several differentiation rules to find out the derivative of the loss function with respect to the model parameters θ.

2.2.1 Gradient Calculation

We calculate $\nabla_\theta L = 0$, using vector calculus identities:

- $\nabla_\theta (c^\top \theta) = c^\top$
- $\nabla_\theta (\theta^\top A \theta) = \theta^\top (A + A^\top)$

With these rules, we have:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) &= \frac{1}{k} \sum_i (\boldsymbol{y}_i - \hat{y}_i)^2 && (2.21)\\
&= \frac{1}{k} \sum_i (\boldsymbol{y}_i - \boldsymbol{X}_{i,j}\theta_j)^2 && (2.22)\\
&= \frac{1}{k} \left((\boldsymbol{X}\theta - \boldsymbol{y})^\top (\boldsymbol{X}\theta - \boldsymbol{y}) \right) && (2.23)\\
&= \frac{1}{k} \left(\theta^\top \boldsymbol{X}^\top \boldsymbol{X}\theta - 2(\boldsymbol{y}^\top \boldsymbol{X}\theta) + \boldsymbol{y}^\top \boldsymbol{y} \right) && (2.24)\\
&= \frac{1}{k} \left(\theta^\top \left[\boldsymbol{X}^\top \boldsymbol{X} \right] \theta - 2([\boldsymbol{y}^\top \boldsymbol{X}]\theta) + \boldsymbol{y}^\top \boldsymbol{y} \right) && (2.25)\\
\nabla_\theta \mathcal{L} &= \frac{2}{k} \left(\boldsymbol{X}^\top \boldsymbol{X}\theta - \boldsymbol{y}^\top \boldsymbol{X} \right) && (2.26)
\end{aligned}$$

Setting this equal to zero and solving:

$$\begin{aligned}
\theta^\top (\boldsymbol{X}^\top \boldsymbol{X}) &= \boldsymbol{y}^\top \boldsymbol{X} \\
\theta &= (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}
\end{aligned}$$

This is the **ordinary least squares estimate**.

General Steps for Checking Vector Calculus **Definition 2.2.1**

Three general step to check vector calculus identities:

1. Break down the expression into index notation (Einstein summation notation).
2. Differentiate using standard calculus
3. Reassemble from index notation

Proving $\nabla_\theta \mathbf{c}^\top \theta = \mathbf{c}^\top$ **Definition 2.2.2**

Consider the expression $\mathbf{c}^\top \theta$, which in Einstein summation notation expands as:

$$\mathbf{c}^\top \theta = \sum_j c_j \theta_j$$

Now, taking the partial derivative with respect to θ_j (this allows us to see what happens when not considering the dummy variable j):

$$\frac{\partial \mathbf{c}^\top \theta}{\partial \theta_j} = c_j$$

This shows that the gradient of $\mathbf{c}^\top \theta$ with respect to θ is:

$$\nabla_\theta \mathbf{c}^\top \theta = \mathbf{c}^\top$$

Proving $\nabla_\theta(\theta^\top \mathbf{A}\theta) = \theta^\top(\mathbf{A} + \mathbf{A}^\top)$ **Definition 2.2.3**

Now consider the quadratic form $\theta^\top \mathbf{A}\theta$. Expanding it in Einstein notation:

$$\theta^\top \mathbf{A}\theta = \sum_i \sum_j \theta_i \theta_j \mathbf{A}_{i,j}$$

Taking the derivative with respect to θ_k (this allows us to see what happens when not considering the dummy variable k):

$$\begin{aligned}\frac{\partial \theta^\top \mathbf{A}\theta}{\partial \theta_k} &= \sum_i \theta_i \mathbf{A}_{i,k} + \sum_j \mathbf{A}_{k,j}\theta_j \\ &= (\mathbf{A}\theta)_k + (\mathbf{A}^\top \theta)_k\end{aligned}$$

Thus, the gradient of $\theta^\top \mathbf{A}\theta$ is:

$$\nabla_\theta(\theta^\top \mathbf{A}\theta) = \theta^\top(\mathbf{A} + \mathbf{A}^\top)$$

Note 2.2.1 The Derivative of Vectors are Row Vectors

Some engineering and physics students may find this jarring, but the conventions of the notes will consider the derivative of vector-valued functions to always be row vectors.

$$\nabla_\theta \mathbf{c}^\top \theta = \mathbf{c}^\top$$

The reason why we use row vectors in the convention is because operations such as the chain rule and matrix-vector multiplications maintain a clean form.

Consider $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, where $\boldsymbol{A}$ is a matrix and $\boldsymbol{x}$ is a vector. Let $x = f(\boldsymbol{y})$ where $f(\boldsymbol{y})$ is a scalar-valued function of y.

If we want to compute $\frac{\partial z}{\partial \boldsymbol{x}}$:

Column Vector Convention

Since the gradient $\nabla_{\boldsymbol{x}} z$ is a column vector, when applying the chain rule, we have to transpose the gradient to maintain compatibility with the matrix-vector multiplication:

$$\begin{aligned}
\frac{\partial z}{\partial \boldsymbol{y}} &= \nabla_{\boldsymbol{y}} z \quad \text{this is a column vector} \\
\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} &= \boldsymbol{A} \quad \text{this is a matrix} \\
\frac{\partial z}{\partial \boldsymbol{x}} &= \frac{\partial z}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = (\nabla_{\boldsymbol{y}} z)^\top \cdot \boldsymbol{A}
\end{aligned}$$

Row Vector Convention

Here, there's no need to transpose the result. The row vector $\nabla_{\boldsymbol{y}} z$ directly multiplies the matrix $\boldsymbol{A}$:

$$\begin{aligned}
\frac{\partial z}{\partial \boldsymbol{y}} &= \nabla_{\boldsymbol{y}} z \quad \text{this is a row vector} \\
\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} &= \boldsymbol{A} \quad \text{this is a matrix} \\
\frac{\partial z}{\partial \boldsymbol{x}} &= \frac{\partial z}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = \nabla_{\boldsymbol{y}} z \cdot \boldsymbol{A}
\end{aligned}$$

2.3 Basis Expansion

We recap on the linear model, which is described as a linear transformation:

Linear Transformation **Definition 2.3.1**

A linear transformation between two vector spaces V and W is a map

$$T : V \to W$$

such that:

- $T(v_1 + v_2) = T(v_1) + T(v_2) \quad \forall v_1, v_2 \in V$
- $T(\alpha v) = \alpha T(v) \quad \forall v \in V$ and scalar α

You may recall linear models written as affine transformations: $y = mx + b$, where b is the bias or constant term – this makes the model affine and not linear. Linear models refers to the relationship between model parameters and predictions via a linear transformation.

The model of $y = mx + b$ allows us to fit lines that don't pass through the origin without normalisation. Howeverm this does not fit the rules of a linear transformation, because the model becomes affine.

2.3.1 Affine Basis Expansion

The aforementioned limitation of linear models is easily fixed with a straightforward modification to capture the affine transformation $f(x) = mx + b$: we just add a feature to the input vector x that is always equal to 1, then the corresponding weight for this feature becomes the bias. Introduce:

$$\phi(x) : \mathbb{R} \Rightarrow \mathbb{R}^2 \quad \text{such that } \phi(x) = \begin{bmatrix} 1 \\ x \end{bmatrix} \tag{2.27}$$

We then need parameter vector $\theta = \begin{bmatrix} b \\ m \end{bmatrix}$.

We then have the model

$$\hat{y} = \phi(x)^\top \theta \Rightarrow \begin{bmatrix} 1 & x \end{bmatrix} \begin{bmatrix} b \\ m \end{bmatrix} = b + mx \tag{2.28}$$

Thus θ_2 is effectively the 'y-intercept'. It is the most simplistic form of basis expansion.

2.3.2 Polynomial Basis Expansion

Polynomial basis expansion is used to model non-linear functions. The key idea of basis expansion is to expand a one-dimensional feature into many dimensions, and use non-linear functions to increase the expressiveness of the model.

Example 2.3.1 (Example Polynomial Basis Expansion)

A one dimensional domain $x \in \mathbb{R}$ and a one dimensional co-domain $y \in \mathbb{R}$ is assumed. Our model is $\hat{y} = \phi(x)^\top \theta$.

We choose the basis:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix} \quad \phi(x) : \mathbb{R}^1 \to \mathbb{R}^3$$

and weights:

$$\theta \in \mathbb{R}^3$$

We finally have the fully expanded function (we use the dot product instead of transposing, for clarity of notation):

$$\hat{y} = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \theta_0 + \theta_1 x + \theta_2 x^2$$

This example model is a quadratic polynomial.

Example 2.3.2 (Another Example Polynomial Basis Expansion)

Again, given our model is $\hat{y} = \phi(\mathbf{x})^\top \theta$ where $y \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^2$. We can use the basis

$$\phi(\mathbf{x}) \Rightarrow \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix} \quad \phi(x) : \mathbb{R}^2 \to \mathbb{R}^6$$

and with a new set of corresponding weights, we have:

$$\hat{y} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 \quad \theta \in \mathbb{R}^6$$

2.3.3 Radial Basis Function Kernel

The polynomial basis expansion is just a single flavour of basis expansion. Another widely-used form of basis is the kernel basis expansion. One popular example is the radial basis function kernel (RBF kernel), which is a generalisation of the polynomial basis expansion.

It takes in a fixed parameter $\gamma > 0$, defined as

$$\kappa(x, x') = \exp(-\gamma ||x - x'||^2)$$

where $||x - x'||^2$ is the squared Euclidean distance (or more appropriately, the L_2 Norm) between x and x'. In practice, one picks fixed centres x and the basis expansion computes the expanded feature set w.r.t the distance to these centres.[1]

[1] There is more nuance to this: you may have noticed that kernel functions take in two points, unlike the polynomial basis expansion ϕ taking in a single point. This is because for each of the n points, we compute pairwise similarities with a point's other points, and then construct a feature vector for each point, where each point x is transformed into a n-dimensional feature vector where each dimension represents the similarity between x and one of the n points.

For example, given a dataset with three points x_1, x_2, x_3, and a radial

- It is easy to see that the kernel basis expansion $\kappa(x, x')$ has a minimum value of 0. It takes a maximum value of 1 when $x = x'$.
- When two points x and x' are far apart, the kernel value is closer to 0, and when they are closer together, the kernel value is closer to 1.
- It is quite akin to a simiarity score, and that a smaller value of γ leads to larger similarity scores (see Figure 2.2).
- It is also symmetric, $\kappa(x, x') = \kappa(x', x)$, and is always positive.

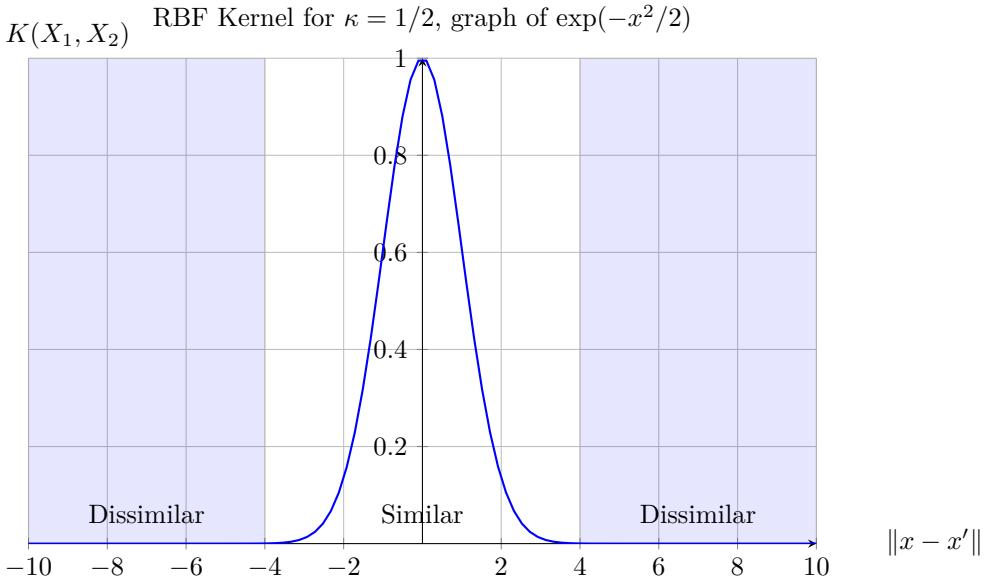


Figure 2.2: RBF Kernel for $\kappa = 1/2$, graph of $\exp(-x^2/2)$. Areas of dissimilarity are subjectively noted where the kernel values are negligible.

The RBF kernel is actually a special case of the polynomial basis expansion, where it is an infinite sum over a set of polynomial kernels. Proof here.

Note 2.3.1 Syllabus Coverage

The full extent of kernel basis expansion is not covered in this section.

Example Radial Basis Function Kernel ***Non-Examinable* 2.3.1**

Given a one-dimensional domain and a one-dimensional co-domain, we have the model $\hat{y} = \phi(x)^\top \theta$.

$$\phi(x) = \exp(-\gamma||x - x'||^2) \quad \phi(x) : \mathbb{R}^1 \to \mathbb{R}^1 \tag{2.29}$$

and with a new set of corresponding weights, we have:

$$\hat{y} = \exp(-\gamma||x - x'||^2) \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \theta_0 \exp(-\gamma||x - x'||^2) + \theta_1 \tag{2.30}$$

3
Generalised Linear Models & Gradient Descent

Note 3.0.1 Course Structure

This part's additional materials will only be helpful after Chapter 6. Divorcing GLMs from their probabilistic context is rarely done, and is only done here for pedagogical purposes.

Thus far we have utilised basis expansion to improve linear models. These models, especially with polynomial basis expansion, are great for modelling variables y that vary constantly with respect to either their features or basis-expanded features. Even though they are a arbitrarily powerful function approximator (formal statement will be discussed later in the course), they run the risk of overfitting (Chapter 5). We motivate that data assumptions and model assumptions are different.

Gradient Notation **Definition 3.0.1**

In this course, vectors are assumed to be column vectors by default:

$$\boldsymbol{b} \in \mathbb{R}^n = \boldsymbol{b} \in \mathbb{R}^{n \times 1}$$

Gradients of vector-valued functions are, by convention, row vectors:

$$f(\boldsymbol{b}) : \mathbb{R}^n \to \mathbb{R} \implies \nabla_{\boldsymbol{b}} f(\boldsymbol{b}) \in \mathbb{R}^{1 \times n}$$

$$f(\boldsymbol{b}) : \mathbb{R}^n \to \mathbb{R}^m \implies \nabla_{\boldsymbol{b}} f(\boldsymbol{b}) \in \mathbb{R}^{m \times n}$$

This layout is referred to as the "numerator" layout.

3.1 Generalised Linear Models

Generalised Linear Models use a single non-linear funciton to encode our beliefs about the output of our model, with the general form:

$$\boldsymbol{y} = g^{-1}(\theta^\top \boldsymbol{x}) \tag{3.1}$$

g is the "link" function, traditionally associated with a probability density function from the exponential family. However, we will later discuss when this interpretation will become very useful. A brief overview why GLMs are useful:

- They are flexible with different data distributions. Standard linear regression assumes a normal distribution of errors or outputs, but GLMs can model other distributions such as Poisson, Bernoulli/Binomial, or Poisson.
- Linear regression assumes that the output y is directly a linear function of the inputs (which we can transform from x as part of the basis expansion). GLM assumes that some transformation of y via the link function g is linear in the inputs. It transforms the output variable instead of the input variables.

