Karlsruhe Institute of Technology

ETP-KA/2021-8

Fast Simulation and Validation of the Time of Propagation Detector at the Belle II Experiment

Isabel Haide

Masterthesis

16th July 2021

Institute of Experimental Particle Physics (ETP)

Advisor:	Prof. Dr. Günter Quast
Coadvisor:	Prof. Dr. Ulrich Husemann

Editing time: 7th June 2020 – 16th July 2021

Karlsruhe Institute of Technology

ETP-KA/2021-8

Fast Simulation und Validierung des Time of Propagation Detektors am Belle II Experiment

Isabel Haide

Masterarbeit

16. Juli 2021

Institut für Experimentelle Teilchenphysik (ETP)

Referent:	Prof. Dr. Günter Quast
Korreferent:	Prof. Dr. Ulrich Husemann

Bearbeitungszeit: 7. Juni 2020 – 16. Juli 2021

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, 16. Juli 2021

. . . .

(**Isabel Haide**)

Contents

1. Introduction

The study of nature of matter and light has been one of the main focuses of physics since its origins. The theory of matter being made out of discrete building blocks originated in ancient Greece and ancient India and today has culminated in the Standard Model (SM) of Particle Physics [1]. The SM describes the elementary particles and three of their four fundamental interactions, currently excluding gravity, with high precision and has correctly predicted several results, such as the experimental detection of the top quark [2] and the Higgs boson [3]. Despite its great success, there are still several discrepancies and inexplicable phenomena, including as the baryon asymmetry, the inclusion of gravity and the existence of neutrino oscillations.

The SuperKEKB collider is an electron-positron collider operating at the $\Upsilon(4S)$ resonance with the Belle II experiment positioned at the collision point. This allows high precision measurements of the properties of B mesons and their decays and thus to find possible deviations of Standard Model predictions. [4]

For this purpose the Belle II detector is built with several layers of detector components centered around the collision point. The different interactions of the collision products with each detector part can be measured and stored for further purposes. These measurements are then analysed and compared to theory predictions to possible find discrepancies to the SM or to further verify its predictions.

For a correct analysis, an accurate simulated dataset is crucial. Large amounts of collisions need to be created, where both the particle and its decays as well as the interaction of each detector component with the particle have to be simulated to a high precision. As simulation time and resources become bottlenecks for physical analyses due to higher luminosities, the study of more efficient simulation methods gains higher traction in the field of physics ([5], [6], [7]).

The currently largest contributor to overall detector simulation time at the Belle II experiment is the imaging Time of Propagation (TOP) detector. This detector is part of the particle identification done through the detection of Cherenkov photons emitted by charged particles crossing the detector. This thesis is exploring the concept of replacing the TOP simulation by a simulation done through machine learning methods, also called fast simulation. A focus of this thesis is validating a possible fast simulation, thus ensuring a correct modeling of the detector.

The thesis is structured as follows: chapter 2 explains the theoretical and mathematical background needed for a fast simulation in particle physics, including a high-dimensional Kolmogorov-Smirnov metric which was developed during this thesis. In chapter 3 the experimental setup is explained, including the Time of Propagation detector. chapter 4 then explores the uncertainty of the current simulation and the development of validation processes, with the creation of a possible validation framework. In chapter 5 two neural network architectures, conditional variational autoencoders and generative adversarial networks, are tested and applied to simplified datasets as a proof-of-principle.

2. Theoretical Background

2.1 The Standard Model

The idea of a grand unified theory combining all fundamental forces has been the objective of physics since their discovery. To our current knowledge, particle interaction is governed by four different forces, the electromagnetic, the weak, and the strong force together with the well known force of gravity. This following chapter is mainly based on [1].

Two of those four, the electromagnetic force and gravity, had been discovered before the 20th century. Enrico Fermi first postulated the existence of a non-contact force that has a finite range as opposed to the long ranges of the electromagnetic and the gravitational force to explain beta decay. The name weak interaction stems from the short range of this force.

To explain why an atomic nucleus is stable even though the electromagnetic force between the protons is repulsive, a stronger attractive force inside the nucleus was postulated. Through collider experiments, the existence of quarks that make up neutrons and protons inside the nucleus was experimentally shown. The theory of quantum chromodynamics could then explain the attractive force between quarks through the exchange of gluons.

All three forces are combined in the Standard Model (SM) of Particle Physics. It is used to describe the interactions between elemental particles while not including the influence of gravity. The model was developed in stages with the last addition being the Higgs mechanism to explain the masses of particles. With the experimental confirmation of the existence of quarks in the 1970s, the current version of the model was finalized.

The widely accepted version of the SM includes 12 fermions, which are particles with a half-integer spin, and 5 bosons, which have integer spin. The fermions are broken up into six quarks, three leptons, and their respective lepton neutrinos. Each fermion has its corresponding anti-particle. In figure 2.1 all fundamental particles with their corresponding mass, charge and spin are listed. Every fermion carries types of charge. While all fermions carry electric charge and weak isospin, which defines their interaction through the electromagnetic and the weak interaction, quarks are defined by also possessing color charge which is the charge of the strong interaction. Bosons are separated into gauge bosons and the Higgs boson. The gauge bosons are the carriers of the fundamental forces. Fermions exchange gauge bosons to interact with each other through the respective force. Photons are the exchange bosons of the electromagnetic force which can be described by the theory of quantum electrodynamics. As the photon is massless, the electromagnetic force

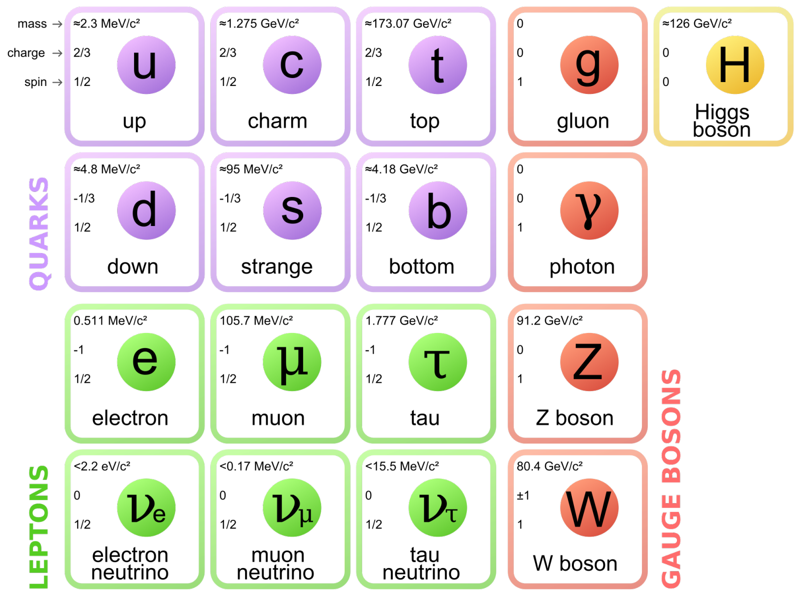

Figure 2.1: The particles of the Standard Model of Particle Physics. The three columns on the left list the fermions with the six quarks on top and the six leptons on the bottom. On the second to right column the four gauge bosons are shown, while the right column shows the Higgs boson. The figure is taken from [8].

is long-range. The gauge bosons of the weak interaction are the W^+, W^-, and Z-bosons. In comparison to the electromagnetic force, the weak force is short-range which can be explained by the masses of the gauge bosons. Those masses exceed the masses of the neutron and proton, making the exchange bosons short-lived and the field strength of the weak force over a given distance smaller than that of the strong or electromagnetic force. In the 1960s the electromagnetic force and the weak interaction were unified in the electroweak theory. The strong force describes the interaction between colored particles inside the nucleus and is mediated by gluons. As the current theory includes three colors and every gluon carries a color-anticolor pair, eight different gluons exist. All gluons carry an effective color charge and are therefore able to interact amongst themselves as well. The Higgs boson is the only spin-0 particle in the SM, as the gauge bosons all carry spin 1. It is massive and therefore short-lived. Its role inside the SM is to explain the existence of masses of all other particles. In particular, the existence of the Higgs boson explains why three of the four bosons of the electroweak theory (W^+, W^-, and Z-bosons) are massive and the fourth boson, the photon, is massless. Through a different interaction, the mass of the leptons and quarks are also generated by the Higgs boson. The experimental confirmation of the existence of the Higgs boson in 2012 ([3]) further strengthened the position of the Standard Model. Even though many experimental results support the validity of the SM, several effects cannot be explained by it. A prominent example is the appearance of Dark Matter in the universe, which to our current knowledge does not consist of any of the known particles. Other examples are neutrino oscillations, the matter-antimatter asymmetry, and the task of combining gravity with the other fundamental forces. To provide more information on physics beyond the Standard Model, high-energy and high-precision experiments are built to measure processes that may contain information on new physics. One of these experiments is the Belle II experiment designed to measure flavor physics to very high precision, which will be explained in section 3.3.

2.2 Machine Learning and Generative Models

Machine learning algorithms are a class of computer algorithms where the optimal model is learned from the data instead of classically implemented into an algorithm structure. Through machine learning the computer iteratively learns to make decisions and improve on its predictive performance through experience.

In general, there exists three types of machine learning: *supervised*, *unsupervised*, and *reinforcement* learning. Different application domains need different types of machine learning and all three types have their own regions of expertise. We will focus on supervised learning in this thesis.

In supervised learning, the task is to learn a function g that maps an input to an output based on example data. The algorithm is given a set of inputs X and their corresponding labels Y. The learning task is then to infer a function $g : X \rightarrow \hat{Y}$, where the space of predicted labels is denoted by $\hat{Y}$ and ideally given through $\hat{Y} = Y$. In most cases, g is a subset of the hypothesis space G, which is the space of possible functions. To evaluate the performance of g, oftentimes a loss function $l : Y \times \hat{Y} \rightarrow \mathbb{R}_+$ measures the difference between the predicted and true label. The optimal function g is then defined as predicting the labels, that minimizes the loss function f.

Many different algorithms can be used in supervised learning, most widely known are for example support-vector machines, decision trees, K-nearest neighbor algorithms, and neural networks. As only neural networks have been used in this thesis, they will be explained in the next section.

2.2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are computational models crudely based on the workings of a biological neuron. They were invented as an attempt to model the human brain and perform tasks too difficult for conventional algorithms. The algorithm is taught to perform certain tasks by giving examples of correctly performed actions and thus iteratively improving its performance. [9] An artificial neural network consists of nodes, which resemble neurons in the brain. These neurons are connected to each other similar to biological synapses and can transmit signals. Generally these connections are called edges. Each connection is weighted to determine the strength of the influence between nodes. The explanation in the following paragraphs is based on [10].

The structure of a single node can be seen in figure 2.2. The artificial neuron receives one or several signals, which are real numbers, being either the original input or coming from another neuron. This signal is then processed and put into a connection. This connection is usually a non-linear function and transmits this signal to the next neuron. Both neurons and edges are weighted by a real number, which is adjustable during the learning process. Through this weight the impact of the respective neuron or edge can be adjusted. Neurons

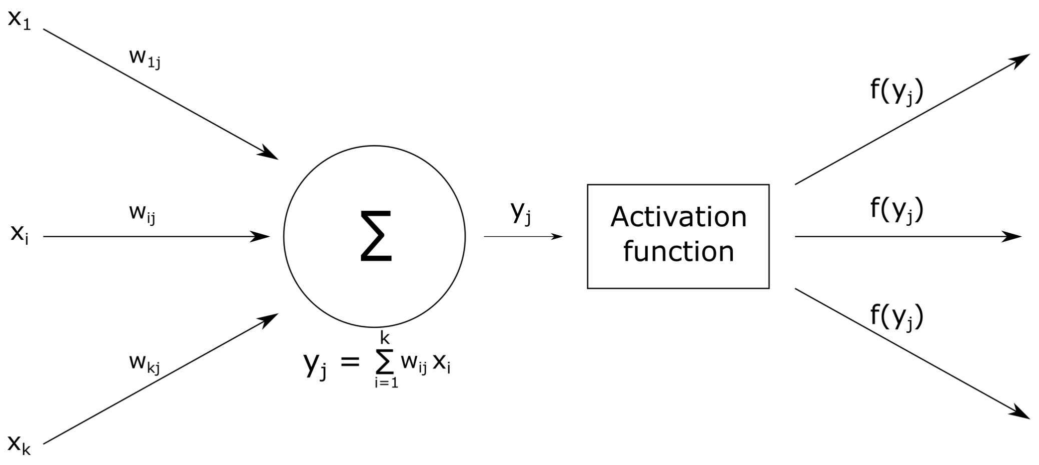

Figure 2.2: A singular node of an artificial neural network. The node receives several inputs with their respective weights. All inputs and their corresponding weights are multiplicated and summed up in that node. This node output is then put through a typically nonlinear activation function and then propagated to the next nodes.

are typically aggregated into layers of nodes, as can be seen in figure 2.3. In this figure, the layers are fully connected layers, where every node in the n-th layer is connected to every node in both the $n-1$-th and the $n+1$-th layer. Other layer types include convolutional layers, recurrent layers etc. depending on the application. Different layers can then learn different aspects of the problem posed.

The parameters of an ANN can be separated into learnable parameters and hyperparameters. Hyperparameters are variables that need to be specified by the user and are not determined by the learning algorithm. The learnable parameters on the other hand are adjusted during the process by the neural network itself.

Artificial neural networks are a way to approximate any well-behaved function g through its learnable parameters. This has been shown by the universal approximation theorem, which proved that an ANN with either arbitrary width and bounded depth or an ANN with bounded width and arbitrary depth with a non-linear activation function can approximate any such function g. That however only proves the existence of such a solution, but gives no information about the design details of such a solution.

The most commonly used algorithm for adjusting the learnable parameters during training is called *backpropagation*. It is mostly used in feedforward neural networks, where connections between nodes are acyclic. During the training process, the gradient of the loss function is computed with respect to the weights of the networks for every input-output pair. This can be done efficiently by computing the gradient of the loss function by the chain rule. the gradient is computed one layer at a time, iterating backward from the last layer. This is faster than a direct computation of the gradient and therefore feasible to use in multilayer networks. According to the computed gradient, the weights are adjusted so that the loss function can be minimized. The key requirement for the use of backpropagation is therefore that the gradient is defined everywhere to perform parameter updates. This introduces

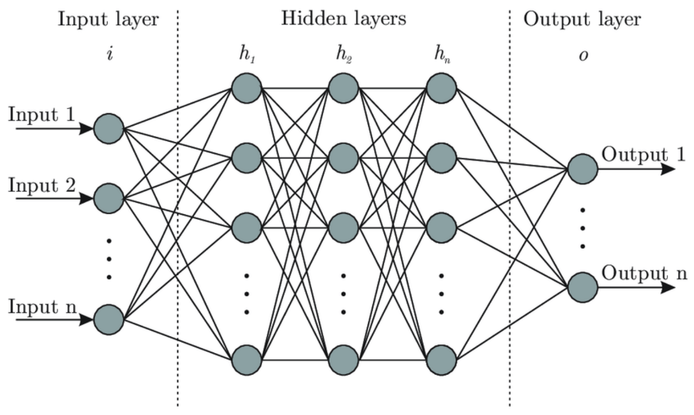

Figure 2.3: Overview of an ANN architecture. ANNs typically consist of an input layer, one or several hidden layers and an output layer. The nodes inside the layers are connected to each other via learnable parameters which are adjusted during the training process. The figure is taken from [11].

requirements on the used loss function and on the design of the networks itself, as seen later in section 2.2.2.2.

The learning process based on backpropagation is called gradient descent. Gradient descent uses the gradient of a multivariate function, in the case of a neural network the loss function f, at the current position $\mathbf{a}$ to adjust the parameters of the network in the direction of the negative gradient $-\nabla f(\mathbf{a})$. For standard gradient descent, the calculation of the gradient is done for every data sample and summed up to calculate the empirical gradient

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \eta \nabla f(\mathbf{a}) = \mathbf{a}_n - \frac{\eta}{n} \sum_{i=1}^{n} \nabla f_i(\mathbf{a}) \quad . \tag{2.1}$$

The parameter η is called the learning rate and controls the speed at which the model learns. It is a configurable hyperparameter needed to be chosen in the range of [0.0, 1.0]. It can be proven that gradient descent will converge to a local minimum of $f(\mathbf{x})$ [12].

If the training set is very large, calculating the gradient for the complete dataset has a high computational cost. For a more efficient calculation of the gradient, stochastic gradient descent is used as an approximation. In stochastic gradient descent the gradient is calculated on one randomly chosen example of the whole dataset and then applied to all weights.

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \eta \nabla f_i(\mathbf{a}) \quad . \tag{2.2}$$

If the chosen examples are i.i.d., the algorithm will update the parameters in the direction of a minimum of the empirical gradient as long as the data set is finite. Additionally, through the noise introduced by the randomly chosen samples, the algorithm may converge towards the global instead of a local minimum as may happen in standard gradient descent. [12]

To alleviate some of the randomness in the full stochastic gradient descent, in most applications mini-batch gradient descent is used. The gradient is computed on more than one training example, where the number of training examples used, called batch size, is a hyperparameter that needs to be set before training. A larger batch size leads to a more exact gradient calculation, while also resulting in a higher computation time.

An important factor of an ANN is the non-linear activation function. In theory, any such function would suffice, but popular functions are amongst others the sigmoid function

$$f_{\text{sigmoid}}(x) = \frac{1}{1+e^{-x}}\,, \tag{2.3}$$

the rectified linear unit (ReLu) function

$$f_{\text{ReLu}}(x) = \max\{0, x\}\,, \tag{2.4}$$

or the leaky ReLu function

$$f_{\text{leaky ReLu}}(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases} \tag{2.5}$$

Activation functions should have three main properties to work in an ANN. Firstly they should be non-linear as explained earlier, to ensure the universal approximation theorem. Secondly activation functions with a finite range seem to stabilize gradient-based training. Lastly for an activation function used in combination with backpropagation, a gradient needs to be defined everywhere. For easier computation, a continuously or mostly continuously differentiable activation function is of advantage.

The success of ANNs can be explained by their variability. Many different neural network techniques have been developed to tackle different kinds of problems. One network design that has been particularly successful for example are convolutional neural networks in image recognition [13]. Other notable types of networks are transformer networks [14], which are very useful in natural language processing, or generative models such as generative adversarial networks (GAN) or variational autoencoders (VAE). As this work focuses on the generative modeling of the TOP detector, both GANs and VAEs have been used. The general concept of these network types is presented in the following section.

2.2.2 Generative Modeling

Generative modeling is a subset of machine learning algorithms. Given a set of data X and a corresponding set of labels Y, generative modeling describes the process of capturing the joint probability distribution $P(X|Y)$ and being able to sample data points x_i from this distribution. The model distribution is learned by giving the network data points drawn from the original distribution (e.g. pictures of handwritten numbers where the overall distribution is all handwritten numbers). The network then has to be able to accurately model $P(X|Y)$ without ignoring a subset of this distribution, e.g. certain labels in Y. [?]

The difficulty of generative modeling is generating new samples instead of reproducing samples given in the training dataset. Deciding if a given sample is part of the overall distribution and mimicking this sample is a much simpler task than modeling the distribution itself so that new samples can be drawn.

The ability to generate new data from a learned distribution has long been sought after by the community of information science. Classical methods have three main problems: they often require strong assumptions about the data, have to make severe approximations for the modeled distribution, or might rely on computationally expensive algorithms, such as Markov chain Monte Carlo [15]. Deep learning solutions for generative modeling via backpropagation do not share the same drawbacks as classical algorithms. Different kinds of generative networks have been designed and offer promising results. In the following section two popular network designs, generative adversarial networks and conditional variational autoencoders, will be explained.

2.2.2.1 Generative Adversarial Networks

Generative adversarial networks (GANs) were first proposed in [16] as a class of machine learning algorithms to generate new data given a training set. Two models are simultaneously trained and set to compete against each other. One of these networks, called the generator $\mathcal{G}$, is trained to generate the wanted data, while the other network (the discriminator $\mathcal{D}$) is trying to discriminate between real and generated data. The networks then contest against each other. The training objective of $\mathcal{G}$ is maximizing the probability of $\mathcal{D}$ making a mistake. This competition improves the performance of both networks until the generator produces samples indistinguishable from the originals. Most often the input to $\mathcal{G}$ is random noise, which the network then has to learn to map to the distribution $P(X)$.

Both networks $\mathcal{G}$ and $\mathcal{D}$ are trained alternately. The training goal is to learn the generator's distribution p_g over data $\mathbf{x}$ with a true distribution p_t. This distribution p_g is mapped from the input noise variables $\mathbf{z}$ that follow the distribution $p_z(\mathbf{z})$ through a differentiable function $\mathcal{G}(\mathbf{z};\theta_g)$. $\mathcal{G}$ is represented by a neural network with parameters θ_g. The second network $\mathcal{D}(\mathbf{x};\theta_d)$ is trained to differentiate between samples from p_g and p_t. The output of $\mathcal{D}$ is a single number, which represents the probability of $\mathbf{x}$ originating from p_t instead of p_g. The training objective is the following minimax function defined with a value function $V(\mathcal{G},\mathcal{D})$:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D},\mathcal{G}) = \mathbb{E}_{\mathbf{x}\sim p_t(\mathbf{x})}[\log \mathcal{D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-\mathcal{D}(\mathcal{G}(\mathbf{z})))] \tag{2.6}$$

In figure 2.4 the structure of a GAN is shown. The input noise variables $\mathbf{z}$ drawn from p_z are fed into the generator, which outputs samples drawn from p_g. These samples together with real training samples drawn from p_t are then used as an input to the discriminator. A variation of the classical GAN is the conditional GAN (CGAN) shown in red. Both the generator and the discriminator of the CGAN receive an additional input c, which defines the mode to be generated by the networks. This can for example be one digit of the MNIST dataset or particle energies or momenta.

In [16], the authors show that given the training criterion equation (2.6), the generator $\mathcal{G}$ can learn the optimal result $p_g = p_t$ if both $\mathcal{G}$ and $\mathcal{D}$ have enough capacity. While this result is theoretically possible, optimizing equation (2.6) means finding a Nash equilibrium of a non-convex game with continuous, high-dimensional parameters [17]. As $\mathcal{G}$ and $\mathcal{D}$ are typically trained by finding a minimum of a cost function using backpropagation instead of

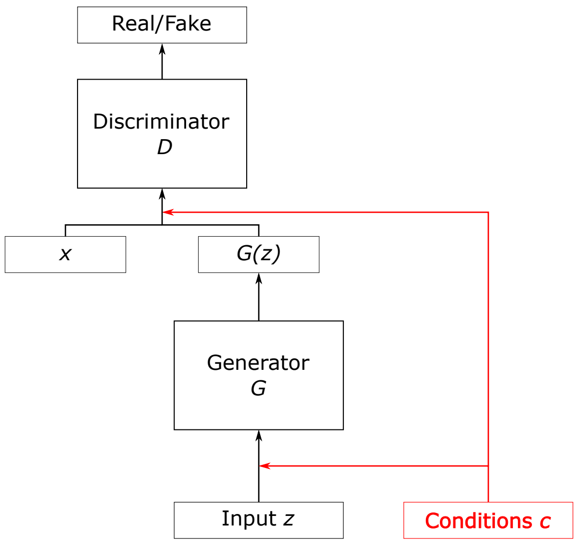

Figure 2.4: Model of a generative adversarial autoencoder. A GAN consists of two networks, the generator and the discriminator. The modules in black show the structure of a standard GAN while the red modules are the additions needed for a conditional GAN.

directly searching for the Nash equilibrium, the training process may, and often does, fail to converge.

One reason why the network may not learn the optimal result is the discriminator loss converging too quickly. In [18] the authors show that p_t and p_g are both concentrated on a low dimensional manifold, although the space they represent is often high-dimensional. The distributions p_t and p_g can therefore be disjoint and may have no overlap. This leads to a nearly trivial discriminator with a quickly converging gradient that can distinguish real samples from fake samples with perfect accuracy. As the discriminator's gradient is used for updates to the generator, the learning process for the generator is terminated as well.

