Deep generative clustering of spatial wafer patterns

An unsupervised machine learning approach within the framework of Industry 4.0

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

eingereicht von

Peter Tulala, BSc
Matrikelnummer 1528202

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Mitwirkung: Projektass. Hamidreza Mahyar, PhD

Wien, 25. April 2019

Peter Tulala Radu Grosu

Deep generative clustering of spatial wafer patterns

An unsupervised machine learning approach within the framework of Industry 4.0

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computational Intelligence

by

Peter Tulala, BSc
Registration Number 1528202

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Assistance: Projektass. Hamidreza Mahyar, PhD

Vienna, 25th April, 2019

Peter Tulala   Radu Grosu

Erklärung zur Verfassung der Arbeit

Peter Tulala, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. April 2019

Peter Tulala

Danksagung

Ich möchte meinem Betreuer, Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu, meine aufrichtige Dankbarkeit ausdrücken für die Erlaubnis, dieses Papier zu schreiben, für seine Führung durch den gesamten Prozess sowie für viele interessante Einblicke und immense Unterstützung. Ein besonderer Dank gilt meinem Forscherkollege Projektass. Hamidreza Mahyar PhD., der mir wichtige Hilfestellung bei der Bewertung der Arbeit und Ratschläge zu vielen Punkten gegeben hat. Ich möchte mich auch bei meinem Kollegen Paul Stelzhammer für die Hilfe bei einer deutschen Übersetzung des Abstracts bedanken.

Acknowledgements

I would like to express sincere gratitude to my advisor, Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Gros, for allowing me to write this paper and guiding me through the whole process while providing many interesting insights and immense support. A special gratitude belongs to a fellow researcher Projektass. Hamidreza Mahyar PhD. who has provided me significant help with evaluating the work and advising on many technical aspects of the thesis. I would like to thank also my colleague Paul Stelzhammer for help with a German translation of the abstract.

Kurzfassung

Automatisierung der Produktion, maschinelles Lernen, Big Data, Internet der Dinge und computerunterstütztes Entscheidungsfinden sind Schlüsselfaktoren der vierten industriellen Revolution (auch Industrie 4.0 genannt). Hoch automatisierte Industriezweige, wie die Halbleiter herstellende Industrie, werden zur Grenze von Industrie 4.0. Obwohl die hochautomatisierte Halbleiterproduktion in Reinraum-Umgebungen stattfindet, ist der in der Komplexität steigende und aus hunderten von Schritten bestehende Herstellungsprozess, trotzdem anfällig für gewöhnliche Produktionsfehler. Um diese Fehler zu erkennen und ihnen vorzubeugen werden regelmäßig elektromechanische Messungen von jedem Wafer nach den verschiedenen Schritten in der Produktion genommen. Es wird angenommen, dass die Schritte welche Fertigungsfehler verursachen in einer frühen Phase der Produktionskette in charakteristischen Mustern der Wafermap-Messdaten erkannt werden. Basierend auf den erkannten Mustern kann eine automatische ausgleichende Aktion gesetzt werden um die Herstellungskosten so gering wie möglich zu halten und Engpässen vorzubeugen. Das Ziel dieser Arbeit ist es einen Algorithmus zu entwickeln der automatisch Muster in den Wafermap-Messdaten erkennt und gruppiert und dabei Methode des unüberwachten Lernen verwendet. Die unüberwachte Art solch eines Algorithmus beseitigt den Bedarf eines Experten auf dem jeweiligen Gebiet, welcher ansonsten manuell definieren müsste welche möglichen Muster in den Daten auftreten. Der erste Teil der Arbeit beschreibt die vorverarbeitenden Schritte, um den Messdatensatz zu bereinigen und zu normalisieren. Der bereinigte Datensatz wird dann verwendet, um ein generisches Model zu trainieren, welches die charakteristischsten Eigenschaften lernt und die Dimension der Daten reduziert. Spezielles Augenmerk gilt zwei Algorithmen - Variation Autoencoder (VAE) und Generative Adversarial Network (GAN). Der letzte Teil der Arbeit beschreibt zwei einfache Clustering Methoden welche die Features je nach Ähnlichkeitsmetrik in eindeutige Cluster gruppieren. Das Fazit ist, dass generische Modelle zur Feature Extrahierung in der Halbleiter Fertigung nützlich sein können und in manchen Fällen traditionelle Modelle übertreffen.

Abstract

Computerization of manufacturing, machine learning, big data, Internet of Things (IoT) and automated decision-making are becoming the key driving forces of the fourth industrial revolution (also referred to as *Industry 4.0*). Highly automated industries, such as semiconductor manufacturing industry, are becoming the frontiers of Industry 4.0. Despite the fact that the highly automated semiconductor production takes place in cleanroom environments, the increasingly complex manufacturing processes driven by hundreds of production steps are still susceptible to random production errors. In order to detect and prevent these errors, electromechanical measurements are regularly taken from each wafer in production after some steps. It is assumed that steps that are causing manufacturing disturbances can be identified in an early stage of production chain by recognizing characteristic patterns in the wafermap measurements data. Based on the recognized patterns, an automatic corrective action can be taken in order to minimize manufacturing cost and prevent bottlenecks. The aim of this work is to develop an algorithm to automatically recognize and cluster patterns in wafermap measurements data in an unsupervised manner. The unsupervised nature of such algorithm eliminates the need for a domain expert that would otherwise had to manually define all possible patterns occurring in the data. The first part of this work describes preprocessing steps to cleanse and normalize the measurements dataset. The cleansed dataset is then used to train a generative model that learns the most characteristic features and reduces the dimensionality of the data. A particular focus is given to two algorithms – *Variational Autoencoder (VAE)* and *Generative Adversarial Network (GAN)*. The last part of the work discusses two simple clustering methods that group the extracted features into distinct clusters according to a similarity metric. It is concluded, that generative models could be useful for feature extraction in semiconductor manufacturing domain and in some cases even outperform more traditional discriminative models.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Motivation 1
1.2 Problem statement & scope of the work 2
1.3 State of the art 3
1.4 Methodological approach 4
1.5 Peer review 5
1.6 Structure of the work 5

2 Data preprocessing **7**
2.1 Wafer clipping mask 7
2.2 Removing outliers 13
2.3 Imputing missing values 16
2.4 Normalization 17
2.5 Denoising 18
2.6 Results 18

3 Machine learning preliminaries **19**
3.1 Learning paradigms 19
3.2 Parametric models 22
3.3 Convergence of random variables 23
3.4 Empirical risk minimization 24
3.5 Bias-variance trade-off 25
3.6 Point estimation 26
3.7 Maximum Likelihood Estimation (MLE) 29
3.8 Maximum A Posteriori (MAP) estimation 30
3.9 Information entropy 31
3.10 Kullback–Leibler (KL) divergence 34
3.11 Stochastic Gradient Descent (SGD) 35

3.12 Artificial Neural Network (ANN) 39
3.13 Regularization 42

4 Feature extraction **47**
4.1 Variational Autoencoder 48
4.2 Generative Adversarial Network (GAN) 52

5 Clustering **63**
5.1 *k*-means clustering 63
5.2 Hierarchical agglomerative clustering 64
5.3 Silhouette coefficient 67
5.4 Experiments 67

6 Conclusion **71**

List of Figures **73**

List of Tables **75**

List of Algorithms **77**

Bibliography **81**

CHAPTER 1

Introduction

Many disruptive innovations in recent years are propelling existing industries to adapt their business and manufacturing processes towards new paradigms in order to sustain competitiveness in increasingly global markets. As a response for the incoming changes, German government developed a strategy promoting computerization of manufacturing, collectively called "Industry 4.0" [KHHW13]. Value proposition of this strategy includes increased production agility, faster reactions to market requirements, automated decision-making, smart maintenance and manufacturing processes management close to real time. Goals of this diploma thesis are in line with the objectives of SemI40 consortium (`http://www.semi40.eu`) coordinated by Infineon Technologies whose strategic focus lies in the adoption of recent ICT innovations (such as machine learning, big data and automated decision-making) to strengthen sustainable competitiveness of European semiconductor production.

1.1 Motivation

Semiconductor manufacturing is a complex process consisting of several hundreds processesing steps in a cleanroom environment facility. These steps are characterized by certain amount of process deviations. Automated detection of these production issues followed by an automated root cause analysis has a potential to increase effectiveness of semiconductor production. Manufacturing defects exhibit typical patterns in measured wafer test data, e.g. rings, spots, repetitive textures or scratches. Recognizing these patterns is an essential step for finding the root cause of production issues and eventually to take automatic corrective actions to eliminate the risks to acceptable levels.

1.2 Problem statement & scope of the work

The aim of this work is to develop and train a statistical model capable to cluster given raw wafer test data into groups according to similarities in their spatial wafer patterns. It is assumed that such model could be used for making predictions about the quality of the wafer under construction and in case of anomalies to infer the interplay of the various processes responsible for these abnormalities. The statistical model developed in this work could also serve as a basis for an automatic decision-making and corrective calibration of factory equipment such that the production abnormalities are resolved on the fly without human intervention.

A crucial step before applying machine learning methods is to preprocess raw wafer data. This step involves removing outliers, imputing missing measurements, noise removal and data normalization. The result of the data preprocessing step is a clean dataset that can be used for subsequent feature extraction and clustering tasks.

The following two generative approaches will be considered in order to extract features from the wafers:

- **Variational Autoencoder (VAE)** [KHHW13] that formalizes the feature extraction problem as a probabilistic graphical model with the objective to maximize the evidence lower bound (ELBO) on the log likelihood of the data.
- **Generative Adversarial Network (GAN)** [GPAM+14] that formalizes the feature extraction problem in a game-theoretic approach where two different networks (called *generator* and *discriminator*) compete against each other in order to reach Nash equilibrium. Many extensions to GAN have been recently proposed, namely InfoGAN [CCD+16] is especially useful in context of this diploma thesis to learn disentangled representations in a completely unsupervised manner. GANs are notoriously difficult to train, hence methods for training stabilization are likely to be needed, for example the improvements discussed in [SGZ+16], Wasserstein GAN [ACB17a] or Improved Training of Wasserstein GAN [GAA+17a].

The extracted features can be plugged in into some traditional clustering algorithms like *k-means* or *agglomerative hierarchical clustering* for which many well-optimized implementations already exist. The clustering task groups wafers into clusters according to patterns they exhibit. The main outcome of this work is a statistical model for recognizing patterns from given raw wafer test data.

Finally, the performance of different feature extraction methods will be compared with other dimensionality reduction algorithms like *Principal Component Analysis (PCA)*, *Non-negative Matrix Factorization (NMF)*, *Independent Component Analysis (ICA)* or *Singular Value Decomposition (SVD)* in terms of clustering performance measured by *Silhouette score*. This work demonstrates that deep generative modeling is a suitable

approach for extracting the most characteristic features from sensory wafer data and in various aspects outperforms traditional unsupervised dimensionality reduction techniques.

It is assumed, that such statistical model could lead to a deeper understanding of the relation between electromechanical parameters of a wafer measured after several processing steps and individual unit processes. Combining information from different data sources with advanced data analysis methods may be used for automated decision making and change in the equipment parameters has a potential to improve the process stability. An overview of tasks that are in scope of this diploma thesis is shown in Figure 1.1.

The model proposed in this diploma thesis has been trained using Tensorflow framework and the program for training this model given wafer test data has been delivered as a Docker container that can be deployed and executed on a personal computer or a server.

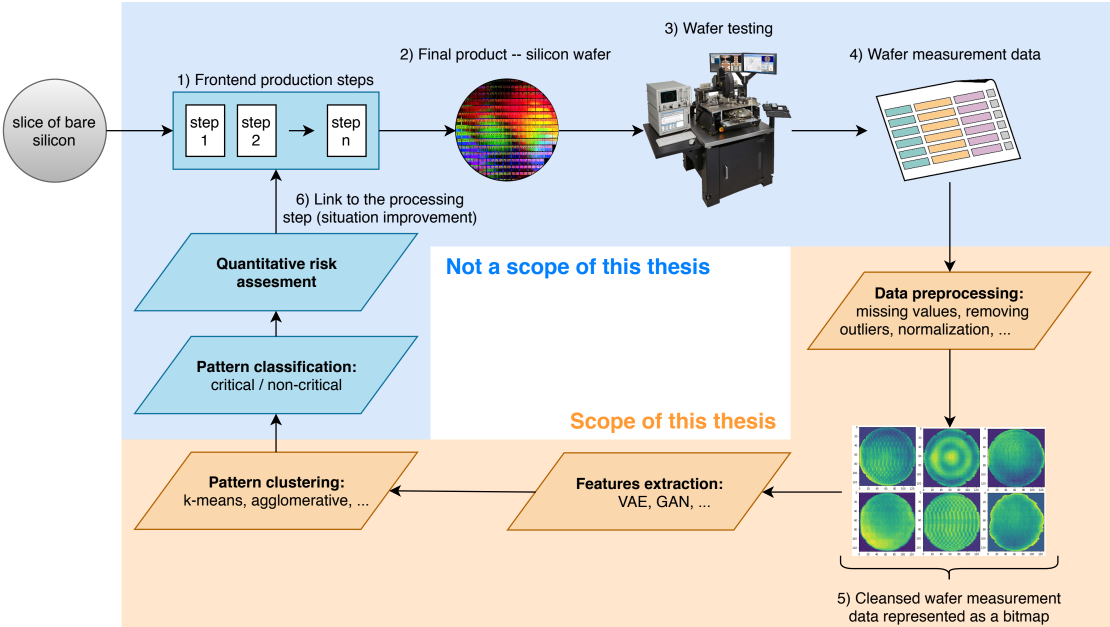


Figure 1.1: High-level overview of an automated improvement of a wafer production process. Tasks considered in this diploma thesis are depicted in orange color. The outcome of the work is a statistical model that clusters given raw electromechanical measurement data of produced wafers in order to be used for a subsequent automated corrective adjustments and elimination of risks. This will lead to an optimized production sequence of different products according to product requirements and specifications.

1.3 State of the art

[Ser83] Several methods based on traditional image processing approach for detecting patterns in wafer test data have been proposed [BS95] [Duv99].

More robust methods utilize some machine learning techniques to recognize more complex patterns in wafer data. There have been proposed methods based on supervised training of mixture models [LCC96], neural networks [CLYYDY] or support-vector machines [CT09]. Although these methods are powerful, their supervised nature still requires a human expert to craft a training dataset with manually labeled data.

The apparent advantage of unsupervised pattern detection approach lies in an elimination of subjective factors from pattern recognition task, which in turn reduces costs and number of classification errors. The hidden dependencies between different types of wafer defects are detected automatically without intervention of human expert which enables detection of patterns that were unknown or overlooked before. This type of approach includes self-organizing neural networks [CLCJ09], self-organizing maps [PNM+05] as well as techniques based on dimensionality reduction like diffusion maps [MC14] or discriminant analysis [YL16].

In this diploma thesis, there are proposed two different unsupervised methods for generative clustering of wafermap patterns based on two approaches – Variational Autoencoder (VAE) [KHHW13] and Generative Adversarial Network (GAN) [GPAM+14] and its extensions such as InfoGAN [CDH+16] and Wasserstein GAN [ACB17b].

1.4 Methodological approach

The methodological approach consists of the following steps:

1. **Literature review**
   Explore strategic objectives of SemI40 consortium and existing research papers in semiconductor production industry. Explore research papers related to machine learning, especially different pre-processing methods, statistics, information theory, generative models and clustering techniques.

2. **Implementation**
   Use available raw wafer test data to develop a statistical model for feature extraction and clustering of wafers. The overview of all tasks in scope of this work are depicted in Figure 1.1, the tasks implemented as a part of this diploma thesis include:

   a) Data pre-processing
      i. Handling missing values
      ii. Removing of outliers
      iii. Noise removal and smoothing
      iv. Normalization

   b) Feature extraction with deep generative techniques (VAE and GAN).

   c) Clustering of patterns in the wafer data. It is assumed that these patterns can be further classified and used to find hidden links between production

issues and specific processing steps. Try different types of clustering methods (k-means, hierarchical clustering, etc.).

3. **Draw conclusion**
   Compare the results of methods used in implementation and compare them with existing methods. Compare VAE and GAN approach with traditional feature extractions methods like PCA, SVD or ICA. Measure the clustering performance with Silhouette score and by a visual inspection of the feature space.

1.5 Peer review

Some parts of this work has been peer-reviewed in the following publications:

- P. Tulala, H. Mahyar, E. Ghalebi and R. Grosu. Unsupervised Wafermap Patterns Clustering via Variational Autoencoders. *International Joint Conference on Neural Networks (IJCNN).* Rio de Janeiro, Brazil, July 8-13, 2018.
- H. Mahyar, E. Ghalebi, P. Tulala and R. Grosu. Generative Adversarial Networks for Clustering Semiconductor Wafer Maps. *Workshop on ML for Systems at NeurIPS.* Montreal, Canada, December 3-8, 2018.

1.6 Structure of the work

The content of this diploma thesis is divided into six chapters:

1. **Introduction** – this chapter.
2. **Data preprocessing** (theory + experiments) – describes steps how to transform raw wafer measurements data into a cleansed dataset that can be used as an input for training of a machine learning model.
3. **Machine learning preliminaries** (only theory) – methodological part of the work. Describes basic machine learning concepts that are used in this work.
4. **Feature extraction** (theory + experiments) – the main and the most challenging part of this work. Describes two feature extraction methods based on generative modeling, namely *Variational Autoencoder (VAE)* and *Generative Adversatial Network (GAN).*
5. **Clustering** (theory + experiments) – describes two simple methods of grouping the extracted features according to a similarity metric.
6. **Conclusion**.

CHAPTER 2

Data preprocessing

Wafer measurements data are stored in CSV files, where each line of the file represents a single chip. Position of the chip within a wafer is stored as a tuple with coordinates and individual test values are stored in corresponding columns as floating point numbers. We have found that treating each wafer measurement as a bitmap is more suitable for the purpose of finding spatial patterns in the wafer test data.

Data preprocessing is a crucial step addressing several data quality issues before applying the machine learning algorithm – especially finding and removing outliers, imputing missing measurements and data normalization. The result of data preprocessing step is a clean dataset that can be used for further feature extraction and classification tasks (shown in Figure 2.10). The overall procedure is depicted in Figure 2.1.

2.1 Wafer clipping mask

Wafers maps have irregular shape with some missing values (holes) within the wafer area caused during the measurement of test data. This step creates a clipping mask of the wafer without holes, so that the missing values can be addressed within the masked area in the data imputation step discussed in Section 2.3.

The first step is to binarize the wafer by replacing all present values with 1 and all missing values with 0. Mathematical morphology approach is then used to close small holes withing the wafer.

2.1.1 Mathematical morphology

Mathematical morphology is a set theory approach for image processing, providing a collection of mathematical tools for manipulating with geometrical structure of images. In *binary morphology*, all images are assumed to be grayscale and are represented as

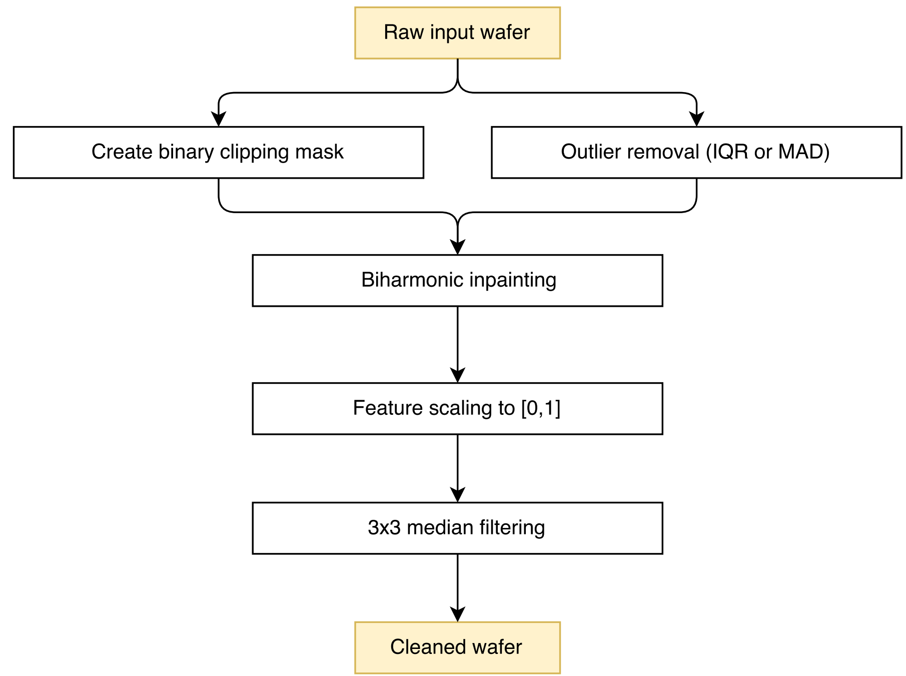


Figure 2.1: Overall wafer preprocessing procedure.

a subset of 2-dimensional Euclidean space $\mathbb{R}^2$. There have also been proposed further extensions for grayscale images of higher dimensions (where images are represented as a function mapping: $\mathbb{R}^d \to \mathbb{R} \cup \{-\infty, \infty\}$) or generalizations based on lattice theory for manipulating with morphological structures like color images or video sequences [Ser83]. In this section, only binary morphology is considered.

Morphological operators for binary image processing

In binary morphology, images are represented as sets from R^2. All operations in binary morphology are a result of standard set theory operations and two basic morphological operators – *erosion* and *dilation* (defined later):

A	the image
A^C	complement (inversed image)
$A \cap B$	intersection if two images A and B
$A \cup B$	union if two images A and B
$A - B = A \cap B^C$	difference between images A and B
$A \oplus S$	dilation of image A by structure S

$A \ominus S$ erosion of image A by structure S

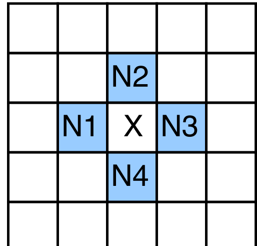
Additional morphological operations can be defined by combining the operations above.

Structuring elements

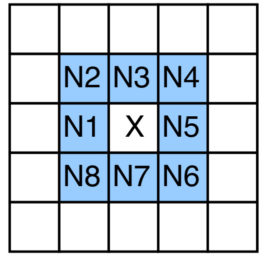
The right-hand-side operand S of basic morphological operators is also called the *structuring element*. The *origin* of the structuring element is the element $X = (0, 0)$.

