Post-processing of Monte Carlo calculated dose distributions

LINN ÖSTRÖM
Post-processing of Monte Carlo calculated dose distributions

LINN ÖSTRÖM

Degree Project in Mathematical Statistics (30 ECTS credits)
Degree Program in Applied and Computational Mathematics (120 credits)
KTH Royal Institute of Technology, year 2019
Supervisor at Elekta: Markus Eriksson
Supervisor at KTH: Henrik Hult
Examiner at KTH: Henrik Hult
Degree Project in Mathematical Statistics (30 ECTS credits)
Degree Program in Applied and Computational Mathematics (120 credits)
KTH Royal Institute of Technology, year 2019
Abstract

This Master Thesis focuses on denoising of Monte Carlo calculated dose distributions of radiosurgery treatment plans. The objective of this project is to implement a Denoising Autoencoder (DAE) and investigate its denoising performance when it has been trained on Monte Carlo calculated dose distributions generated with lower number of photon showers. The DAE is trained in a supervised setting to learn the mapping between corrupted observations and clean ones. The questions this thesis aims to answer are: (i) Can a DAE be used to denoise Monte Carlo calculated dose distributions, and thus predict the dose prior to a full simulation? Additionally, (ii) does incorporating prior knowledge of shot position increase the denoising performance? The results in this investigation have shown that the network successfully predicts the dose for low number of photon showers. In very heavy noise inputs the network denoising was in general successful, and the network could fill in missing data. The results indicated that the DAE could reduce the level of noise with an amount comparable with simulations that were done with $10^2$ times more samples.
Sammanfattning

Acknowledgements

I would like to thank my thesis supervisor at the Department of Mathematics KTH, Professor Henrik Hult for his guidance and useful thoughts during this project. His office was always open whenever I needed some guidance. Further, I would like to thank him and Martina Scolamiero for introducing me to new areas such as Topological Data Analysis. I would like to thank my colleague Oscar Pastor at Elekta, our interesting discussions in the lunchroom was of most importance. Additionally, I would like to thank my supervisor Markus Eriksson at Elekta for his support in the ups and downs in this project.

Additionally I want to thank my parents for their encouragement in my studies during these 5 years which also has motivated to continue my work with a PhD in Mathematics at Lund University. Last but not least, I would like to thank Fredrik Anderson for his encouragement and endless support, this would never have been possible without you by my side.

Stockholm, February 2019
Linn Öström
Glossary

AE autoencoder. 50, 51, 52, 53, 54, 55, 56, 64
ANN artificial neural networks. 29, 30
CAE contradictive autoencoder. 53
CDF cumulative distribution function. 15, 16
CLT central limit theorem. 14
CNN convolutional neural network. 3, 29, 41, 42, 43, 44, 46, 47
CT computed tomography. 57
DAE denoising autoencoder. 50, 51, 53, 54, 56, 64, 66, 67, 79
DAG directed acyclic graph. 30, 51, 53
DNN deep neural network. 46
FCL fully connected layer. 30, 38, 41, 43
FNN feedforward neural network. 29, 30, 33, 37, 41, 48, 51
GPU graphical processor unit. 17
IS importance sampling. 14
MC Monte Carlo. 2, 3, 6, 10, 13, 14, 16, 17, 19, 20, 21, 24, 57, 59, 60, 66, 67, 79
MCMC Markov chain Monte Carlo. 13
ML machine learning. 3, 23, 24, 25
MLP multi layer perceptrons. 30
MRI magnetic resonance imaging. 57
NN neural network. 2, 3, 23, 27, 29, 31, 37, 42, 46, 48, 50, 51, 54, 64
OAR organs at risk. 1, 2, 5
PDF probability distribution function. 14, 15, 19
RAE regularized autoencoder. 51, 52, 53
RNG random number generator. 15, 19, 60
RNN recurrent neural network. 29
ROI region of interest. 1
RS radiosurgery. 1, 2, 3, 5, 6, 7, 8, 17, 57, 58, 59, 73, 79
RT radiotherapy. 1, 2, 5, 8
SGD stochastic gradient descent. 37
SL statistical learning. 23, 24
VOI volume of interest. 5, 6, 8, 9, 10, 57, 60
## Contents

1 Introduction
---
1.1 Problem statement ................................................. 2
1.2 Purpose .......................................................... 2
   1.2.1 Methodology .................................................. 3
   1.2.2 Questions to be answered .................................. 3
1.3 Setup ............................................................. 3

2 Dosimetry
---
2.1 Dose distribution .................................................. 6
2.2 Dose distributions in stereotactic radiosurgery ..................... 6
2.3 Dose distribution evaluation ..................................... 7
   2.3.1 Van Dyke or global gamma index .......................... 8
   2.3.2 Total absolute and relative dose difference .............. 9
   2.3.3 Isodose comparing of relative dose ....................... 10
   2.3.4 Voxel-test .................................................. 11

3 Monte Carlo methods
---
3.1 Basic Monte Carlo .................................................. 14
3.2 Sampling methods .................................................. 15
   3.2.1 Transformation sampling .................................... 15
   3.2.2 Rejection sampling .......................................... 16
   3.2.3 Variance reduction ......................................... 16
3.3 Monte Carlo dose calculation .................................... 17

4 Statistical Learning
---
4.1 Inference problem .................................................. 24
4.2 Parametric and non-parametric models ............................. 25
4.3 Training and test data in statistical learning ...................... 25
   4.3.1 Data augmentation ......................................... 25
4.4 Overfitting and underfitting ..................................... 26
   4.4.1 Methods to detect and reduce the risk of overfitting ... 27

5 Neural Networks
---
5.1 Feedforward Neural Network (FNN) ................................ 30
   5.1.1 Artificial neuron ............................................. 31
   5.1.2 Training a feedforward network ............................ 33
   5.1.3 Activation functions ....................................... 38
   5.1.4 Loss functions ............................................. 40
5.2 Convolutional Neural Network (CNN) .................................. 41
  5.2.1 Convolution layer ............................................. 41
  5.2.2 Local receptive field ......................................... 44
  5.2.3 Pooling layers ................................................. 45
  5.2.4 Residual connections ......................................... 46
  5.2.5 Backpropagation in CNNs .................................... 47
5.3 Methods to speed up training of a FNN ................................ 48
5.4 Denoising using Neural Networks ................................... 50
  5.4.1 State of the art and denoising networks ....................... 50

6 Autoencoders .......................................................... 51
  6.1 Regularized autoencoders ......................................... 52
    6.1.1 Sparse Autoencoders ....................................... 53
  6.2 Denoising Autoencoder ........................................... 53
    6.2.1 Score matching ............................................. 54
    6.2.2 DAE for Gaussian corruption .............................. 55
    6.2.3 Manifold interpretation of DAE ......................... 56

7 Data ................................................................. 57
  7.1 Treatment plans .................................................. 57
  7.2 Dose distributions ............................................... 58
    7.2.1 Generation of dose distributions ......................... 59
    7.2.2 Simulation parameters .................................... 60
    7.2.3 Noise levels in dose kernels ............................. 61
    7.2.4 Error in dose distributions ............................... 62

8 Implementation ....................................................... 63
  8.1 Denoising AE structure ......................................... 64
    8.1.1 Denoising framework .................................... 66
    8.1.2 Considered inputs ......................................... 66
  8.2 Considered data .................................................. 67
  8.3 Software and hardware .......................................... 67

9 Result ................................................................. 69
  9.1 Denoising performance depending on number of photon showers . 69
  9.2 Denoising performance depending on input structure ............ 73
    9.2.1 DC3 input .................................................. 73
    9.2.2 DSP input .................................................. 74
    9.2.3 Comparison between DC1, DC3 and DSP .................... 74
  9.3 Resolution impact ................................................ 75

10 Discussion .......................................................... 77

11 Conclusion .......................................................... 79

Bibliography .......................................................... 81
A Appendix

A.1 MC unbiased estimate ........................................ 87
A.2 SNIS ......................................................... 87
A.3 Rejection sampling .......................................... 88
A.4 Slices of test and validation set ......................... 88
A.5 Isodose contours of prediction samples ............... 88
Chapter 1

Introduction

Cancer is a collection of diseases associated with abnormal cell growth which can invade or spread to other parts of the body, the cluster of these cells in the neoplasm tissue outline the mass that is called a tumor. Different tumor tissues can either be malignant or benign, and the severity of the tumor types can differ significantly. A benign tumor is a non-cancerous tumor, where the tumor cells only grow locally and hence cannot spread by invasion or metastasis. These cells are often surrounded by a protective sac, keeping the cells contained in the tissue. On the contrary, the malignant (cancerous) tumor can invade neighboring tissues and enter blood vessels, i.e. metastasis, spreading the cancer to other parts of the body. The cancerous tissues can be treated curative, adjuvant, neoadjuvant or palliative [40] where the palliative treatment aims to relieve symptoms of cancer and reduce the pain for the patient [62]. The curative aims at treating the tumor and curing the disease, and the adjuvant treatment aims at preventing illness recurrence, while the neoadjuvant treatment aims to reduce the tumor size before other types of treatment such as operation.

These benign or malignant tumors can, among other actions, be treated by radiotherapy (RT) or radiosurgery (RS). RT and RS are therapies performed by ionizing radiation which objective is to break the DNA-strings and hence to damage or kill these benign or malignant cells. The tumorous tissue is often referred to as the region of interest (ROI) or target in practice. The ionization in RT is typically performed using a linear accelerator, that shapes the radiation beams to focus on the target area and aims to spare normal surrounding tissues or vital surrounding organs which are referred to as the organs at risk (OAR). Nevertheless, the shaped radiation beams will harm some surrounding healthy tissue on the path to the target area as the radiation beam passes the healthy tissue at the path to the target. radiosurgery (RS), which is mainly used to treat brain tumors tissues, however, reduces this harm to the surrounding tissues. The method aims to reduce the radiation to the non-target tissues by radiating from multiple angles, which all together focus at an isocenter and jointly creating a focused shot towards the target. Radiosurgery (RS) refers to destruction or surgery through radiation of precise particular areas using ionization radiation rather than traditional surgical operation. radiosurgery (RS) was introduced by the Swedish neurosurgeon Lars Leksell [33] which subsequently developed the stereotactic RS tool Leksell Gamma Knife for "managing small intracranial brain lesions of volume typically less than 25 cm$^3$" [5] [36,37]. The Leksell Gamma Knife uses high-intensity gamma radiation and achieves high precision delivery by focusing the sector beams in the collimator body to a focus spot in the target.
1.1 Problem statement

Precise dose calculations are essential in the planning stage of RT and RS treatments, foremost to assure that the cancerous tissue receives the planned dose but moreover to reassure that the surrounding organs at risk (OAR) are left unharmed. An integrated dose calculation algorithm would ideally capture the sharp dose gradients and scattering effects when radiating complex geometries and materials. However, this generally boils down to very complex models and computationally demanding tasks for the algorithm. Monte Carlo (MC) dose calculation is one such algorithm. In literature, it is often referred to as the Golden standard of dose calculations \[12, 28, 61, 67\] and is considered to be a reliable technique for advanced treatment planning [19]. Nevertheless, MC dose calculation algorithms have the main drawback that they are computationally intensive which limits the possibility of being integrated into some dose planning optimization algorithms. Instead, simplified reduced accuracy dose calculation algorithms are integrated into the RT and RS planning systems, and Monte Carlo dose calculations are often left to be used as a reference for evaluation.

Inherently, the statistical fluctuations in the Monte Carlo estimate of the dose distribution decrease with the increasing number of simulations and eventually becomes negligible for a large number of simulations. Thence, the estimate is approaching the true target distribution but coming at the cost of exceedingly long computation time. Hence there is a trade-off regarding the statistical uncertainty and computation time. Useful methods in the last decades have been developed to remove this statistical uncertainty for shorted dose distribution simulations. These familiar published methods for reducing the MC dose calculation time involves Wavelet threshold denoising (WTD) \[16\], Locally-adaptive Savitzky-Golay (LASH) filter, and iterative reconstruction of noise \[21\] aiming to denoise MC photon simulations with few source particles. All methods are facing the same challenges of simultaneously smoothing local areas, capture sharp edges and drops in the dose distribution.

1.2 Purpose

Prior to stereotactic RS a treatment plan is optimized in Gamma Plan, which is a software used for dose planning of the Gamma Knife and is developed and maintained by Elekta. Gamma Plan aims to optimize an objective function based on the weighted selectivity, coverage, gradient index and beam-on time, for the given geometry, tumor and dose prescription. The optimization is done based on dose distributions that are calculated using simplified geometry considerations and assuming translation invariant properties of the dose distributions, which introduces bias. A more precise dose calculation would be modeling the radiation transportation equation using Monte Carlo (MC) methods. Monte Carlo dose calculation is the most accurate dose calculation as it is an unbiased estimator. However, the asymptotic variance of the estimate decreases slowly and the number of samples or simulations needed to obtain a statistically precise estimate is currently limited by the computational power of today’s machines and the need of efficient work-flow.

This Master Thesis is carried out at Elekta Instrument AB, together with the Department of Mathematics at KTH. This thesis aims to investigate the ability to, using statistical learning methods, capture the mapping of the asymptotic mean of the Monte Carlo dose
calculation. More specifically, using a Densoing Autoencoder. Furthermore, it aims to investigate the possibility of integrating the denoiser, trained to denoise Monte Carlo (MC) calculated dose distributions, in treatment optimization systems by considering the prediction accuracy of the denoiser. Additionally, it investigates the network denoising performance when prior knowledge of the shot-positions in the RS are given to the network, and lastly investigate the performance of the neural network (NN) when a stack of adjacent dose slices are provided to the network.

1.2.1 Methodology

The construction of this investigation is formed accordingly; the network is trained to minimize the average squared error based on a training data set. The network denoising performance is subsequently evaluated on unseen test and validation data sets. Further, the denoising performance is quantified using a set of defined metrics.

1.2.2 Questions to be answered

Through the previously described methodology this thesis aims to answer the following questions

- Can a denoising Autoencoder capture the true dose distribution of Monte Carlo calculated dose distributions prior to a full simulation?

- Can the denoising performance be improved by adding engineered priors to the denoising Autoencoder?

1.3 Setup

The setup of this is carried out accordingly, Chapter 2 presents the basics of dosimetry, RS and methods for dose evaluation. An overview of the Monte Carlo (MC) estimate and sampling techniques are described in Chapter 3, together with a brief overview of MC simulation of radiation transportation. Statistical learning and common problems in machine learning (ML) are presented in Chapter 4. Chapter 5 describes the basics of neural networks and the setup of its methods such as training and propagation, and the powerful convolutional neural networks. Autoencoders are described in detail in Chapter 6 and the structure of the data is presented in Chapter 7. Lastly, the implementation and results are presented in Chapter 8 and Chapter 9, followed by the discussion in Chapter 10 and the conclusion in Chapter 11.
Chapter 2

Dosimetry

Radiation dosimetry is the field of measuring and assessing the ionizing radiation dose absorbed by the human body. More specifically, the absorbed dose is a measure of the energy deposited in a medium by ionization radiation per unit mass and is commonly measured in gray $Gy (J \times kg^{-1})$. The distribution of this absorbed dose is what is referred to as the dose distribution, detailing the spread of the deposited energy in the volume of interest (VOI). Let us now shortly return back to radiation. Generally, radiation can act internally or externally, where the former refers to radiation due to ingested or inhaled radioactive substances and the latter refers to external irradiation of sources of radiation. External dose absorption originates for example from radiation through the previously discussed RT or RS. The latter, RS, will be the area of focus in this Chapter.

The preciseness of the measured absorbed dose in a target tissue and the OARs is essential to obtain good RS and RT plans. Indeed, if the target tumorous tissue is underdosed the growth of the cells may not be halted and thence can allow the targeted tumor to continue growing. A too high dose can in the most severe cases lead to scarrings or radiation overdose toxicity at OARs, resulting in e.g. the blindness of the patient. However, finding an optimal treatment plan is arduous and in some cases an "underdose or overdose can rarely be completely eliminated"\textsuperscript{1} [44].

Let us now shortly introduce some important concepts. In the field of dosimetry the relative dose intensity is the ratio between the planned dose and the actually delivered dose, where 1 indicates that 100 % of the scheduled dose was administrated per protocol. The dose rate is the speed in which the dose is delivered to the target, measured in $Gy/s$, and is used to calculate the accumulated dose by scaling of irradiation time. The total accumulated dose is then given by dose rate $\times$ radiation time.

\textsuperscript{1}[word] means that the quote has been altered with word
2.1 Dose distribution

The dose delivered to the volume of interest (VOI) is a quantity depending on the irradiated geometry and material [56] and can vary sharply throughout the considered geometry. The spatial changes of the dose are captured in a multidimensional dose distribution which describes the outcome of the deposited energy in each location of the considered volume. The distribution is often complex and must be estimated through modeling or simulation of the radiation transportation. Furthermore, to estimate the dose distribution for some considered geometry, the volume must first be discretized into 3-dimensional cubic elements referred to as voxels. Naturally, the smaller size of the voxels the higher accuracy of the estimated dose at each coordinate in the volume, whilst the storage and computational time increases for the studied object. Further, computing dose distributions for a given object requires some spatial arrangement of these voxels, thence let the voxel coordinate be indexed \((x, y, z) \in V\). The mass averaged absorbed dose \(\bar{D}_T\) received to a target volume \(T\) given a dose distribution \(D(x, y, z)\) and geometry \(\rho(x, y, z)\) in a volume space \(V\) is thence the weighted dose integrated over the target volume \(T\), which can be expressed as

\[
\bar{D}_T = \frac{\int_T D(x, y, z)\rho(x, y, z)dV}{\int_T \rho(x, y, z)dV}.
\] (2.1)

The distributed dose in each location \(D(x, y, z)\) is mainly the region of interest in this thesis and thence when we refer to dose, it is the dose distribution that is considered. In general, the distributed dose \(D(x, y, z)\) for a given geometry and irradiation setup can be efficiently calculated using Monte Carlo (MC) sampling techniques and statistics\(^2\). However, the stochastic integration method results in statistical uncertainties which are referred to as signal-dependent noise in the dose distributions. Inherently the statistical uncertainty approaches zero when the number of samples approaches infinity, and the estimated dose approaches the true target dose distribution. This method will be further discussed in Chapter 3 and therefore will be left for now.

2.2 Dose distributions in stereotactic radiosurgery

Let us now shortly review the aim of stereotactic radiosurgery, mainly considering the Gamma Knife that was briefly discussed in the introduction. Again, the Gamma Knife is a stereotactic radiosurgery tool developed and maintained by Elekta which is used for treating e.g. small tumors in the human brain. The radiosurgery tool is constructed of 192 Cobalt-60 sources distributed in eight sectors placed in a collimator body, targeting a focus spot referred to as the isocenter, see Figure 2.1a. Further, each collimator sector can be adjusted to deliver higher or lower radiation by engineering the collimator sector setups. The collimator sizes available are 4, 8 or 16 mm, where a larger collimator size indicates a higher delivered dose. The shape of the dose distribution is thence directly dependent on these sector configurations, see Figure 2.1b. A configured setting of the 8 sectors, \(s_{\text{info}}\), delivering the dose at isocenter \(p(x, y, z)\) for a duration time of \(t_{\text{info}}\), will further be referred to as a shot. Additionally, in the Gamma Knife the patient can be translated in 3 orthogonal directions between the shot settings. This enables precise and custom treatment plans, since multiple shots may be delivered to the tumor using different sector configurations and isocenter positions.

\(^2\)Among other methods
The RS treatment plans used in the Gamma Knife are constructed in Gamma Plan, and are optimized based on some calculated dose distribution for the different sector configurations, shot positions and shot durations (beam-on-time). For now, we will leave RS and subsequently return to the subject in Chapter 7.

2.3 Dose distribution evaluation

General dose evaluation techniques in RT and RS involves comparing Dose-Volume Histograms (DVH) which is a measure of the fraction of voxels in the VOI that has received some dose level. The DVH however, "lack information about the spatial arrangement of the dose" [17] and hence is not considered to be an optimal metric to evaluate dose distributions. However, a conventional method for quantitative dose distribution evaluation, most commonly used in RS, is the Gamma index or the Van Dyk test (Global Gamma Index). The methods "have been developed to facilitate quantitative comparisons, including superimposed isodoses, dose-difference, and distance-to-agreement (DTA) distributions" [35] and in its simplest form can be considered as a test of whether an evaluated dose significantly differs from a reference dose.