Another problem of GANs is the so-called mode collapse. This happens when the generator fails to reproduce the entire distribution p_t but instead collapses on a subset of it. The generator maps several different input values to the same output point, leading to low diversity in the generated samples [16].

Several features and training techniques have been proposed to improve and simplify the training of GANs. Those methods include feature matching, which sets a new objective for the generator which minimizes the statistical difference between the features of the real and generated images; minibatch discrimination, where real and generated images are fed into the generator separately to avoid mode collapse; or one-sided label smoothing, which penalizes the discriminator if the prediction for real images goes beyond a certain probability. A detailed explanation and further methods are discussed in [17].

An extension to the standard GAN is the Wasserstein GAN (WGAN) ([19]). The WGAN can improve the stability of learning, reduce mode collapse and provide meaningful learning curves. In a WGAN, instead of the discriminator predicting if generated data is "real“, the discriminator works as a critic model deciding on the ”realness“ of the generated data. If such a critic neural network is trained, it can approximate the Wasserstein or Earth-Mover’s distance and therefore more effectively train the generator model. Unlike a discriminator model, a critic model trained to approximate the Wasserstein distance reliable provides a linear gradient even after being well trained. This leads to a more stable training process.

Several changes have to made to a GAN implementation in order to build a WGAN. The output layer of the critic model needs a linear activation function and the critic model weights need to be constrained to a limited range after each mini batch update. Weight clipping is a way to enforce a Lipschitz constraint, which is needed for the convergence of the critic. For both the critic and the generator, the Wasserstein loss should be used. For a better training process, the critic should additionally be trained more times than the generator each iteration.

2.2.2.2 Conditional Variational Autoencoders

Conditional Variational Autoencoders (CVAEs) are a different class of generative machine learning algorithms. Unlike GANs, a CVAE only consists of one network receiving training data as input and directly generating new data. CVAEs received their name from classical autoencoders which are used to learn a representation of a dataset, typically in a reduction in dimensionality. Because CVAEs also have an encoder and a decoder, the structure of the network is similar to that of an autoencoder even though they do not share the same mathematical basis. The following derivation is adapted from [15], where a more detailed explanation can be found. The theory of a conditional variational autoencoder is based on a simple variational autoencoder (VAE) which will be explained first.

VAEs rely on the use of latent variables. To have an accurate representation of the distribution of our dataset the model should have a set of latent variables z, distributed according to the probability density function $P(z)$ which helps the model generate samples from the wanted distribution. If our data points X are samples of a high-dimensional space $\mathcal{X}$, then there exist deterministic functions $f(z;\theta)$, parametrized by a vector θ in space Θ, that map $f : \mathcal{Z} \times \Theta \rightarrow \mathcal{X}$. If z is a random variable and θ is fixed, then $f(z;\theta)$ is a random variable in the space $\mathcal{X}$. θ can then be optimized so that $f(z;\theta)$ (if z is sampled from $P(z)$) is similar to the original data points X. The optimization can then be defined as

$$\mathrm{P}(X) = \int \mathrm{P}(X|z;\theta)\mathrm{P}(z)dz. \tag{2.7}$$

The goal of a VAE is finding a solution of equation (2.7). The main difficulty lies in the definition of the latent variables as well as in solving the integral over z. VAEs solve the first problem by not making assumptions about the dimensionality of z. Any distribution in d dimensions can be mapped by using d variables distributed normally and transforming them, as per the inverse transform sampling method [20]. Since any continuous function can be accurately represented by a neural network with a sufficient amount of parameters, the transformation from the normal to the wanted distribution can be done through a network. If such a function exists, then with enough training the network will be able to approximate it. Therefore, the optimization problem reduces to maximizing equation (2.7) with $P(z) = \mathcal{N}(z|0, I)$.

One way to compute $P(x)$ approximately is sampling a large number of z values $\{z_1, ..., z_n\}$ and reducing the integral into a sum, so that $P(X) \approx \frac{1}{n}\sum_i P(X|z_i)$. Sampling enough values to have an accurate representation of $P(x)$ in high dimensional spaces is, however, often not feasible. Variational autoencoders, therefore, try to change the sampling procedure by only sampling those z values that produced samples more similar to X and approximate $P(X)$ from those samples. There should be a function $Q(z|X)$ which takes a value of X and can return a distribution of z values that are more likely to produce X. As this subset is a lot smaller than the original space, computing for example $E_{z \sim Q}P(X|z)$ is possible.

To relate $E_{z \sim Q}P(X|z)$ and $P(X)$, we use the Kullback-Leibler divergence $KL_{\text{div}}($ [21]) between between $P(z|X)$ and $Q(z)$ for an arbitrary Q.

$$KL_{\text{div}}[Q(z) \| P(z|X)] = E_{z \sim Q}[\log Q(z) - \log P(z|X)] \quad . \tag{2.8}$$

Here we can use the Bayes' theorem for conditional probabilities to insert $P(X|z)$ and $P(X)$ into the equation. As $\log P(x)$ does not depend on z, we can put it outside of the expectation value. If we reorder the equation and substitute equation (2.8), the equation yields:

$$\log P(X) - KL_{\text{div}}[Q(z) \| P(z|X)] = E_{z \sim Q}[\log P(X|z)] - KL_{\text{div}}[Q(z) \| P(z)] \quad . \tag{2.9}$$

Up until here, Q could have been any distribution. As Q should depend on X and minimize $KL_{\text{div}}[Q(z) \| P(z|X)]$, we can insert $Q(z|X)$:

$$\log P(X) - KL_{\text{div}}[Q(z|X) \| P(z|X)] = E_{z \sim Q}[\log P(X|z)] - KL_{\text{div}}[Q(z|X) \| P(z)] \quad . \tag{2.10}$$

In equation (2.10) the left side shows $\log P(X)$ including an error term which leads Q to produce useful z-values. This is the quantity we want to maximize. The right side of the equation can be optimized with the help of stochastic gradient descent. This equation is similar to that of an autoencoder: Q is encoding X into z whereas P is decoding it to reconstruct X.

To perform stochastic gradient descent on the right side of equation (2.10), $Q(z|X)$ has to be defined. A standard choice is

$$Q(z|X) = \mathcal{N}(z|\mu(X;\theta), \Sigma(X;\theta)) \quad , \tag{2.11}$$

with μ and Σ being arbitrary deterministic functions with θ learned from data. Σ is constrained to be a diagonal matrix, which has the computational advantage of simplifying equation (2.10). We can now define $KL_{\text{div}}[Q(z) \| P(z)]$ as a KL-divergence between two multivariate Gaussian distributions. We also directly insert $P(z) = \mathcal{N}(z|0, I)$ so that

$$KL_{\text{div}}[\mathcal{N}(\mu(X), \Sigma(X)) || \mathcal{N}(0,1)] = \frac{1}{2}\left(\text{Tr}\left(\Sigma(X)\right) + (\mu(X))^\top (\mu(X)) - k - \log\det\left(\Sigma(X)\right)\right) \tag{2.12}$$

For the first term in the right hand side of equation (2.10) we approximate the expectation value by taking one sample of z and using $\log P(X|z)$ as an approximation. The exact

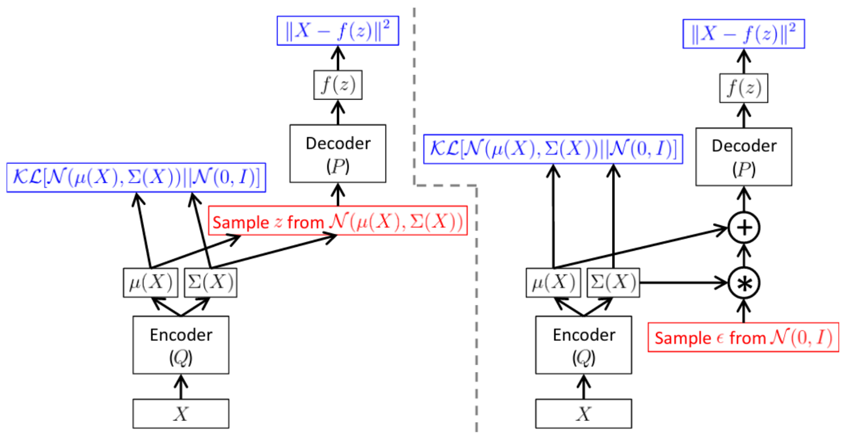
$\|X - f(z)\|^2$
$f(z)$
Decoder (P)
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)]$
Sample z from $\mathcal{N}(\mu(X), \Sigma(X))$
$\mu(X)$
$\Sigma(X)$
Encoder (Q)
X
$\|X - f(z)\|^2$
$f(z)$
Decoder (P)
$\mathcal{KL}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)]$
$\mu(X)$
$\Sigma(X)$
Sample ϵ from $\mathcal{N}(0, I)$
Encoder (Q)
X

Figure 2.5: Model of a variational autoencoder. Loss functions are shown in blue, non-differentiable procedures are marked in red. Both networks are feedforward networks, but only the right network can be trained with backpropagation. Figure is taken from [15].

equation we want to optimize while performing stochastic gradient descent over different values of X samples from a dataset D is

$$E_{X\sim D}[E_{z\sim Q}[\log P(X|z)] - KL_{\text{div}}[Q(z|X)\|P(z)]]. \tag{2.13}$$

The gradient can be performed inside the expectation values, so we compute the gradient of

$$\log P(X|z) - KL_{\text{div}}[Q(z|X)\|P(z)], \tag{2.14}$$

with one sample of X and one sample of z.

In equation (2.14) there is no dependency of the first term $\log P(X|z)$ on Q. This network can be visualised by the network in the left of figure 2.5. The sampling procedure of sampling z from $Q(z|X)$ is a non-continuous operation and therefore has no gradient. Because backpropagation has to be performed throughout the whole network, the sampling is moved to an input layer. This is called the “reparametrization trick” and is shown on the right side of figure 2.5. The final equation on which the gradient is computed is then

$$E_{X\sim D}\left[E_{\epsilon\sim\mathcal{N}(0,I)}[\log P(X|z = \mu(X) + \Sigma^{\frac{1}{2}}(x)\cdot\epsilon)] - KL_{\text{div}}[Q(z|X)||P(z)]\right]. \tag{2.15}$$

In many applications the sample space $\mathcal{X}$ is multimodal, which is a difficult distribution for a VAE to learn out of finite samples. For that case, conditional variational autoencoders have an extra input variable Y. This input determines the mode that the network is supposed to generate. In figure 2.6 the structure of a CVAE at training (left) and test time (right) is shown. The input variable Y can for example be the category of a CIFAR dataset or the energy and momentum of a simulated particle.

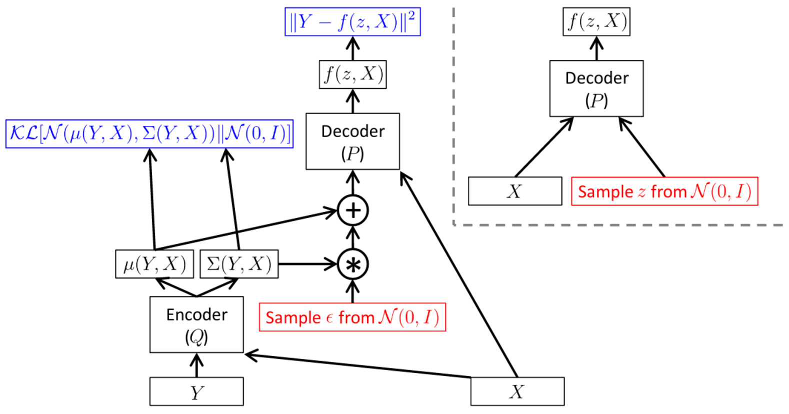
$\|Y - f(z,X)\|^2$
$f(z,X)$
$\mathcal{KL}[\mathcal{N}(\mu(Y,X),\Sigma(Y,X))\|\mathcal{N}(0,I)]$
Decoder (P)
$\mu(Y,X)$
$\Sigma(Y,X)$
Encoder (Q)
Sample ϵ from $\mathcal{N}(0,I)$
Y
X
$f(z,X)$
Decoder (P)
X
Sample z from $\mathcal{N}(0,I)$

Figure 2.6: Model of a conditional variational autoencoder. On the left is the training procedure, with a sample input X and the input variable Y. On the right side, the same model is shown at test time. The figure is taken from [15].

2.3 Probability Metrics and Two-Sample Tests

Two-sample tests are a method used in statistical hypothesis testing to decide whether two samples are drawn from the same distribution. To decide whether or not that hypothesis is true, a test statistic t is usually applied on the data and returns a usually scalar variable. If t lies in the before defined critical region, the hypothesis is rejected. [22]

As probability metrics measure the distance between random quantities, these metrics can be used as a test statistic in two-sample tests. Data drawn from two probability distributions can be compared through an appropriate probability metric. While one-dimensional test statistics are widely used in two-sample testing, reducing high-dimensional data to one dimension results in a loss of information and therefore in a worse test performance. One popular high-dimensional probability metric is the Kullback-Leibler (KL) divergence ([21]), which is explained in the following chapter. During these thesis, a fast computation of a high-dimensional ddKS metric was developed ([23]) which outperforms the KL divergence in high dimensions and for non-symmetric distributions. This metric is explained in section 2.3.2.

2.3.1 Kullback-Leibler Divergence

The Kullback-Leiber divergence, also called relative entropy, is a probability metric that can be applied in a two-sample test. For a two-sample test, two samples are drawn respectively from P and Q, which are two probability distributions defined on the same probability space X. The two-sample test then tests the hypothesis of $P = Q$ through the calculation of the KL divergence KL_{div}between both samples. This is calculated by

$$KL_{\text{div}}(P||Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \quad . \tag{2.16}$$

This equation only holds if $Q(x) = 0$ implies $P(x) = 0$. For $P(x) = 0$, the contribution of the corresponding term is also set to 0. While the KL divergence is non-negative, it is neither symmetric nor does it satisfy the triangle equality. It is therefore not a distance metric. [21]

The KL divergence is applicable in high dimensions and is widely used in machine learning and statistical tests. [24]

2.3.2 D-Dimensional Kolmogorov-Smirnov Metric

The main drawbacks of common probability metrics are firstly the need for assumptions about the underlying data, such as assuming a Gaussian distribution, and secondly the difficulty expanding to higher dimensions.

One widely used one-dimensional metric is the Kolmogorov-Smirnov (KS) metric [25] [26], which has several advantages over other metrics such as the χ^2 test or the Earth Mover's metric. The KS test, where the KS metric is applied in a two-sample test, uses the maximum difference between the two cumulative distribution functions as a distance. Besides being nonparametric, the KS test is sensitive to difference in both location and shape of the two tested probability distributions. Additionally, the KS test is applicable on samples with a small number of data points, unlike for example the Ξ^2 test.

The generalization of the one-dimensional KS test is a d-dimensional KS test (ddKS) based on the two-dimensional version of Fasano and Franceschini [27]. The difficulty of extending KS to higher dimensions is that the calculation of differences between higher-dimensional CDFs is ambiguous.

For a two-sample test, there are two samples drawn from probability distributions in $\mathbb{R}$, denoted P and T here. The principle of calculating the KS test statistic between both samples remains the same in ddKS, calculating the difference between the two CDFs of P and T. For this distance calculation, the d-dimensional space is separated into 2^d orthants respective to a given point. In figure 2.7 this separation is shown in the example of three dimensions.

The cumulative distribution function estimate is then the number of points positioned inside one orthant. Given these two estimates $C_P(\vec{x})$ and $C_T(\vec{x})$, he ddKS test statistic is simply the maximum distance between those.

$$D = \max |C_P(\vec{x}) - C_T(\vec{x})| \tag{2.17}$$

As estimated CDFs do not change except at points existing in the respective sample, the evaluation of equation (2.17) is only done at points in either sample. Therefore, both C_P and C_T do not need to be continuous functions, but can be treated as tensors with the shape $\mathbb{C} \in \mathbb{R}^m \times 2^d$, where m is the number of evaluation points. As m can denote either the points of sample T or P, these cumulative distribution tensors are named $C_{P,m}$ and $C_{T,m}$. As using either sample is valid for calculating these tensors, but leads to varying results, ddKS is defined as the maximum absolute difference between CDFs, averaged over both evaluation point samples.

$$D = \frac{\max |C_{P,P}(\vec{x}) - C_{T,P}(\vec{x})| + \max |C_{P,T}(\vec{x}) - C_{T,T}(\vec{x})|}{2} \tag{2.18}$$

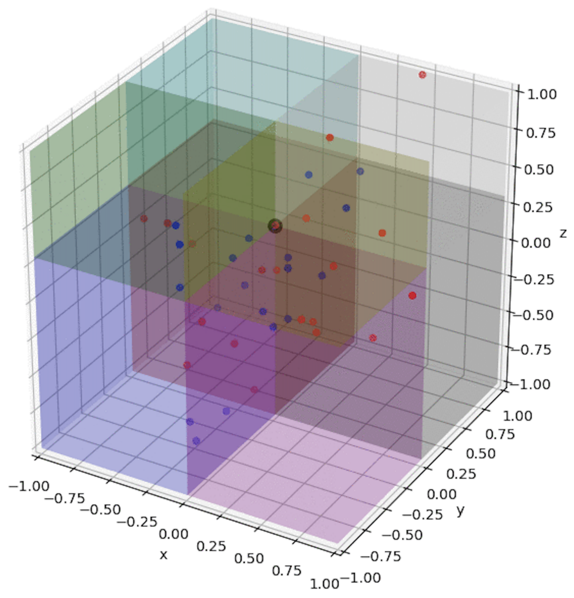

Figure 2.7: Separation of a three-dimensional space into octants respective to a given point. The point marked black is the origin of the octant separation. The blue and red points are samples drawn from different distributions. The number of points per octant is the cumulative distribution estimation. This figure is taken from [28].

This can be implemented as a loop based calculation. An iteration is done for both samples P and T, looping over every point in each respective sample. For every point, the orthants are defined and the number of points of each sample located in every orthant is stored. For each test point this orthant membership number is stored for both samples. For both sets of test points, the maximum average difference between orthant membership for P and T is calculated and averaged over both sets to achieve the result in equation (2.18).
The computational complexity for this method is $\mathcal{O}(2^d N^2)$ with N as the size of the largest sample P or T. As N is often large in applications where probability metrics are needed, an accelerated computation has been developed.

The accelerated computation is based on using tensor primitives from the `pytorch` library. This ensures that implicit parallelism can be used for small N, thus reducing time complexity in this case to $\mathcal{O}(1)$ as well as enabling calculation on GPU. For larger N, the accelerated computation trades time complexity for memory complexity, which is not a bottleneck on modern computational devices. The accelerated computation has been published in [23].

For this method we first construct tensors P and T with dimensions $N_P \times d$ and $N_T \times d$, respectively. $N_{P(T)}$ is the number of points in the sample, while d denotes the number of dimensions. Both tensors are then expanded along a new axis, copying all elements in the first two axes, creating new tensors $\mathcal{P}$ and $\mathcal{T}$, being $N_{P(T)} \times d \times N_{P(T)}$ dimensional. We then clone $\mathcal{P}$ and $\mathcal{T}$ and permute their first and third axis. These new tensors we call $\mathcal{Q}_P$ and $\mathcal{Q}_T$. With this permutation $\mathcal{P}[i, j, k]$ is the jth element of the ith point while $\mathcal{Q}_P[i, j, k]$ is the jth element of the kth point. With these tensors we first do an element-wise comparison of each point in P and T

$$\mathcal{G}_{P,P} = \mathcal{P} \geq \mathcal{Q}_P \quad \mathcal{G}_{T,P} = \mathcal{T} \geq \mathcal{Q}_P \quad \mathcal{G}_{P,T} = \mathcal{P} \geq \mathcal{Q}_T \quad \mathcal{G}_{T,T} = \mathcal{T} \geq \mathcal{Q}_T\,. \tag{2.19}$$

Next we define a square wave function $S(x, f)$, which is needed to sort points into their respective orthants.

$$S(x, f) = \frac{(-1)^{\lfloor 2fx \rfloor} + 1}{2} \tag{2.20}$$

S varies between 0 and 1 with a given frequency f. The membership tensor can then be defined through the element-wise comparisons $\mathcal{G}$ and the square wave function.

$$\mathcal{M}_{X,X}[i, l] = \sum_{k=1}^{N} \prod_{j=1}^{d} S\left(\mathcal{G}_1\left[i, j, k\right], \frac{l}{2^{j-1}}\right) \tag{2.21}$$

The frequency of S is set by the orthant $l \in [0, 2^d - 1]$ which we are currently evaluating, while the evaluated point x is given by $\mathcal{G}$. After constructing the membership tensor, we can calculate the difference between the membership tensors belonging to the samples P and T and finding the maximum. To fully recreate equation (2.18), we average over the distance between membership tensors evaluated on points of sample P and the distance evaluated on points of sample T:

$$D = \frac{\max|\mathcal{M}_{P,P}(\vec{x}) - \mathcal{M}_{T,P}(\vec{x})| + \max|\mathcal{M}_{P,T}(\vec{x}) - \mathcal{M}_{T,T}(\vec{x})|}{2}\,. \tag{2.22}$$

Using this implicit parallelization, this tensor based method has a time complexity of $\mathcal{O}(1)$ for small N and d. The restriction is the memory of the processing unit used, which for modern GPUs can store up to 10,000 points per sample in three dimensions.

Additionally, two methods to accelerate the computation of the ddKS have been developed as well. The accelerated methods approximate ddKS using voxelization- and sorting-based methods, using a tradeoff between speed and statistical efficiency. All three methods can be found in [23] together with an analytic significance calculation.

3. Experimental Setup

3.1 The Belle II experiment

The Belle II detector is a particle physics experiment located at the SuperKEKB collider in Tsukuba, Japan. The SuperKEKB collider is an e^+e^--collider designed to operate at the $\Upsilon(4S)$ resonance, which further decays into a B meson pair (see figure 3.3). It is an upgrade of the original KEKB accelerator and recorded first collisions in April 2018. This chapter is based on the Belle II technical report ([29]). The accelerator consists of two storage rings, one for the high-energy electron beam and one for the low-energy positron beam. Four experimental halls are positioned at every straight point in the accelerator ring, which has a circumference of 3016 m. To study decay channels of B mesons the Belle II detector is build around the only collision point to track the decay particles and measure their properties. In figure 3.1 a schematic overview of the SuperKEKB accelerator is shown.

The Belle II detector consists of several sub-detector parts to maximize the information measured during the decay process, which can be seen in figure 3.2. Centered around the collision point are two layers of pixelated silicon sensors (PXD) and four layers of double-sided silicon strip sensors (SVD). Those sensors are able to measure the decay vertex positions of mainly B mesons but also of other particles that may be daughters of the collision resonance. Further away from the collision point is the central drift chamber (CDC). The CDC is a cylindrical wire chamber to track trajectories of charged particles. Furthermore, it is also able to measure the momenta and dE/dx information of those particles.
For the purpose of particle identification, the Time-of-Propagation (TOP) detector is located in the barrel. It consists of 16 quartz bars positioned circular around the collision point where each bar has a reflecting mirror on one side and an array of photomultiplier tubes on the other. The TOP detector measures the position and propagation time of Cherenkov photons propagating inside the TOP, which were created by particles passing through the quartz bar. As this detector part is the focus of this thesis, its exact function will be explained later (section 3.1.1). Another detector part used for particle identification is the aerogel ring imaging Cherenkov (ARICH) counter in the forward region of the detector.
To measure the total energy of charged particles the electromagnetic calorimeter (ECL) comprised of scintillator crystals is positioned just inside the superconducting solenoid coil. The main tasks of the ECL are photon detection and exact determination of the photon energy and angular coordinates as well as the separation between electrons and hadrons.