In discrete case, structuring elements are defined on integer grid $\mathbb{Z}^d$. For example for $d = 2$, the commonly used structural elements are 4-neighborhood S_4 and 8-connection S_8 (Equation 2.1 and 2.2, depicted in Figure 2.2):

$$S_4 = \{(-1, 0), (0, 1), (1, 0), (0, -1))\} \tag{2.1}$$
$$S_8 = \{(-1, 0), (-1, 1), (0, 1), (1, 1), (1, 0), (1, -1), (0, -1), (-1, -1)\} \tag{2.2}$$



(a) 4-neighborhood structuring element S_4.



(b) 8-neighborhood structuring element S_8.

Figure 2.2: Commonly used discrete structural elements with the origin $X = (0, 0)$.

Another commonly used structuring element is an open disc of radius r, centered at the origin.

Dilation and erosion

In binary morphology, the dilation resp. erosion are translation invariant operations closely related to Minkowski addition resp. Minkowski difference of vectors in Euclidean space. Informally, the dilation grows all features in a binary image, while the erosion shrinks them.

Definition 1. *Let A be a set. The **reflection** of the set A is given by $\hat{A} = \{a \mid -a \in A\}$. The **translation** of the set A by z is given by $(A)_z = \{a + z \mid a \in A\}$.*

Definition 2. *Let A be a binary image and S a structuring element. The dilation of A by S is given by:*

$$A \oplus S = \bigcup_{s \in S} (A)_s = \{z \mid ((\hat{S})_z \cap A) \neq \emptyset\}$$

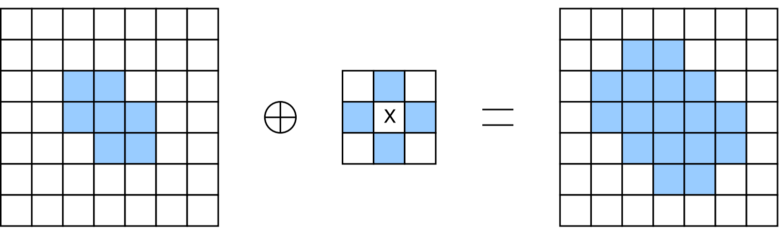


Figure 2.3: Example of dilation with a 4-neighborhood structural element.

Clearly, $A \oplus \{0\} = A$ and $A \oplus \emptyset = \emptyset$. Some other properties of dilation:

- translation invariance: $(A)_z \oplus B = (A \oplus B)_z$
- commutativity: $A \oplus B = B \oplus A$
- associativity: $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
- extensivity: $0 \in B \implies A \subseteq A \oplus B$
- increasing: $A \subseteq B \implies A \oplus C \subseteq B \oplus C$
- linearity: $(A \oplus B) \oplus C = (A \oplus C) \oplus (B \oplus C)$

Definition 3. *Let A be a binary image and S a structuring element. The erosion of A by S is given by:*

$$A \ominus S = \bigcap_{s \in S} (A)_{-s} = \{z \mid (S)_z \subseteq A\}$$

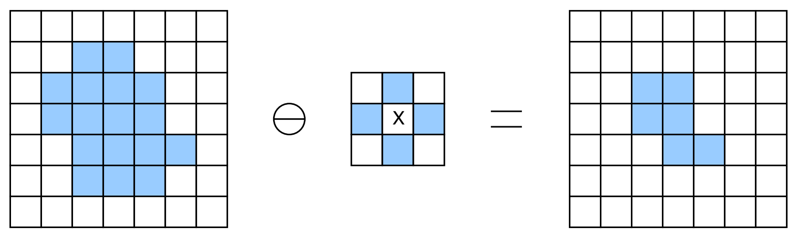


Figure 2.4: Example of erosion with a 4-neighborhood structural element.

Some properties of erosion:

- translation invariance: $(A)_z \ominus B = (A \ominus B)_z$
- not commutative: $A \ominus B \neq B \ominus A$
- not associative: $A \ominus (B \ominus C) \neq (A \ominus B) \ominus C$ (but [1])
- extensivity: $0 \in B \implies A \ominus B \subseteq A$
- increasing: $A \subseteq B \implies A \ominus C \subseteq B \ominus C$
- linearity: $(A \ominus B) \ominus C = (A \ominus C) \ominus (B \ominus C)$

The dilation can be implemented as a processes performed by laying the origin of the structuring element S on the image A and sliding it across all pixels of the image, which leads to Algorithm 2.1.

Algorithm 2.1: Dilation algorithm of image A by structure S.

Input: Input image A, structuring element S

Output: $A \oplus S$

```
1 I := ∅                          ▷ Empty image
2 for ∀s ∈ S do
3     A_s := {a + s | a ∈ A}      ▷ Shifted image
4     I := I ∪ A_s
5 end
6 return I
```

Lemma 1. *Dilation and erosion satisfy the duality:* $(A \ominus B)^C = A^C \oplus \hat{B}$.

Proof. First we prove $A \cap B^C \iff A \subseteq B$:

$$\begin{aligned} A \cap B^C = \emptyset &\iff \forall x \in A : x \notin B^C \\ &\iff \forall x \in A : x \in (B^C)^C \\ &\iff \forall x \in A : x \in B \\ &\iff A \subseteq B \end{aligned}$$

[1] Although erosion is not associative, it holds that $A \ominus (B \oplus C) = (A \ominus B) \ominus C$. The proof follows from the duality of erosion and dilation (Lemma 1): $(A \ominus B) \ominus C = ((A \ominus B)^C \oplus \hat{C})^C = (A^C \oplus \hat{B} \oplus \hat{C})^C = (A^C \oplus (B \hat{\oplus} C))^C = A \ominus (B \oplus C)$. □

Then we have:

$$\begin{aligned}(A \ominus B)^C &= \{z \mid (B)_z \subseteq A\}^C \\ &= \{z \mid (B)_z \cap A^C = \emptyset\}^C \\ &= \{z \mid (B)_z \cap A^C \neq \emptyset\} \\ &= A^C \oplus \hat{B}\end{aligned}$$

□

Corollary 1. *Erosion can be implemented using Algorithm 2.1 as* $A \ominus S = (A^C \oplus S)^C$.

(a) Original image A. (b) Dilated image $A \oplus S_8$. (c) Eroded image $A \ominus S_8$.

Figure 2.5: Example of dilation and erosion on a binary image.

Hole filling

Small holes can be closed using the closing operator:

$$A \bullet B = (A \oplus B) \ominus B \tag{2.3}$$

The closing operator in Equation 2.3 works well for closing very small holes with an area of just a few pixles. However, bigger holes may sometimes also occur in the wafer dataset, hence a more robust method must be used. As shown in Algorithm 2.2, repeated dilation of the outer area of the wafer can be used to close all holes in the wafer. The comparison of these two approaches is depicted in Figure 2.6.

Algorithm 2.2: Region filling algorithm for binary bitmap of size $m \times n$ represented by the set of vectors A.

Input: Input image A

Output: Image with closed holes

1 $T := \{(a, b) \mid a \notin [0, m], b \notin [0, n]\}$ ▷ Boundary of image

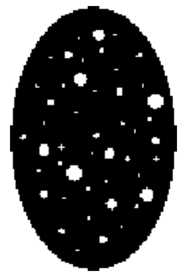
2 $S_4 :=$ 4-neighborhood structuring element

3 **repeat**

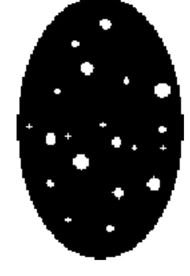
4 | $T := (T \oplus S_4) \cap A^C$

5 **until** *convergence*;

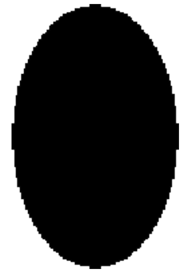
6 **return** T^C



(a) Wafer mask with holes.



(b) Closing operation.



(c) Region filling algorithm.

Figure 2.6: Wafer mask with holes of different size. Closing operation closes only very small holes, bigger holes must be iteratively closed with the hole filling algorithm.

2.2 Removing outliers

Real-world data are contaminated with measurement errors. Although random and systematic measurement errors caused by physical limitations of manufacturing devices lower the quality of recorded data, they are usually within certain range and do not cause major problems when building a predictive model. However, occasional huge inaccuracies or malfunctions in measurement process can introduce erroneous values outside of a resonable range called *outliers*.

Outliers may be informally defined as observations that are too distant from our expectations. Many predictive models are based on estimating probability distribution of the data samples. Yet even a single outlier can significantly change characteristics of an estimated probability distribution. Hence, removing these outliers is often an essential step for many statistical predictive models.

The mean and standard deviation characterizing normal distribution, are especially susceptible to perturbations caused by outliers. If any data value $x_i \to \pm\infty$, then also $\overline{x} \to \pm\infty$. On the other hand, median is resistant to gross errors in up to 50% of total samples. Statistics resistant to outliers is also called *robust statistics*. Two robust outlier removal methods are presented in this section.

2.2.1 Interquartile range (IQR) method

This is a very simple method based on interquartile range, proposed in [Tuk77]. Quartiles are three points that split the dataset into four groups, each of them comprising of quarter of the data. First quartile (Q_1) corresponds to the 25th percentile, second quartile (Q_2) corresponds to the 50th percentile (median) and third quartile (Q_3) corresponds to the 75th percentile of the dataset. The interquartile range is then defined as $\mathrm{IQR} = Q_3 - Q_1$. Values outside of the following range are considered outliers:

$$[Q_1 - k \cdot \mathrm{IQR}, Q_3 - k \cdot \mathrm{IQR}] \tag{2.4}$$

where k is a chosen constant (typically $k = 1.5$).

2.2.2 Median absolute deviation (MAD) method

The main idea of this method is to use a modified Z-score to detect outliers. The Z-score is expressed in terms of mean and standard deviation, which removes the effects of scale and location from the original dataset. Given a dataset $\mathbb{X} = \{x_1, \ldots, x_n\}$, the Z-score for i-th element in the dataset is defined as follows:

$$z_i = \frac{x_i - \mu(\mathbb{X})}{\sigma(\mathbb{X})} \tag{2.5}$$

The distribution of Z-score has mean 0 and standard deviation 1, since the effects of scale and location are removed. All values "too far" from the mean are considered outliers, i.e. elements with $|z_i| > \lambda$ for some cutoff threshold λ are outliers.

Modified Z-score

However, identifying outliers with the Z-score is problematic, because as discussed before, the mean and the standard deviation themselves are highly affected by outliers. To make this method robust, Iglewicz and Hoaglin [IH93] developed a median-based outlier detection by modifying the Z-score method. The location parameter is replaced by a median and the scale parameter is estimated using a median absolute deviation (MAD) as follows:

$$\hat{\sigma}(\mathbb{X}) = k \cdot \text{MAD} \tag{2.6}$$

where k is a constant scale factor and MAD is defined as the median of the absolute deviations:

$$\text{MAD} = \text{med}(\{|x_i - \text{med}(\mathbb{X})| \mid x_i \in X\}) \tag{2.7}$$

For normal distribution, the scale factor is defined as follows:

$$k = 1/\phi^{-1}(3/4) \approx 1/0.6745 \approx 1.4826 \tag{2.8}$$

where ϕ^{-1} is the inverse of the cumulative distribution function of the normal distribution (also called percent point function). In Equation 2.8, the $\phi^{-1}(3/4)$ is the 75th percentile of the normal distribution.

The modified Z-score is then defined as follows:

$$|z'_i| = \frac{|x_i - \text{med}(\mathbb{X})|}{\hat{\sigma}(\mathbb{X})} = \frac{\phi^{-1}(3/4) \cdot |x_i - \text{med}(\mathbb{X})|}{\text{MAD}} \tag{2.9}$$

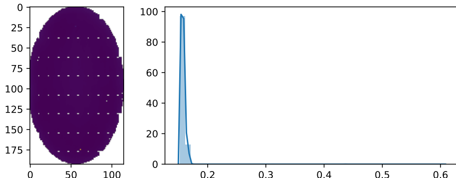
Values with $|z_i'| > \lambda$ are considered outliers, with the threshold λ typically set to a constant cutoff value $\lambda = 3.5$.

Skewed distributions

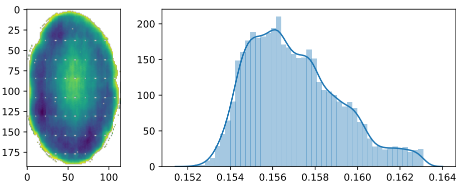
Detecting outliers using the Equation 2.9 works well for symmetric distributions, however as shown in Figure 2.7, calculating MAD independently for data points greater than (resp. less than) or equal to median using Equations 2.10 and 2.11 performs better for skewed distributions:

$$\mathrm{MAD}_L = \mathrm{med}(\{|x_i - \mathrm{med}(X)| \mid x_i \in X \leq \mathrm{med}(X)\}) \tag{2.10}$$

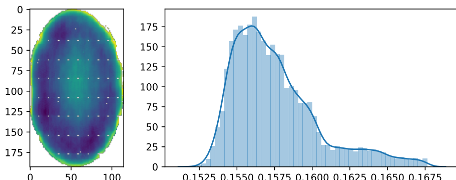
$$\mathrm{MAD}_R = \mathrm{med}(\{|x_i - \mathrm{med}(X)| \mid x_i \in X \geq \mathrm{med}(X)\}) \tag{2.11}$$



(a) The original wafer and the distribution of values in the bitmap.



(b) Outliers removed using a simple MAD – many false positives (values greater than 0.163) have also been removed due to distribution skewness.



(c) Outliers removed using a double-sided MAD – outliers greater than (resp. less than) or equal to median have been removed independently.

Figure 2.7: Outliers removal using a simple and double-sided MAD-based method.

This approach leads to an Algorithm 2.3.

Algorithm 2.3: MAD-based outlier detection algorithm.

Input: Input dataset X (pixel color values)

Output: Set of outliers

```
1  k := 1.4826                                          ▷ scale factor
2  λ := 3.5                                             ▷ cutoff threshold
3  m := med X
4  σ̂_L = k · med({|x_i − m| | x_i ∈ X ≤ m})
5  σ̂_R = k · med({|x_i − m| | x_i ∈ X ≥ m})
6  outliers := ∅
7  for ∀x ∈ X do
8  |   if |x − m|/σ̂_L ≥ λ ∨ |x − m|/σ̂_R ≥ λ then
9  |   |   outliers := outliers ∪ x
10 |   end
11 end
12 return outliers
```

Further improvements of this method are possible. For example as proposed in [RC93], by estimating the standard deviation with the median deviation of medians:

$$\hat{\sigma}'(X) = c \cdot \mathrm{med}(\{\mathrm{med}\{|x_i - x_j| \mid x_j \in X\} \mid x_i \in X\}) \tag{2.12}$$

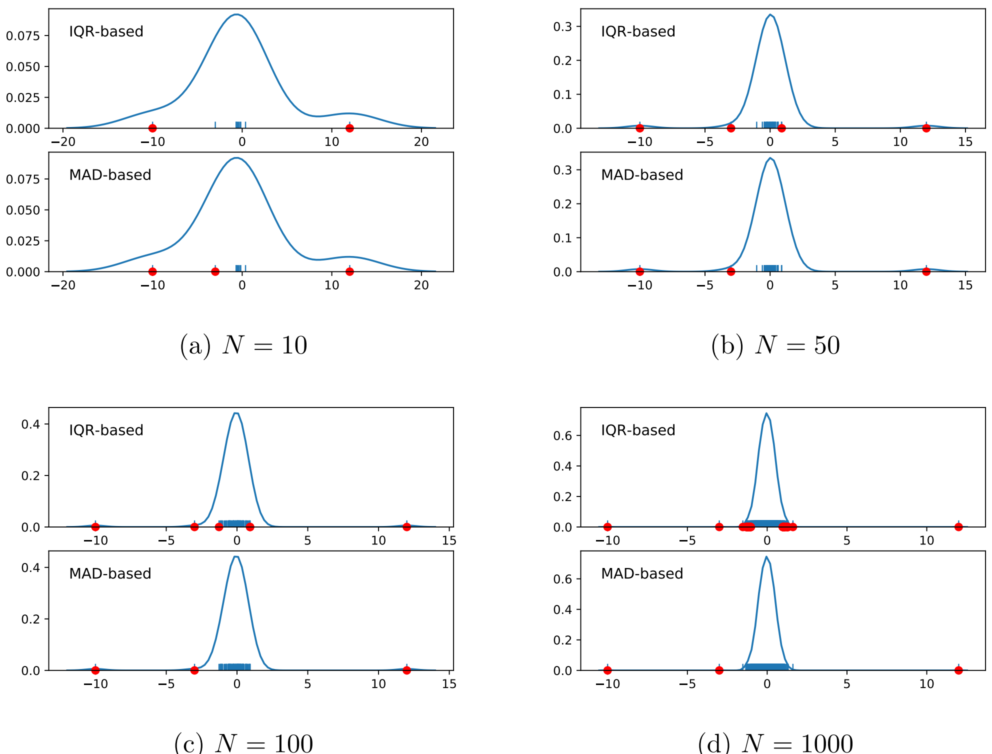
where c is a scale factor, for normal distribution $c \approx 1.1926$.

2.2.3 Comparison of IQR-based and MAD-based methods

The interquartile range is simple to calculate and the number of identified outliers depends on number of samples in the dataset, as demonstrated in Figure 2.8. MAD-based method is not affected by the number of samples. From this reason, MAD-based method has been chosen for the outlier removal task in this work.

2.3 Imputing missing values

Missing values within the area of the clipping mask created in Section 2.1 are replaced with substitute values reconstructed from information present in neighborhood of each missing region. Chui-Mhaskar inpainting algorithm [CM10] was utilized for the inpainting task. This algorithm is based on solving biharmonic equations, however a detailed description



(a) $N = 10$ (b) $N = 50$

(c) $N = 100$ (d) $N = 1000$

Figure 2.8: Comparison of IQR-based and MAD-based outlier removal methods. N points are randomly sampled from the normal distribution, with three additional outliers $-10, -3, 12$. The detected outliers are depicted with red color. The MAD-based method correctly identified all outliers without any false positives, while the outliers identified by the IQR-based method depends on the number of samples N.

is out of scope of this thesis. An existing implementation from a machine learning library `scikit-learn` [PVG+11] has been used in this preprocessing step.

2.4 Normalization

The pixel color values of wafer image have been scaled to interval $[0.05, 1]$ as follows:

$$f(x) = \frac{0.95 \cdot (x - \min)}{\max - \min} + 0.05 \tag{2.13}$$

All values outside of the clipping mask (i.e. the background around the wafer) have been replaced with constant value 0.

2.5 Denoising

The last step of the wafer preprocessing is a smoothing of the wafer by reducing stochastic noise using median filtering procedure. Each pixel of the input image f is iteratively processed by a sliding window W of size (k, l), producing smoothened image g. This overall procedure can be defined as follows [KBP07]:

$$g(x, y) = \text{med}\{f(x - k, y - l) \mid (k, l) \in W\} \tag{2.14}$$

where $g(x, y)$ represent a point at position (x, y) in output image g and $f(x - k, y - l)$ is a point in the input image f. In this work, a sliding window of size 3×3 has been used, that is $k = l = 3$.

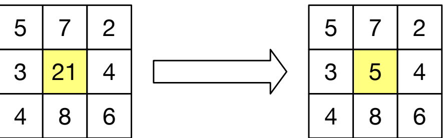
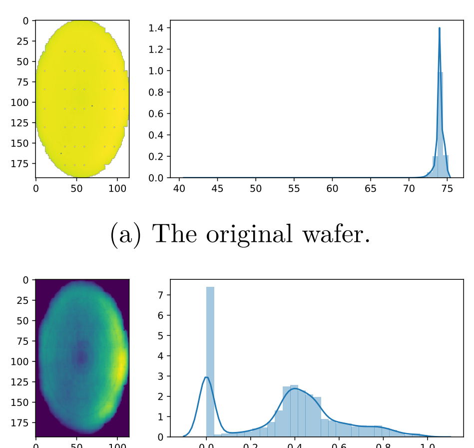


Figure 2.9: One iteration of median filtering algorithm. Current data point with value 21 is replaced with value 5, which is the median value within the 3×3 sliding window.

2.6 Results

The result of the wafer data preprocessing is demonstrated in Figure 2.10.



(a) The original wafer.

(b) The preprocessed wafer with clearly visible crescent moon pattern.

Figure 2.10: The result of wafer preprocessing.

CHAPTER 3

Machine learning preliminaries

Machine learning is an area of computer science and statistics concerned with developing knowledge discovery algorithms and techniques. Machine learning algorithms try to find patterns in observed data samples generated by some random process in order to make assumptions about previously unseen data. The process of finding these patterns in available data is called *generalization*. Generalization algorithms typically assume that the observed data has been sampled from some underlying probability distribution characterized by unknown parameters that are being estimated. According to [GBC16], machine learning is a form of applied *statistical learning theory* with increased emphasis on computational estimation of complicated functions and decreased emphasis on proving confidence intervals around these functions.

It is important to note, that all practical machine learning algorithms rely on an inductive reasoning which assumes some underlying structure of the problem under consideration. In general, no machine learning algorithm is inherently superior than other machine learning algorithms, when the performance is averaged over all possible problems. This phenomenon is referred to as the *No Free Lunch (NFL) theorem* and it has been proven in [WM97]. The goal of machine learning is not to find a universal learning algorithm, but rather to develop algorithms that perform well with data samples encountered in real-world applications.

3.1 Learning paradigms

Different machine learning algorithms can be categorized according to their distinctive characteristics. An overview of common categories of machine learning algorithms is depicted in Figure 3.1.

Depending on whether the samples in the dataset are labeled or not, we can categorize ML algorithms into the following classes:

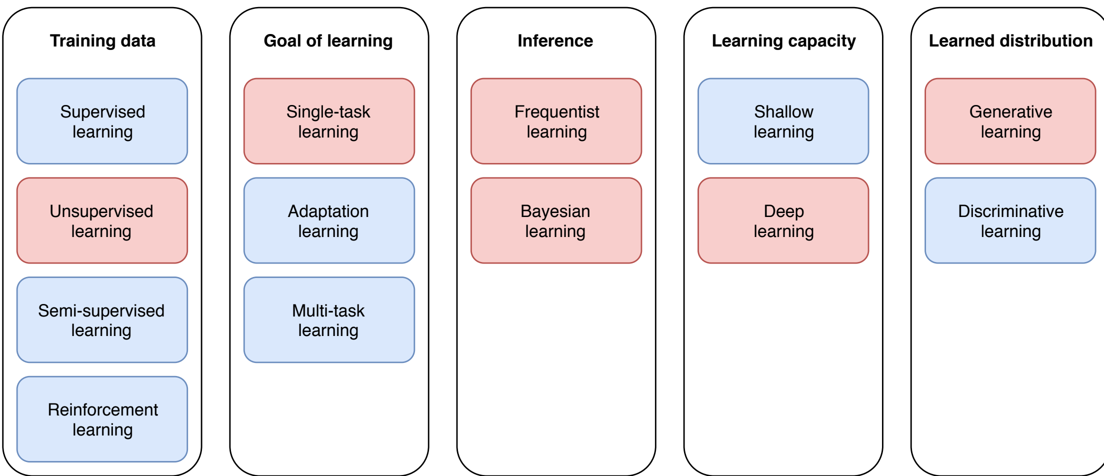


Figure 3.1: Overview of different machine learning paradigms. The main focus of this diploma thesis is depicted in red color.

