TECHNISCHE UNIVERSITÄT DRESDEN

Technische Universität Dresden
Bereich Mathematik und Naturwissenschaften
Fakultät Physik
Institut für Kern- und Teilchenphysik

Masterarbeit
zur Erlangung des Hochschulgrades
Master of Science
im Studiengang
Physik

Investigation of machine learning enhanced integrators applied to processes in strong-field QED

vorgelegt von
Tom Jungnickel
geboren am 14.02.2000 in Rochlitz

eingereicht am 12.02.2024

Erstgutachter: Prof. Dr. Thomas E. Cowan
Zweitgutachter: Dr. Frank Siegert

Abstract

Established methods to adaptively sample differential cross sections in high energy physics are not universally applicable or suffer from low efficiencies when applied to high dimensional scattering processes. An efficient method to generate samples from differential cross sections will be especially important to simulate future experiments with high statistics, i.e., at the European XFEL. In the present thesis, we investigate Neural Importance Sampling, a machine learning enhanced method to generate proposal distributions for use in Monte Carlo integration and Monte Carlo event generation, which is universally applicable. The method is implemented in the Julia programming language and applied to the pulsed-perturbative Compton process in strong-field quantum electrodynamics and to the higher dimensional perturbative trident process in quantum electrodynamics. The resulting proposal distributions are compared with proposal distributions generated by the established VEGAS algorithm. The physical processes, as well as the machine learning enhanced method, are implemented in a vectorized way to allow for efficient computation on GPUs.

Zusammenfassung

Etablierte Methoden zur adaptiven Stichprobenentnahme von differentiellen Wirkungsquerschnitten in der Hochenergiephysik sind nicht universell anwendbar oder besitzen eine geringe Effizienz bei Anwendung auf hochdimensionale Streuprozesse. Eine effiziente Methode zur Generierung von Stichproben aus differentiellen Wirkungsquerschnitten wird insbesondere wichtig sein, um zukünftige Experimente mit hoher Statistik, z.B. am European XFEL, zu simulieren. In der vorliegenden Arbeit untersuchen wir Neural Importance Sampling, eine durch maschinelles Lernen verbesserte, universell anwendbare Methode zur Generierung einer Vorschlagsverteilung zur Verwendung in Monte Carlo Integration und Monte Carlo Event-Generation. Die Methode wird in der Programmiersprache Julia implementiert und auf den puls-perturbativen Compton-Prozess in der Starkfeld-Quantenelektrodynamik und auf den höher-dimensionalen perturbativen Trident-Prozess in der Quantenelektrodynamik angewendet. Die resultierenden Vorschlagsverteilungen werden mit jenen durch den etablierten VEGAS Algorithmus generierten verglichen. Die physikalischen Prozesse sowie die Methode werden vektorisiert implementiert, um eine effiziente Berechnung auf GPUs zu ermöglichen.

Contents

1 Introduction

Experiments in high-energy physics aim to understand the fundamental building blocks of nature and the forces that govern their interactions. Therefore, large facilities are built to accelerate particles to high energies and to collide them to gain insights into their interactions through the observation of the scattering products. Theoretical descriptions of the involved processes need to be compared to the observed events, or phase space points, to test the agreement of theory with nature.

However, scattering processes are stochastic in nature, as described by their differential cross sections, which are a measure of the probability of phase space points occurring, e.g., for the probability of the outgoing particles to scatter into certain directions with certain energies. Therefore, the comparison of experimental data with theoretical predictions requires the generation of phase space points distributed according to the differential cross section of the process of interest. Additionally, large numbers of events need to be generated to achieve precise predictions.

One facility where such a need arises is the European XFEL (European X-Ray Free-Electron Laser Facility), where a free electron laser produces highly intense and energetic X-ray pulses at a high repetition rate. Of special interest for the present thesis are planned experiments where the laser pulses interact with an electron beam. Due to the high repetition rate and high intensity of the X-ray pulses, numerical simulations of the interaction using perturbative quantum electrodynamics are not feasible, as the number of photons and, therefore, the rate of interactions becomes too large. Also, due to the influence of the background field, descriptions in classical quantum electrodynamics are not accurate anymore. However, strong-field quantum electrodynamics can be used to model the laser pulse as a classical background field, which reduces the number of Feynman diagrams that need to be evaluated.

Monte Carlo methods are used to generate events distributed according to the differential cross section of a process. However, the efficiency of these methods decreases with the number of dimensions of the phase space, leading to an exponential increase in the number of events required to achieve a certain precision. Importance sampling methods can be used to improve the efficiency of Monte Carlo methods by sampling the phase space according to a particular proposal distribution. The most commonly used algorithm for this purpose is the VEGAS algorithm. It utilizes one-dimensional projections of the phase space to adaptively improve a proposal distribution, which aims to approximate the target distributions through a set of step functions. However, this approach can cause the adap-

tion of artifacts if the target distribution does not factorize into one-dimensional distributions. As this property is often not satisfied by scattering cross sections, the efficiency of the produced proposal decreases, which is amplified in higher dimensional phase spaces.

Therefore, a new algorithm for the generation of proposal distributions is desirable. One such algorithm is presented in this thesis. Neural Importance Sampling (NIS) is a method that uses neural networks to generate proposal distributions for arbitrary target distributions. It uses a type of normalizing flow that transforms a simple base distribution into the target distribution through a series of invertible transformations that each transform a part of the phase space.

These two importance sampling methods are compared based on the weight distributions of their generated samples, as the methods are investigated within the framework of event generation. Since the methods also act as integrators, the resulting Monte Carlo integrals, as well as their errors, will be compared. However, as will be shown, the weight distributions give more insight into the performance of the algorithms than the error of the integrals can give alone.

Section 2 introduces the two investigated processes, the pulsed-perturbative Compton process and the perturbative trident process, and briefly describes the theoretical framework they are described in. Section 3 gives an overview of the Monte Carlo methods used for integration and event generation. The significance of suitable proposals for importance sampling is explained, and the VEGAS algorithm is introduced. Section 4 explains the basics of machine learning and introduces the NIS architecture. Section 5 investigates the application of NIS to Gaussian distributions to gain insights into the behavior of the methods. Afterward, VEGAS and NIS are applied to the pulsed-perturbative Compton process and the perturbative trident process. The proposal distributions generated by NIS are compared to those generated by VEGAS for each process and the performance of the algorithms are evaluated.