The remaining dose evaluation techniques described in this section are somewhat specialized for this thesis, and hence we want to stress that some of these measures are not common metrics for evaluation of stereotactic radiosurgery dose distributions. The remaining metrics and evaluation methods are (i) the total absolute and relative dose difference, a constructed test which we will refer to as (ii) the Voxel-test and the last
evaluation technique, (iii) comparing isodose contours. The latter (iii) is based on a visual inspection of the isodose contours and hence is not quantifiable. However, it provides us with a tool to evaluate the dose-contour shapes. In summary, the considered evaluation measures that will be presented in this section are

- Quantitative gamma index
- Total absolute and relative dose difference
- Isodose comparing of relative dose
- Voxel-test

### 2.3.1 Van Dyke or global gamma index

The Gamma index, occasionally in literature referred to as the gamma test, "is one of the most commonly used [metrics] for the verification of complex modulated radiotherapy." [25]. The objective of the test is to compare an evaluated dose distribution $D_{\text{eval}}$ with a reference or true distribution $D_{\text{ref}}$, and quantify the difference between the considered doses. The distribution test has the nice attribute of transforming the multivariate evaluation problem into an informative scalar value, yielding the fraction of non-failed tests. The Generalized $\Gamma$ function for an evaluated and reference dose $D_{\text{eval}}, D_{\text{ref}}$ at the evaluated position $\vec{r}_{\text{eval}}$ and reference position $\vec{r}_{\text{ref}}$ is given by

$$\Gamma(\vec{r}_{\text{ref}}, \vec{r}_{\text{eval}}) = \sqrt{\frac{|\vec{r}_{\text{ref}} - \vec{r}_{\text{eval}}|^2}{\delta_{\text{DTA}}^2} + \frac{(D_{\text{ref}}(\vec{r}_{\text{ref}}) - D_{\text{eval}}(\vec{r}_{\text{eval}}))^2}{\delta_{\text{DD}}^2}}, \quad \Gamma(\vec{r}_{\text{ref}}, \vec{r}_{\text{eval}}) \geq 0$$

(2.2)

where $\delta_{\text{DTA}}$ is the Distance-To-Agreement criterion and $\delta_{\text{DD}}$ is the Dose Difference criterion, which are used to set the maximal deviations for the test to be passed. These parameters are called the gamma acceptance criterions [29] and are in RT often set to $\delta_{\text{DTA}} = 2$ mm and $\delta_{\text{DD}} = 0.03 \max(D_{\text{ref}})$, allowing the dose to differ by 3 % and vary with 2 mm [35]. Hence from the definition, the $\Gamma$ test is based on the scaled Euclidean distance between coordinates and the scaled dose-values in these coordinates. Then, the global $\gamma$-index is given by

$$\gamma(\vec{r}_{\text{eval}}) = \min_{\vec{r}_{\text{ref}}} \{\Gamma(\vec{r}_{\text{ref}}, \vec{r}_{\text{eval}})\} \forall\{\vec{r}_{\text{ref}}\}$$

(2.3)

i.e. the smallest generalized $\Gamma$ function in the set of evaluated points. Further, a test is defined to be passed if $1 \geq \gamma(\vec{r}_{\text{eval}}) \geq 0$ and failed if $\gamma(\vec{r}_{\text{eval}}) > 1$. In simpler terms, combining (2.2) and (2.3), the Van dyke or global gamma index can be rewritten as

$$\gamma(\vec{r}_{\text{eval}}) = \min_{\vec{r}_{\text{ref}}} \left(\frac{(D_{\text{eval}}(\vec{r}_{\text{eval}}) - D_{\text{ref}}(\vec{r}_{\text{ref}}))^2}{\delta_{\text{DD}}^2} + \frac{|\vec{r}_{\text{eval}} - \vec{r}_{\text{ref}}|^2}{\delta_{\text{DTA}}^2}\right)$$

(2.4)

and the fraction of passed tests in the evaluated dose is hence given by

$$\gamma = \sum_{\vec{r} \in \mathcal{F}_{\text{eval}}} \mathbb{I}(\gamma(\vec{r}) \leq 1) / \sum_{\vec{r} \in \mathcal{F}_{\text{eval}}} 1$$

(2.5)

where $\mathbb{I}$ is an indicator function returning 1 if the entity is true. Note that due to the formulation of (2.4), the $\gamma$-test is computationally intensive for large dose grids [25],

[^word: means that the quote has been adjust with word]
especially when stressing the test to 3D data. Thence, the less correct but more convenient 2D-gamma test can be used instead. As previously mentioned, the gamma index is commonly used in RT and the standard criterion of distance-to-agreement and maximal dose difference must be adjusted when considering RS dose distribution evaluation. This is since the magnitude of the VOI differs and the preciseness of the delivered dose in RS is crucial, thence one can change the acceptance criterion to $\delta_{DTA} = 1$ mm.

Figure 2.2: Example of a 2D–gamma index for a reference dose and evaluated dose slice. The fraction of passed voxels in this example is 91 %.

An example of a 2D-gamma test for an example dose distribution is shown in Figure 2.2, where the red pixels indicates a failed test in coordinate $\bar{r}_{eval}$ i.e. $(\gamma(\bar{r}_{eval}) > 1)$, while blue and white coordinates are non-failed tests $(1 \geq \gamma(\bar{r}_{eval}) \geq 0)$. The Gamma index calculated in Figure 2.2 uses the gamma acceptance criterion $\delta_{DTA} = 1$ mm and dose difference criterion $\delta_{DD} = 0.03 \times \max(D_{ref})$, and has a total acceptance rate of 91%. Additionally, hereafter in this thesis, the gamma index will always be based on the gamma criterions $\delta_{DTA} = 1$ mm and $\delta_{DD} = 0.03 \times \max(D_{ref})$.

### 2.3.2 Total absolute and relative dose difference

In this section, we will consider a different type of evaluation technique, which essentially gives us a more general measure of the error in the evaluated distribution. Let us consider some VOI which is discretized into a set of voxels $M = m \times n \times k$ with resolution $\varepsilon$. Thence we defined the total absolute dose difference $\Delta_T$ and the relative error $\Delta_R$ between an evaluated and a reference dose to be the following

$$
\Delta_T = \varepsilon^3 \sum_{i=1}^{M} |D_{ref}(i) - D_{eval}(i)|
$$

$$
\Delta_R = \frac{\Delta_T}{\varepsilon^3 \sum_{i=1}^{M} D_{ref}(i)}
$$

(2.6)

where $\varepsilon$ is added to correct for the increase in error when the number of voxels is increased. Thence, the metric can be used compare doses with different resolutions.
2.3.3 Isodose comparing of relative dose

Isodoses refers to points or zones in a medium that receives some equal dose of radiation. Naturally the isodose curvatures or shapes are smooth capturing the joint set in the VOI that has received some particular dose. Further, as the particular dose value is not of interest for this evaluation technique, the doses are scaled to the relative dose, i.e. \( D_{relative} = D \times \max(D)^{-1} \). Let us now consider a set of MC calculated dose distributions shortly. The isodose contours from MC low photon simulations contain high statistical uncertainty and fluctuations which results in noisy and inconsistent isodose contours which does not reflect the true underlying contours of the dose. Indeed, as the number of photon simulations increases the contours approaches the true contours, see Figure 2.3.

![Isodose-contour obtained from MC dose estimation with \( N \) being the number of photon showers. The isodose-contours presented are expressed in relative dose.](image)

(a) Estimated dose with \( N = 10^6 \).  
(b) Estimated dose with \( N = 10^7 \).

(c) Estimated dose with \( N = 10^8 \).  
(d) Estimated dose with \( N = 10^9 \).

Figure 2.3: Isodose-contour obtained from MC dose estimation with \( N \) being the number of photon showers. The isodose-contours presented are expressed in relative dose.

Thence, evaluating a dose distribution can also be done by visual inspection. Then the isodose contours of the evaluated relative dose is compared to the isodose contours of the reference relative dose.
2.3.4 Voxel-test

Let us now for the sake of simplicity assume that we have at hand the voxel-wise dose deviation $\sigma_{D_{ref}}$ for the reference dose $D_{ref}$. Further, assume that we can construct a confidence interval for each estimated voxel dose. Then $(1 - \alpha)$ confidence interval for the estimated reference dose $\hat{D}_{ref}$ and the estimated standard deviation $\hat{\sigma}_{D_{ref}}$ is given by

\[
I_{D_{ref}(i)} = \hat{D}_{ref}(i) \pm \frac{t_{\alpha/2} \hat{\sigma}_{D_{ref}(i)}}{\sqrt{N}}
\]

(2.7)

and can be used to form a custom dose test. In Equation (2.7) the $t$ denotes the $t$-distribution. We define a voxel-test to be non-rejected if a dose value $D_{eval}(i) \in I_{D_{ref}(i)}$ and rejected if $D_{eval}(i) \notin I_{D_{ref}(i)}$. Thence the proposed measure quantifies the fraction of non-rejected dose voxels in the evaluated dose $D_{eval}$, and will be defined

\[
\beta = \frac{1}{M} \sum_{i=1}^{M} \mathbb{I}(D_{eval}(i) \in I_{D_{ref}(i)})
\]

(2.8)

where $\mathbb{I}$ is an indicator function returning 1 if the entity is true. Further, $M$ is the number of voxels in the evaluated dose.
Chapter 3

Monte Carlo methods

The power of Monte Carlo (MC) methods was initially revealed during the second world war when the first electronic computers were built [8]. At this time Monte Carlo was applied in particle movement simulations, in connection with the Manhattan project with the aim to develop the atomic bomb. Since then the use and knowledge of MC have been tremendously broadened, for example, the Markov chain Monte Carlo (MCMC) method published in the paper by Metropolis, Rosenbluth and Teller in 1953 [41]. Today MC is commonly multidisciplinary, integrated into systems in almost any field "such as physical sciences, engineering, statistics, finance, and computing, including machine learning and graphics"1 [47]. The power and flexibility of the methods are still unquestionable but their application comes at the cost of long computation time and stressing of memory budget. A lot of eager work has been done to decrease the computation time such as variance reduction techniques and exponential twisting in rare event simulations, but the disparity between more simplified models and the MC methods remain sizeable. Nevertheless, MC methods are often considered to be "the simplest way to solve a problem and sometimes it is the only feasible way"2 [47].

The main objective of MC methods is to learn about some system or examine a property of some unknown distribution through random sampling and simulation. Naturally, as the method is based on random events, it is referred to be a stochastic method as opposed to deterministic models. Deterministic models aim to evaluate some quantity directly, without any random event considered. Let us now consider MC in more detail, first diving into the definition.

---

1 Among other fields
2 In the case where the evaluation is considered complex
3.1 Basic Monte Carlo

Assume that the objective is to determine the parameter $\tau$, which is a function $\phi$ depending on the random variable $X$ which is distributed according to the PDF $f$, $X \sim f$. Furthermore, assume that the r.v. exists in a sample space $X \in \mathcal{X}$. Then, the objective is to evaluate $\tau = \mathbb{E}_f[\phi(x)]$, which straightforwardly can be written as

$$\tau = \int_{\mathcal{X}} \phi(x)f(x)dx. \quad (3.1)$$

Additionally, assume that solving the integral (3.1) directly, is unfeasible. This might be since the distribution function $f$ is unknown, or only known up to some normalizing constant. Then the integral (3.1) can be evaluated using Monte Carlo methods. The Monte Carlo estimate $\tau^{MC}_N$ is formulated through a set of randomly drawn samples of some size $N$, i.e. $\{X_i\}^{N}_{i=1} \sim f$ and the estimate is defined as Equation (3.2). Furthermore, it can be shown that the estimator is unbiased\(^3\), Equation (3.3).

$$\tau^{MC}_N = \frac{1}{N} \sum^{N}_{i=1} \phi(x_i) \quad (3.2)$$

$$\mathbb{E}[\tau^{MC}_N] = \tau \quad (3.3)$$

As the number of random samples $N$, $\{X_i\}^{N}_{i=1} \sim f$ increases, the estimator gets closer to the true parameter value and hence the Monte Carlo estimate is also consistent. Indeed, using the central limit theorem (CLT) it can be shown that the error of the estimate is normally distributed with statistical uncertainty that is decaying negatively proportional to the number of samples $N$,

$$(\tau^{MC}_N - \tau) \xrightarrow{d} \mathcal{N}(0, \frac{\sigma(\phi)}{\sqrt{N}}) \quad (3.4)$$

where $\text{Var}(\phi) = \sigma^2(\phi)$. Expression (3.4) assures that as $N \to \infty$ the statistical uncertainty will approach zero, however the variance of the error decays slowly. Thence, note that if we want to decrease the statistical uncertainty with a factor of 10 we need to include 100 times more samples. The asymptotic variance may be influenced by different variances reduction techniques such as control variates, antithetic sampling or interaction forcing, the latter described briefly in Section 3.2.3. Nevertheless, the computational time is still adherent.

Techniques to estimate $\tau$ in cases where the distribution function $f$ is known only up to some normalizing constant is for example the importance sampling (IS) method, or more specifically the self-normalized IS. Self-normalized importance sampling is a sampling method that makes it possible to sample from $X \sim f$ through an instrumental density $g$ on the same sample space $\mathcal{X}$. Thence, by sampling from the instrumental distribution $g$ we can recover an unbiased estimate of $\tau$,

$$\tau = \mathbb{E}_f[\phi(X)] = \int_{\mathcal{X}} \phi(x)f(x)dx = \frac{\mathbb{E}_g[\phi(X)w(X)]}{\mathbb{E}_g[w(X)]} \quad (3.5)$$

where the importance weights are defined accordingly $w(x) = c_n f(x)/g(x)$ and $c_n$ is the normalizing constant dependent on the number of current samples drawn $n$. Thence, see

\(^3\)Proof can be found in Appendix A.1
Equation (3.5), the problem has been cast into estimating one Monte Carlo estimate for the numerator and one for the denominator. Furthermore the asymptotic variance of the estimate can be controlled by engineering the importance weights \( w(x) \) and the sampling distribution \( g(x) \). However, as previously stated, the problem still stands regarding the computational time of the MC estimate, even though IS methods can increase the convergence rate significantly with a proper instrumental distribution.

### 3.2 Sampling methods

Many methods have been developed in the theme of sampling, and in this section we will solely discuss a few of them. Even though one can never obtain a sampling method that directly reflects the unknown distribution that we wish to sample from, one can come pretty close. Some methods are based on the assumption that we have at hand some instrumental distribution \( g \) which we can sample from and indirectly recover samples from the target distribution \( f \). One such method is the importance sampling methods described above and another is the rejection sampling. Lastly we have the transformation sampling method which is an efficient way of generating univariate r.v.'s.

#### 3.2.1 Transformation sampling

Given a pseudo-random number generator (pseudo-RNG) generating pseudo uniform random variables, one can transform drawn samples using the inverse-transform to be of some desired distribution. In simple terms, the transformation sampling method aims to transform the uniform random variable \( U \sim \mathcal{U}(0,1) \) into \( T(U) = X \sim f \) where \( f \) is the PDF of the target distribution, using the transformation Theorem. The method is however only feasible for univariate r.v.'s and when the cumulative distribution function (CDF) \( F \) has a known inverse \( F^{-1} \). For a univariate random variable \( X \) with CDF \( F \) and PDF \( f \), where \( \exists F^{-1} \), samples \( X \sim f \) can be generated accordingly

\[
\text{draw } U \sim \mathcal{U}(0,1) \\
\text{set } X \leftarrow F^{-1}(U).
\]

Then the inverse-transform theorem assures that \( X \sim f \). Indeed, for e.g. \( g(u) = F_X^{-1}(u) \) and \( u \sim \mathcal{U}(0,1) \), the transformation theorem yields

\[
f_X(x) = f_U(F_X(x)) \frac{d}{dx}F_X(x) = f_X(x) \tag{3.6}
\]

Since \( f_U(F_X(x)) = 1 \ \forall x \), the transformed random variable \( X \) has the desired distribution.

---

\(^4\)The full derivation of self normalizing IS can be found in Appendix A.2
3.2.2 Rejection sampling

The rejection sampling method is a useful sampling technique when the inverse of the CDF $F$ cannot be evaluated. The main core of the sampling method is to perform random sampling on an instrumental distribution $g$ that covers the target distribution $f$ and accept the samples according to an acceptance probability. The instrumental density $g$ must be on the same sample space $\mathcal{X}$ as the objective r.v. $X$ and it’s density $g$ must cover the target distribution $f$ for all $x$, i.e. $f(x) \leq Kg(x)$ where $K \in \mathbb{R}$ and is called the acceptance rate.

![Probability density](image)

Figure 3.1: Visualization of the rejection sampling method. The target distribution $f$ (red) is sampled indirectly through sampling from the instrumental distribution $g$ (blue) and are accepted with some acceptance rate.

Furthermore, assume that $\exists K < \infty$ such that $f(x) \leq Kg(x) \ \forall x \in \mathcal{X}$, then one can generate samples $X \sim f$ by drawing samples $X^* \sim g$ and accept the samples with probability $f(X^*)(Kg(X^*))^{-1}$. Again, then the accepted samples are distributed according to the target distribution $X^* \sim f^5$. In Figure 3.1 a visualization of the rejection sampling is shown, where the black samples are the accepted samples from the instrumental distribution. Furthermore, the choice of the instrumental density directly affects the rate of generated samples since if the acceptance rate is low, the samples are generated inefficiently.

3.2.3 Variance reduction

As previously mentioned, variance reduction techniques involve increasing the precision of some considered estimate. More specifically, it aims to decrease the variance with lower computational effort than the basic MC. In this short section we will very briefly consider interaction forcing, which as its name reveals forces interactions to decrease the variance and maintain the estimate to be unbiased. This method is photon specific and hence the only considered reduction technique. The method aims to force particle interactions by modifying the event probability with a weight factor referred to as the duplication factor, increasing the probability of the process of interest. Thus, in this manner one can speed up the convergence and still obtain an unbiased estimate.

---

5A pseudo code of the sampling technique can be found in Appendix A.3 together with the proof
3.3 Monte Carlo dose calculation

As mentioned, the MC method is commonly used in complex particle simulations and "are considered as the gold standard for modeling photon and electron transport in heterogeneous medium" [61] [12, 28, 67]. They are regularly used for dose verification in treatment plans [9, 46, 63] as the methods can be evaluated for complex geometries and arbitrary materials. The MC integration technique simulates the radiation transportation equation applying different sampling techniques such as the previously presented methods RS and inverse-transform sampling. When a simulated particle travels in some space it might interact with some medium, e.g. through bouncing of electrons or molecules. These interaction probabilities are obtained from the attenuation coefficients $\mu$, which is defined as the differential integration probability. Furthermore, each primary particle "originates a shower of electrons and photons, which are individually tracked down to the corresponding absorption energy" [50]. This absorption energy is what will be denoted $E_{\text{threshold}}$. In general, the branching of particle simulations results in complex calculations, and due to this, graphical processor unit (GPU)s or cluster operations are conventionally used.

The main advantage of MC dose distribution calculations is principal that the dose can be evaluated in complex geometries and various materials, and ergo from the property of MC generate unbiased estimates. The MC dose distribution calculation is based on multiple single photon simulations, referred to as showers, where each parameter such as mean-free path, interaction type, deposited energy and post direction of travel is sampled from known distributions or distributions known up to some normalizing constant, to create a trajectory for this simulated particle. In the MC photon simulation each primary and secondary particle is tracked till it is considered to be absorbed at some threshold energy or the particle has disappeared from the simulation scene. In the next section we will consider this simulation for a single particle in more detail.

Simulation of radiation transport

In this subsection an overview of one particle simulation is discussed and for the sake of simplicity we disregard the production of secondary particles and only consider a particle traveling in homogeneous medium. Indeed, each trajectory path can be modeled by a Markov process where the next event is only dependent on the previous state. Thence, from the very nice property of Markovian processes "we can stop the generation of a particle history at an arbitrary state [...] and resume the simulation from this state without introducing any bias in the results." [50]. This nice property greatly simplifies the expanding of the simulation to the heterogeneous medium case, since the simulation can easily be initialized at in the new medium using previous state information.
Let us now return to our original topic, each particle originates at an initial state $(\hat{r}_0, E_0, \hat{d}_0)$ where $E_0$ is the initial energy, $\hat{r}_0 = (x_0, y_0, z_0)$ is the position coordinate and $\hat{d}_0$ is the direction of travel of the considered particle. Subsequently, the next state is simulated in the medium using a set of probability distributions where the random variables such as length of the free flight $s$, the polar scattering angle $\theta$, azimuthal scattering angle $\phi$ and the energy loss $W$ are sampled using different random sampling methods. See Figure 3.2 for visualization of parameters and interaction types. All of the presented quantities are summarized in Table 3.1.

| $E_n$ | energy of the particle at state $n$ |
| $\hat{r}_n$ | position of the particle at state $n$ |
| $\hat{d}_n$ | direction of movement at state $n$ |
| $\theta$ | polar scattering angle |
| $\phi$ | azimuthal scattering angle |
| $W$ | energy loss |
| $s$ | distance before next interaction |
| $q_n$ | interaction type $n \in \{1, \ldots, k\}$ |

Table 3.1: Variables for particle simulation

For example, the length of the free flight $s$ and the interaction type $p$ is sampled though inverse-transform sampling described in Section 3.2. A pseudo-code for a single particle simulation is presented in Algorithm 1.