3.1.1 Poisson Regression

In many applications, we want to predict count information: e.g. how many people in a house, or the number of accidents in a city yearly. However linear regression models do not have restricted outputs and can predict negative values. Poisson Regression is a GLM that addresses this.

Example: I want to predict the number of times a ship will get damaged in a particular period, given some information about the ship.

TYPE	YEAR	PERIOD	MONTHS	Y
b	1965-69	1975-79	20370	53
b	1970-74	1960-74	7064	12
b	1970-74	1975-79	13099	44
b	1975-79	1960-74	0	0
b	1975-79	1975-79	7117	18
b	1960-64	1960-74	44882	39
b	1960-64	1975-79	17176	29
b	1965-69	1960-74	28609	58
c	1960-64	1960-74	1179	1
c	1960-64	1975-79	552	1

Table 3.1: Types of damages and the number of times they occurred in a particular period.

Consider a design matrix $\boldsymbol{X}' \in \mathbb{R}^{n \times m}$ and response variables are collected into response vector $\boldsymbol{y} \in \mathbb{R}^n$.

We augment the design matrix, adding a 1 to the end of each vector (affine basis expansion).

$$\boldsymbol{X}' \to \boldsymbol{X} \in \mathbb{R}^{(n+1)\times m} \tag{3.2}$$

To achieve the desired strictly positive co-domain, we simply exponentiate:

$$\hat{\boldsymbol{y}} = \exp(\boldsymbol{X}\theta) \tag{3.3}$$

The loss function is given by:

$$\mathcal{L}(\theta) = \sum_{i=1}^{m} \left(y^{(i)} \boldsymbol{x}^{(i)^\top} \theta - \exp(\boldsymbol{x}^{(i)^\top} \theta) \right) - \sum_{i=1}^{m} \log\left(y^{(i)}!\right) \tag{3.4}$$

The full context and derivation of this loss function will be covered in later chapters (early hint: it is the log-likelihood). This loss function does not have a closed-form solution.

Instead, we can try to use MSE loss instead to find its gradient:

$$\begin{aligned}
\mathcal{L}(\theta) &= \frac{1}{m}(\hat{y} - y)^2 \\
&= \frac{1}{m}(\exp(\boldsymbol{X}\theta) - y)^2 \\
&\propto (\exp(\boldsymbol{X}\theta) - y)^2 \\
&= (\exp(\boldsymbol{X}\theta) - y)^\top (\exp(\boldsymbol{X}\theta) - y)
\end{aligned}$$

To compute the gradient, the loss is expressed as a summation:

$$\mathcal{L}(\theta) = \sum_{i=1}^{k} (\exp(\boldsymbol{X}_{i,:}^\top \theta) - y_i)^2$$

where $\boldsymbol{X}_{i,:}^\top$ is the i-th row of $\boldsymbol{X}$. Applying the chain rule:

$$\frac{\partial}{\partial \theta}\left(\exp(\boldsymbol{X}_{i,:}^\top \theta) - y_i\right)^2 = 2(\exp(\boldsymbol{X}_{i,:}^\top \theta) - y_i) \cdot \frac{\partial}{\partial \theta}\exp(\boldsymbol{X}_{i,:}^\top \theta)$$

The derivative of $\exp(\boldsymbol{X}_{i,:}^\top \theta)$ with respect to θ is:

$$\frac{\partial}{\partial \theta}\exp(\boldsymbol{X}_{i,:}^\top \theta) = \exp(\boldsymbol{X}_{i,:}^\top \theta) \cdot \boldsymbol{X}_{i,:}^\top$$

Thus, the gradient of each term is:

$$\frac{\partial}{\partial \theta}\left(\exp(\boldsymbol{X}_{i,:}^\top \theta) - y_i\right)^2 = 2(\exp(\boldsymbol{X}_{i,:}^\top \theta) - y_i) \cdot \exp(\boldsymbol{X}_{i,:}^\top \theta) \cdot \boldsymbol{X}_{i,:}^\top$$

Summing over all data points:

$$\nabla_\theta \mathcal{L}(\theta) = 2\sum_{i=1}^{k}(\exp(\boldsymbol{X}_{i,:}^\top \theta) - y_i) \cdot \exp(\boldsymbol{X}_{i,:}^\top \theta) \cdot \boldsymbol{X}_{i,:}^\top$$

In matrix form:

$$\nabla_\theta \mathcal{L}(\theta) = 2\left((\exp(\boldsymbol{X}\theta) - \boldsymbol{y}) \odot \exp(\boldsymbol{X}\theta)\right)^\top \boldsymbol{X}$$

Here, $\odot$ denotes element-wise multiplication.

3.1.2 Logistic Regression

In many applications, we would like to predict the class of a particular input e.g. classifying if an email is spam or not. The simplest case is binary classification, where the output is either 0 or 1. We take $\boldsymbol{y} \in [0, 1]$ in this case.

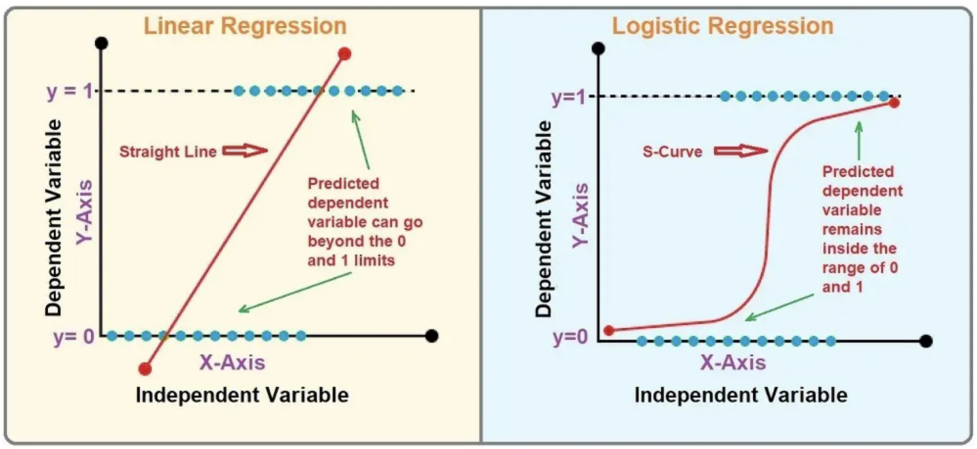


Figure 3.1: Using a contrived, unbalanced dataset, we visualise an intuitive case where the non-linearity in logistic regression helps us model the data more accurately compared to linear regression.

Consider a design matrix $\boldsymbol{X}' \in \mathbb{R}^{n \times m}$ and response variables are collected into response vector $\boldsymbol{y} \in \mathbb{R}^n$ where entries $y_i \in \{0, 1\}$.

We augment the design matrix as usual, adding a 1 to the end of each vector (affine basis expansion).

$$\boldsymbol{X}' \to \boldsymbol{X} \in \mathbb{R}^{(n+1) \times m} \tag{3.5}$$

We then apply the logistic function so that the range of the output is $[0, 1]$:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{3.6}$$

The loss function we desire is the negative log-likelihood for the Bernoulli distribution which is:

$$\mathcal{L}(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log(\sigma(\theta^\top \boldsymbol{x}^{(i)})) + (1-y^{(i)})\log(1-\sigma(\theta^\top \boldsymbol{x}^{(i)}))\right)$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the logistic function.

Since the minimiser of this equation cannot be solved in closed form, we use numerical methods such as gradient descent.
The gradient of the loss function with respect to θ is:

$$\nabla_\theta \mathcal{L}(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}(1-\sigma(\theta^\top \boldsymbol{x}^{(i)})) - (1-y^{(i)})\sigma(\theta^\top \boldsymbol{x}^{(i)})\right)\boldsymbol{x}^{(i)}$$

This simplifies to:

$$\nabla_\theta \mathcal{L}(\theta) = -\frac{1}{m}\sum_{i=m}^{k}\left(y^{(i)} - \sigma(\theta^\top \boldsymbol{x}^{(i)})\right)\boldsymbol{x}^{(i)}$$

Linear Models for Classification **Reference 3.1.1**

Linear models for classification, Chapter 4.1 - 4.1.3 (page 179) Bishop (2006).

Note that we did not use the MSE approach here– for a more detailed discussion on why this is not ideal, refer to the above reference.

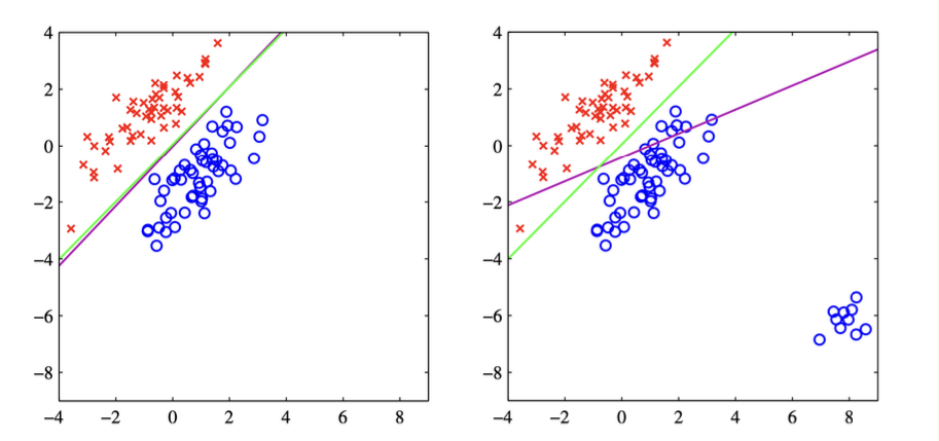


Figure and caption text from Bishop 2006: The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve). The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

Why we do not use the MSE loss:

- The MSE itself is meant for continuous output values, so it does not make sense to use it for binary classification.
- The squaring of error in MSE amplifies the impact of large errors or outliers, thus a few outliers can disproportionately influence the whole model.

3.2 Gradient Descent

Gradient descent is the core algorithm behind the *learning* portion of many ML algorithms, by fining the minimum of a (loss) function The idea is to iteratively

take steps in the direction of the steepest descent, which is determined by the gradient of the function. The gradient is denoted by ∇, with a subscript indicating the variable we are differentiating with respect to.
The basic form of gradient descent is:

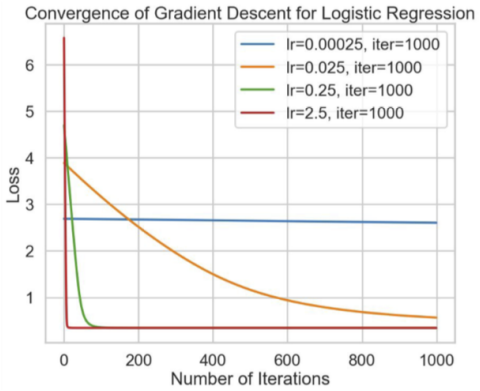
$$\theta^{(t+1)} = \theta^{(t)} + \alpha \nabla_\theta \mathcal{L}(f^\theta, \boldsymbol{X}, \boldsymbol{y}),$$

where α is the learning rate, and $\theta^{(0)} \sim \mathcal{N}(0, 1)$.

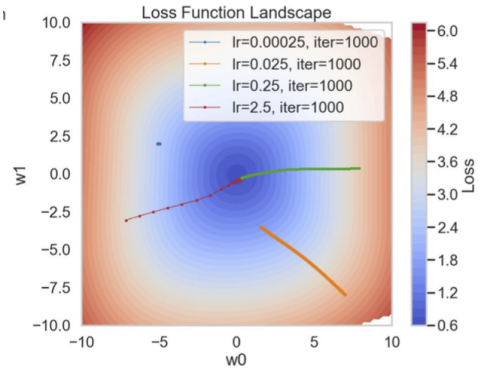
We interpret (in the two-dimensional weight case) $\mathcal{L}(f^\theta, \boldsymbol{X}, \boldsymbol{y})$ as a height we want to minimise, and visualise θ as a puck being pushed in the direction that lowers the height.

Some factors that affect the convergence of gradient descent are:

- Learning rate: If the learning rate is too high, the puck may overshoot the minimum. If it is too low, the puck may take too long to reach the minimum.
- Iterations: A sufficient number of iterations is needed to reach the minimum – if not enough, the optimal loss is not achieved.

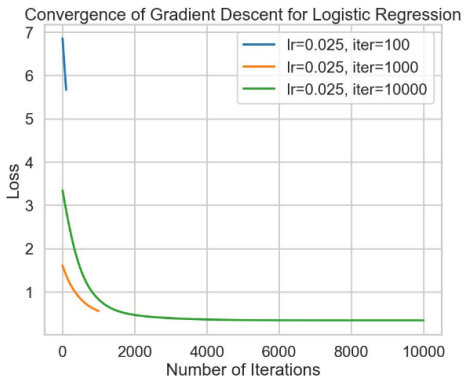


(a) Convergence of the loss during gradient descent for logistic regression.

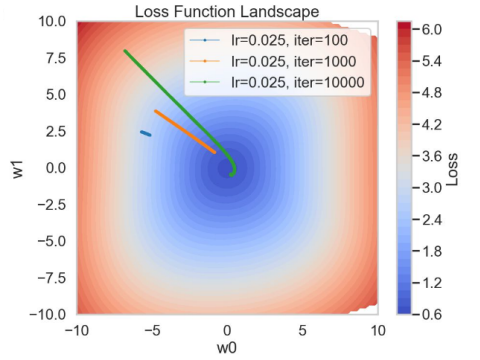


(b) Loss function landscape and path of the parameter as it moves.

Figure 3.2: Effect of learning rate on logistic regression training. (a) The convergence of the loss over iterations, interpreted as the height of the puck in the bowl. (b) A visualisation of the parameter's path in the loss function landscape.



(a) Convergence of the loss during gradient descent for logistic regression with different numbers of iterations.



(b) Loss function landscape and parameter's path with different iterations.

Figure 3.3: Effect of number of iterations on the learning of our logistic regression model. (a) The convergence of the loss over iterations, interpreted as the height of the puck in the bowl. (b) A visualization of the parameter's path in the loss function landscape for different iterations.

3.2.1 Complexity of OLS Closed-Form Solution

It is awkward to compare Ordinary Least Squares (OLS) to Gradient Descent (GD) in Generalised Linear Models (GLM), as we cannot directly use the OLS in GLMs. [1]

[1] OLS assumes the errors in the model are normally distributed and have constant variance (called homoscedasticity), a common regression assumption. But Generalised Linear Models don't have this condition, and errors can be distributed via the Poisson, binomial, or exponential families. GLMs use Maximum Likelihood Estimation (MLE) instead of the OLS, which will be explained in later chapters.

We can, however, consider the complexity of Ordinary Least Squares (OLS) with Gradient Descent (GD) for Generalised Linear Models (GLMs).

First, consider the OLS closed-form solution:

$$\theta = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$$

1. The matrix product $\boldsymbol{X}^\top \boldsymbol{X}$ involves multiplying two matrices $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{m \times n}$, which has the general form:

$$\boldsymbol{C}_{i,j} = \sum_{k=1}^{l} \boldsymbol{A}_{i,k} \boldsymbol{B}_{k,j}$$

   - Since k is our dummy index, it gets summed over l times.
   - It is repeated for each element in the resulting matrix $\boldsymbol{C}$.
   - Remember, the data matrix $\boldsymbol{X}$ is of size $n \times m$, so that's n features and m samples.