2 Physical Background

The simulation of scattering processes using event generators plays a crucial role in the understanding of interactions in particle physics. They allow a comparison between theoretical predictions and experimental observations. Since experiments in particle accelerators can only observe the final state of a scattering process in the form of phase space points, also called events, a simulation is required to gain an understanding of which processes lead to the observed events. Specifically, this work aims to investigate processes that may occur at the European XFEL [1], a free-electron laser that can produce high-intensity X-ray pulses. In these conceivable experiments, the XFEL is used as a driver (not a probe) and interacts with an electron beam[1]. Parameters assumed for the XFEL are given in table 2.1. The simulation of these experiments requires two key properties: First, due to the high repetition rate of the XFEL, simulations need to be highly efficient and able to generate a large number of events in a short period of time. Secondly, simulations of classical QED processes are not sufficient. As the XFEL is able to produce highly energetic and intense laser pulses with a spectrum that is not negligible, the strong-field description of QED is required to accurately describe the interactions between the laser background field and the electron beam. To describe the intensity properties of the laser light produced by the European XFEL, the classical non-linearity parameter a_0 is used [2], which can be written as:

$$a_0 = \frac{7.5eV}{\omega}\sqrt{I/10^{20}W/cm^2}\,, \tag{2.1}$$

where ω denotes the photon energy of the laser and I is its peak intensity. For an overview on laser-matter interactions and the different interpretations of a_0, see [3–5].

ω $[keV]$	λ $[nm]$	I $[W/cm^2]$	a_0	τ $[fs]$	N_{cycles}
~ 10	0.12	10^{21}	$\sim 10^{-3}$	20	$\sim 10^5$

Table 2.1: Parameters assumed for the European XFEL [1]

[1] The electrons in this electron beam that interacts with the laser should not be confused with the electrons used to produce the laser.

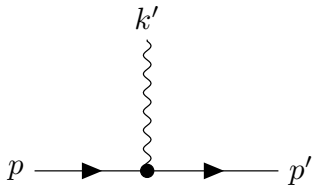
Within the present thesis, we will investigate two processes, the Compton process and the trident process. Compton scattering will be investigated in pulsed-perturbative QED, an approximation to the general strong-field QED, where bandwidth effects are included by the leading order in an a_0-expansion [6, 7]. This is a valid approach because we have $a_0 << 1$ for the European XFEL. Furthermore, as an example of a higher dimensional process, we investigate the trident pair production in perturbative QED; see [8–10] for reviews of the extensive original literature. An example spectrum will be used to model the background field, which is not the spectrum of the XFEL. This work aims to investigate methods that enhance event generation, so no exact modeling of the XFEL is required. Currently used methods for event generation are either not suitable for the strong-field description or are inefficient, so there is a strong need for new methods that can take advantage of the current advancements in hardware and software. Note that Thomson scattering is not investigated, as it is not an accurate approximation for Compton scattering at the energy levels of the XFEL, and no particle-in-cell techniques are used, as their aim is to model the dynamics of large particle systems within a classical framework, which do not include QED-effects [11]. The focus of this work is the implementation and evaluation of a new method for increasing the efficiency of Monte Carlo integration and Monte Carlo event generation taking advantage of the ability of neural networks to approximate any continuous function [12] and apply this method to the processes mentioned above.

2.1 QED-like models

2.1.1 QED

Quantum electrodynamics (QED) is a quantum field theory that describes the interaction between electrically charged particles, in our case electrons and positrons, and photons as the bosons that mediate the electromagnetic force. We want to focus on the description using Feynman rules; for a more detailed description of QED, refer to [13, 14]. The probability for a scattering process to happen at a given phase space point is given by the differential cross section, which can be calculated using the matrix element of the process. Feynman diagrams can be used to visualize the scattering processes and calculate their matrix elements using the Feynman rules [15]. Each part of a diagram corresponds with a term in the matrix element. Real incoming and outgoing particles are represented by external lines and contribute spinors. Internal lines represent virtual particles and contribute a propagator term. Vertices represent the interaction between particles

and contribute the QED vertex term given by[2]



$$= -ie\gamma^{\mu}(2\pi)^4\delta^{(4)}(p - p' - k')\,, \tag{2.2}$$

where γ^{μ} are Dirac's gamma matrices, and the delta-distributions ensure the local energy-momentum conservation, with the electron momenta[3] p and p' and the photon momentum k'.

Fermion propagator:	$\frac{i(\not{p}+m)}{p^2-m^2+i\epsilon}$
Photon propagator:	$\frac{ig_{\mu\nu}}{k^2+i\epsilon}$
QED vertex:	$-ie\gamma^{\mu}$
Incoming fermion:	$u_{\sigma p} = \frac{\not{p}+m}{\sqrt{\lvert p_0\rvert+m}}\eta_{\sigma}$
Incoming anti-fermion:	$\overline{v}_{\sigma p} = y^{\dagger}_{\sigma p}\gamma^0$
Outgoing fermion:	$\overline{u}_{\sigma p} = u^{\dagger}_{\sigma p}\gamma^0$
Outgoing anti-fermion:	$v_{\sigma p} = \frac{-\not{p}+m}{\sqrt{\lvert p_0\rvert+m}}\chi_{\sigma}$
Incoming photon:	$\varepsilon^{\mu}_{\lambda}(k)$
Outgoing photon:	$\varepsilon^{*\mu}_{\lambda}(k)$

Table 2.2: Feynman rules for monochromatic QED, where p is the momentum of the fermion/anti-fermion and k is the momentum of the photon. This table is taken from [16]; an implementation can be found in [17].

To calculate the matrix element, one can proceed as follows: Move along one of the fermion lines in the opposite direction of the arrows and apply the rules for each diagram component. Repeat the steps for other fermion lines and obtain the propagator term. Finally, multiply all terms to obtain the matrix element of the diagram and average the absolute squared matrix elements over all spins and polarizations to obtain the unpolarized differential cross section. The required terms for each element of QED Feynman diagrams are given in table 2.2, where the Feynman slash notation $\not{k} := \gamma^{\mu}k_{\mu}$ is used. Diagrams of higher orders are possible for each process involving the addition of loops; only tree-level diagrams without loops are considered in the present thesis.

2.1.2 Strong-field QED

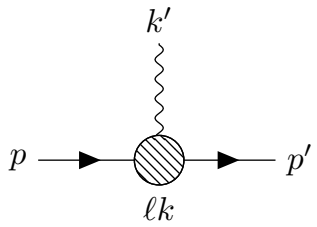
Quantum electrodynamics is able to accurately describe the interactions between electrically charged particles and photons. However, with the high intensity of

[2] In the present thesis, we use natural units, where $\hbar = c = 1$.

[3] Unless stated otherwise, the term *momentum* is used to refer to four-momenta in the present thesis.

lasers, such as at the XFEL, the number of coupled external photons and, therefore, the number of Feynman diagrams that need to be evaluated becomes very large, and the calculation of the matrix elements becomes unfeasible. Therefore, the use of the strong-field description of QED, where the single photons of the laser are replaced with a classical background field, is worthwhile. The following description is taken as given for the investigations presented in the present thesis and is taken from [6] and [16].