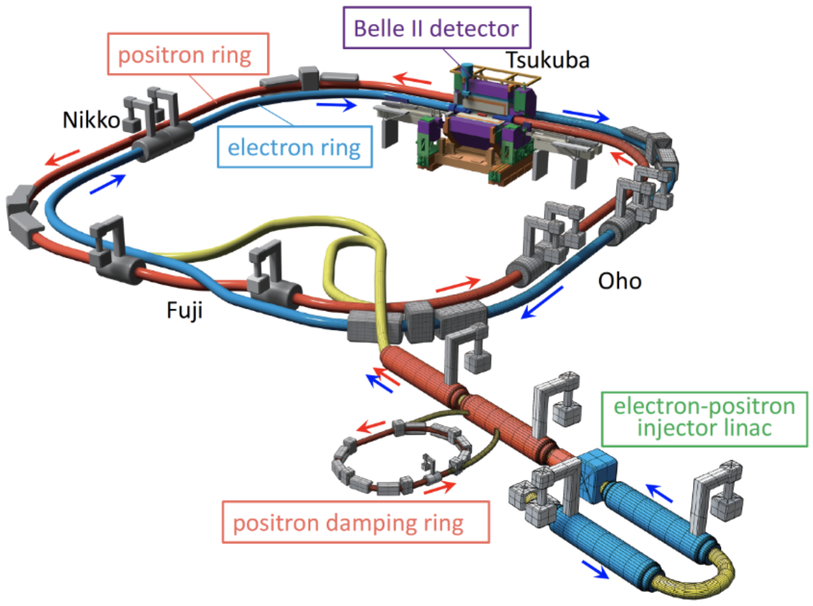

Figure 3.1: The SuperKEKB accelerator as a schematic view. The four straight sections with the respective experimental halls are called Tsukuba, Oho, Fuji, and Nikko. The Belle II detector is located at the interaction point in the Tsukuba section. The figure is taken from [30].

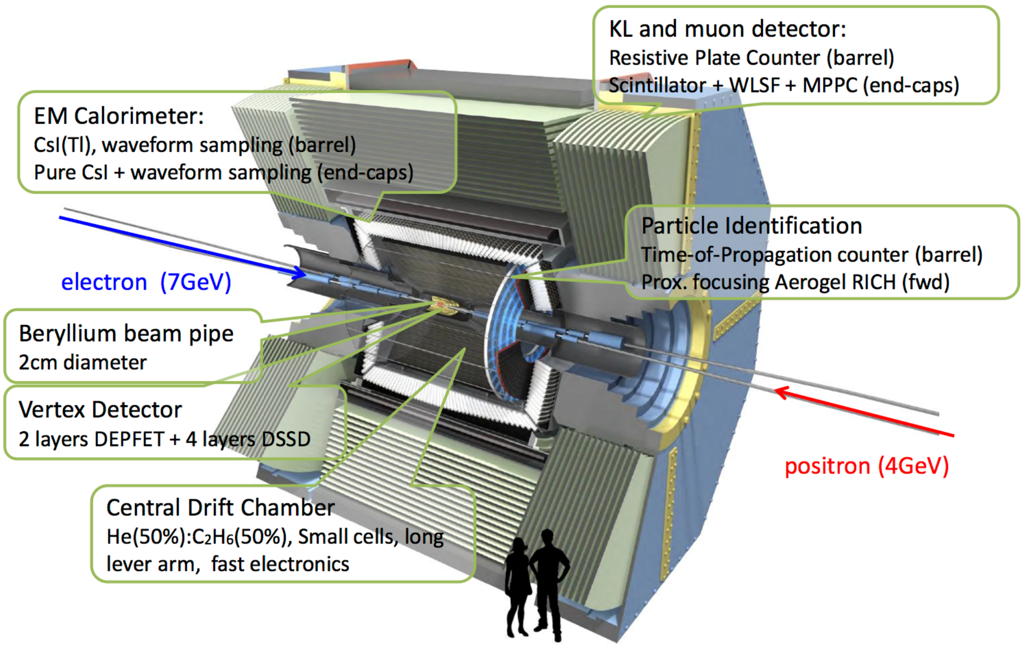
KL and muon detector:
Resistive Plate Counter (barrel)
Scintillator + WLSF + MPPC (end-caps)
EM Calorimeter:
CsI(Tl), waveform sampling (barrel)
Pure CsI + waveform sampling (end-caps)
Particle Identification
Time-of-Propagation counter (barrel)
Prox. focusing Aerogel RICH (fwd)
electron (7GeV)
Beryllium beam pipe
2cm diameter
Vertex Detector
2 layers DEPFET + 4 layers DSSD
positron (4GeV)
Central Drift Chamber
He(50%):C2H6(50%), Small cells, long lever arm, fast electronics

Figure 3.2: An overview of the Belle II detector. The detector separated into subdetector parts is centered around the collision point. The figure is taken from [31].

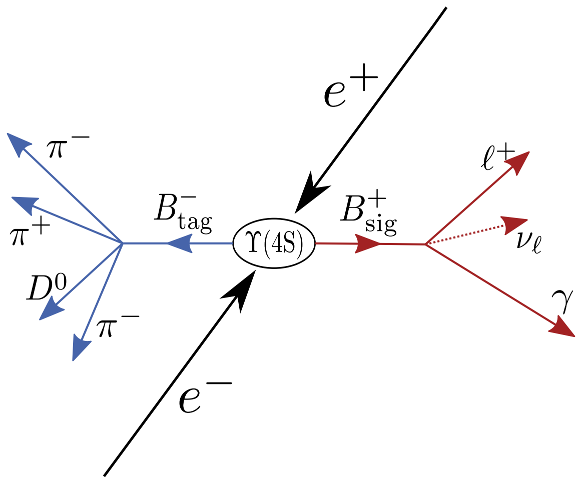

Figure 3.3: The decay of the $\Upsilon(4S)$ resonance into two B mesons. At the collision point electrons and positrons collide at the $\Upsilon(4S)$ resonance to produce B mesons. Those B mesons then further decay into leptonic and hadronic particles which can be measured by the detector.

Because of this, the ECL is also a major part of the particle identification process. The superconducting solenoid coil generates a magnetic field of 1.5 T inside the detector. This is necessary for the function of the CDC as particles traveling through a magnetic field have a curved trajectory dependent on their momentum. With a constant magnetic field strength the radius of the trajectory then directly translates to the momentum. Outside of the magnetic coil, the outermost sub-detector is the K^0_{L} and muon detector (KLM). It serves as an absorber to decelerate particles as well as the magnetic flux return yoke of the magnet. As muons as well as K^0_{L} are very long lived, the KLM consists of thick metal plates alternating with active detector parts to provide > 3.9 hadronic interaction lengths λ_0.

The Belle II experiment has been an upgrade to the previous Belle experiment and has started taking data in April 2017. Belle II is expected to collect a dataset of $50\,\mathrm{ab}^{-1}$ which is about 50 times larger than the dataset collected by the Belle experiment. This is mostly because of the accelerator upgrade to the SuperKEKB collider which has an over 40 times larger instantaneous luminosity compared to the KEKB collider which was operating during the Belle experiment.

3.1.1 The TOP Detector

In the Belle II experiment, the TOP detector together with the ARICH detector is used for particle identification purposes. This chapter is based on [32], where further information to the TOP detector can be found.

The detector consists of 16 quartz bars centered around the collision point (see figure 3.4). Every quartz bar is built out of two separate bars glued together. To contain the majority of photons inside the bar, a mirror is positioned at the forward end of every bar. The backwards end has a small expansion prism to maximize the photon detection surface. An array of photomultipliers (PMTs) measures the photon hits on the backward surface of the

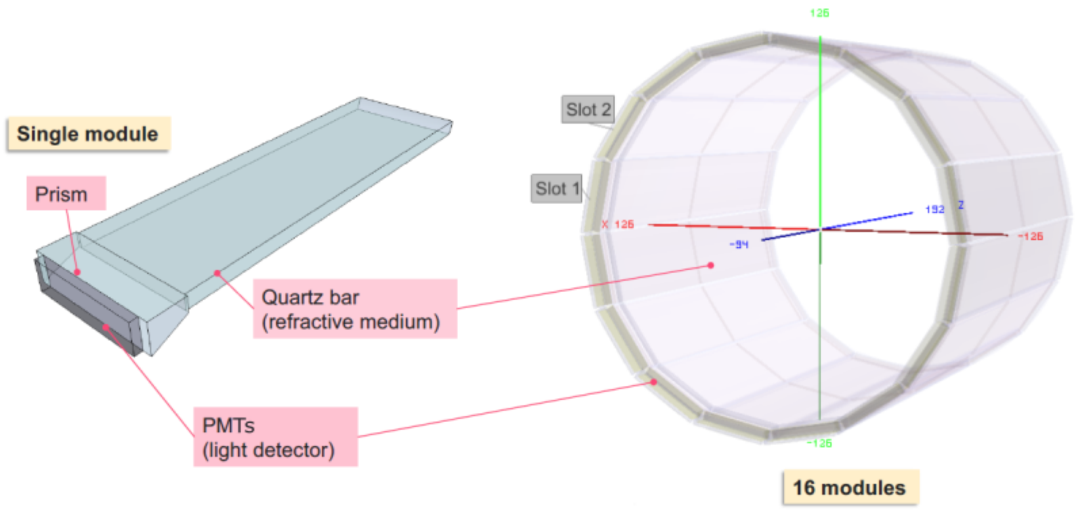

Figure 3.4: An overview of the TOP detector and the PMTs. 16 identical quartz modules, are centered around the beamline and the collision point. Every module consists of two quartz bars combined by a glue layer. On the forward end of the module a reflective mirror is positioned. The backward end consists of an additional prism and the PMts to maximize photon detection.

prism. 32 16-channel microchannel plate photomultiplier tubes (MCP-PMTs) can readout the photon hits with 512 MCP-PMT pixels in total. Each quartz bar is 125 cm long, 45 cm wide and 2 cm high. The prism has an additional length of 10 cm and extends 3.5 cm below the bar.

Particles crossing the TOP detector emit Cherenkov light when their speed is greater than the phase velocity of the crossed medium. The angle and energy of the emitted photons depends on the refractive index of the crossed medium and on the momentum of the charged particle. The emission angle θ can be calculated by

$$\cos\theta = \frac{1}{\beta n} \quad \text{with} \quad \beta = \frac{v_p}{c} \tag{3.1}$$

where n is the refractive index and v_p the speed of the charged particle. The emitted photons propagate through the detector, while either being reflected or absorbed by the walls of the quartz bar, if they collide with it. This propagation process is shown in figure 3.5. Each reflection adds a measure of uncertainty to the photon track, which leads to a probability distribution of detection values. Photons that reach the PMTs are then used for particle identification.

This is done by measuring the x, y and t coordinate of the emitted photons. The spatial coordinates x and y are the hit positions of the photons on the PMTs, while t is the propagation time from emission to measurement. In figure 3.6 an example of hit positions of photons in the TOP detector in the x-y plane can be seen. For each particle, about 15-30 photons are measured by the TOP detector, with additional photons from beam

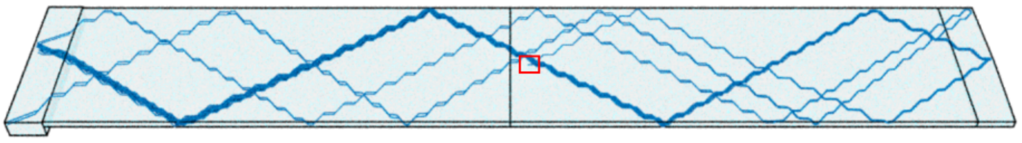

Figure 3.5: The propagation of photons inside a TOP module. A generated photon gets propagated through the detector by the simulation software. At every collision point the new direction is calculated until it either escapes the detector or is detected by the PMT array.

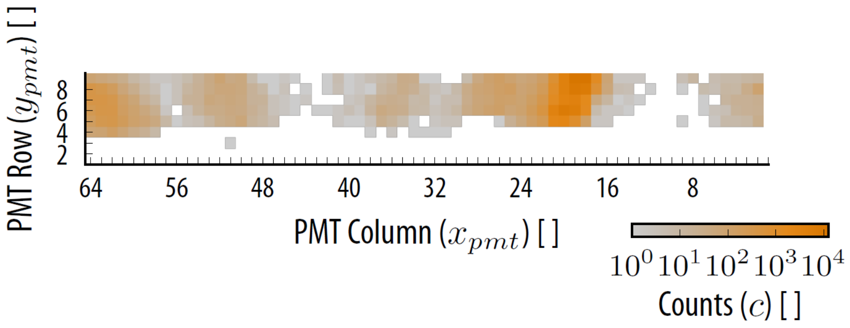

Figure 3.6: Photon hit distribution for the PMTs in the x-y plane. The hits are generated by several tracks and show the possible distribution of photon detection values.

background. The arrival time and hit position distributions for each event are compared to analytical probability density functions through a likelihood analysis for each particle hypothesis (e, μ, π, K, p).

The simulation of the TOP detector currently consumes the largest portion of the Belle II simulation CPU time budget, as shown in figure 3.7. This is mostly because of the tracking of photons through the detector. Each photon is propagated through the detector until a wall collision, where it is either reflected or absorbed. In the case of reflection, this process gets repeated until the photon reaches the PMTs. Replacing this simulation would lead to large savings in the overall computing budget.

3.2 The Belle II Analysis Framework

As the Belle II experiment creates large amounts of data that need to be analyzed, reconstructed or visualized, additionally to simulating new data, the Belle II collaboration uses the Belle II Analysis Software Framework *basf2* to provide tools of working with data. The code is written in C++ and the framework dynamically loads a series of modules to process events independently. The user can select, configure and chose the execution order of these modules through a Python interface. The data is exchanged by the modules via the *DataStore*, a globally shared storage. Typically, a user performs a task via a "steering file",

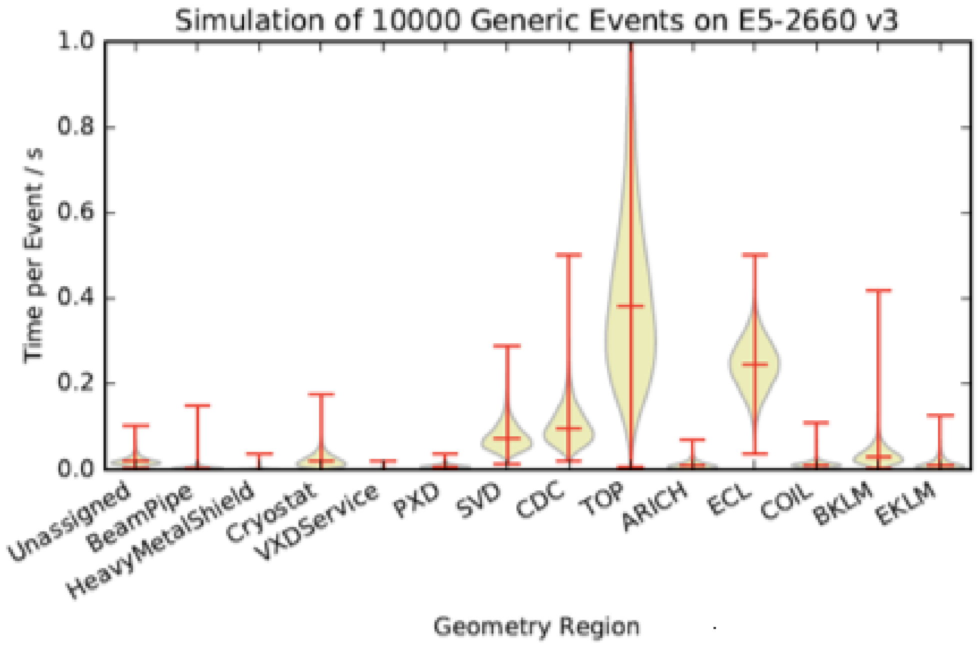

Figure 3.7: The simulation time of detector modules in Geant4. The simulation of the TOP detector is currently the biggest bottleneck in regards to simulation time. This plot is taken from [33].

which is a Python script defining and arranging the wanted modules and then starting the event processing. In addition to the Belle II specific code, the core software also contains *externals*, third-party code needed for the full analysis done by *basf2*. The *externals* contain basic tools like GCC, Python 3 etc as well as specific software such as ROOT ([34]) and Geant4, which is explained in the following section. [35]

This framework ensures a fast processing of data, while providing a simple user interface for easy and adaptable workflows.

3.3 Geant4

Geant4, which stands for GEometry ANd Tracking, is a toolkit for the simulation of the interaction of particles with matter. It is used by many experiments in high energy physics, astrophysics, medical physics and more. It is designed to handle complex geometries dependent on the experiment and to enable an easy adaption for different applications. It can correctly handle geometry, tracking, and detector response while also offering options for run management, visualization and a user interface. The physics processes can handle electromagnetic, hadronic and optical processes over a wide energy range. It is implemented in the C++ language and was the first to use object-oriented programming. The international Geant4 collaboration takes care of development, maintenance and user support and continuously implements improvements to accommodate the needs of the users and to utilize new technology. The Belle II experiment also utilizes Geant4 as an external library to model the passage of particles through matter. [36]

4. Validation Framework

Validation mechanisms of generative models highly depend on the field in which the generative model is used. As one of the most prominent applications of GANs and other neural network algorithms for generative modeling is the creation of pictures, oftentimes human recognition is used as a validation metric. The ability to recognize real people or animals on pictures and to spot mistakes made by a generative model is highly developed in humans and widely used as a decision mechanism.

For a quantitative measurement of the quality of generated samples, standard mathematical metrics have been applied or new metrics have been developed. One widely used metric is the KL divergence, which has been explained in section 2.3.1. For the generation of pictures, a widespread metric is the Inception Score ([17]) or adaptive metrics, such as the Frechét Inception Distance ([37]). Those metrics are based on the similarity of outputs of different layers in the Inception network, which is one of the best performing image classification networks.

For applications of generative modeling in physics or other natural sciences human recognition is not a good metric to be used. Humans are not able to evaluate statistical processes or pick up slight differences in event variables or shower shapes for example. Furthermore, most quantitative validation methods either only apply for the generation of pictures or are computationally expensive to compute in higher dimensions and with large amounts of data. Physical problems need to be broken down to easy to compare high level variables, that still represent the underlying physical mechanism. This choice of high level variable can be difficult. Therefore the need for a generalized method to validate trained generative models arises with the widespread use of those models.

A validation framework should be able to confirm that the necessary underlying physics are correctly reproduced by the model. This confirmation should be done in a way that covers most of the phasespace of the physics problem modeled as well as being quickly and computationally cheap calculated. The accuracy with which the standard simulation software is modeling the physics process has to be determined and set as a training limit for the generative model. Then validation techniques have to be defined and implemented in an accessible way.

I apply this concept for the problem of simulating the Time-of-Propagation detector at the Belle II experiment. As I show in section 3.1.1, the TOP detector is currently the bottleneck of simulation in terms of overall simulation time. As the majority of this simulation time is

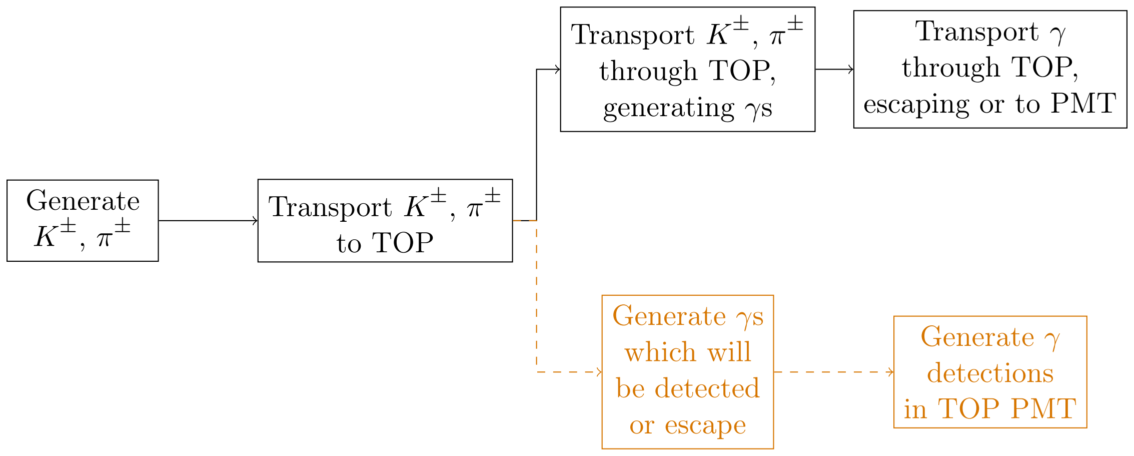
Generate $K^\pm$, $\pi^\pm$
Transport $K^\pm$, $\pi^\pm$ to TOP
Transport $K^\pm$, $\pi^\pm$ through TOP, generating γs
Transport γ through TOP, escaping or to PMT
Generate γs which will be detected or escape
Generate γ detections in TOP PMT

Figure 4.1: Flow chart of current and planned TOP simulation process. The black boxes represent the current simulation process, done through Geant4 while the copper boxes represent the planned neural network simulation. The focus of this work is on the generation of detections in the TOP PMTs. A second neural network would decide if a emitted photon is being detected by the PMTs.

used by the propagation of photons inside the detector, we want to replace this part with a fast simulation mechanism.

In figure 4.1 the concept of the replacement of parts of the current simulation procedure is shown. The current simulation done by Geant4 transports particles to the TOP detector. When they enter the detector, Cherenkov photons are produced with direction and energy according to the properties of the original particle. These photons then get transported through the TOP bar. When a photon reflects at a wall of the quartz bar, two probabilistic functions decide if it reflects or gets absorbed and the direction of the reflection. A percentage of photons then get detected at the PMTs at one end of the quartz bar. These detection values, which are given in x, y, and t, are used for the likelihood calculation. The current goal of the fast simulation is to replace the propagation part of the simulation with a neural network. This gives a clear goal of the network training, as it has to generate the correct detection values given photon origin points and momentum.

4.1 Baseline of Current Simulation

The current simulation is done by Geant4 implemented into basf2. Geant4 is a highly sophisticated simulation framework, which aims to simulate the interaction between particles and detector components. As Geant4 has a very high agreement with data, we aim for a neural network solution that agrees with the current simulation.

To figure out the baseline to reach for in a fast simulation training, the uncertainty of Geant4 has to be determined. This cannot be done analytically as two probabilistic processes are part of the photon transportation mechanism. In Geant4 the surface of the quartz bars has an applied roughness to correctly model the real detector. Therefore the reflection of a photon hitting one of the walls does not purely depend on the incidence angle of the photon but also on the exact angle of the surface in respect to a smooth surface. This angle cannot be efficiently predicted and therefore not modeled by a function. The second probabilistic process is the decision if a photon is reflected inside the detector, absorbed by the detector wall or escapes the detector fully. This is a purely stochastic function with each possibility having a certain probability depending on the angle and the energy of the photon.

These processes show that we cannot estimate a function that returns the uncertainty of photon detection values dependent on their origin coordinates and momentum. A major part of the following sections is finding a way to exactly determine these uncertainties, which would firstly give a baseline for training a neural network simulation and secondly a validation mechanism for a trained network.

4.1.1 Time Requirement

Training and validating a neural network solution for the TOP detector simulation requires labelled data on which training is performed. While training could be done on only parts of the phasespace, validation samples should cover the entire possible space to ensure correct simulation in every case.