---

\(^8\)Images are borrowed from PENELLOPE manual [51]
Algorithm 1 Simulation trajectory of a particle

**Result:** Trajectory \( \{(\hat{r}_i, E_i, \hat{d}_i)\}_{i=0}^{N} \)

**Given:** Initial state \((\hat{r}_0, E_0, \hat{d}_0)\)

**Set:** \((\hat{r}_n, E_n, \hat{d}_n) \leftarrow (\hat{r}_0, E_0, \hat{d}_0)\)

while \(E_{\text{threshold}} < E_n\) do

\[
\xi_1, \xi_2 \sim \mathcal{U}(0, 1)
\]

Sample the distance traveled \(s \leftarrow \lambda_T \ln \xi_1\)

Sample interaction type \(q_n \sim p\)

Sample polar scattering angle and energy loss \((\theta, W) \sim p_{q_1, \ldots, q_k}(E_n; \theta, W)\)

Sample azimuthal scattering angle \(\phi \leftarrow 2\pi \xi_2\)

\[
\hat{r}_{n+1} \leftarrow \hat{r}_n + s \hat{d}_n
\]

\[
E_{n+1} = E_n - W
\]

\[
\hat{d}_{n+1} = R(\theta, \phi) \hat{d}_n
\]

\[
\{(\hat{r}_{i+1}, E_{i+1}, \hat{d}_{i+1})\}_{i=1}^{n+1} \leftarrow \{(\hat{r}_i, E_i, \hat{d}_i)\}_{i=1}^{n}, (\hat{r}_{n+1}, E_{n+1}, \hat{d}_{n+1})\}
\]

end

Where \(\xi\) is a uniform random variable generated with an RNG, \(\lambda_T\) is the mean free path for the different active interaction mechanisms, \(R(\theta, \phi)\) is a rotation matrix updating the direction of travel and \(p\) is the joint interaction probability for all the different interaction types \(q_i \in \{1, \ldots, k\}\). All the presented expressions in Algorithm 1 are obtained from the physics derived in the PENELOPE manual [50].

Thence, each simulated trajectory is characterized by a series of states \(\{(\hat{r}_n, E_n, \hat{d}_n)\}_{i=1}^{n}\) where \(\hat{r}_i\) is the position of the particle at the \(i\)-th event in the simulation and \(E_i\) and \(\hat{d}_i\) are the energy and the direction after the \(i\)-th scattering event.

**Statistical averages and uncertainties**

Now let us evaluate some quantity of interest \(Q\) based on all the simulated showers, this quantity is evaluated as the average score from the \(N\) simulated random showers. Hence, referring back to Section 3.1 this quantity can be expressed as

\[
Q = \int q(x)p(x)dx
\]

(3.7)

where \(X \sim \mathcal{X}\) is a random variable with PDF \(p(x)\) and \(q(x)\) is referred to as the score [50].

Furthermore, as commonly the case in complex simulations, the PDF is unknown, and the \(p(x)\) is now not only considering one trajectory but instead a cascade of random interaction events each with its own characteristic PDF [52]. Using the definition of MC estimate in Equation (3.2), the integral (3.7) can be written as

\[
Q_{\text{MC}}^N = \frac{1}{N} \sum_{i=1}^{N} q_i
\]

(3.8)

where \(q_i = q(X_i)\) is the score at the \(i\)-th simulated shower. The corresponding statistical uncertainty of the estimate is furthermore given by

\[
\sigma_Q = \sqrt{\frac{\text{Var}(q)}{N}} = \sqrt{\frac{1}{N} \left( \frac{1}{N} \sum_{i=1}^{N} (q_i^2 - Q_{\text{MC}}^N^2) \right)}
\]

(3.9)
providing a 99.7% confidence bound of the estimate, \((Q^\text{MC}_N \pm 3\sigma_Q)\) [50]. This considered quantity \(Q\) of interest is often the average energy deposited \(E_{\text{dep}}\) in some geometry considered, or the depth-dose distribution \(D(z)\). The depth-dose distribution is "defined as the average energy deposited per unit depth and per incident electron" [50] and will be the main quantity of interest in this thesis. Furthermore, we define a discretization interval in the range \([z_{\text{min}}, z_{\text{max}}]\) which contains the area of interest in the simulation output. This interval is uniformly discretized into \(M\) bins \((z_{k-1}, z_k)\) \(k = 1, \ldots, M\). Now we define \(e_{ij,k}\) as the "amount of energy deposited into the \(k\)-th bin by the \(j\)-th particle of the \(i\)-th shower" [50]. Thence the average energy deposited in each bin can be formulated accordingly

\[
E_{\text{dep},k} = \frac{1}{N} \sum_{i=1}^{N} e_{i,k}, \quad \text{where} \quad e_{i,k} \equiv \sum_{j} e_{ij,k}
\]  

(3.10)

with the statistical uncertainty

\[
\sigma_{E_{\text{dep},k}} = \sqrt{\frac{1}{N} \left( \frac{1}{N} \sum_{i=1}^{N} (e_{i,k}^2 - E_{\text{dep},k}^2) \right)}
\]  

(3.11)

and the \(k\)-th bin MC depth-dose distribution and its statistical uncertainty are given by

\[
D_k \equiv \frac{E_{\text{dep},k}}{z_k - z_{k-1}} \quad \sigma_{D_k} \equiv \frac{\sigma_{E_{\text{dep},k}}}{z_k - z_{k-1}}.
\]  

(3.12)

In summary, the objective of simulation of radiation transportation equation boils down to recover the dose distribution \(D\) in some considered volume. Again, this recovered dose include some statistical uncertainty referred to as noise. In theory, for a large number of particle simulations this noise is negligible but "the calculation time becomes prohibitive as the number of simulated particles is increased" [55] and hence in practice there will always be some present noise, see Figure 3.3. In simple terms, the simulated outcome with this present noise can be expressed accordingly

\[
\tilde{Q}_k = \begin{cases} 
Q_k \sqrt{\frac{\text{Var}(Q_k)}{N}} & \text{if the } k\text{-th bin has obtained at least one sample} \\
0 & \text{otherwise}
\end{cases}
\]  

(3.13)

where the quantity at the \(k\)-th bin has score zero if there exist no samples in the considered bin and otherwise has some statistical uncertainty present. The uncertainty can furthermore be reduced with different variance reduction techniques such as the previously reviewed weighted duplication of particles, but the computational time still remains as an obstacle. Next, an overview of current denoising methods for MC dose distribution calculation will be presented.
(a) $N = 10^4 (81\%)$.  
(b) $N = 10^5 (30\%)$.  
(c) $N = 10^6 (10\%)$.  
(d) $N = 10^7 (3\%)$.  
(e) $N = 10^8 (1\%)$.  
(f) $N = 10^9 (-)$. 

Figure 3.3: Example of estimated dose distribution using MC simulation techniques. Here the estimated dose is presented at a center slice, and intensity maps of the dose for different number of primary photon particles $N$, are shown in the sub-figures together with the relative error. Here $N = 10^9$ is considered as the reference.

**Denoising methods for MC dose distributions**

Previously published dose distributions denoisers involves filter methods such as wavelet threshold denoising (WTD) [16] where the dose distributions are transformed into the wavelet space and small wavelet coefficients are removed and hence the distribution is reconstructed with the remaining coefficients. Other filtering methods used are locally-adaptive Savitzky-Golay (LASH) filter [30] which is a 3D generalization of a Savitzky-Golay filter, or the adaptive anisotropic diffusion filter [42]. Furthermore, the iterative reduction of noise methods which essentially cast the noise reduction into a minimization problem introduces bias to obtain smoothing in less curvature regions [21].

Recent methods involve denoising using Deep Neural Networks. J. Asensi Madrigal investigates the denoising performance of MC dose distributions in proton therapy, with single shooting using the familiar U-net and DenseNet architectures [6]. He states that the deep neural networks outperform the state-of-art denoising of MC dose distributions.
Chapter 4

Statistical Learning

This chapter aims to give an overview of recurrent concepts used throughout the report and further present the general ideas in statistical learning (SL). In SL theory the core objective is studying the problem of inference. This means that we aim to gain knowledge, construct a predictive function or construct some decision rule based on some data studied. More specifically, gaining knowledge, for example, indicates detect key features in the data studied or simply estimating parameters for some parametric family of distributions. An example is the likelihood function, which quantifies the probability of the parameters of some statistical model, given a data set \( \{X_i, Y_i\}_{i=1}^n \). Statistical learning modeling is associated with a training and test phase. The training phase is when the model is learned through a set of samples referred to as the training set \( T \) and gain knowledge about the data distribution studied. In the test phase, the obtained model, from the training phase, is tested on unseen data referred to as the test set \( S \). These two phases aim to see how well the model capture features in the data and how well the model generalizes.

In statistical learning (SL) and machine learning (ML), methods are referred to as supervised or unsupervised and most statistical learning problems fall into one of these categories\(^1\). Supervised learning is related to a structure of a mapping \( f : X \rightarrow Y \), i.e. an input has a corresponding response or an output that the mapping function \( f \) aims to predict. The output can be either categorical, nominal or scalar, in contrast to the general regression methods which require the output to be quantitative. Thence in supervised learning, the transfer function \( f \) is modeled and its objective might be to perform predictions. The modeling therefore incorporates estimate model parameters \( \theta \) of the predictive function \( f_\theta \). Some familiar examples of supervised methods in ML are regression models and classification algorithms, where the latter aims to categorize some input to some corresponding categorical output. Let us now consider the unsupervised setting, which has a different structure. In simple terms, an unsupervised set is a set of data \( \{X_i\}_{i=1}^n \) which is not associated with some response. Further, an unsupervised model aims to capture features of the data distribution and hence the core interest is the underlying data distribution \( f_{data}(x) \). Nevertheless, the power of unsupervised learning is still very strong as it aims at detecting patterns or latent features in the data studied. A well-known example of an unsupervised SL method is the familiar K-means clustering, which based on some metric derives disjoint clusters in the data.

\(^1\)Some models are considered to be semi-supervised
Some classical ML methods, supervised and unsupervised, involves; regression methods, clustering methods, Bayesian learning methods, support vector machines as well as neural networks. Neural networks have been studied for decades in academia but have in the recent years exploded in power regarding applications in the industry. It is commonly stated that the main factor to this is the rapid increase of computational resources such as Graphics Processing Units which are enabling fast parallel computations. However, leaving NN to Chapter 5, this chapter will as mentioned focus mainly be on the overall theory of statistical learning and machine learning (ML), and some of the issues that these tackle with.

4.1 Inference problem

Let us now review inference problem where we restrict the considered example to be a supervised setting for simplicity. Assume that we are considering some vector space \(X\) including all possible inputs and some vector space \(Y\) including all possible outputs, jointly constructing the product space \(X \times Y\). Furthermore, let us define a sample to be a random outcome from this product space and denote it \((x_i, y_i)\). In SL theory it is assumed that there exists some unknown underlying probability distribution \(p_{X \times Y}(x, y)\) over the product space \(X \times Y \in \mathcal{X} \times \mathcal{Y}\) from which these samples originates. Thence, assuming that each sample \((x_i, y_i)\) is drawn from this distribution and therefore \(S = \{X_i,Y_i\}_{i=1}^n \sim p_{X \times Y}(x,y)\) and \(T = \{X_i,Y_i\}_{i=1}^n \sim p_{X \times Y}(x,y)\).

In this supervised setting the task is to find some mapping function \(f: X \rightarrow Y\) such that \(f(x) \approx y\), which is referred to as the inference problem. However, if there are no restrictions of this function (non-parametric model), this function can take forms over all parametric families and the problem may have a large set of solutions. The space in which all of possible mapping functions exist will be called the hypothesis space \(\mathcal{H}\), where \(f \in \mathcal{H}\) and is the space which the algorithm searches through aiming to find the optimal \(f^*\). Now, let us define a very central concept in SL, the loss function \(\mathcal{L}(x,y)\), which is a metric defining a difference measure between the predicted value \(f(x)\) and the true value \(y\). Then the expected risk can be defined as

\[
R(f(x), y) = \int_{\mathcal{X} \times \mathcal{Y}} \mathcal{L}(f(x), y)p_{X \times Y}(x,y)dxdy \tag{4.1}
\]

which is the objective function in finding the optimal function \(f^*\) in the hypothesis space. The search of the optimal function can therefore be cast into a minimization problem, aiming to minimize the expected risk

\[
\min_{f \in \mathcal{H}} R(f(x), y) \tag{4.2}
\]

with solution \(f^* = \inf_{f \in \mathcal{H}} R(f(x), y)\). However, the probability distribution \(p_{X \times Y}(x,y)\) is generally unknown and a sample set is used to derive a MC estimate of the quantity instead. Reflecting back to Chapter 3, the MC estimate of the integral (4.1) is formulated according to (4.3), and is referred to as the empirical risk

\[
R_{\text{emp}}(f(x), y) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x_i), y_i) \tag{4.3}
\]

derived based on some training set \(\mathcal{T}\) of size \(n\). Further, the algorithm used in deriving the optimal function \(f^*\), based on the empirical risk, is called empirical risk minimization.

\[^2\text{Among other methods}\]
4.2 Parametric and non-parametric models

Models in ML are usually considered to be parametric or non-parametric. The classification of the model into one of these categories is based on which assumptions the model makes about the unknown function \( f \). A model which is non-parametric makes no assumptions about the functional form or shape of \( f \) and hence derives the function based solely on the data. An example of a non-parametric approach is clustering methods using some specified metric, where the function \( f \) learns the Voronoi region as the decision rule. In this manner, the function learns the mapping solely on the considered data samples. In parametric models, however, we make an assumption about the functional form or shape of the unknown function \( f \). For example, in regression tasks, we assume that the response has a linear dependence structure on the features. The modeling is then reduced to estimating the model parameters \( \beta \),

\[
Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n
\]

for a feature in an \( n \)-dimensional space, i.e. \( X \in \mathbb{R}^n \). Hence, based on a set of samples the model parameters can be estimated \( \hat{\beta} = \hat{\beta}_0, \ldots, \hat{\beta}_n \), using least squares. The main disadvantage of non-parametric models is that they require a large number of samples, in contrast to parametric models. However, since no assumption about the model function is made, the complexity of \( f \) is clearly larger than e.g. a linear parametric model and other parametric models. Although a too complex model is not necessarily better as it often leads to overfitting the model.

4.3 Training and test data in statistical learning

As been repeatedly stressed, in statistical modeling we aim to find an optimal mapping function \( f^* \) given some data. Additionally, we want the model to be robust to small changes in the input such as additional noise, and the model would ideally perform equally good on unseen data. This is why dividing our data into training and test set is of most importance. Again, the test set is a set of samples which the optimized model has not seen while the algorithm has searched through the hypothesis space. The training set is the set which the model actually has been trained under, and in which the model parameters are updated according to. Dividing a data-set into these two disjoint set help one detect overfitting and analyze the robustness of the model. For example, if the model performs well on the training data but poorly on the test data, the model is likely to be overfitted.

4.3.1 Data augmentation

Data augmentation is a central concept in ML which refers to transforming available data samples to increase the training set size. This transformation could for example, in the case where the samples are images, be rotating objects in the image, flipping the image or translating objects. Furthermore, other augmentations transformations used are enlarging (scale transformation) and changing of channel values in RGB images. The reason why data augmentation is effective is since, given that the augmentation is properly done, the model will consider the augmented samples independent from the original. Hence, it is a sufficient way of enlarging your training set. However, if the augmentation is poorly augmented the network will instead tend to overfit the data.
A data set containing \( n \) samples can directly be duplicated to \( 2 \times n \) only using one transformation per sample. However, the quality of the data set cannot be improved using data augmentation, as in general the model obtained is only as good as the data you feed it. A simple visualization of data augmentation is shown in Figure 4.1.

![Visualization of data augmentation](image)

Figure 4.1: Visualization of data augmentation of a original sample (left), augmented samples (right). Note that there are a number of ways the data can be transformed to obtain new samples.

### 4.4 Overfitting and underfitting

Overfitting is a concept in statistical learning that is used when a model fits to the noise of the data more closely than the data structure itself. It is closely related to the variance-bias tradeoff, which is a known optimization problem in statistical learning tasks. The variance-bias tradeoff means that the model cannot minimize the two quantities simultaneously, as they are often negatively correlated. In short, the bias is the quantity with which the mean deviates from the true mean, whereas the "variance is refer\(^\text{3}\) to the amount by which \( \hat{f} \) would change if we estimated it using a different - data set\(^3\) \({\text{27}}\). Let us now connect these concepts to flexible non-parametric models. A flexible function \( f \) can fit close to the data and hence have a low bias. However, the robustness is low and variance of the estimate is often large, this is a simple example of an overfitted model. On the contrary, underfitting refers to a model which is not capturing the underlying structure of the data. Underfitting generally occurs when the model is not allowed enough complexity to capture the true data generating distribution. An example of an underfitted model is when a linear regression is parametrized to fit the true underlying distribution \( x^2 \), as shown in Figure 4.2.

\(^3\)[word] means that the quote has been altered with word
Figure 4.2: Visualization of underfitting (left) and overfitting (right), the true data generating function (middle) is $f(x) = x^2$ and the samples are generated with some additional Gaussian noise $\epsilon \sim \mathcal{N}(0,1)$.

4.4.1 Methods to detect and reduce the risk of overfitting

Dividing the data into a test and training set is as previously mentioned a convenient way to detect overfitting of a model as an overfitted model will perform poorly on the test set and superior on the training set. Other classical methods to detect overfitting and estimate the prediction error is to use cross-validation, where the modeling is essentially performed on altering the ordering of test and training data in a $K$-fold manner.

Now let us shortly focus on what can be done to decrease the risk of overfitting. Firstly, as described in previous the section models are exposed to the variances bias trade-off. A model with very low bias has a high risk of being overfitted to the data as the model thence is often very complex. In some cases, it might be sufficient to include a regularizer in the modeling to penalize the model for high complexity $f$. Then a regularizer term is essentially added in the objective function which is engineered using some regularizer parameter $\gamma$. An example of such regularizer in a deep NN will be presented in Chapter 6. More classical models which use the same strategy of penalizing complex models are Ridge and Lasso-regression, where the latter method also performs feature selection. Another method for reducing the magnitude of overfitting in deep NN is batch normalization which will be presented in more detail in Section 5.3.
Chapter 5

Neural Networks

Artificial neural networks, abbreviated ANN, a framework resembling the network structure of brain neurons using electric circuits, was initially introduced in 1943 through a paper by McCulloch et al. [39]. More specifically, their main idea was that the artificial neurons connected with adjacent neurons through links or edges could resemble the structure of dendrites and axons in the biological brain neurons. Neural networks has been an active research field ever since, although, the application of these NN was mainly limited to academia for decades due to lack of computational power and the shortage of large databases. The first notable achievement by the NN was solving the classification problem exclusive-or [18], with Rosenblatt’s perceptron, where the linear input was transformed into a nonlinear binary decision rule through a set of hidden states. As a matter of fact, the introduction of learning of parameter weights and backpropagation of the gradients was important keys in solving this problem [49]. Previous to this, the modelers manually tried to engineer the network weights for the specific task. In recent years, deep NN, or specifically convolutional neural network (CNN) has been successfully applied on a variety of tasks such as classification [31], dimension reduction [10], denoising [65] and reconstruction applications\(^1\).

Neural networks, as the name implies, uses this previously mentioned neural structure to construct what is referred to as a network. The sorting and ordering of the units or neurons in the network is referred to as the architecture and is commonly visualized in a directed graph. NN is often categorized into feedforward neural network (FNN) or recurrent neural network (RNN), where the former has an acyclic directed graph structure forcing the information to flow forward. RNN is essentially an extension of an FNN which includes a feedback connection yielding a cyclic structure of information propagation. RNNs are often used in sequence modeling [22] where the information flows from one state to a subsequent. However, in this chapter and this thesis, the main focus will be on FNN.

\(^1\)Among other applications
5.1 Feedforward Neural Network (FNN)

The feedforward neural network (FNN), also called MLP or deep FNN, is the simplest form of ANN. Further, it is generally considered to be the simplest to implement in terms of modeling and training. The FNNs are acyclic and hence are conveniently represented by a directed acyclic graph (DAG), which describes in which fashion the information flows between the nodes. Let us now define the general FNN in more detail. The objective of an FNN is to define some mapping often denoted \( f(x; \theta) \) or \( f_\theta \) which is based on some parameter values \( \theta \). In a supervised setting, this entails that an input \( x \) is to be mapped to a response \( y \) through this network function \( f_\theta : x \mapsto y \). A DAG of a general single layer FNN structure is presented in Figure 5.1.

![DAG of an FNN with one hidden layer.](image)

Figure 5.1: DAG of an FNN with one hidden layer.

The units linking the input \( x \) to the output \( y \) is referred to as hidden layers, for example, the network in Figure 5.1 comprises one hidden layer. The input can take on various different shapes and structures, depending on which data is considered. However, the output structure is task dependent. For example, if the network task is binary classification, the output shape is naturally a vector of size two. A visualization of a template architecture for a binary classification task is presented in Figure 5.2.