One can transition from the perturbative QED description to the strong-field QED description by exchanging the perturbative QED vertex given in equation (2.2) by a dressed vertex given by



$$= -ie\Gamma^{\mu}(\ell, p, p', k)(2\pi)^4\delta^{(4)}(p + \ell k - p' - k')\,, \quad (2.3)$$

with the dressed vertex function

$$\Gamma^{\mu}(\ell|p, p', k) = \Gamma_0^{\mu} B_0(\ell) + \Gamma_1^{\mu\nu} B_{1\nu}(\ell) + \Gamma_2^{\mu} B_2(\ell)\,, \quad (2.4)$$

where ℓ denotes the photon number parameter, which represents the equivalent number of photons absorbed from the background field; it is not necessarily an integer. The dressed vertex represents the interaction with the background field where energy and momentum is exchanged. The elementary vertices are given as

$$\Gamma_0^{\mu} := \gamma^{\mu} \quad (2.5)$$

$$\Gamma_1^{\mu\nu} := e\left(\frac{\gamma^{\mu}\not{k}\gamma^{\nu}}{2(kp)} + \frac{\gamma^{\nu}\not{k}\gamma^{\mu}}{2(kp')}\right) \quad (2.6)$$

$$\Gamma_2^{\mu} := -e^2\frac{\not{k}}{2(kp)(kp')}k^{\mu}\,. \quad (2.7)$$

The laser itself is described by a classical four-vector potential $A^{\mu}(\phi = kx)$. The phase integrals are defined as

$$B_0(\ell|p, p', k) := \int_{-\infty}^{\infty} \mathrm{d}\phi \exp(i\ell\phi + iG(\phi|p, p', k))\,, \quad (2.8)$$

$$B_1^{\mu}(\ell|p, p', k) := \int_{-\infty}^{\infty} \mathrm{d}\phi A^{\mu}(\phi) \exp(i\ell\phi + iG(\phi|p, p', k))\,, \quad (2.9)$$

$$B_2(\ell|p, p', k) := \int_{-\infty}^{\infty} \mathrm{d}\phi A^{\mu}(\phi)A_{\mu}(\phi) \exp(i\ell\phi + iG(\phi|p, p', k))\,, \quad (2.10)$$

$$(2.11)$$

with the abbreviation

$$G(\phi|p,p',k) := \alpha_1^\mu(p,p',k)\int_0^\phi \mathrm{d}\phi' A_\mu(\phi') + \alpha_2(p,p',k)\int_0^\phi \mathrm{d}\phi' A^2(\phi') \quad (2.12)$$

using the kinematic factors

$$\alpha_1^\mu(p,p',k) := e\left(\frac{p'^\mu}{kp'} - \frac{p^\mu}{kp}\right), \qquad \alpha_2(p,p',k) := e^2\left(\frac{1}{kp} - \frac{1}{kp'}\right). \quad (2.13)$$

This description applies to any plane wave background field A^μ.

The strong-field QED vertex can be expanded in orders of the laser intensity parameter a_0. For small a_0, the leading order is sufficient to describe the interaction, which is the case for the XFEL with $a_0 \sim 10^{-3}$. The expansion is given by

$$\Gamma^\mu(l) = \Gamma^\mu_{\mathrm{pp}} + \mathcal{O}(a_o^2), \quad (2.14)$$

with the pulsed-perturbative QED vertex

$$\Gamma^\mu_{\mathrm{pp}} = \pi\delta(\ell)2\Gamma^\mu + a_0\left(\Gamma_1^{\mu\nu} - \Gamma_0^\mu \not{p}\frac{\not{\alpha}_1}{\ell}\right)\varepsilon_\nu F(\ell) \quad (2.15)$$

$$=: \delta(\ell)\tilde{\Gamma}^\mu_0 + a_0\tilde{\Gamma}^\mu_{\mathrm{pp}}(\ell,p,p'), \quad (2.16)$$

where $F(\ell) = \int \mathrm{d}\phi g(\phi)\cos\phi\exp(i\ell\phi)$ with the pulse envelope function $g(\phi)$, i.e. we assume a linear polarized background field given by $A^\mu(\phi) = \varepsilon^\mu g(\phi)\cos\phi$. Within the present thesis, we use $g(\phi) = \cos^2(\frac{\pi\phi}{2\Delta\phi})$ as the pulse envelope, refered to as $\cos^2$-pulse in the following. We refer to $|F(\ell)|^2$ as the generic spectrum, which is shown in Fig. 2.1.

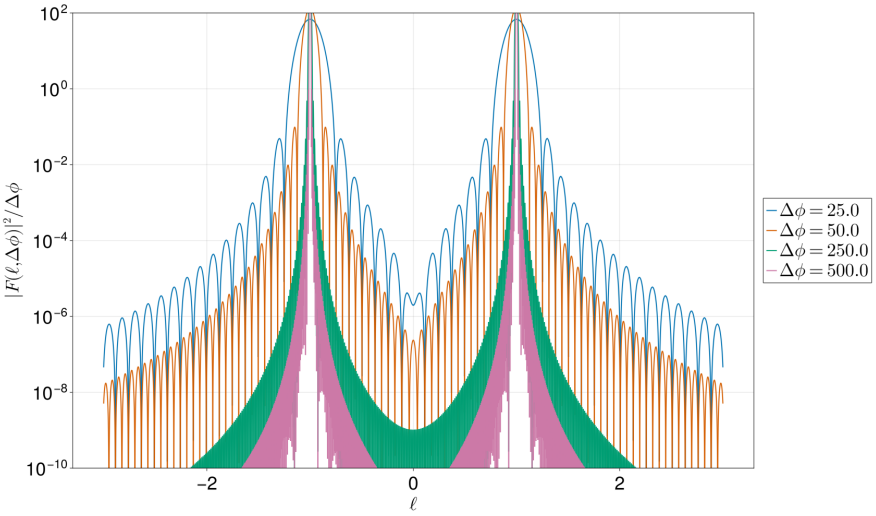
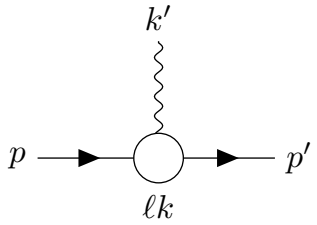


Figure 2.1: Generic spectrum as a function of ℓ for different values of $\Delta\phi$.

This pulsed-perturbative QED vertex can be used to build Feynman diagrams analogous to the perturbative QED case, where only a different vertex is used. The pulsed-perturbative QED vertex is given by



$$= -ie\Gamma^{\mu}_{\mathrm{pp}}(\ell, p, p')(2\pi)^4\delta^{(4)}(p + \ell k - p' - k')\,. \quad (2.17)$$

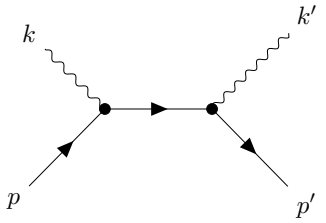
Consequently, the differential cross section in the pulsed-perturbative QED description can be obtained as

$$\mathrm{d}\sigma_{\mathrm{pp}} := \lim_{a_0\to 0} \mathrm{d}\sigma \equiv \lim_{a_0\to 0} \mathrm{d}\sigma\Big|_{\Gamma^{\mu}\to\Gamma^{\mu}_{\mathrm{pp}}}\,. \quad (2.18)$$

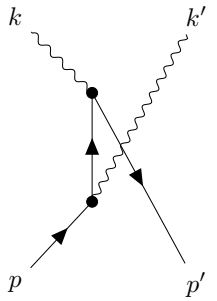
2.2 Compton process

2.2.1 Perturbative QED case

$iM_{\mathrm{C}} =$



$+$



(2.19)

A simple, non-trivial QED-process is the Compton process, which involves one electron and one photon, which scatters off the electron. Feynman diagrams can be used to calculate their matrix elements; each process may have many possible diagrams. The matrix element of the Compton process in perturbative QED is calculated from the two tree-level Feynman diagrams as given by equation (2.19). In the present thesis, this process will be investigated on tree-level in pulsed-perturbative QED, as it allows the study of bandwidth effects of the incoming light. The four-momenta of the incoming and outgoing electron are denoted by p and p', and the electron four-momenta of the incoming and outgoing photon are denoted by k and k' respectively. There are two possible diagrams for this process: the s-channel and the u-channel (the first and second terms in equation (2.19) respectively), named after the Mandelstam variables of the involved propagators.