- **Supervised learning** – it is assumed that all samples in training dataset are labeled. For example: *classification* (discrete labels) or *regression* (continuous labels).
- **Unsupervised learning** – it is assumed that no labels are available in the training dataset. For example: *clustering* or *anomaly detection*.
- **Semi-supervised learning** – it is assumed that only a subset of training dataset is labeled. Providing even a small amount of labels can bring a significant improvement in accuracy compared to unsupervised learning.
- **Reinforcement learning** – the dataset does not consist of individual datapoints, but rather of sequences of actions with associated rewards. The learner attempts to find the sequence of actions that maximizes the total reward. For example: learning the sequence of moves in a chess game.

Depending on the source and target distribution, we can categorize ML algorithms into the following classes:

- **Single-task learning** – the learning algorithm learns only one independent goal at the same time,
- **Multi-task learning** – the learning algorithm learns all tasks simultaneously. Shared knowledge between the tasks can result in improved learning efficiency, compared to a sequential single-task learning.

- **Domain adaptation learning** – the learning algorithm learns a target task by transferring knowledge from an already learned source task that is in some ways similar to the target task. Also called *transfer learning*.

Depending on the approach to statistical inference, we can categorize ML algorithms into the following classes:

- **Frequentist learning** – the learning algorithm is based on a frequentist statistics. This approach assumes that all data samples are drawn from a true probability distribution described by an unknown parameter θ. The learning algorithm estimates θ by a point estimator $\hat{\theta}$. The value of the true parameter θ is assumed to be fixed (not a random variable) but unknwon, while the value of the estimate $\hat{\theta}$ is a random variable. An example of frequentist estimation is discussed in Section 3.7.
- **Bayesian learning** – the learning algorithm is based on Bayesian statistics. In this approach, the parameter θ is not assumed to be fixed. The algorithm makes some initial a priori assumptions about the parameter value θ and the posterior belief about the parameter θ is then updated by applying the Bayes' theorem. When we have only limited training data available, this approach usually generalizes much better than the frequentist learning but with a higher computational cost when trained on bigger datasets. An example of Bayesian estimation is discussed in Section 3.8.

Depending on the learning capacity, we can distinguish two vaguely defined classes:

- **Shallow (surface) learning** – characterized by low learning capacity, typically useful for simple or task-specific problems.
- **Deep learning** – characterized by high learning capacity, typically use a cascade of multiple nonlinear processing units that can learn a hierarchy of concepts directly from the data.

Given the input data x and associated labels y, we can distinguish two types of learning algorithms:

- **Discriminative learning** – learns a conditional probability distribution $p(y|x)$.
- **Generative learning** – learns a joint probability distribution $p(x, y)$.

Example 1 (Generative vs. discriminative models)**.** *Generative models are more expressive but also more difficult to train than discriminative models. As an example, we can train a discriminative model $p(y|x)$ for recognizing hand-written digits: given a bitmap $x \in \mathbb{R}^n$, the discriminative model returns probabilities for labels $y \in [0, 9]$. However, if*

we train a generative model, it can be easily transformed into two discriminative models: $p(x, y) = p(y|x)p(x) = p(x|y)p(y)$. *This means that while the discriminative model can be used only for recognizing hand-written digits, the generative model can be also used for sampling of new hand-written digits that match a given label.*

3.2 Parametric models

One of the central central concepts in machine learning is called a *statistical model*, which we define as a pair $(S, \mathcal{P})$, where S is a the sample space and $\mathcal{P}$ is a set of probability distributions on S. The intuition behind the statistical model is, that there is a "true" (but unknown) probability distribution that generates observed data from the sample space S and a set of distributions $\mathcal{P}$ which is the embodiment of assumptions about the true distribution.

If the set of distributions $\mathcal{P}$ can be described by a finite number of parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$, we call such model a *parametric model.*

Definition 4 (Parametric model)**.** *A parametric model is a pair* $(S, \mathcal{P})$*, where* S *is a sample space and* P *collection of probability distributions:*

$$\mathcal{P} = \{P_\theta \mid \theta \in \Theta\}$$

where P_θ *is a probability distribution described by a finite-dimensional vector of parameters* $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$.

From now on, it will be assumed that the sample space S from Definition 4 is given implicitely and the probability distribution is defined by a probability density function $p : S \to S$. Such parametric model will be denoted by $p(\cdot; \theta)$.

Example 2. *The family of normal distributions* $\mathcal{N}(\mu, \sigma^2)$ *is a very simple parametric model described by only two paramers – the mean* μ *and the variance* σ^2*, or written in a vector form as* $\theta = (\mu, \sigma^2)$.

Example 3. *A notable subset of parametric models are so called “mixture models”. For example the Gaussian Mixture Model (GMM) consists of* K *weighted normally-distributed components described by parameters* $\theta = (\mu_1, \mu_2, \ldots, \mu_K, \sigma_1^2, \sigma_2^2, \ldots, \sigma_K^2, \phi_1, \phi_2, \ldots, \phi_K)$ *and is defined by:*

$$p(\boldsymbol{x}; \theta) = \sum_{i=1}^{K} \phi_i \mathcal{N}(\boldsymbol{x}; \mu_i, \sigma_i^2)$$

Complex parametric models trained by deep learning algorithms can have several thousands of parameters. The measure of “complexity” of a parametric model is described by the term *capacity of a model* (or *Vapnik–Chervonenkis dimension* [VC15]). The exact

mathematical formulation of the capacity is out of scope of this diploma thesis. Very roughly speaking, we usually expect that the more parameters the model has, the more information about the sample space it can store. Models with too low capacity can only learn the most significant features from the training data (underfitting), while models with too high capacity can fit to the observations too tightly and "memorize" the training dataset (overfitting). Building models that are not underfitting nor overfitting is one of the main challenges of machine learning. The term capacity is also closely related to terms *bias* and *variance* which are discussed in Section 3.5.

3.3 Convergence of random variables

Parametric models are described by a sequence of parameters $\hat{\theta} = (\hat{\theta_1}, \hat{\theta_2}, \ldots, \hat{\theta_n})$, that are iteratively updated during the training of the model by a machine learning algorithm. After a certain number of iterations, these parameters tends to *converge* towards the true but unknown parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$ that characterize some sort of "idealized model". The estimated parameters $\hat{\theta}$ are modeled by a sequence of random variables.

When we talk about a sequence of real numbers (x_n), the notion of *convergence* towards x (denoted by $x_n \to x$) is an unambiguous concept, meaning that as n increasing, we are getting "closer and closer" towards x (or formally written as $\lim_{n\to\infty} x_n = x$). However, when we talk about a sequence of random variables, we can recognize different ways of interpreting what getting "closer and closer" actually means.

3.3.1 Modes of convergence

Various iterative machine learning algorithms are constrained by different *modes of convergence*.

Definition 5. *Let $\{X_n\}$ be a sequence of random variables and X a random variable. Then we say that X_n converges towards X:*

- ***almost surely*** *(denoted by $X_n \xrightarrow{a.s.} X$) if:*

$$\mathbb{P}[\lim_{n\to\infty} X_n = X] = 1,$$

- ***in the L_p-norm*** *for $p > 0$ (denoted by $X_n \xrightarrow{L_p} X$) if:*

$$\lim_{n\to\infty} \mathbb{E}[|X_n - X|^p] = 0,$$

- ***in probability*** *(denoted by $X_n \xrightarrow{P} X$) if for all $\epsilon > 0$:*

$$\lim_{n\to\infty} \mathbb{P}[|X_n - X| > \epsilon] = 0,$$

- ***in distribution*** *(denoted by* $X_n \xrightarrow{D} X$*) if for all bounded continuous functions* f*:*

$$\lim_{n\to\infty} \mathbb{E}[f(X_n)] = \mathbb{E}[f(X)].$$

Lemma 2. *The following implications hold:*

- $X_n \xrightarrow{L_s} X \implies X_n \xrightarrow{L_p} X$ *for* $s > p \geq 1$
- $X_n \xrightarrow{L_p} X \implies X_n \xrightarrow{P} X$
- $X_n \xrightarrow{a.s.} X \implies X_n \xrightarrow{P} X$
- $X_n \xrightarrow{P} X \implies X_n \xrightarrow{D} X$

Proof. Provided in [Vaa98]. □

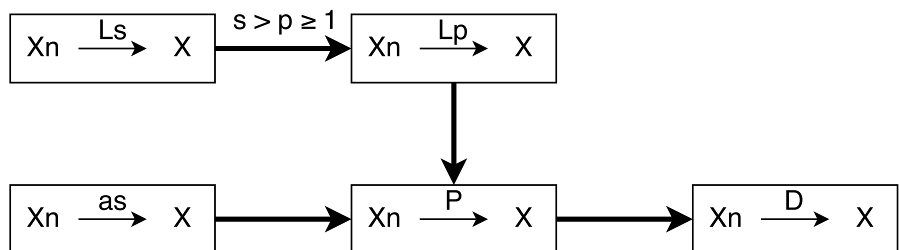


Figure 3.2: Implication graph of convergence modes.

3.4 Empirical risk minimization

Some machine learning problems can be formalized in terms of minimizing a generalization error called the *expected risk*. Assume a training dataset of m samples $\mathbb{X} = \{\mathbf{x_1}, \ldots, \mathbf{x_m}\}$ and desired outputs $\mathbb{Y} = \{\mathbf{y_1}, \ldots, \mathbf{y_m}\}$ where $\mathbf{x_i} \in \mathcal{X}, \mathbf{y_i} \in \mathcal{Y}$ are drawn from some underlying data-generating probability distribution $p_{data}(\mathbf{x}, \mathbf{y})$. We define a *decision function* $f : \mathcal{X} \to \mathcal{Y}$ parametrized by θ that takes a sample x and predicts the desired output $\mathbf{y}$.

Definition 6 (Expected risk)**.** *The expected risk* $R(\theta)$ *is given by:*

$$R(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim p_{data}}[\mathcal{L}(f(\mathbf{x};\theta), \mathbf{y})]$$

where $f(\mathbf{x};\theta)$ *is a decision function parametrized by* θ *and* $\mathcal{L}$ *is a loss function that measures the cost of making a decision* $f(\mathbf{x};\theta)$ *while the true output is* $\mathbf{y}$*.*

Calculating $R(\theta)$ is intractable because the underlying true data-generating distribution $p_{data}(\mathbf{x}, \mathbf{y})$ is unknown. In practice, we only have available a finite number of samples $\mathbb{X}$, so we replace the true distribution $p_{data}(\mathbf{x}, \mathbf{y})$ with an empirical distribution $\hat{p}_{data}(\mathbf{x}, \mathbf{y})$ defined only by the training dataset $\mathbb{X}$. By strong law of large numbers it holds that $\hat{p}_{data}(\mathbf{x};\theta) \xrightarrow{a.s.} p_{data}(\mathbf{x};\theta)$ and provided enough training samples we get a meaningful approximation of the true distribution p_{data}. Hence, we can optimize the *empirical risk* $R_{emp}(\theta)$ instead of the intractable expected risk $R(\theta)$.

Definition 7 (Empirical risk)**.** *The empirical risk $R_{emp}(\theta)$ is given by:*

$$R_{emp}(\theta) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\hat{p}_{data}}[\mathcal{L}(f(\mathbf{x};\theta), \mathbf{y})] = \frac{1}{m}\sum_{i=1}^{n} \mathcal{L}(f(\mathbf{x}_i;\theta), \mathbf{y_i})$$

where $n \in \mathbb{N}$ is the number of samples in the training dataset $\mathbb{X} = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}\}$.

The goal of empirical risk minimization is then to find parameters $\hat{\theta}$ that minimizes the difference between the prediction $f(\mathbf{x};\theta)$ and the desired output $\mathbf{y}$ the most among all considered parameters θ.

Definition 8 (Empirical risk minimization)**.** *The empirical risk minimization is the process of finding the best parameters $\hat{\theta}$ that minimizes the empirical risk $R_{emp}(\theta)$ among all considered $\theta \in \Theta$:*

$$\hat{\theta} = \arg\min_{\theta\in\Theta} R_{emp}(\theta) \tag{3.1}$$

Empirical risk minimization is not an algorithm, but rather a very general optimization approach. The concrete algorithm can be designed by constructing the procedure of "guessing" different values of θ and by defining the decision function f and the loss function $\mathcal{L}$.

3.5 Bias-variance trade-off

A common issue in machine learning is that the model fails to generalize because it fits too tightly or too loosely to the training dataset. The quality of the estimation can be measured with *mean squared error (MSE)*, which is an average of the square of the deviations between the estimated parameter $\hat{\theta}$ and the true parameter θ.

Definition 9. *The mean squared error of the estimate $\hat{\theta}$ with respect to the true parameter θ is given by:*

$$MSE_\theta(\hat{\theta}) = \mathbb{E}[(\hat{\theta} - \theta)^2]$$

The MSE is always non-negative and decreases with increasing quality of the estimator. Lemma 3 shows that the MSE can be decomposed into two components – *variance* and *bias*. Bias is the difference between the average value of the estimator and the true parameter. Variance is the variability of estimates of the true parameter.

Definition 10. *The bias of the estimate $\hat{\theta}$ with respect to the true parameter θ is given by:*

$$Bias_\theta[\hat{\theta}] = \mathbb{E}[\hat{\theta}] - \theta$$

The variance of the estimate $\hat{\theta}$ with respect to the true parameter θ is given by:

$$Var[\hat{\theta}] = \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2]$$

Lemma 3 (Bias-variance trade-off)**.** *Let θ be the true parameter and $\hat{\theta}$ the estimate of this parameter. The MSE of $\hat{\theta}$ w.r.t. θ can be decomposed as follows [GBD92]:*

$$MSE_\theta(\hat{\theta}) = Bias_\theta[\hat{\theta}]^2 + Var[\hat{\theta}]$$

Proof.

$$\begin{aligned}
\mathrm{MSE}_\theta(\hat{\theta}) &= \mathbb{E}[(\hat{\theta} - \theta)^2] \\
&= \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}] + \mathbb{E}[\hat{\theta}] - \theta)^2] \\
&= \underbrace{\mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2]}_{\mathrm{Var}[\hat{\theta}]} + \underbrace{(\mathbb{E}[\hat{\theta}] - \theta)^2}_{\mathrm{Bias}_\theta[\hat{\theta}]^2} + 2\mathbb{E}[(\mathbb{E}[\hat{\theta}] - \theta)(\hat{\theta} - \mathbb{E}[\hat{\theta}])]
\end{aligned}$$

And the cross-product term can be eliminated as follows:

$$\begin{aligned}
\mathbb{E}[(\mathbb{E}[\hat{\theta}] - \theta)(\hat{\theta} - \mathbb{E}[\hat{\theta}])] &= \mathbb{E}[\hat{\theta}\mathbb{E}[\hat{\theta}] - \theta\hat{\theta} - \mathbb{E}[\hat{\theta}]^2 + \theta\mathbb{E}[\hat{\theta}]] \\
&= \mathbb{E}[\hat{\theta}]^2 - \theta\mathbb{E}[\hat{\theta}] - \mathbb{E}[\hat{\theta}]^2 + \theta\mathbb{E}[\hat{\theta}] \\
&= 0
\end{aligned}$$

□

A model with the variance too high is referred to as an *overfitting* model. On the other hand, a model with bias too high is referred to as an *underfitting* model. This phenomenon is illustrated in Figure 3.3 and demonstrated on a concrete instance of a regression problem in Figure 3.4.

3.6 Point estimation

Although the term *estimator* has already been intuitively used in Section 3.5, a formal definition is useful in order to explore some interesting properties. Assume that there exists

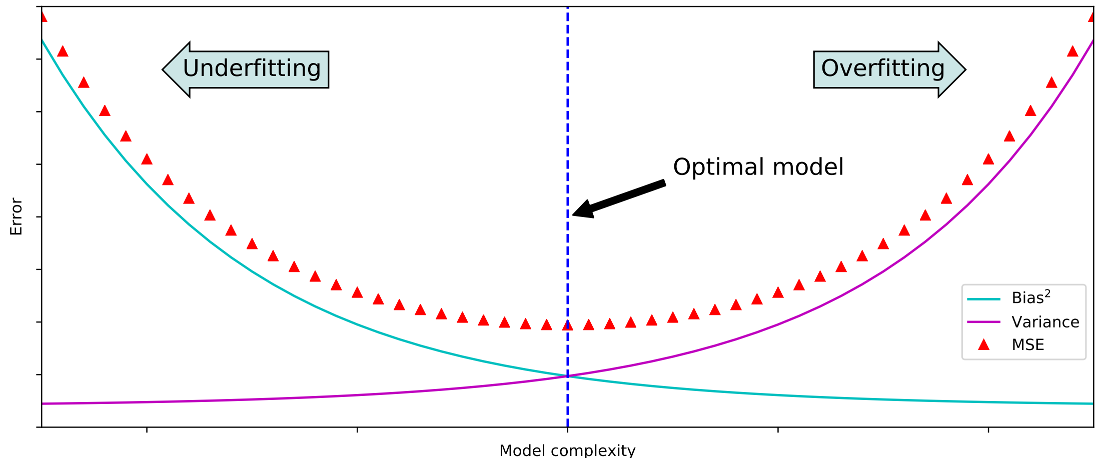
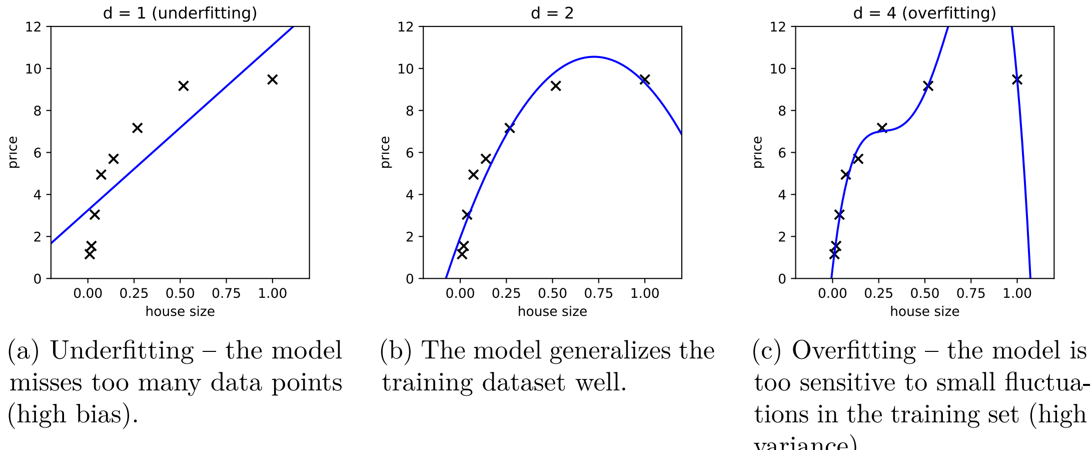


Figure 3.3: Bias-variance trade-off.



(a) Underfitting – the model misses too many data points (high bias).

(b) The model generalizes the training dataset well.

(c) Overfitting – the model is too sensitive to small fluctuations in the training set (high variance).

Figure 3.4: Generalization issues demonstrated on an instance of a regression problem.

a true (but unknown) data-generating distribution function $p_{data}(x, y; \theta)$ parametrized by some unknown parameters θ. Also assume that we are given a *training dataset* $\mathbb{X} = \{x_1, x_1, \ldots, x_m\}$. Depending on the goal of a given machine learning task, we are typically interested in finding the conditional probability distribution $p_{model}(y|x; \hat{\theta}_m)$ (discriminative model) or the joint probability distribution function $p_{model}(x, y; \hat{\theta}_m)$ (generative model) parametrized by $\hat{\theta}_m$, where m is a number of samples in the dataset (or just $\hat{\theta}$ if the number of samples is not important in a given context), that estimate the true probability distribution function $p_{data}(x, y; \theta)$. If $\hat{\theta}$ is only a single best estimation of the true parameter θ, we will call such estimation a *point estimation*. In contrast, if $\hat{\theta}$ is an interval of plausible estimations of θ, it is referred to as *interval estimation*.

Definition 11 (Point estimator)**.** *A point estimator (or statistic) of a parameter* θ *is any function* $g : \mathcal{X}^m \to \Theta$ *of the data* $\mathbb{X} = \{\mathbf{x_i} \in \mathcal{X}\}_{i=1}^m$*. The output of the point estimator* g *is called the point estimate* $\hat{\theta}_m$*, i.e.:*

$$\hat{\theta}_m = g(\mathbb{X}) = g(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m})$$

From now on, if not explicitly specified, an *estimator* will be considered a *point estimator* (and not *interval estimator*). Because the function g from Definition 11 can be any function of the data, we define some desired properties that make the estimate $\hat{\theta}$ of true parameter θ useful.

Definition 12 (Properties of an estimator)**.** *A point estimator* $g : \mathcal{X}^m \to \Theta$ *is:*

- *(simply) consistent if:*
$$\hat{\theta}_m \xrightarrow{P} \theta,$$
- *MSE consistent if:*
$$\hat{\theta}_m \xrightarrow{L_2} \theta,$$
- *asymptotically normal if it converges to a normal distribution:*
$$\sqrt{m}(\hat{\theta}_m - \theta) \xrightarrow{D} \mathcal{N}(0, \sigma_\theta^2)$$
where σ_θ^2 *is called an asymptotic variance of the estimate,*
- *unbiased if:*
$$Bias_\theta[\hat{\theta}_m] = 0,$$
- *asymptotically unbiased if:*
$$Bias_\theta[\hat{\theta}_m] \xrightarrow{D} 0.$$

3.6.1 Bias vs. consistency

It is worth to outline the difference between these two related concepts. Assume that we are estimating an unknown parameter θ with a point estimate $\hat{\theta}_m$ on a training dataset of m samples, then vaguely speaking:

- **consistency** ($\hat{\theta}_m \xrightarrow{P} \theta$) says that with an increasing number of samples in the dataset, the probability that the estimate $\hat{\theta}_m$ will be different from the true parameter θ is very low,

- **asymptotic unbiasedness** ($\mathbb{E}[\hat{\theta}_m] \xrightarrow{D} \theta$) says that with an increasing number of samples in the dataset, the estimates $\hat{\theta}_m$ will be "centered" around the true parameter θ.