![A visualization of a fully connected layer (FCL) for e.g. binary classification, where the input is a vector of 6 features and the output a vector of size 2.](image)

Figure 5.2: A visualization of a fully connected layer (FCL) for e.g. binary classification, where the input is a vector of 6 features and the output a vector of size 2.

Furthermore, layers are said to be **fully connected** if all the outputs are connected to all of the input components, as in the layers in Figure 5.2. If this is not the case, meaning the output is dependent on a subset of the input, the layer is said to be a **dense** layer. This dependence structures of a neuron on ancestor neurons is referred to as the neuron’s **receptive field** and in an FCL each output and hidden neuron has the whole input as their receptive field. We will return to this concept in a later chapter and stress its power. Returning back to the general formulation of FNN, expanding the network to contain multi-layer structures. The input is transferred to the output through a set of hidden layers described by a set of transition functions \( f^{(l)} \), where \( l \) denotes the ordering of layers \( l \in \{1, \ldots, L\} \) and \( L \) indicate the output layer. The single-layer network presented in Figure 5.1 is referred to as **shallow** in contrast to multilayer architectures which are called **deep** networks. An example of a deep \( L \)-layer network is shown in Figure 5.3.
Figure 5.3: A visualization of a deep FNN with \( L - 1 \) hidden layers and input \( x \), yielding the output \( y = f^{(L)} \circ \cdots \circ f^{(1)}(x) \).

where the input is transferred through a set of \( L \) transition functions where the operator \( \circ \) defines the how the functions operate, for example \( f^{(i)} \circ f^{(j)}(x) = f^{(i)}(f^{(j)}(x)) \) [23]. In deep networks this mapping function can thence be expressed as a composition of functions, \( f(x) = f^{(L)} \circ \cdots \circ f^{(1)}(x) \), where each \( f^{(l)} \) can be interpreted as the transition between hidden layers. Furthermore, \( f \) is parameterized by the network parameters of each layer \( \theta = \{\theta^{(l)}\}_{l=1}^{L} \), constructing a complex modeling function defining the FNN. Let us now look at each neuron in more detail and gain knowledge about what the transition \( f^{(l)} \) incorporates.

5.1.1 Artificial neuron

A neuron is comprised of four main components, an input \( x \), a vector containing the neuron weights \( w \), the neuron bias \( b \) and lastly an activation function \( g \). In each neuron, the linear input is transformed into a non-linear response referred to as the output or activation. Mathematically, this transformation consists of a point-wise multiplication of the input with the connection neuron weights and subsequently adding the neuron bias. Further, the sum is activated through the activation function \( g \), which is a non-linear function and will be discussed in more detail shortly. A neuron is shown in Figure 5.4 where the output is formed into \( g(\sum_{k=1}^{3} w_{jk}x_k + b_j) = y_j \) or in matrix notation \( g(w_j \cdot x + b_j) \).

Figure 5.4: Visualization of the transformation of input in a neuron \( j \). Note that the input to the neuron can also be an output of a previous layer, i.e. \( x_i = y_i \), yielding a subsequent chain of neurons.

Note that the input \( x \) fed to the neuron is transformed into a scalar output \( y_i \). However, the number of neurons in each layer can be larger or less than the input, yielding a wider structure, this is referred to as the width of the network. Further, the neuron structure presented above is a single neuron architecture, although, in NN and essentially deep NN these neurons are connected in large network structures, and hence the input to a neuron is often an output of an ancestor neuron. For convenience, we now need to introduce
some important notations\textsuperscript{2}. First, let $j$ denote the considered neuron and $l$ the considered layer. Thence, let $\mathbf{w}^l$ be defined as the matrix of weights for the $l$th layer and $b^l$ bias in layer $l$. These neuron parameters are what previously have been denoted $\theta^l$, i.e. $\theta^l = (\mathbf{w}^l, b^l)$. Lastly, we define $g_l$ to be the activation function in layer $l$ and $a^l_j$ to be the activation of the $j$th neuron in the $l$th layer. To keep track of the variables introduced, the reader is referred to table (5.2) which is influenced from [45].

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_l$</td>
<td>number of neurons in layer $l$</td>
</tr>
<tr>
<td>$b^l_j$</td>
<td>bias for layer $l$, in the $j$th neuron</td>
</tr>
<tr>
<td>$\mathbf{w}^l$</td>
<td>matrix of weights for layer $l$</td>
</tr>
<tr>
<td>$w^l_{jk}$</td>
<td>entry in $j$th row and $k$th column of $\mathbf{w}^l$</td>
</tr>
<tr>
<td>$a^l_j$</td>
<td>output of layer $l$ in neuron $j$</td>
</tr>
<tr>
<td>$g_l(\cdot)$</td>
<td>activation function in neuron $l$</td>
</tr>
</tbody>
</table>

Table 5.1: The introduced network notations

Further, let $z^l = \mathbf{w}^l a^{l-1} + b^l$ be referred to as the intermediate quantity which is sent to the activation function. The operation $g_l(\mathbf{w}^l a^{l-1} + b^l)$ will recurrently be denoted $g_l(z^l)$. Moreover, the general neuron activation of an input to the $l$th layer is formulated

$$a^l_j = g_l(\mathbf{w}^l_j \cdot a^{l-1} + b^l_j), \quad \forall j \in \{1, \ldots, n_l\}$$

forming a vector input $a^l = (a^l_1, \ldots, a^l_{n_l})$ to the subsequent layer $l + 1$. Lastly, we emphasize that $a^L = f(x)$ correspond to the final output. These notations will be of great importance to keep track of the forthcoming derivations.

\textsuperscript{2}Notations are influenced from [45]
5.1.2 Training a feedforward network

Now we will consider the task of choosing the set of network parameters \( \theta = \{(w^l, b^l)\}_{l=1}^L \), which is referred to as training the network. Initially, the network parameters was manually engineered, and this approach were referred to as the prior dominant approach \[22\]. However, the deep learning approach attacked this task in a more efficient way. Instead, the aim was to learn the parameters through an optimization algorithm referred to as the gradient descent, performed under the training or learning phase. This optimization algorithm, along with the familiar backpropagation, will therefore be the main focus in this subsection.

Gradient descent

Reflecting back to Chapter 4 and the section treating the training phase, the training of an FNN is essentially the phase in which the network parameters optimized to find the optimal mapping function. This objective, as stated in Section 4.1, could be formulated into an optimization problem where the aim was to minimize the empirical risk \( R_{emp} \) which subsequently will be denoted \( \mathcal{L} \). Again, for a given loss function, the network aims in the training phase to minimize the risk with respect to the network parameters \( \theta \), i.e.

\[
\min_{\theta} \mathcal{L}(f_\theta(x), y)
\]

(5.1)

where \( f_\theta(x) = a^L(x) \) is the output from the FNN and \( y \) is a vector containing the corresponding true target response in a supervised setting. Formally, the gradient descent method entails solving a multi-dimensional optimization problem initialized at some point and update the parameters according to the gradient direction resulting in the largest negative change for the current state, leading to a local or global minimum. The parameter updates are done using a step size referred to as the learning rate and will further be denoted \( \eta \). An example of the gradient descent is presented in Figure 5.5, where this initialization is denoted \( \theta_0 \). Thence, if an optimum has been found and if the optimization problem is convex\(^3\), the algorithm will finally arrive at the global optimal point.

\[ \theta_0 \]
\[ \theta_1 \]
\[ \theta_2 \]
\[ \theta_3 \]
\[ \theta_4 \]
\[ \theta_5 \]

Figure 5.5: Visualization of gradient descent algorithm for a simplified 2-dimensional parameter space.

In summary, the gradient descent algorithm in deep learning, aims to update the network parameters according to the direction in which the gradients \( \nabla_{\theta} \mathcal{L} \) indicates the greatest

\(^3\)However this is seldom the case in Deep learning
loss reduction. Further, the initialization of the network parameters \( \theta_0 \) is often done using an isotropic Gaussian distribution.

Let us now dive into the updates of parameter values in more detail, initiating with the change in parameter values which we here consider to be a \( n \)-dimensional vector \( \Delta \theta = (\Delta \theta_1, \ldots, \Delta \theta_n)^T \). Furthermore, the gradient of the loss function with respect to the network parameters is given by

\[
\nabla_{\theta} \mathcal{L}(\theta) = \left( \frac{\partial \mathcal{L}(\theta)}{\partial \theta_1}, \ldots, \frac{\partial \mathcal{L}(\theta)}{\partial \theta_n} \right)^T
\]

(5.2)

and will be used to approximate the change of the loss dependent on the current network parameters. Further, if we consider a small change \( \Delta \theta \) of the parameters, the approximate change of the loss can be expressed according to

\[
\Delta \mathcal{L}(\theta) \approx \nabla_{\theta} \mathcal{L}(\theta) \Delta \theta = \sum_{i=1}^{n} \frac{\partial \mathcal{L}(\theta)}{\partial \theta_i} \Delta \theta_i
\]

(5.3)

where the former is in matrix-notations and the latter formulated into a sum of element-wise multiplication. This quantity is thence the approximate change in loss, which we are aiming to maximize. Although first we need to identify in which direction the loss is reduced, to make sure that the parameters are updated correctly. Assume that the change in network parameters with the step-size \( \eta \in \mathbb{R}^+ \) can be expressed

\[
\Delta \theta = -\eta \nabla_{\theta} \mathcal{L}(\theta)
\]

(5.4)

thence, using the approximate change in loss (5.3), one can attain the direction in which the loss reduction is maximized. The reformulation is thence given by

\[
\Delta \mathcal{L}(\theta) \approx -\eta \nabla_{\theta} \mathcal{L} \nabla_{\theta} \mathcal{L}^T = -\eta \| \nabla_{\theta} \mathcal{L} \|^2,
\]

(5.5)

note that the last quantity is strictly negative and therefore updating the parameters in the direction \( \Delta \theta \) would involve a reduction of the loss. The parameters can thence be updated iteratively, in a similar manner as the Euler’s method,

\[
\theta_{i+1} = \theta_i - \eta \nabla_{\theta} \mathcal{L}(\theta_i)
\]

(5.6)

until an optimum has been reached. Now we have derived the general updating formula of the network parameters. The weights and biases updating formulas are thence

\[
\mathbf{w}_{i+1} = \mathbf{w}_i - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}_i}, \quad b_{i+1} = b_i - \eta \frac{\partial \mathcal{L}}{\partial b_i}
\]

(5.7)

The influence of the learning rate \( \eta \) is of most importance in the optimization or learning of the network. Similarly as for general numerical solvers, a too high learning rate can overshoot the updates and result in a poor local minimum for the network. A too low learning rate will decrease the learning and thence the convergence rate will be slow, see Figure 5.6.
A suitable learning rate $\eta$. A too large learning rate $\eta$.

Figure 5.6: Simple visualization of the influence of the learning parameter $\eta$. In (a) the network parameters approached the local minimum slowly but robustly while in (b) the step size of the updates overshoots the local minimum.

Choosing the right learning rate for a network is not straightforward and one often needs to find the proper $\eta$ in a trial and error manner. Common choices of $\eta$ are $\eta \in \{10^{-3}, \ldots, 10^{-5}\}$. Further, if the chosen learning rate is improper this will forthrightly be detected though large diverging movements of the loss during training, see Figure 5.6b.

**Backpropagation**

In the training phase, the forward propagation previously discussed propagates til it produces some scalar loss $L(\theta)$, based in the present parameters of the network. Thereafter the performance is evaluated and the gradient of the loss is used to update the current parameters. Naturally, we aim to compute an analytical expression of gradients, however, this can be severely computationally expensive. Backpropagation refers to the algorithm of updating the parameters $\theta$ based on this cost, in a fashionable way, reducing the computational cost significantly. The concept of backpropagation is often misinterpreted as the algorithm of optimizing the network parameters, when it in fact solely refers to the algorithm of updating the gradients, in a recursive manner. Before moving further, one has to assure that the considered loss function fulfills the two requirements in order to perform backpropagation. First, the loss function must be able to be decomposed into a sum of losses for each individual sample. Second, the loss function must be a function of the output of the network $a^L$ [45]. Now, assume that we are considering a loss function that fulfills these two requirements. Let the error in layer $l$ for neuron $j$ be defined according to

$$\delta_j^l = \frac{\partial L}{\partial z_j^l}$$

and recall the expression of the intermediate quantity $z_j^l = w_j^{l-1}a^{l-1} + b_j^l$ [23]. Further let us consider the error in the final layer $l = L$ in some neuron $j$. This $\delta_j^L$ can be derived using the familiar chain rule of the derivative in Equation (5.8). The chain rule yields

$$\delta_j^L = \frac{\partial L}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial L}{\partial g_L(z_j^L)} \frac{\partial g_L(z_j^L)}{\partial z_j^L} = \frac{\partial L}{\partial g_L(z_j^L)} g'_L(z_j^L)$$

35
which can be cast into vector form \( \delta^L = \nabla_a \mathcal{L} \odot g'_L(z^L) \), where the operator \( \odot \) denotes the Hadamard product [45]. Now we have derived a recursive expression of the error in each layer for each neuron. Thus the error in the ancestor layer is a function of descendant layer error, accordingly

\[
\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot g'(z^l).
\]  

(5.10)

Note that the entity \( z_j \) incorporates the two parameters \( w^l_j \) and \( b^l_j \), hence, from expression (5.9) the elaboration of the derivative yields the gradient loss with respect to each of the network parameters

\[
\frac{\partial \mathcal{L}}{\partial b^l_j} = \delta^l_j
\]  

(5.11)

\[
\frac{\partial \mathcal{L}}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j
\]  

(5.12)

and the updating formulas of the network parameters can be implemented straightforwardly. The calculation of gradients can be computed recursively though Algorithm 2.

---

**Algorithm 2 Backpropagation**

**Input:** \( x \)

Set the corresponding activation \( x \rightarrow a^1 \) for the input layer.

**for** \( l = 2 : L \) **do**

\[
\begin{align*}
  z^l &= w^l a^{l-1} + b^l \\
  a^l &= g_l(z^l)
\end{align*}
\]

**end**

Set error in the output layer to \( \delta^L = \nabla_a \mathcal{L} \odot g'_L(z^L) \)

**for** \( l = (L-1) : 2 \) **do**

\[
\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot g'_l(z^l)
\]

**end**

**Return:** Gradients \( \frac{\partial \mathcal{L}}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \) and \( \frac{\partial \mathcal{L}}{\partial b^l_j} = \delta^l_j \) \( \forall j, k \)

The first **for**-loop is commonly referred to as the *forward pass*, and essentially consists of forward propagation of the input. The second loop is called the *backward pass* and mainly derives the error in each layer [22] [23]. Thence, this presented backpropagation algorithm enables the network to be optimized with gradient descent in an efficient way.
Stochastic gradient descent

Performing the previously presented gradient descent on a large data set is often too complicated or in some cases unfeasible to perform\(^4\). Thence, instead the stochastic gradient descent, abbreviated (SGD), is performed on a smaller batch of samples from the considered data set. The SGD is based on the assumption that the samples are \(i.i.d\) and thence the empirical risk can be replaced with the average empirical risk based on a randomly drawn data sample. This set of samples are referred to as a batch of the full data set and are used for deriving an estimate of the gradient \(\nabla_{\theta} L_{emp}(\theta)\). Reformulating the expression of updating the network parameters to be based on the average empirical risk

\[
\nabla_{\theta} L_{emp}(\theta) = \frac{1}{k} \sum_{i=1}^{k} \nabla_{\theta} L_{emp}^{(k)}(\theta)
\]

for a batch of samples of size \(k\). When all of the data batches have been fed to the estimator once, one epoch is entailed. In general, training of deep FNN results in hundreds or thousands of epochs. Furthermore, the reformulated updating formula is hence given by

\[
\theta_{t+1} = \theta_{t} - \eta \frac{1}{k} \sum_{i=1}^{k} \nabla_{\theta} L_{emp}^{(k)}(\theta)
\]

and the method is referred to as stochastic as it is based on a set of random samples. Other methods involving similar strategies of performing gradient descent is the RMSprop and Adam, where the latter can be viewed at as a combination of first and the previously presented SGD with momentum.

Some remarks

As opposed to convex optimization, deep neural networks can be non-convex problems which can have multiple local minima depending on the loss one chooses to minimize. The optimum that has been found has the risk of being only a local optimum. Thence, the method batch normalization can in some cases be used to help the network of jumping out of the local minimization point, which is a method that will be discussed in more detail shortly. Furthermore, other problems may occur during training of an FNN such as saturation of gradients or ill-posed Hessians, where the first will be described more in detail in the subsequent Section 5.1.3. This saturation essentially stops the training of the network, as the gradient has saturated. This problem is a well-studied problem in NN and is dependent on the structure of the activation functions. This will be the next sections main content.

\(^4\)Due to computational effort
5.1.3 Activation functions

In this section, we will consider the previously presented activation function $g(\cdot)$ in more detail. As mentioned, the main purpose of the activation is to transform a linear input to a non-linear response. Hence, naturally the activation function is a non-linear function. Let us shortly consider Figure 5.7 and remind us of the neuron structure in Figure 5.4.

The activation function can be viewed as a regularizer, which modulates which information that the network should propagate further. For example, recalling the binary classification shortly discussed in Section 5.1.1, the activation function modulates which neurons, for a specific input, that should be activated in the classification of each true response. A very nice visualization of the firing in the activations can be found in the paper *An Interactive Node-Link Visualization of Convolutional Neural Networks* by Adam Harley [24].

The activations play a key role in computing the gradients. As have been seen, the derivative of the activation w.r.t. the intermediate quantity must be evaluable in order to perform the gradient descent smoothly. The simplest activation function $g(\cdot)$ is naturally the linear one, mapping a linear input to a linear output. However, as have been stressed, the aim of the activation function is to map linear inputs to a nonlinear output and thence nonlinear activation functions are used in practice. These are for example the soft-plus function, sigmoid function, the Rectified Linear Unit (ReLU) or the leaky ReLU, which are all shown in Figure 5.8.
ReLU and Leaky ReLU

The rectified linear unit, in short the ReLU, is a function which modulates the input to be mapped to the positive real axis, using the max function.

\[
\text{ReLU}(z) = \max(z, 0)
\]

Hence each intermediate quantity \( z \) is mapped to the positive domain \( \mathbb{R} \rightarrow \mathbb{R}^+ \). The ReLU has the very nice property of being is differentiable for almost all \( z \), \( \forall g'(z)|_{z=0} \neq 0 \). Furthermore, one of the main advantages of the rectified linear unit is that it handles the saturation of gradients smoothly, in the positive domain. However, this does not hold for the negative axis since \( \forall g'(z)|_{z<0} = 0 \) meaning that the problem of saturation of gradients still stand in the negative domain. This problem can be addressed with modifying the ReLU to the leaky ReLU, in which the gradients of the negative domain doesn’t saturate due to the parameter tilt \( \alpha \),

\[
\text{leaky ReLU}(z | \alpha) = \max(z, \alpha z)
\]

where the parameter \( \alpha \) is typically chosen to be a small positive number. Again, the leaky ReLU has the same differentiable qualities as the ReLU, but elegantly solves the problem with saturating gradients in the negative domain.

Sigmoid function

The familiar sigmoid function is a desired activation function foremost since it has the ability to transform a discrete input into a continuous response but moreover since it is differentiable in the whole domain. The sigmoid function is given by

\[
\sigma(z) = \frac{1}{1 + e^{-z}}
\]

and yields a response in \( \sigma(z) \in (0, 1) \). This activation function thence maps the input space to the narrowed range, \( \mathbb{R} \rightarrow (0, 1) \). This activation function, however, suffers from a large downside of saturating gradients outside the input interval \((-4, 4)\). Note how the derivative of the activation approaches zero outside this interval, see Figure 5.8.

Softplus

The softplus activation likewise has the property of modulating a discrete input into a continuous response. The activation is defined accordingly

\[
\text{softplus}(z) = \log(1 + e^z),
\]

and can be considered as a smoothed version of the previously presented ReLU activation, see Figure 5.8. The activation maps to the space \( \text{softplus}(z) \in (0, \infty) \) and is differentiable in the whole domain of \( z \). Nevertheless, the problem of saturates gradients still stand for the negative domain. Moreover, the derivative of the softmax w.r.t \( z \) is, in fact, the sigmoid function.
5.1.4 Loss functions

Ultimately, the last important general element for the network training is the loss function. As detailed in Section 4.1, the integrated loss over the data generating distribution is essentially the objective of the considered minimization problem. Recent work by Zhao et al. showed that the quality of the optimization results improves significantly with better loss functions, even when the network architecture is left unchanged [66]. However, what is a better loss function? The answer to this is directly dependent on the network task and some examples of this will be presented shortly. First, let us underline that the total empirical loss for a training sample $T = \{(x_k, y_k)\}_{k=1}^{K}$ is essentially the sum of all individual losses $L_{(x_k, y_k)}$, i.e.

$$L(\theta) = \frac{1}{K} \sum_{k \in K} L_{(x_k, y_k)}(\theta) \tag{5.19}$$

where $K$ is the number of samples in the training set. The choice of loss function hence directly reflects the average performance of the network $f_\theta(x)$. Again, a loss function must fulfill two criteria in order to be used for training a network [45], (i) the loss function must be a function of the network output, and hence also differentiable w.r.t. the network parameters. Second, (ii) the loss must be decomposed as a sum of individual losses $L_k$ over the training set. In this section, will the two most commonly used loss metrics that fulfill these criteria will be presented, namely the absolute difference metric $\ell_1$ and least squared metric $\ell_2$.