These are defined as:

$$s = (p+k)^2 = 2pk + m^2 = 2p'k' + m^2 \tag{2.20}$$
$$t = (p'-p)^2 = -2pp' + 2m^2 = -2kk' \tag{2.21}$$
$$u = (k'-p)^2 = -2pk' + m^2 = -2kp' + m^2\,. \tag{2.22}$$

The frame of reference is chosen as the rest frame of the electron with the z-axis aligned with the momentum of the incoming photon. In spherical coordinates, the involved momenta can be expressed as:

$$k = (\omega, 0, 0, \omega)^T \tag{2.23}$$
$$p = (m_e, 0, 0, 0)^T \tag{2.24}$$
$$k' = (\omega', \omega' \sin\theta\cos\varphi, \omega' \sin\theta\sin\varphi, \omega'\cos\theta)^T \tag{2.25}$$
$$p' = (\omega + m_e - \omega', -\omega' \sin\theta\cos\varphi, -\omega' \sin\theta\sin\varphi, \omega - \omega'\cos\theta)^T\,, \tag{2.26}$$

Where p' results from the conservation of four-momentum.

In the perturbative QED case, and if the incident electron is at rest, it can be shown that the energy of the outgoing photon ω' is given by

$$\omega' = \frac{\omega}{1 + \frac{\omega}{m_e}(1-\cos\theta)} \tag{2.27}$$

with the energy of the incoming photon ω, the electron mass m_e and the polar angle θ (see A.1 for a derivation). This means that for a fixed energy ω, the phase space of the described Compton process is one-dimensional. The lower bound for the incoming photon energy is zero, so the Compton process is not limited by the energy of the incoming photon.

A measure of the probability that the Compton process happens for any given phase space point is the differential cross section. The calculation of the latter often involves long computations of matrix elements. But in the case of perturbative Compton scattering at tree-level, an analytical solution exists, which is given by the Klein-Nishina formula:

$$\frac{\mathrm{d}\sigma_{pc}}{\mathrm{d}\cos\theta \mathrm{d}\varphi} = \frac{1}{2}\frac{\alpha^2}{m^2}\left(\frac{\omega'}{\omega}\right)^2\left(\frac{\omega'}{\omega} + \frac{\omega}{\omega'} - \sin^2\theta\right)\,. \tag{2.28}$$

Fig. 2.2 shows the Klein-Nishina formula for different incoming photon energies. For low energies, the cross section is almost symmetric around $\cos\theta = 0$ with two local maxima at $\cos\theta = -1.0$ and $\cos\theta = 1.0$. The maximum at $\cos\theta = 1.0$ stays the same for all photon energies, which is the elastic case of the Compton process where the energy of the photon stays unchanged (see (2.27)). For higher energies, the process is suppressed for all angles except $\cos\theta = 1$, where the suppression is strongest at $\cos\theta = -1.0$ for high energies. This differential cross section can be integrated by using simple numerical integration methods like Gaussian quadrature or solved analytically.

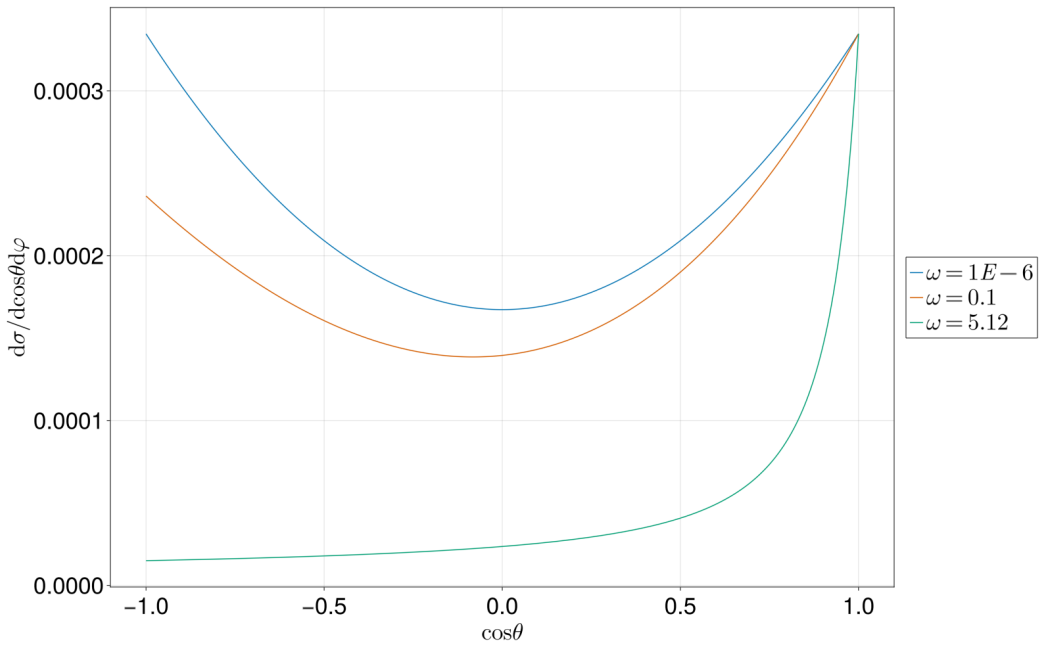


Figure 2.2: Differential cross section of the Compton process depicted as a function of $\cos\theta$ for different incoming photon energies.

2.2.2 Pulsed-Perturbative QED case

In the application of interest (see Sec. 1), a strong-field description is required to include the bandwidth of the XFEL. This also changes the description of the Compton process as described in 2.1.2.

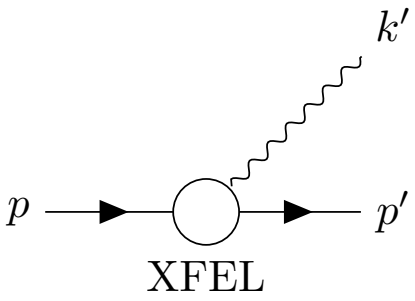


Figure 2.3: Feynman diagram of the pulsed-perturbative Compton process.