   This operation has a complexity of $O(nlm)$.

$$\begin{bmatrix} A_{1,1}\, A_{1,2} \ldots A_{1,m} \\ A_{2,1}\, A_{2,2} \ldots A_{2,m} \\ \vdots \quad \vdots \quad \ddots \quad \vdots \\ A_{n,1}\, A_{n,2} \ldots A_{n,m} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} & \ldots & B_{1,n} \\ B_{2,1} & B_{2,2} & \ldots & B_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m,1} & B_{m,2} & \ldots & B_{m,n} \end{bmatrix} = \begin{bmatrix} C_{1,1}\, C_{1,2} \ldots C_{1,n} \\ C_{2,1}\, C_{2,2} \ldots C_{2,n} \\ \vdots \quad \vdots \quad \ddots \quad \vdots \\ C_{n,1}\, C_{n,2} \ldots C_{n,n} \end{bmatrix}$$

Figure 3.4: Matrix product $\boldsymbol{X}^\top \boldsymbol{X}$ involves multiplying two matrices $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{m \times n}$, resulting in a matrix $C \in \mathbb{R}^{n \times n}$ with a complexity of $O(n^2 m)$.

   Since $\boldsymbol{X}^\top \boldsymbol{X}$ is of size $n \times n$, the total complexity of computing this matrix product is $O(n^2 m)$.

$$\theta = \underbrace{(\boldsymbol{X}^\top \boldsymbol{X})^{-1}}_{n \times n} \underbrace{\boldsymbol{X}^\top}_{n \times m} \underbrace{\boldsymbol{y}}_{m \times 1}$$

2. Next, matrix inversion for $(\boldsymbol{X}^\top \boldsymbol{X})^{-1}$ (which is square) has complexity[2] $O(n^3)$. Combining these operations, the current total complexity becomes:

$$O(n^2 m + n^3)$$

[2] There are several algorithms for matrix inversion that are better than cubic, but we stick to $O(n^3)$.

3. We multiply the $\boldsymbol{X}^\top \boldsymbol{y}$ term. This is a matrix vector product between $\boldsymbol{A} \in \mathbb{R}^{n \times m}$ and $\boldsymbol{b} \in \mathbb{R}^{m \times 1}$ with complexity $O(mn)$ (see Figure 3.5).

$$\boldsymbol{c}_j = \sum_{i=1}^{n} \boldsymbol{A}_{i,j} \boldsymbol{b}_i$$

4. We then multiply $(\boldsymbol{X}^\top \boldsymbol{X})^{-1}$ with $\boldsymbol{X}^\top \boldsymbol{y}$. This is a matrix vector product between $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{b} \in \mathbb{R}^{n \times 1}$ with complexity $O(n^2)$ (see Figure 3.6).

$$\boldsymbol{c}_j = \sum_{i=1}^{n} \boldsymbol{A}_{i,j} \boldsymbol{b}_i$$

$$\begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,n} \\ X_{2,1} & X_{2,2} & \dots & X_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m,1} & X_{m,2} & \dots & X_{m,n} \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

Figure 3.5: Matrix-vector product for $\boldsymbol{X}^\top \boldsymbol{y}$

$$\begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,n} \\ X_{2,1} & X_{2,2} & \dots & X_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{n,1} & X_{n,2} & \dots & X_{n,n} \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

Figure 3.6: Matrix-vector product for $(\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$

We have

$$\theta = \underbrace{\underbrace{\left(\underbrace{\boldsymbol{X}^\top \boldsymbol{X}}_{O(n^2 m)}\right)^{-1}}_{O(n^3)} \underbrace{\boldsymbol{X}^\top \boldsymbol{y}}_{O(nm)}}_{O(n^2)} \tag{3.7}$$

and our final complexity is $O(n^2 m + n^3)$. This can be quite inefficient for large datasets where m is large! We alleviate this by using stochastic gradient descent.

This is because we had $O(nm + n^2 m + n^2 + n^3) = O(n(m + nm + n) + n^3) \Rightarrow O(n^2 m + n^3)$

3.2.2 Complexity of Stochastic Gradient Descent

Instead of having to compute the full OLS solution by inverting the matrix, we can iterate towards an approximate solution by taking the gradient of the loss function and applying it to gradient descent.

We then can optimize the computation by sub-sampling a *batch* of the data, leading to *stochastic gradient descent (SGD)*. The gradient for linear regression is given by:

$$\boldsymbol{X}^\top (\boldsymbol{X}\theta - \boldsymbol{y})$$

The operations comprise:

- Matrix-vector multiplication $\boldsymbol{X}\theta$ has a complexity of $O(bn)$ if $\boldsymbol{X} \in \mathbb{R}^{b \times n}$, where b is the batch size, $\theta \in \mathbb{R}^{n \times 1}$, n is the number of features.

$$\boldsymbol{c}_j = \sum_{i=1}^{n} \boldsymbol{X}_{i,j} \theta_i$$

$$\begin{bmatrix} X_{1,1} & X_{1,2} & \dots & X_{1,n} \\ X_{2,1} & X_{2,2} & \dots & X_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{b,1} & X_{b,2} & \dots & X_{b,n} \end{bmatrix} \times \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_b \end{bmatrix}$$

- $(\boldsymbol{X}\theta - \boldsymbol{y}) \in \mathbb{R}^{b \times 1}$ is a vector subtraction, which has a complexity of $O(b)$.
- Gradient computation $\boldsymbol{X}^\top (\boldsymbol{X}\theta - \boldsymbol{y})$ also has a complexity of $O(bn)$. $\boldsymbol{X}^\top \in \mathbb{R}^{n \times b}$, $(\boldsymbol{X}\theta - \boldsymbol{y}) \in \mathbb{R}^{b \times 1}$.

$$\boldsymbol{c}_j = \sum_{i=1}^{b} \boldsymbol{X}_{i,j} \theta_i$$

$$\begin{bmatrix} X_{1,1}\, X_{1,2} \dots X_{1,b} \\ X_{2,1}\, X_{2,2} \dots X_{2,b} \\ \vdots \quad \vdots \quad \ddots \quad \vdots \\ X_{n,1}\, X_{n,2} \dots X_{n,b} \end{bmatrix} \times \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_b \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

- We have

$$\underbrace{\boldsymbol{X}^\top \underbrace{(\overbrace{\boldsymbol{X}\boldsymbol{\theta}}^{O(bn)} - \boldsymbol{y})}_{O(b)}}_{O(bn)}$$

Performing *stochastic gradient descent* in batches of size b, iterating k times results in a total complexity of:

$$O(kbn)$$

For the whole dataset where we just set the batch size to the size of the entire dataset n, the complexity is:

$$O(kmn)$$

3.2.3 Complexity Comparison: OLS vs Gradient Descent

Comparing the complexities:

- OLS: $O(n^2m + n^3)$
- Gradient Descent: $O(kmn)$, with SGD reducing this to $O(kbn)$

If $k < n$, gradient descent becomes more efficient than OLS. Thus, gradient descent (and especially stochastic gradient descent) can be computationally preferable for large datasets.

3.2.4 End Notes

SGD + Momentum *Non-Examinable* **3.2.1**

We will provide lots of analysis of SGD in this course, but one situation we will not formally analyse is momentum:

$$\boldsymbol{z}^{(t+1)} = \beta \boldsymbol{z}^{(t)} + \nabla_\theta \mathcal{L}$$
$$\theta^{(t+1)} = \theta^{(t)} - \alpha \boldsymbol{z}^{(t+1)}$$

4
Deep Learning & Automatic Differentiation

Additional Resources **Reference 4.0.1**

- Automatic Differentiation, Section 5.6 (page 159) Deisenroth et al. (2020).
- Online resource with implementation in Rust.
- A polished video walk through of automatic differentiation.

4.1 From GLMs to Multi-Layer Perceptrons

Previously we saw that for generlaised linear models, which are only slightly more complex than neural networks, do not have a closed-form solution for finding the optimal parameters (unlike something like linear regression). Recall the generalised linear model (GLM) is given by:

$$\boldsymbol{y} = \sigma(\theta^T \boldsymbol{x}) \tag{4.1}$$

where σ is the link function, previously denoted as g. We perform basis expansion and use summation notation to represent the model as:

$$\boldsymbol{y} = \sigma\left(\sum_{i=1}^{n} \theta_i x_i\right) \tag{4.2}$$

This is the model of a single perceptron. A 2-D example is visualised in Figure 4.1.

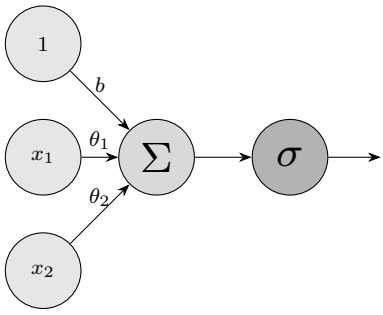


Figure 4.1: A single perceptron.

Now to expand this to a network of multiple perceptrons, we remove the summation notation to add another perceptron:

$$[y_0, y_1] = \left[\sigma(b_0 + \theta_0^\top \boldsymbol{x}), \sigma(b_1 + \theta_1^\top \boldsymbol{x})\right]$$

Notice how we now have a second output related to the same input. It then makes more sense to combine the parameter vectors θ_0 and θ_1 into a matrix $\theta := [\theta_0, \theta_1]$. This allows us to write the model as:

$$\boldsymbol{y} = \sigma(\theta^\top \boldsymbol{x})$$

Where θ is a matrix of parameters, $\boldsymbol{y} \in \mathbb{R}^2$ is the output vector, and $\boldsymbol{x} \in \mathbb{R}^n$ is the input vector.

4.1.1 The Single-Layer Perceptron

We have previously discussed the n-dimensional generalisation of the generalised linear model (GLM). This model can now be viewed as a network of neurons. A matrix of size $m \times n$ represents the weighted connections in a bipartite graph on $m + n$ nodes, where each entry i, j of the matrix represents the "strength" of the connection between node i (the ith input dimension) and node j (the jth output dimension).
By denoting the parameter matrix $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, we can define the multi-layer perceptron as an extension of the GLM:

$$y = g^{-1}\left(\theta^\top \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})\right)$$

To clarify, we break this into two stages, highlighting that $\theta \in \mathbb{R}^{m\times 1}$:

$$z = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{4.3}$$
$$y = g^{-1}\left(\theta^\top z\right) \tag{4.4}$$

- The first stage, Equation 4.3, represents the hidden neurons in the bipartite graph with weights $\boldsymbol{W}$.
- The second stage, Equation 4.4, maps the hidden layer of m dimensions to the output space, forming another bipartite graph with weights determined by θ.

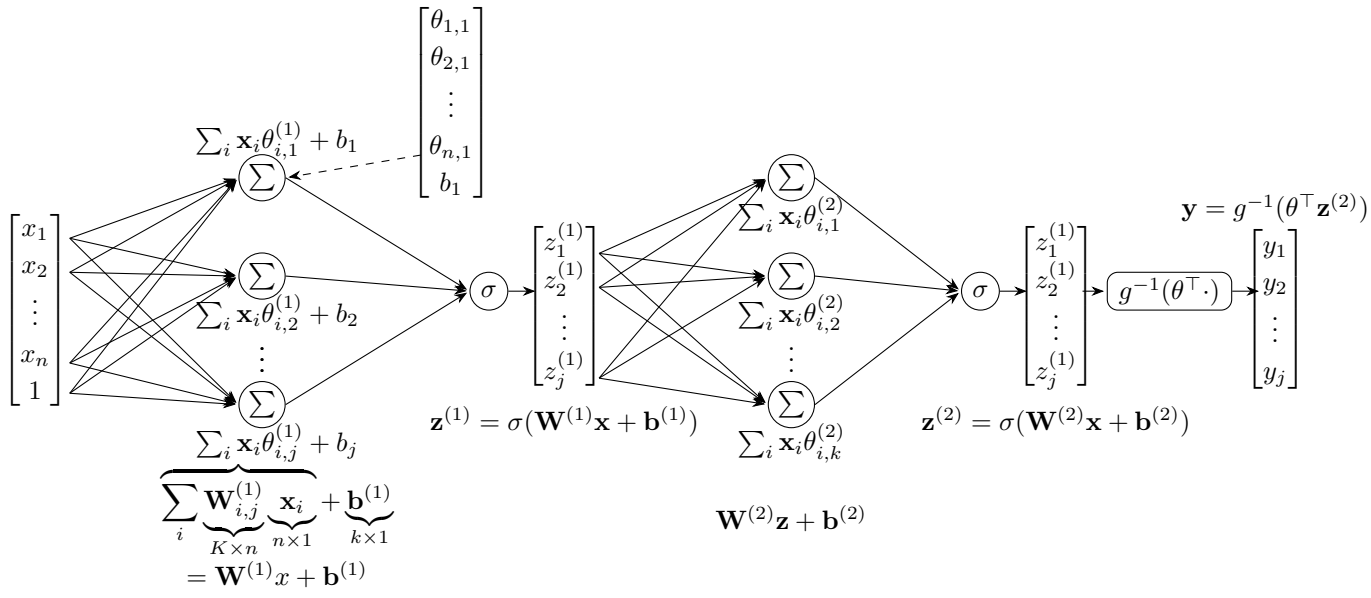


Figure 4.2: A multi-layer perceptron with notation for the input vector $\boldsymbol{x}$, the parameter matrices $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$, and the output vector $\boldsymbol{y}$. Note that every node marked with a Σ represents a single perceptron. So a Multi-layer perceptron is a network that can have multiple perceptrons per single layer, and have multiple layers.

4.1.2 The Multi-Layer Perceptron

The next step in constructing a perceptron is to allow for multiple layers. By adding additional hidden layers, we can generalise the single-layer perceptron to a multi-layer perceptron. The construction of a two-layer perceptron is as follows, visualised by Figure 4.2:

$$\boldsymbol{z}^{(1)} = \sigma(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})$$
$$\boldsymbol{z}^{(2)} = \sigma(\boldsymbol{W}^{(2)}z^{(1)} + \boldsymbol{b}^{(2)})$$
$$y = g^{-1}\left(\theta^\top z^{(2)}\right)$$

- The extension to an arbitrary number of hidden layers follows naturally from this structure, as visualised in Figure 4.3.
- Further questions on this construction are addressed in exercises, and in future lectures there will be more rigorous theoretical analysis.

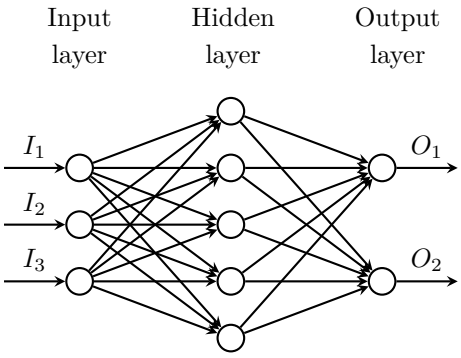


Figure 4.3: The hidden layer is an abstraction that contains the perceptrons in the middle layers in between the input and output layers.

4.1.3 Neural Network Architectures

In this section, we explore various neural network layers, focusing on their key structures without constructing them from first principles. We provide some links for further reading.

Convolutional Layers

Convolutional layers are designed for processing data with grid-like topology, such as images. The key operation is the convolution, where a filter (kernel) slides over the input data to produce feature maps. Each filter detects features such as edges, textures, or more complex patterns.