Cherenkov photons created by crossing charged particles can originate from anywhere inside the TOP detector, which gives us the coordinate intervals $x \in [-22.5, 22.5]$, $y \in [-1.0, 1.0]$ and $z \in [-125, 125]$. The momentum direction is given by two angles, ϑ and ψ. The original momentum vector is $\mathbf{p} = (0, 0, 1)^{\mathrm{T}}$ and is rotated by rotation around the x-axis given by ϑ and a following rotation around the z-axis by ψ. The resulting vector is then given by

$$\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\vartheta & -\sin\vartheta \\ 0 & \sin\vartheta & \cos\vartheta \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad . \tag{4.1}$$

In the calculation inside basf2, a first rotation around the z-axis is defined by the angle ϕ. Since the original vector is parallel to the z-axis, this rotation does not change the value of the vector and can therefore always be set to 0. Both angles ϑ and ψ can be chosen inside the interval $[0, 360)$.

For a step size of 0.1 for the spatial coordinates and a step size of 1° for the angles, the number of points in the phasespace reaches $2.916 \cdot 10^{12}$. Smaller steps or variations in energy lead to an even bigger phasespace.

As the absorption or escape probability per photon is proportional to the number of reflections, photons with a high number of reflections are detected very rarely by the PMTs. In figure 4.2 the average detection percentage dependent on the rotation angle ϑ is shown. An angle of 180° returns a momentum vector of $(0, 0, -1)^{\mathrm{T}}$, resulting in the emission of the photon towards the PMTs. In addition,figure 4.3 shows the dependence of the average photon track length on the rotation angle ϑ. By comparing figure 4.2 and figure 4.3, the detection probability increases if the track length decreases. Photons with a shorter track length, which is given by $\vartheta \geq 135^\circ$, have a higher probability of reaching the PMTs and therefore being counted as detected. This anticorrelation shows one of the difficulties in mapping out the phasespace of the whole detector. To determine the detection value distribution of photon origin points in the phasespace with a small detection

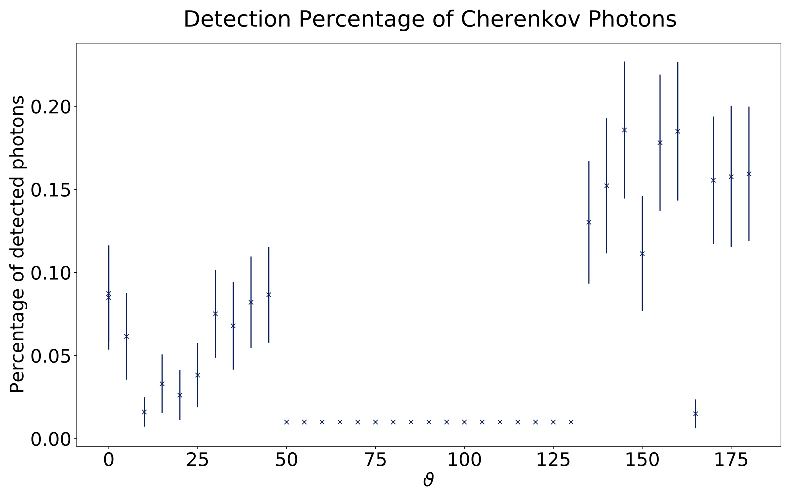

Figure 4.2: Percentage of detected photons for Cherenkov photons dependent on momentum direction. The percentage of detected photons is shown over the angle ϑ which determines the direction of the photon. An angle of 180° translates to a momentum direction pointed towards the PMTs, while an angle of 0° translates to a momentum vector pointed towards the mirror. For this percentage 200 events with 500 photons were simulated and the number of detected photons was analysed.

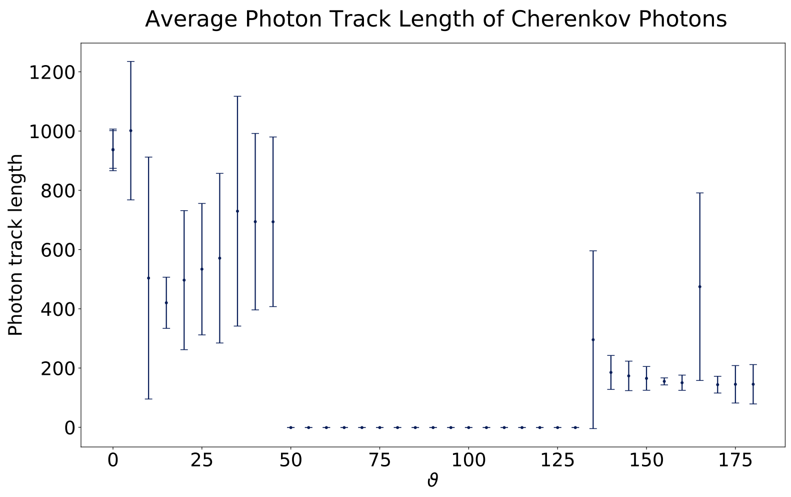

Figure 4.3: Average photon track length for Cherenkov photons dependent on momentum direction. The average track length in mm of 500 identical photons inside the TOP detector with an errorbar of 1σ is shown over the angle ϑ which determines the direction of the photon. An angle of 180° translates to a momentum direction pointed towards the PMTs, while an angle of 0° translates to a momentum vector pointed towards the mirror.

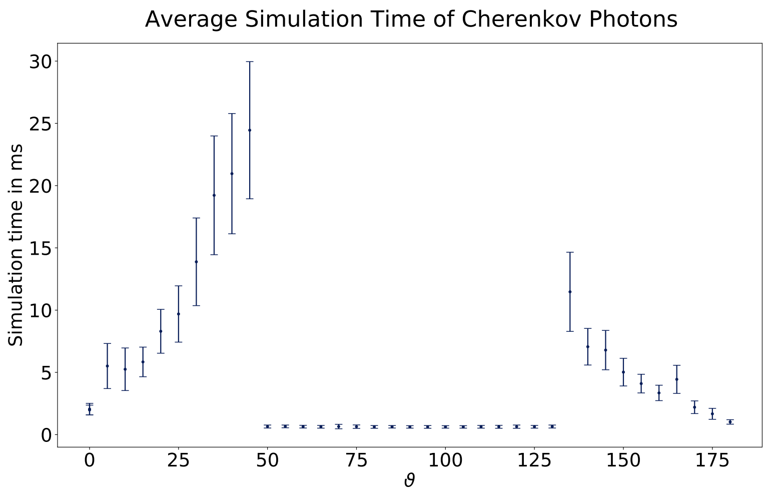

Figure 4.4: Average simulation time for Cherenkov photons dependent on momentum direction. The average simulation time of 500 identical photons with an errorbar of 1σ is shown over the angle ϑ. An angle of 180° translates to a momentum direction pointed towards the PMTs, while an angle of 0° translates to a momentum vector pointed towards the mirror.

probability, a higher number of simulated photons is needed which is time consuming. An additional problem is introduced due to momentum directions that lead to vanishing detection probabilities. In figure 4.2 this is the effect seen in the interval $\vartheta \in [50^\circ, 130^\circ]$. The momentum vector is angled towards a side wall of the TOP detector, requiring a high number of reflections for the photon before reaching the PMTs. While these photons are detected very seldom, a validation mechanism should still be able to compare the respective detection value distribution. Simulating enough photons to determine a detection value pattern for these origin phasespace points would further increase the overall simulation time.

Furthermore, as seen in figure 4.4 the average simulation time over ϑ also correlates with the track length. The simulation time is the time for the simulation of all detected photons out of 500 identically generated photons, which has been set here as one event. The total time requirement for simulating a set amount of detected photons therefore depends on the track length as well as the detection probability. Because of this, creating a validation dataset that covers the entire phasespace is not feasible in terms of simulation time. In the following chapters, varying methods to reduce the size of the necessary simulation dataset are evaluated.

4.1.2 Photon Detection Values

One possibility to determine the underlying uncertainty in Geant4 is to figure out the similarity between photon detection values of origin points and momenta close to each other in phasespace. Empirically determining a function that depends on the difference between photon origin coordinates and momenta and returns an uncertainty measure would be a training baseline for a neural network. To find this function, detection value patterns have to be compared with the help of an appropriate probability metric. To measure the uncertainty of Geant4, the data can be generated with the *OpticalGun* module of basf2. This module generates photons inside the TOP detector that are then propagated through the detector. The user can define the origin coordinates and momentum of the photon as well as its energy.

Two measures are important in this context: the distance in the spatial as well as in the momentum phasespace and the number of photons per origin point needed to differentiate between two different origins.

The first measure is to be determined by the analysis. As uncertainties also play a role in the determination of incidence time of the charged particle and in the measurement of the photons itself, positions very close to each other in phasespace will not vary much in respect to their detection value pattern. If an appropriate metric cannot differentiate between two photon origin points simulated by Geant4, then this should also be the case for the neural network solution.

For the second measure we need to consider the range of the total photon origin phasespace. As calculated in the previous chapter, the number of points in the phasespace exceeds 10^{12}. To get an exact probability density function from a measurement, we would need to simulate every point several times, preferably several hundred times. This is not feasible in terms of calculation time and memory requirements. A metric that can already differentiate detection values of photons originating from different positions and with different momentum directions with a small sample of photons per point is therefore needed.

This uncertainty measurement is not only useful to determine a baseline for the neural network training, but also serves as a probability metric test. A useful metric that is easy to compute and can correctly differentiate small samples drawn from similar probability distributions can be used in the validation of the fast simulation solution. If the metric found is also differentiable, it can even serve as a training metric. The measurements that need to be compared are in the shape of three-dimensional data points $(x_{\text{det}}, y_{\text{det}}, t_{\text{det}})$. Oftentimes, easy-to-compute metrics are one-dimensional, ignoring correlations between dimensions. To solve this problem, the high-dimensional KS metric explained in section 2.3.2 has been developed during this thesis. As this metric is nonparametric, fast computable and applicable in high dimensions, it is applied as a main distance measure in this thesis.

In this thesis the photon uncertainty is evaluated as follows: first an origin point is chosen as a comparison measurement. This origin point is fixed in the spatial coordinates $(x_{\text{ground}}, y_{\text{ground}}, z_{\text{ground}})$ and in the two angles $(\vartheta_{\text{ground}}, \psi_{\text{ground}})$. Then either the angles or the spatial coordinates are varied in a given interval with a chosen step size. For every point in this multi-dimensional grid, which has two dimensions for the angle coordinates and three dimensions for the spatial coordinates, a set number of photons is simulated. To have a higher sample of photon detection values, the variable $PhotonFraction$ inside basf2 is set to 1.0. As a baseline for this test, at least 100 photons per point should be simulated to be able to differentiate between different origin coordinates.

A subset with a specified number of detected photons per grid point is then selected for comparison. For every point in the grid a two-sample test is then performed, where sample one is the detected photons of the origin point while sample two is the detection values of the chosen grid point. For this two-sample test two metrics were used. The first metric is the Kullback-Leibler divergence, which is a widely used metric for high-dimensional data. The second metric is the newly developed ddKS metric, that has been explained in section 2.3.2.

As a first baseline, photons pointing directly towards the PMTs and in a deviation of 2° for ϑ and ψ in both directions are simulated. The starting coordinates of those photons are $(x, y, z) = (1.0\,\text{cm}, 0.0\,\text{cm}, 0.0\,\text{cm})$ in the coordinate system of the TOP detector. The slight deviation in x-direction from the center of the detector is necessary, so that photons with a momentum of $(p_x, p_y, p_z) = (0, 0, -1.0)$, which point towards the PMTs, do not hit the non-sensitive borders between pixels. The resulting distance grids can be seen in figure 4.5. The left plots have been calculated with the ddKS distance, while the plots on the right side have been evaluated with the KL divergence. The center point should contain the smallest distance measurement, if the detection values of $(\vartheta, \psi) = (180^\circ, 0^\circ)$ are distinguishable from every other origin point. The center line only contains the origin point, as a rotation around the z-azis done through ψ has no effect if the momentum vector is parallel to the z-axis. Points where no distance could be calculated because no photons reached the detector are set to -1. The distance measurement done with the ddKS distance is noticeably less noisy than that of the KL divergence. The good performance of the ddKS distance on datasets with small sample sizes in comparison to the KL divergence has been shown in section 2.3.2 and can be seen here as well. In figure 4.6 only points with a distance within a set margin of the ground distance in the center point are shown. The distances of the center point to itself are

$$KS_{\text{min}} = 0.4 \quad KL_{\text{min}} = -0.12 \quad . \tag{4.2}$$

The margin of error to the ddKS distance is set at 0.4, while for the KL divergence it is set at 7. The figure shows, that while the ddKS test excludes nearly all points in the grid, the KL divergence still retains some of the pattern seen in the earlier figure. Nevertheless, both tests can differentiate the center from the surrounding grid points very well.

For further analysis, the distance measurement is compared to the average number of reflections per grid point. The pattern seen very prominently in the KL divergence plots coincides with the number of reflections shown in figure 4.7. For each degree one reflection is added to the simulation, which then reproduces the detection value distribution of the origin. In comparison, the ddKS distance can differentiate between samples with different reflections, which supports its use in a validation technique.

For a further analysis of the influence of the average number of reflections on the probability distance and to find a possible baseline for a simulation, phasespace points with a higher number of reflections inside the detector were simulated. While different points showed very different behaviours, one prominent pattern could be seen throughout the detector. In figure 4.8 the point (x, y, z) = (17.88 cm, 0 cm, 46.95 cm) is chosen as the origin. To increase the number of reflections and therefore the spread of the detected photons, the angle interval was set as $\vartheta \in [10.7^\circ, 20.7^\circ]$ and $\psi \in [281.6^\circ, 301.6^\circ]$. Analagous to the analysis of photons pointed directly towards the PMTs, a ground "event" is chosen. The detection values of photons originating with euler angles of $(\vartheta, \psi) = (20.7^\circ, 291.6^\circ)$ are compared to every point in the grid of both angles.

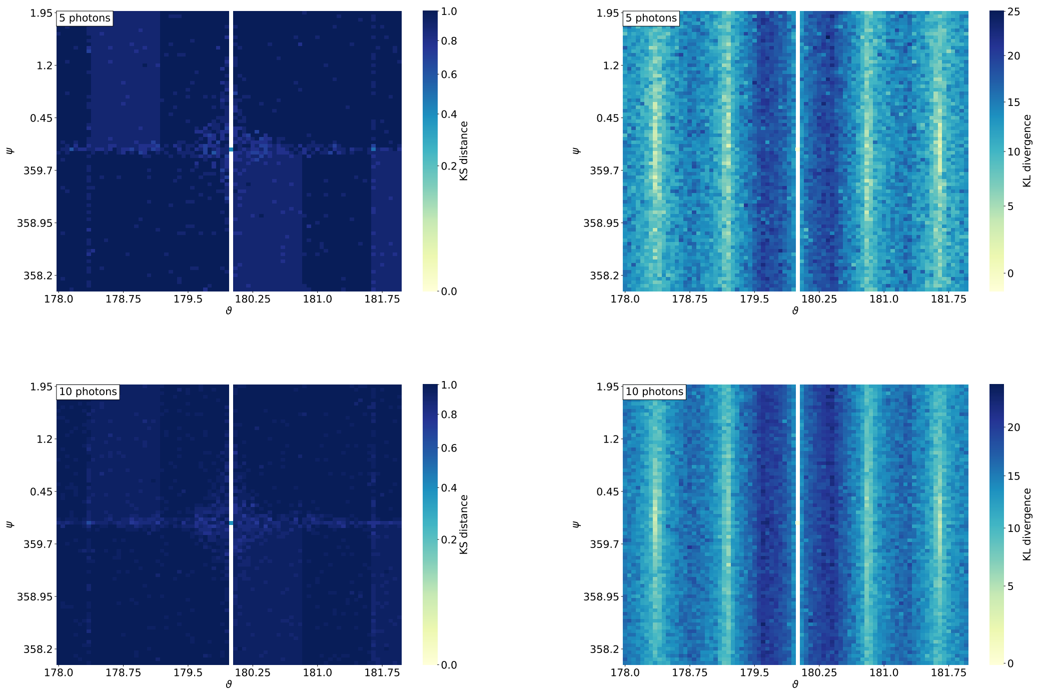

Figure 4.5: Photon detection value comparison. Photons originate at (x, y, z) = (0.7 cm, 0 cm, 0 cm) with their momentum given by $\vartheta \in [178^\circ, 182^\circ]$ and $\psi \in [358^\circ, 2^\circ]$. The comparison point is $\vartheta = 175^\circ$ and $\psi = 0^\circ$. On the left side the ddKS distance is used as a probability metric, on the right side the KL divergence. The upper row uses 5 detected photons per grid point, the lower row 10 photons.

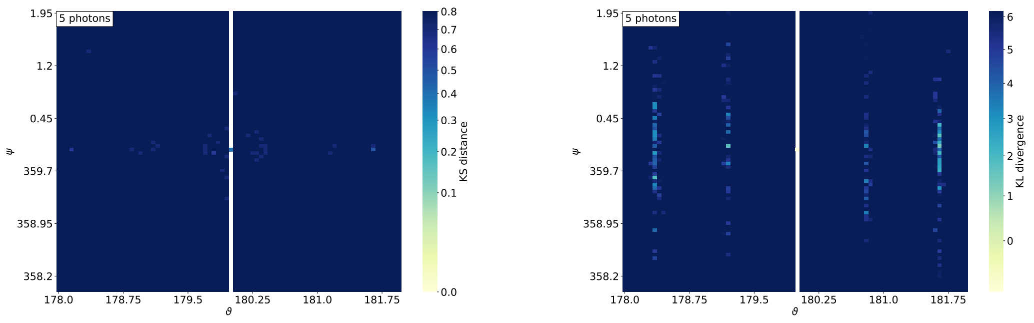

Figure 4.6: The ddKS distance (left) and the KL divergence (right) of photon detection values cut off at the distance of the center point plus a margin of variation. Every visible point shows the angles with which detection value distributions are created that are as similar to the detection value distribution in the center as it is to itself. In both plots 5 photons per sample have been used, the to the ground value added margin for the ddKS is set to 0.4, for the KL divergence to 7.

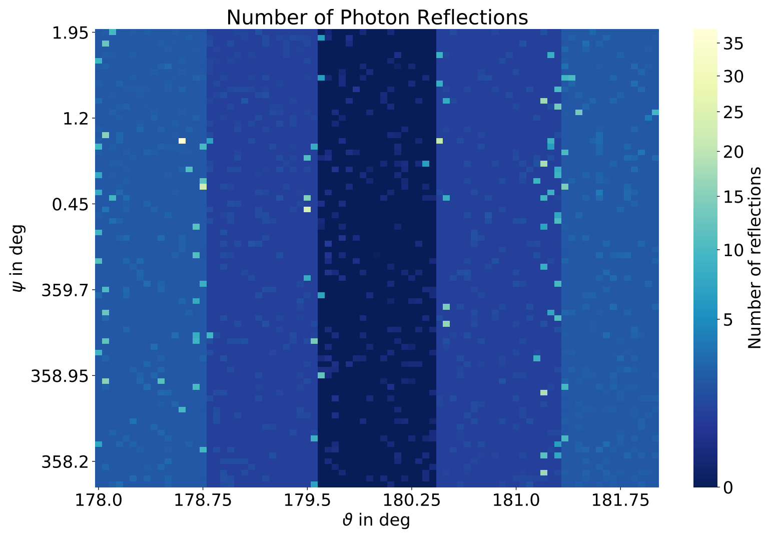

Figure 4.7: Number of photon reflections. The photons originate at (x, y, z) = (0.7 cm, 0 cm, 0 cm) with their momentum given by $\vartheta \in [178^\circ, 2^\circ]$ and $\psi \in [358^\circ, 2^\circ]$. The number of reflections of photons successfully reaching the PMTs is counted and their mean is shown here. The number of reflections can be linked to the probability distance between detection values.

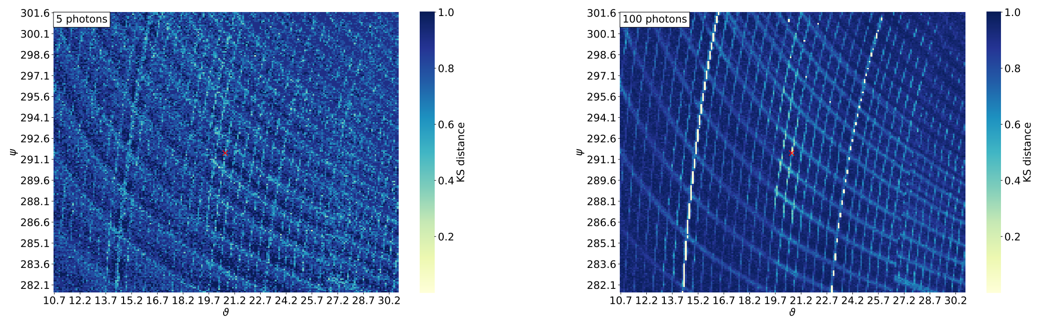

Figure 4.8: The ddKS distances of photon detection values. Photons starting at (x, y, z) = (17.88 cm, 0 cm, 46.95 cm) with points in the angle interval of $\vartheta = 20.7^\circ \pm 10^\circ$ and $\psi = 291.6^\circ \pm 10^\circ$ are generated. The detection values of every point in the ϑ-ψ grid are compared to those of the origin marked by the red x. For every momentum direction 500 photons were simulated, but only a random subset was selected for comparison. In the top row 5 photons were used, while the lower row shows 100 photons per sample.

The periodical pattern in figure 4.8 shows the difficulty in finding a base uncertainty for the detector. While the noise of the distance measurements is reduced by using higher numbers of photons per sample, the recurring lines of small distances are appearing in both small and large sample sizes. These coincide exactly with the pattern in figure 4.10, which displays the number of reflections before a photon is detected.

To evaluate if the chosen origin would be distinguishable from all other detection value distributions, in figure 4.9 only the points with a smaller ddKS distance than the origin to itself are displayed. While the number of phasespace points non-distinguishable from the origin is reduced with a higher sample size, setting a minimum phasespace distance to ensure distinguishability. This is additionally visualized in figure 4.11. The detection values in x and y are shown for the origin and two additional phasespace points. While the left and middle plot have a smaller phasespace distance, their average ddKS distance with a sample size of 5 photons is ddKS = 0.8. In comparison, the average ddKS distance between the left and right plot is ddKS = 0.41.

This has shown that the definition of an exact baseline for the detector is not possible due to the unpredictability of stochastic processes introduced by the reflections inside the detector. This analysis still is useful as a validation mechanism, which is explained in section 4.2.1.1.

4.1.3 Track Identification

As finding a minimum difference between photon origin values to be able to differentiate between those given the detection values is firstly not feasible and secondly very probably not generalizable, as shown in section 4.1.2, another training baseline needs to be defined. The TOP detector is one of the two sub detectors used for particle identification in the Belle II experiment, being able to identify particles given the photon detection values of their tracks. In basf2 these detection values are then input to a likelihood function which

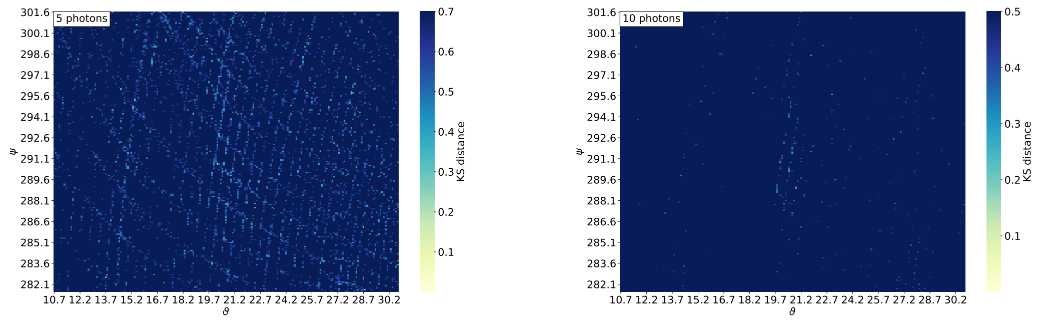

Figure 4.9: The ddKS distance of photon detection values cut off at the distance of the center point. Every visible point shows the angles with which detection value distributions are created that are as similar to the detection value distribution in the center as it is to itself. In the left plot, using 5 photons per sample, the origin distance is $\mathrm{ddKS}_{\mathrm{orig}} = 0.42$, while in the right plot with 10 photons per sample the origin distance is $\mathrm{ddKS}_{\mathrm{orig}} = 0.31$.