Since the incoming photon is replaced by an interaction with the background field, the Feynman diagram only shows an outgoing photon, and the classical QED-vertex is replaced by the dressed vertex, as is shown in Fig. 2.3. Sandwiching the pulsed-perturbative vertex (equation (2.17)) between the spinors of the incoming

and outgoing electron, one can show that the differential cross section reads:

$$\frac{\mathrm{d}\sigma_{ppc}}{\mathrm{d}\ell d\cos\theta} = N\frac{\mathrm{d}\sigma_{pc}}{\mathrm{d}\cos\theta}\bigg|_{\omega=\omega\ell} \cdot |F(\ell)|^2 \,. \tag{2.29}$$

This means that the differential cross section for the pulsed-perturbative case is equal to the Klein-Nishina formula, where ω is replaced by $\omega\ell$ and the expression is multiplied with the generic spectrum $|F(\ell)|^2$. For narrow pulses, ℓ is favored, and the pulsed-perturbative cross section transitions to the Klein-Nishina formula. Also, N represents a constant, which will be set to one in the following because the focus of the present thesis lies on the investigation of sampling methods for scattering cross sections rather than the processes themselves. Since the presence of the background field adds an additional degree of freedom, the photon-number parameter ℓ, the phase space of pulsed-perturbative Compton scattering on tree-level is two-dimensional, in contrast to the one-dimensional perturbative Compton cross section.

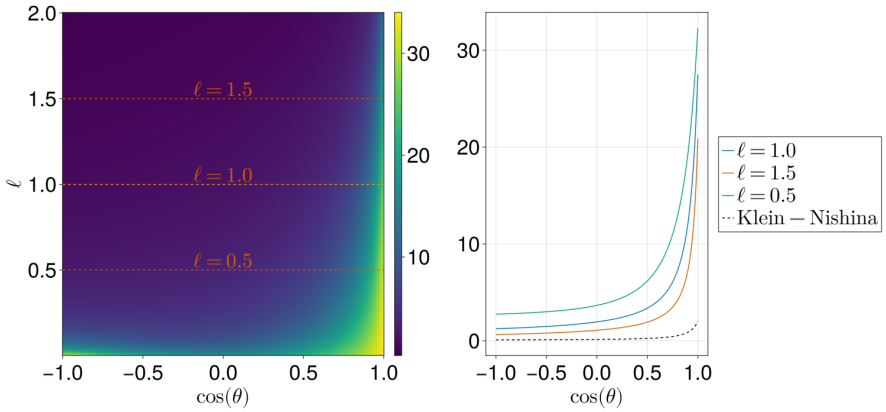


Figure 2.4: Differential cross section of the pulsed-perturbative Compton process in units of Nm_e^2 as a function of ℓ and $\cos\theta$ with slices evaluated at $\ell = 0.5$, $\ell = 1.0$, $\ell = 1.5$ and the Klein-Nishina formula. Background field parameters: $\omega = 5.12\ m_e$, $\Delta\phi = 1.43$, pulse shape = $\cos^2$-pulse.

Fig. 2.4 shows the differential cross section of the pulsed-perturbative Compton process as a function of $\cos\theta$ and ℓ for the parameters given in the caption of the figure. First, we observe that evaluating the function at a slice at $\ell = 1$ results in values below those given by the Klein-Nishina formula. As the process is influenced by the background field, it does not correspond with the perturbative description anymore. However, for a very narrow pulse as the background field, the resulting differential cross section will approach the perturbative solution. Furthermore, the pulsed-perturbative cross section shows two local maxima at $\cos\theta = 1.0$ and $\cos\theta = -1.0$ similar to the Klein-Nishina formula. However, caused by the ℓ-dependence, a deformation of the distribution is visible. The peak at $\cos\theta = 1.0$

does not change much with increasing ℓ and stays constant at exactly $\cos\theta = 1.0$. The left peak near $\cos\theta = -1.0$ changes in size depending on ℓ. At $\ell \to 0$, the differential cross section stays the same independently of ω since equation (2.29) only contains $\omega\ell$ terms.

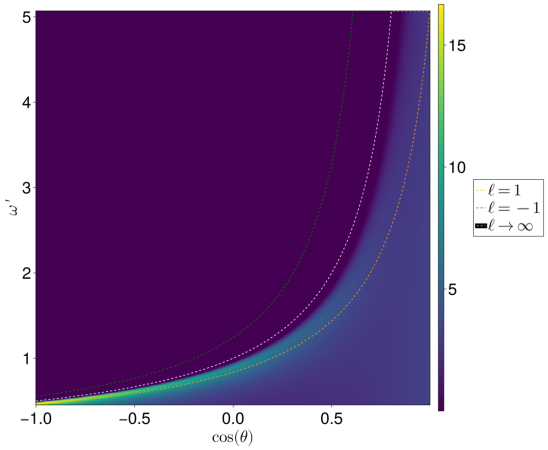


Figure 2.5: Differential cross section of the pulsed-perturbative Compton process in units of Nm_e as a function of ω' and $\cos\theta$ with slices evaluated at different values of ℓ. Phase space points above the $\ell \to \infty$ line are physically not possible. Background field parameters: $\omega = 5.12\ m_e$, $\Delta\phi = 1.43$, pulse shape = $\cos^2$-pulse.

In the current form, the differential cross section depends on ℓ, a value that can not be measured directly in the experiment. A description using measurable quantities is required to build simulations that are comparable with the experiments. Therefore, a transformation to the energy ω' of the outgoing photon is reasonable. First, performing the mentioned replacement $\omega \to \omega\ell$ on (2.27) and rearranging leads to an expression for ℓ using only known quantities:

$$\omega'(\ell) = \frac{\omega\ell}{1 + \frac{\omega\ell}{m}(1 - \cos\theta)}\,, \tag{2.30}$$

$$\implies \qquad \ell = \frac{\omega'}{\omega}\frac{1}{1 - \frac{\omega'}{m}(1 - \cos\theta)}\,. \tag{2.31}$$

This expression can be used to obtain the required Jacobian determinant for the transformation; see appendix A.2 for details.

$$\begin{aligned}\frac{\mathrm{d}\sigma_{ppc}}{\mathrm{d}\omega'\mathrm{d}c\theta} &= \frac{\mathrm{d}\sigma_{ppc}}{\mathrm{d}\ell\mathrm{d}c\theta}\left|\frac{\mathrm{d}\omega'\mathrm{d}c\theta}{\mathrm{d}\ell\mathrm{d}c\theta}\right|^{-1} \\ &= \frac{\mathrm{d}\sigma_{pc}}{\mathrm{d}c\theta}\bigg|_{\omega=\omega\ell} |F(\ell)|^2 \frac{(m + \omega\ell(1 - c\theta))^2}{2\omega m^2}\,,\end{aligned} \tag{2.32}$$

which is an expression for the differential cross section of the pulsed-perturbative Compton process that only uses constants (m_e), known parameters (ω), and measured obersavables (ω', θ). In Fig. 2.5, the differential cross section is shown for the same parameters as Fig. 2.4.