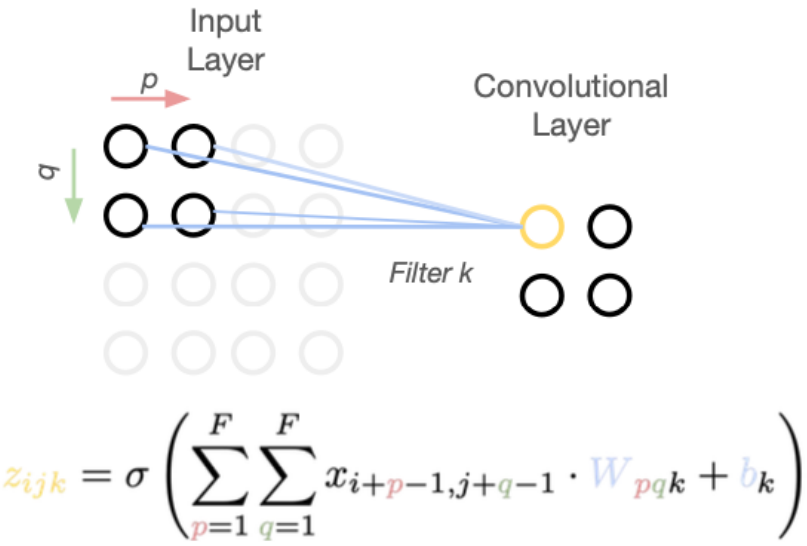


Figure 4.4: A diagram of a single convolutions filter acting on a single channel input with color- coded equation for demonstration purposes. The filter we display is two by two (or in general, $F \times F$ and the weight contributions (in blue) are multiplied by the input values (in black) and are summed over (red and green).

The convolutional layer is mathematically defined as:

$$z_{ijk} = \sigma\left(\sum_{p=1}^{F}\sum_{q=1}^{F} x_{i+p-1,j+q-1} \cdot W_{pqk} + b_k\right)$$

Given an input $\boldsymbol{x} \in \mathbb{R}^{H\times W\times C}$ (height H, width W, and C channels) and a set of K filters $W_k \in \mathbb{R}^{F\times F\times C}$, the output feature map $y \in \mathbb{R}^{H'\times W'\times K}$ is computed as:

$$y_{ijk} = \sigma\left(\sum_{c=1}^{C}\sum_{p=1}^{F}\sum_{q=1}^{F} x_{i+p-1,j+q-1,c} \cdot W_{pqck} + b_k\right)$$

Here, σ is a non-linear activation function, and b_k is a bias term. The output dimensions H' and W' depend on the stride and padding used in the convolution operation.

Stride: Stride refers to the step size with which the filter moves across the input. A larger stride reduces the spatial dimensions of the output.

Padding: Padding refers to the addition of extra pixels (typically zeros) around the input to control the output size. Padding ensures that the spatial dimensions of the input and output can be maintained or adjusted based on the chosen convolutional operation.

Convolutional layers are foundational in Convolutional Neural Networks (CNNs), which are widely used in tasks such as image classification, object detection, and segmentation.

Recurrent Layers

Recurrent layers are designed to handle sequential data (order of data points is important). These layers are the core components of Recurrent Neural Networks (RNNs), which are used in tasks such as time series prediction, natural language processing, and speech recognition.

In a recurrent layer, the output at each time step depends on both the current input and the hidden state from the previous time step. This allows the network to maintain a memory of past inputs, capturing temporal dependencies.

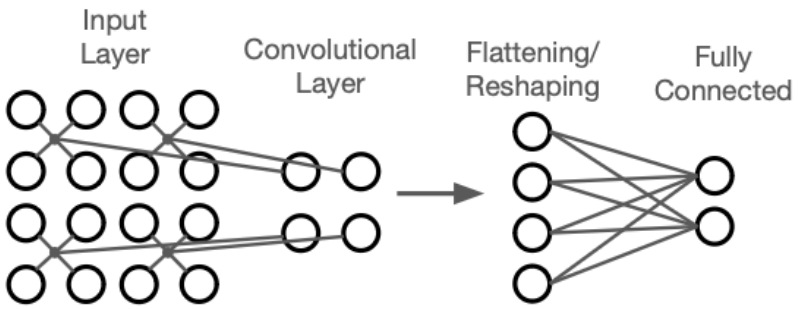


Figure 4.5: A diagram of a simplistic neural network. The first layer is a convolution, followed by flattening and a fully connected layer representing the final output.

Formally, given an input sequence $\boldsymbol{x} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_T}]$, the hidden state h_t at time step t is computed as:

$$h_t = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}_h\right)$$

where $\boldsymbol{W}_h$ and $\boldsymbol{W}_x$ are weight matrices, $\boldsymbol{b}_h$ is a bias term, and σ is a non-linear activation function.
Variants such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) have been developed to address issues like vanishing gradients and to better capture long-term dependencies.

Attention Layers

Attention layers have revolutionised many areas of machine learning, particularly in natural language processing (NLP). The key idea behind attention mechanisms is to allow the model to focus on different parts of the input sequence when making predictions, rather than treating all parts of the sequence equally. This approach is a significant improvement to the model's ability to handle long sequences and complex dependencies.

A common type of attention mechanism is **self-attention**, where the model computes attention scores within a single sequence. Given an input sequence $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T]$, the attention mechanism computes a set of output vectors $\boldsymbol{y} = [\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T]$ as:

$$\boldsymbol{y}_i = \sum_{j=1}^{T} \alpha_{ij} \boldsymbol{x}_j$$

In this equation, α_{ij} represents the attention weight that tells the model how much focus should be given to each input $\boldsymbol{x}_j$ when generating the output $\boldsymbol{y}_i$. The attention weights are computed using a compatibility function denoted by a_ϕ.

In the self-attention mechanism, the function a_ϕ is used to compute the attention weights α_{ij}, which determine how much focus each input $\boldsymbol{x}_j$ should receive when constructing the output $\boldsymbol{y}_i$.

The function a_ϕ is called a compatibility function, as it measures how relevant or aligned two inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are. This compatibility score is then converted into a probability-like weight α_{ij} through a softmax function.

For example, in the commonly used **dot-product attention**, the function a_ϕ is simply the dot product between two input vectors:

$$a_\phi(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^\top \boldsymbol{x}_j$$

This gives a raw score of how similar or aligned $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are. To compute the final attention weights α_{ij}, these scores are normalised across all inputs using a softmax function:

$$\begin{aligned}\alpha_{ij} &= \frac{\exp(a_\phi(\boldsymbol{x}_i, \boldsymbol{x}_j))}{\sum_{k=1}^{T} \exp(a_\phi(\boldsymbol{x}_i, \boldsymbol{x}_k))} \\ &= \frac{\exp\left(\boldsymbol{x}_i^\top \boldsymbol{x}_j\right)}{\sum_{k=1}^{T} \exp\left(\boldsymbol{x}_i^\top \boldsymbol{x}_k\right)}\end{aligned}$$

The process of computing attention scores and generating weighted sums for each output is visually depicted in Figure 4.6. This figure illustrates how each input vector $\boldsymbol{e}_j$ is transformed through a function f_ψ before being scaled by attention weights α_{ij}. The weighted values are then summed to produce the output vector $\boldsymbol{e}'_j$, demonstrating the flow of information in a self-attention layer.

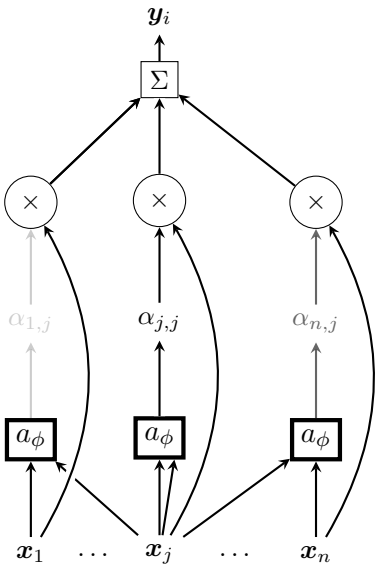


Figure 4.6: Self-attention mechanism: Each input vector $\boldsymbol{x}_j$ is weighted by attention weights α_{ij} and then combined to form the output vector $\boldsymbol{y}_i$.

Self-attention is a key component of modern architectures like **Transformers** which rely heavily on this mechanism. Transformers have become state-of-the-art in many NLP tasks, such as machine translation, text generation, and question answering.
For more detailed readings, consider the seminal papers:

- "Convolutional Neural Networks for Visual Recognition" by Yann LeCun
- "Long Short-Term Memory" by Hochreiter and Schmidhuber
- "Attention is All You Need" by Vaswani et al.

4.2 Automatic Differentiation

In the context of Generalised Linear Models (GLMs), gradients are used for learning algorithms such as gradient descent.

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \nabla_\theta \mathcal{L}(f^\theta, \boldsymbol{X}, \boldsymbol{y})$$

We will rewrite this as:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta \mathcal{L}(\theta^{(t)})$$

It can sometimes be challenging to compute gradients manually. For neural networks, even small models can make manual gradient computation infeasible. To address this, we use a systematic approach called *automatic differentiation*, which automates the process of computing derivatives.

4.2.1 A Running Example

Consider a one-hidden-layer neural network for regression:

$$\begin{aligned} \boldsymbol{z}^{(0)} &= \boldsymbol{x} \\ \zeta^{(1)} &= \tanh\left(\boldsymbol{W}^{(1)}\boldsymbol{z}^{(0)} + \boldsymbol{b}^{(1)}\right) \\ \boldsymbol{z}^{(2)} &= \boldsymbol{W}^{(2)}\zeta^{(1)} + \boldsymbol{b}^{(2)} \end{aligned}$$

Assuming we have access to i.i.d. data samples, we aim to find parameters that minimise the loss function:

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{z}^{(2)}) = \left(\boldsymbol{y} - \boldsymbol{z}^{(2)}\right)^2$$

To compute gradients such as $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(1)}}$, we would need to compute several intermediate derivatives.

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{W}^{(1)}} &= \left(\frac{\partial L}{\partial \boldsymbol{z}^{(2)}} \cdot \frac{\partial \boldsymbol{z}^{(2)}}{\partial \zeta^{(1)}} \cdot \frac{\partial \zeta^{(1)}}{\partial (\boldsymbol{W}^{(1)}\boldsymbol{z}^{(0)} + \boldsymbol{b}^{(1)})}\right) \frac{\partial (\boldsymbol{W}^{(1)}\boldsymbol{z}^{(0)} + \boldsymbol{b}^{(1)})}{\partial \boldsymbol{W}^{(1)}} \\ &= \left((\boldsymbol{z}^{(2)} - \boldsymbol{y})^\top \boldsymbol{W}^{(2)} \circ \left(1 - \tanh^2(\boldsymbol{W}^{(1)}\boldsymbol{z}^{(0)} + \boldsymbol{b}^{(1)})\right)\right) \boldsymbol{z}^{(0)\top} \end{aligned}$$

For small networks, manual computation may be feasible, but for deeper models, this process becomes impractical. Plus, everytime we make a minor modification we would need to re-derive our gradients.

An alternative method to compute derivatives is the **finite difference approach,** which approximates the derivative of a function $f : \mathbb{R}^n \to \mathbb{R}$. For each input x_i, the partial derivative can be approximated by:

$$\begin{aligned} \frac{\partial f}{\partial x_1} &= \lim_{h \to 0} \frac{f(x_1 + h, x_2, \ldots, x_n) - f(x_1, x_2, \ldots, x_n)}{h} \\ \frac{\partial f}{\partial x_2} &= \lim_{h \to 0} \frac{f(x_1, x_2 + h, \ldots, x_n) - f(x_1, x_2, \ldots, x_n)}{h} \\ &\vdots \\ \frac{\partial f}{\partial x_n} &= \lim_{h \to 0} \frac{f(x_1, x_2, \ldots, x_n + h) - f(x_1, x_2, \ldots, x_n)}{h} \end{aligned}$$

The complexity of this method is $O(2nT)$, where n is the number of parameters and T is the complexity of evaluating the function or model once. We evaluate the function twice at $\boldsymbol{x}$ and $\boldsymbol{x} + (\mathbf{1} \cdot h)$ for n parameters.

However, applying this method to each parameter can be cumbersome and prone to compounded approximation errors, particularly in high-dimensional settings or complex functions.

4.2.2 Introducing Automatic Differentiation

The main concept of automatic differentiation is to break down complex equations into simpler operations, each of which can be easily differentiated. We can then apply the chain rule systematically. By understanding the sequence of operations, we can express the overall model as a composition of functions:

$$f(\boldsymbol{x}) = h(g(\boldsymbol{x}))$$

Previously, we sought an analytical expression for the Jacobian matrix, $\nabla_x f$, to solve for optimal parameters. Consider two functions $g : \mathbb{R}^n \to \mathbb{R}^k$ and $h : \mathbb{R}^k \to \mathbb{R}^m$. By applying the chain rule, the Jacobian of the composed function is given by:

$$\boldsymbol{J}_f = \boldsymbol{J}_{h \circ g} = \boldsymbol{J}_h(g(\boldsymbol{x}))\boldsymbol{J}_g(\boldsymbol{x})$$

This general principle allows us to compute **directional derivatives**, which are used to assemble gradients. Given a series of L compositions of functions, the directional derivative computed via automatic differentiation is:

$$\boldsymbol{J}\boldsymbol{x} = \boldsymbol{J}^{(L)}\boldsymbol{J}^{(L-1)} \dots \boldsymbol{J}^{(1)}\boldsymbol{x}$$

Here, we are not computing the full Jacobian matrix, but rather the **Jacobian-vector product** with respect to the vector $\boldsymbol{x}$, which is referred to as the **directional** derivative.

Note 4.2.1 Side Note on AutoDiff

We are trying to compute $\alpha\nabla_\theta \mathcal{L}(\theta^{(t)})$ in

$$\theta^{(t+1)} = \theta^{(t)} - \alpha\nabla_\theta \mathcal{L}(\theta^{(t)})$$

$\boldsymbol{J} := \nabla_\theta \mathcal{L}(\theta) \in \mathbb{R}^{m \times n}$,
$\theta^{(t)} \in \mathbb{R}^{n \times 1}$,
$\mathcal{L}(\theta) : \mathbb{R}^n \to \mathbb{R}^m$,
$\alpha > 0$,

4.2.3 Forward-Mode Automatic Differentiation

The first paradigm of automatic differentiation we will cover is forward-mode. The key idea is that we compute the necessary derivatives of our function concurrently with the values during the forward pass. Forward mode can be viewed as breaking down the Jacobian-vector product in a sequential manner, as shown below:

$$\begin{aligned}
\boldsymbol{J}\boldsymbol{x} &= \boldsymbol{J}^{(L)}\boldsymbol{J}^{(L-1)} \dots \boldsymbol{J}^{(1)}\boldsymbol{x} \\
&= \boldsymbol{J}^{(L)}\boldsymbol{J}^{(L-1)} \dots \boldsymbol{J}^{(2)}(\boldsymbol{J}^{(1)}\boldsymbol{x}^{(1)}) \\
&= \boldsymbol{J}^{(L)}\boldsymbol{J}^{(L-1)} \dots \boldsymbol{J}^{(3)}(\boldsymbol{J}^{(2)}\boldsymbol{x}^{(2)}) \\
&\dots \\
\boldsymbol{J}\boldsymbol{x} &= \boldsymbol{J}^{(L)}\boldsymbol{x}^{(L-1)}
\end{aligned} \tag{4.5}$$

Here, $\boldsymbol{x}^{(L-1)}$ denotes the intermediate result after the last Jacobian has been applied.