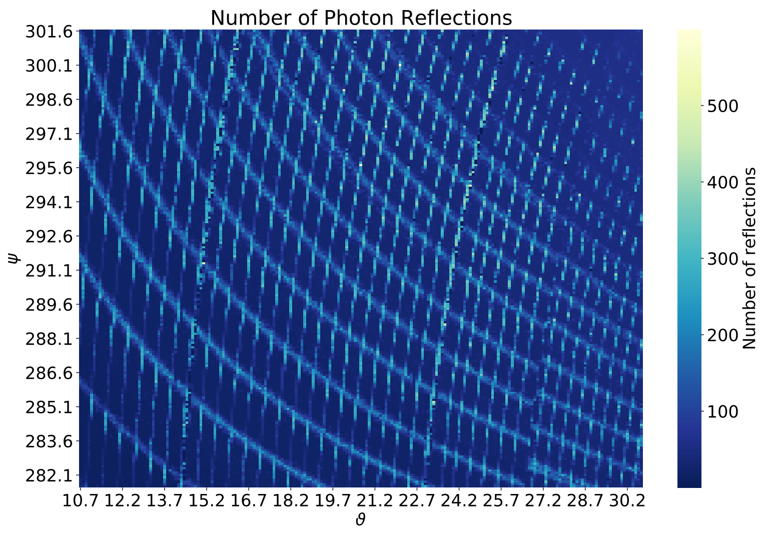

Figure 4.10: Number of photon reflections. Photons starting at (x, y, z) = (17.88 cm, 0 cm, 46.95 cm) with points in the angle interval of $\vartheta = 20.7^{\circ} \pm 10^{\circ}$ and $\psi = 291.6^{\circ} \pm 10^{\circ}$ are generated. For every track the number of reflections is stored and the mean of reflections per starting direction is shown here.

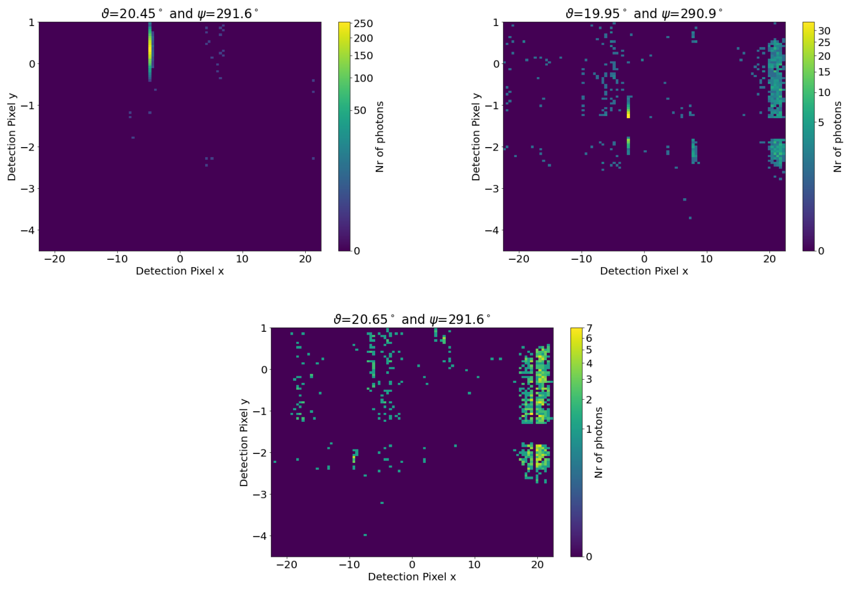

Figure 4.11: Photon detection values in x and y for three different starting directions. The detection values of 500 photons are shown for three momentum directions. While the angles of all three plots differ less than 1°, the average ddKS distance for samples with 5 photons between the lower and the top left distribution is 0.81. The average ddKS distance between the lower and the top right plot is 0.41.

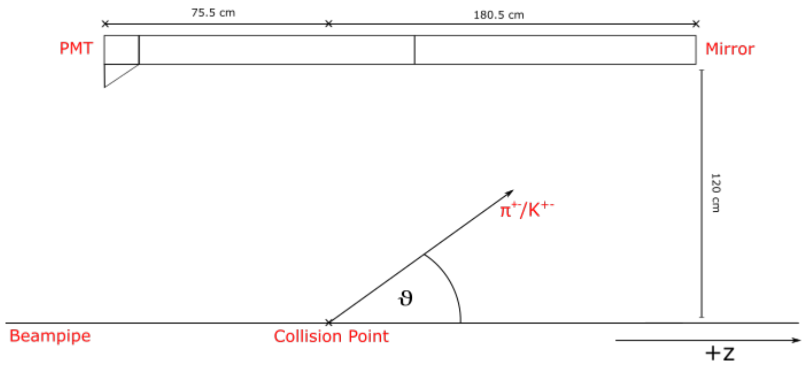

Figure 4.12: Schematic view of the *ParticleGun* simulation. A particle created at the collision point leaves towards the detector with an inclination angle ϑ and an azimuthal angle ψ (not shown in figure). One module of the TOP detector is shown with the relevant distances. Through these distances ϑ_{min} and ϑ_{max} can be calculated to ensure that the particle will cross the TOP detector.

returns the likelihood of the track belonging to a particle type. The input values of the likelihood function are the detection values (x, y, t) as well as the number of photons. One of the most difficult tasks is to distinguish between kaons and pions. Other particles that may cross the TOP detector and be identified with the likelihood function are protons, muons and electrons.

The performance of the track identification is dependent on the energy of the incoming particle as well as on the incident position. Evaluating the performance of the likelihood function depending on the charged particle's properties helps understanding the Geant4 simulation and the precision needed by a neural network.

As we developed the ddKS metric during the time of this thesis, most analyses are performed with ddKS additionally to the already implemented methods in basf2. This is especially helpful for the validation mechanisms explained in section 4.2, as an integration of an external module into basf2 is difficult. To show that ddKS can be used as a validation mechanism, we need a comparison between the performance of the ddKS metric and the original likelihood analysis.

For this analysis pions and kaons crossing the TOP detector were simulated. Pions and kaons were chosen, as distinguishing between those is the main task for the TOP detector. These particles can be simulated with the *ParticleGun* module of basf2. This module simulates the track of a given particle originating, if not specified otherwise, from the collision point inside the Belle II detector. For this particle both the inclination angle ϑ and the azimuthal angle ψ with respect to the beampipe can be set as well as the energy of this particle. The current simulation done by Geant4 then simulates the interaction between a crossing particle, in this case a kaon or pion with a specified entry position and energy, with the matter of the TOP detector. This setup is shown in figure 4.12. The azimuthal angle rotates the emission direction around the beampipe and is not depicted in this figure. Through the interaction of the particle with the material of the TOP detector the Cherenkov photons are created and then propagated through the detector.

As the detector is designed for particle identification purposes, a neural network simulation needs to hold up to current performance of the PID.

In figure 4.13 the comparison between the log likelihood function of the Cherenkov photons and the ddKS test dependent on the inclination angle as well as the energy is shown. For each comparison 1000 kaons/pions with a set absolute momentum and set inclination angle were simulated. Energies that are reasonable to occur in data are contained in the energy range of [0.4 GeV, 3.0 GeV]. While there is no restriction for the inclination angle given by physics, the particle should cross the TOP detector to emit Cherenkov photons. This leads to a minimum angle of $\vartheta = 34^\circ$ and a maximum angle of $\vartheta = 123^\circ$. As the TOP detector is built symmetrically around the beampipe, the azimuthal angle has a smaller effect on this analysis and is therefore set to 0° here.

The TOP likelihood module of $basf2$ returns a log-likelihood for different particle hypotheses, such as kaons or pions. For each track the TOP likelihood for the particle hypothesis "pion" and the particle hypothesis "kaon" is returned. The difference between those likelihoods then decides on the more likely hypothesis. The likelihood difference is calculated by

$$\Delta \log L = \log L_{\text{pion}} - \log L_{\text{kaon}} \quad , \tag{4.3}$$

where a positive difference confirms the pion hypothesis and a negative difference confirms the kaon hypothesis.

In addition the ddKS distance for each track is also calculated. The photons of one track were compared to photons generated by 1000 tracks of a pion/kaon with the same momentum and inclination angle. The ddKS distance difference should then be able to distinguish the probability distribution of pion generated photons and kaon generated photons. The ddKS difference is then given by

$$\Delta KS = KS_{\text{kaon}} - KS_{\text{pion}} \quad , \tag{4.4}$$

where analogous to the log likelihood a positive difference confirms the pion hypothesis, while a negative difference confirms the kaon hypothesis. This was done for 1000 kaons and pions. Fig. 4.13 shows the result for two momenta, p$\in [0.4, 3.0]$ GeV, and inclination angle $\vartheta \in [35^\circ, 115^\circ]$ in 5° steps. The black line marks y = 0. For every inclination angle the mean and the standard deviation of 1000 particles is shown. Blue points mark true pions while red points mark true kaons. The left side is done with the ddKS test, while the right side shows the log likelihood.

For small energies, the ddKS test actually outperforms the log likelihood analysis. For a momentum of 0.4 GeV, while the pions always have a higher log likelihood difference than the kaons, the absolute value of the kaons is in many cases above zero. This would mean a false particle identification. For higher energies, this also happens for some angles with the ddKS analysis. As the ddKS test does not factor in the number of photons, which is part of the likelihood function, this may improve its power in this case.

What can be derived from this analysis is the use of the ddKS test in particle identification. It correctly identifies the particles in most cases and especially in low energy regions.

A neural network simulation would need to reproduce these track identification features. As this has shown, the ddKS test can be applied for particle identification and can thus be used as a validation measure for particle identification.

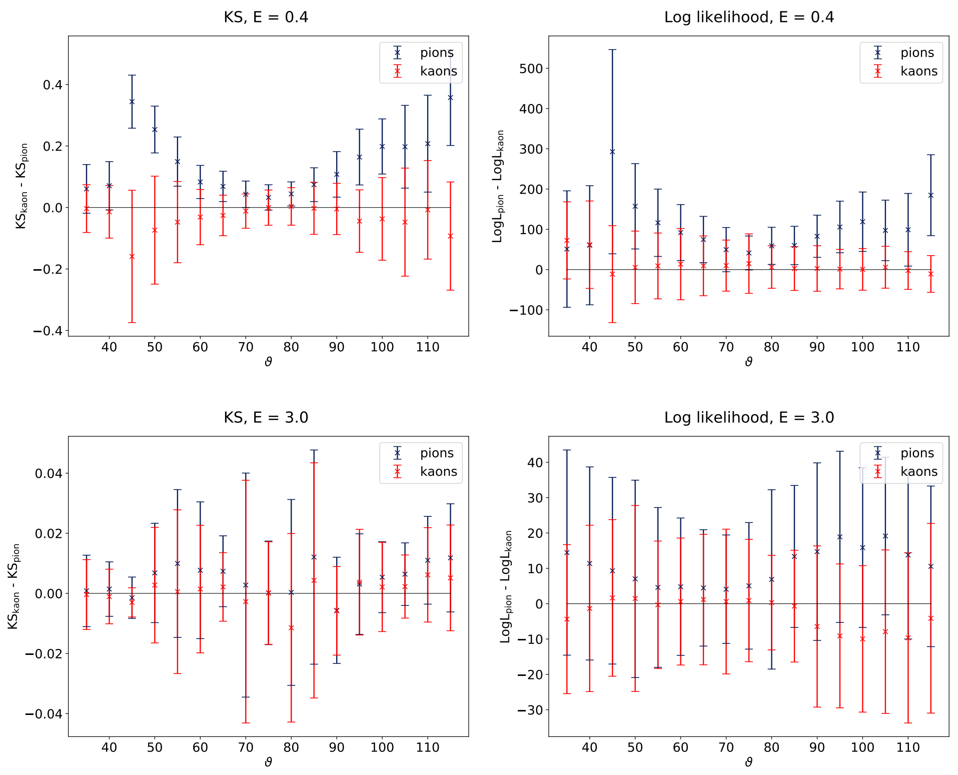

Figure 4.13: Comparison of ddKS test to likelihood analysis depending on the inclination angle and the momentum. Pions and kaons are created and the Cherenkov photons detected by the TOP PMTs are stored. Through these photons the hypothesis of the particle crossing the detector being a pion or a kaon is tested with the likelihood module in basf2 (right) as well as the ddKS test (left). Values above zero mean that the particle is more likely a pion while values below 0 suggest the kaon hypothesis. Blue points are generated pions, red points are generated kaons. The particles generated for the upper row have a momentum of 0.4 GeV, while the lower row contains the analysis of particles at 3.0 GeV.

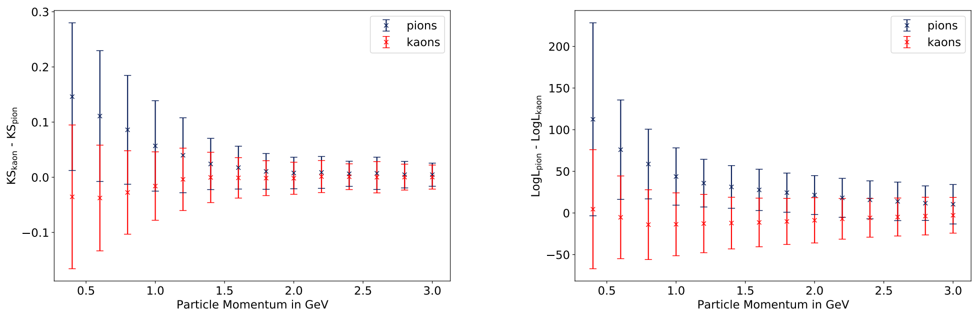

Figure 4.14: Comparison of ddKS test to likelihood analysis depending on the incoming particle's momentum. Values above zero can be interpreted as a confirmation of the pion hypothesis while values below zero suggest the kaon hypothesis. Blue points are true pions, red points are true kaons.

4.2 Design

The validation mechanisms explained in the previous section should be implemented in a way that is easy to apply to a trained network. I created a validation framework, tailored for a fast simulation solution of the TOP detector but with the possibility of generalization. While the goal was to set a limit on maximum deviation of the performance of the neural network solution for it to be a good replica of the Geant4 simulation, my work has shown that this is not possible. Nonetheless, the found patterns in both the photon detection value analysis and the track identification can be used to test a simulation replacement.

The validation procedure is split into two parts, depending on whether the performance on simulating the detection values of given photon position and momentum is judged or if the particle identification feature is used. The overall procedure can be seen in figure 4.15. The framework can either simulate new data given the model or work with already given data, depending on the use case. For an easy comparison, a method to simulate data through $basf2$ is also available.

4.2.1 Validation Mechanisms

The identifying features determined in the previous section have to be translated to validation mechanisms in order to be build a validation framework. As sections 4.1.2 and 4.1.3 have shown, a general training goal for the whole detector could not be found. While this makes the task of validating the training of a neural network solution more difficult than in standard neural network applications, there are still validation mechanisms that can be defined and used in a framework. I separate these into low-level and high-level validation features. Low-level validation mechanisms are comparisons on photon level, which can only be done by sampling random points of the photon phasespace. To feature the detector as a whole, the particle identification performance is implemented as a high-level validation feature.

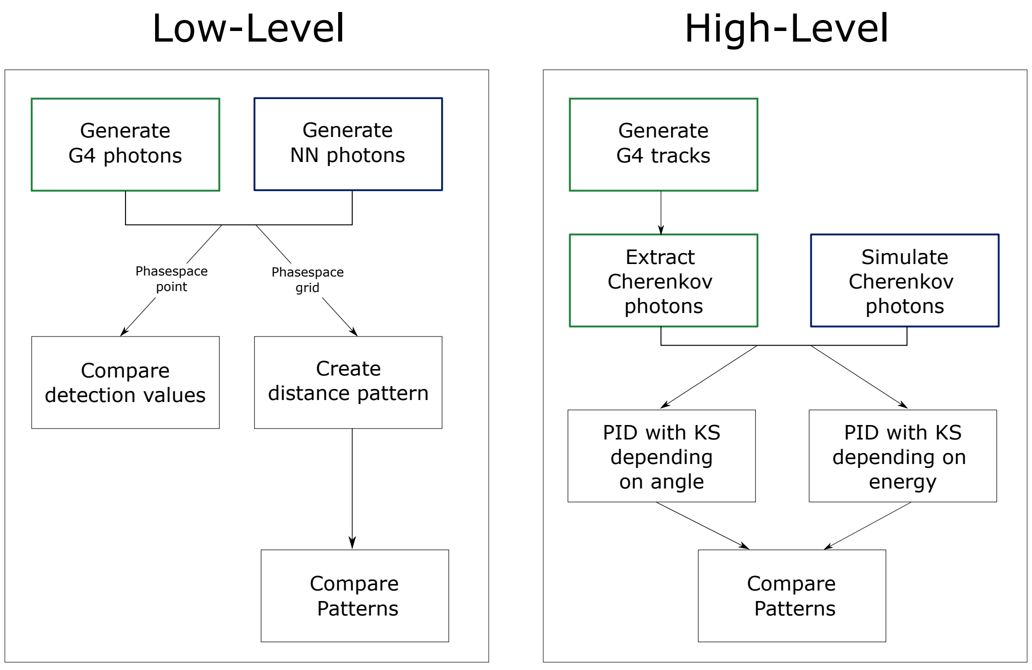

Figure 4.15: The structure of the validation framework. The general workflow is separated into low-level and high-level comparison. Depending on the wanted comparison, the simulation workflow varies as well. Steps marked in green are simulation steps of Geant4, while steps marked in blue have to be simulated by the neural network.

4.2.1.1 Low-level Validation

The low-level validation is based on the findings in section 4.1.2. As stated in this section, the determination of a general baseline for the current Geant4 simulation is not possible. For the neural network training it is therefore impossible to set a hard limit on the phasespace distance of the origin of photons that have to be distinguishable. Nonetheless evaluating the performance of a neural network simulation on detection value distributions is an important control mechanism to check small differences in the output of the neural network compared to Geant4.

For this two validation mechanisms are implemented into the framework. The first technique is a simple comparison of detection values. The framework is given two samples on which a two-sample test is applied using either the ddKS distance or the KL divergence. It returns the distance between the Geant4 and the neural network generated sample.

The second technique is the comparison of patterns. A range of photon starting values is given additionally to a origin tensor. These are then evaluated following the method explained in section 4.1.2. The framework creates dataframes that contain every evaluated point with the corresponding probability distance. Both the ddKS distance and the KL divergence can be chosen here as well. The patterns generated by both Geant4 and the neural network can then again be compared through a statistical test to have a higher-level validation possibility.

4.2.1.2 High-level Validation

Using high-level validation features removes the necessity of actively covering the whole phasespace for validating the performance of the neural network solution. The inherent randomness of the position and emission direction of Cherenkov photons created by charged particles crossing the TOP detector ensures a covering of at least the important parts of the detector phasespace. If points in the phasespace are not covered by those Cherenkov photons they also do not need to be simulated as they play no role in the actual physical process.

While we reduce the dimensionality problem of the phasespace, the creation of correct Cherenkov photons is more difficult than solely propagating them through the detector. In $basf2$ this is done through an interface to Geant4 which correctly simulates the interaction between particles and matter. As our neural network simulation only propagates photons given their position and momentum through the detector to the PMTs, these origin values need to be determined in a different way. As creating a separate interface to Geant4 would be outside of the scope of this thesis, a simpler method for validating track features is chosen. For high level features, first the track and the corresponding photons have to be simulated by the *ParticleGun* module. The photon origin and detection values created by basf2 are then stored. For the neural network, these generated photon origin values are then used as input variables. While this reduces the variation given by the emission of actual Cherenkov photons, as their emission time and emission direction is not completely determined, it is the easiest way to ensure a useful comparison.

After generating Cherenkov photons both for Geant4 and the neural network solution, particle identification is applied on the tracks. As the likelihood analysis in $basf2$ is not

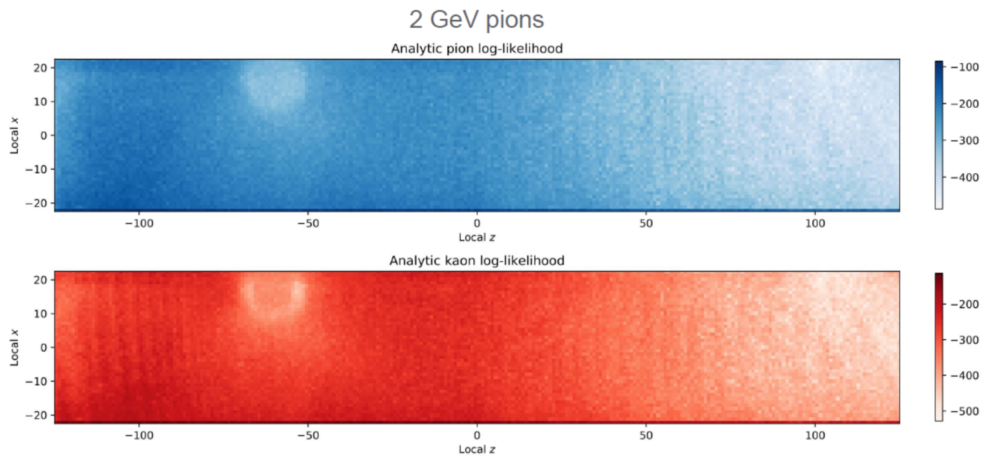

Figure 4.16: Features of the analytic likelihood calculation dependent on incident angle. Each event is binned by its incident position using a binning of 1 cm in both x and z direction. Then the pion and kaon likelihood is averaged and plotted respectively. This analysis was done in Belle II by [38].

accessible, the ddKS distance is used. As shown in section 4.1.3, ddKS is also applicable on particle identification. The framework then returns the ddKS distance difference between the kaon and pion hypothesis, which translates to the particle identification function. In future other particle hypotheses could also be included. Either the particle identification itself or the dependence on inclination angle and particle momentum can then be compared between both simulation types.

4.3 Generalization Possibilities

While the detection features analysed in this thesis are able to judge the performance of a neural network solution, for further applications a better generalization is preferred. Generalization can be seen as two separate aspects; the first aspect is the generalization for the simulation of the TOP detector, using features that can alone rate the success of a replacement simulation; whereas the second aspect is the generalization to other detector types.

In figure 4.16 features of the analytic likelihood dependent on the incident position of kaons and pions can be seen. Reproducing these features with a neural network simulation should guarantee a correct simulation, as this covers the phasespace of the detector and additionally tests the particle identification feature. The difficulty of introducing this as a validation feature is the implementation into $basf2$. As both the generation of Cherenkov photons through the interaction of particles with matter and the likelihood analysis is done in the workflow inside $basf2$, an integration with a neural network simulation is a complex task and could be done in further research.

The generalization to other detector types is a more advanced task. Fast simulations for different Cherenkov detectors such as the DeepRICH network [7] for the GlueX Cherenkov detector [39] and Cherenkov detectors at LHCb [5] and for high granularity calorimeters [6] have already been tested in high energy physics. The validation procedures are in all cases customized to the detector type and difficult to expand to other detectors. The per-cell-hit-energy used in the validation of a high granularity calorimeter does not apply in a Cherenkov detector or in other types of non-calorimetric detectors. Vice versa, the validation techniques developed in this thesis would be difficult or impossible to generalize to a hadron or electron calorimeter.

One generalization possibility is the use of the ddKS distance in most types of detector simulations. The data used in high energy physics is often high-dimensional and a fast computation is necessary for the application in neural network simulations. As the performance of the ddKS is in many cases better than most test statistics, especially in high-dimensions, its use in the validation of fast simulation can be beneficial.