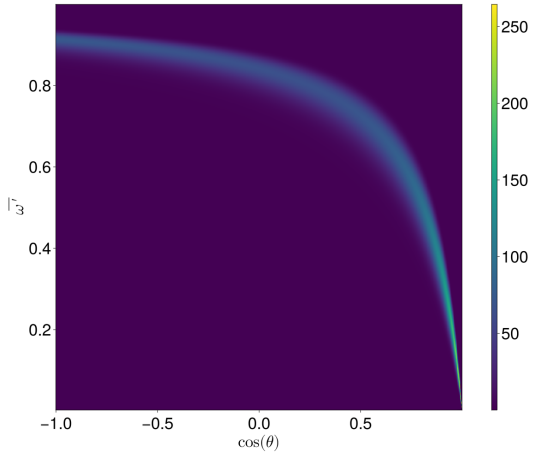


Figure 2.6: Differential cross section of the pulsed-perturbative Compton process in units of Nm_e^2 as a function of $\overline{\omega'}$ and $\cos\theta$. Background field parameters: $\omega = 5.12\ m_e$, $\Delta\phi = 1.43$, pulse shape = $\cos^2$-pulse.

This form of the differential cross section shows some new features that were not visible in the ℓ-dependant form. There is a particular region where the differential cross section is zero because those phase space points are not allowed due to energy-momentum conservation. Inspecting the behavior of different ℓ-lines gives insights into the cause. Lowering ℓ moves the line to the bottom right (lower ω', higher $\cos\theta$) while increasing ℓ moves it to the top left. For $\ell \to \infty$ it approaches the line indicated in white. This is the phase space limit of ω' for a given $\cos\theta$. For ω' beyond this limit, the resulting ℓ would be negative, implying an emission of a photon by the resting electron into the background field. However, since it is kinematically not allowed that an electron at rest only emits photons, the differential cross section vanishes for these phase space points.

The limit can be obtained from equation (2.30):

$$\begin{aligned} \omega'_{ex} &= \lim_{\ell\to\infty} \omega'(\ell)\,, \\ \implies \omega'_{ex} &= \frac{m_e}{1-\cos\theta}\,. \end{aligned} \tag{2.33}$$

The region below the limit shall be sampled to perform Monte Carlo integration or Monte Carlo event generation (see Sec. 3). For both cases, it is beneficial if this domain is rectangular. Therefore, a transformation is performed to scale each ω' to the maximum possible value by defining

$$\overline{\omega'} = \frac{\omega'}{\omega'_{ex}}\,. \tag{2.34}$$

Using the associated Jacobian determinant (see appendix (A.3)), the transformation is given by

$$\begin{aligned}\frac{\mathrm{d}\sigma_{ppc}}{\mathrm{d}\overline{\omega'}\mathrm{d}c\theta} &= \frac{\mathrm{d}\sigma_{ppc}}{\mathrm{d}\omega'\mathrm{d}c\theta}\left|\frac{\mathrm{d}\overline{\omega'}\mathrm{d}c\theta}{\mathrm{d}\omega'\mathrm{d}c\theta}\right|^{-1} \\ &= \frac{\mathrm{d}\sigma_{ppc}}{\mathrm{d}\omega'\mathrm{d}c\theta}\frac{m}{1-c\theta+\frac{m}{\omega'}}\,. \end{aligned} \tag{2.35}$$

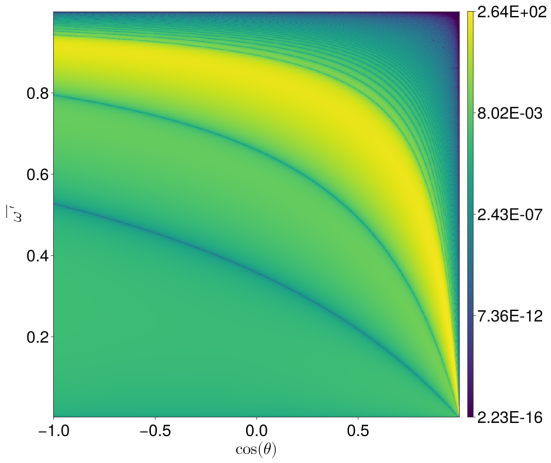


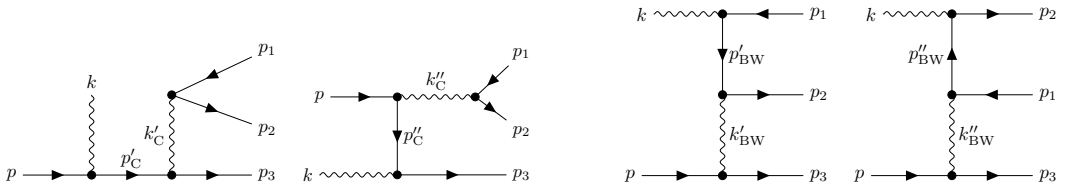
Figure 2.7: Differential cross section of the pulsed-perturbative Compton process in units of Nm_e^2 as a function of $\overline{\omega'}$ and $\cos\theta$ with a logarithmic color scale. Background field parameters: $\omega = 5.12\ m_e$, $\Delta\phi = 1.43$, pulse shape = $\cos^2$-pulse.

This gives the final form of the differential cross section, which will be investigated in the present thesis. It is shown in Fig. 2.6 for the same parameters as in Fig. 2.5. The differential cross section features a narrow band and has a value close to zero in the rest of the domain.

Also, the pulsed-perturbative Compton cross section dynamically depends on the spectrum of the background field. Caused by the connection of the photon number parameter and the external momenta, the fringe patterns from the generic spectrum are imprinted onto the phase space, as can be observed in Fig. 2.7, where the pulsed-perturbative Compton cross section is shown using a logarithmic color scale.

2.3 The Perturbative Trident process

The trident process will be investigated as the second physical process in this work because its phase space has a higher dimensionality compared to the Compton process. Another difference is that this process has a kinematic threshold, i.e., it is not possible for center-of-momentum energies below three electron masses.



(a) Virtual Compton diagrams. (b) Virtual Breit-Wheeler diagrams.

Figure 2.8: The first four tree-level Feynman-diagrams of the trident process in perturbative QED. There are eight diagrams in total, where the remaining four diagrams are obtained by exchanging the outgoing electrons.

However, the European XFEL at photon energies of 10 keV could be combined with a $\sim$ 50 MeV electron beam to reach the trident threshold. Effects of the bandwidth of the background field on the trident process are beyond the scope of this work. The majority of the following section is collected from [16].