**Absolute difference $\ell_1$**

The absolute difference loss, commonly denoted $\ell_1$, quantifies the absolute value error between the target and the network prediction and is defined accordingly

$$L_{(x_k, y_k)}(\theta) = |f_\theta(x_k) - y_k|, \tag{5.20}$$

as a consequence, the error penalizes each location error equally. For example if the entities $x$ and $y$ are images, each pixel error in the image is penalized equally. This leads to very sharp predictions and the loss is often preferred in denoising tasks of higher patterns images [66]. Furthermore the derivative of the loss w.r.t. the input is simply $\text{sign}(f_\theta(x_k) - y_k)$.

**Squared difference $\ell_2$**

The squared difference loss, as its name states, is a metric based on the squared error in each prediction [64]

$$L_{(x_k, y_k)}(\theta) = (f_\theta(x_k) - y_k)^2. \tag{5.21}$$

The loss measure, in contrast to $\ell_1$, penalizes regions with higher difference considerably, while low error regions have a lower effect on the loss. This is due to the squared term of the loss. However, a well-known downside with this loss function is that it tends to over-smooth the predictions due to the property that the optimal solution to (5.21) is $f_\theta^*(x_k) = \mathbb{E}[Y_k]$.

The loss function that is chosen to find the optimal network parameters hence affects the obtained optimum. For example, if one wishes to train a network where the output is pointed towards a more smooth output, the loss function $\ell_2$ could be used. However,
if one wants to penalize for small deviations and obtain sharp predictions, $\ell_1$ or absolute difference can be used. Furthermore, one can engineer a loss function with the weighted $\ell_1$ and $\ell_2$, obtaining a loss function containing both of these metrics if both these qualities are of importance.

5.2 Convolutional Neural Network (CNN)

A convolutional neural network (CNN), also called a ConvNet, is a feedforward neural network specialized in handling data with spatial or temporal topological ordering [22], using a linear operation called convolution. In fact, "convolutional networks are simply neural networks that use convolution in place of general matrix-multiplication" [22]. Examples of such suitable inputs for CNNs are temporal data such as time-series or spatial data such as a 2-D grid of pixels in an image. While in the previously introduced FCL networks the topological information is forthrightly lost, the CNN take advantage of the spatial structure of the input, and hence the networks are conveniently used for image processing tasks such as segmentation or object detection. Further, the preceding deep fully connected layer engendered a huge set of network parameters, causing the training of these networks to be painful. This problem was fundamentally solved with the CNNs, which was comprised of sparse interactions and parameter sharing reducing the number of network parameters considerably. Hence CNN has been an important block in the research and development of deep neural network training and architectures.

This section will first present the basics of the mathematical formulation of convolution, and subsequently present the general CNN with additional important concepts. Further, the focus will be on the main ideas of CNNs and specifically the local receptive fields, shared weights and pooling.

5.2.1 Convolution layer

As quoted in the chapter introduction, basically the main dissimilarity between a convolutional layer and the perceptron discussed in Section 5.1.1 springs from the convolution operator, that replaces the matrix-multiplication of the input and the fully connected weights. Analogously, each layer is comprised of an activation function, input and layer parameters. However, now these parameters take form into a filter that is swept over the input creating what is referred to as a feature map. Now, let us first define the operation convolution.

The convolutional operator

Convolution operators are commonly used in computer vision and signal processing tasks and extract local patches of information through the sweeping of a kernel. Let us first consider two discrete 1-D functions, an input $x$ and a kernel $w$. Thence, the discrete convolution is defined accordingly

$$s(t) = (x * w)(t) = \sum_a x(t)w(t-a)$$ (5.22)

where $*$ is the convolution operator and the convolution results in a feature map $s(t)$. Further, the convolution is commutative meaning that the order can be changed $(x * w)(t) = (w * x)(t)$. Extending this into the multidimensional case, we should now consider
the convolution of an image. Consider a 2-D input image \( I \in \mathbb{R}^{H \times W} \) convoluted with a 2-D kernel or filter \( K \) of size \( k_1 \times k_2 \). A convolution of the two entities is given by

\[
s(i,j) = (I * K)(i,j) = \sum_{n=0}^{k_1} \sum_{m=0}^{k_2} I_{n,m} K_{i-n,j-m}.
\]

(5.23)

Note that these discrete convolution operations can be performed with matrix-multiplication. Further, similarly the commutative property of convolution yields \((I * K)(i,j) = (K * I)(i,j)\). Let us now link this convolution to the convolution used in CNNs. The convolution operators implemented in many NN packages uses the cross-convolution operator [22]. The cross convolution is essentially a convolution performed on a non-flipped kernel and hence is given by

\[
s(i,j) = (K * I)(i,j) = \sum_{n=0}^{k_1} \sum_{m=0}^{k_2} I_{i+n,j+m} K_{n,m}.
\]

(5.24)

recall that we have used the commutative property in expression (5.24). When forming the CNN into network structures, then the kernel is essentially considered to be the layer parameters which the network learns, where each layer share the same kernel. Thence, CNN is said to have shared parameters. This sharing of parameters drastically reduces the number of network parameters and are one of the main great characteristics of CNNs. Moreover, regarding the flipping of kernels in the implementations of CNN, the orientation of the kernel has no affect on network operations as long as this orientation is consistent, as the network essentially learns the flipped kernel instead. CNNs are said to consist of sparse interactions meaning that the kernel is often of a less size than the convoluted image input. More specifically, common kernel window used in practice are \( k \times k \) where \( k \in \{2, \ldots, 8\} \) and the appropriate size depending on the input shape.

![Visualization of sparse interactions.](image1)

(a) Visualization of sparse interactions.

![Visualization of kernel sweeping.](image2)

(b) Visualization of kernel sweeping.

Figure 5.9: Convolutional layer with kernel \( K^{4 \times 4} \), input \( 12 \times 12 \) and stride \( S = 2 \), creating a feature map \( s \) of size \( 5 \times 5 \).

A visualization of these sparse interactions is shown in Figure 5.9a using a kernel of size \( 4 \times 4 \) with an input of size \( 12 \times 12 \), and a visualization of the convolution operator with a kernel \( K \) in Figure 5.9b. The propagation in a neuron consists of sparse links which reduce the computations significantly. Hereon the kernel operator will be marked with a box-shape indicator as in Figure 5.9a instead of linking of nodes as in Figure 5.9b.

**Notations**

Let us now convert these convolution operations into terms of the previously presented quantities. The formulation of convolution is based on the cross-convolution in expression
Let \( a_{ljk} = g_l(z_{ljk}) \) denote the output in the \( j,k \)th neuron in layer \( l \), where \( g_l(\cdot) \) is the layer activation function and \( z_{ljk} \) is the intermediate quantity. Further, let the kernel weights for layer \( l \) be denoted \( w^l \) of some size depending on the current layer \( k_1 \times k_2(l) \). Lastly, let \( b^l \) be the shared filter bias for layer \( l \). Each of these quantities is summarized in the subsequent table.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_{ljk} )</td>
<td>output in the ( j,k )th neuron in layer ( l )</td>
</tr>
<tr>
<td>( g_l(\cdot) )</td>
<td>activation for layer ( l )</td>
</tr>
<tr>
<td>( z_{ljk} )</td>
<td>intermediate quantity for layer ( l )</td>
</tr>
<tr>
<td>( b^l )</td>
<td>shared bias in layer ( l )</td>
</tr>
<tr>
<td>( w^l )</td>
<td>shared weights in layer ( l )</td>
</tr>
</tbody>
</table>

Table 5.2: Notations of quantities in the CNN

The intermediate quantity and the output of the \( j,k \)th hidden neuron in the \( l \)th layer is analogously given by

\[
\begin{align*}
  z_{ljk}^l &= \sum_{n=0}^{k_1} \sum_{m=0}^{k_2} w^l_{n,m} a_{l-1,j+n,k+m}^l + b^l \\
  a_{ljk}^l &= g_l(z_{ljk}^l)
\end{align*}
\] (5.25)

with a kernel of weights of size \( k_1 \times k_2 \) in layer \( l \). Further, the \( g_l \) can be any of the activation functions presented in Section 5.1.3 e.g. the ReLU.

Padding and stride

Two central concepts for convolutional layers are padding \( P \) and stride \( S \). The former refers to how the convolution handles the edges and hence it naturally affects the output size of the feature map \( s \). Padding can either be same (zero-padded) or valid (no zero-padded). Zero padding refers to enlarging the input frame with zeros such that the output size is decreased less. If an input is not zero-padded the feature map will naturally result in a smaller size.

![Zero-padded input](image)

Figure 5.10: Zero-padded input to obtain same shape of the output as the input, with \( P = 1, S = 1 \) and \( K^{3 \times 3} \).

The stride \( S \) refers to the step-size in which the kernel sweeps through the input \( I \) and hence the stride also affects the shape of the feature map. In Figure 5.10 an input of size \( 5 \times 5 \) is zero-padded to remain the size, and the convolution is done with \( S = 1 \) and kernel \( K^{3 \times 3} \).
5.2.2 Local receptive field

A receptive field is defined to be the set of ancestor units which some current unit is dependent on, or in other words the units that it is connected to. Further, large receptive fields are said to be superior to small receptive field since the local extracted information which is propagated further to the descendent unit is thence increased. Hence, ultimately having the whole input as a receptive field would entail a lot of information extraction. However, this involves the huge set of parameters in the FCL that we usually want to prevent. Nevertheless, in CNN one can modulate the layers resulting in a fairly large receptive field. This can essentially be done using pooling layers and large kernel sizes.

![Figure 5.11: Visualization of local receptive field in a CNN, where the output neuron (right) has the gray indicated local receptive field of the input (left).](image)

*Local receptive field* hence denotes that a local patch of the input is the receptive field in CNNs. An example of the receptive field for a second layer unit can be seen in terms of the input in Figure 5.11, where the darker units indicate the receptive field. Here no pooling layer has been added in the convolution, and yet the receptive field consists of almost the whole input.

**General CNN architecture**

In general, each layer is equipped with multiple filters allowing a wider spectrum of features to be extracted from the input as each filter $K$ results in a unique feature map $s$. Thence, the convolutional layers are generally enlarged through these filters. A general architecture of a CNN is shown below.

![Figure 5.12: A CNN comprised of $s = \{2, 4, 2\}$ filters in the three layers, $f_l$ here denotes the corresponding activation function at layer $l$.](image)

In this CNN, the number of feature maps is changed throughout the layers to $s = \{2, 4, 2\}$. This wide structure of features in the middle layers is a common architecture of CNN and in some well-known architectures the middle layers consist of 1024 filter maps [48]. Moreover, CNNs are frequently extended with a *skip connection* also called *residual connections* transferring information through the layers. These connections will be presented shortly.
Some remarks- kernels

The size of the kernel affects the features captured in the domain, meaning that is the kernel size is small, the receptive field if small and hence small changes in the image can be extracted. On the other hand, using a larger kernel size, the convolutional layer can capture larger structures in the image. Hence, there is some fine-tuning regarding the kernel size. Furthermore, the more filters a network incorporates the more types of features and salient features can be extracted from the input. As the number of filters increases the extracted features are increased.

A larger filter also results in a larger set of parameters and hence also the computational effort. Let us now consider how these extracted features might look like. In Figure 5.13a 4 convolutions with kernels $K^{3\times3}$ and the familiar image Lena512 is shown, where the network has solely chosen the network kernels. The large image in 5.13a is the input to the convolution, and the 4 sub-figures are the feature maps that the network has produced. Further, a noisy version of Lena is shown in Figure 5.13b. Here the noisy version has been fed as input and the network succeed in extracting more information that could be detected by sight.

Figure 5.13: Example of outputs from a convolutional layer using kernel size $K^{3\times3}$ and four filters represented as the small images, the input image is (a) original Lena image, to the left and the 4 outputs, one for each corresponding filter. (b) Input image is original Lena image with quantum or Poisson noise added.

5.2.3 Pooling layers

Pooling layers are often referred to as downsampling layers that commonly is placed after a convolutional layer. The aim of the pooling layer is to reduce the number of parameters in the network and hence ensures higher computational speeds. The most common pooling layers are the max-pooling layer and the averaging-pooling layer. The former uses a max-operator and extract the max value in the considered window or kernel and the latter uses an averaging operator and hence lower the resolution of the downsampled input or feature map.
The max-pooling operator is often preferred since it only extracts the dominant values. An example of a pooling operator \( P \) performed on a feature map \( s \in \mathbb{R}^{4 \times 4} \) with a kernel \( K^{2 \times 2} \), is shown in Figure 5.14. Further, adding pooling layers in the network entails enlarging the receptive field which is a very nice characteristic. Furthermore, conveniently max-pooling layers are used in deep CNN for extraction of the most prominent features in the downsampling of filters. In fact, some powerful network architectures perform up to 6 downsamplings leading to images of size \( 32 \times 32 \) in the bottleneck layer.

### 5.2.4 Residual connections

Residual NN refers to networks which contains what is referred to as skip or short-cuts connections. As its name suggests, network send intermediate information through short-cut links, as shown in Figure 5.15. The filters are thence sent through a skip connection which is subsequently concatenated with the later layers.

The skip connections have been shown to be successfully used in the networks for reconstruction task "as its connections preserve details in the high resolution features" [54] [38]. The skip connections are vital in DNN since its framework enables the updating of parameters in the first few layers to be done very quickly. It allows gradients that otherwise would have saturated, to be propagated throughout the network to the very first layer. Hence, it reduces the risk of the vanishing of gradients that is often tackled within DNN training. The connections were first introduced and proved their power in the U-net which was used for biomedical segmentation [48, 54]. To prevent massive information leak through these channels (SSC), gate coefficients can be modified during training to force learning at bottleneck layers. [64]
5.2.5 Backpropagation in CNNs

Following the same procedure as in Section 5.1.2, we will now consider the BP extended to this 2-dimensional case and aimed to derive the recursive expression of the gradients. Again, let the error in the \( j, k \)th neuron in layer \( l \) be defined as the derivative of the loss with respect to the intermediate quantity \( z_{jk}^l \)

\[
\delta_{jk}^l \equiv \frac{\partial L}{\partial z_{jk}^l} \tag{5.27}
\]

and recall that (5.27) can be rewritten using the chain rule. The error can thence be expressed according to

\[
\frac{\partial L}{\partial z_{jk}^l} = \sum_{j'} \sum_{k'} \delta_{jk'}^l \frac{\partial}{\partial z_{jk}^l} \left( \sum_a \sum_b w_{a,b}^{l+1} g_l(z_{j'-a,k'-b}) + b_{j,k'}^{l+1} \right) \tag{5.28}
\]

where the first derivative is simply the error in the subsequent layer. Further, the second derivative has been expressed over a sum of \( a \) and \( b \) where \( j = j' - a \) and \( k = k' - b \).

Now, lets evaluate the last derivative in Equation (5.29).

\[
\frac{\partial}{\partial z_{jk}^l} \left( \sum_a \sum_b w_{a,b}^{l+1} g_l(z_{j'-a,k'-b}) + b_{j,k'}^{l+1} \right) = w_{a,b}^{l+1} g_l(z_{j',k}) \tag{5.30}
\]

Using the formulations \( a = j' - j \) and \( b = k' - k \) the previous expression of the derivative can be reformulated into

\[
w_{j'-j,k'-k}^{l+1} g_l(z_{j,k}) \tag{5.31}
\]

Lastly we wish to derive an expression of the gradients w.r.t the kernel weights and biases. Similarly as above this derivative can be rewritten using the chain rule

\[
\frac{\partial \mathcal{L}}{\partial w_{a,b}^l} = \sum_j \sum_k \frac{\partial \mathcal{L}}{\partial z_{jk}^l} \frac{\partial z_{jk}^l}{\partial w_{a,b}^l} \tag{5.32}
\]

\[
= \sum_j \sum_k \delta_{jk}^l \frac{\partial}{\partial w_{a,b}^l} \left( \sum_{a'} \sum_{b'} w_{a',b'}^l g_l(z_{j'-a',k'-b'}) + b_{j,k}^l \right) \tag{5.33}
\]

\[
= \sum_j \sum_k \delta_{jk}^l g_l(z_{j-a,k-b}) \tag{5.34}
\]

\[
\frac{\partial \mathcal{L}}{\partial b_{j,k}^l} = \sum_j \sum_k \delta_{jk}^{l+1} \tag{5.35}
\]

and the derived expressions of the gradients (5.34) and (5.35) can be used in Algorithm 2, backpropagation of gradients, which slightly need to be modified. This modification comprises of reformulating the forward pass into terms of the CNN notations.
5.3 Methods to speed up training of a FNN

In many deep learning networks, the most excessive part of the setup is the training stage i.e. the updates and computation of the gradient. The computational time is constrained by the computer performance but is also dependent on the architecture of the network. In this section, we will present some of the aspects of this matter and thence present some tricks for reducing the training time.

Batch normalization

The batch normalization has been an essential tool in the optimization of deep NN and "it is a method of adaptive reparameterization, motivated by the difficulty of training very deep models" [22]. More specifically, the previously presented gradient derivation of each parameter is based on the assumption that all other layers do not change when the considered layer changes, which is not true in general. This assumption entails problems of second and higher order effects making it very hard to find a proper learning rate and complicates the training of deep FNN. "Second-order optimization algorithms address this issue by computing an update that takes these second-order interactions into account, but we can see that in very deep networks, even higher-order interactions can be significant" [22]. However, these second-order optimization algorithm is very computationally intensive and therefore other methods are looked for. One such method is batch normalization, which can be used to reduce these problems of parameter updates.

![Visualization of batch normalization](image.png)

(a) Learning when the data is not normalized, for the simplified case of only two network parameters.

(b) Learning when the data is normalized, for the simplified case of only two learning parameters.

Figure 5.16: Visualization of batch normalization, for a simplified version where there are only two network parameters to be optimized.
Let us now explain this normalization in more detail. Let the matrix $H$ contain all the activations $l \in \{1, \ldots, L\}$ for each example in the row $l$. The batch normalization, as the name reveals, normalizes $H \rightarrow H'$ in the similar fashion as general standardization. The normalized batch is formulated

$$H' = \frac{H - \mu}{\sigma} \quad (5.36)$$

where $\mu$ is a vector containing the mean of each corresponding unit and $\sigma$ their corresponding standard deviation.

$$\mu = \frac{1}{m} \sum_i H_i \quad (5.37)$$

$$\sigma = \sqrt{\delta + \frac{1}{m} \sum_i (H - \mu)^2} \quad (5.38)$$

and $\delta$ is a small number, e.g. $10^{-8}$ [22] and is introduced to avoid the situation of the undefined gradient at $z = 0$ for $\sqrt{z}$. Consider the normalization in Equation (5.36), here simply each row of $H$ is normalized with the corresponding mean and standard deviation in the training phase. In the test phase, $\mu$ and $\sigma$ "may be replaced by running averages that were collected during training time" [22].

The use of batch normalization is also often motivated with faster convergence of the network, see Figure 5.16a. However, it also inherits the property as a regularizer to the network and hence can as well be used to decrease the overfitting. As for each epoch, the weights are normalized, and if the network is caught in a bad local minimum it has the ability to take itself out. Similarly, if the network parameters are overfitting for the training set, the batch normalization pushes the parameters from these states softly.

**Transfer learning**

Transfer learning refers to the strategy of training a network on some shallower or simpler architecture and thence use this pre-trained optimized network parameters as starting points or initialization in a more complex architecture. The method is often useful for network architectures of high dimensional data, complex structures, stacked structures or ultra-deep networks. The main idea of transfer learning is to reduce the training time by initializing the parameters at some good initial point for the training.

One example of this is if a 2D network would be expanded to a 3D network of depth 3, i.e. adjacent images are inputted to the network. Then one can initialize the parameters or weights of the middle layer using the pre-trained 2D network. This tool is referred to as transferred learning and is very useful for networks which generally are slow to train, e.g. networks with volumetric inputs [54].
5.4 Denoising using Neural Networks

Denoising in NN refers to the concept of removing some added or multiplicative noise present in the input. More specifically, the noise is often assumed to be generated through some stochastic process called corruption process $C(\tilde{x}|x)$ where $x$ is the clean data and $\tilde{x} \sim C(\tilde{x}|x)$ is the corrupted noisy data. For example, if we are studying an additive Gaussian noise, the corrupted data is formulated $\tilde{x} \sim \mathcal{N}(\tilde{x}; \mu = x, \sigma)$. The aim of a denoising network is thence to find the function $f_\theta$, mapping the corrupted samples to the corresponding clean data, $f_\theta : \tilde{x} \mapsto x$. Thence, the loss function is reformulated into a slightly different formulation based on clean and noisy samples $\{\tilde{x}_k, x_k\}_{k=0}^K$

$$\min_\theta \sum_k \mathcal{L}(f_\theta(\tilde{x}_k), x_k)(\theta) \quad (5.39)$$

where $f_\theta$ is a parametric family of mappings under the loss function $\mathcal{L}$. Furthermore, note that the notation $\tilde{x}$ is used to underline the fact that the corrupted input $\tilde{x}_i \sim C(\tilde{x}|x_i)$ is a random variable dependent on the true target. These denoising networks can thence simply be constructed by replacing the general loss function into terms of the samples of $\{\tilde{x}, x\}$. Additionally, in a very recent paper named Noise to Noise it was shown that a network trained on mapping noise to noise, could in fact denoise equally good as networks that had been trained on clean samples [32]. This is an advantageous result since it proposed that we actually in some settings don’t need the clean data, which often is not accessible.