5. Generative Modeling of Reduced Datasets

To replace the current simulation done through Geant4 by a fast simulation, an appropriate network architecture has to be chosen and trained on the full dataset of the TOP detector. As this is a huge task that probably requires a very deep and optimized network design, this is outside of the scope of this thesis. Additionally, introducing an active learning process that would optimize the generation of photons in the phasespace region where the network's loss is the highest, would also be necessary for an effective training. In this thesis the generative modeling is only applied to simplified versions of the TOP detector. This is useful for testing different network designs as well as testing validation techniques on smaller use cases. This chapter first presents the general experiment design, the used network architectures and the datasets. After this introduction the training and validation results of the respective network architectures to the corresponding datasets are shown and interpreted.

5.1 Experiment Design

In this chapter two datasets are used. The first one is a very simplified cone model, which models the cherenkov light propagation. This dataset is used as a proof of principle for different network designs, showing the behaviour of those designs with stochastic input. The second one is a dataset consisting of a reduced phasespace of the TOP detector, containing only photon trajectories with a set number of reflections inside the detector. Both datasets are modeled with a generative adversarial network approach as well as with a conditional variational autoencoder design. In all cases the hyperparameter optimization was done with the optuna framework [40].

5.1.1 GAN design

The GAN used in this chapter is a Wasserstein GAN introduced in section 2.2.2.1. The generator is a network consisting of three dense layers and the ReLu activation function. The number of nodes per layer n is set as a hyperparameter. The output activation function is the tangens hyperbolicus. The same architecture applies to the critic. The activation function inbetween layers for the critic is the LeakyReLu function and, as needed for a WGAN, the last layer does not use a non-linear activation function. For both networks, the optimizer and the respective learning rate is defined in the hyperparameter optimization

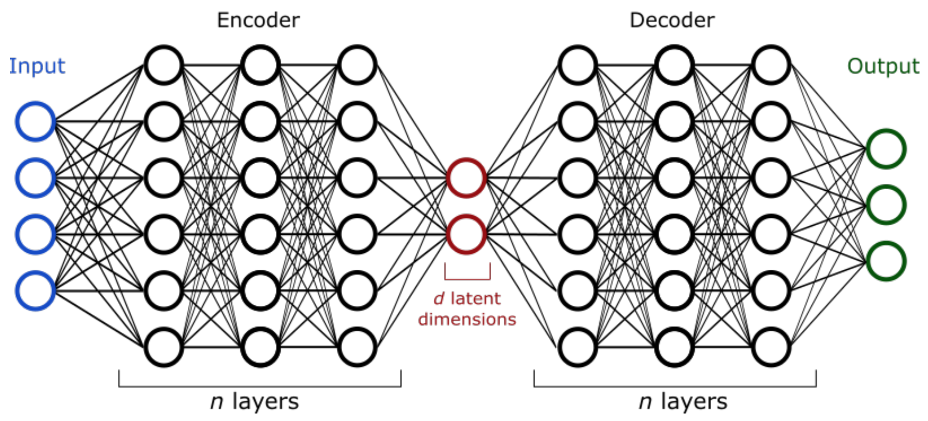

Figure 5.1: Overview of the CVAE structure used in this work. The encoder and decoder are identical, consisting of n fully connected layers with each z nodes. The input data consisting of both the conditions and the noise input is encoded into a d-dimensional latent space, which is then decoded to the output data. The parameters n, z and d are treated as hyperparameters and therefore set through a hyperparameter optimization depending on the application.

Additionally, the number of training iterations for the critic per training of the generator is also set as a hyperparameters.

5.1.2 CVAE Design

The CVAE used in this process follows the general architecture shown in section 2.2.2.2. It consists of an encoder, which encodes the input variables to a latent d-dimensional space, followed by a decoder, which then decodes this latent space into output variables. In this case both encoder and decoder are built out of n layers of fully connected layers with z neurons in each layer. An overview of the CVAE structure is shown in figure 5.1. The dimensionality of the latent space d, the number of layers n, and the number of neurons per layer z are hyperparameters and optimized with optuna. Additionally, the hyperparameter optimization contains the choice between the Adam optimizer [41] and the SGD optimizer with momentum [42] including the learning rate for the chosen optimizer.

For the actual training procedure, different loss functions have been tested. While the network still has to deal with probabilistic distributions, the Mean Squared Error (MSE) is applicable here as a comparison function between single points.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{5.1}$$

For further testing, two other loss functions were used. The first one is the ddKS function developed during this thesis (see section 2.3.2). As the ddKS metric is not differentiable, it is slightly adapted to be used as an objective function during network training. Instead of counting the membership numbers inside the orthants, a tangens hyperbolicus is used as a smoothing function. Additionally, the maximum difference between membership matrices is also softened through an exponential function. This leads to a differentiable loss function.

A second loss function designed to deal with probability distributions is an unbiased Sinkhorn divergence implemented in the geomloss package [43]. The Sinkhorn divergence interpolates between the Wasserstein distance and the kernel distance dependent on a “blur” parameter. This parameter defines the level of detail that the loss function should take into account, thus preventing overfitting on exact sample location.

$$S_\epsilon(\alpha,\beta) = \mathrm{OT}_\epsilon(\alpha,\beta) - \frac{1}{2}\mathrm{OT}_\epsilon(\alpha,\alpha) - \frac{1}{2}\mathrm{OT}_\epsilon(\beta,\beta) \tag{5.2}$$

5.1.3 Cone Model

To evaluate network designs and choose the most promising option, a simple cone model is tested. This cone model is modelled after the Cherenkov cone produced by a particle entering the TOP detector explained in section 3.1.1. If a network design performs well on this model, which mimics some of the physics processes in the overall TOP detector simulation as well, it may be useful in the TOP simulation. Additionally, the performance of generative models on stochastic processes can also be tested if such processes are introduced into this model.

For this cone model, firstly Cherenkov light created by a particle with a fixed momentum and therefore with a fixed cone opening angle is simulated. At different positions the x_e, y_e and z_e coordinate in relation to the emission point is simulated as well as the time t_e needed for the emitted photon to reach that position. The vector $\mathbf{y} = (x_e, y_e, t_e)$ contains the target coordinates for the generative model training. In figure 5.2 the simulated cone can be seen.

For conditional generative models, an input differentiating between modes of the generated space is needed. In the full TOP simulation, this input would consist of the position and the momentum of the simulated photon, while the output would be the detection values. Here we use a simplified version of this simulation, which additionally introduces a stochastic mechanism. The main input variable is the z_e variable, which determines the end position of the photon. For the stochastic influence, in each dimension x_e, y_e and t_e random numbers in the interval [0, 1] are produced. Both the points on the cone as well as the random numbers are sorted in each dimension. Every tuple (x_e,y_e,t_e) has therefore a corresponding tuple of three random numbers (x_r,y_r,t_r) which is somewhat proportional. The relation between both tuples cannot be defined by a function but is not completely random either. This leads to a stochastic uncertainty in the training of a generative model. In figure 5.3 the mapping of x_e to x_r can be seen. As the distribution of x_e is not uniform, the mapping to the stochastic variable is not linear. Additionally, a stochastically based input depending on the polar angle of the cone position is added to the conditional input variables. An array of random angles ϕ in the interval [0, 2π] is produced and then sorted according to the sorting of $\arctan x_e/y_e$. This returns a stochastically varied polar angle. Instead of the direct angle, $\sin\phi$ and $\cos\phi$ are used as input variables. The complete input vector to train a conditional neural network is therefore $\mathbf{x} = (z_e, x_r, y_r, t_r, \sin\phi, \cos\phi)$.

For the training procedure the input vector $\mathbf{x}$ is given as the conditional input to either the CVAE or GAN. The network is then additionally given random noise, which should be mapped to the correct output vector $\mathbf{y}$ according to the conditional input. To ensure the

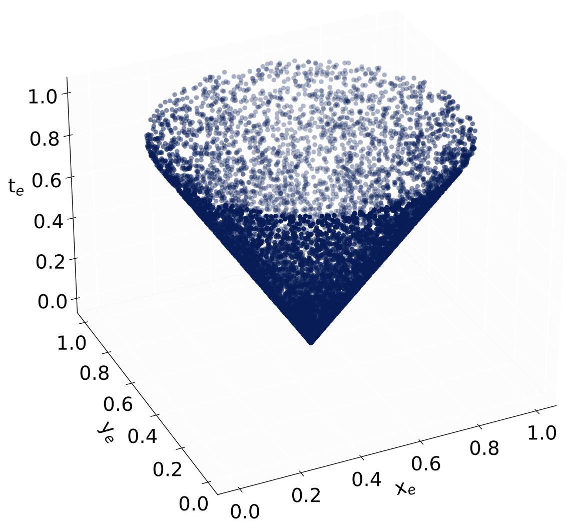

Figure 5.2: The cone distribution of $\mathbf{y} = (x_e, y_e, t_e)$. At a given point z_e the width of the cone is calculated. The coordinates (x_e, y_e) define a random point on the surface of the cone, while t_e is the time needed for a photon to reach this point. The network is supposed to model this distribution.

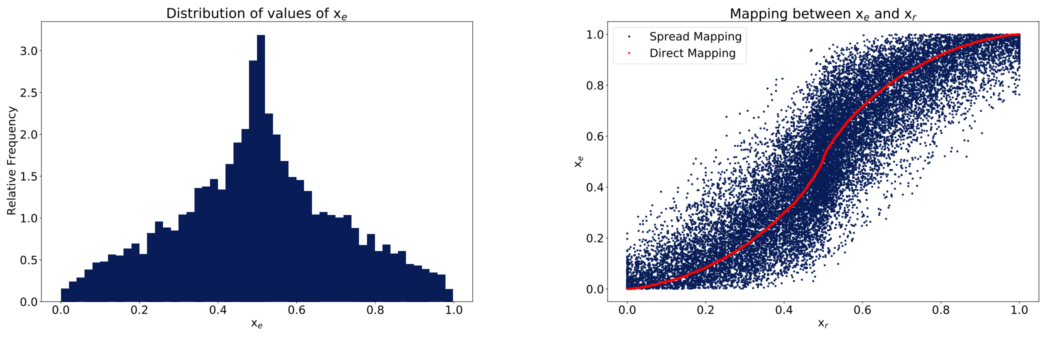

Figure 5.3: Distribution and mapping of the x position of the emitted photon to the random variable used for training. On the left side the distribution of x_e can be seen. For a given time t_e the radius of the cone is fixed, while the exact position (x_e, y_e) on the cone is variable. An even distribution between x_e and y_e is more likely than one coordinate being maximized. Instead of a straight mapping, on the right side the mapping between x_e and x_r can be seen. If 200000 x_e values are generated at once, the mapping is shown as the red line. As a stochastic variation is wanted, generating 200 x_e values 1000 times generates the mapping shown in blue.

stochastic variance shown in figure 5.3, for every training iteration only a small training set is produced, each containing 200 points. During training the loss function is continuously evaluated on each training set. After every hundredth iteration, a larger validation sample containing 10000 points is produced and evaluated with both the loss function and the ddKS metric. This ensures an exact evaluation of the performance of the network, as the stochastic variances are reduced in this validation sample.

5.1.4 Reduced Number of Reflections

To use the validation techniques explained in earlier chapters, I trained generative models on reduced TOP datasets. As every reflection of a photon inside the detector is done through a stochastic process, the measure of uncertainty increases with the number of reflections. To simplify the training, I generated data containing only photon trajectories with a set number of reflections. A possibility for future network designs could be a chain of two networks, one to categorize the photon starting point into the number of reflections and the second one trained on generating detection values dependent on the given number of reflections.

This dataset is generated through a slightly varied $basf2$. Through this variation $basf2$ returns the full path of the photon propagated through the TOP detector. Additionally, the detection values of the generated photons are stored as well. The detection values are matched to the last stored track position, as the tracks may also include non-detected photons. Only tracks made by detected photons are kept for further use. The number of reflections is then calculated by counting the number of changes in the momentum of the photon and can be filtered by the number of reflections wanted.

For generating photon trajectories that contain only one reflection, starting positions of photons are randomly sampled in the intervals

$$x \in [-20, 20]\,\mathrm{cm}, \quad y \in [-1, 1]\,\mathrm{cm}, \quad z \in [-120, 120]\,\mathrm{cm}, \quad \vartheta \in [90^\circ, 275^\circ], \quad \psi \in [0^\circ, 360^\circ] \quad . \tag{5.3}$$

Both tracks and detection values are generated for 10^6 photons and are then filtered for tracks that are reflected once or have no reflections inside the detector at all. The dataset then consists of $\mathrm{N}_\gamma = 3700$, where 1722 photons are not reflected and 1978 photons are reflected once.

In figure 5.4 the variable distribution before and after filtering for the numbers of reflections can be seen. On the left side the distributions of x, y, z is shown. The distribution of x and y did not change significantly, while the distribution for the reduced dataset in z has a skew towards the negative numbers. This is due to the positioning of the PMTs, which are at $z = -135\,\mathrm{cm}$. Starting positions closer to the PMTs are less likely to be reflected inside the detector.

For the momentum distribution, both datasets vary more prominently. A photon momentum vector pointing towards the PMTs would be $(p_x, p_y, p_z) = (0, 0, -1)$. The favoring of this vector is very visible on the left side of figure 5.4. While the distribution for both p_x and p_y on the unfiltered dataset is nearly uniform, the distributions on the filtered dataset are centered at 0. For p_z, while the unfiltered distribution is also a skewed right distribution, this trend is amplified in the filtered distribution. All photons that are reflected 0 or 1 time have a p_z momentum of nearly -1.

This change in distribution and therefore the uncertainty in photon detection values will reduce the difficulty for a neural network training while also reducing its ability to generalize. A network trained on this dataset is expected to perform well on all points belonging to this dataset, while performing bad on the general dataset of the TOP detector.

5.2 Generative Adversarial Network

5.2.1 Cone Model

Even with the improvement of Wasserstein GANs in comparison to Vanilla GANs, the training process of a GAN is difficult. With wrong hyperparameters oftentimes GANs collapes during training or do not improve at all. This can be seen here. The hyperparameters have been optimized with optuna and can be seen in table 5.1.

The training has been tested extensively. In figure 5.7 the loss for both the generator and the discriminator during 25000 iterations can be seen. The discriminator was trained for three iterations during one training iteration of the generator. The loss of the generator is low, leading to the expectation of realistic generated data. The ddKS distance during training did not improve. In figure 5.6 the variable distribution for x, y and t can be seen. The agreement between true and predicted distributions did not improve during training, which is also reflected in the predicted cone in figure 5.5.

As the discriminator was outperformed by the generator, a second trial with five iterations of discriminator training during one iteration of generator training was done. This should

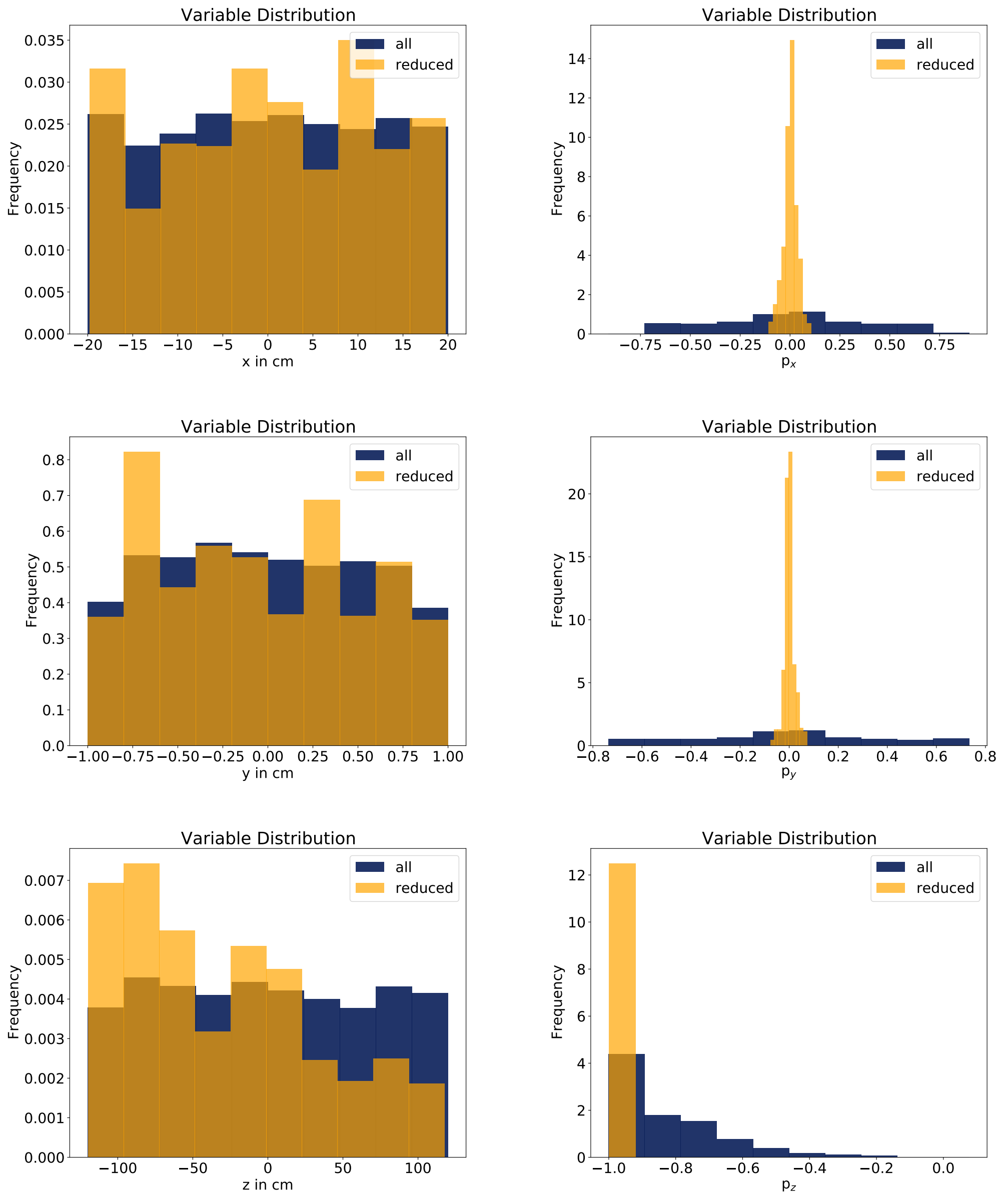

Figure 5.4: Variable distribution before and after filtering the number of reflections. On the left side the variables x, y and z are shown, while the right side shows the momentum distribution p_x, p_y, and p_z. All histograms are normed.

Table 5.1: Hyperparameters for the Wasserstein GAN trained on the cone model.

Hyperparameter	Value
Number of nodes	1000
Learning Rate G	0.00072
Optimizer G	SGD
Learning Rate D	0.0009
Optimizer D	SGD
Iterations D	3/5

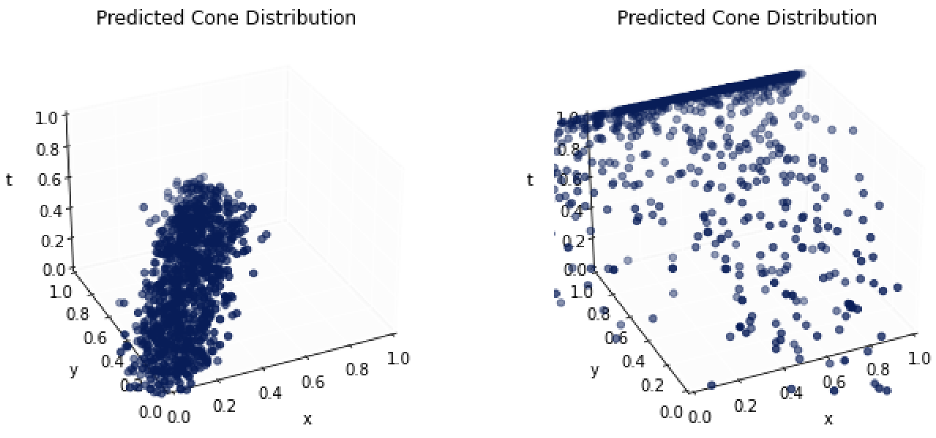

Figure 5.5: Predicted distribution of the cone model during training for the Wasserstein GAN. On the left side the predicted distribution before training is shown, while the right plot shows the predicted cone after 25000 iterations. The discriminator was trained for three iterations during one iteration of the generator.

improve the performance of the discriminator, forcing the generator to also improve on the data generation. In figure 5.10 the loss function of both the generator and discriminator during 25000 iterations is shown. The discriminator's loss function is less than that of the generator, which is expected at first. Over training an improvement of the performance of the generator should be visible. This is not the case here. The generator does not generate realistic images, which is supported by the high ddKS distance on the validation set during training. In figure 5.9 and figure 5.8 both the variable distributions and the predicted cone data before and after training is shown. In both cases there is no improvement after training.

Possible improvements could be made with a further fine-tuning of the hyperparameters, with the use of training techniques mentioned in section 2.2.2.1 or with a change of the network architecture. This has not been done during this thesis, but could be done in the future. As the performance of the CVAE had surpassed that of the WGAN on these dataset, the focus was set on this network architecture.

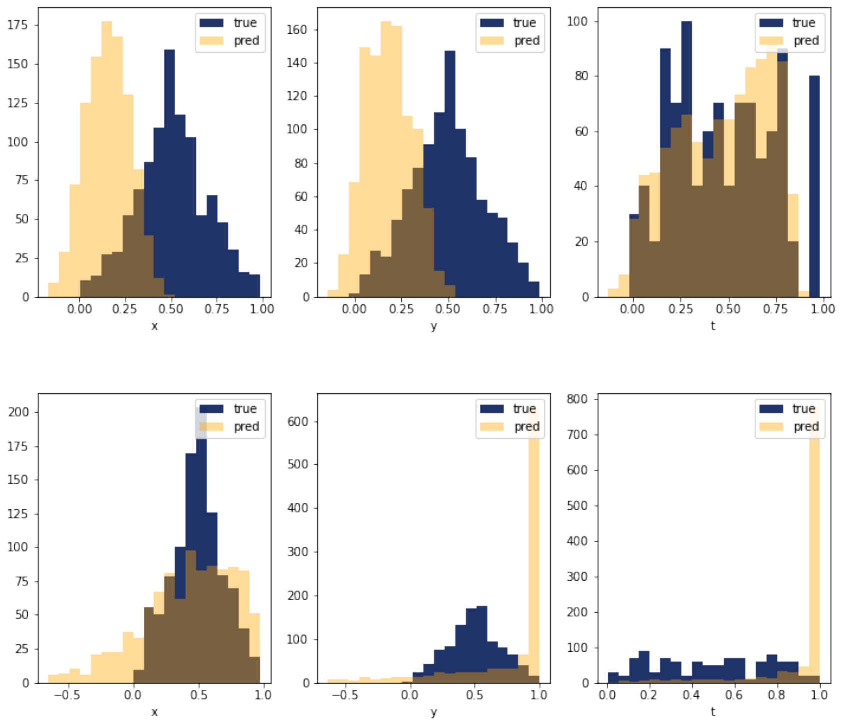

Figure 5.6: True and predicted distribution of the respective variables of the cone model during training for the Wasserstein GAN. Plots on the left show the distribution of the x coordinate, the middle column shows the distribution of the y coordinate and the right column shows the distribution of the t coordinate. The upper plots are before training, while the lower plots are the training results after 25000 iterations. The discriminator was trained for three iterations during one iteration of the generator.