Four of the tree-level diagrams for the trident process in perturbative QED are shown in Fig. 2.8. The remaining four diagrams are obtained by exchanging the two outgoing electrons: $p_2 \leftrightarrow p_3$. The trident process involves an incoming photon and electron, an outgoing positron, and two outgoing electrons. Comparing the first two diagrams to the Compton process (cf. Fig. 2.19), it can be seen that these diagrams correspond with the Compton process, where the outgoing photon is virtual and decays into an electron-positron pair. Therefore, these diagrams are also referred to as virtual Compton diagrams. Similarly, the third and fourth Feynman diagrams in Fig. 2.8 correspond with the electron-positron pair creation from two photons (also known as the Breit-Wheeler process), where one of the photons couples virtually to a fermion line. Therefore, these diagrams are also referred to as virtual Breit-Wheeler diagrams. The sum of the matrix elements for the first two diagrams in Fig 2.8 is given by

$$M_{\mathrm{C}} = -ie^3 \frac{g_{\mu\tau}}{(p_1+p_2)^2 + i\epsilon} [\overline{u}(p_3)\mathcal{C}^{\mu\nu}(p, p_3|k)u(p)] \times [\overline{u}(p_2)\gamma^\tau v(p_1)]\varepsilon_\nu \,, \quad (2.36)$$

with the electron charge e, the Minkowsi metric $g_{\mu\nu} = \mathrm{diag}(1, -1, -1, -1)$, and the Compton tensor $\mathcal{C}^{\mu\nu}$, which is given by

$$\mathcal{C}^{\mu\nu}(q, q_2|k) = \frac{\gamma^\mu(\not{q}_1 + \not{k} + m)\gamma^\nu}{(q_1+k)^2 - m^2 + i\epsilon} + \frac{\gamma^\nu(\not{q}_2 - \not{k} + m)\gamma^\nu}{(q_2-k)^2 - m^2 + i\epsilon} \,, \quad (2.37)$$

where q_1 and q_2 are four-momenta and m is the electron mass. The sum of the matrix elements of the third and fourth diagrams is given by

$$M_{\mathrm{BW}} = -ie^3 \frac{g_{\mu\nu}}{(p+p_3)^2} [\overline{u}(p_3)\gamma^\mu u(p)] \times [\overline{u}(p_2)\mathcal{C}^{\nu\tau}(-p_1, p_2|k)v(p_1)]\varepsilon_\tau \,. \quad (2.38)$$

Caused by indistinguishability, each diagram in Fig. 2.8 has a corresponding diagram where the outgoing electrons are exchanged:

$$M_{\mathrm{Cx}} = M_C(p_2 \leftrightarrow p_3) \tag{2.39}$$

$$M_{\mathrm{BWx}} = M_{BW}(p_2 \leftrightarrow p_3)\,. \tag{2.40}$$

Consequently, the matrix element of the trident process on tree-level reads

$$M_{\mathrm{T}} = \frac{1}{2}(M_{\mathrm{C}} - M_{\mathrm{Cx}} + M_{\mathrm{BW}} - M_{\mathrm{BWx}})\,, \tag{2.41}$$

where the relative minus sign represents the Fermi statistic of the exchanged final electrons. The polarization and spin averaged differential cross section for the trident process on tree-level is given as

$$d\sigma_T = \frac{1}{4I}\frac{1}{N}\sum_{\mathrm{spin,pol}} |M_{\mathrm{T}}|^2 (2\pi)^4 \delta^{(4)}(p + k - p_1 - p_2 - p_3) d\Phi_3\,, \tag{2.42}$$

where $I = kp$ denotes the incident energy flux and $N = \frac{1}{8}$ is the normalisation factor caused by the averaging over spins and polarizations.

3 Monte Carlo Methods

Monte Carlo methods are a valuable tool in the field of particle physics and beyond, as they allow working with complex, high-dimensional problems. In particular, for the integration of high-dimensional functions, applying numerical integration methods like quadrature will lead to an exponential number of required function evaluations with increasing dimensionality[1] - formally known as the curse of dimensionality.

3.1 Monte Carlo Integration

Consider an square-integrable n-dimensional function $f(u_1, ..., u_n)$ and let $x = (u_1, ..., u_n)$. The integral of $f(x)$ over the unit hypercube $\Omega = [0, 1]^n$ is given by

$$I = \int_\Omega f(u_1, ..., u_n) \, \mathrm{d}^n u \, . \tag{3.1}$$

and by Monte Carlo integration, it can be approximated using the estimator

$$I_N[f] = V \frac{1}{N} \sum_{i=1}^{N} f(x_i) \quad x_i \sim U[\Omega] \, , \tag{3.2}$$

where random numbers $x_i \in \Omega$ are uniformly drawn and $V = \int_\Omega \mathrm{d}^n u$ is the volume of Ω. Since $V = 1$ for the unit hypercube, it will be suppressed in this section. For large N, this approximation will always converge to the true integral value:

$$I_{N \to \infty}[f] = I \, . \tag{3.3}$$

Consider the sample variance $\sigma^2(f)$ as an unbiased estimate of the variance:

$$\sigma^2(f) \approx \frac{1}{N-1} \sum_{i=1}^{N} (f(x_i) - I_N)^2 \, , \tag{3.4}$$

which leads to the variance of the integral estimate

$$\sigma^2(I_N[f]) \approx \frac{1}{N^2} \sum_{i=1}^{N} \sigma^2(f) = \frac{\sigma^2(f)}{N} \tag{3.5}$$

[1] For example, using the trapezoidal rule, the error scales as $N^{-2/d}$ where n is the number of function evaluations and d is the dimension [18].

and the estimate of the error of the integral is thus

$$\sigma(I_N[f]) \approx \frac{\sigma(f)}{\sqrt{N}}\,. \tag{3.6}$$

The error scales with $1/\sqrt{N}$ and is independent of the problem's dimensionality. Although this solves the problem of an exponential increase of evaluations with dimensionality that quadrature and similar integration techniques suffer from, it still means the error will only slowly decrease with increasing N and in any real-world scenario where $N \to \infty$ is not feasible. For instance, in particle physics, the evaluation of a differential cross section can be very costly. Therefore, a reduction of $\sigma(I_N[f])$ is required to achieve an accurate estimate of the integral[2]. This is especially important in our case, where f is the differential cross section of a scattering process, and an accurate value of the total cross section is required.

3.1.1 Importance Sampling - Integration

Importance sampling is a method to achieve a reduction in variance by replacing f itself with f/g where g is a proposal distribution. If the proposal distribution g is as similar as possible to f, the integrand becomes almost constant, and the variance vanishes. And since the error of the integral estimate (3.6) scales with the variance of the integrand, a constant integrand will lead to a vanishing error. This also helps with another problem with higher dimensions. With increasing n, uniformly distributed points in Ω become more and more sparse. If f has sharp peaks in certain regions, it will become unlikely to 'find' them in high dimensions with classical Monte Carlo integration. But if f can be rescaled to an almost constant function, this issue of higher dimensions can be alleviated and for a constant integrand, the integral approximation becomes exact for a single sample. The integral can be rewritten by a change of integration variables [18]:

$$\int f(x)dx = \int \frac{f(x)}{g(x)} g(x)dx = \int \frac{f(x)}{g(x)} dG(x) \tag{3.7}$$

with the jacobian

$$g(x) = \frac{\partial^d}{\partial x_1, ...\partial x_d} G(x)\,. \tag{3.8}$$

Using a set of N random samples x_i drawn from the distribution $P(x)$, the integral can be estimated by

$$I_N[f] = \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{g(x_i)} \quad x_i \sim g(x)\,. \tag{3.9}$$

[2]The error σ should not be confused with the σ used for scattering cross sections

This way, the target function will be sampled more precisely at regions with a large value of $f(x)$, and fewer samples are required to achieve a certain accuracy. Therefore, finding an appropriate proposal distribution p can significantly reduce the required samples and, in turn, function evaluations of f. So, for costly f, investing some effort into finding a good proposal distribution p is worthwhile.