5.4.1 State of the art and denoising networks

Here we will give a very short overview of classical and modern denoisers. Classical denoisers involve the familiar block-matching and 3D-filtering BM3D [15], and the K-SVD [2]. Further, denoisers in NN often involves autoencoder (AE) which will be presented in next chapter. These powerful denoisers are for example generative adversarial networks, namely denoising adversarial autoencoder (DAAE) [14] which "combines denoising and regularization, shaping the distribution of latent space using adversarial training" [13]. In [11] they propose recurrent denoising autoencoder (AE) for iterative reconstruction of Monte Carlo Image sequences (temporal denoising), which elegantly denoises the Monte Carlo renderings by propagating information through states. In [53] they propose a DAE learned on local patches of the considered data, for surface completion and inpainting of 3-D data. Further, in the LLNet they "propose [...] a deep-autoencoder [...] to identify signal features from low-light images and adaptive brighten images without over-amplifying/saturating the lighter parts in the images [34]."

The above previous methods suggest that an autoencoder (AE) or more specifically a DAE structure is a suitable approach for denoising. Thence, this is what will be discussed in the next chapter.
Chapter 6

Autoencoders

Autoencoders, abbreviated AE, is a particular type of FNN which initially was intended to be used for dimension reduction and feature learning but has also been shown to be "successfully applied to [...]\footnote{ [...] means that the quote as been shortened} information retrieval tasks", [which is]\footnote{[word] means that the quote as been altered with word} the task of finding entries in a database that resemble a query entry" [22]. Further, a specific type of AE namely the denoising autoencoder (DAE) was motivated to use the dimension reduction techniques that the AE incorporated, to filter out noise through the forced compression of information. This DAE will be the main focus in this chapter and will be studied further shortly. First, this chapter will consider the main structure of the general AE and its objective. Also, we will consider the aspiration of the regularized autoencoder (RAE).

The objective of the AE is relatively different from the previously reviewed supervised neural networks. In fact, instead of training the network to learn the mapping of some input $\mathbf{x}$ to some output $\mathbf{y}$, the AE is trained to \textit{copy} its input to its output. This mapping, however, needs to be constrained somewhat, since training the network to map the identity is not particularly useful. Therefore the AE is designed to be unable to perform the identity mapping and instead is forced into learning the most prominent features to perform the coping or reconstruction. Or in other words, in this manner, the AE learns the hidden properties of the considered data. A visualization of a DAG of an autoencoder is shown in Figure 6.1.

![Directed acyclic graph (DAG) of the AE](image)

Figure 6.1: Directed acyclic graph (DAG) of the AE, with input $\mathbf{x}$, encoder function $f$, decoder function $g$ and reconstruction $r$ [22].
The autoencoder consists of two main components, an encoder, and a decoder. The encoder part aims to map the input $x$ into what is called the code or the latent representation $h$, using an encoder function $f$, i.e. $f : x \mapsto h$. Further, the decoder part target to map this intermediate latent representation $h$ to what is called the output or reconstruction $r$, using the decoder function $g : h \mapsto g(h) = r$. The objective of the autoencoder is then to learn the mapping $g(f(x)) = r \mapsto x$, and is trained to minimize the reconstruction loss which is given by the entity

$$\mathcal{L}(g(f(x)), x)$$

which commonly is chosen to be the average squared loss $\ell_2$ [58]. The structure and restrictions of the intermediate representation $h$ directly affect the performance of the autoencoder and are hence well studied. As mentioned, if the dimension of the latent representation is the same as the input, the autoencoder simply learns the identity mapping yielding a very ineffective autoencoder. In contrary, if the dimension of the latent representation is smaller than the input dimension, the autoencoder is referred to as undercomplete, and the autoencoder can capture the most dominant features of the input to perform the reconstruction. Several authors [20] [22], connects the autoencoder with a single linear layer structure trained on the $\ell_2$ loss, to the principal components of $x$, meaning that "PCA can be interpreted as a linear autoencoder with mean-squared error loss function where the eigenvectors represent the tied encoder and decoder weights" [20].

### 6.1 Regularized autoencoders

Undercomplete autoencoders, compressing the input into a smaller dimension ($\dim(x) > \dim(h)$) can learn salient features of the data distribution better than overcomplete AE. Furthermore, if the encoder and decoder are given too much capacity, the AE can fail to capture the essential features in the data distribution [22]. Thence, one can penalize the AE for high complexity in the latent representation $h$ by casting it into the regularized autoencoder (RAE) formulation, i.e., aiming to enforce the AE only to capture the most important features.

Similar problems occur for overcomplete autoencoders. For overcomplete AE the dimension of the latent representation is the same or higher than the input dimension. Without any constraints on $h$ the training promptly leads to the case where the AE that solely learns the identity. Hence results in unsuccessful learning of the AE. These obstacles can thus also be dealt with using RAE, which uses an additional term in the loss function to penalize for undesirable mappings. For example, penalizing for the complexity of the model or other regularizing aspects. One such regularized autoencoder (RAE) is the sparse AE that uses an added sparsity term $\Omega(h)$ in the loss function to penalize for high complexity in the latent representation $h$. 

52
6.1.1 Sparse Autoencoders

A sparse autoencoder includes a sparsity penalty term $\Omega(h)$ penalizing for high complexity in the latent representation $h$, in the loss function

$$L(g(f(x)), x) + \Omega(h).$$

Furthermore, the regularizer in the RAE can also penalize the derivatives of $h$ w.r.t. the input $x$, and then the regularizer is also dependent on the input $\Omega(x, h)$

$$\Omega(x, h) = \lambda \sum_i ||\nabla_x h_i||^2,$$

where the AE is now penalized for large gradients w.r.t. $x$ and $\lambda$ is the regularizer hyperparameter. Indeed, this penalizing "forces the model to learn a function that does not change much when $x$ changes slightly" [22]. This RAE, using the penalizing term (6.3), is referred to as the contradictive autoencoder (CAE) and has been shown by G. Alain et al. to have a strong connection to the DAE [3].

6.2 Denoising Autoencoder

A denoising autoencoder (DAE) is of a form of AE which is trained to map noisy samples $\tilde{x}$ to the clean ones $x$ using an AE framework, hence denoising autoencoder. Nevertheless, instead of mapping the input to a reconstruction of itself, which is the objective of the AE, the DAE is trained to map a corrupted input $\tilde{x}$ to its corresponding noise-free sample through the autoencoder $g(f(\tilde{x})) := r(\tilde{x}) \to x$. The "denoising autoencoders must therefore undo this corruption rather than simply copying their input" [22]. A DAG of the DAE is shown in the subsequent figure.

Figure 6.2: DAG of the DAE where $x$ is the input, $h$ latent representation, $r$ reconstruction and $C(\tilde{x}|x)$ the corruption process. The DAE is evaluated according to some loss function of removing the corruption $L(g(f(\tilde{x})), x)$. 
The underlying setup of a DAE is as follows, the AE receives corrupted data points as input, affected by some corruption process $C(\tilde{x}|x)$. While training on some loss function $L$ and training set $\mathcal{T} = \{\tilde{x}_k, x_k\}_{k=0}^K$, the autoencoder learns what is called a reconstruction distribution $p_{\text{rec}}(x|\tilde{x}) = p_{\text{decoder}}(x|h)$, i.e., learns the mapping between a corrupted sample to a noise-free sample $f: \tilde{x} \mapsto h$ and subsequently $g: h \mapsto r$. The DAE can in simple terms be thought of as a stochastic version of the AE which is trained to minimizing the reconstruction loss

$$L(g(f(\tilde{x})), x))$$  \hspace{1cm} (6.4)

which can be accomplished by e.g. minimizing the negative log-likelihood of the reconstruction distribution. The decoder distribution is commonly a factorial distribution, "whose mean parameters are emitted by a feedforward network $g" [22]. Additionally, a very nice property follows if the encoder function is a deterministic function. Then the network can be trained by straightforwardly using the methods that have been proposed in the NN chapter, and thence perform stochastic gradient descent on the entity

$$-E_{x \sim p_{\text{data}}(x)}E_{\tilde{x} \sim C(\tilde{x}|x)} \left[ \log p_{\text{decoder}}(x|h = f(\tilde{x})) \right].$$  \hspace{1cm} (6.5)

The DAE can further be extended to be a generative model, and thence be expressed as a Markov Chain where each step [...] is associated with a trained denoising autoencoder, which generates the samples from the probabilistic model implicitly trained by the denoising log-likelihood criterion. [22]. However, the convergence rate of these generative models are generally slow and the subsequent walk-back algorithm [7] was proposed to accelerate the training. The walk-back procedure is based on the framework that instead of single evaluation of the reconstruction, a chain of samples would be evaluated wherein each state of the chain, new realizations of the corruption would be drawn to recover $p_{\text{data}}(x)$.

### 6.2.1 Score matching

Score matching is a general tool for comparing two distributions and can be used to perform learning of the DAE. Hence, it is an alternative to the previously proposed negative-log likelihood. Mathematically, the score is defined as the gradient of the log data distribution w.r.t the input $x$, i.e.

$$\nabla_x \log p(x).$$  \hspace{1cm} (6.6)

Comparing the score of two distributions is mathematically related to examining the derivatives of the distributions, and training a model on the score essentially encourage the model to have the same score as the data distribution at each sampled data. Further, "a very important property of the DAE is that their training criterion (with conditional Gaussian $p(x|h)$) make the AE to learn a vector field $(g(f(x)) - x)$ that estimates the score of the data distribution with a fixed noise level. Regularized score matching is not a consistent estimator, it instead recovers a blurred version of the data distribution. If the noise level is chosen to approach 0 when the number of examples approach infinity, however, the consistency is recovered" [22].
6.2.2 DAE for Gaussian corruption

Let us now assume that the corruption process is an isotropic Gaussian with mean \( \mu \) and covariance \( \Sigma_0 \) or more specifically \( \mathcal{C}(\tilde{x}|x) = \mathcal{N}(\tilde{x}|x, \sigma^2(x)I) \), which can be interpreted as \( x \) is pushed away by some random factor to \( \tilde{x} \). Further, in this particular considered case, let the AE be trained on minimizing the squared expected loss

\[
L_{DAE,t_2} = \mathbb{E}_{x,\tilde{x}}[|g_\theta(f_\theta(\tilde{x}))) - x|^2] = \mathbb{E}_{x,\tilde{x}}[|r(\tilde{x}) - x|^2]
\]

(6.7)

for some set of encoder parameters \( \theta \) and decoder parameters \( \theta' \). Additionally, recall the defined reconstruction function, \( g(f(\cdot)) = r(\cdot) \). Now we will consider how this circumstance relates to denoising score matching.

Relation between score matching and DAE

Score matching, as discussed in Section 6.2.1, is performed by comparing scores between distributions, where the score is given by the entity (6.6). In this section we will aim to compare scores of the smoothed distribution \( p_{\text{smoothed}}(\tilde{x}) = \int p_{\text{data}}(x)p_{\mathcal{C}(\tilde{x}|x)}(\tilde{x}|x)dx \) [22], rather than the data generating distribution itself \( p_{\text{data}}(x) \). The motivation of comparing these scores is based on the idea that if we follow the gradient of the estimated score \( \psi \) of the log density then at some point along the way we will recover the clean sample \( x \) [57]. Further, minimizing the expected difference between the score of some estimated associative score function \( \psi(\tilde{x}; \theta) \) and the score of \( p_{\mathcal{C}(\tilde{x}|x)}(\tilde{x}|x) \), is achieved by minimizing the entity

\[
L_{DAE,SM} = \mathbb{E}_{x,\tilde{x}}\left[|\psi(\tilde{x}; \theta) - \frac{\partial \log p_{\mathcal{C}(\tilde{x}|x)}(\tilde{x}|x)}{\partial \tilde{x}}|^2\right]
\]

(6.8)

and "is equivalent to performing score matching [...] with estimator \( \psi(\tilde{x}) \) and target density \( p_{\text{smoothed}}(\tilde{x})" [3] [26]. Let us now consider the very nice result derived by Alain et al. [4] together with Hyvärinen. First, let us define an associative score function to be of the form \( \psi(\tilde{x}; \theta) = -\partial \mathcal{E}(\tilde{x}; \theta)/\partial \tilde{x} \), where the energy function \( \mathcal{E}(\tilde{x}) \) can be parametrized by [26]

\[
\mathcal{E}(\tilde{x}; \theta) = -\log p(\tilde{x}; \mu, \Sigma) = \frac{1}{Z(\mu, \Sigma)}\exp(-\frac{1}{2}(\tilde{x} - \mu)^T \Sigma^{-1}(\tilde{x} - \mu))
\]

(6.9)

yielding to the parametrized score \( \psi(\tilde{x}; \mu, \Sigma) = -\Sigma(\tilde{x} - \mu) \) and if \( \Sigma = \sigma^2 I \) \( \mu = r(\tilde{x}) \) then \( \psi(\tilde{x}; \theta) = (r(\tilde{x}) - x)/\sigma^2 \). In fact, we define the reconstruction function \( r(\tilde{x}) = \tilde{x} + \sigma^2 \psi(\tilde{x}; \theta) \). Further, following from the assumption of isotropic Gaussian noise the second score is given by

\[
\frac{\partial \log \mathcal{N}(\tilde{x}|x, \sigma^2(x)I)}{\partial \tilde{x}} = \frac{x - \tilde{x}}{\sigma^2}
\]

(6.11)

Then the expected score matching loss \( L_{SM} \) takes the form

\[
\mathbb{E}_{x,\tilde{x}}\left[\frac{|r(\tilde{x}) - x|^2}{\sigma^2} - \frac{x - \tilde{x}}{\sigma^2}||^2\right] = \frac{1}{\sigma^2}\mathbb{E}_{x,\tilde{x}}||r(\tilde{x}) - x||^2 \propto L_{DAE,t_2}
\]

(6.12)

\[^{3}[...]\] mean the the quote has been shortened
and hence $\mathcal{L}_{DAE,SM} \propto \mathcal{L}_{DAE,\ell_2}$. This result basically tells us is that when the reconstruction function is parametrized do as to correspond to the score of a model density, [...] the denoising criterion on $r$ with Gaussian corruption noise is equivalent to score matching with respect to a smooth [version] of the data generating density, i.e. a regularized form of score matching” [57]. Further, [4] showed that these results hold even when $r$ is not parametrized as $r(x) = x + \sigma^2 \psi(x)$.

6.2.3 Manifold interpretation of DAE

Assume that the high-dimensional data $x$ concentrated near a non-linear low-dimensional manifold. Based on this assumption we can formulate an intuitive geometric interpretation of the mapping function $g(f(\cdot))$ [59]. Further, assume that due to the corruption process the corrupted samples $\tilde{x}$ are pushed from this latent manifold. Thence, the DAE "learns a map that tends to go from lower probability points $\tilde{x}$ to nearby higher probability points $x$, on or near the manifold" [59]. The AE projects the corrupted data samples back to the manifold [60], as the DAE is forced to learn only the robust features rather than only the identity. A visualization of this projection is presented in Figure 6.3 and "the denoising AE can thus be seen as a way to define and learn a manifold” [59].

![Figure 6.3: Visualization of the manifold interpretation of the DAE. The prior assumption is that the data $x$ concentrates near a low dimensional manifold (blue). The corruption process shifts the data away from this manifold (red), and thence, the DAE reconstruction is mapping the corrupted samples back to the manifold. Image inspiration [3].](image)

When the DAE is trained on the squared reconstruction error, $\ell_2$, then the reconstruction estimates $\mathbb{E}_{x, \tilde{x} \sim p_{data}, \mathcal{C}(\tilde{x}|x)}[x|\tilde{x}]$ and thence the vector field $(g(f(\tilde{x})) - \tilde{x})$ pointing towards the mean point on the manifold, see Figure 6.3.

---

4 [...] means that the quote has been shortened
5 [word] means the the quote has been altered with word
Chapter 7

Data

The quantitative data used in this analysis is based on data generated using PENELOPE, which is a Monte Carlo "code system for the simulation of coupled electron-photon transport in arbitrary materials" [51]. The PENELOPE simulation uses the Monte Carlo sampling techniques described in Chapter 3, to estimate the radiation transportation equation for a given geometry and radiation setup, yielding a dose distribution for the considered radiosurgery treatment plan. The phase space, containing information of the sampled particles initial state, used in the simulations is based on the phase spaces of collimator sectors of the Gamma Knife. These phase spaces are essentially used to decrease the computation time of the simulation. Recall the RS tool Gamma Knife for gamma radiation presented in Section 2.2. The treatment plans used in this experiment are in-house data from Elekta and are real patient data plans constructed in Gamma Plan.

In this chapter, the overall structure and generation of data will be discussed. Since there is a shortage of data, the dose distribution data used in this thesis will be constructed with the aims of the data augmentation described in Section 7.2.1. The shortage of data is due to limited resources of cluster-computations along with the vast computation time for large photon simulations. The data augmentation is based on a set of dose kernels, which are generated with the previously presented code system and are subsequently assembled to a dose distribution of an RS treatment plan. We stress that this is mainly done since there is a shortage of data. This data augmentation will be discussed in more detail shortly. Furthermore, in this chapter we will define the reference of the clean dose and noisy doses in terms of the number of photon showers.

7.1 Treatment plans

The considered RS treatment plans are given in digital imaging and communication in medicine-format, abbreviated DICOM where each treatment plan consists of information regarding the volumetric geometry of the patients head and tumor along with a plan of the scheduled treatment. The RS treatment is often multiple-shooting, i.e., multiple shots are delivered to the VOI to achieve the planned dose. These treatment plans have been created prior through an optimization algorithm in Gamma Plan, and are based on the computed tomography (CT) and magnetic resonance imaging (MRI) images for the considered patients. Each shot contains information regarding the isocenter position jointly with sector information and shot time for the specific setup. Each of the previously mentioned quantities is assembled into one plan. In this experiment, 40 stereotactic RS treatment plans are used as the training set, containing treatment plans of tumors.
types such as meningeom and metastasis. The test set contains five plans for evaluation. Further, we have five independent treatment plans for the validation set.

### 7.2 Dose distributions

The RS treatment plans are realized using a set of prior simulated dose kernels describing the dose distribution for the different sector configurations at a given number of photon showers \( N \). The generation of these dose distributions from a set of plans are described further in the subsequent Section 7.2.1. The noisy dose distributions that the network will train to denoise are based on the simulation parameter \( N_{\text{noisy}} \), and the corresponding clean target dose distribution are generated with \( N_{\text{clean}} \). Further, the prior generated dose kernels are simulated for a simplified geometry, with the rough approximation of a human head as a sphere containing water. This approximation is motivated by the high percentage of water contained in the human head, e.g. in [43] they state that the human brain contains up to 60% water. The 3-dimensional internal volume of the considered geometry is further discretized into voxels, converting the continuous space into a discrete digital topology and enables approximate dose calculations on the considered geometry.

![Figure 7.1: Slice of a realized treatment plan for different number for photon showers \( N \), generated with the prior simulated dose kernels.](image-url)
Through each realized treatment plan in this custom generator, a pair of dose distributions is supplied, a clean dose distribution and a dose distribution containing some level of noise. See how the level of noise is dependent on the simulation parameter $N$, Figure 7.1. At the highest resolution, the dose distributions cover a $6 \times 4 \times 3$ volume with voxel resolution of $0.25 \text{ mm}$, yielding a dose distribution $D_{\text{plan}} \in \mathbb{R}^{256 \times 256 \times 256}$. Furthermore, to analyze the propagation time as well as the denoising performance for lower resolutions, the generated dose plans are downsampled using a standard median filter with kernel size $K^{2 \times 2 \times 2}$, obtaining resolutions $0.5 \text{ mm}$ and $1 \text{ mm}$ with the corresponding downsampled dose volumes $D_{\text{plan}}^{128 \times 128 \times 128}$ and $D_{\text{plan}}^{64 \times 64 \times 64}$.