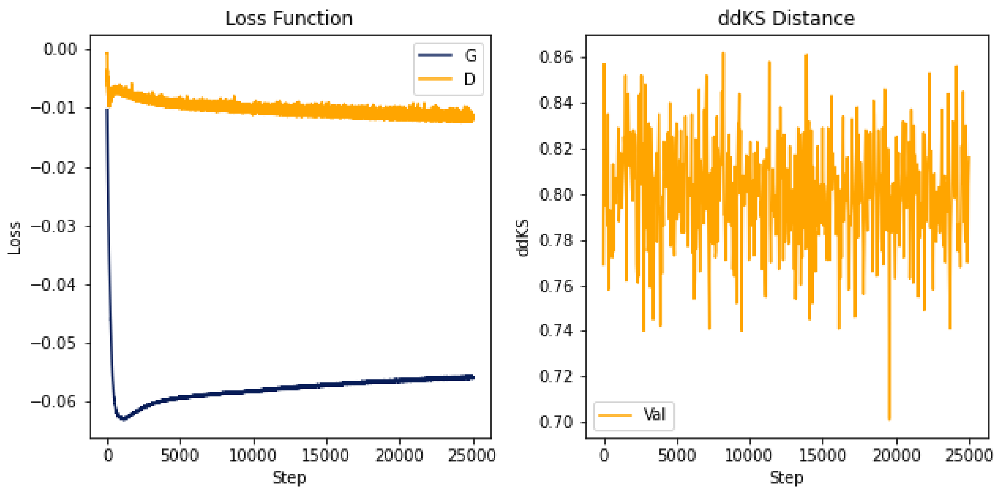

Figure 5.7: The discriminator D and generator G loss of the Wasserstein GAN during 25000 iterations. The discriminator is not able to differentiate real from predicted data. The discriminator was trained for three iterations during one iteration of the generator. The ddKS distance between the true and predicted cone shows that the agreement is very small.

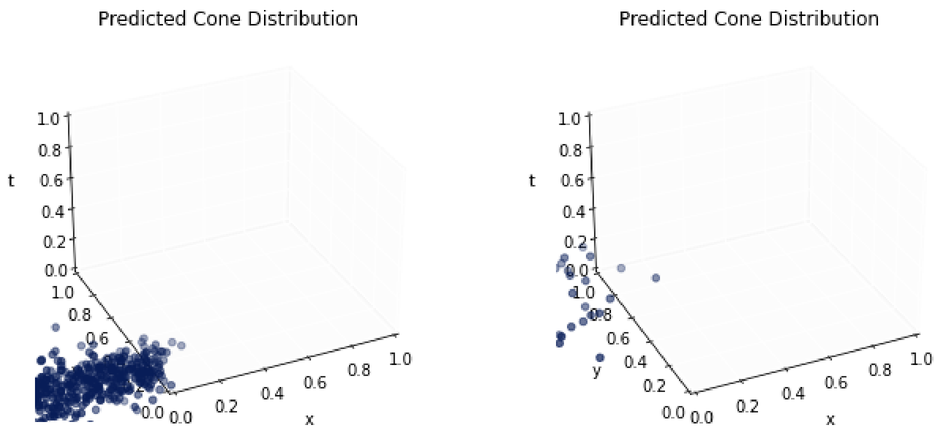

Figure 5.8: Predicted distribution of the cone model during training for the Wasserstein GAN. On the left side the predicted distribution before training is shown, while the right plot shows the predicted cone after 25000 iterations. The discriminator was trained for five iterations during one iteration of the generator.

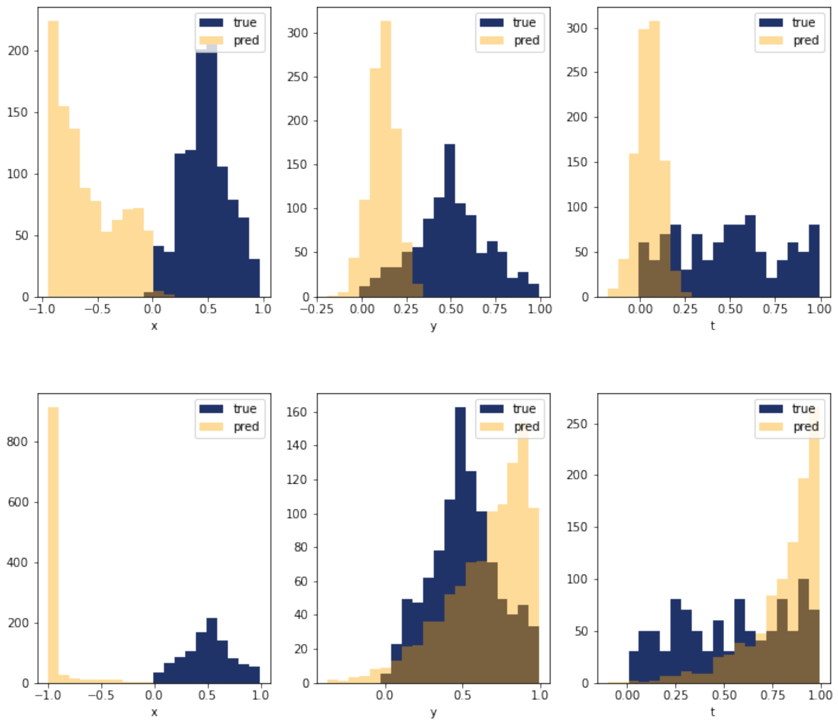

Figure 5.9: True and predicted distribution of the respective variables of the cone model during training for the Wasserstein GAN. Plots on the left show the distribution of the x coordinate, the middle column shows the distribution of the y coordinate and the right column shows the distribution of the t coordinate. The upper plots are before training, while the lower plots are the training results after 25000 iterations. The discriminator was trained for five iterations during one iteration of the generator.

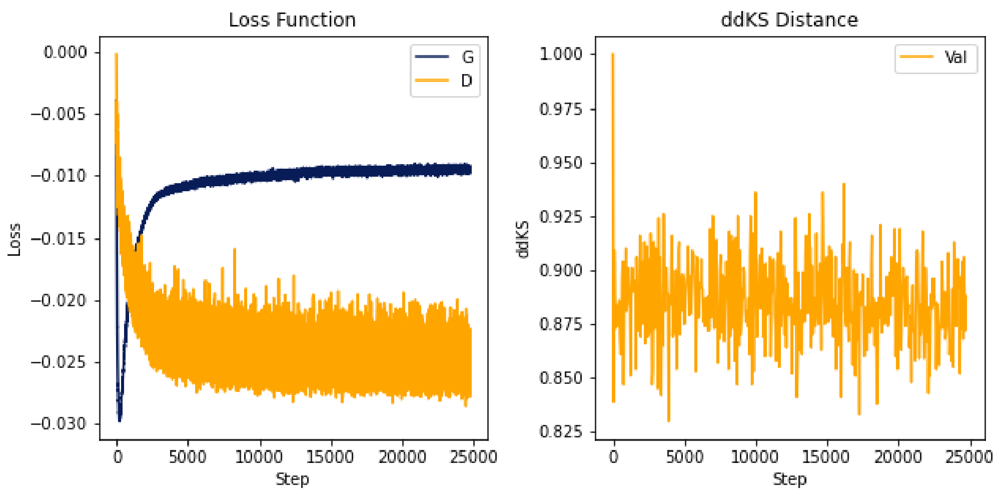

Figure 5.10: The discriminator D and generator G loss of the Wasserstein GAN during 25000 iterations. The discriminator continues to improve while the generator is not able to fool the discriminator. The ddKS distance between the true and predicted cone shows that the agreement is very small. The discriminator was trained for five iterations during one iteration of the generator.

5.3 Conditional Variational Autoencoder

5.3.1 Cone Model

For the CVAE trained on the cone model, the used hyperparameters optimized through the optuna optimization framework are shown in table 5.2, table 5.3, and table 5.4 for the networks trained with the Sinkhorn, MSE and soft ddKS loss respectively. Each network is trained until the convergence of the ddKS validation loss. The network training process is shown here through the training and validation loss in figure 5.16, figure 5.13, and figure 5.19. In each figure the training and validation ddKS distance is also plotted. While the loss converges quickly in all cases, the ddKS distance reduces slower, showing that the networks continue to improve.

To visualize the actual improvement, for each network the distribution of the output variables x_e, y_e, t_e is compared to the true distribution for each variable before training and after 3000 training iterations. For the networks trained with the Sinkhorn and the MSE loss function, this can be seen in figure 5.15, figure 5.12 and in figure 5.18. What can be seen here is that the networks correctly reproduce the variable distribution given by the cone mapping. This is also shown by the plotting of the predicted cones in figure 5.14 and figure 5.11. After 3000 iterations the training correctly reproduces the cone distribution.

In the case of the CVAE trained with the soft ddKS loss function, while the loss function does converge, the predicted distributions differ greatly from the true distributions. This is shown in the distributions of the output variables in figure 5.18. No variation of the output variables can be seen. Additionally, this is also seen in the predicted cone shown in figure 5.17. In this figure we see that the distribution has collapsed onto one point centered

Table 5.2: Hyperparameters for the CVAE trained on the cone model. The loss used is the Sinkhorn loss.

Hyperparameter	Value
Latent dimension	5
Number of hidden layers	4
Number of nodes per layer	27
Optimizer	Adam
Learning Rate	0.0057

Table 5.3: Hyperparameters for the CVAE trained on the cone model. The loss used is the MSE loss.

Hyperparameter	Value
Latent dimension	5
Number of hidden layers	8
Number of nodes per layer	28
Optimizer	Adam
Learning Rate	0.0032

in the middle of cone. Training with the soft ddKS loss always resulted in similar mode collapses. This may be explained through the high power of the ddKS test. As the ddKS test is able to distinguish distributions using small samples and detecting small differences in distributions in comparison to other probability metrics, it is not very useful as a loss function. Changing the distribution of the predicted cone function slightly towards the true distribution may not change the ddKS distance. This makes the update process of the optimizer difficult and leads to a mode collapse of the training process. Further fine-tuning of the hyperparameters and very careful observation of the training process might alleviate this problem, but as the network performs well with different loss functions, this has not been done extensively during this thesis.

5.3.2 Reduced Number of Reflections

As the WGAN already had difficulties on learning the cone model, only the CVAE was trained on the reduced photon dataset. In table 5.5 the optimized hyperparameters are shown. The used loss function here was the MSE loss, training on the Sinkhorn loss did

Table 5.4: Hyperparameters for the CVAE trained on the cone model. The loss used is the soft ddKS loss.

Hyperparameter	Value
Latent dimension	18
Number of hidden layers	12
Number of nodes per layer	92
Optimizer	Adam
Learning Rate	0.0065

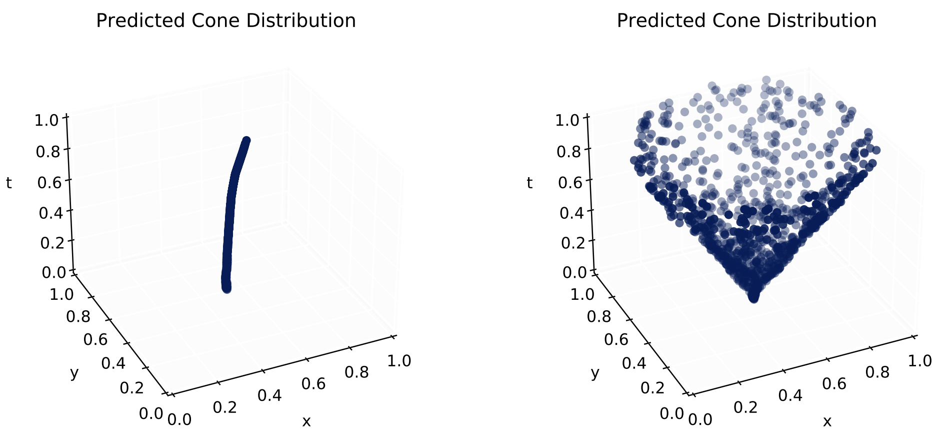

Figure 5.11: Predicted distribution of the cone model during training. The loss function used on this training is the MSE loss. On the left side the predicted distribution before training is shown, while the right plot shows the predicted cone after 3000 iterations.

not return different results. In figure 5.20 the training and validation loss is shown. Both converged robustly during training and were not prone to overfitting as the performance on the validation set also returned good results. For further validation, the ddKS distance was evaluated on the validation dataset during training (see figure 5.21). Additionally, the ddKS distance distribution for 2000 generated samples is also shown, showing a good agreement between true and predicted data with an average ddKS distance of

$$\mathrm{ddKS} = 0.13 \quad . \tag{5.4}$$

For visual results the true and predicted photon detection values for 400 photons in the x-y plane are shown in figure 5.22, the true and predicted time distribution in figure 5.23. The visual agreement is good, further showing the performance of the CVAE. The concentration of the detection points in the x-y plane in the upper part of the PMTs is given through the reduction on zero or one reflection. As the photons enter the prism with a small reflection angle, the lower PMTs are not reached by those photons.

To evaluate the performance of this model on the general dataset, 400 photons with an arbitrary number of reflections are generated and their starting coordinates given to the trained model. In figure 5.24 and figure 5.25 the true and predicted distribution in the x-y plane and time respectively is shown. As visual evaluation shows that the agreement between both distributions is vanishingly small, no further distance measurements have been done.

Nevertheless, the performance of the CVAE surpasses the performance of the GAN and is therefore a reliable candidate to be trained on the full dataset. As a reduction of the number of reflections simplified the task, a possible next step could be separating the full simulation into two parts. A first network could be trained on the prediction of the number

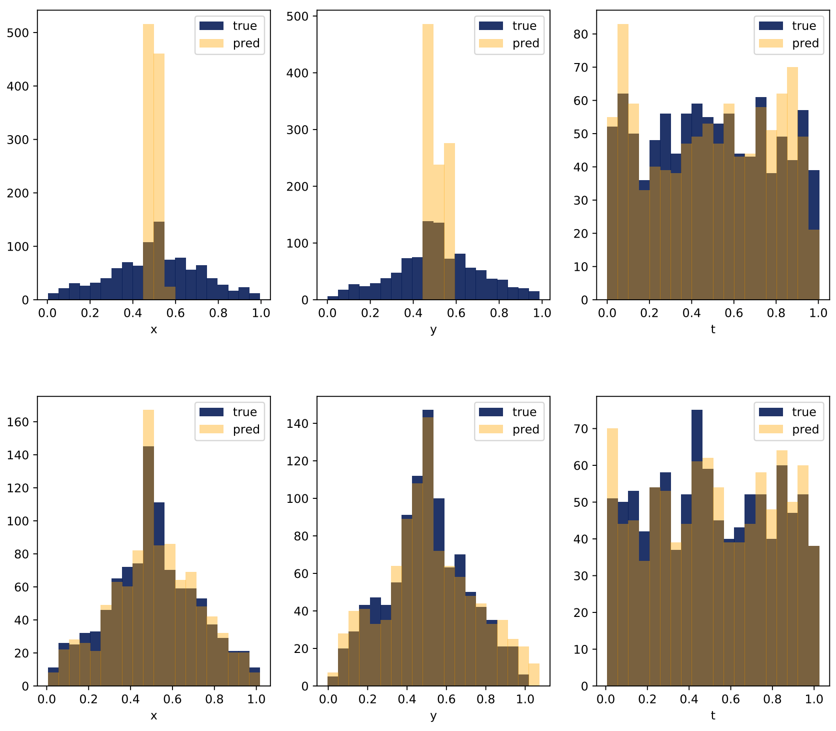

Figure 5.12: True and predicted distribution of the respective variables of the cone model during training. The loss function used on this training is the MSE loss. Plots on the left show the distribution of the x coordinate, the middle column shows the distribution of the y coordinate and the right column shows the distribution of the t coordinate. The upper plots are before training, while the lower plots are the training results after 3000 iterations.

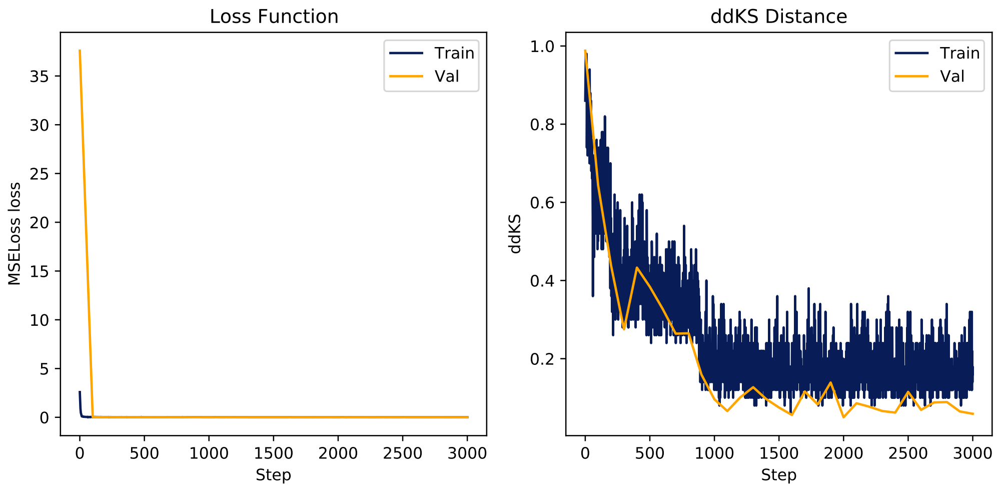

Figure 5.13: The MSE loss and the ddKS distance between the true and predicted cone model after 3000 iterations. On the left side the behaviour of the MSE loss function during training and validation can be seen, while the right side shows the ddKS distance. The training loss and ddKS distance is calculated after every iteration, whereas the validation loss and ddKS distance is calculated after every 100th iteration.

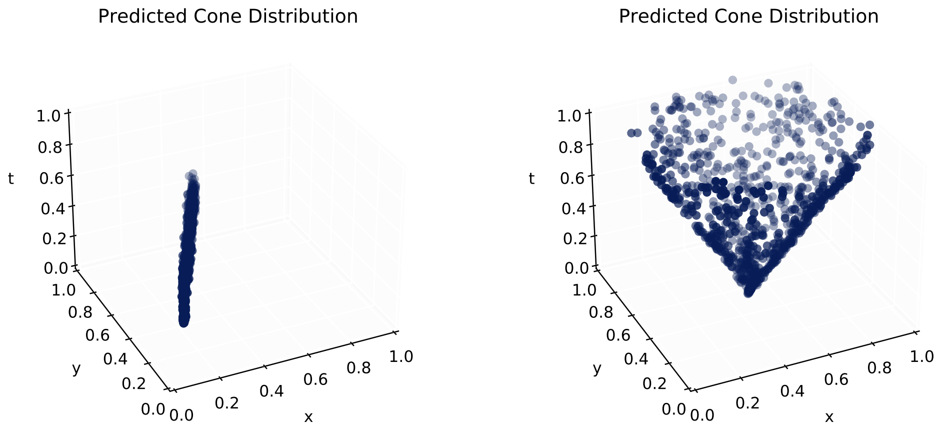

Figure 5.14: Predicted distribution of the cone model during training. The loss function used on this training is the Sinkhorn loss. On the left side the predicted distribution before training is shown, while the right plot shows the predicted cone after 3000 iterations.

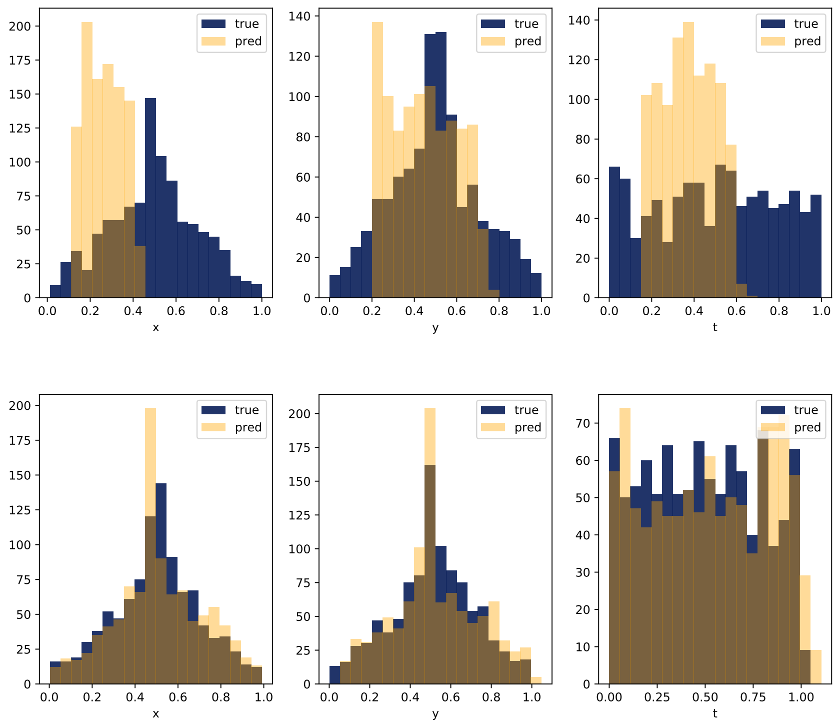

Figure 5.15: True and predicted distribution of the respective variables of the cone model during training. The loss function used on this training is the Sinkhorn loss. Plots on the left show the distribution of the x coordinate, the middle column shows the distribution of the y coordinate and the right column shows the distribution of the t coordinate. The upper plots are the distributions before training, while the lower plots are the training results after 3000 iterations.

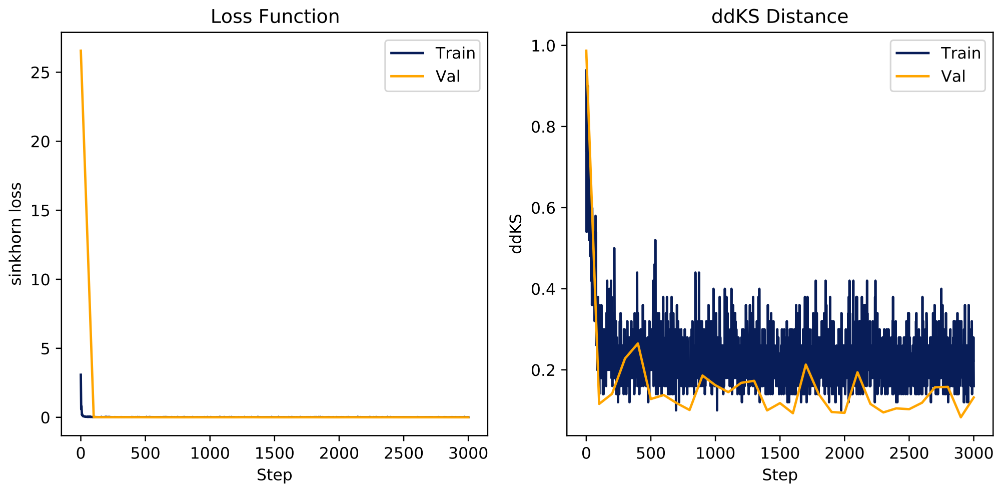

Figure 5.16: The Sinkhorn loss and the ddKS distance between the true and predicted cone model during 3000 iterations. On the left side the behaviour of the Sinkhorn loss function during training and validation can be seen, while the right side shows the ddKS distance. The training loss and ddKS distance is calculated after every iteration, whereas the validation loss and ddKS distance is calculated after every 100th iteration.