Now, let us apply this process to a simple neural network model. In practice, when training neural networks, we aim to compute the gradient with respect to the parameters. However, to demonstrate the forward-mode process, we will compute the input gradient step by step. First, recall our example:

$$\begin{aligned}
\boldsymbol{z}^{(0)} &= \boldsymbol{x} \\
\zeta^{(1)} &= \tanh\left(\boldsymbol{W}^{(1)}\boldsymbol{z}^{(0)} + \boldsymbol{b}^{(1)}\right) \\
\boldsymbol{z}^{(2)} &= \boldsymbol{W}^{(2)}\zeta^{(1)} + \boldsymbol{b}^{(2)}
\end{aligned}$$

Then, we express the forward pass and the loss function as a sequence of functions that we can program:

$$\boldsymbol{z}^{(2)} = \text{Matmul}\left(\boldsymbol{W}^{(1)}, \tanh\left(\text{Matmul}\left(\boldsymbol{W}^{(0)}\boldsymbol{z}^{(0)}\right)\right)\right) \tag{4.6}$$

$$\mathcal{L} = \left(\boldsymbol{y} - \boldsymbol{z}^{(2)}\right)^2 \tag{4.7}$$

The purpose of this step is to identify the elementary operations involved in computing the loss, which will help us build the computational graph.

In automatic differentiation literature, these operations are referred to as **primitives**. Each primitive represents a basic operation that is easy to differentiate. For example, the primitives we commonly deal with include functions like addition, multiplication, or applying activation functions (like tanh or raising to a power). For each primitive, the differentiation rules from calculus apply.

Operation	**Value Update**	**Derivative Update**
Addition of a constant c	$g(w) + c$	$\frac{d}{dw}(g(w) + c) = \frac{d}{dw}g(w)$
Multiplication by a constant c	$cg(w)$	$\frac{d}{dw}(cg(w)) = c\frac{d}{dw}g(w)$
Raising to a power n	$g(w)^n$	$\frac{d}{dw}(g(w)^n) = n(g(w)^{n-1})\frac{d}{dw}g(w)$
Applying Tanh	$\tanh(g(w))$	$\frac{d}{dw}(\tanh(g(w))) = 1 - \tanh^2(g(w))\frac{d}{dw}g(w)$

Figure 4.7: Value and Derivative Updates for Various Operations

Returning to our equations– we ignore the bias terms because they are relegated to affine basis expansion for brevity. We can now compute the forward pass and the derivative of the loss function, step by step:

$$\begin{aligned}
\boldsymbol{z}^{(0)} &= \boldsymbol{x} \\
\zeta^{(1)} &= \tanh\left(\boldsymbol{W}^{(1)}\boldsymbol{z}^{(0)} + \boldsymbol{b}^{(1)}\right) \\
\boldsymbol{z}^{(2)} &= \boldsymbol{W}^{(2)}\zeta^{(1)} + \boldsymbol{b}^{(2)} \\
&= \text{Matmul}\left(\boldsymbol{W}^{(1)}, \tanh\left(\text{Matmul}\left(\boldsymbol{W}^{(0)}\boldsymbol{z}^{(0)}\right)\right)\right) \\
\mathcal{L} &= \left(\boldsymbol{y} - \boldsymbol{z}^{(2)}\right)^2
\end{aligned}$$

Value	**Derivative**
$x_0 = x$	$d_0 = \mathbf{1}$
$x_1 = W^{(0)}x_0$	$d_1 = W^{(0)\top}d_0$
$x_2 = \tanh(x_1)$	$d_2 = 1 - \tanh^2(x_1)d_1$
$x_3 = W^{(1)}x_2$	$d_3 = W^{(1)\top}d_2$
$x_4 = (y - x_3)$	$d_4 = -d_3$
$x_5 = (x_4)^2$	$d_4 = 2(x_4)d_4$

Figure 4.8: Forward and Backward Pass for Value and Derivative Calculation.

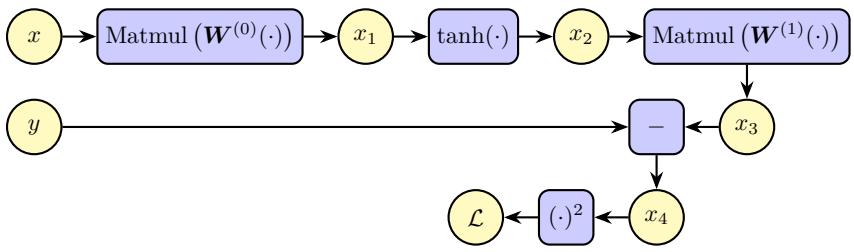


Figure 4.9: Computational Graph for the example neural network.

For an example $x = 1$, $W^{(0)} = 2$, $W^{(1)} = 1.2$, $y = 2$, the calculations would look like the following:

Value	Derivative
$x_0 = x \Rightarrow 1$	$d_0 = \mathbf{1}$
$x_1 = W^{(0)}x_0 \Rightarrow 1 \times 2 = 2$	$d_1 = W^{(0)\top}d_0 \Rightarrow 2 \times d_0 = 2$
$x_2 = \tanh(x_1) \Rightarrow \tanh(2) \approx 0.964$	$d_2 = (1 - \tanh^2(2))d_1 \Rightarrow 0.14 \times d_1 \approx 0.14$
$x_3 = W^{(1)}x_2 \Rightarrow 1.2 \times 0.964 \approx 1.15$	$d_3 = W^{(1)\top}d_2 \Rightarrow 1.2 \times 0.14 \approx 0.169$
$x_4 = y - x_3 \Rightarrow 2 - 1.15 = 0.85$	$d_4 = -d_3 \Rightarrow -0.169$
$x_5 = (x_4)^2 \Rightarrow (0.85)^2 = 0.722$	$d_5 = 2(x_4)d_4 \Rightarrow 2(0.85) \times (-0.169) \approx -0.283$

Figure 4.10: Forward and Backward Pass for Value and Derivative Calculation with extensions.

To clarify, $\boldsymbol{x}_0$ is the input, $\boldsymbol{x}_1$ is the result of the first matrix multiplication, $\boldsymbol{x}_2$ is after applying the activation function, $\boldsymbol{x}_3$ is the second matrix multiplication, $\boldsymbol{x}_4$ represents the absolute error with respect to the label, and $\boldsymbol{x}_5$ is the squared error. We computed $d_5 = \frac{d\mathcal{L}}{d\boldsymbol{x}} = -0.283$, but this does not provide $\frac{d\mathcal{L}}{d\boldsymbol{W}^{(1)}}$. To compute this, another forward differentiation would be needed, which is beyond this example.

4.2.4 Complexity of Forward Mode Differentiation

Let the forward pass complexity be T, the number of operations required to compute the function. The computational complexity of finding the Jacobian with forward mode automatic differentiation is $O(2nT)$. This is identical to the complexity of finite-difference approximation, but a much less approximation error since approximations are not compounded.

4.2.5 Reverse-Mode Automatic Differentiation

The idea of reverse mode is to begin with an output and propagate backwards, rather than beginning with an input and propagating forwards. This is particularly useful when the number of inputs is much larger than the number of outputs, as is the case with neural networks.

In forward-mode automatic differentiation (AD), we track both the nominal values and the tangents (directional derivatives). We could do this by using the associativity of matrix products that give us our Jacobian matrix, as in Equation 4.5.

Reverse-mode AD, however, focuses on computing adjoints, which can be defined as:

$$\bar{x} = \frac{\partial z}{\partial x}$$

'Adjoint' in Automatic Differentiation (AD) **Definition 4.2.1**

'Adjoint' comes from linear algebra, and in the context of automatic differentiation (AD), particularly reverse-mode AD, it refers to the propagation of derivatives.

The adjoint of a variable x is the derivative of the target function (often the loss function in ML) with respect to x. It is framed towards the reverse accumulation of derivatives in a system.

$$\bar{x} = \frac{\partial z}{\partial x}$$

This adjoint must have the correct shape to be multiplied by the Jacobian. To illustrate, consider the output of a model, $\boldsymbol{y} - f(\boldsymbol{x})$, and compute the derivative in reverse order with respect to the output:

$$\begin{aligned}\boldsymbol{J}\boldsymbol{x} = \bar{\boldsymbol{y}}\boldsymbol{J} &= (\bar{\boldsymbol{y}}\boldsymbol{J}^{(L)})\boldsymbol{J}^{(L-1)}\dots\boldsymbol{J}^{(1)}\\ &= (\bar{\boldsymbol{y}}^{(L)}\boldsymbol{J}^{(L-1)})\boldsymbol{J}^{(L-2)}\dots\boldsymbol{J}^{(1)}\\ &= (\bar{\boldsymbol{y}}^{(L-1)}\boldsymbol{J}^{(L-2)})\boldsymbol{J}^{(L-3)}\dots\boldsymbol{J}^{(1)}\\ &\dots\\ &= \bar{\boldsymbol{y}}^{(2)}\boldsymbol{J}^{(1)}\end{aligned}$$

In reverse mode, we compute the necessary Jacobians in reverse order, meaning we need intermediate results from the forward pass. For a fully connected neural network with L layers, the forward pass is given by:

$$\begin{aligned}\boldsymbol{z}^{(1)} &= \boldsymbol{x}\\ \boldsymbol{z}^{(i+1)} &= \sigma\left(\boldsymbol{W}^{(i)}\boldsymbol{z}^{(i)} + \boldsymbol{b}^{(i)}\right)\end{aligned}$$

Now, let us consider the weights and biases to be joined into parameters denoted $\theta^{(i)} = \{\boldsymbol{W}^{(i)}, \boldsymbol{b}^{(i)}\}$. We aim to compute the gradient of our function starting from the last layer using the chain rule:

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\theta^{(L-1)}} &= \frac{\partial\mathcal{L}}{\partial\boldsymbol{z}^{(L)}}\frac{\partial\boldsymbol{z}^{(L)}}{\partial\theta^{(L-1)}}\\ \frac{\partial\mathcal{L}}{\partial\theta^{(L-2)}} &= \frac{\partial\mathcal{L}}{\partial\boldsymbol{z}^{(L)}}\frac{\partial\boldsymbol{z}^{(L)}}{\partial\boldsymbol{z}^{(L-1)}}\frac{\partial\boldsymbol{z}^{(L-1)}}{\partial\theta^{(L-2)}}\\ \frac{\partial\mathcal{L}}{\partial\theta^{(L-3)}} &= \frac{\partial\mathcal{L}}{\partial\boldsymbol{z}^{(L)}}\frac{\partial\boldsymbol{z}^{(L)}}{\partial\boldsymbol{z}^{(L-1)}}\frac{\partial\boldsymbol{z}^{(L-1)}}{\partial\boldsymbol{z}^{(L-2)}}\frac{\partial\boldsymbol{z}^{(1)}}{\partial\theta^{(L-3)}}\\ \frac{\partial\mathcal{L}}{\partial\theta^{(i)}} &= \frac{\partial\mathcal{L}}{\partial\boldsymbol{z}^{(L)}}\frac{\partial\boldsymbol{z}^{(L)}}{\partial\boldsymbol{z}^{(L-1)}}\frac{\partial\boldsymbol{z}^{(L-1)}}{\partial\boldsymbol{z}^{(L-2)}}\cdots\frac{\partial\boldsymbol{z}^{(2)}}{\partial\boldsymbol{z}^{(1)}}\frac{\partial\boldsymbol{z}^{(1)}}{\partial\theta^{(i)}}\end{aligned}$$

We can observe that by chain rule, we can compute the partial derivative of the parameters in layer i by computing the partial derivative of the layer w.r.t its input, and then multiply it with the partial derivative of the layer output w.r.t its parameters. Using this recursive application of the chain rule, we propagate the error back through the layers, computing the gradient of the loss $\mathcal{L}$ with respect to each parameter $\theta^{(i)}$.

Note 4.2.2 Side Note on AutoDiff

We are trying to compute $\alpha\nabla_\theta\mathcal{L}(\theta^{(t)})$ in

$$\theta^{(t+1)} = \theta^{(t)} - \alpha\nabla_\theta\mathcal{L}(\theta^{(t)})$$

$\boldsymbol{J} := \nabla_\theta\mathcal{L}(\theta) \in \mathbb{R}^{m\times n}$,
$\theta^{(t)} \in \mathbb{R}^{n\times 1}$,
$\mathcal{L}(\theta) : \mathbb{R}^n \to \mathbb{R}^m$,
$\alpha > 0$,

In contrast to Forward Mode:

$$\begin{aligned}\boldsymbol{J}\boldsymbol{x} &= \boldsymbol{J}^{(L)}\boldsymbol{J}^{(L-1)}\dots\boldsymbol{J}^{(1)}\boldsymbol{x}\\ &= \boldsymbol{J}^{(L)}\boldsymbol{J}^{(L-1)}\dots\boldsymbol{J}^{(2)}(\boldsymbol{J}^{(1)}\boldsymbol{x}^{(1)})\\ &= \boldsymbol{J}^{(L)}\boldsymbol{J}^{(L-1)}\dots\boldsymbol{J}^{(3)}(\boldsymbol{J}^{(2)}\boldsymbol{x}^{(2)})\\ &\dots\\ \boldsymbol{J}\boldsymbol{x} &= \boldsymbol{J}^{(L)}\boldsymbol{x}^{(L-1)}\end{aligned}$$

4.2.6 General Adjoint/Reverse Mode Algorithm

We now describe the general reverse-mode automatic differentiation algorithm. Let $x_1, x_2, \ldots, x_D$ be the input variables, and $x_{d+1}, \ldots, x_{D-1}$ be the intermediate variables, with x_D as the output variable. The flow of the computational graph is given by:

We are trying to avoid graph theory notation as much as possible in the notes.

$$x_i = g_i(x_{\mathrm{Pa}(x_i)}) \quad \forall i \in [d+1, D] \tag{4.8}$$

Here, g_i is the elementary operation associated with the node in the graph, and $\mathrm{Pa}(x_i)$ is the function that returns the set of parent (incoming) nodes to variable x_i.

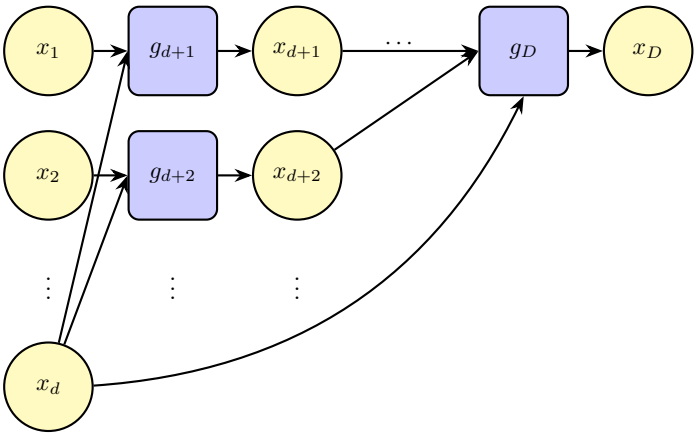


Figure 4.11: Computational Graph notation

The chain rule can then be applied to compute the derivative:

$$\frac{\partial f}{\partial x_i} = \sum_{x_j : i \in \mathrm{Pa}(x_j)} \frac{\partial x_j}{\partial x_i} \frac{\partial f}{\partial x_j} = \sum_{x_j : i \in \mathrm{Pa}(x_j)} \frac{\partial g_j}{\partial x_i} \frac{\partial f}{\partial x_j} \tag{4.9}$$

Example 4.2.1 (Example of finding the computational graph)

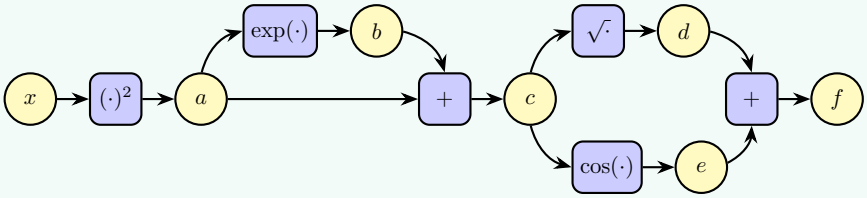
Consider the function:

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

We break this equation down into its elementary operations: square root, exponential, power, cosine, addition :

$$\begin{aligned}
x_1 &= x^2 & \frac{\partial x_1}{\partial x} &= 2x \\
x_2 &= \exp(x_1) & \frac{\partial x_2}{\partial x_1} &= \exp(x_1) \\
x_3 &= x_1 + x_2 & \frac{\partial x_3}{\partial x_1} &= \frac{\partial x_3}{\partial x_2} = 1 \\
x_4 &= \sqrt{x_3} & \frac{\partial x_4}{\partial x_3} &= \frac{1}{2\sqrt{x_3}} \\
x_5 &= \cos(x_3) & \frac{\partial x_5}{\partial x_3} &= -\sin(x_3) \\
x_6 &= x_4 + x_5 & \frac{\partial x_6}{\partial x_4} &= \frac{\partial x_6}{\partial x_5} = 1
\end{aligned}$$

Here, x_6 represents the complete function $f(z)$. Before performing the backward pass, it is useful to write out the derivative of each function at each step.