As shown in Lemma 4, asymptotic unbiasedness implies consistency under the assumption that the variance also converges towards 0. However, the converse does not hold, as demonstrated on a pathological example of a distribution in Example **??**.

Lemma 4. *If the variance of an asymptotically unbiased estimator of θ converges in distribution towards 0, then it is a consistent estimator of θ, i.e.:*

$$Bias_\theta[\hat{\theta}_m] \xrightarrow{D} 0 \wedge Var[\hat{\theta}_m] \xrightarrow{D} 0 \implies \hat{\theta}_m \xrightarrow{P} \theta.$$

Proof. From Lemma 3:

$$\begin{aligned} \hat{\theta}_m \xrightarrow{L_2} \theta &\iff \lim_{m\to\infty} \mathbb{E}[|\hat{\theta}_m - \theta|^2] = 0 \\ &\iff \lim_{m\to\infty} (\mathrm{Bias}_\theta[\hat{\theta}_m]^2 + \mathrm{Var}[\hat{\theta}_m]) = 0 \\ &\iff \mathrm{Bias}_\theta[\hat{\theta}_m]^2 \xrightarrow{D} 0 \wedge \mathrm{Var}[\hat{\theta}_m]^2 \xrightarrow{D} 0. \end{aligned}$$

From Lemma 2 then follows that MSE consistency implies simple consistency:

$$\hat{\theta}_m \xrightarrow{L_2} \theta \implies \hat{\theta}_m \xrightarrow{P} \theta.$$

□

3.7 Maximum Likelihood Estimation (MLE)

A very simple method for estimating the parameters θ of a statistical model given a set of observed data $\mathbb{X} = \{x_1, x_2, \dots, x_n\}$ is based on maximizing the likelihood that the data was generated by our statistical model.

Definition 13 (Maximum Likelihood Estimation)**.** *Let $\mathbb{X} = \{x_1, \dots, x_m\}$ be a set of samples drawn from a true (but unknown) probability distribution $p_{data}(x)$. Let $p_{model}(x;\theta)$ be a distribution parametrized by a set of parameters θ and estimating the true distribution $p_{data}(x)$. The maximum likelihood estimate $\hat{\theta}_{MLE}$ is then defined by:*

$$\hat{\theta}_{MLE} = \arg\max_{\theta} p_{model}(\mathbb{X};\theta) = \arg\max_{\theta} \prod_{i=1}^{m} p_{model}(x_i;\theta)$$

The product in Definition 13 is undesirable, because finding the maximum often involves calculating a derivative of the function being maximized. However, we can observe that $\forall f : \arg\max f(x) = \arg\max g(f(x))$ if g is a monotonically increasing function. Hence, it is more convenient to obtain an equivalent estimator by putting $g(x) = \log(x)/m$ and optimizing what is called an *expected log-likelihood*:

$$\begin{aligned}\hat{\theta}_{MLE} &= \arg\max_{\theta} \frac{1}{m} \log \prod_{i=1}^{m} p_{model}(x_i;\theta) \\ &= \arg\max_{\theta} \frac{1}{m} \sum_{i=1}^{m} \log p_{model}(x_i;\theta) \\ &= \arg\max_{\theta} \mathbb{E}_{x\sim p_{data}}[\log p_{model}(x;\theta)]\end{aligned} \tag{3.2}$$

As shown in [NM86], the estimate given by $\hat{\theta}_{MLE}$ has favourable properties – it is a consistent, asymptotically unbiased and asymptotically normal estimator of the true parameter θ.

3.8 Maximum A Posteriori (MAP) estimation

Maximum a posteriori (MAP) estimation is closely related to Maximum Likelihood Estimation (MLE) described in Section 3.7. While MLE is based on frequentist inference and estimations depend entirely on the training data, MAP employs Bayesian inference and requires to make some additional assumptions about the prior distribution. [GBC16]

Assume there exist a prior distribution $p(\theta)$ over the true parameter θ and the probability of the evidence is fixed. Using Bayes' theorem we have:

$$\underbrace{p(\theta \mid x)}_{posterior} = \frac{\overbrace{p(x \mid \theta)}^{likelihood}\,\overbrace{p(\theta)}^{prior}}{\underbrace{\int p(x)}_{evidence}} \propto p(x \mid \theta)p(\theta) \tag{3.3}$$

In Equation (3.3), we omit the normalizing constant $p(x)$, since it's always positive and we are only interested in optimization of parameter θ. We can define the optimization problem in a similar way as in case of MLE.

Definition 14. *Let $\mathbb{X} = \{x_1, \ldots, x_m\}$ be a set of observations. The maximum a posteriori estimation $\hat{\theta}_{MAP}$ is then defined by:*

$$\hat{\theta}_{MAP} = \arg\max_{\theta} p(\theta \mid x) = \arg\max_{\theta} \log p(x \mid \theta) + \log p(\theta)$$

MAP can be seen as a regularized version of MLE – more about regularization is discussed in Section 3.13. In fact, if the prior is a uniform distribution (i.e. if $p(\theta)$ is a constant) then the penalty has no effect and in such cases MAP is equal to MLE.

3.9 Information entropy

Many machine learning algorithms have their roots in information theory. The central concept of the information theory is called an "information entropy", which is illustrated on Example 4 for each coin.

Example 4. *Suppose we make an experiment and flip three different coins, 20 times each. We observe the following values (H = head, T = tail):*

1. *HTTTHHTHTHHHHTTTTTHH*
2. *HHHHHHHHHHHHHHH**T**HHH*
3. *HHHHHHHHHHHHHHHHHHHH*

The first case is an unbiased coin which gave us an equal number of heads and tails, i.e. $p(H) = p(T) = 1/2$. Hence, we are very uncertain about the next flip – we say that each observed flip has the same amount of self-information. The second coin has $p(H) = 19/20$ and $p(T) = 1/20$ – flips where we hit a head provided only a small amount of information (i.e. we wouldn't be very surprised if the next flip in a row is also head), while the single tail that we hit was very informative. The last coin hits only heads, so each of the flips provide no self-information at all.

This simple example illustrates the concept of entropy, which measures the average amount of self-information over all observations of some random event. The entropy of an unbiased coin is maximal, while the coin that hits only heads (or only tails) has zero entropy.

Definition 15 (Self-information)**.** *The self-information of a discrete random variable X with a set possible outcomes $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ and probability distribution $p(x)$*

is defined by:

$$I(x) = -\log p(x).$$

Definition 16 (Entropy)**.** *The entropy $H(X)$ of a discrete random variable X is defined by:*

$$H(p) = \mathbb{E}[I(x)] = -\sum_{x} p(x) \log p(x).$$

The entropy can be also defined for outcomes of two random variables X and Y, informally speaking:

- **joint entropy** $H(X, Y)$ – the average self-information of observing the outcomes of two random variables X and Y,
- **conditional entropy** $H(X|Y)$ – the average self-information needed to describe the outcome of a random variable X given the outcome of Y,
- **mutual information** $I(X, Y)$ – the mutual dependence between random variables X and Y.

Definition 17 (Entropy of two variables)**.** *The **joint entropy** $H(X, Y)$ of two discrete random variables X and Y with respective sets of possible outcomes $\mathcal{X}$ and $\mathcal{Y}$ is defined as:*

$$H(X, Y) = -\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x, y) \log p(x, y).$$

*The **conditional entropy** $H(X|Y)$ of a discrete random variable X given Y with respective sets of possible outcomes $\mathcal{X}$ and $\mathcal{Y}$ is defined as:*

$$H(X|Y) = H(X, Y) - H(Y) = -\sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(y)}.$$

*The **mutual information** $I(X, Y)$ of two discrete random variables X and Y with*

respective sets of possible outcomes $\mathcal{X}$ and $\mathcal{Y}$ is defined as:

$$\begin{aligned} I(X,Y) = &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X) \\ &= H(X) + H(Y) - H(X,Y) \\ &= H(X,Y) - H(X|Y) - H(Y|X) \\ &= - \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}. \end{aligned}$$

The entropy of two random variables is illustrated on Figure 3.5.

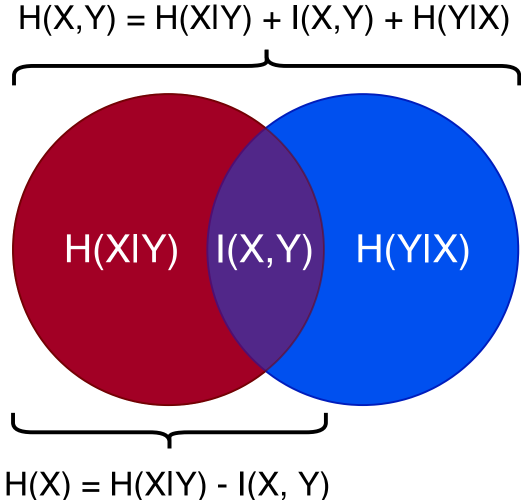


Figure 3.5: The entropy of two random variables X (red) and Y (blue).

Some interesting properties of the entropy:

- Information is non-negative, i.e. $\forall X : I(X) \geq 0$ and $\forall X, Y : I(X,Y) \geq 0$. As a corollary, also $H(X) \geq H(X|Y)$.
- Entropy (an average information) is non-negative, i.e. $H(\cdot) \geq 0$.
- Joint entropy is symmetric, i.e. $H(X,Y) = H(Y,X)$.
- Conditional entropy is not symmetric, i.e. $H(X|Y) \neq H(Y|X)$.
- The random variable has the maximum entropy if its outcomes are normally distributed [CT91].

3.10 Kullback–Leibler (KL) divergence

Let $p(x)$ and $q(x)$ be two probability distribution functions. Informally, KL-divergence $D_{KL}(p\|q)$ is a non-negative number that tells us, how different these two distributions are. $D_{KL}(p\|q) = 0$ indicates that distributions p and q have similar behaviour. Formally, if $\{p_1, p_2, \dots\}$ is a sequence of probability distributions, then:

$$\lim_{n\to\infty} D_{KL}(p_n\|q) = 0 \implies p_n(x) \xrightarrow{D} q(x).$$

Definition 18 (KL-divergence)**.** *Let $p(x)$ and $q(x)$ be two probabilisty distributions. KL-divergence between p and q is given by:*

$$D_{KL}(p\|q) = \mathbb{E}_{x\sim p}\Big[\log\frac{p(x)}{q(x)}\Big] = \mathbb{E}_{x\sim p(\theta)}[\log p(x) - \log q(x)].$$

Some important properties can be immediately seen from the definition:

- $D_{KL}(p\|q) \geq 0$ and $D_{KL}(p\|q) = 0 \iff p = q$ (*Gibb's inequality*)
- $D_{KL}(p\|q) \neq D_{KL}(q\|p)$
- $D_{KL}(p\|q) = D_{KL}(p_1\|q_1) + D_{KL}(p_2\|q_2)$ for joint distributions $p(x) = p_1(x)p_2(x)$ and $q(x) = q_1(x)q_2(x)$.

Lemma 5. *If p is fixed, minimizing KL-divergence is equivalent to maximizing empirical likelihood.*

Proof. Let p be a fixed but unknown sampling distribution indexed by parameter θ and q the model distribution indexed by estimator $\hat{\theta}$.

The KL-divergence can be decomposed into *entropy* $H(p) = \mathbb{E}_{x\sim p(\theta)}[-\log p(x)]$ and *cross-entropy* $H(p, q) = \mathbb{E}_{x\sim p}[-\log q(x)]$ as follows:

$$\begin{aligned} D_{KL}(p\|q) &= \mathbb{E}_{x\sim p(\theta)}[\log p(x) - \log q(x)] \\ &= \mathbb{E}_{x\sim p(\theta)}[\log p(x)] - \mathbb{E}_{x\sim p}[\log q(x)] \\ &= -\underbrace{H(p)}_{entropy} + \underbrace{H_p(q)}_{cross-entropy} \end{aligned}$$

Now, suppose that distributions p and q are parametrized by θ. Since the sampling distribution p is fixed, the entropy $H(p)$ is constant and can be omitted in a parameter optimization problem:

$$
\begin{aligned}
\arg\min_\theta D_{KL}(p\|q) &= \arg\min_\theta H_p(q) \\
&= \arg\min_\theta \mathbb{E}_{x\sim p}[-\log q(x)] \\
&= \arg\max_\theta \mathbb{E}_{x\sim p}[\log q(x)] \\
&= \hat{\theta}_{MLE}
\end{aligned}
$$

□

Definition 19 (Cross-entropy)**.** *The cross-entropy $H_p(q)$ for probability two distributions p and q is defined as:*

$$H_p(q) = -\mathbb{E}_{x\sim p}[\log q(x)] = -\sum_x p(x)\log q(x)$$

The cross-entropy is an important function that is widely used in practice as a loss function for training deep learning models. Efficient GPU implementations are available in most popular deep learning frameworks like TensorFlow, PyTorch or Theano.

3.11 Stochastic Gradient Descent (SGD)

Methods based on *gradient descent* are one of the most frequently used iterative methods for estimation of parameters of deep learning models. The simplest version of the gradient descent algorithm starts by random initialization of parameters θ_0. Then in each i-th iteration it calculates the average gradient [1] ν_i over all n samples in the training dataset as follows:

$$\nu_i := \nabla J(\theta_i) = \frac{1}{n}\nabla_{\theta_i}\sum_{j=1}^{n}\mathcal{L}(f(\mathbf{x}_j;\theta_i),\mathbf{y_j}) \tag{3.4}$$

where $J(\cdot)$ is an empirical risk as defined in Section 7. The gradient $\nabla J(\theta_i)$ is a generalization of the derivative for functions with multiple variables and it represents the direction and steepness of the tangent of the function $J(\cdot)$. The average gradient ν is a vector that determines towards which direction and how much we update parameters in each iteration. The new parameters $\theta^{[i+1]}$ of the model are updated in each i-th iteration towards the direction of the average gradient as follows:

$$\theta_{i+1} := \theta_i - \epsilon\nu_i \tag{3.5}$$

[1] The gradient of a given function $f(x_1, x_2, \dots, x_n)$ (denoted by ∇f) is a vector of all partial derivatives of f, i.e. $\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n}\right)$

where ϵ is a *learning rate*. The disadvantage of this approach is that for very large datasets, it may take a long time to calculate even a single parameter update. Taking only a subset of the training samples in each iteration allows to efficiently work with even very large datasets [BB08].

A randomized version of a full gradient descent algorithm is known *Stochastic Gradient Descent (SGD)*. The gradient in SGD is evaluated only over a single sample x_i in every i-th iteration, which means that new parameters are updated as follows:

$$\theta_{i+1} := \theta_i - \nu \nabla_{\theta_i} \mathcal{L}(f(\mathbf{x}_i; \theta^i), \mathbf{y_i}). \tag{3.6}$$

In practice, subsets of the training dataset of size around 10 to 10.000 samples (so called "mini-batches") are commonly used. The pseudocode of a SGD algorithm with mini-batches is shown in Algorithm 3.1 and its runtime progress over 5 iterations is illustrated on two examples in Figure 3.6. The algorithm stops when we reach the stopping criterion – for example if the average gradient $\nu \approx \mathbf{0}$, which means we have reached a local minimum and the parameters are not updated anymore.

Algorithm 3.1: Stochastic Gradient Descent (SGD) with mini-batches.

Input: Learning rate ϵ, training dataset $\mathbb{X}$ and desired outputs $\mathbb{Y}$

Output: Parameters θ of the given parametric model f, s.t.
$\forall \mathbf{x_i} \in \mathbb{X}, \mathbf{y_i} \in \mathbb{Y} : \mathbf{y_i} \approx f(\mathbf{x_i}; \theta)$

1 Choose initial parameters θ

2 **while** *stopping criterion not met* **do**

3   Choose a mini-batch of m samples $\{x_1, x_2, \ldots, x_m\} \subseteq \mathbb{X}$ with corresponding outputs $\{y_1, y_2, \ldots, y_m\} \subseteq \mathbb{Y}$

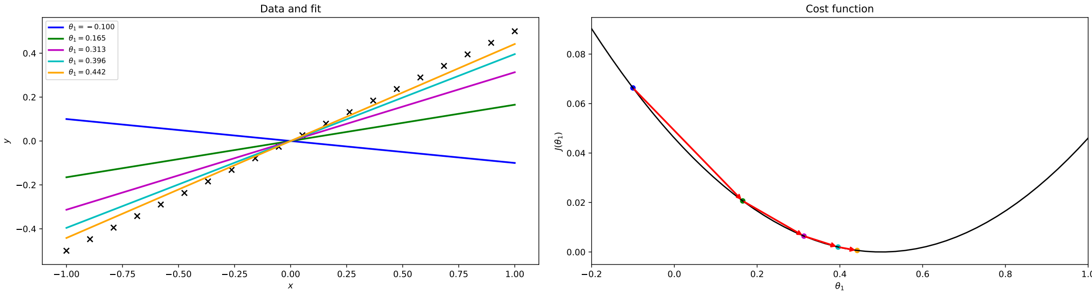
4     $\nu := \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} \mathcal{L}(f(\mathbf{x}_i; \theta), \mathbf{y_i})$ ▷ Calculate gradient

5     $\theta := \theta - \epsilon\nu$ ▷ Update parameters

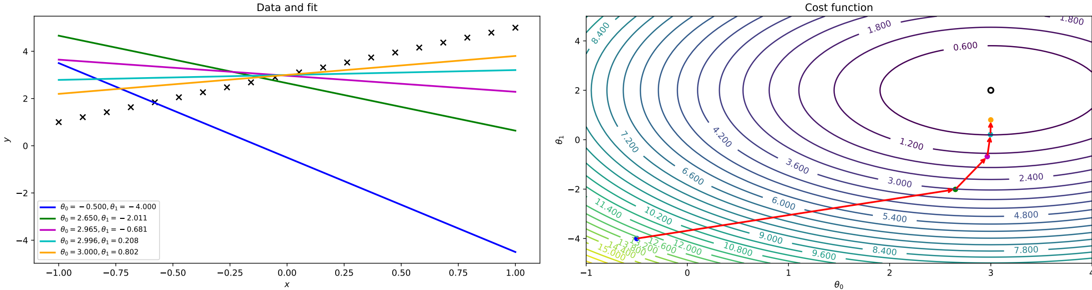
6 **end**

7 **return** θ

The SGD is a general optimization algorithm and its performance significantly depends on the choice of *hyperparameters* – learning rate ν, cost function $\mathcal{L}(\cdot)$ and the decision function $f(\cdot)$. Sections 3.11.1 and 3.11.2 provide more details on choosing the learning rate and the cost function. An obvious limitation of the SGD algorithm is that the decision function f must be differentiable. The only decision function considered in this diploma thesis is a *multilayered perceptron* and is described in Section 3.12.



(a) Stochastic Gradient Descent (SGD) optimizing a single parameter.



(b) Stochastic Gradient Descent (SGD) optimizing two parameters.

Figure 3.6: First 5 iterations of a SGD algorithm trying to find the optimum of a normal distribution function in 2 and 3 dimensions. The blue line depicts a randomly initialized parameters θ_0, that are then iteratively improved. The gradient is decreasing as we are reaching the local optimum.

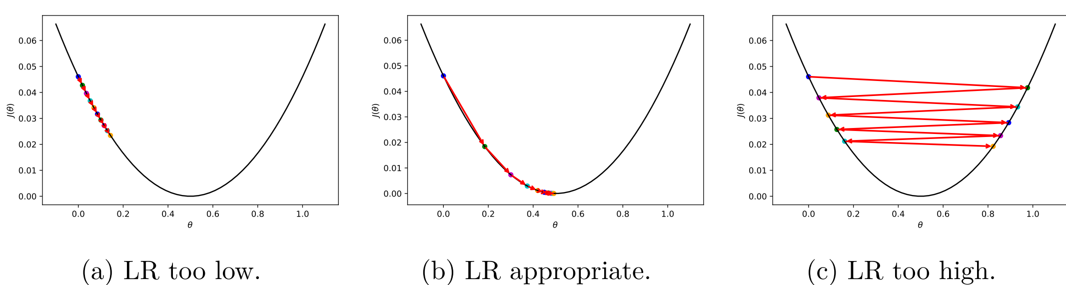
3.11.1 Learning rate

The learning rate ϵ is a hyperparameter that controls the speed of updating parameters of the model the SGD algorithm. In general, the higher the value ϵ is, the faster the SGD algorithm learns. However, too high value of ϵ may result in oscillating changes of parameters θ and divergent behaviour. On the other hand, too low values of ϵ may cause the algorithm to get stuck in a critical point (i.e. local optimum or a saddle point). The effect of too high and too low learning rate is illustrated in Figure 3.7. A common practice to avoid these issues is to adaptively change the learning rate in each iteration of the algorithm. Parameters of the model are then updated as follows:

$$\theta_{i+1} := \theta_i - \epsilon_i \nu_i$$

where ϵ_i is the adaptive learning rate at i-th iteration of SGD algorithm. Sufficient condition that guarantee the convergence of SGD algorithm are the following [GBC16]:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty$$
$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$



(a) LR too low. (b) LR appropriate. (c) LR too high.

Figure 3.7: The effect of a learning rate (LR) on the performance of a SGD algorithm. If the LR is too low, the gradient is changed only very slowly and the algorithm may get stuck in a local optimum. On the other hand, if the LR is too high, the gradient oscillates too aggressively which may lead to a divergent behaviour.

There are many different adaptive learning rates used in practice. One of the simplest approaches with a guaranteed convergence is a linearly decaying learning rate, defined as follows for i-th iteration:

$$\epsilon_i = \frac{\epsilon_0}{1 + i \cdot \alpha} \tag{3.7}$$

where $\alpha > 0$ is a chosen decay rate.

3.11.2 Cost function

The cost function $\mathcal{L}(\hat{y}, y)$ measures how well our parametric model $p_{model}(\cdot; \theta)$ estimates the true probability distribution $p_{data}(\cdot)$ by comparing the distance between samples from these two distribution, i.e. $y \sim p_{data}$ and $\hat{y} \sim p_{model}$.

Quadratic cost

One of the simplest loss functions is *Mean Squared Error (MSE)*, already discussed in Section 3.5 in context of point estimators. It is the average squared prediction error between two sets of n samples, defined as follows:

$$\mathcal{L}_{MSE} = \frac{1}{n} \|y_i - \hat{y}_i\|_2^2 = \frac{1}{n} \sum_{i=0}^{n} (y_i - \hat{y}_i)^2 \tag{3.8}$$

In practical implementation where the Equation 3.8 is used only as a loss function for an optimization problem, the constant $1/n$ can be ommited – such loss function is known as *quadratic cost*. As explained in [KK17], quadratic cost function is useful for normally distributed data. However, many real-world problems are not normally distributed which leads to erroneous estimates and hence we need to use other cost function.