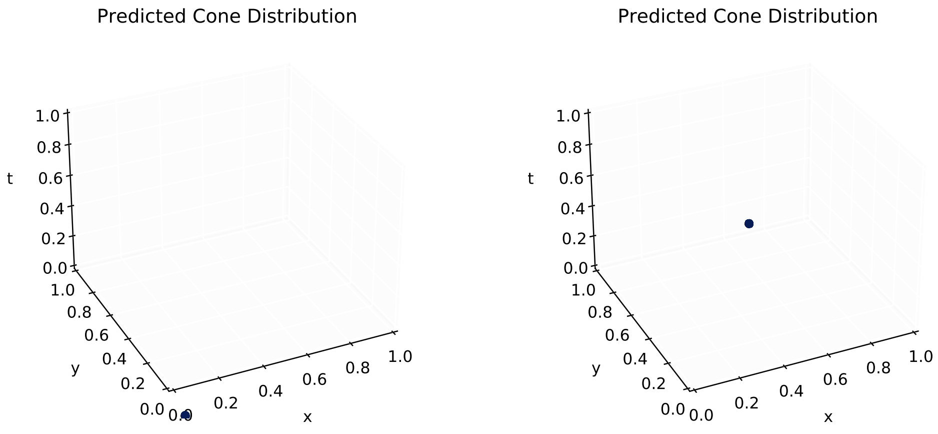

Figure 5.17: Predicted distribution of the cone model during training. The loss function used on this training is the soft ddKS loss. On the left side the predicted distribution before training is shown, while the right plot shows the predicted cone after 3000 iterations.

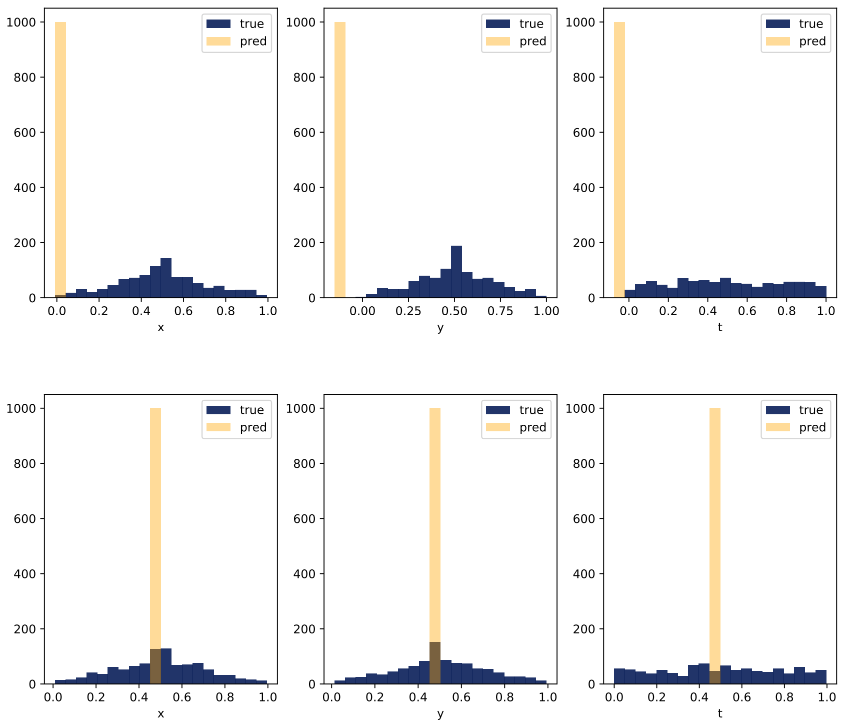

Figure 5.18: True and predicted distribution of the respective variables of the cone model during training. The loss function used on this training is the soft ddKS loss. Plots on the left show the distribution of the x coordinate, the middle column shows the distribution of the y coordinate and the right column shows the distribution of the t coordinate. The upper plots are the distributions before training, while the lower plots are the training results after 3000 iterations.

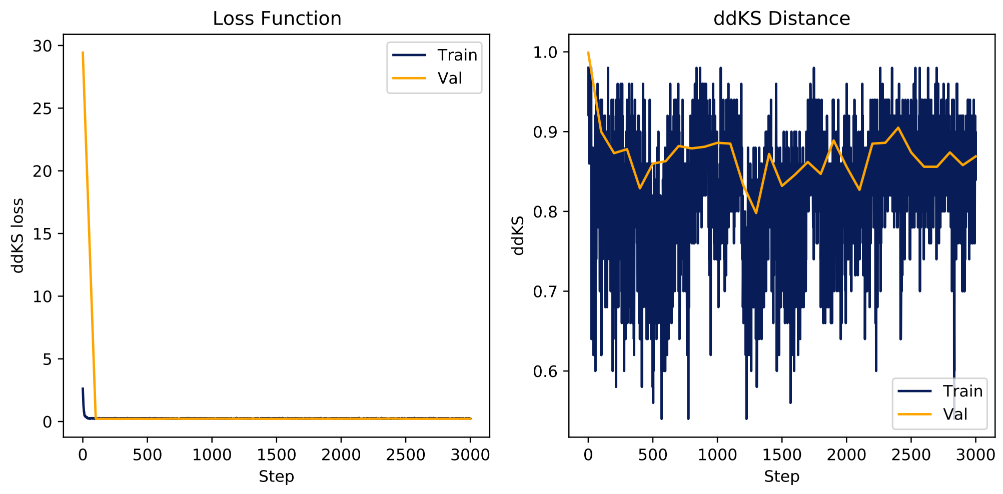

Figure 5.19: The soft ddKS loss and the ddKS distance between the true and predicted cone model during 3000 iterations. On the left side the behaviour of the soft ddKS loss function during training and validation can be seen, while the right side shows the ddKS distance. The training loss and ddKS distance is calculated after every iteration, whereas the validation loss and ddKS distance is calculated after every 100th iteration.

Table 5.5: Hyperparameters for the CVAE trained on the reduced photon dataset. The loss used is the MSE loss.

Hyperparameter	Value
Latent dimension	10
Number of hidden layers	10
Number of nodes per layer	42
Optimizer	Adam
Learning Rate	0.0075

of reflections depending on the photon starting coordinates. The output of this first network could then be an additional conditional input to a second network that would generate the detection values depending on the starting coordinates and the predicted number of reflections. This would separate the full phasespace into parts depending on the reflections and therefore possibly reducing the stochastic uncertainty.

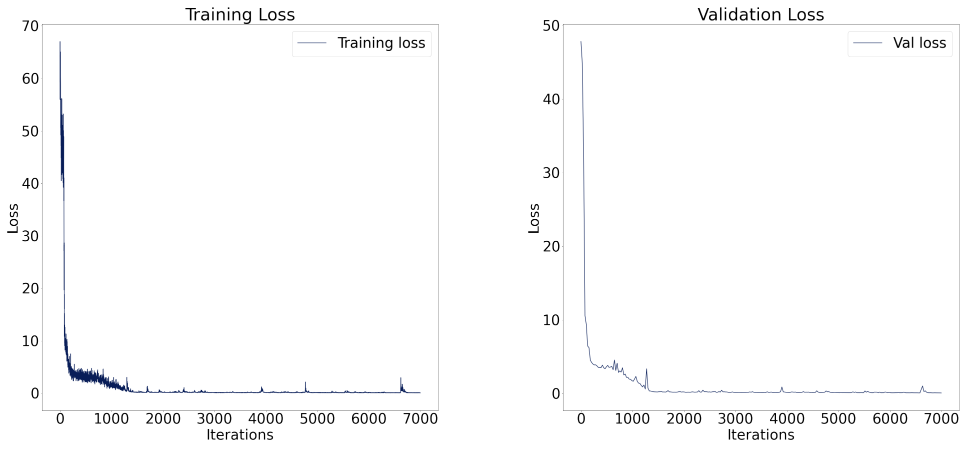

Figure 5.20: Training loss and validation loss of the CVAE trained on single reflections. The plot on the left shows the training loss during 7000 iterations, whereas the plot on the right shows the validation loss evaluated after every epoch. The loss used is the MSELoss.

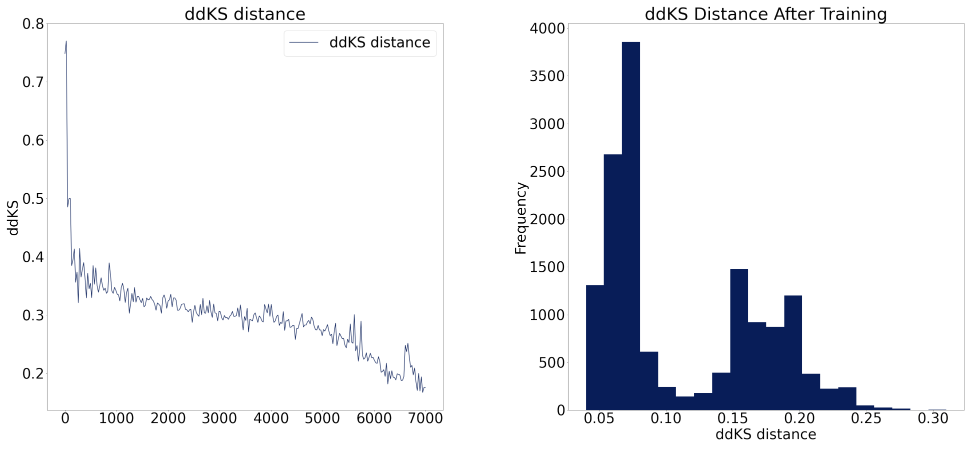

Figure 5.21: The ddKS distance evaluated on a validation set during and after training for the CVAE trained on single reflections. On the left side the ddKS distance is continuously calculated after every epoch for a independent validation dataset, on the right side the ddKS distance distribution on the validation dataset after training is shown. For both sides 100 points per sample were used.

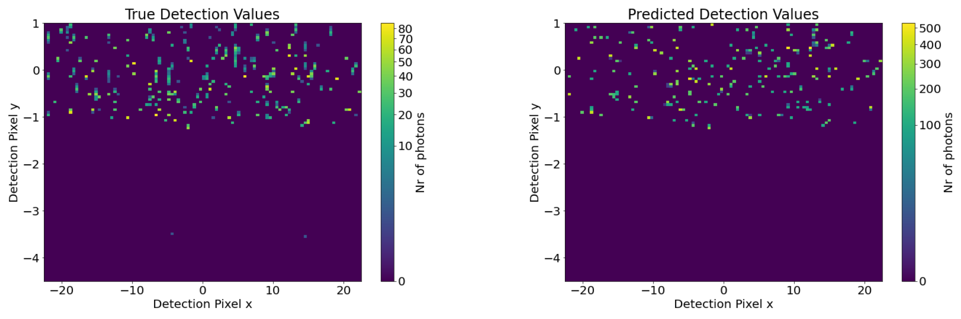

Figure 5.22: True and predicted photon detection values for 400 photons in the $x-y$-plane belonging to the filtered dataset. The left side shows the true detection values of photons that were only reflected 0 or 1 time, while the right side shows the predicted values.

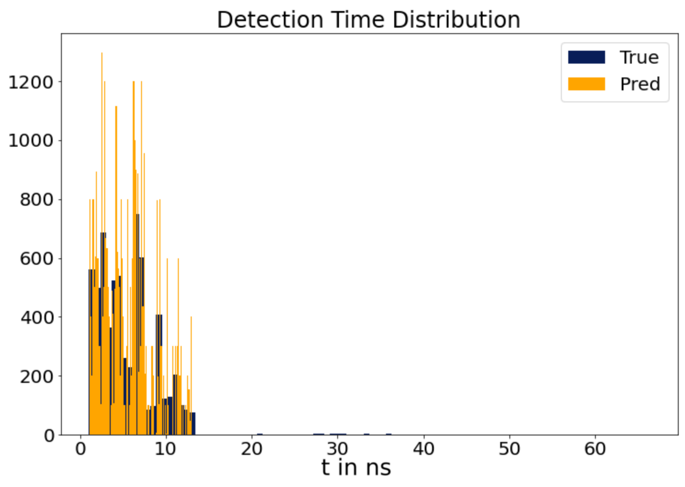

Figure 5.23: True and predicted photon detection times for 400 photons belonging to the filtered dataset. The distribution of predicted (orange) and true (blue) detection times is shown.

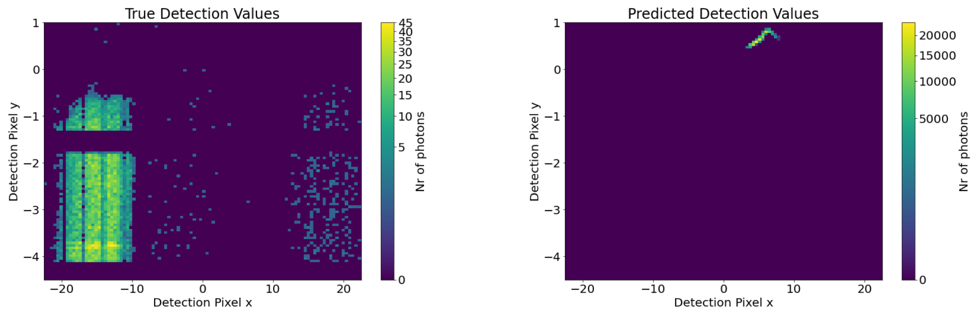

Figure 5.24: True and predicted photon detection values for 400 photons in the $x - y$-plane belonging to the unfiltered dataset. The left side shows the true detection values of random photons, while the right side shows the predicted values.

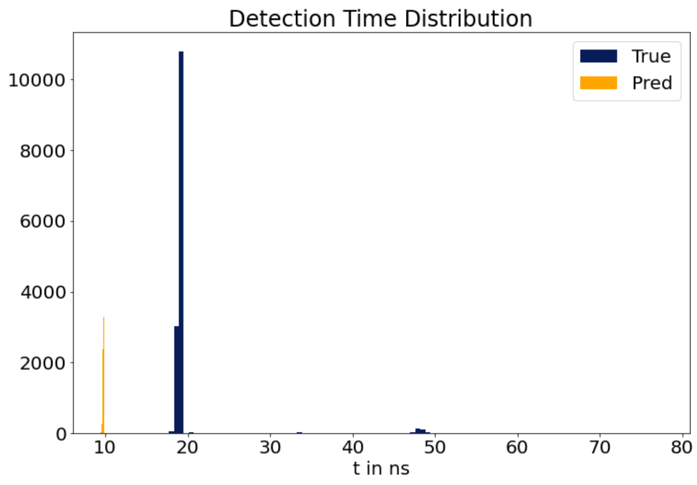

Figure 5.25: True and predicted photon detection times for 400 photons belonging to the unfiltered dataset. The distribution of predicted (orange) and true (blue) detection times

6. Conclusion

In this thesis the concept of replacing the simulation of the Time of Propagation detector at the Belle II experiment as well as validating the results of a fast simulation approach was explored. As neural network simulations gain higher publicity in high energy physics, robust and reliable validation mechanisms are needed.

For a comparison of probability distributions, which is necessary in the validation of the detector, during this thesis a high-dimensional Kolmogorov-Smirnov metric was developed (section 2.3.1).

In chapter 4 the inherent uncertainty of the current simulation by Geant4 was investigated. As simulating photons covering all possible phasespace points is not feasible, alternative validation techniques need to be found. While the premise was to find a lower limit of accordance between photon detection values generated by photons emitted with different directions or from different positions, the stochastic influence of reflections inside the detector does not allow to find this limit. Nevertheless, a correlation between the number of reflections and the probability distances between detection value distributions could be found. This correlation creates distinctive patterns of distances at certain points. A probability distance between those patterns created by Geant4 and a neural network solution can then be used as a validation mechanism.

For a high-level validation mechanism, the behaviour of particle identification depending on the energy and the inclination angle was investigated. Tracks generating Cherenkov photons are less likely to favor parts of the phasespace and can therefore be used as general validation data. As the particle identification in $basf2$ is done with a likelihood analysis, which is difficult to extract and combine with a neural network simulation, the performance of the ddKS metric for particle identification was studied. While at high energies the likelihood analysis performs slightly better, the ddKS test is still able to perform the particle identification. This result enables particle identification to be included in the validation mechanisms.

The simulation and validation of both photons and tracks is assembled into a framework. This leads to an easy evaluation of the performance of a neural network solution in comparison to Geant4. For further steps the development of a $basf2$ module to integrate a fast simulation would be beneficial to improve the validation workflow.

Additionally, different networks have been trained both on a cone model as well as on a reduced photon dataset (chapter 5). For the cone model, a conditional variational autoencoder and generative adversarial network was trained on generating correct data. The CVAE was able to robustly replicate the cone and is a promising network type for the full dataset. In comparison, a GAN trained on the cone model had difficulties to converge and could not replicate the cone model correctly. Following this result, a CVAE was also trained on a dataset containing photons with zero or one reflection inside the detector. On this reduced dataset the CVAE performed good, having an average KS distance of 0.13 on the validation set. Testing this trained model on the full dataset did not return usable results.

Several approaches to continue could be taken. One possible pathway could be breaking up the full simulation into two steps, where the number of reflections gets decided by a first network followed by a second network that is conditional on the output of the first. This could reduce the problem size and return better results. Alternatively, a CVAE could be trained and evaluated extensively on the full dataset, as the network architecture shows good behaviour on the photon data.

Bibliography

[1] B. Povh, K. Rith, C. Scholz, F. Zersche, and W. Rodejohann, *Particles and nuclei: An Introduction to the physical concepts*. Graduate Texts in Physics. Springer, 1995.

[2] **CDF**, F. Abe *et al.*, "Observation of top quark production in $\bar{p}p$ collisions," *Phys. Rev. Lett.* **74** (1995) 2626–2631, arXiv:hep-ex/9503002.

[3] **CMS**, S. Chatrchyan *et al.*, "Observation of a New Boson with Mass Near 125 GeV in pp Collisions at $\sqrt{s} = 7$ and 8 TeV," *JHEP* **06** (2013) 081, arXiv:1303.4571 [hep-ex].

[4] B. Wang, "Searches for New Physics at the Belle II Experiment," in *Meeting of the APS Division of Particles and Fields*. 11, 2015. arXiv:1511.00373 [hep-ex].

[5] **LHCb**, A. Maevskiy, D. Derkach, N. Kazeev, A. Ustyuzhanin, M. Artemev, and L. Anderlini, "Fast Data-Driven Simulation of Cherenkov Detectors Using Generative Adversarial Networks," *J. Phys. Conf. Ser.* **1525** no. 1, (2020) 012097, arXiv:1905.11825 [physics.ins-det].

[6] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, and K. Krüger, "Getting High: High Fidelity Simulation of High Granularity Calorimeters with High Speed," *Comput. Softw. Big Sci.* **5** no. 1, (2021) 13, arXiv:2005.05334 [physics.ins-det].

[7] C. Fanelli and J. Pomponi, "DeepRICH: Learning Deeply Cherenkov Detectors," *Mach. Learn. Sci. Tech.* **1** (11, 2019) 015010, arXiv:1911.11717 [physics.data-an].

[8] E. Mueller, "Applications of Quantum Mechanics," Nov, 2018. http://quantum.lassp.cornell.edu/lecture/elementary_particle_physics.

[9] M. Minsky and S. Papert, *Perceptrons; an Introduction to Computational Geometry*. MIT Press, 1969. https://books.google.de/books?id=Ow1OAQAAIAAJ.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] F. Bre, J. Gimenez, and V. Fachinotti, "Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks," *Energy and Buildings* **158** (11, 2017) .

[12] L. Bottou, "Online Algorithms and Stochastic Approximations," in *Online Learning and Neural Networks*, D. Saad, ed. Cambridge University Press, Cambridge, UK, 1998. http://leon.bottou.org/papers/bottou-98x. revised, oct 2012.

[13] Y. LeCun and Y. Bengio, *Convolutional Networks for Images, Speech, and Time Series*, p. 255–258. MIT Press, Cambridge, MA, USA, 1998.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," 2017.

[15] C. Doersch, "Tutorial on Variational Autoencoders," 2021.

[16] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," 2014.

[17] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs," *CoRR* **abs/1606.03498** (2016) , arXiv:1606.03498. http://arxiv.org/abs/1606.03498.

[18] M. Arjovsky and L. Bottou, "Towards Principled Methods for Training Generative Adversarial Networks," 2017.

[19] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," 2017.

[20] L. Devroye, "Chapter 4 Nonuniform Random Variate Generation," in *Simulation*, S. G. Henderson and B. L. Nelson, eds., vol. 13 of *Handbooks in Operations Research and Management Science*, pp. 83–121. Elsevier, 2006. https://www.sciencedirect.com/science/article/pii/S0927050706130042.

[21] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics* **22** no. 1, (1951) 79 – 86. https://doi.org/10.1214/aoms/1177729694.

[22] G. Cowan, "Statistical data analysis," 1998. http://www.gbv.de/dms/goettingen/241026571.pdfhttps://ebookcentral.proquest.com/lib/kxp/detail.action?docID=49634796;http://lib.myilibrary.com/?id=81976.

[23] A. Hagen, S. Jackson, J. Kahn, J. Strube, I. Haide, K. Pazdernik, and C. Hainje, "Accelerated Computation of a High Dimensional Kolmogorov-Smirnov Distance," 2021.

[24] A. Clim, R. D. Zota, and G. TinicĂ, "The Kullback-Leibler Divergence Used in Machine Learning Algorithms for Health Care Applications and Hypertension Prediction: A Literature Review," *Procedia Computer Science* **141** (2018) 448–453. https://www.sciencedirect.com/science/article/pii/S1877050918317939. The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2018) / The 8th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2018) / Affiliated Workshops.

[25] A. Kolmogorov, "Sulla determinazione empirica di una lgge di distribuzione," *Inst. Ital. Attuari, Giorn.* **4** (1933) 83–91.

[26] N. Smirnov, "Table for Estimating the Goodness of Fit of Empirical Distributions," *The Annals of Mathematical Statistics* **19** no. 2, (Jun, 1948) 279–281. http://projecteuclid.org/euclid.aoms/1177730256.

[27] G. Fasano and A. Franceschini, "A multidimensional version of the Kolmogorov–Smirnov test," *Monthly Notices of the Royal Astronomical Society* **225** no. 1, (Mar, 1987) 155–170. `https://academic.oup.com/mnras/article-lookup/doi/10.1093/mnras/225.1.155`.

[28] A. Hagen, J. Strube, I. Haide, J. Kahn, S. Jackson, and C. Hainje, "A Proposed High Dimensional Kolmogorov-Smirnov Distance," in *Third Workshop on Machine Learning and the Physical Sciences*. 2020. `https://ml4physicalsciences.github.io/2020/files/NeurIPS_ML4PS_2020_75_poster.pdf`.

[29] **Belle-II**, T. Abe *et al.*, "Belle II Technical Design Report," `arXiv:1011.0352 [physics.ins-det]`.

[30] **SuperKEKB**, K. Akai, K. Furukawa, and H. Koiso, "SuperKEKB Collider," *Nucl. Instrum. Meth. A* **907** (2018) 188–199, `arXiv:1809.01958 [physics.acc-ph]`.

[31] D. Sanders, Aug, 2019. `http://www.phy.olemiss.edu/HEP/belle2/index.html`.

[32] **Belle-II Barrel Particle Identification Group**, J. Fast, "The Belle II imaging Time-of-Propagation (iTOP) detector," *Nucl. Instrum. Meth. A* **876** (2017) 145–148.

[33] D. Y. Kim *et al.*, "The simulation library of the Belle II software,".

[34] R. Brun *et al.*, "root-project/root: v6.18/02," Aug., 2019. `https://doi.org/10.5281/zenodo.3895860`.

[35] T. Kuhr, C. Pulvermacher, M. Ritter, T. Hauth, and N. Braun, "The Belle II Core Software,".

[36] **GEANT4**, S. Agostinelli *et al.*, "GEANT4–a simulation toolkit," *Nucl. Instrum. Meth. A* **506** (2003) 250–303.

[37] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," 2018.

[38] C. Hainje and J. Strube, "Studies of Likelihood distributions of simulated single particles in the TOP detector,".

[39] J. Stevens *et al.*, "The GlueX DIRC Project," *JINST* **11** no. 07, (2016) C07010, `arXiv:1606.05645 [physics.ins-det]`.

[40] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[41] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," 2017.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature* **323** no. 6088, (Oct., 1986) 533–536.

[43] J. Feydy, T. Séjourné, F.-X. Vialard, S.-i. Amari, A. Trouve, and G. Peyré, "Interpolating between Optimal Transport and MMD using Sinkhorn Divergences," in *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 2681–2690. 2019.