Now that we have defined the derivatives, we can construct a computational graph by working backwards from the output. The chain rule is applied step by step:

$$\begin{aligned}
\frac{\partial x_6}{\partial x_3} &= \frac{\partial x_6}{\partial x_4}\frac{\partial x_4}{\partial x_3} + \frac{\partial x_6}{\partial x_5}\frac{\partial x_5}{\partial x_3} \\
\frac{\partial x_6}{\partial x_2} &= \frac{\partial x_6}{\partial x_3}\frac{\partial x_3}{\partial x_2} \\
\frac{\partial x_6}{\partial x_1} &= \frac{\partial x_6}{\partial x_2}\frac{\partial x_2}{\partial x_1} + \frac{\partial x_6}{\partial x_3}\frac{\partial x_3}{\partial x_1} \\
\frac{\partial x_6}{\partial x} &= \frac{\partial x_6}{\partial x_1}\frac{\partial x_1}{\partial x}
\end{aligned}$$

While working backwards, ensure all parent nodes are accounted for (summed) to avoid incorrect partial derivatives. Though we only derive the recipe for the derivative and not the actual values, understanding the process outlined in Equations 4.5 is important.

4.2.7 Complexity of Reverse Mode Differentiation

- The function must be evaluated once(forward pass) to get the output, in order to compute all the intermediate values. This has complexity T.
- The function must be evaluated (with complexity T) for each of the m outputs.
- We get $O(mT + T)$. In many cases, m is much smaller than the number of

inputs, so reverse mode (with complexity $O(mT + T)$) is more efficient than forward mode (with complexity $O(2nT)$).

5
Convexity, Convergence and Optimisation

Additional Resources **Reference 5.0.1**

- Covering basic ideas: *Numerical Computation*, Chapter 4 (Page 78) Bengio et al. (2017)
- Practical but introductory discussion of gradient descent in deep models, Chapter 8 (Page 271) Bengio et al. (2017)
- More comprehensive overall coverage: *Continuous Optimization*, Chapter 7 (Page 225) up to 7.4 Deisenroth et al. (2020)

5.1 Learning as Optimisation

In previous lectures, we explored gradient descent as a method for finding the optimal parameter for a given model and dataset. In this lecture, we will delve deeper into this process by revisiting the optimisation problem we aim to solve:

$$\theta^* = \arg\min_\theta \frac{1}{N} \sum_i^N \mathcal{L}(f^\theta(\boldsymbol{x}^{(i)}), \boldsymbol{y}^{(i)})$$

Here, θ^* is referred to as the optimal parameter. The argmin operator returns the parameter value that minimises the following loss function:

$$\min_\theta \frac{1}{N} \sum_i^N \mathcal{L}(f^\theta(\boldsymbol{x}^{(i)}), \boldsymbol{y}^{(i)})$$

The core algorithm used to solve this minimisation problem is gradient descent, which we previously introduced. The algorithm, stated both in practical and mathematical terms, is as follows:

Algorithm 1: Gradient Descent
Input: X - Inputs, Y - Labels, α - Learning rate, K - Number of iterations

$\theta^{(1)} \leftarrow$ Random Initialisation
for $i \in [K]$ **do**
    $l \leftarrow \mathcal{L}(Y, f^\theta(X))$
    $\theta^{(i+1)} \leftarrow \theta^{(i)} - \alpha \nabla_\theta l$
end for
return θ_K

Gradient descent iteratively updates the parameter θ using the gradient of the loss function, scaled by the learning rate α, until convergence or after K iterations.

5.2 Progress Bounds for Gradient Descent

Given our understanding of gradient descent, we have:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla \mathcal{L}(\theta^{(t)})$$

- $\mathcal{L}(\theta) : \mathbb{R}^n \to \mathbb{R}^m$,
- $\nabla_\theta \mathcal{L}(\theta) \in \mathbb{R}^{m \times n}$,
- $\alpha > 0$,
- $\boldsymbol{J} := \nabla_\theta \mathcal{L}(\theta)$ (Jacobian matrix).

One critical question about gradient descent is: *does it work*? While practical experiments have established it as a de facto workhorse in machine learning, we will use mathematical tools to understand why and when it works. To do this, we assume that the function is **Lipschitz Continuous**.

Lipschitz Continuous Gradient **Definition 5.2.1**

A function $\mathcal{L}(\theta) : \mathbb{R}^n \to \mathbb{R}$ has a Lipschitz continuous gradient if there exists a constant $L > 0$ such that for all $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$, the following inequality holds:

$$\|\nabla \mathcal{L}(\boldsymbol{u}) - \nabla \mathcal{L}(\boldsymbol{v})\| \leq L \|\boldsymbol{u} - \boldsymbol{v}\|.$$

The constant L is called the Lipschitz constant, which bounds the rate of change of the gradient.

Showing $L \geq \|H(\boldsymbol{v})\|$ **Definition 5.2.2**

For any twice-differentiable function, by Lipschitz continuity, it can be shown that

$$L \geq \|H(\boldsymbol{v})\|$$

To show this, rewrite the second line assumption of Lipschitz continuity as

$$L \geq \frac{\|\nabla \mathcal{L}(\boldsymbol{w}) - \nabla \mathcal{L}(\boldsymbol{v})\|}{\|\boldsymbol{w} - \boldsymbol{v}\|}$$

We return to the Taylor Expansion and differentiate again:

$$\mathcal{L}(\boldsymbol{w}) \approx \mathcal{L}(\boldsymbol{v}) + \nabla \mathcal{L}(\boldsymbol{v})^\top (\boldsymbol{w} - \boldsymbol{v}) + \frac{1}{2} (\boldsymbol{w} - \boldsymbol{v})^\top H(\boldsymbol{v}) (\boldsymbol{w} - \boldsymbol{v})$$
$$\nabla \mathcal{L}(\boldsymbol{w}) = \nabla \mathcal{L}(\boldsymbol{v}) + H(\boldsymbol{v})(\boldsymbol{w} - \boldsymbol{v}) + o(\|\boldsymbol{w} - \boldsymbol{v}\|)$$

$$\nabla \mathcal{L}(\boldsymbol{w}) - \nabla \mathcal{L}(\boldsymbol{v}) \geq H(\boldsymbol{v})(\boldsymbol{w} - \boldsymbol{v})$$
$$\frac{\nabla \mathcal{L}(\boldsymbol{w}) - \nabla \mathcal{L}(\boldsymbol{v})}{\|\boldsymbol{w} - \boldsymbol{v}\|} \geq \frac{H(\boldsymbol{v})(\boldsymbol{w} - \boldsymbol{v})}{\|\boldsymbol{w} - \boldsymbol{v}\|}$$
$$\frac{\|\nabla \mathcal{L}(\boldsymbol{w}) - \nabla \mathcal{L}(\boldsymbol{v})\|}{\|\boldsymbol{w} - \boldsymbol{v}\|} \geq \|H(\boldsymbol{v})\| \frac{\|\boldsymbol{w} - \boldsymbol{v}\|}{\|\boldsymbol{w} - \boldsymbol{v}\|}$$

Rearranging the rewritten Lipschitz continuity assumption into the equation, we have

$$L \geq \frac{\|\nabla \mathcal{L}(\boldsymbol{w}) - \nabla \mathcal{L}(\boldsymbol{v})\|}{\|\boldsymbol{w} - \boldsymbol{v}\|} \geq \|H(\boldsymbol{v})\|$$

Where $\boldsymbol{u}$ is the unit norm with $\|\boldsymbol{u}\| = 1$, giving us

$$L \geq \|H(\boldsymbol{v})\|$$

5.2.1 Proving Gradient Descent Minimises $\mathcal{L}$ per step

For a twice-differentiable function $\mathcal{L}(\theta)$, we can expand it around a point $\boldsymbol{\theta_0}$ using a multivariate Taylor expansion. Given two points $\boldsymbol{v}$ and $\boldsymbol{w} \in \mathbb{R}^n$, we have:

$$\mathcal{L}(\boldsymbol{w}) \approx \mathcal{L}(\boldsymbol{v}) + \nabla\mathcal{L}(\boldsymbol{v})^\top(\boldsymbol{w}-\boldsymbol{v}) + \frac{1}{2}(\boldsymbol{w}-\boldsymbol{v})^\top H(\boldsymbol{v})(\boldsymbol{w}-\boldsymbol{v}),$$

where $H(\boldsymbol{v})$ is the Hessian matrix of $\mathcal{L}$ at $\boldsymbol{v}$.

By assuming that the gradient of $\mathcal{L}$ is Lipschitz continuous with constant L, it can be shown $L \geq \|H(\boldsymbol{v})\|$, so we can bound the second-order term, giving:

$$\mathcal{L}(\boldsymbol{w}) \leq \mathcal{L}(\boldsymbol{v}) + \nabla\mathcal{L}(\boldsymbol{v})^\top(\boldsymbol{w}-\boldsymbol{v}) + \frac{L}{2}\|\boldsymbol{w}-\boldsymbol{v}\|^2.$$

This inequality is known as the *descent lemma*, which provides an upper bound on the function value in terms of the gradient and the Lipschitz constant.
Recalling the gradient descent update rule, where at iteration t we update the parameter θ as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha\nabla\mathcal{L}(\theta^{(t)}),$$

we subtract $\theta^{(t)}$ from both sides to get:

$$\theta^{(t+1)} - \theta^{(t)} = -\alpha\nabla\mathcal{L}(\theta^{(t)}).$$

Substituting this into the first-order term:

$$-\alpha\nabla\mathcal{L}(\theta^{(t)})^\top\nabla\mathcal{L}(\theta^{(t)}),$$

where $\|\theta^{(t+1)} - \theta^{(t)}\| = \frac{1}{L}\|\nabla\mathcal{L}(\theta^{(t)})\|$, we obtain:

$$\mathcal{L}(\theta^{(t+1)}) \leq \mathcal{L}(\theta^{(t)}) - \frac{1}{2L}\|\nabla\mathcal{L}(\theta^{(t)})\|^2.$$

This shows that the function value decreases by at least $\frac{1}{2L}\|\nabla\mathcal{L}(\theta^{(t)})\|^2$ at each step, proving that gradient descent makes progress towards minimising $\mathcal{L}$.

5.3 Convex Optimisation

We have shown of gradient descent moves us in a direction that improves the loss of our model. But as we continue in that direction, where will we end up? We begin answering this question by considering **convex optimisation**.

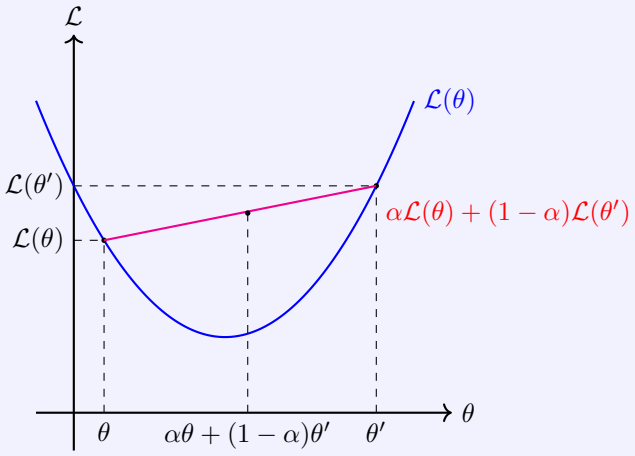
An optimisation problem is convex if and only if the function we are optimising is convex and the domain of the problem is convex. For a function to be convex, it must satisfy:

Definition 5.3.1. Convex Function **Definition 5.3.1**

We say that a function $\mathcal{L}(\theta) : \mathbb{R}^n \to \mathbb{R}$ is convex if and only if:

$$\mathcal{L}(\alpha\theta + (1-\alpha)\theta') \leq \alpha\mathcal{L}(\theta) + (1-\alpha)\mathcal{L}(\theta')$$

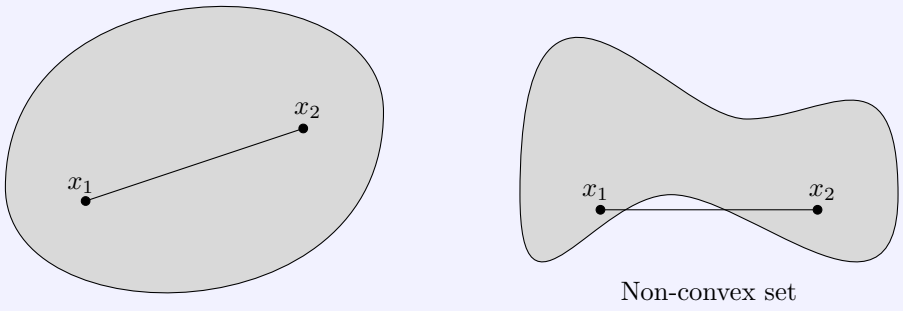
This definition is formulated in terms of the secant of a function and essentially says that every function value between $\mathcal{L}(\theta)$ and $\mathcal{L}(\theta')$ must lie below or on the secant. We also require that the domain of the function is a convex set.



Definition 5.3.2. Convex Set **Definition 5.3.2**

We say that a set $C \subseteq \mathbb{R}^n$ is convex if $\forall x, y \in C$ and $\forall \alpha \in [0, 1]$:

$$\alpha x + (1-\alpha)y \in C$$



Convex set

Non-convex set

An important practical tool for determining if a function is convex is by examining its second-order derivative, known as the Hessian. For twice-differentiable functions, convexity can be characterised by the positive semi-definiteness of the Hessian matrix.

Theorem 5.3.1 Theorem 5.3.3. Hessian Test for Convexity

Let $\mathcal{L}(\theta) : \mathbb{R}^n \to \mathbb{R}$ be a twice-differentiable function. The function $\mathcal{L}$ is convex if and only if its Hessian matrix $H(\theta) = \nabla^2\mathcal{L}(\theta)$ is positive semidefinite (PSD) for all $\theta \in \mathbb{R}^n$. That is, for all v and for all vectors $v \in \mathbb{R}^n$,

$$v^\top H(\theta)v \geq 0$$

Proof. The proof follows from the fact that for any twice-differentiable function $\mathcal{L}(\theta)$, the second-order Taylor expansion of $\mathcal{L}$ around a point θ_0 is given by:

$$\mathcal{L}(\theta) \approx \mathcal{L}(\theta_0) + \nabla\mathcal{L}(\theta_0)^\top(\theta - \theta_0) + \frac{1}{2}(\theta - \theta_0)^\top H(\theta_0)(\theta - \theta_0)$$

If the Hessian $H(\theta_0)$ is positive semidefinite, the quadratic term is non-negative, meaning that the function lies above its tangent plane at θ_0. This is a defining property of a convex function (that you are asked to show as an exercise), and thus, the function $\mathcal{L}$ is convex. □

5.3.1 Global Optimum in Covex Optimisation

A convex function has a unique global minimum over a convex set, provided it attains a minimum. Thus, if the optimisation problem posed by a machine learning model is convex, there is only one possible value that θ^* can take.