### 7.2.1 Generation of dose distributions

For each RS treatment plan, a realized dose distribution $D_{\text{plan}}$ of fixed size $256 \times 256 \times 256$ and a resolution of $0.25 \text{ mm}$ is constructed with the prior simulated dose kernels. The simulated dose kernels are based on phase spaces $\mathcal{W}_{ij}$ for the sector sizes and configurations $i = 4, 8, 16$ mm and $j = 1 \ldots, 8$ of the Gamma Knife. These are generated through a center shot into a dose box yielding what here is referred to as dose kernels $D_{K_i}$, $i = 4, 8, 16$. Further, note that each sector configuration consists of 8 assembled dose kernels, one for each sector setting. Thence in this manner, the dose kernels can be translated to custom the shots of the treatment plan. Therefore we want to stress that these dose kernels are the basis for the generated samples. Let us now formalize this data augmentation of the dose distributions. A treatment plan $P = \{C_i\}_{i=1}^M$ consists of $M$ sets of single-shooting parameters $C_i = \{s_{i,\text{info}}, t_{i,\text{info}}, p_i(\bar{x})\}$, where $M$ is the number of shots in the scheduled plan. Further let $p_i(\bar{x})$ be the shot position of shot $i$, $t_{i,\text{info}}$ the shot duration and $s_{i,\text{info}}$ a vector of size $1 \times 8$ containing the sector configuration of shot $i$. The elements in $s_{i,\text{info}}$ indicate which collimator size, 4, 8 or 16 mm, that the sectors have in the setup. Let us summarize these entities in the following table

<table>
<thead>
<tr>
<th>$D_{\text{plan}}$</th>
<th>dose distribution generated from a treatment plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_K$</td>
<td>dose kernels $D_K = {D_{K4}, D_{K8}, D_{K16}}$</td>
</tr>
<tr>
<td>$C_i$</td>
<td>single-shot information for shot $i$, $C_i = {s_{i,\text{info}}, t_{i,\text{info}}, p_i(\bar{x})}$</td>
</tr>
<tr>
<td>$s_{i,\text{info}}$</td>
<td>sector configuration for shot $i$, vector of size $1 \times 8$</td>
</tr>
<tr>
<td>$t_{i,\text{info}}$</td>
<td>shot duration for shot $i$, scalar</td>
</tr>
<tr>
<td>$p_i(\bar{x})$</td>
<td>shot position for shot $i$, vector of size $1 \times 3$</td>
</tr>
</tbody>
</table>

Table 7.1: Data augmentation and treatment plan variables

Further, as has been stressed, a treatment plan naturally consists of multiple shots. The dose distribution of these multi-shooting plans is constructed through adding each of the single-shot realization $D(C_i|D_K)$, using the prior simulated dose kernels $D_K$. A schematic view of the construction of a dose distribution for a RS treatment plan $P_i = \{C_i\}_{i=1}^M$ presented in Figure 7.2. Some examples of generated dose distributions
using this augmentation are presented in Figures 7.1a-7.1f and in Figures 3.3a-3.3f, for
the different number of photon simulations. This construction of the dose distributions,
for a given treatment plan, is unbiased due to the definition of the MC estimate of the
deposited energy, see Equation (3.10).

Additionally, the test set and training set are both realized using the same set of gener-
atated dose kernels. However, the validation data is generated through independent dose
kernels. This is done to investigate the network performance for total completely new
random realizations of the noise.

7.2.2 Simulation parameters

The dose kernels are simulated using a Gamma Knife phase space $W$ containing initial
simulation information of each collimator sector phase space, i.e., containing information
of the initial states of the photons. The MC photon/electron simulation for the dose
kernels have been realized with absorption energy threshold $E_{\text{threshold}} = 5 \times 10^{-3}$ and
the variance reduction technique interaction forcing with duplication factor 10. Further,
the dose kernels have been constructed with fixed voxel resolution of $0.25 \text{ mm}$ and for
the number of photon showers $N \in \{10^4, \ldots, 10^9\}$. The geometry considered in the
simulation is a homogeneous sphere of water with a radius of $5 \text{ cm}$, approximating a
human head. The target is assumed to be located approximately in the center of the
sphere, and thence the shots are assumed to be targeting the VOI. Lastly, the dose kernels
for the training and test set have been simulated using a different RNG seed than the
dose kernels for the validation set.
### 7.2.3 Noise levels in dose kernels

In this section, we will consider the adherent statistical uncertainties or noise present in the dose distribution estimates, or more specifically in the dose kernels. Further we will define at which threshold, concerning the number of primary particle simulations the dose distributions can be considered noise free and thence will be regarded as the true target dose distribution. Let us consider the simulated dose kernels described in Section 7.2.1. From each simulated dose kernel we can formulate a corresponding 99, 7% confidence bound at each voxel, $I_{D_{x,y,z}}$ as stated in Chapter 3. Further, let us consider the dose profile for the kernel of sector configuration 16 mm, i.e. $D_{K16}$ which is used to construct the training and test sets. The dose profiles we will consider are located at a center slice of the dose volume kernels and for a variable number of photon showers $N$.

![Graphs showing dose profiles for different numbers of photon simulations.](image)

(a) $N \in \{10^4, 10^5, 10^6\}$. (b) $N \in \{10^7, 10^8, 10^9\}$. (c) $N \in \{10^6, 10^7, 10^9\}$.

Figure 7.3: Dose profile (solid lines) for the kernel $D_{K16}$, at a different number of primary photon simulations $N$. The dose profiles are extracted from the $x$-axis and are plotted with their corresponding 99, 7%-confidence bound (dashed lines).

In Figure 7.3b one can observe the apparent convergence to a consistent dose distribution for a vast number of primary photon simulations, $N = 10^8, 10^9$. Furthermore, at very low $N$ the true dose distribution is significantly harder to detect, see e.g. $N = 10^4, 10^5$ in Figure 7.3a. In fact, at these low simulation quantities, the majority of voxels has not yet received a single sample, and the noise is large. The noise level for $N = 10^6 \ldots, 10^7$ remains large, however, at these realizations the estimate is somewhat converging to the mean, see Figure 7.3c. The dose profiles for the remaining dose kernels have the same type of characteristics and thence will not be presented.

From these profiles, we can now define a *noise-free* or *clean* dose kernels $D_K$ at $N_{clean} = 10^9$. Further, the *noisy* kernels are defined at $N_{noisy} \in \{10^4, \ldots, 10^8\}$ primary particles. Thence, these dose kernels are used in the realization of noisy dose distributions $D_{plan}(N_{noisy})$ and the noise-free dose distributions $D_{plan}(N_{clean})$, in this supervised setting.
7.2.4 Error in dose distributions

In this section we will look at error levels in the test and validation set, using the metrics presented in Section 7.2.1. The noise will be expressed in terms of the metrics \( \beta \), \( 2D - \gamma \), \( \Delta_T \) and \( \Delta_R \). We will consider five dose volumes each in the test and validation set, where the validation set is generated through a set of independent dose kernels from the ones used in the construction of the test set, i.e. new random realizations of the dose kernels.

The metrics for the data sets are presented in Table 7.2.

<table>
<thead>
<tr>
<th>( \mathcal{P}_i )</th>
<th>( \mathcal{P}_i^S )</th>
<th>( \mathcal{P}_i^N )</th>
<th>( \mathcal{P}_i^N )</th>
<th>( \mathcal{P}_i^V )</th>
<th>( \mathcal{P}_i^V )</th>
<th>( \mathcal{P}_i^V )</th>
<th>( \mathcal{P}_i^V )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2D - \gamma ) (%)</td>
<td>65</td>
<td>77</td>
<td>78</td>
<td>61</td>
<td>91</td>
<td>86</td>
<td>47</td>
</tr>
<tr>
<td>( \Delta_T ) [eV/g]</td>
<td>187</td>
<td>363</td>
<td>329</td>
<td>206</td>
<td>199</td>
<td>357</td>
<td>346</td>
</tr>
<tr>
<td>( \Delta_R )</td>
<td>0.16</td>
<td>0.11</td>
<td>0.10</td>
<td>0.13</td>
<td>0.09</td>
<td>0.08</td>
<td>0.19</td>
</tr>
<tr>
<td>( \beta ) (%)</td>
<td>38</td>
<td>56</td>
<td>54</td>
<td>34</td>
<td>51</td>
<td>60</td>
<td>37</td>
</tr>
</tbody>
</table>

Table 7.2: Statistics for test data \( \mathcal{P}^S \) and validation data \( \mathcal{P}^V \) for 5 dose volumes each \( i \in \{1, \ldots, 5\} \), where the true target distribution is defined at \( N_{\text{clean}} = 10^9 \) and the evaluated noisy samples here are based on \( N_{\text{noisy}} = 10^6 \).

The \( 2D - \gamma \) index has been evaluated at a center slice of the volume. Further, in the subsequent plot, the mean of the statistics is plotted for the different number of photon showers.

In Figure 7.4, one can see how the error decreases as the number of samples are increased. Recall that large values of \( \beta \) and \( \gamma \) indicate low differences between the evaluated and reference dose distributions, see Section 7.2.1. Furthermore, the statistics for the test set and validation set have similar error levels, see Figures 7.4a-7.4b.

---

\(^{1}\)Training set will not be considered, but have similar numbers
Chapter 8

Implementation

In this chapter, we will use the previously presented background and form it into the motivation of the two main components of the implementation, the network structure and the data. The latter has been presented in detail in Chapter 7 and thence, will only be discussed very briefly. The overall aim of the implementation is to investigate if the constructed network is able to map the noisy dose distributions, generated at the noise level $N_{\text{noisy}}$, to noise-free dose distributions generated with the simulation parameter $N_{\text{clean}}$. This will be investigated for different input structures, resolutions and prior information. Lastly, before digging into the details, we want to state that the network is trained to denoise slices or images of the dose distribution rather than volumes of the dose distribution. This is mainly done due to the shortage of data. Hence, the volumes are split into 256 dose slices which are denoised by the network and which subsequently are assembled into the original shape, although in a denoised version. These denoised versions are afterward evaluated to the clean reference dose volume.

The structure of this chapter is the following. Section 8.1 will present the network architecture along with the considered loss function in this thesis. Additionally, the section will consider the input structures used in the implementation, and underline the implementation considerations. Section 8.2 will give a brief overview of what was presented in Chapter 7, and hence state the data used in the implementation. Lastly, Section 8.3 will describe the setups and packages used in the implementation.
8.1 Denoising AE structure

The network used for this denoising task has the familiar U-net [48] architecture, containing convolutional layers with residual connections, max-pooling layers for downsampling, intermediate batch normalization and the general AE structure of encoder and decoder parts which are commonly used in modern deep NN architectures. The purpose of these components is outlined in Chapter 5 and 6. More specifically, the implemented network consists of an encoder part transforming the input images of size $256 \times 256 \times C$ to $32 \times 32 \times s$ where $s = 256$ is the number of feature maps in the bottleneck layer and $C = 1, 3$ is the number of *channels* in the input image. Furthermore, the network consists of a decoder part which step-wise up-sample the images to the reference format $256 \times 256 \times 1$, see Figure 8.1. An overview of the constructed network is presented in the subsequent figure, note that the network has the familiar U-shape.

Let us now consider the blocks in more detail. Each level on the encoder side in Figure 8.1 consist of a stack of convolutional layers followed by a skip connection and a max-pooling operator. The skip connections are copying the feature maps to the corresponding levels on the decoder side, while the feature maps sent to the max-pooling layers are downsampled to a quarter of their image size, i.e. reaching the level below in the abstraction shown in Figure 8.1. From this level on, the procedure is repeated til the feature maps are of size $32 \times 32 \times 256$, and thence outputs the latent representation $z$. Each convolutional block consist of one convolutional layer doubling the number of feature maps and another outputting the same number of feature maps as it was inputted. Additionally, the parameters of the very first convolutional layer are presented in the left box of Figure 8.2, and the remaining convolution parameters in the right box.

Let us now consider the decoder part. The output of the encoder part is sent as an input to the decoder. Similarly, each level in the decoder part consist on a block of convolutional layers followed by an upsampling and a concatenation of the copied feature maps sent over the skip connections. Additionally, the convolutional block on the decoder side consist of two convolutional layers. Where the first halves the number of

![Figure 8.1: Architecture of implemented network.](image-url)
filters and the second output the same number of feature maps as it was inputted. This is a similar structure to the one in the convolutional block on the encoder side, however here it halves the number of filters rather than doubling. Additionally, at the very highest level on the decoder side the last convolutional layer consists of a mapping to a single feature map yielding a decoded output of size $256 \times 256 \times 1$. Furthermore, batch normalization is realized during training after the first convolution operator to decrease the risk of overfitting and speed up convergence rate. The encoder structure is presented in the subsequent schematic graph.

![Figure 8.2: Architecture of encoder part.](image1)

Further, in both the encoder and decoder part the convoluted quantity is sent through the activation function leaky ReLU with parameter $\alpha = 0.1$. Additional parameters for the blocks in each level for the encoder part is presented in Figure 8.2. The left box contains parameters for the first convolution at the highest level and the right square the remaining. Furthermore, the parameters for each layer in the decoder part is presented in Figure 8.3. The right box indicates the very last convolution followed by a softplus activation, assuring that the voxel values of the predicted doses are positive quantities.

![Figure 8.3: Architecture of decoder part.](image2)

Hence, the output of the denoising autoencoder (DAE) framework is an image of same the size as the input containing one channel and will be denoted $r(\tilde{x}) \in \mathbb{R}^{256 \times 256 \times 1}$ in the next section.
8.1.1 Denoising framework

The network is trained on the denoising criterion presented in Section 6.2.2, which we motivate by the result derived in Section 6.2.2. Hence the DAE is trained to minimize the average reconstruction loss $\ell_2$

$$\mathcal{L}_{\text{DAE}, \ell_2} = \mathbb{E}_{x, \tilde{x}} [||r(\tilde{x}) - x||^2]$$

formulated into the empirical loss based on the training set $\mathcal{T} = \{\tilde{x}_k, x_k\}_{k=1}^K = \{\tilde{D}_k, D_k\}_{k=1}^K$, where $\tilde{D}_k$ is a noisy dose distribution for a treatment plan $k$, and $D_k$ is the corresponding clean dose distribution. Gradient descent is performed using Adams Algorithm, which uses a similar approach as Stochastic gradient descent. The used learning rate is $\eta = 10^{-5}$ and the batch size fed is 32. Further, the accuracy is defined as the fraction of voxels that pass the voxel-test presented in Section 2.3.4.

8.1.2 Considered inputs

Let us now consider the inputs used in the implementation of the considered network structure. As have been mentioned, the considered network can take up to 3 channels as input, where the information in each channel varies between the implemented methods. In this thesis we will consider 3 different input quantities, 1 one channel input and 2 three channel inputs. The combinations of such channels are presented in Table 8.1.

<table>
<thead>
<tr>
<th>Reference name</th>
<th>Channels</th>
<th>Ch 1</th>
<th>Ch 2</th>
<th>Ch 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC1</td>
<td>$C = 1$</td>
<td>Dose $D_i$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DC3</td>
<td>$C = 3$</td>
<td>Dose $D_{i-1}$</td>
<td>Dose $D_i$</td>
<td>Dose $D_{i+1}$</td>
</tr>
<tr>
<td>DSP</td>
<td>$C = 3$</td>
<td>Dose $D_i$</td>
<td>Variance $\sigma_{E_i}$</td>
<td>Prior $P_i$</td>
</tr>
</tbody>
</table>

Table 8.1: Various placeholders for input used in the analysis, index $i$ denotes the slice to be denoised.

The inputs are referred to as $DC1$, $DC3$ and $DSP$. The first input structure, $DC1$, is essentially trained to map a noisy dose slice to a clean dose slice. For the subsequent, $DC3$, the two neighboring noisy slices are fed along with the intermediate, to be mapped to the intermediate clean dose. In this input the network is provided more information through these neighboring dose slices. Lastly, for the input structure $DSP$, the network is fed with a dose slice together with (i) the standard deviation of the dose slice and (ii) a generated prior. This standard deviation is an additional output from the MC simulation, and is obtained as described in Section 3.3. The prior slice is fed to the network to provide information of the shot positions in the treatment plan, and is constructed using a multivariate Gaussian distribution with mean at the shot position $p_i(x, y, z)$ and covariance is dependent of the sector-configuration, $\sum_{s\in\text{info}} \mathcal{N}(p_i(x, y, z), \Sigma(s, \text{info}))$. The Gaussian distribution is motivated by the general scatter around the shot-position, in a rough approximation.
The output is fixed to contain only one channel, which subsequently is used for the evaluation in the network learning. Moreover, the input structure $DC1$ will be considered for different voxel resolutions, by downsampling the generated dose kernels with a median filter. The dimension of the considered inputs of the network is presented in Table 8.2.

<table>
<thead>
<tr>
<th>Input size</th>
<th>$d = 1$</th>
<th>$d = 2$</th>
<th>$d = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$256 \times 256 \times 1$</td>
<td>$128 \times 128 \times 1$</td>
<td>$64 \times 64 \times 1$</td>
</tr>
</tbody>
</table>

Table 8.2: Implementation of downsampling ($DC1$), input sizes.

Note that the resolution impact is only considered for $DC1$. The variable $d$ denotes how many downsamplings that have been performed with a mean-kernel. Let us now state the considered implementations. The three main implementations are as follows

- Investigation of network denoising performance dependent on $N_{noisy}$. In this consideration, one channel noisy doses ($DC1$) are trained to be mapped with clean doses $N_{clean} = 10^9$. The considered values of $N_{noisy}$ are $10^4, \ldots, 10^8$.

- Investigation of network denoising performance with additional information, i.e. comparing $DC1$ with $DC3$ and DSP. Here the network performance will be considered for $N_{noisy} = 10^6$, and the three different input structures presented in Table 8.1.

- Investigation of network denoising performance with different resolutions, for setup $DC1$. Again, here we will consider $N_{noisy} = 10^6$ and $C = 1$, for the downsampling operations $d = 1, 2, 3$ with input sizes presented in Table 8.2.

All of the above considerations are trained to be mapped to clean doses defined at $N_{clean} = 10^9$, and are subsequently evaluated in terms of the assembled dose volume.

### 8.2 Considered data

The construction of the considered data is outlined in full detail in Chapter 7. In summary, the dose kernels for the different sector configurations are generated with the MC simulation code system PENELOPE and are used for the realizations of 40 treatment plans for the training set $T$ and 5 independent treatment plans for the test set $S$. Further, 5 validation plans $V$ are generated with independent dose kernels and independent treatment plans.

### 8.3 Software and hardware

As mentioned, the data used in implementation is generated with PENELOPE\(^1\). The implementation of the DAE is done in the open source software package Tensorflow-gpu version 1.10.0. The hardware used in the implementation is the Graphical Processor Unit NVIDIA Titan (GeForce GTX 1080 Titan), ran on an Ubuntu 16.04 and CUDA compiler V7.5.17. All implementation is done using Tensorflow software and programming is done in Python 3.6. For license issues the code is not available for public and owned by Elekta.

\(^1\)Monte Carlo "code system for the simulation of coupled electron-photon transport" [51]
Chapter 9

Result

In this chapter, the results from the implementation which was described in Chapter 8 will be presented. The results boil down to 3 different main areas, considering the three different experiments that was outlined in the implementation chapter. These areas evaluate the network denoising performance dependent on (i) the level of noise in the input, (ii) the provided input information, and lastly (iii) the resolution of the considered dose distributions. These three categories will be presented in the three main sections, namely Denoising performance depending on $N_{\text{noisy}}$, Denoising performance depending on input structure and Resolution impact. Further, the network’s denoising performance will be evaluated with the metrics outlined in Section 7.2.1 and with additional visual inspection of the isodose contours.

9.1 Denoising performance depending on number of photon showers

Let us first consider the network denoising performance dependent on the simulation parameter $N_{\text{noisy}}$. As described in Chapter 7, the level of noise in the dose distribution data set is strictly dependent on the number of photon showers $N$. Moreover, the level of noise is decreasing proportionally to $N^{-1/2}$. Therefore, the denoising task becomes tougher for inputs generated with a lower number of photon showers $N_{\text{noisy}}$. Let us now consider the obtained results. The network trained on the training set $T$, generated with simulation parameters described in Section 7.2.2 and $N_{\text{noisy}}=10^6$, yields the following statistics of the predicted noise-free dose distributions, see Table 9.1. Recall that the stacks of predicted dose slices have been assembled into the corresponding dose volume, i.e. $D = \{D_{z,j}^{256}\}_{j=1}^{256} \in \mathbb{R}^{256 \times 256 \times 256}$, which subsequently is evaluated.