Cross-entropy cost

Kullback-Leibler divergence (also called *information gain*), which was discussed in Section 3.10, is a commonly used as a loss function for many machine learning problems. This measure has its root in information theory and reflects the amount of information in bits lost when we approximate distribution q with another distribution p:

$$\mathcal{L}_{KL} = \mathbb{E}_{x \sim p}\Big[\log \frac{p(x)}{q(x)}\Big] = \underbrace{H(P,Q)}_{cross-entropy} - \underbrace{H(p)}_{entropy} \tag{3.9}$$

The entropy $H(p)$ of the true distribution p is a constant, so it can be omitted for optimization problems.

3.12 Artificial Neural Network (ANN)

Artificial Neural Network (ANN) is a parametrized model inspired by a biological neural network in animal brain. This network can be defined as a directed computational graph consisting of a set of vertices (called *neurons* or *units*) connected by edges. Each neuron holds two parameters that are being optimized – *weight* and *bias*.

Formally, a single artificial neuron (also called a *perceptron*) with n input edges and a single output edge is modeled by the following function:

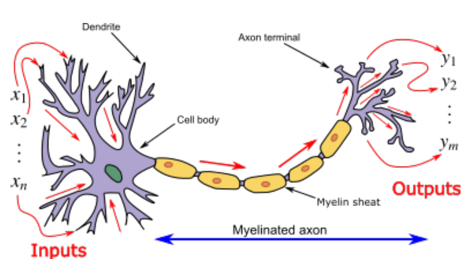
$$y = f(\mathbf{w}^T\mathbf{x} + b) = f(\sum_{i=1}^{n} w_i x_i + b) \tag{3.10}$$

where $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is a column vector of inputs, y is a single output, b is a parameter called *bias*, $\mathbf{w} = (w_1, w_2, \dots, w_n)$ is a column vector of weights and f is an *activation function*. A comparison of biological and artificial neurons is depicted in Figure 3.8.

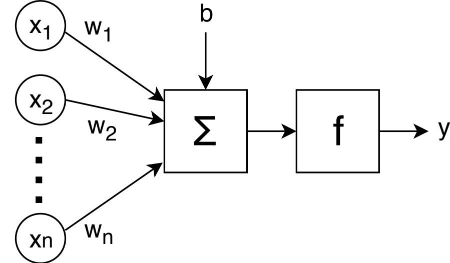
3.12.1 Feed-forward multilayer perceptron

It is a common practice to group multiple neurons into *layers* that form a *multilayer feed-forward network*, as depicted in Figure 3.9. Formally, each layer is a function:

$$\mathbf{y} = f(W\mathbf{x} + \mathbf{b}) \tag{3.11}$$



(a) Biological neuron with a signal flow from input from dendrites to output axon terminals. [Wik18]



(b) Artificial neuron (perceptron) with inputs $x_1, x_2, \ldots, x_n$ and single output y modeled as a mathematical function $y = f(\sum_{i=1}^{n} w_i x_i + b)$.

Figure 3.8: Comparison of a biological and artificial neuron.

where $\mathbf{x} \in \mathbb{R}^n$ is the input vector of the layer, $\mathbf{y} \in \mathbb{R}^m$ is the output vector of the layer, $f : \mathbb{R}^m \to \mathbb{R}^m$ is an activation function, $W \in \mathbb{R}^{m \times n}$ is the weight matrix and $\mathbf{b} \in \mathbb{R}^m$ is a bias vector.

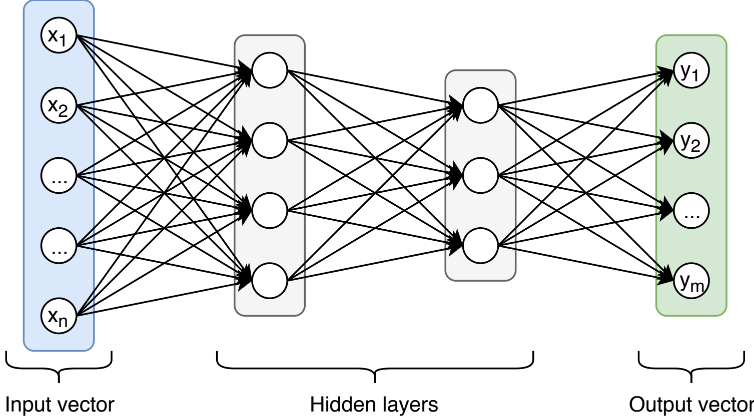


Figure 3.9: An example of a feed-forward neural network with two hidden layers, n inputs and m outputs.

For simplicity, the weight matrix W and the bias vector $\mathbf{b}$ can be “packed” into a single matrix with model parameters θ by appending the bias $\mathbf{b}$ as a column to the weight matrix W and extending the vector $\mathbf{x}$ by a constant 1, as follows:

$$\begin{aligned}\mathbf{y} = f(W\mathbf{x} + \mathbf{b}) &= f\Big(\begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}\Big) \\ &= f\Big(\begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} & b_1 \\ w_{21} & w_{22} & \dots & w_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} & b_m \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix}\Big) \\ &= f(\theta\tilde{\mathbf{x}})\end{aligned} \tag{3.12}$$

where $\tilde{\mathbf{x}}$ is a vector that was created from vector $\mathbf{x}$ by appending constant 1 to it.

Example 5. *A 4-layered feed-forward neural network can be defined as follows:*

$$\mathbf{y} = f_4(\theta_4 f_3(\theta_3 f_2(\theta_2 f_1(\theta_1 \mathbf{x})))) \tag{3.13}$$

where $\theta_1, \theta_2, \theta_3, \theta_4$ are the parameters of the model being optimized and f_1, f_2, f_3, f_4 are the activation functions of each layer.

3.12.2 Back-propagation algorithm

The basic idea of back-propagation is to iteratively "guess" values of the parameters $\theta_1, \theta_2, \dots, \theta_l$ in a feed-forward neural network based on given inputs $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ and an expected outputs $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_m\}$.

The parameters of the neural network in each layer are updated using the Stochastic Gradient Descent algorithm described in Section 3.11. The back-propagation algorithm takes advantage of a chain rule for computing the gradients of a composed function (such as the neural network function). Given a composed function $z = f(y), y = g(x)$, the chain rule applies as follows:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}. \tag{3.14}$$

The number of gradients that need to be computed grows linearly with the depth of a composed function (i.e. the number of layers of a neural network), which allows to use the back-propagation algorithm for training even very deep feed-forward neural networks.

The algorithm consists of two alternating steps, that are repeated till the stopping criterion is not met:

1. **Forward propagation.** Given an input $\mathbf{x}$ and the expected output $\mathbf{y}$, the algorithm calculates layer-by-layer the actual output of the network $\hat{\mathbf{y}} = \text{ANN}(\mathbf{x})$. The loss of the network is given by the loss function $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$.

2. **Backward propagation.** Using the chain rule for derivatives, the algorithm calculates gradients for each layer and updates the parameters $\theta_1, \theta_2, \ldots, \theta_l$ accordingly.

3.12.3 Activation function

A good choice of an activation function for each layer is crucial for designing the neural network. The activation function provides a source of non-linearity to the network and bounds the output values to a desired range. In order to calculate gradients using the back-propagation algorithm, the activation function should be differentiable.

The simplest activation function is a *linear function* $f(x) = x$. However, this function is only rarely useful in practice. A neural network that uses only linear activation functions in not capable to express non-linear dependencies between its inputs and outpus, i.e. the output of such network is just a linear combinations of the input. Only a network with non-linear activation functions is able to learn non-trivial problems with a reasonably small number of neurons [Knu].

Sigmoid $f(x) = \frac{1}{1+e^{-x}}$ or *hyperbolic tangent* $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}-1}$ are non-linear activation functions that are bounded to a constant range (sigmoid lies on an interval $(0, 1)$, tanh on $(-1, 1)$). This activation function is typically used for the output layer in a neural network with inputs normalized to the same interval (for example grayscale images with color vales of pixels between -1 to 1).

Rectified Linear Unit (ReLU) or *Leaky ReLU* is usually used for hidden layers, because calculating their derivatives requires only very little computational cost in both forward and backward propagation steps.

An overview of commonly used activation functions is provided in Table 3.1.

3.13 Regularization

Regularization is a strategy to prevent overfitting by providing additional information to the ML algorithm [BvdG11]. An effective regularizer works by reducing variance while not increasing bias too much. A machine learning algorithm may use a combination of several different regularization techniques in order to stabilize the training of the network.

An important regularization technique to prevent overfitting is to impose a penalty $\Omega(\theta)$ on the loss function $\mathcal{L}(\cdot;\theta)$ in certain regions of the problem space:

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\cdot;\theta) + \lambda\Omega(\theta) \tag{3.15}$$

where λ is a scaling constant controlling the importance of the regularizer term $\Omega(\theta)$.

Name	Plot	Activation function	Derivative
Linear		$f(x) = x$	$f'(x) = 1$
Sigmoid		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Binary step		$f(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & x \neq 0 \\ ? & x = 0 \end{cases}$
Hyperbolic Tangent		$f(x) = \tanh(x)$	$f'(x) = 1 - f(x)^2$
Arc Tangent		$f(x) = \tan^{-1}(x)$	$f'(x) = 1$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$
Leaky ReLU		$f(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & x < 0 \\ 1 & x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Table 3.1: An overview of commonly used activation functions.

3.13.1 L_p-norm penalty

Let consider a parametric model described with parameters θ. The L_p-norm is defined as follows:

$$\|\theta\|_p = \sqrt[p]{\sum_{i=1}^{n} |\theta|^p} \tag{3.16}$$

A regularization strategy based on L_p-norm moves the parameters of the model θ towards the origin by setting the regularizer penalty term Ω from Equation 3.15 as a p-squared L_p-norm as follows:

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\cdot;\theta) + \lambda\|\theta\|_p^p. \tag{3.17}$$

L_0-norm [2] regularizer is the number of non-zero elements in parameters of the model. Such regularizer tends to lower the capacity of the model by penalizing non-zero elements and hence making the vector of parameters θ sparse. However, it has been shown that optimizing a L_p-regularized problem with $0 \leq p < 1$ is NP-hard [GJY11].

In practical applications, L_1-norm regularizer (also called *LASSO* = least absolute shrinkage and selection operator) can be used instead of L_0 regularizer. This regularizer also induces sparsity of θ by shrinking parameters towards the origin and possibly making some of them zero [Tib96].

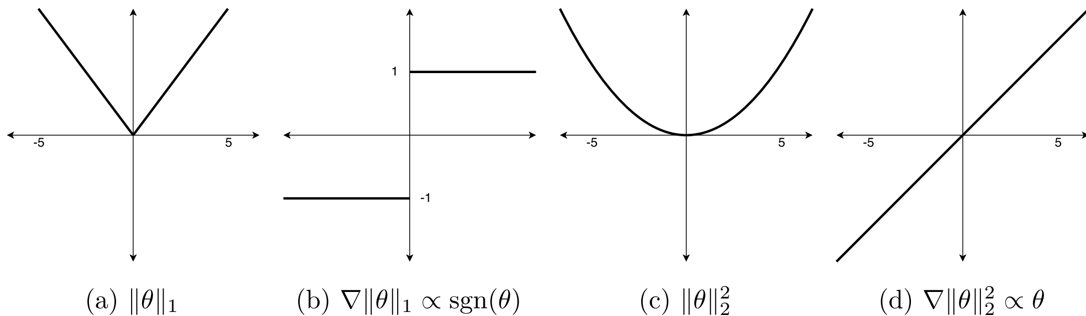
L_2-norm regularizer (also called *Ridge regression* or *Tikhonov regularization*) also encourages shrinking of the coefficients by making the sum of the squares of the coefficients to be small, but without inducing sparsity [HK70]. The sparsity-inducing effect is illustrated in Figure 3.10.

A compromise between L_1-norm and L_2-norm is called *elastic net* regularizer. In fact, both L_1-norm and L_2-norm regularizers can be seen as a special case of the elastic net regularizer, defined as follows:

$$\Omega(\theta) = \alpha\|\theta\|_1 + (1-\alpha)\|\theta\|_2^2 = \alpha \sum_{i=1}^{n} \theta_i + (1-\alpha)\theta^T\theta \tag{3.18}$$

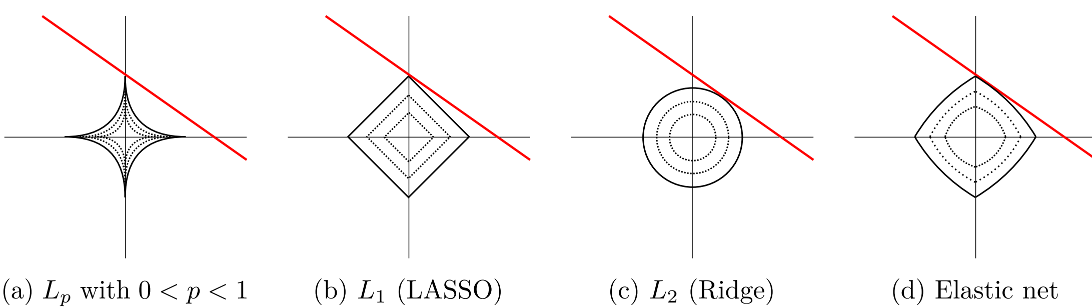
where $\alpha \in [0, 1]$ is a chosen constant. The elastic net regularizer encourages a grouping effect, which means that strongly correlated features tend to be assigned similar parameters [ZH03]. A comparison of L_p-norm regularizer is depicted in Figure 3.11.

[2] L_0 is not a norm in usual sense, in literature the following definition is used instead: $\|\theta\|_0 = \#(i \mid \theta_i \neq 0)$.



(a) $\|\theta\|_1$ (b) $\nabla\|\theta\|_1 \propto \text{sgn}(\theta)$ (c) $\|\theta\|_2^2$ (d) $\nabla\|\theta\|_2^2 \propto \theta$

Figure 3.10: L_1-norm and L_2-norm regularizers demonstrated on a gradient descent algorithm with an update rule $\theta \leftarrow \theta - \epsilon\nabla\|\theta\|_p^p$. The gradient ∇L_1 is moving the coefficients towards 0 at a constant rate -1 or 1 in each iteration. On the other hand, the magnitude of ∇L_2 is decreasing as the coefficient move towards 0, discouraging the coefficients to become sparse.



(a) L_p with $0 < p < 1$ (b) L_1 (LASSO) (c) L_2 (Ridge) (d) Elastic net

Figure 3.11: Geometric interpretation of L_p-norm regularizers.

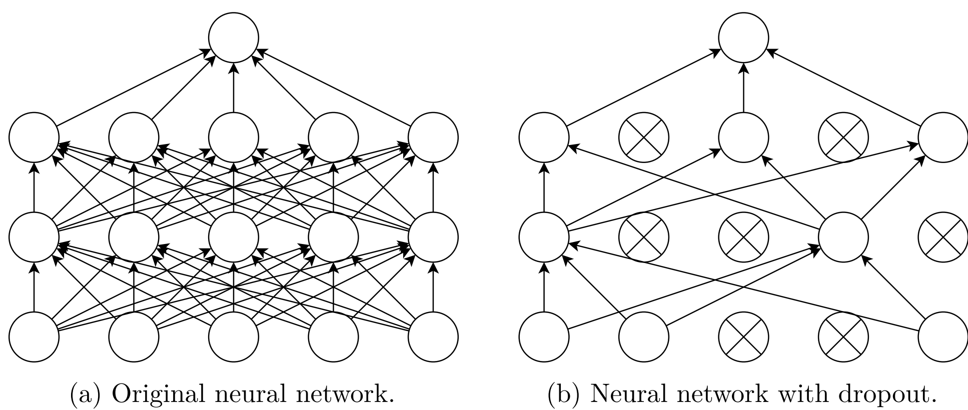
3.13.2 Early stopping

This regularization strategy works by splitting the original training dataset $\mathcal{D}$ into a new training dataset $\mathcal{D}_t$ and a validation dataset $\mathcal{D}_v$. The machine learning algorithm is trained on the dataset $\mathcal{D}_t$ and in each iteration, an error measured on the dataset $\mathcal{D}_v$ is compared with an error measured in the previous iteration. If the error was improved, parameters of the model are stored. The algorithm is stopped after p iterations without improvement (when it's expected that the model starts to overfit) and the stored parameters from the best model are returned. [GJP95].

This regularization is very easy to implement and can be wrapped around an existing machine algorithm without making any changes to the objective function. More complicated versions of this algorithm use *cross-validation* approach by splitting the dataset $\mathcal{D}$ into multiple partitions.

3.13.3 Dropout

Dropout is a regularization technique developed specifically for artificial neural networks. In each training iteration, each individual neuron are omitted is omitted with probability p as shown in Figure 3.12. The dropout is applied only during the training phase, during the validation phase the dropout parameter is set to $p = 1$. It has been shown that this technique significantly reduces overfitting by learning a redundant representation of the estimated distribution that does not rely only on a small subset of neurons when making predictions. [SHK+14].



(a) Original neural network. (b) Neural network with dropout.

Figure 3.12: Example of dropout regularizer applied on a neural network.

3.13.4 Ensemble averaging

The idea behind a model averaging is to train k different models separately and make predictions by averaging individual predictions across all k models. Combining multiple models and averaging their predictions often performs better than training only one model with the same total capacity [PR91].

3.13.5 Dataset augmentation

A model trained on more data generalizes better, but in practice only a limited amount of data is available. However, in some problems it is possible to generate new samples in the dataset. New samples can be generated for example by adding artificial noise to existing samples or by applying transformations of the data – for example if we have a dataset consisting of bitmap images, we can generate new samples by rotating or resizing existing images.

CHAPTER 4

Feature extraction

As the dimensionality of a training dataset increases, the volume of the space in which a machine learning algorithm operates grows exponentially in volume which prevents to effectively identify patterns in the dataset. This issue is also referred to as24nota.ru a *curse of dimensionality* [Bel61]. It has been shown, that predictive power of machine learning model decreases as the dimensionality increases [Hug68]. The goal of dimensionality reduction is to overcome the cusrse of dimensionality by reducing the sample space while trying to preserve as much structure in the data as possible.

Assume that each sample x from the training dataset consist of n features, i.e. $x = (f_1, f_2, \dots, f_n)$. There are two types of dimensionality reduction approaches:

- **feature selection** – the dimensionality of each sample is reduced by selecting only a subset of $m \leq n$ features. Each high-dimensional input sample x is transformed into a feature vector: $y = (f_{i_1}, f_{i_2}, \dots, f_{i_m})$.
- **feature extraction** – the dimensionality is reduced by projecting high-dimensional samples into a feature space with less dimensions. Each high-dimensional input sample x is transformed into a feature vector: $y = (g_1(x), g_2(x), \dots, g_m(x))$.

Many dimensionality reduction approaches have been developed, however there is no clear evidence that any of them is superior to the other on all types of tasks. In fact, all dimensionality reduction techniques are equivalent if their performance is averaged over all possible problems. This phenomenon is called *No Free Lunch theorem* [WM97]). The performance of a machine learning algorithm always depends on the underlying structure that is assumed to be present in the training dataset – the same algorithm may perform well on one dataset, but poorly on another.

In this work, two unsupervised feature extraction techniques based on generative modeling will described in detail – namely *Variational Autoencoder* and *Generative Adversarial*

Network (GAN). These methods will be then compared with more traditional unsupervised dimensionality reduction techniques that are commonly used in various domains. Finally, it will be concluded that these approaches in some aspects outperform other compared methods when evaluated on the wafermap measurement data and might be suitable for a consideration in a practical application.

4.1 Variational Autoencoder

Suppose we are given a set of n samples $\mathbb{X} = \{x_1, x_2, \ldots, x_n\}$ and suppose these samples are generated by some random process that involves a unobserved (latent) continuous variable z. In this scenario, given the value of the latent variable z, each sample x is generated by the true (but unknown) data-generating conditional probability distribution, i.e. $x \sim p(x|z)$. This model is illustrated in Figure 4.1.

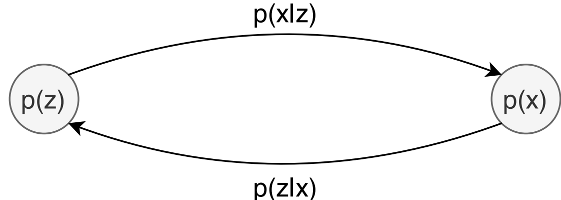


Figure 4.1: Schematic view of the generative model $p(x, z)$ that defines the relationship between available samples modeled by a random variable x and the latent variable z.

Furthermore, we also assume that the true data-generating probability distribution $p(x|z)$ can be approximated by a parametric model $p_\theta(x|z)$ characterized by a parameter θ. The generative model parametrized by θ is then given by:

$$p_\theta(x, z) = p_\theta(x|z)p_\theta(z) = p_\theta(z|x)p_\theta(x). \tag{4.1}$$

We make a strong assumption about the latent variable distribution $p_\theta(z)$ – for instance we assume that each latent variable z is normally distributed. So in such model, we know two things:

1. we have available data samples generated by the true (but unknown) distribution $p(x|z)$,
2. we fix the latent variable distribution $p(z)$ according to our strong assumption.

In order to extract features from the dataset, we would like to build an algorithm for an effective way of sampling from $p_\theta(z|x)$. A naive idea would be to apply a Bayes' rule as follows:

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}. \tag{4.2}$$

The $p(x)$ in Equation 4.2 cannot be omitted – such simplyfing step is possible only for parameter optimization problems, in this case we are interested in an exact value of z sampled from $p_\theta(z|x)$. The probability distribution $p(x)$ could be calculated by marginalizing-out the variable z as follows:

$$p_\theta(x) = \int p_\theta(x,z)dz = \int p_\theta(x|z)p_\theta(z)dz, \tag{4.3}$$

but this is intractable for real-world problems when we are given only a finite number of samples.

An approach suggested in [KW13] introduces a new conditional probability distribution $q_\phi(z|x)$ which is an approximation of $p_\theta(z|x)$. Parameters ϕ of the probability distribution $q_\phi(z|x)$ are then optimized by the Kullback-Leibler divergence between $p_\theta(z|x)$ and $q_\phi(z|x)$ as follows:

$$\mathcal{L}(x,z) = D_{KL}(q_\phi(z|x) \| p_\theta(z|x)). \tag{4.4}$$

The probability distribution function $q_\phi(z|x)$ will be referred to as an *encoder*, since it takes a data sample x and encodes it to a latent code z. Similarly, the probability distribution function $p_\theta(x|z)$ will be referred to as a *decoder*, since it takes a latent code z and generates the respective data sample x.