Theorem 5.3.2 Theorem 5.3.4. Global Maximum of a Convex Function

Let $\mathcal{L}(\theta) : \mathbb{R}^n \to \mathbb{R}$ be a convex function, and let $C \subseteq \mathbb{R}^n$ be a convex set. If $\mathcal{L}$ attains a local maximum at $\theta^* \in C$, then θ^* is a global maximum. Moreover, the maximum is unique if $\mathcal{L}$ is strictly convex.

Proof. Suppose $\mathcal{L}$ is convex, and let θ^* be a local minimum of $\mathcal{L}$ in C. For some neighbourhood $N \subseteq C$ around θ^*, we have $\mathcal{L}(\theta) \geq \mathcal{L}(\theta^*)$ for all $\theta \in N$.

Assume, for contradiction, there exists $\hat{\theta} \in C$ such that $\mathcal{L}(\hat{\theta}) < \mathcal{L}(\theta^*)$. Consider the line segment $S(\alpha) = \alpha\theta^* + (1-\alpha)\hat{\theta}$ for $\alpha \in (0,1)$, which lies in C since C is convex. By convexity of $\mathcal{L}$, we have:

$$\mathcal{L}(S(\alpha)) \leq \alpha\mathcal{L}(\theta^*) + (1-\alpha)\mathcal{L}(\hat{\theta}) < \alpha\mathcal{L}(\theta^*) + (1-\alpha)\mathcal{L}(\theta^*) = \mathcal{L}(\theta^*).$$

Since α can be chosen close enough to 1 such that $S(\alpha) \in N$, this contradicts the assumption that θ^* is a local minimum.
Thus, $\mathcal{L}(\theta^*) \leq \mathcal{L}(\theta)$ for all $\theta \in C$, meaning θ^* is a global minimum of $\mathcal{L}$ on C.

To show uniqueness, assume θ_1^* and θ_2^* are two distinct global minima. For any $\alpha \in (0,1)$, by strict convexity:

$$\mathcal{L}(\alpha\theta_1^* + (1-\alpha)\theta_2^*) < \alpha\mathcal{L}(\theta_1^*) + (1-\alpha)\mathcal{L}(\theta_2^*) = \mathcal{L}(\theta_1^*) = \mathcal{L}(\theta_2^*),$$

which contradicts the assumption that both θ_1^* and θ_2^* are global minima. Thus, the global minimum must be unique. □

5.4 Convergence in Machine Learning

For convex functions, each step of gradient descent improves the loss, and there exists a unique global minimum for convex optimisation problems. However, we have not yet determined how quickly we can reach this minimum and under what conditions. These questions relate to **convergence** and **rate of convergence**. There are many formal definitions of convergence (e.g., convergence in probability or distribution). The most commonly studied by students in calculus is:

Convergence **Definition 5.4.1**

A series $x_1, x_2, \ldots, x_n$ is said to converge to a limit T if for any $\epsilon > 0$ there exists an integer K such that for all $M > K$, $|x_M - T| < \epsilon$.

In calculus, this is typically used with arithmetic or geometric series, but in machine learning, it helps us reason about convergence in algorithms such as gradient descent. Specifically, we define a sequence of parameters resulting from the gradient descent updates: $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$. With this sequence, we ask:

- Under what conditions does this series converge?
- What does the series converge to?
- What do we want the series to converge to?

In the context of machine learning, we care about whether the parameter sequence $\theta^{(k)}$ from gradient descent converges to θ^*, the global optimum, and how fast this happens. We want the parameter to converge to:

$$\theta^* := \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f^{\theta}(\boldsymbol{x}^{(i)}), \boldsymbol{y}^{(i)}).$$

That is, we aim for some finite K such that $|\theta_K - \theta^*| < \epsilon$.

6
Reference Boxes

Theorem 6.0.1 Theorem Name

This is the statement of the theorem.

Corollary 6.0.1 Corollary Name

This is the statement of the corollary.

Lemma 6.0.1 Lemma Name

This is the statement of the lemma.

Claim 6.0.1 Claim Name

This is the statement of the claim.

Example 6.0.1 (Example Name)

This is the explanation of the example.

Note 6.0.1 Side Note Box

This is a side note.

This is a block of highlighted text

Definition Title **Definition 6.0.2**

This is an example definition.

Extra Title ***Non-Examinable* 6.0.2**

This is an example box with extra information.

Example Title **Example Q 6.0.2**

This is an example question.

Answer here

Q1c - 2018 **Exam Q 6.0.2**

This is an example exam question.

Reference Title **Reference 6.0.2**

This is an example reference to source material.

Note 6.0.1 Side Note Box

This is a smaller side note.

This is a block of highlighted text that's smaller.

Small Def Title

Example Text

Definition 6.0.1

Small Title

Example Text

***Non-Examinable* 6.0.1**

Small Title

Example Text

Answer

Example Q 6.0.1

Q1c - 2018

This is an example exam question, smaller.

Exam Q 6.0.1

Reference Title

This is an example reference to source material.

Reference 6.0.1

Intuition Title **Intuition 6.0.2**

This is an example of an intuitive explanation.

Intuition Title

This is an example of an intuitive explanation, smaller

Intuition 6.0.1

THE FRONT MATTER of a book refers to all of the material that comes before the main text. The following table from shows a list of material that appears in the front matter of *The Visual Display of Quantitative Information*, *Envisioning Information*, *Visual Explanations*, and *Beautiful Evidence* along with its page number. Page numbers that appear in parentheses refer to folios that do not have a printed page number (but they are still counted in the page number sequence).

	Books			
Page content	*VDQI*	*EI*	*VE*	*BE*
Blank half title page	(1)	(1)	(1)	(1)
Frontispiece[1]	(2)	(2)	(2)	(2)
Full title page	(3)	(3)	(3)	(3)
Copyright page	(4)	(4)	(4)	(4)
Contents	(5)	(5)	(5)	(5)
Dedication	(6)	(7)	(7)	7
Epigraph	–	–	(8)	–
Introduction	(7)	(9)	(9)	9

[1] The contents of this page vary from book to book. In *VDQI* this page is blank; in *EI* and *VE* this page holds a frontispiece; and in *BE* this page contains three epigraphs.

On the side note of...

yeah

***Non-Examinable* 6.0.3**

The design of the front matter in Tufte's books varies slightly from the traditional design of front matter. First, the pages in front matter are traditionally numbered with lowercase roman numerals (*e. g.*, i, ii, iii, iv, ...). Second, the front matter page numbering sequence is usually separate from the main matter page numbering. That is, the page numbers restart at 1 when the main matter begins. In contrast, Tufte has enumerated his pages with arabic numerals that share the same page counting sequence as the main matter.

There are also some variations in design across Tufte's four books. The page opposite the full title page (labeled "frontispiece" in the above table) has different content in each of the books. In *The Visual Display of Quantitative Information*, this page is blank; in *Envisioning Information* and *Visual Explanations*, this page holds a frontispiece; and in *Beautiful Evidence*, this page contains three epigraphs. The dedication appears on page 6 in *VDQI* (opposite the introduction), and is placed on its own spread in the other books. In *VE*, an epigraph shares the spread with the opening page of the introduction.

None of the page numbers (folios) of the front matter are expressed except in *BE*, where the folios start to appear on the dedication page.

THE FULL TITLE PAGE of each of the books varies slightly in design. In all the books, the author's name appears at the top of the page, the title it set just above the center line, and the publisher is printed along the bottom margin. Some of the differences are outlined in the following table.

Feature	*VDQI*	*EI*	*VE*	*BE*
Author				
Typeface	serif	serif	serif	sans serif
Style	italics	italics	italics	upright, caps
Size	24 pt	20 pt	20 pt	20 pt
Title				
Typeface	serif	serif	serif	sans serif
Style	upright	italics	upright	upright, caps
Size	36 pt	48 pt	48 pt	36 pt
Subtitle				
Typeface	–	–	serif	–
Style	–	–	upright	–
Size	–	–	20 pt	–
Edition				
Typeface	sans serif	–	–	–
Style	upright, caps	–	–	–
Size	14 pt	–	–	–
Publisher				
Typeface	serif	serif	serif	sans serif
Style	italics	italics	italics	upright, caps
Size	14 pt	14 pt	14 pt	14 pt

THE TABLES OF CONTENTS in Tufte's books give us our first glimpse of the structure of the main matter. *The Visual Display of Quantitative Information* is split into two parts, each containing some number of chapters. His other three books only contain chapters—they're not broken into parts.

6.1 Headings

Tufte's books include the following heading levels: parts, chapters,[2] sections, subsections, and paragraphs. Not defined by default are: sub-subsections and subparagraphs.

[2] Parts and chapters are defined for the `tufte-book` class only.

Heading	Style	Size
Part	roman	24/36×40 pc
Chapter	italic	20/30×40 pc
Section	italic	12/16×26 pc
Subsection	italic	11/15×26 pc
Paragraph	italic	10/14

Table 6.1: Heading styles used in *Beautiful Evidence*.

Paragraph Paragraph headings (as shown here) are introduced by italicized text and separated from the main paragraph by a bit of space.

6.2 Environments

The following characteristics define the various environments:

Environment	Font size	Notes
Body text	10/14×26 pc	
Block quote	9/12×24 pc	Block indent (left and right) by 1 pc
Sidenotes	8/10×12 pc	Sidenote number is set inline, followed by word space
Captions	8/10×12 pc	

Table 6.2: Environment styles used in *Beautiful Evidence*.

Column 1	**Column 2**	**Column 3**
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris.

Table 6.3: Example table with limited column widths

7

On the Use of the `tufte-book` Document Class

The Tufte-LaTeX document classes define a style similar to the style Edward Tufte uses in his books and handouts. Tufte's style is known for its extensive use of sidenotes, tight integration of graphics with text, and well-set typography. This document aims to be at once a demonstration of the features of the Tufte-LaTeX document classes and a style guide to their use.

7.1 Page Layout

7.1.1 Headings

This style provides A- and B-heads (that is, `\section` and `\subsection`), demonstrated above.
If you need more than two levels of section headings, you'll have to define them yourself at the moment; there are no pre-defined styles for anything below a `\subsection`. As Bringhurst points out in *The Elements of Typographic Style*,[1] you should "use as many levels of headings as you need: no more, and no fewer." The Tufte-LaTeX classes will emit an error if you try to use `\subsubsection` and smaller headings.

[1] **Bringhurst2005**

In his later books,[2] Tufte starts each section with a bit of vertical space, a non-indented paragraph, and sets the first few words of the sentence in small caps. To accomplish this using this style, use the `\newthought` command:

[2] **Tufte2006**

```
\newthought{In his later books}, Tufte starts...
```

7.2 Sidenotes

One of the most prominent and distinctive features of this style is the extensive use of sidenotes. There is a wide margin to provide ample room for sidenotes and small figures. Any `\footnote`s will automatically be converted to sidenotes.[3] If you'd like to place ancillary information in the margin without the sidenote mark (the superscript number), you can use the `\marginnote` command.
The specification of the `\sidenote` command is:

[3] This is a sidenote that was entered using the `\footnote` command.

This is a margin note. Notice that there isn't a number preceding the note, and there is no number in the main text where this note was written.

`\sidenote[`⟨*number*⟩`][`⟨*offset*⟩`]{`*Sidenote text.*`}`

Both the ⟨*number*⟩ and ⟨*offset*⟩ arguments are optional. If you provide a ⟨*number*⟩ argument, then that number will be used as the sidenote number. It will change of the number of the current sidenote only and will not affect the numbering sequence of subsequent sidenotes.
Sometimes a sidenote may run over the top of other text or graphics in the margin space. If this happens, you can adjust the vertical position of the sidenote by providing a dimension in the ⟨*offset*⟩ argument. Some examples of valid dimensions are:

```
1.0in     2.54cm     254mm     6\baselineskip
```

If the dimension is positive it will push the sidenote down the page; if the dimension is negative, it will move the sidenote up the page.

While both the ⟨*number*⟩ and ⟨*offset*⟩ arguments are optional, they must be provided in order. To adjust the vertical position of the sidenote while leaving the sidenote number alone, use the following syntax:

`\sidenote[]`[⟨*offset*⟩]{*Sidenote text.*}

The empty brackets tell the `\sidenote` command to use the default sidenote number.
If you *only* want to change the sidenote number, however, you may completely omit the ⟨*offset*⟩ argument:

`\sidenote`[⟨*number*⟩]{*Sidenote text.*}

The `\marginnote` command has a similar *offset* argument:

`\marginnote`[⟨*offset*⟩]{*Margin note text.*}

7.3 References

References are placed alongside their citations as sidenotes, as well. This can be accomplished using the normal `\cite` command.[4]
The complete list of references may also be printed automatically by using the `\bibliography` command. (See the end of this document for an example.) If you do not want to print a bibliography at the end of your document, use the `\nobibliography` command in its place.
To enter multiple citations at one location,[5] you can provide a list of keys separated by commas and the same optional vertical offset argument: `\cite{Tufte2006,Tufte1990}`.

`\cite`[⟨*offset*⟩]{*bibkey1,bibkey2,...*}

[4] The first paragraph of this document includes a citation.

[5] **Tufte2006**, **Tufte1990**

7.4 Figures and Tables

Images and graphics play an integral role in Tufte's work. In addition to the standard `figure` and `tabular` environments, this style provides special figure and table environments for full-width floats.
Full page–width figures and tables may be placed in `figure*` or `table*` environments. To place figures or tables in the margin, use the `marginfigure` or `margintable` environments as follows (see figure 7.1):

```
\begin{marginfigure}
    \includegraphics{helix}
    \caption{This is a margin figure.}
    \label{fig:marginfig}
\end{marginfigure}
```

Figure 7.1: This is a margin figure. The helix is defined by $x = \cos(2\pi z)$, $y = \sin(2\pi z)$, and $z = [0, 2.7]$. The figure was drawn using Asymptote (`http://asymptote.sf.net/`).

The `marginfigure` and `margintable` environments accept an optional parameter ⟨*offset*⟩ that adjusts the vertical position of the figure or table. See the "Sidenotes" section above for examples. The specifications are:

```
\begin{marginfigure}[⟨offset⟩]
    ...
\end{marginfigure}

\begin{margintable}[⟨offset⟩]
    ...
\end{margintable}
```

Figure **??** is an example of the `figure*` environment and figure **??** is an example of the normal `figure` environment.

As with sidenotes and marginnotes, a caption may sometimes require vertical adjustment. The `\caption` command now takes a second optional argument that enables you to do this by providing a dimension ⟨*offset*⟩. You may specify the caption in any one of the following forms:

`\caption`{*long caption*}
`\caption[`*short caption*`]`{*long caption*}
`\caption[]`[⟨*offset*⟩]{*long caption*}
`\caption[`*short caption*`]`[⟨*offset*⟩]{*long caption*}

A positive ⟨*offset*⟩ will push the caption down the page. The short caption, if provided, is what appears in the list of figures/tables, otherwise the "long" caption appears there. Note that although the arguments ⟨*short caption*⟩ and ⟨*offset*⟩ are both optional, they must be provided in order. Thus, to specify an ⟨*offset*⟩ without specifying a ⟨*short caption*⟩, you must include the first set of empty brackets `[]`, which tell `\caption` to use the default "long" caption. As an example, the caption to figure **??** above was given in the form

```
\caption[Hilbert curves...][6pt]{Hilbert curves...}
```

Table 7.1 shows table created with the `booktabs` package. Notice the lack of vertical rules—they serve only to clutter the table's data.