<table>
<thead>
<tr>
<th>$P_i$</th>
<th>$P_{i}^{S}$</th>
<th>$P_{i}^{S}$</th>
<th>$P_{i}^{S}$</th>
<th>$P_{i}^{S}$</th>
<th>$P_{i}^{V}$</th>
<th>$P_{i}^{V}$</th>
<th>$P_{i}^{V}$</th>
<th>$P_{i}^{V}$</th>
<th>$P_{i}^{V}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2D - \gamma$ (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>$\Delta_T$ [eV/g]</td>
<td>74</td>
<td>140</td>
<td>120</td>
<td>94</td>
<td>93</td>
<td>146</td>
<td>132</td>
<td>158</td>
<td>42</td>
</tr>
<tr>
<td>$\Delta_R$</td>
<td>0.03</td>
<td>0.03</td>
<td>0.01</td>
<td>0.03</td>
<td>0.02</td>
<td>0.01</td>
<td>0.03</td>
<td>0.01</td>
<td>0.06</td>
</tr>
<tr>
<td>$\beta$ (%)</td>
<td>73</td>
<td>90</td>
<td>86</td>
<td>58</td>
<td>67</td>
<td>85</td>
<td>74</td>
<td>89</td>
<td>37</td>
</tr>
</tbody>
</table>

Table 9.1: Evaluation of predictions for each test $P^S$ and validation $P^V$ data sample. Here the input setup considered is DC1 and estimation parameter is $N_{\text{noisy}} = 10^6$. 

69
Recall that the noisy input doses have $2D - \gamma$ values in the range $47 - 91\%$, see Table 7.2, and further recall that all gamma tests are done with the gamma criterions $\delta_{DTA} = 1\,mm$ and $\delta_{DD} = 0.03 \times \max(D_{ref})$. In Table 9.1 we can see that all of the considered denoised dose volumes in test and validation set, generated with $N_{noisy} = 10^6$, obtained acceptance rates above 98%. Additionally, the majority of the denoised dose volumes have a relative error less than 5%. However, sample $P^V_4$ is an outlier with comparatively low absolute error and high relative error. This sample will be further discussed in the discussion chapter.

Further, let us now consider the network denoising performance for different noise levels and compare them with the initial errors which were presented in Section 7.2.4. The number of photon shower considered are $N_{noisy} = 10^4, \ldots, 10^8$, and the network output and the input statistics of the dose distributions are shown in Figure 9.1.

- (a) Mean of total absolute error $\Delta_T$ for $S$ and $V$.
- (b) Mean of relative error $\Delta_R$ for $S$ and $V$, compared with $\Delta_R \propto \epsilon \times N^{-1/2}$ where $\epsilon = 1.2$.
- (c) Mean of $2D - \gamma$ for $S$ and $V$.
- (d) Mean of $\beta$ for $S$ and $V$.

Figure 9.1: Mean statistics of the five test and validation samples for different number of photon showers $N$. Prior and after network denoising.
Further, let us now consider the network performance for different noise levels and compare it with the initial errors which were presented in Section 7.2.4. The number of photon shower considered are $N_{\text{noisy}} = 10^4 \ldots, 10^8$. The network output statistics and the input statistics of the dose distributions are shown in Figure 9.1.

We can see that there is a reduction in total and relative error for the prediction outputs compared to the initial inputs, see Figure 9.1a and 9.1b. This holds for both the test and validation set, however the validation set values are slightly above the test set for all $N$. We will return to look at Figure 9.1b in more detail shortly. Moreover, in Figure 9.1c we can see that the mean of the $2D - \gamma$ values for the predicted doses reach approximately $100\%$ at $N = 10^6$, for both test and validation set. Additionally, in Figure 9.1d we can see the same type of shift of the accepted dose voxels as in Figure 9.1c.

Let us now consider Figure 9.1b more closely. Here we can state that the network denoised volumes have relative errors which is consistently below 0.2, which are comparably lower than the original relative mean error which initially is roughly 1. To achieve a better view of the relative errors at larger number of photon showers, Figure 9.1b is instead shown in the loglog-graph Figure 9.2.

![Loglog-graph of the mean relative error of the denoised dose volumes and the initial dose volumes, for test $S$ and validation $V$ set. The error parameter has value $\epsilon = 1.2$.](image)

The mean relative errors of network denoised dose distribution, with $N = 10^6$, are of the same magnitude as doses that have been generated with $N = 10^8$. Additionally, the network denoised doses with $N = 10^4$ achieves the same range of the relative errors as distributions that have been simulated with parameter $N = 10^6$. For an overview of the noise levels in the samples, see Figures in 3.3.
Comparison between isodose contours of predictions and true distributions

Now, let us consider the isodose contours of the predicted dose distribution for two samples of the test set, and two of the validation set. Further, let us consider the isodose contours of the relative dose at a center slice of the predicted dose distributions and compare them with the isodose contours for the reference dose contours.

![Comparison of isodose contours](image)

Figure 9.3: Comparison of isodose contours of true dose contours (black) and contours of the predicted dose (blue), for test and validation samples for setup $N_{\text{noisy}} = 10^6$.

It can be stated that the contours of predicted doses and the true contours are alike, for the considered case $N_{\text{noisy}} = 10^6$. In fact, almost all of the samples have almost indistinguishable dose contours, see Figures 9.3a, 9.3b and 9.3d. Although, there was one sample that had discrepant contours, compared to the true contours. This sample is shown in Figure 9.3c.

![Comparison of isodose contours](image)

Figure 9.4: Comparison of isodose contours of true dose contours (black) and contours of the predicted dose (blue). Shown for a validation sample $P^V_2$ for different number of photon showers $N_{\text{noisy}} (e)-(h)$.

In Figure 9.5 the isodose contours for the different considered inputs are shown. Here we can see that for the $N_{\text{noisy}} = 10^4$ case the contours are choppy compared to the true contours. Although, the other cases have similar shapes as the true contours.

1See Appendix A.5 for the other contours
9.2 Denoising performance depending on input structure

The input data structure is now extended to 3 channel inputs, as described in Section 8.1.2, providing additional information to the network. Recall that we here compare the three input structures $DC_1$, $DC_3$ and $DSP$. The former input structure, $DC_1$, was presented in the previous Section 9.1 and hence will only be referenced for comparison to the remaining two. Let us firstly remind us of the remaining input structures. The considered input structure for $DC_3$ is; 3 channel dose input attaching the two neighbouring dose slices to the network, which are to be mapped to the intermediate clean dose slice. The second considered input structure, $DSP$, contains the dose at the considered slice, the corresponding standard deviation for each voxel of the dose in that slice and an engineered prior. Recall that this prior is generated through a mixture of multivariate Gaussians, with distribution parameters dependent on the RS treatment plan. Further, the network’s denoising performance are considered for $N_{noisy} = 10^6$, and are compared with the dose input structure $DC_1$ in the last subsection.

9.2.1 DC3 input

The network trained with input structure $DC_3$ on the training set $T$, predicts dose distributions with the statistics presented in the subsequent table.

<table>
<thead>
<tr>
<th>$P_i$</th>
<th>$P_1^S$</th>
<th>$P_2^S$</th>
<th>$P_3^S$</th>
<th>$P_4^S$</th>
<th>$P_5^S$</th>
<th>$P_1^V$</th>
<th>$P_2^V$</th>
<th>$P_3^V$</th>
<th>$P_4^V$</th>
<th>$P_5^V$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2D - \gamma$ (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>99</td>
<td>100</td>
<td>100</td>
<td>97</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>$\Delta_T$ [eV/g]</td>
<td>75</td>
<td>146</td>
<td>120</td>
<td>95</td>
<td>96</td>
<td>148</td>
<td>136</td>
<td>165</td>
<td>42</td>
<td>131</td>
</tr>
<tr>
<td>$\Delta_R$</td>
<td>0.03</td>
<td>0.02</td>
<td>0.01</td>
<td>0.03</td>
<td>0.02</td>
<td>0.01</td>
<td>0.03</td>
<td>0.02</td>
<td>0.06</td>
<td>0.02</td>
</tr>
<tr>
<td>$\beta$ (%)</td>
<td>73</td>
<td>90</td>
<td>86</td>
<td>58</td>
<td>67</td>
<td>85</td>
<td>75</td>
<td>89</td>
<td>36</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 9.2: Network prediction statistics for test data $P^S$ and validation data $P^V$. The input doses are generated with $N_{noisy} = 10^6$ and evaluated with doses generated with $N_{clean} = 10^9$. The input structure type was DC3.

Note that most of the test and validation samples have 100% non-rejected $2D - \gamma$ tests, however, $P_2^V$ has a comparatively lower gamma acceptance rate of 97%. In Table 9.2, it is shown that validation set 4, i.e. $P_4^V$ deviates from the other, regarding the relative error. However, the total error in $P_4^V$ is lower than the other samples. Let us now consider the isodose contours.

Figure 9.5: Comparison of isodose contours of true dose contours (black) and contours of the predicted dose (blue). Here we are considering the input $DC3$ and setup $N_{noisy} = 10^6$.  

\footnote{Note that a sum of Gaussians is a mixture of Gaussians}
9.2.2 DSP input

The network trained with DSP input, yield predictions of the dose distributions of the test and validation, which has the following statistics

<table>
<thead>
<tr>
<th></th>
<th>$P_i^S$</th>
<th>$P_2^S$</th>
<th>$P_3^S$</th>
<th>$P_4^S$</th>
<th>$P_5^S$</th>
<th>$P_i^V$</th>
<th>$P_2^V$</th>
<th>$P_3^V$</th>
<th>$P_4^V$</th>
<th>$P_5^V$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2D - \gamma$ (%)</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>99</td>
<td>100</td>
<td>99</td>
<td>100</td>
</tr>
<tr>
<td>$\Delta_T$ [eV/g]</td>
<td>92</td>
<td>152</td>
<td>136</td>
<td>102</td>
<td>111</td>
<td>154</td>
<td>141</td>
<td>166</td>
<td>64</td>
<td>136</td>
</tr>
<tr>
<td>$\Delta_R$</td>
<td>0.04</td>
<td>0.02</td>
<td>0.02</td>
<td>0.03</td>
<td>0.03</td>
<td>0.02</td>
<td>0.03</td>
<td>0.02</td>
<td>0.13</td>
<td>0.02</td>
</tr>
<tr>
<td>$\beta$ (%)</td>
<td>66</td>
<td>87</td>
<td>83</td>
<td>55</td>
<td>62</td>
<td>83</td>
<td>66</td>
<td>87</td>
<td>25</td>
<td>79</td>
</tr>
</tbody>
</table>

Table 9.3: Network prediction statistics for test $P^S$ and validation $P^V$ data samples. The input doses are generated with $N_{\text{noisy}} = 10^6$ and evaluated with doses generated with $N_{\text{clean}} = 10^9$. The input structure was type DSP.

The $2D - \gamma$ rates are roughly 100% for almost all of the validation set samples. Moreover, all of the test samples have $2D - \gamma$ rates of 100%. The prediction of validation sample $P_i^V$ has roughly 13% relative error while the remaining relative errors are in the range 2 – 4%. Let us further consider the isodose contours of the predicted doses, for this specific setup.

Figure 9.6: Comparison of isodose contours of true dose contours (black) and contours of the predicted dose (blue). Here we are considering input setup $N_{\text{noisy}} = 10^6$ and $DC3$.

9.2.3 Comparison between DC1, DC3 and DSP

Let us now compare the performance for the three different input structures. Since all of the considered dose distributions for the different input setups have $2D - \gamma$ values over 97%, and the majority has 100%, this metric will not be used in the comparison. Instead, let us consider the remaining metrics, $\Delta_T$, $\Delta_R$ and the voxel-test $\beta$. In Figure 9.7a scatter plots are shown, together with their projected kernels, of the different metrics for the predictions of the different input structures. Note that in Figure 9.7a it is shown that the network which been has trained on the DSP input structure performs comparatively worse than the remaining input structures. Indeed, both of the projected kernels are shifted to higher values of the total and relative error.

Note that kernel here refers to the smoothed conditional distribution.
Let us now consider the scatter-plot of voxel-test and the relative error, see Figure 9.7b. Again, the network which has been trained on input structure DSP, generally has lower $\beta$-values for the considered samples. Furthermore, note that the networks that have been trained on DC3 and DC1, have very similar statistics over the considered samples. Before moving further, we will compare the isodose contours in the Figures 9.3, 9.5 and 9.6.

The networks trained on considered inputs DC1, DC3 and DSP obtained very accurate dose contours in the very center of the doses. However, the latter, DSP had irregular dose contours in the outer levels, see Figure 9.6a.

### 9.3 Resolution impact

Firstly we want to stress that as the dose distributions are downsampled with the mean filter, the number of samples in each voxel turn into $2^3 = 8$ times greater. Thence, the level of noise is decreased for lower resolution inputs. Additionally, as the input sizes are increased the width of the network is increased as well, recall the architecture in Figure 8.1.

<table>
<thead>
<tr>
<th>$d_i$</th>
<th>$d=1$</th>
<th>$d=2$</th>
<th>$d=3$</th>
<th>$d=1$</th>
<th>$d=2$</th>
<th>$d=3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2D-\gamma$ (%)</td>
<td>100</td>
<td>100</td>
<td>97</td>
<td>100</td>
<td>100</td>
<td>92</td>
</tr>
<tr>
<td>$\Delta T$ [eV/g]</td>
<td>137</td>
<td>140</td>
<td>177</td>
<td>157</td>
<td>156</td>
<td>199</td>
</tr>
<tr>
<td>$\Delta R$</td>
<td>0.03</td>
<td>0.04</td>
<td>0.04</td>
<td>0.05</td>
<td>0.05</td>
<td>0.06</td>
</tr>
<tr>
<td>$\beta$ (%)</td>
<td>46</td>
<td>47</td>
<td>47</td>
<td>46</td>
<td>49</td>
<td>48</td>
</tr>
</tbody>
</table>

Table 9.4: Mean statistics of test data (left) and validation data (right), for $d$ downsampleings of the input.
Chapter 10

Discussion

Let us now ponder the results presented in Chapter 9. Starting with the dependence on the number of primary photon showers. Generally, we can say the network trained on the disjoint training set succeed to generalize the denoising task to unseen dose distributions and unseen noise. This conclusion can be drawn since the denoising of the validation set yields roughly identical values in all of the considered metrics for all of the different $N$, see Figure 9.1. Additionally, Figure 9.2 showed that the network manages to denoise the input such that it comprised the same relative error as a dose simulation with a $10^2$ times larger simulation parameter $N$. For example, a prior simulation generated with $N_{\text{noisy}} = 10^6$, can be mapped to a dose distribution which has been generated with $N_{\text{noisy}} = 10^8$ samples, meaning $10^2$ times longer simulation. For comparison, see Figure 9.2, a $N_{\text{noisy}} = 10^6$ simulation takes roughly 20s in PENELOPE with the considered hardware, while a $N_{\text{noisy}} = 10^8$ simulation takes approximately 40min. Further, for the few photon shower case, $N_{\text{noisy}} = 10^5$, the simulated doses can be mapped to contain less than 10% relative error in the whole dose distribution volume.

Moving on to the input structures. The results shown in Section 9.2.3 indicates that the prior knowledge regarding neighboring slices does not affect the network denoising performance, as $DC_3$ have very similar performance as $DC_1$. Additionally, the engineered prior in this setup seems to decrease the network denoising performance rather than improving it. These results then suggest that the engineered prior was poorly constructed and was not accurate enough. Thence, in summary we can conclude from the results presented in Section 9.2.3 that the network denoising performance was not affected by this type of prior knowledge. Let us now consider the results shown in Section 9.3. For the constructed network and the obtained results, we can conclude that the network’s denoising performance is decreased for lower resolution and smaller network architecture. These results indicate that the networks denoising performance is decreased for a smaller architecture which is a consequence that was expected.

Generally, for all the considered setups, the predictions of the dose distribution for validation sample $P_4^V$ yielded comparably high relative error rates. This treatment plan differs from the others in terms of the number of shots and thence in terms of delivered dose, see Appendix A.5. Additionally, this sample had relatively low local information, as the scattering effects were very locally compact. Although it was expected that the input structure $DSP$ would perform better than the others on this sample since the prior would incorporate this information. This was however not the case. In fact, it was shown to be the opposite.
The metrics that have been used in this experiment suffer from some downsides and have some advantages. More specifically, the metrics total and relative error in fact provided us with additional information regarding the denoising performance which we were unable to quantify with the used gamma test. The additional information was how much the integrated error was, and how relatively large this error was between the samples. The voxel test $\beta$ did not provide any useful information regarding the denoising performance but was useful since it gave us information regarding the voxel-wise dose precision of the predictions.

Let us now contemplate these results in a more general manner. Based on the familiar saying in deep learning community, the network can never be better than the data that you feed it with. In the sense of that the networks performance and accuracy are only based on the information that is fed to it, and hence, the results presented in this thesis must be considered with cautiousness. This is mainly since the network denoising is trained on the constructed dose distributions rather than full simulated dose distributions. Additionally, the underlying data generator is based on wide approximations regarding the geometry of a human head. This homogeneous geometry assumption we propose to be extended to a more general heterogeneous case in future work.

**Future work**

In this work, I have considered the general ability for a deep neural network to predict the forthcoming dose distributions, for a simplified geometry. The geometry considered is homogeneous and symmetric. Hence, in future work I hope this setup will be considered in a more realistic setting, feeding the network a wider spread of geometries and non-homogeneous considerations. Furthermore, I propose that the network will be trained with a larger set of training samples. Lastly, we propose future work to examine the networks denoising performance when a more accurate prior is incorporated into the network.
Chapter 11

Conclusion

The constructed denoising autoencoder (DAE), with an architecture similar to the U-net, showed promising results in denoising of MC calculated radiosurgery (RS) dose distributions. The results indicated that the denoiser could reduce the level of noise with an amount comparable with simulations that were done with $10^2$ times more samples. This conclusion could be drawn based on the considered metrics, total and relative error. Furthermore, the network denoising performance was not notably increased with additional information such as the engineered prior and the neighbouring dose slices. In fact, when the engineered prior was provided to the network, the denoising performance was decreased. The overall results in this thesis suggest that the DAE successfully decrease the computational time. Additionally, it suggests that it can be deployed in a radiosurgery (RS) dose optimization algorithm using MC calculated dose distributions, to decrease the computational time while not reducing the accuracy.
Bibliography


Appendix A

A.1 MC unbiased estimate

Note that $\tau = \mathbb{E}[\phi(X)]$, and let us now derive that the estimate is unbiased. The Monte Carlo estimate is given by

$$\tau^{MC}_N = \frac{1}{N} \sum_{i=1}^{N} \phi(X) = \tau. \quad (A.1)$$

Thence, the mean of the estimate is given by

$$\mathbb{E}[\tau^{MC}_N] = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}[\phi(X)] = \tau \quad (A.2)$$

and hence $\tau^{MC}_N$ is an unbiased estimate of $\tau$.

A.2 SNIS

Deriving the expression in equation (3.5), recall the defined importance weights are defined $w(x) = cf(x)/g(x)$ and $c \in \mathbb{R}$

$$\tau = \mathbb{E}_f[\phi(X)] = \int_X \phi(x)f(x)dx$$

$$= \frac{c \int_{f(x)>0} \phi(x)f(x)dx}{c \int_{f(x)>0} f(x)dx} = \frac{\int_{g(x)>0} \phi(x)\frac{cf(x)}{g(x)}g(x)dx}{\int_{g(x)>0} \frac{cf(x)}{g(x)}g(x)dx} \quad (A.3)$$

$$= \frac{\int_{g(x)>0} \phi(x)w(x)g(x)dx}{\int_{g(x)>0} w(x)g(x)dx} = \frac{\mathbb{E}_g[\phi(X)w(X)]}{\mathbb{E}_g[w(X)]} \quad (A.4)$$

$$= \frac{\mathbb{E}_g[\phi(X)w(X)]}{\mathbb{E}_g[w(X)]} \quad (A.5)$$
A.3 Rejection sampling

The rejection sampling Algorithm presented below generates one sample \( X \sim f \)

\[ \text{Algorithm 3 Rejection Sampling} \]

\[ \text{Result: A sample } X \sim f \]

\[ \text{accepted} \leftarrow \text{False} \]

\[ \text{while } \text{accepted} = \text{False} \text{ do} \]

\[ \text{Draw } X^* \sim g \]

\[ \text{Draw } U \sim \mathcal{U}(0,1) \]

\[ \text{if } U \leq \frac{f(X^*)}{\mathcal{K} g(X^*)} \text{ then} \]

\[ \text{accepted} = \text{True} \]

\[ \text{end} \]

\[ \text{end} \]

\[ \text{Return } X^* \]

A.4 Slices of test and validation set

![Visualization of test and validation set](image)

(a) Test sample 1, \( P_{S1} \)  
(b) Test sample 2, \( P_{S2} \)  
(c) Test sample 3, \( P_{S3} \)  
(d) Test sample 4, \( P_{S4} \)  
(e) Test sample 5, \( P_{S5} \)  
(f) Validation sample 1, \( P_{V1} \)  
(g) Validation sample 2, \( P_{V2} \)  
(h) Validation sample 3, \( P_{V3} \)  
(i) Validation sample 4, \( P_{V4} \)  
(j) Validation sample 5, \( P_{V5} \)

Figure A.1: Visualization of test and validation set, intensity maps at a center dose slice in each volume. These samples are generated with simulation parameter \( N = 10^9 \).
A.5 Isodose contours of prediction samples

Figure A.2: Visualization of test and validation set, contours at a center dose slice in each volume. The predicted dose contours are shown in blue and the true contours in black.