4.1.1 Evidence Lower Bound (ELBO)

Because our dataset $\mathbb{X}$ has been assumed to be sampled from the probability distribution $p(x|z)$ (i.e. not from $p(z|x)$), we are not able to optimize $\mathcal{L}$ directly using the Equation 4.4. Instead, we continue the derivation of the loss function $\mathcal{L}$:

$$\begin{aligned}
\mathcal{L}(x,z) &= \mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\Big] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\Big[\log \frac{q_\phi(z|x)p_\theta(x)}{p_\theta(x,z)}\Big] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}[-\log p_\theta(x,z) + \log q_\phi(z|x) + \log p_\theta(x)]
\end{aligned} \tag{4.5}$$

where the term $\log p_\theta(x)$ is a constant and hence it can be ommited from parameter optimization problems. This way, we get rid of the intractabile part of Equation 4.3. The new formula is known in literature as *Evidence Lower Bound (ELBO)* [HBWP13] and it will be denoted by $\mathcal{L}_{ELBO}$. ELBO can be further decomposed as follows:

$$
\begin{aligned}
\mathcal{L}_{ELBO}(x,z) &= \mathbb{E}_{z\sim q_\phi(z|x)}[-\log p_\theta(x,z) + \log q_\phi(z|x)] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\Big[-\log p_\theta(x|z) + \log \frac{q_\phi(z|x)}{p_\theta(z)}\Big] \\
&= \underbrace{H(q_\phi(z|x), p_\theta(x|z))}_{\text{reconstruction error}} + \underbrace{D_{KL}(q_\phi(z|x)\|p_\theta(z))}_{\text{regularizer}}.
\end{aligned}
\tag{4.6}
$$

Intuitively, the first term from the Equation 4.6 is a reconstruction error, i.e. the cross-entropy between the input to the encoder and the output of the decoder. The second term serves as a regularizer that measures the amount of information lost when $p_\theta(z)$ is used to approximate $q_\phi(z|x)$.

4.1.2 Reparametrization trick

Both the encoder $q_\phi(z|x)$ and the decoder $p_\theta(x|z)$ can be implemented as a neural network and trained with a backpropagation algorithm described in Section 3.12.2. The forward step of the backpropagation algorithm goes as follows:

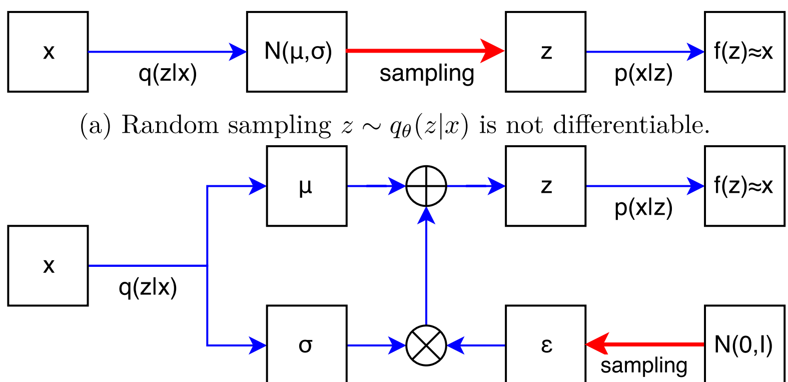
1. we sample a latent code z from the encoder $q_\phi(z|x)$ by providing a sample x from the training dataset,
2. the latent code z is then passed to the decoder $p_\theta(x|z)$ in order to decode it to an approximation of x.

However, the backward step in such a network is problematic, because the random sampling of the latent code z from the encoder $q_\phi(z|x)$ is a non-differentiable operation and hence we are not able to calculate a gradient for the backpropagation algorithm. A solution known as *reparametrization trick* has been suggested in [KW13].

The idea of this trick is to shift the randomness of the sampling operation $z \sim q_\phi(z|x)$ to a separate random variable $\epsilon \sim \mathcal{N}(0, I)$. Given a strong assumption that z is normally distributed, the latent code $\hat{z} \approx z$ can be then sampled deterministically as follows:

$$
\hat{z} = \mu + \sigma \odot \epsilon \tag{4.7}
$$

where ϵ is a random noise vector, $\odot$ is an element-wise product and vectors μ and σ are two parameters to be trained in the last layer of the encoder neural network. This trick is illustrated in Figure 4.2.



(a) Random sampling $z \sim q_\theta(z|x)$ is not differentiable.

(b) Deterministic sampling $\hat{z} = \mu + \sigma \odot \epsilon$ is differentiable if we shift the randomness into a separate random variable ϵ.

Figure 4.2: Reparametrization trick for an auto-encoding generative model. Blue lines represent differentiable operations for which we are able to calculate gradients in back-propagation algorithm. Red lines depict non-differentiable operations.

4.1.3 Implementation & experiments

As already discussed, distribution $p_\theta(z)$ and $q_\phi(z|x)$ from Equation 4.6 are both normally distributed. The regularization term of the ELBO then yields [1]:

$$
\begin{aligned}
D_{KL}(q_\phi(z|x) \| p_\theta(z)) &= \int q_\phi(z|x)(\log p_\theta(z) - \log q_\phi(z|x))dz \\
&= \frac{1}{2} \sum_{i=1}^{n} (1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2).
\end{aligned}
$$

The reconstruction error is given by:

$$
\begin{aligned}
H(q_\phi(z|x), p_\theta(x|z)) &= \mathbb{E}_{z \sim q_\phi(z|x)}[-\log p_\theta(x|z)] \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(\log y_i + (1 - x_i) \log(1 - y_i) \right)
\end{aligned}
$$

where x_i are the training samples provided to the encoder $p_\phi(z|x)$ and y_i are the respective outputs returned by the decoder $p_\theta(x|z)$.

An architecture of the neural network used for experiments in this work for training the variational autoencoder is depicted in Figure 4.3. The activation function used for hidden layers is *Rectified Linear Unit* (ReLU):

[1] The full derivation the KL-divergence between two multivariate Gaussian distribution functions is out of scope of this diploma thesis and is available in the original VAE paper [KW13]

$$s_{ReLU}(x) = \max(0, x). \tag{4.8}$$

The input of the encoder has been normalized to interval $(0, 1)$ as described in Chapter 2, hence also the output of the decoder should be on the same interval. For this reason, the last layer of the decoder uses sigmoid activation function:

$$s_{sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{4.9}$$

The activation function for the last layer of the encoder is a simple linear function. More discussion about the choice of activation functions is in Section 3.12.3.

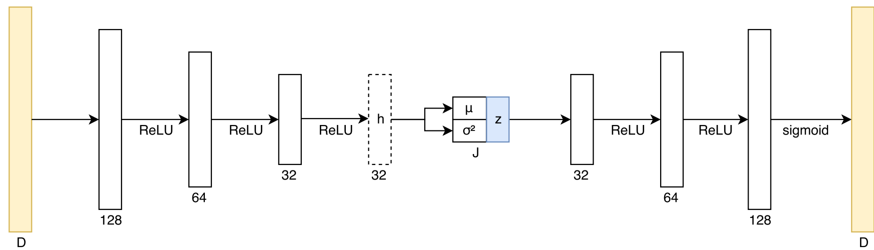


Figure 4.3: An architecture of the neural network used for experiments in this work to train the variational autoencoder.

The result of the Variational Autoencoder algorithm trained on the cleansed wafermap datataset is depicted in Figure 4.4.

4.2 Generative Adversarial Network (GAN)

Generative Adversarial Network (GAN) is a game-theoretic approach to generative modeling. Since the GAN was introduced a few years ago, this approach immediately gained a lot of attention from the machine learning research community. Several improvements or derived method based on GAN has been proposed some of them will be discussed in this section.

The basic idea of GANs is to model the optimization problem as a min-max game with two players competing against each other:

- **generator** – a function $G(\mathbf{z}; \theta_G)$ that takes an input vector $\mathbf{z} \sim p_{noise}$ and produces a fake sample $\mathbf{x} \sim p_{gen}$,
- **discriminator** – a function $D(\mathbf{x}; \theta_D)$ that takes an input vector $\mathbf{x}$ and returns a probability indicating whether the sample $\mathbf{x}$ is real (i.e. from the real dataset) or fake (i.e. produced by the generator).

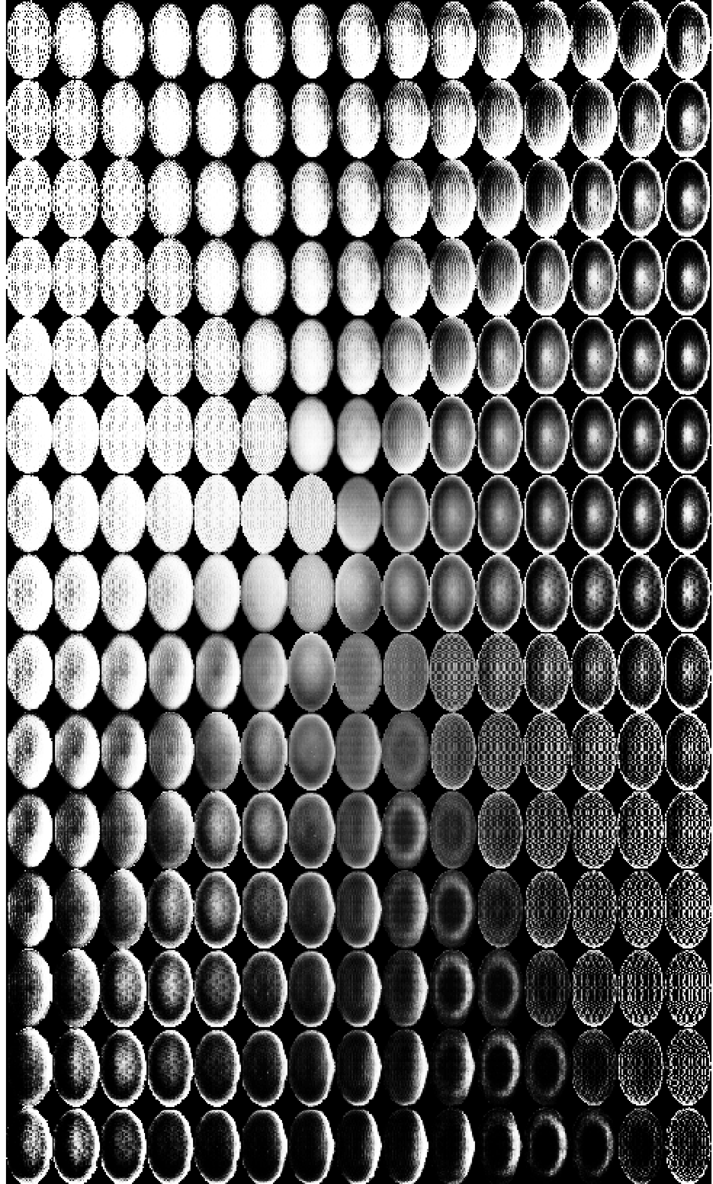


Figure 4.4: A visualization of the latent space learned by the Variational Autoencoder used in this work.

This situation of competing generator and discriminator is illustrated in Figure 4.5.

The objective of GAN can be formalized as a non-cooperative minimax game with two players optimizing the following problem [GPAM+14]:

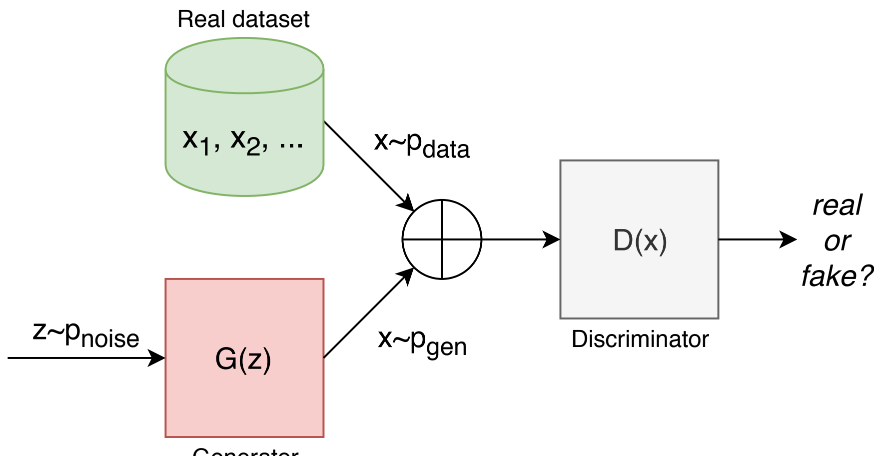


Figure 4.5: The simplest GAN architecture, where G is trained to generate new samples similar to the samples from the real dataset, while D is trained to recognize whether the provided sample is real or fake.

$$\min_{G} \max_{D} V(D, G) = \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_{noise}}[\log(1 - D(G(z)))] \tag{4.10}$$

where G and D are differentiable functions modeled as neural networks. The Equation 4.10 is useful for theoretical analysis of the GAN problem, however it does not perform well in practice because the gradient for effectively learning G with an iterative numerical computation might be too low [GPAM$^+$14]. As suggested in the GAN paper [GPAM$^+$14], instead of training G by minimizing $\mathbb{E}_{x \sim p_{noise}}[\log(1 - D(G(z)))]$, we can train G by maximizing $\mathbb{E}_{x \sim p_{noise}}[\log(D(G(z)))]$. The two loss functions for training the generator and the discriminator neural networks then become:

$$\begin{aligned}
\mathcal{L}_D^{GAN} &= \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_{noise}}[\log(1 - D(G(z)))] \\
\mathcal{L}_G^{GAN} &= \mathbb{E}_{z \sim p_{noise}}[\log D(G(z))]
\end{aligned}$$

4.2.1 Feature extraction with GAN

The aim of this chapter is to extract the most characteristic features from the wafer dataset samples, but the GAN in its simplest form is not capable to do this task. The generator can generate new photo-realistic samples, but since the only input to the generator is a random noise, we are not able to interpret what was actually generated. Similarly, the discriminator in a simple GAN is only trained to recognize real samples from fake samples without providing any additional information about the sample in question.

Fortunately, an information-theoretic extension to the GAN, known as as InfoGAN, has been proposed in [CDH+16]. InfoGAN is capable to encode descriptive properties of samples and to learn so called "*disentangled representation*" of the sample space in an unsupervised manner. While simple GAN can only generate new samples randomly, InfoGAN provides also a meaningful *latent code* that characterizes the sample.

The generator $G(z)$ in a simple GAN takes only one trivial code, i.e. a random noise vector $z \sim p_{noise}$. In case of InfoGAN, the input of the generator $G(z, c)$ is extended by a latent code $c \sim p_{latent}$. To ensure that the latent code c is meaningful, the loss function is extended by a regularizer term that maximizes the mutual information $I(c, G(z, c))$ between the latent code c and the generated sample $G(z, c)$ as follows:

$$
\begin{aligned}
I(c, G(z,c)) &= H(c) - H(c|G(z,c)) \\
&= H(c) + \mathbb{E}_{z\sim p_{noise}, c\sim p_{latent}}[\log p(c|G(z,c))].
\end{aligned} \tag{4.11}
$$

The $p(c|G(z, c))$ in Equation 4.11 is intractable the reasoning behind this intractability is similar to the case of Variational Autoencoder described in Section 4.1. As suggested in [CDH+16], we can create a new probability distribution $q(c|x)$ that is an approximation of $p(c|x)$ and derive the lower bound on the mutual information as follows:

$$
\begin{aligned}
I(c, G(z,c)) &= H(c) + \mathbb{E}_{z\sim p_{noise}, c\sim p_{latent}}\Big[\log \frac{p(c|G(z,c))q(c|x)}{q(c|G(z,c))}\Big] \\
&= H(c) + \mathbb{E}_{z\sim p_{noise}}\Big[\mathbb{E}_{c\sim p_{latent}}[q(c|G(z,c))] + \mathbb{E}_{c\sim p_{latent}}\Big[\log \frac{p(c|x)}{q(c|G(z,c))}\Big]\Big] \\
&= H(c) + \mathbb{E}_{z\sim p_{noise}}\Big[\mathbb{E}_{c\sim p_{latent}}[q(c|G(z,c))] + \underbrace{D_{KL}(p(c|G(z,c))\|q(c|G(z,c)))}_{\geq 0}\Big] \\
&\geq H(c) + \mathbb{E}_{z\sim p_{noise}, c\sim p_{latent}}[q(c|G(z,c))]
\end{aligned} \tag{4.12}
$$

where $H(c)$ is a constant and hence it can be omitted in a optimization problems. The distribution $q(c|x)$ is modeled as a neural network $Q(x)$ that shares most of its layers with the discriminator network $D(x)$ and hence the training of InfoGAN adds only a negligible computational cost compared to simple GAN.

Finally, the objective of the InfoGAN min-max game becomes:

$$
\min_G \max_D V_{InfoGAN}(D, G) = V(D, G) - \lambda I(c, G(z, c)) \tag{4.13}
$$

where λ is a regularization coefficient that controls the speed of latent code training.

4.2.2 Training GAN using Wasserstein distance

GANs are a novel machine learning approach with very active ongoing research, but this approach is also notoriously difficult to train [JLLWC18]. Although GAN appears to be a powerful machine learning method, stabilizing the training on the wafermap dataset was a challenging task. The InfoGAN as described in Section 4.2.1 works well with MNIST dataset [LC10] (a dataset of hand-written digits from 0 to 9), but the gradients of the network "explode" to unreasonable values within a few iterations when trained on the available wafer measurement dataset. We believe, that the reason behind such behaviour is the fact that the wafer samples are very similar to each other. While the samples from the MNIST dataset are digits of different shapes, in the wafer dataset almost all samples have the same circular shape and only slightly different patterns. Hence, the wafer samples are located only on a very small part of the high-dimensional sample space which makes the training unstable.

One of the problems of GAN training lies in the KL-divergence loss function used for the training. To recall, KL-divergence is defined as $D_{KL}(p, q) = \mathbb{E}[\log \frac{p(x)}{q(x)}]$ which means that its value is zero when $p(x) = q(x)$ everywhere. On the other hand, in cases where $p(x)$ is close to zero and $q(x)$ is a large number, the value of KL-divergence can be be extremely large. This phenomenon is called "disjoint supports of distributions" and there is an evidence that this may often happend during the GAN training.

Another problem is the competitive nature of the min-max game – since the generator and discriminator are non-cooperative players, after the generator updates weights of the network, the discriminator can just "revert" them in a next iteration. This may lead to an oscillating behaviour and non-convergence.

These issues are contributing to the instability of the GAN training and are discussed in detail in [BALO17]. There have been proposed methods of adding an instance noise to training samples in order to stabilize the training [SCT$^+$17] or adding a regularization penalty to the GAN loss function [RLNH17]. However, experiments with these methods for training the InfoGAN on wafermap dataset were rather unsuccessful.

An interesting way of stabilizing the training of the GAN was proposed in [ACB17b] by replacing the problematic KL-divergence loss function by a distance measure called *Earth-Mover distance (or Wasserstein Metric)*. Using this method, we were able to stabilize the training on the wafer dataset.

Earth-Mover distance

Assume we have two discrete distributions p and q. For simplicity, assume that these two distributions consit of "boxes" of unit size that we want to transport from p to q and obtain a transport plan γ. This situation is depicted on Figure 4.6.

However, there might be more possible travel plans for same two distributions p and q. We can assign a cost $|x - y|$ to each move of a box from position x to y. Figure 4.7 depicts two travel plans over the same two distributions, but each plan with a different

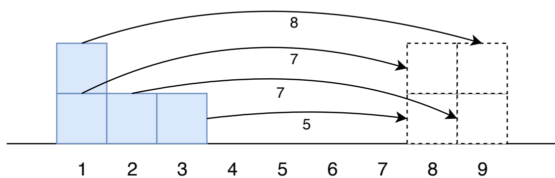
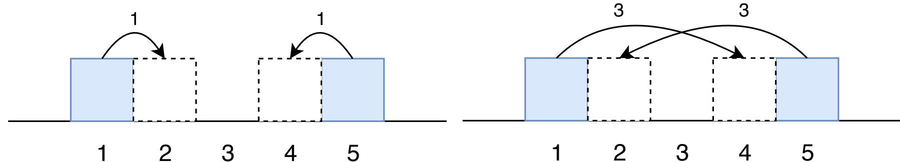


Figure 4.6: Transport plan $\gamma = \{(1,9),(1,8),(2,9),(3,8)\}$ with total cost $8+7+7+5 = 27$.

total cost. We are interested in finding such a transport plan, that has the minimal total transport cost – the cost of such transport plan is called *Earth-Mover distance*.



(a) Transport plan $\gamma = \{(1,2),(5,4)\}$ with total cost $= 1+1 = 2$.

(b) Transport plan $\gamma = \{(1,4),(5,2)\}$ with total cost $= 3+3 = 6$.

Figure 4.7: The same two distributions can have travel plans with different total cost. The Earth-Mover distance is the optimal total cost among all possible travel plans.

Definition 20 (Earth-Mover distance)**.** *Earth-Mover distance (or Wasserstein metric) is a distance measure $W(p,q)$ between two distributions p and q defined as follows:*

$$W(p,q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|]$$

where inf *is the infimum (greatest lower bound) and $\Pi(p,q)$ is a set of all possible transport plans between p and q.*

Optimizing Earth-Mover distance with a neural network directly from the Definition 20 is intractable. In [ACB17b] it is proposed to apply the Kantorovich-Rubenstien duality to obtain an equivalent definition as follows:

$$W(p,q) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)] \tag{4.14}$$

where sup is the supremum (the opposite of inf, i.e. the least upper bound) and f is a 1-Lipschitz function.

Definition 21 (1-Lipschitz function)**.** *Function f is called 1-Lipschitz function if it*

follows the constraint:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|.$$

Wasserstein GAN (WGAN & WGAN-GP)

To recall, the loss functions of the generator and the discriminator in a simple GAN is as follows:

$$\begin{aligned} \mathcal{L}_D^{GAN} &= \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_{noise}}[\log(1 - D(G(z)))] \\ \mathcal{L}_G^{GAN} &= \mathbb{E}_{z \sim p_{noise}}[\log D(G(z))] \end{aligned} \tag{4.15}$$

The loss functions of Wasserstein GAN (WGAN) becomes:

$$\begin{aligned} \mathcal{L}_D^{WGAN} &= \mathbb{E}_{x \sim p_{data}}[D(x)] - \mathbb{E}_{z \sim p_{noise}}[D(G(z))] \\ \mathcal{L}_G^{WGAN} &= \mathbb{E}_{z \sim p_{noise}}[D(G(z))] \end{aligned} \tag{4.16}$$

where D is called a *critic*, i.e. a 1-Lipschitz function implemented as a neural network. To enforce the 1-Lipschitz constraint on D, autors of the original WGAN paper [ACB17b] suggested to restrict all weight values in the neural network to an interval $(-c, c)$ (so called *weight clipping*) as follows:

$$w \leftarrow \text{clip}(w, -c, c) = \begin{cases} -c & w \leq -c \\ c & w \geq c \\ w & \text{otherwise} \end{cases} \tag{4.17}$$

where c is a chosen constant. As authors mention, the weight clipping is a terrible way of enforcing the 1-Lipschitz constraint, because it reduces the capacity of the network.