Margin	Length
Paper width	8½ inches
Paper height	11 inches
Textblock width	6½ inches
Textblock/sidenote gutter	⅜ inches
Sidenote width	2 inches

Table 7.1: Here are the dimensions of the various margins used in the Tufte-handout class.

Occasionally LaTeX will generate an error message:

```
Error:  Too many unprocessed floats
```

LaTeX tries to place floats in the best position on the page. Until it's finished composing the page, however, it won't know where those positions are. If you have a lot of floats on a page (including sidenotes, margin notes, figures, tables, etc.), LaTeX may run out of "slots" to keep track of them and will generate the above error.

LaTeX initially allocates 18 slots for storing floats. To work around this limitation, the Tufte-LaTeX document classes provide a `\morefloats` command that will reserve more slots.

The first time `\morefloats` is called, it allocates an additional 34 slots. The second time `\morefloats` is called, it allocates another 26 slots.

The `\morefloats` command may only be used two times. Calling it a third time will generate an error message. (This is because we can't safely allocate many more floats or LaTeX will run out of memory.)

If, after using the `\morefloats` command twice, you continue to get the `Too many unprocessed floats` error, there are a couple things you can do.

The `\FloatBarrier` command will immediately process all the floats before typesetting more material. Since `\FloatBarrier` will start a new paragraph, you should place this command at the beginning or end of a paragraph.

The `\clearpage` command will also process the floats before continuing, but instead of starting a new paragraph, it will start a new page.

You can also try moving your floats around a bit: move a figure or table to the next page or reduce the number of sidenotes. (Each sidenote actually uses *two* slots.)

After the floats have placed, LaTeX will mark those slots as unused so they are available for the next page to be composed.

7.5 Captions

You may notice that the captions are sometimes misaligned. Due to the way LaTeX's float mechanism works, we can't know for sure where it decided to put a float. Therefore, the Tufte-LaTeX document classes provide commands to override the caption position.

Vertical alignment To override the vertical alignment, use the `\setfloatalignment` command inside the float environment. For example:

```
\begin{figure}[btp]
    \includegraphics{sinewave}
    \caption{This is an example of a sine wave.}
    \label{fig:sinewave}
    \setfloatalignment{b}% forces caption to be bottom-aligned
\end{figure}
```

The syntax of the `\setfloatalignment` command is:

```
\setfloatalignment{⟨pos⟩}
```

where ⟨*pos*⟩ can be either `b` for bottom-aligned captions, or `t` for top-aligned captions.

Horizontal alignment To override the horizontal alignment, use either the `\forceversofloat` or the `\forcerectofloat` command inside of the float environment. For example:

```
\begin{figure}[btp]
    \includegraphics{sinewave}
    \caption{This is an example of a sine wave.}
    \label{fig:sinewave}
    \forceversofloat% forces caption to be set to the left of the float
\end{figure}
```

The `\forceversofloat` command causes the algorithm to assume the float has been placed on a verso page—that is, a page on the left side of a two-page spread. Conversely, the `\forcerectofloat` command causes the algorithm to assume the float has been placed on a recto page—that is, a page on the right side of a two-page spread.

7.6 Full-width text blocks

In addition to the new float types, there is a `fullwidth` environment that stretches across the main text block and the sidenotes area.

```
\begin{fullwidth}
  Lorem ipsum dolor sit amet...
\end{fullwidth}
```

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

7.7 Typography

7.7.1 Typefaces

If the Palatino, Helvetica, and `Bera Mono` typefaces are installed, this style will use them automatically. Otherwise, we'll fall back on the Computer Modern typefaces.

7.7.2 Letterspacing

This document class includes two new commands and some improvements on existing commands for letterspacing.

When setting strings of ALL CAPS or SMALL CAPS, the letterspacing—that is, the spacing between the letters—should be increased slightly.[6] The `\allcaps` command has proper letterspacing for strings of FULL CAPITAL LETTERS, and the `\smallcaps` command has letterspacing for SMALL CAPITAL LETTERS. These commands will also automatically convert the case of the text to upper- or lowercase, respectively.

[6] **Bringhurst2005**

The `\textsc` command has also been redefined to include letterspacing. The case of the `\textsc` argument is left as is, however. This allows one to use both uppercase and lowercase letters: The Initial Letters Of The Words In This Sentence Are Capitalized.

7.8 Document Class Options

The `tufte-book` class is based on the LaTeX `book` document class. Therefore, you can pass any of the typical book options. There are a few options that are specific to the `tufte-book` document class, however.

The `a4paper` option will set the paper size to A4 instead of the default US letter size.

The `sfsidenotes` option will set the sidenotes and title block in a sans serif typeface instead of the default roman.

The `twoside` option will modify the running heads so that the page number is printed on the outside edge (as opposed to always printing the page number on the right-side edge in `oneside` mode).

The `symmetric` option typesets the sidenotes on the outside edge of the page. This is how books are traditionally printed, but is contrary to Tufte's book design which sets the sidenotes on the right side of the page. This option implicitly sets the `twoside` option.

The `justified` option sets alldocclsoptdef and right). The default is to set the text ragged right. The body text of Tufte's books are set ragged right. This prevents needless hyphenation and makes it easier to read the text in the slightly narrower column.

The `bidi` option loads the `bidi` package which is used with XeLaTeX to typeset bi-directional text. Since the `bidi` package needs to be loaded before the sidenotes and cite commands are defined, it can't be loaded in the document preamble.

The `debug` option causes the Tufte-LaTeX classes to output debug information to the log file which is useful in troubleshooting bugs. It will also cause the graphics to be replaced by outlines.

The `nofonts` option prevents the Tufte-LaTeX classes from automatically loading the Palatino and Helvetica typefaces. You should use this option if you wish to load your own fonts. If you're using XeLaTeX, this option is implied (*i. e.*, the Palatino and Helvetica fonts aren't loaded if you use XeLaTeX).

The `nols` option inhibits the letterspacing code. The Tufte-LaTeX classes try to load the appropriate letterspacing package (either pdfTeX's `letterspace` package

or the `soul` package). If you're using XeLaTeX with `fontenc`, however, you should configure your own letterspacing.

The `notitlepage` option causes `\maketitle` to generate a title block instead of a title page. The `book` class defaults to a title page and the `handout` class defaults to the title block. There is an analogous `titlepage` option that forces `\maketitle` to generate a full title page instead of the title block.

The `notoc` option suppresses Tufte-LaTeX's custom table of contents (TOC) design. The current TOC design only shows unnumbered chapter titles; it doesn't show sections or subsections. The `notoc` option will revert to LaTeX's TOC design.

The `nohyper` option prevents the `hyperref` package from being loaded. The default is to load the `hyperref` package and use the `\title` and `\author` contents as metadata for the generated PDF.

8
Customizing Tufte-LaTeX

The Tufte-LaTeX document classes are designed to closely emulate Tufte's book design by default. However, each document is different and you may encounter situations where the default settings are insufficient. This chapter explores many of the ways you can adjust the Tufte-LaTeX document classes to better fit your needs.

8.1 File Hooks

If you create many documents using the Tufte-LaTeX classes, it's easier to store your customizations in a separate file instead of copying them into the preamble of each document. The Tufte-LaTeX classes provide three file hooks: `tufte-common-local.tex`, `tufte-book-local.tex`, and `tufte-handout-local.tex`.

`tufte-common-local.tex` If this file exists, it will be loaded by all of the Tufte-LaTeX document classes just prior to any document-class-specific code. If your customizations or code should be included in both the book and handout classes, use this file hook.

`tufte-book-local.tex` If this file exists, it will be loaded after all of the common and book-specific code has been read. If your customizations apply only to the book class, use this file hook.

`tufte-common-handout.tex` If this file exists, it will be loaded after all of the common and handout-specific code has been read. If your customizations apply only to the handout class, use this file hook.

8.2 Numbered Section Headings

While Tufte dispenses with numbered headings in his books, if you require them, they can be anabled by changing the value of the `secnumdepth` counter. From the table below, select the heading level at which numbering should stop and set the `secnumdepth` counter to that value. For example, if you want parts and chapters numbered, but don't want numbering for sections or subsections, use the command:

```
\setcounter{secnumdepth}{0}
```

The default `secnumdepth` for the Tufte-LaTeX document classes is −1.

Heading level	Value
Part (in `tufte-book`)	−1
Part (in `tufte-handout`)	0
Chapter (only in `tufte-book`)	0
Section	1
Subsection	2
Subsubsection	3
Paragraph	4
Subparagraph	5

Table 8.1: Heading levels used with the `secnumdepth` counter.

8.3 Changing the Paper Size

The Tufte-LaTeX classes currently only provide three paper sizes: A4, B5, and US letter. To specify a different paper size (and/or margins), use the `\geometrysetup` command in the preamble of your document (or one of the file hooks). The full documentation of the `\geometrysetup` command may be found in the `geometry` package documentation.[1]

[1] **pkg-geometry**

8.4 Customizing Marginal Material

Marginal material includes sidenotes, citations, margin notes, and captions. Normally, the justification of the marginal material follows the justification of the body text. If you specify the `justified` document class option, all of the margin material will be fully justified as well. If you don't specify the `justified` option, then the marginal material will be set ragged right.

You can set the justification of the marginal material separately from the body text using the following document class options: `sidenote`, `marginnote`, `caption`, `citation`, and `marginals`. Each option refers to its obviously corresponding marginal material type. The `marginals` option simultaneously sets the justification on all four marginal material types.

Each of the document class options takes one of five justification types:

`justified` Fully justifies the text (sets it flush left and right).

`raggedleft` Sets the text ragged left, regardless of which page it falls on.

`raggedright` Sets the text ragged right, regardless of which page it falls on.

`raggedouter` Sets the text ragged left if it falls on the left-hand (verso) page of the spread and otherwise sets it ragged right. This is useful in conjunction with the `symmetric` document class option.

`auto` If the `justified` document class option was specified, then set the text fully justified; otherwise the text is set ragged right. This is the default justification option if one is not explicitly specified.

For example,

```
\documentclass[symmetric,justified,marginals=raggedouter]{tufte-book}
```

will set the body text of the document to be fully justified and all of the margin material (sidenotes, margin notes, captions, and citations) to be flush against the body text with ragged outer edges.

THE FONT AND STYLE of the marginal material may also be modified using the following commands:

```
\setsidenotefont{⟨font commands⟩}
\setcaptionfont{⟨font commands⟩}
\setmarginnotefont{⟨font commands⟩}
\setcitationfont{⟨font commands⟩}
```

The `\setsidenotefont` sets the font and style for sidenotes, the `\setcaptionfont` for captions, the `\setmarginnotefont` for margin notes, and the `\setcitationfont` for citations. The ⟨*font commands*⟩ can contain font size changes (e.g., `\footnotesize`, `\Huge`, etc.), font style changes (e.g., `\sffamily`, `\ttfamily`, `\itshape`, etc.), color changes (e.g., `\color{blue}`), and many other adjustments.

If, for example, you wanted the captions to be set in italic sans serif, you could use:

```
\setcaptionfont{\itshape\sffamily}
```

9
Compatibility Issues

When switching an existing document from one document class to a Tufte-LaTeX document class, a few changes to the document may have to be made.

9.1 Converting from `article` to `tufte-handout`

The following `article` class options are unsupported: `10pt`, `11pt`, `12pt`, `a5paper`, `b5paper`, `executivepaper`, `legalpaper`, `landscape`, `onecolumn`, and `twocolumn`. The following headings are not supported: `\subsubsection` and `\subparagraph`.

9.2 Converting from `book` to `tufte-book`

The following `report` class options are unsupported: `10pt`, `11pt`, `12pt`, `a5paper`, `b5paper`, `executivepaper`, `legalpaper`, `landscape`, `onecolumn`, and `twocolumn`. The following headings are not supported: `\subsubsection` and `\subparagraph`.

10
Troubleshooting and Support

10.1 Tufte-LaTeX Website

The website for the Tufte-LaTeX packages is located at http://code.google.com/p/tufte-latex/. On our website, you'll find links to our SVN repository, mailing lists, bug tracker, and documentation.

10.2 Tufte-LaTeX Mailing Lists

There are two mailing lists for the Tufte-LaTeX project:

Discussion list The `tufte-latex` discussion list is for asking questions, getting assistance with problems, and help with troubleshooting. Release announcements are also posted to this list. You can subscribe to the `tufte-latex` discussion list at http://groups.google.com/group/tufte-latex.

Commits list The `tufte-latex-commits` list is a read-only mailing list. A message is sent to the list any time the Tufte-LaTeX code has been updated. If you'd like to keep up with the latest code developments, you may subscribe to this list. You can subscribe to the `tufte-latex-commits` mailing list at http://groups.google.com/group/tufte-latex-commits.

10.3 Getting Help

If you've encountered a problem with one of the Tufte-LaTeX document classes, have a question, or would like to report a bug, please send an email to our mailing list or visit our website.
To help us troubleshoot the problem more quickly, please try to compile your document using the `debug` class option and send the generated `.log` file to the mailing list with a brief description of the problem.

10.4 Errors, Warnings, and Informational Messages

The following is a list of all of the errors, warnings, and other messages generated by the Tufte-LaTeX classes and a brief description of their meanings.

`Error: \subparagraph is undefined by this class.`

The `\subparagraph` command is not defined in the Tufte-LaTeX document classes. If you'd like to use the `\subparagraph` command, you'll need to redefine it yourself. See the "Headings" section on page 61 for a description of the heading styles availaboe in the Tufte-LaTeX document classes.

`Error: \subsubsection is undefined by this class.`

The `\subsubsection` command is not defined in the Tufte-LaTeX document classes. If you'd like to use the `\subsubsection` command, you'll need to redefine

it yourself. See the "Headings" section on page 61 for a description of the heading styles availaboe in the Tufte-LaTeX document classes.

```
Error: You may only call \morefloats twice. See the
       Tufte-LaTeX documentation for other workarounds.
```

LaTeX allocates 18 slots for storing floats. The first time `\morefloats` is called, it allocates an additional 34 slots. The second time `\morefloats` is called, it allocates another 26 slots.
The `\morefloats` command may only be called two times. Calling it a third time will generate this error message. See page 63 for more information.

```
Warning: Option '⟨class option⟩' is not supported -- ignoring option.
```

This warning appears when you've tried to use ⟨*class option*⟩ with a Tufte-LaTeX document class, but ⟨*class option*⟩ isn't supported by the Tufte-LaTeX document class. In this situation, ⟨*class option*⟩ is ignored.

```
Info: The 'symmetric' option implies 'twoside'
```

You specified the `symmetric` document class option. This option automatically forces the `twoside` option as well. See page 65 for more information on the `symmetric` class option.

10.5 Package Dependencies

The following is a list of packages that the Tufte-LaTeX document classes rely upon. Packages marked with an asterisk are optional.

- xifthen
- ifpdf*
- ifxetex*
- hyperref
- geometry
- ragged2e
- chngpage *or* changepage
- paralist
- textcase
- soul*
- letterspace*
- setspace
- natbib *and* bibentry
- optparams
- placeins
- mathpazo*
- helvet*
- fontenc
- beramono*
- fancyhdr
- xcolor
- textcomp
- titlesec
- titletoc