An alternative approach to ensure 1-Lipschitz constrain was suggested in [GAA$^+$17b]. A function is 1-Lipschitz if and only if it has gradient with the norm at most 1 everywhere. Instead of clipping weights to a fixed interval, authors propose to impose a penalty on gradients as follows:

$$\mathcal{L}_D^{WGAN-GP} = \underbrace{\mathbb{E}_{\tilde{x} \sim p_{data}}[D(\tilde{x})] - \mathbb{E}_{z \sim p_{noise}}[D(G(z))]}_{\text{WGAN critic loss}} + \underbrace{\lambda \mathbb{E}_{\hat{x} \sim p_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{gradient penalty}} \tag{4.18}$$

where $p_{\hat{x}}$ is a distribution laying between p_{data} and p_{noise}. Hence, the $\hat{x} \sim p_{\hat{x}}$ is a randomly weighted average of real and fake samples and can be sampled as $\hat{x} = t\tilde{x} + (1-t)x$ for some t s.t. $0 \leq t \leq 1$ (in implementation in this work $t = 0.5$).

Implementation & experiments

Combining the WGAN and InfoGAN objectives, the following loss functions for training the discriminator and generator has been used in this work to extract low-dimensional latent codes from high-dimensional wafermap dataset:

$$\mathcal{L}_{\mathcal{D}} = \mathbb{E}_{\tilde{x}\sim p_{data}}[D(\tilde{x})] - \mathbb{E}_{z\sim p_{noise}}[D(G(z,c))] + \mathbb{E}_{\hat{x}\sim p_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2] - I(c, G(z,c))$$
$$\mathcal{L}_{\mathcal{G}} = \mathbb{E}_{z\sim p_{noise}}[D(G(z,c))] \tag{4.19}$$

The architecture of the neural network is depicted in Figure 4.8

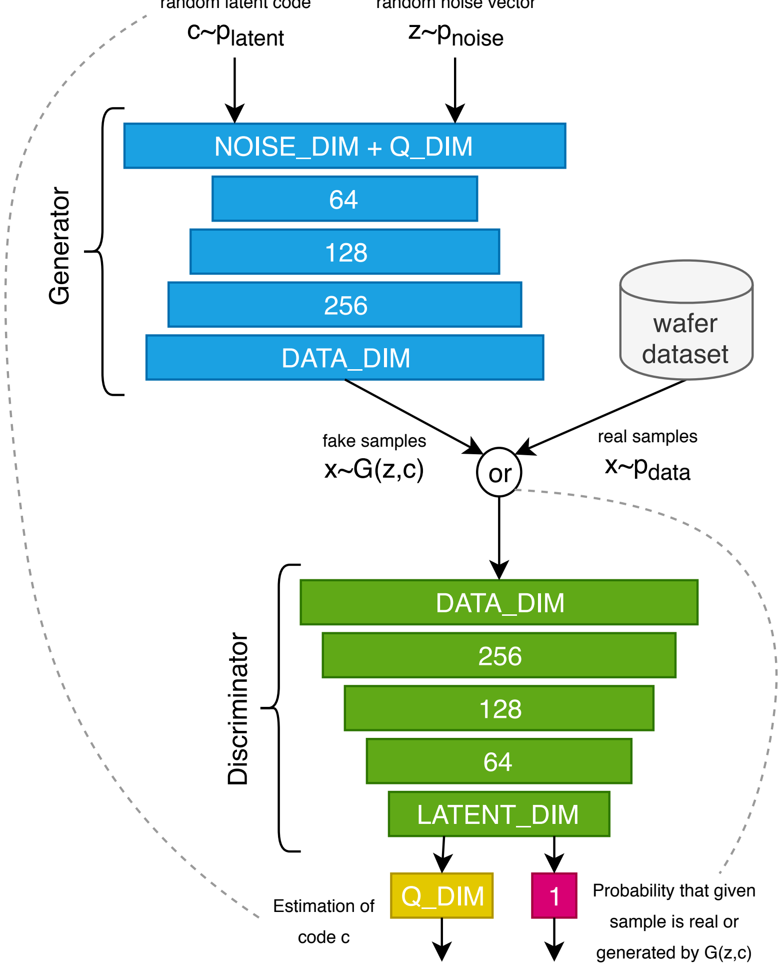


Figure 4.8: Architecture of the Information Maximizing Generative Adversarial Network (InfoGAN) used in this work.

The implementation in this work uses a normally distributed latent code z which yields the mutual information as follows:

$$
\begin{aligned}
\log I(c, G(z,c)) &\geq \log\Big(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\Big) \\
&= \underbrace{\log\frac{1}{\sqrt{2\pi}}}_{const.} - \log\sigma - \frac{(x-\mu)^2}{2\sigma^2} \\
&\geq -\log\sigma - \frac{1}{2}\Big(\frac{x-\mu}{\sigma}\Big)^2.
\end{aligned}
\tag{4.20}
$$

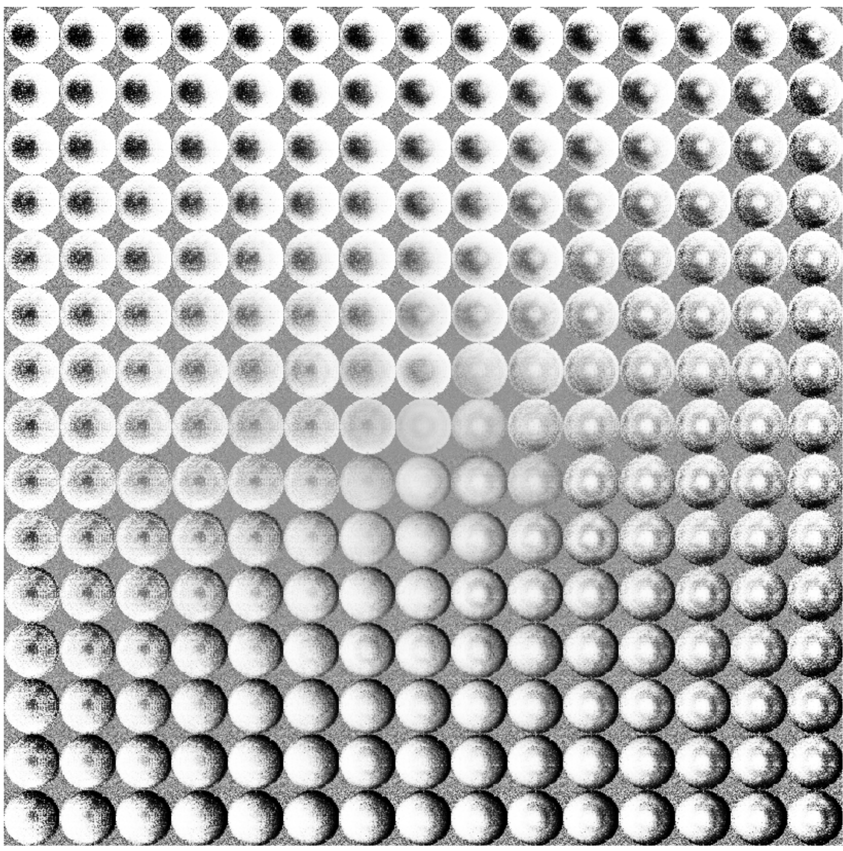


Figure 4.9: Visualization of the InfoGAN latent space trained on the wafermap dataset.

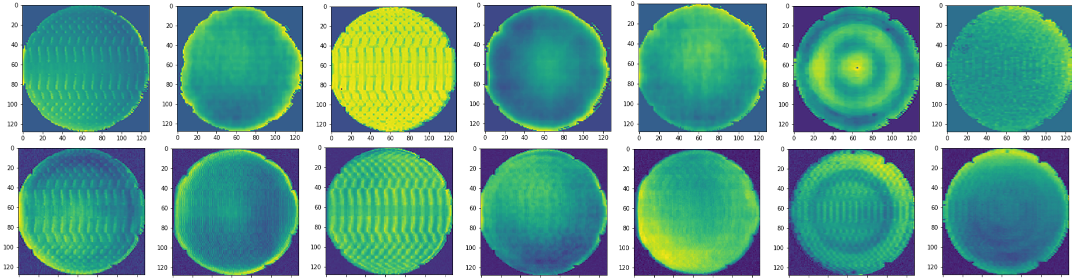


Figure 4.10: An few samples from the real wafermaps dataset (top row) and respective fake samples (bottom row) generated by the GAN model trained on the same dataset. It can be observed, that the quality of the genrated wafermaps is plausible.

CHAPTER 5

Clustering

Clustering or *cluster analysis* of a data is a process of grouping similar data points together according to some similarity metric. There are several different approaches to clustering and two of them will be discussed in this chapter – namely, *k-means* as a representative of a centroid-based methods and *hierarchical agglomerative clustering* as a representative of a connectivity-based methods. These two methods will be used to cluster the features obtained in Chapter 4.

5.1 k-means clustering

This clustering algorithm starts by randomly initializing a set of k centroids $\{m_1, m_2, \ldots, m_k\}$. The centroids form a Voronoi diagram and the data points closest to a centroid m_i are then considered to belong to cluster i. The algorithm continues by iteratively moving the centroids towards the mean of the cluster by repeating of two alternating steps [Mac03]:

1. **Data assignment step.** For each data point $x \in \mathbb{X}$, the centroid m_i with the smallest Euclidean distance $\|x - m_i\|^2$ among all other centroids is assigned to the Voronoi cell S_i. Mathematically, the elements of each set S_i are assigned as follows:

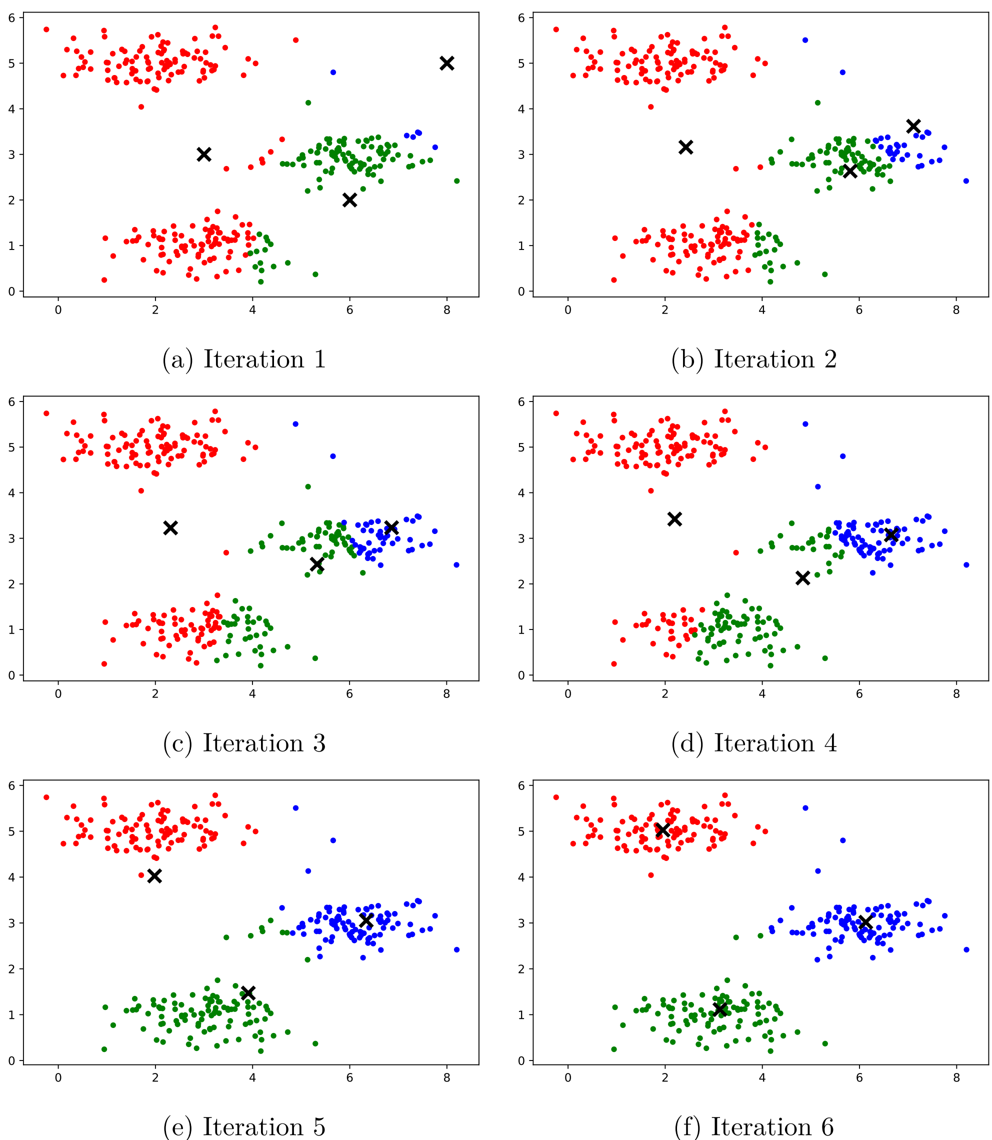
$$S_i := \{x : \|x - m_i\|^2 \leq \|x - m_j\|^2, \forall x \in \mathbb{X}, \forall j : 1 \leq j \leq k\}. \tag{5.1}$$

2. **Centroid update step.** The new position of each centroid m_i is calculated as the mean of all data points in cluster S_i:

$$m_i := \mathbb{E}_{x \sim S_i}[x] = \frac{1}{|S_i|} \sum_{x \in S_i} x. \tag{5.2}$$

The algorithm usually terminates when the positions of centroids are no longer changing, but since the convergence is not guaranteed [HW79], some other stopping criteria can be

also in place (for example stopping after a certain number of iterations). An example of the k-means algorithm runtime is depicted in Figure 5.1.



(a) Iteration 1 (b) Iteration 2

(c) Iteration 3 (d) Iteration 4

(e) Iteration 5 (f) Iteration 6

Figure 5.1: Example of k-means algorithm runtime on 2-dimensional data.

5.2 Hierarchical agglomerative clustering

This algorithm starts by creating a new single-element cluster for each data point, i.e. the initial clusters are $\{x_1\}, \{x_2\}, \ldots, \{x_n\}$. Clusters are then iteratively merged into bigger clusters by pair-wise comparisons with each other based on some distance metric

$d(A, B)$. The distance metric for single-element clusters $\{x\}$ and $\{y\}$ is a simple Euclidean distance:

$$d(\{x\}, \{y\}) = \|x - y\|^2. \tag{5.3}$$

As two clusters P and Q are merged into a bigger cluster $P \cup Q$, the algorithm defines the distance (known as *Lance-Williams formula*) between the new cluster $P \cup Q$ and some other cluster R as follows:

$$d(P \cup Q, R) = \alpha d(P, R) + \beta d(Q, R) + \gamma d(P, Q) + \delta d(R, Q) \tag{5.4}$$

where $\alpha, \beta, \gamma, \delta$ are parameters of the clustering algorithm that may be either constants or depend on the size of the clusters. Some commonly used values of these parameters are for example:

- **Single linkage.** Measures the minimum distance between elements of each cluster:

$$\begin{aligned} \alpha &= 1/2 \\ \beta &= 1/2 \\ \gamma &= 0 \\ \delta &= -1/2 \end{aligned} \tag{5.5}$$

- **Complete linkage.** Measures the maximum distance between elements of each cluster:

$$\begin{aligned} \alpha &= 1/2 \\ \beta &= 1/2 \\ \gamma &= 0 \\ \delta &= 1/2 \end{aligned} \tag{5.6}$$

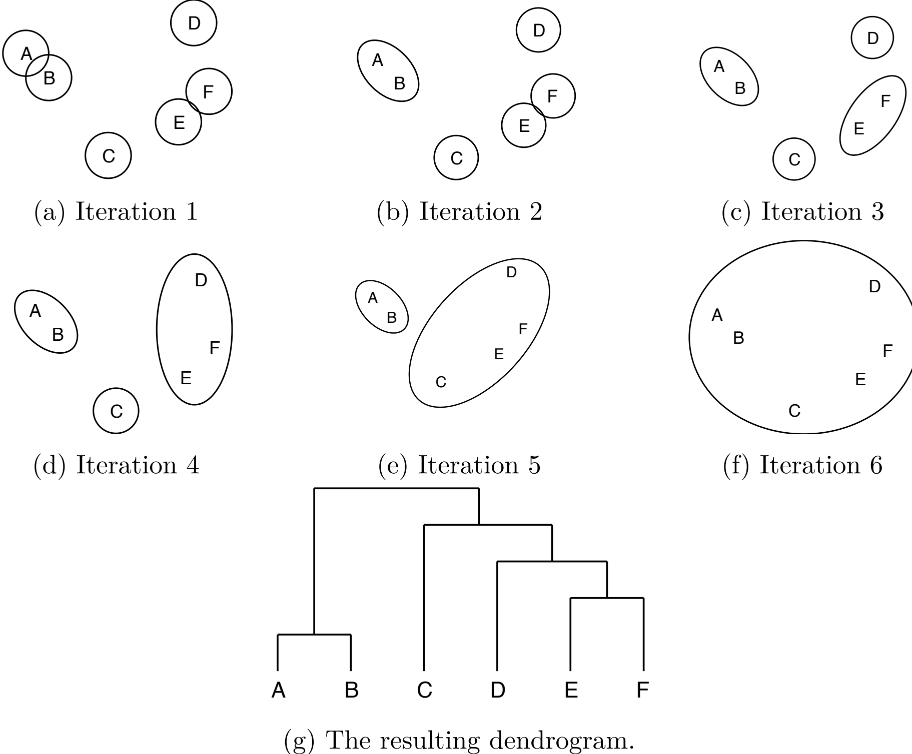
- **Average linkage.** Measures the mean distance between elements of each cluster:

$$\begin{aligned} \alpha &= \frac{|P|}{|P| + |Q|} \\ \beta &= \frac{|Q|}{|P| + |Q|} \\ \gamma &= 0 \\ \delta &= 0 \end{aligned} \tag{5.7}$$

- **Ward's minimum variance method.** Minimizes the total within-cluster variance:

$$
\begin{aligned}
\alpha &= \frac{|P| + |R|}{|P| + |Q| + |R|} \\
\beta &= \frac{|Q| + |R|}{|P| + |Q| + |R|} \\
\gamma &= -\frac{|R|}{|P| + |Q| + |R|} \\
\delta &= 0
\end{aligned}
\tag{5.8}
$$

Ward's minimum variance has been used for experiments in this work, implemented in *SciPy Toolkits* [JOP+]. An example runtime of a hierarchical clustering is depicted in Figure 5.2. The outcome of a hierarchical clustering can be also visualized in form of a *dendrogram* – a diagram representing the merging of clusters.



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

(d) Iteration 4 (e) Iteration 5 (f) Iteration 6

(g) The resulting dendrogram.

Figure 5.2: An example runtime of a hierarchical agglomerative clustering algorithm on 2-dimensional data.

5.3 Silhouette coefficient

In order to compare performances of different clustering methods on different datasets, it's important to define a performance measure. One of such measures is an average intra-cluster distance called *Silhouette coefficient*. [Rou87]

We denote $C(x)$ the cluster containing data point x. For each data point $x \in C(x)$, we define the *average* distance $a(x)$ to all other data points within the same cluster $C(x)$ as follows:

$$a(x) = \frac{1}{|C(x)| - 1} \sum_{y \in C(x), x \neq y} d(x, y) \tag{5.9}$$

where $d(x, y)$ is a distance between two data points x and y within the same cluster $C(x)$. We also define also the *smallest average* distance $b(x)$ between x and all other points from different clusters $C(y)$ as follows:

$$b(x) = \min_{x \neq y} \frac{1}{|C(y)|} \sum_{y \in C(y)} d(x, y) \tag{5.10}$$

Finally, the Silhouette measure $\text{sil}(x)$ for each data point x is then defined as follows:

$$\text{sil}(x) = \begin{cases} 1 - a(x)/b(x) & \text{if } a(x) < b(x) \\ 0 & \text{if } a(x) = b(x) \\ b(x)/a(x) - 1 & \text{if } a(x) > b(x) \end{cases} \tag{5.11}$$

The average Silhouette measure over all data points is known as Silhouette coefficient (or Silhouette score). The value of this coefficient lies between -1 and 1, where values close to 1 indicate a good clustering performance with well separated clusters, while values close to -1 indicate poorly separated clusters.

5.4 Experiments

The experiments consist of comparing VAE and InfoGAN feature extraction methods discussed in detail in this diploma thesis with the following feature extraction methods:

- Principal Component Analysis (PCA)
- Non-negative Matrix Factorization (NMF)
- Independent Component Analysis (ICA)
- Singular Value Decomposition (SVD)

The above-mentioned methods are already implemented in SciPy library [JOP+] and a detailed description is out of scope of this diploma thesis.

The comparison of k-means clustering and hierarchical agglomerative clustering on feature latent space obtained by these methods has been measured by Silhouette coefficients (Figure 5.5) as well as by a visual inspection of two dimensional feature space (Figure 5.4) or assignment of a few samples to their respective clusters (Figure 5.3).

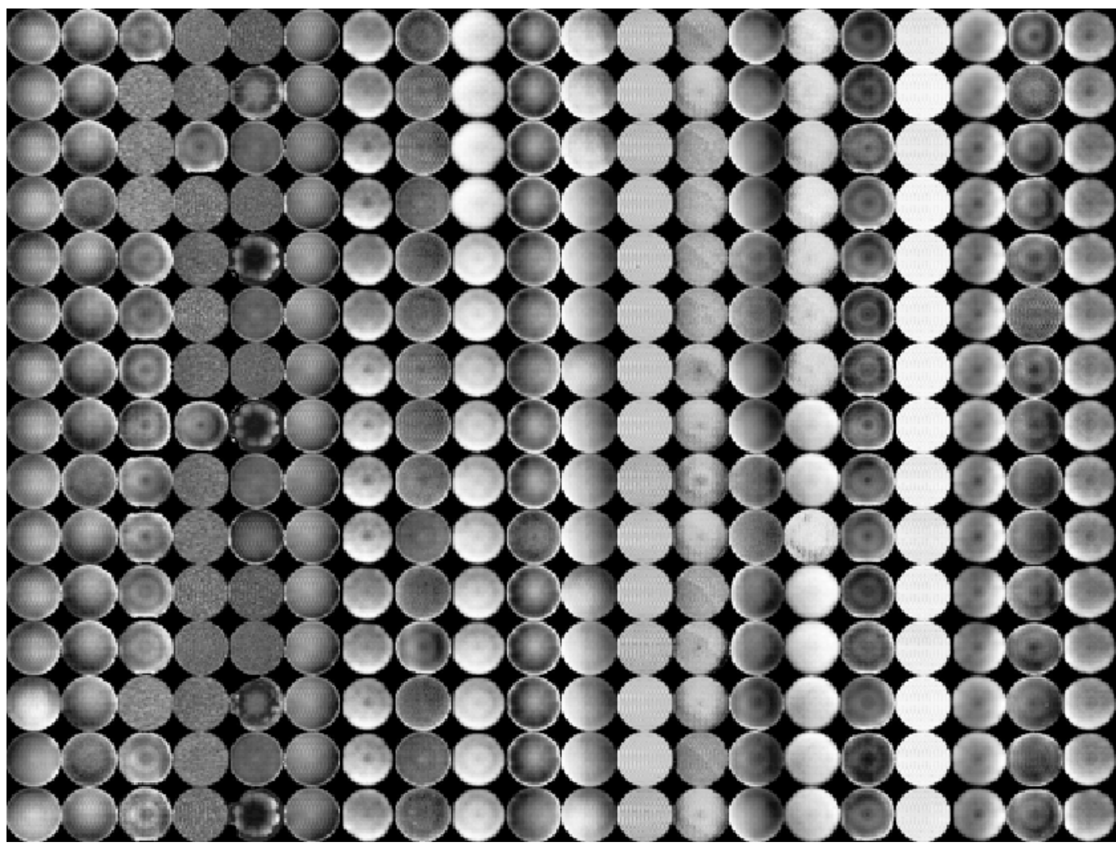
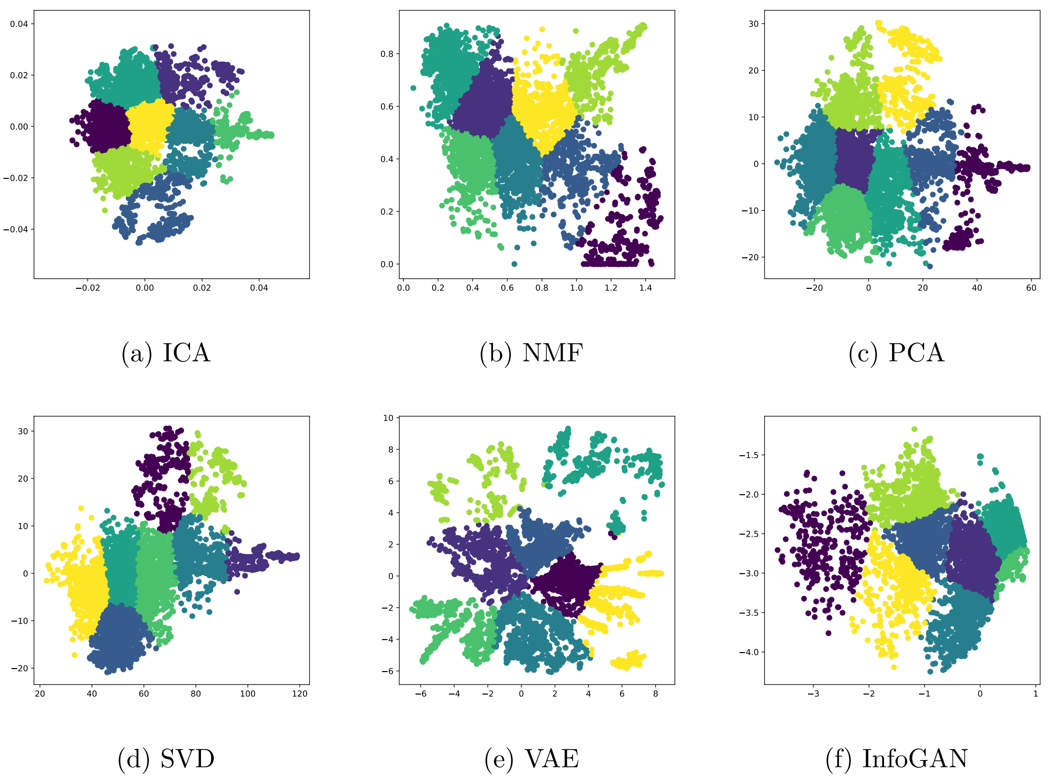


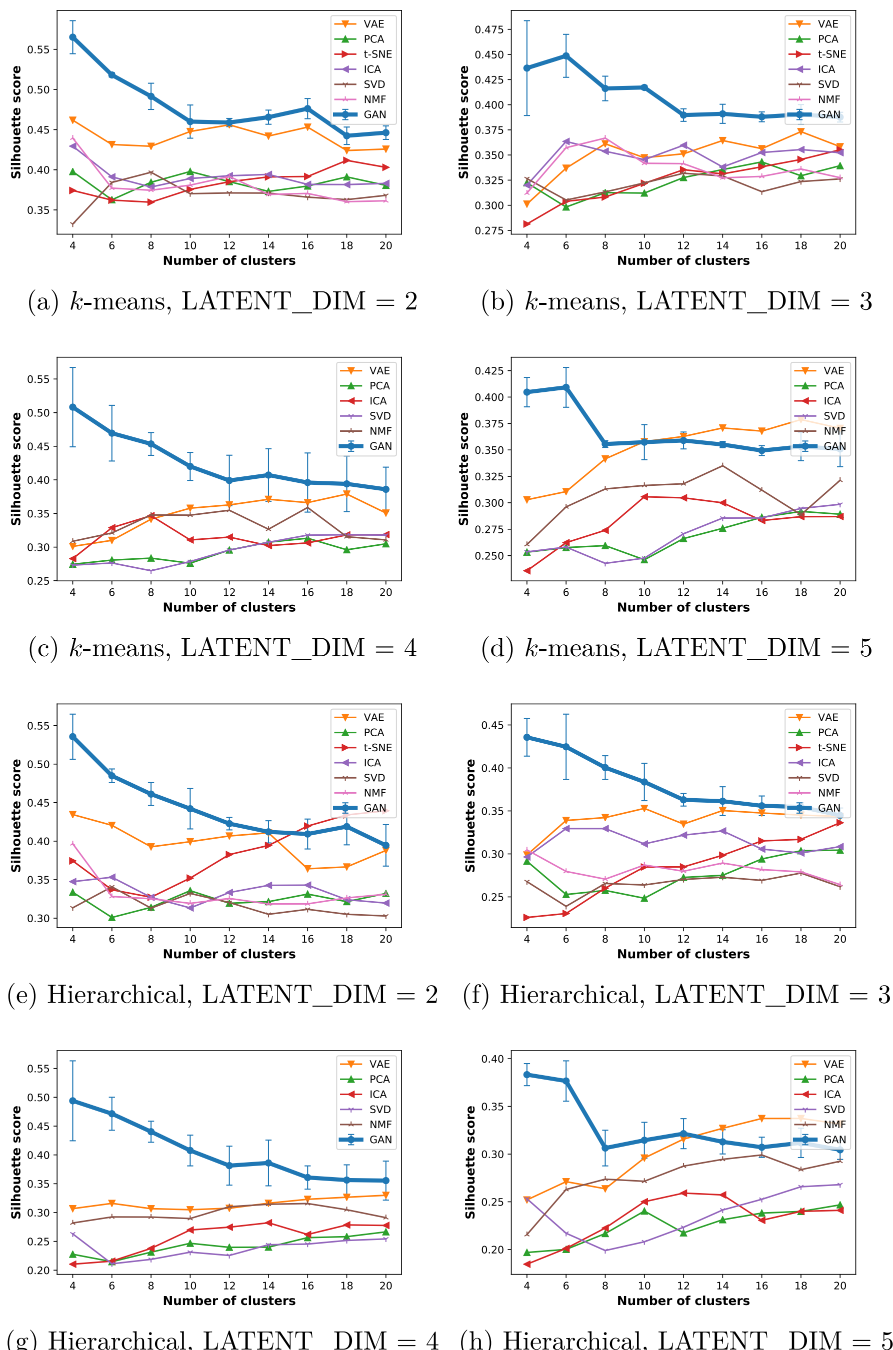
Figure 5.3: A few samples that belong to the same k-means clusters for $k = 20$. The feature latent space used for the clustering has been obtain using the VAE method described in this thesis.



(a) ICA (b) NMF (c) PCA

(d) SVD (e) VAE (f) InfoGAN

Figure 5.4: Visualization of k-means clustering (for $k = 8$) of a two dimensional latent feature space obtained by different machine learning algorithms.



(a) k-means, LATENT_DIM = 2 (b) k-means, LATENT_DIM = 3

(c) k-means, LATENT_DIM = 4 (d) k-means, LATENT_DIM = 5

(e) Hierarchical, LATENT_DIM = 2 (f) Hierarchical, LATENT_DIM = 3

(g) Hierarchical, LATENT_DIM = 4 (h) Hierarchical, LATENT_DIM = 5

Figure 5.5: Evaluation of k-means clustering and agglomerative hierarchical clustering performance in terms of intra-cluster distance measured by Silhouette coefficients. The features obtained by VAE and GAN methods described in this work performs better than other compared feature extraction methods in majority of cases.

CHAPTER 6

Conclusion

Semiconductor manufacturing processes are prone to production errors. It is assumed that that manufacturing equipment that causes production deviations leaves characteristic patterns on final wafer lots. An effective detection of patterns in wafermap measurement data evaluated after a certain number of production steps could be utilized for discovering which processing step is causing production deviations and possibly even to take an automatic corrective action. This diploma thesis has described machine learning approaches for automated detection of different patterns in a wafermap dataset in an unsupervised way.

The first part of this diploma thesis has discussed preprocessing steps that has been performed in order to cleanse the available raw wafer measurement dataset. Each sample from the dataset has been treated as a grayscale image. Outliers from these images have been removed using a *median-based Z-score* filtering method and small holes after this process have been closed with a hole-closing algorithm based on binary morphology followed by an inpainging algorithm. Each image has been smoothened by a median filtering algorithm and normalized to an interval $[0, 1]$.

The cleaned wafer dataset has been then used as an input for a feature extraction task. This part of the diploma thesis has been the main focus of the work. Some important machine learning preliminaries has been described. Based on these preliminaries, two generative modeling deep learning algorithms have been described in detail, namely *Variational Autoencoder (VAE)* and *Information Maximizing Generative Adversarial Network (InfoGAN)*. Especially challenging task has been the stabilization of GAN training using a Wasserstein metric. It has been shown that generative models can be used not only for feature extraction, but also for generating new samples given a feature vector – this has been demonstrated on a visualization of a two-dimensional feature space.

Finally, the last part of this diploma thesis has described two simple clustering algorithms, namely *k-means clustering* and *hierarchical agglomerative clustering*. The performance

of the feature extraction has been measured by assigning features to concrete clusters and calculating the intra-cluster distance known as *Silhouette score* as well as by a visual inspection of the feature space. Feature extraction using VAE and InfoGAN has been compared to other commonly used feature extraction methods.

This diploma thesis has demonstrated that deep generative models could be useful for extracting the most characteristic features from wafer measurement dataset and such methods may in some cases even outperform more traditional discriminative methods.

List of Figures

1.1 High-level overview of an automated improvement of a wafer production process. 3

2.1 Overall wafer preprocessing procedure. 8
2.2 Commonly used discrete structural elements with the origin $X = (0, 0)$. 9
2.3 Example of dilation with a 4-neighborhood structural element. 10
2.4 Example of erosion with a 4-neighborhood structural element. 10
2.5 Example of dilation and erosion on a binary image. 12
2.6 Close operation vs. hole filling algorithm. 13
2.7 Outliers removal using a simple and double-sided MAD-based method. 15
2.8 Comparison of IQR-based and MAD-based outlier removal methods. 17
2.9 Median filtering algorithm. 18
2.10 The result of wafer preprocessing. 18

3.1 Machine learning paradigms. 20
3.2 Implication graph of convergence modes. 24
3.3 Bias-variance trade-off. 27
3.4 Generalization issues demonstrated on an instance of a regression problem. 27
3.5 The entropy of two random variables X (red) and Y (blue). 33
3.6 Stochastic Gradient Descent (SGD) algorithm. 37
3.7 The effect of a learning rate (LR) on the performance of a SGD algorithm. 38
3.8 Comparison of a biological and artificial neuron. 40
3.9 An example of a feed-forward neural network. 40
3.10 L_p-norm regularizers. 45
3.11 Geometric interpretation of L_p-norm regularizers. 45
3.12 Example of dropout regularizer applied on a neural network. 46

4.1 Schematic view of the generative model. 48
4.2 Reparametrization trick. 51
4.3 Variational Autoencoder architecture. 52
4.4 Latent space of a variational autoencoder. 53
4.5 Generative Adversarial Network architecture. 54
4.6 Transport plan $\gamma = \{(1, 9), (1, 8), (2, 9), (3, 8)\}$ with total cost $8+7+7+5 = 27$. 57
4.7 Different transport plans for the same two distributions. 57

4.8 Architecture of the Information Maximizing Generative Adversarial Network (InfoGAN) used in this work. 59
4.9 Visualization of the InfoGAN latent space trained on the wafermap dataset. 60
4.10 GAN-generated samples. 61

5.1 Example of k-means algorithm runtime on 2-dimensional data. 64
5.2 An example runtime of a hierarchical agglomerative clustering algorithm on 2-dimensional data. 66
5.3 Clustered VAE features. 68
5.4 Visualization of k-means clustering on features obtained by different ML algorithms. 69
5.5 Silhouette score of k-means clustering and agglomerative hierarchical clustering. 70

List of Tables

3.1 An overview of commonly used activation functions. 43

List of Algorithms

2.1 Dilation algorithm of image A by structure S. 11
2.2 Region filling algorithm for binary bitmap. 12
2.3 MAD-based outlier detection algorithm. 16

3.1 Stochastic Gradient Descent (SGD) with mini-batches. 36

List of Symbols

This section contains an overview of symbols commonly used in this diploma thesis. The notation is adapted mostly from the book “Deep Learning” [GBC16].

Mathematical objects

a	A scalar or a vector
$\mathbf{a}$	A vector
A	A matrix or a set
$\mathbf{A}$	A matrix
$\mathbb{A}$	A set
$f : \mathbb{X} \to \mathbb{Y}$	A function from domain $\mathbb{X}$ to domain $\mathbb{Y}$
$f(x; \theta)$	A value of function f in x, parametrized by θ

Binary morphology

$A \oplus S$	Dilation of image A by structure S
$A \ominus S$	Erosion of image A by structure S
$\hat{A}$	Reflexion of image A
$(A)_z$	Translation of image A by z

Calculus

$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\int f(x)dx$	Definite integral over the whole domain of x
$\nabla_x y$	Gradient of y with respect to x

Probability & information theory

$P(x)$	Probability distribution over a discrete variable
$p(x)$	Probability distribution over a continuous variable
$x \sim p(x)$	Random variable x sampled from $p(x)$
$\mathbb{E}_{x \sim p}[f(x)]$	Exprected value of $f(x)$ with repect to p
$D_{KL}(P \mid Q)$	Kullback-Leibler divergence of P and Q
$P(X \mid Y)$	Conditional probability
$P(X, Y)$	Joint probability
$H(X)$	Entropy of the random variable X
$H(X \mid Y)$	Conditional entropy
$H(X, Y)$	Joint entropy
$H_X(Y)$	Cross-entropy
$I(X)$	Self-information
$I(X, Y)$	Mutual information
$\mathcal{N}(x; \mu, \Sigma)$	Normal distribution over x with mean μ and covariance Σ

Bibliography

[ACB17a] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN, March 2017.

[ACB17b] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[BALO17] Léon Bottou, Martín Arjovsky, David Lopez-Paz, and Maxime Oquab. Geometrical insights for implicit generative modeling. In *Braverman Readings in Machine Learning. Key Ideas from Inception to Current State - International Conference Commemorating the 40th Anniversary of Emmanuil Braverman's Decease, Boston, MA, USA, April 28-30, 2017, Invited Talks*, pages 229–268, 2017.

[BB08] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, pages 161–168. NIPS Foundation (http://books.nips.cc), 2008.

[Bel61] Richard E. Bellman. *Adaptive Control Processes: A Guided Tour.* MIT Press, 1961.

[BS95] L. Breaux and B. Singh. Automatic defect classification system for patterned semiconductor wafers. In *Proceedings of International Symposium on Semiconductor Manufacturing*, pages 68–73, Sep 1995.

[BvdG11] Peter Bhlmann and Sara van de Geer. *Statistics for High-Dimensional Data: Methods, Theory and Applications.* Springer Publishing Company, Incorporated, 1st edition, 2011.

[CCD+16] Xi Chen, Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In D. D. Lee,

U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances In Neural Information Processing Systems 29*, pages 2172–2180. Curran Associates, Inc., 2016.

[CDH+16] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2172–2180. Curran Associates, Inc., 2016.

[CLCJ09] Chuan-Yu Chang, ChunHsi Li, Jia-Wei Chang, and MuDer Jeng. An unsupervised neural network approach for automatic semiconductor wafer defect inspection. *Expert Systems with Applications*, 36(1):950 – 958, 2009.

[CLYYDY] F.L. Chen, Sheng-Che Lin, K. Yih-Yuh Doong, and K.L. Young. Logic product yield analysis by wafer bin map pattern recognition supervised neural network. *2003 5th International Conference on ASIC. Proceedings (IEEE Cat. No.03TH8690).*

[CM10] Charles K. Chui and H.N. Mhaskar. Mra contextual-recovery extension of smooth functions on manifolds. *Applied and Computational Harmonic Analysis*, 28(1):104–113, Jan 2010.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley-Interscience, New York, NY, USA, 1991.

[CT09] Li-Chang Chao and Lee-Ing Tong. Wafer defect pattern recognition by multi-class support vector machines by using a novel defect cluster index. *Expert Systems with Applications*, 36(6):10158 – 10167, 2009.

[Duv99] Frederic Duvivier. Automatic detection of spatial signature on wafermaps in a high volume production. In *Proceedings of the 14th International Symposium on Defect and Fault-Tolerance in VLSI Systems*, DFT ’99, pages 61–, Washington, DC, USA, 1999. IEEE Computer Society.

[GAA+17a] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

[GAA+17b] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5767–5777. Curran Associates, Inc., 2017.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[GBD92] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Comput.*, 4(1):1–58, January 1992.

[GJP95] Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7:219–269, 1995.

[GJY11] Dongdong Ge, Xiaoye Jiang, and Yinyu Ye. A note on the complexity of lp minimization. *Mathematical Programming*, 129(2):285–299, Oct 2011.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[HBWP13] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic variational inference. *J. Mach. Learn. Res.*, 14(1):1303–1347, May 2013.

[HK70] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67, 1970.

[Hug68] G. Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory*, 14(1):55–63, January 1968.

[HW79] JA Hartigan and MA Wong. Algorithm AS 136: A K-means clustering algorithm. *Applied Statistics*, pages 100–108, 1979.

[IH93] B. Iglewicz and D.C. Hoaglin. *How to Detect and Handle Outliers*. ASQC basic references in quality control. ASQC Quality Press, 1993.

[JLLWC18] Kevin J Liang, Chunyuan Li, Guoyin Wang, and Lawrence Carin. Generative adversarial network training is a continual learning problem. 11 2018.

[JOP+] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed <today>].

[KBP07] A. Khireddine, K. Benmahammed, and W. Puech. Digital image restoration by wiener filter in 2d case. *Advances in Engineering Software*, 38(7):513–516, Jul 2007.

[KHHW13] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry ; Final Report of the Industrie 4.0 Working Group*. Forschungsunion, 2013.

[KK17] Olga Kosheleva and Vladik Kreinovich. Why deep learning methods use kl divergence instead of least squares: A possible pedagogical explanation. 2017.

[Knu] Knut Hinkelmann. Neural networks.

[KW13] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[LC10] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[LCC96] Fourmun Lee, A. Chatterjee, and D. Croley. Advanced yield enhancement: computer-based spatial pattern analysis. part 1. In *IEEE/SEMI 1996 Advanced Semiconductor Manufacturing Conference and Workshop. Theme-Innovative Approaches to Growth in the Semiconductor Industry. ASMC 96 Proceedings*, pages 409–415, Nov 1996.

[Mac03] David J. C. MacKay. *Information theory, inference, and learning algorithms.* Cambridge University Press, 2003.

[MC14] G. Mishne and I. Cohen. Multi-channel wafer defect detection using diffusion maps. In *2014 IEEE 28th Convention of Electrical Electronics Engineers in Israel (IEEEI)*, pages 1–5, Dec 2014.

[NM86] Whitney Newey and Daniel McFadden. Large sample estimation and hypothesis testing. In R. F. Engle and D. McFadden, editors, *Handbook of Econometrics*, volume 4, chapter 36, pages 2111–2245. Elsevier, 1 edition, 1986.

[PNM+05] F. Di Palma, G. De Nicolao, G. Miraglia, E. Pasquinetti, and F. Piccinini. Unsupervised spatial pattern classification of electrical-wafer-sorting maps in semiconductor manufacturing. *Pattern Recognition Letters*, 26(12):1857 – 1865, 2005. Artificial Neural Networks in Pattern Recognition.

[PR91] Barak A. Pearlmutter and Ronald Rosenfeld. Chaitin-kolmogorov complexity and generalization in neural networks. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 925–931. Morgan-Kaufmann, 1991.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RC93] Peter J. Rousseeuw and Christophe Croux. Alternatives to the median absolute deviation. *Journal of the American Statistical Association*, 88(424), 1993.

[RLNH17] Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus,

S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2018–2028. Curran Associates, Inc., 2017.

[Rou87] Peter Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, 20(1):53–65, November 1987.

[SCT+17] Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. Amortised MAP inference for image super-resolution. In *ICLR*. OpenReview.net, 2017.

[Ser83] Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA, 1983.

[SGZ+16] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[Tib96] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288, 1996.

[Tuk77] John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[Vaa98] A. W. van der Vaart. *Asymptotic Statistics*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998.

[VC15] V. N. Vapnik and A. Ya. Chervonenkis. *On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities*, pages 11–30. Springer International Publishing, Cham, 2015.

[Wik18] Wikipedia, the free encyclopedia. Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals, 2018.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr 1997.

[YL16] J. Yu and X. Lu. Wafer map defect detection and recognition using joint local and nonlocal linear discriminant analysis. *IEEE Transactions on Semiconductor Manufacturing*, 29(1):33–43, Feb 2016.

[ZH03] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2003.