3.2 Monte Carlo Event Generation

Besides calculating the total cross section, the generation of events (phase space points) that are distributed like the differential cross section is an important task in the simulation of scattering processes. This can be achieved by using the rejection method [19].

Let $f(x)$ be the target distribution that the generated events should follow. First, uniformly distributed points x are generated. Some of these points shall be discarded to leave a remaining distribution that follows the target distribution in a process called unweighting. Weights are calculated for each x_i. The weights are chosen as the value of the target distribution at these points:

$$w_i := f(x_i)\,. \tag{3.10}$$

These weights are normalized to the maximum weight w_{max}:

$$w_{i,\mathrm{rel}} := \frac{w_i}{w_{\mathrm{max}}}\,, \tag{3.11}$$

and another set of random numbers u_i is picked uniformly:

$$u_i \sim U[0,1]\,. \tag{3.12}$$

The normalized weight is compared with the random number u_i to decide if a sample should be accepted or rejected as described by algorithm 1. This means

Algorithm 1 Unweighting condition

```
if u_i < w_{i,rel} then
    accept x_i
else
    reject x_i
end if
```

that for large weights, there is a high probability that the corresponding samples will be accepted.

In Fig 3.1, this is illustrated for a one-dimensional example. Depicted are generated samples x_i as well as the value of $w_{\mathrm{rel}}(x)$, which is equivalent to the normalized target distribution.

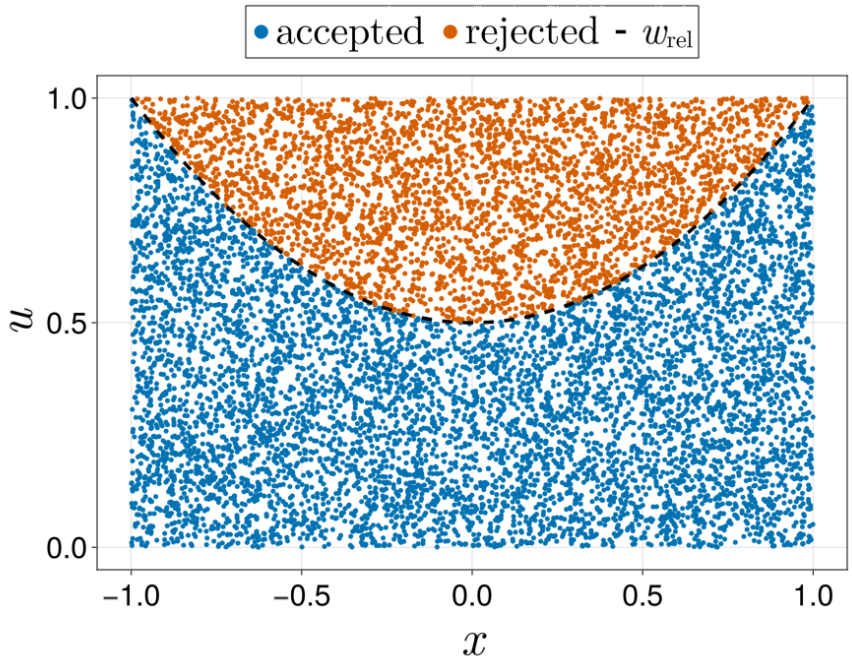


Figure 3.1: Samples for x have been generated for an example function and unweighted according to algorithm 1 with generated random numbers $u_i \in [0, 1]$ for each sample. Shown are the accepted (blue) and rejected (orange) samples at positions (u_i, x_i) as well as the the x-dependent relative weight function $w_{i,\text{rel}}$ which is equal to the normalized differential cross section.

For x, random samples x_i have been uniformly drawn from [-1,1]. Respective $u_i \sim U[0, 1]$ have also been generated for each x_i (see 3.12). These samples are depicted in 3.1 at positions (x_i, u_i). Each sample has a weight $w_{i,\text{rel}}$ assigned, and according to algorithm 1, they are accepted or rejected.

Near $x = -1.0$ and $x = 1.0$, the distribution has its maximum where the most samples should be accepted. At $x = 0.0$, the normalized distribution has a value of 0.5, so half the samples should be rejected, which is ensured by half of the uniformly drawn u being smaller than 0.5. As can be seen, all samples with $u_i < w_{i,\text{rel}}$ are accepted (blue), and the resulting distribution of samples matches the target distribution. For higher dimensions, the method works straightforward.

The efficiency of this method can be quantified by the unweighting efficiency

$$\epsilon := \frac{\mathbb{E}[w_i]}{w_{\max}} \leq 1\,, \tag{3.13}$$

representing the ratio of accepted samples to generated samples. This efficiency should be as close to 1 as possible to reduce the number of costly evaluations of the target function (differential cross section), which is the case when all weights are equal.

3.2.1 Importance Sampling - Event generation

Similar to Sec. 3.1.1, a proposal g can be utilized to increase the efficiency of Monte Carlo event generation. After drawing samples x_i from g, the weights are transformed to new weights

$$\tilde{w}_i = \frac{w(x_i)}{g(x_i)}, \quad x_i \sim g\,. \tag{3.14}$$

If g is similar to the target distribution up to some constant factor c and $w(x) = f(x)$, we have

$$w(x) \approx cg(x) \Leftrightarrow \tilde{w} = \frac{w(x)}{g(x)} \approx c\,. \tag{3.15}$$

and the new weights will be constant. Calculating the unweighting efficiency (3.13) with these new weights leads to $\epsilon \approx 1$. This gives a high incentive to find a proposal distribution g that matches the target f as closely as possible up to a constant. The proposal g will be used to generate phase space points which will eventually be used to investigate the adaption methods analyzed in the present thesis. These generated points will be referred to as samples, but they should not be confused with samples that will be used directly in simulations, which are also called unweighted events. For simulations, the weighted proposal samples need to be unweighted first because, in nature, all measured events have a unit weight, and since the proposal distribution g will not exactly match the target distribution f in practice, g will be biased.

3.3 Established proposal generation: VEGAS

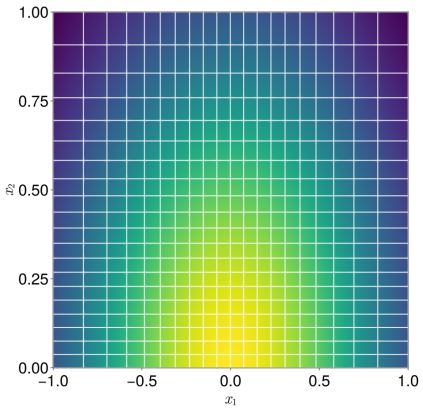


Figure 3.2: Two-dimensional Gaussian depicted a function of x_1 and x_2 with the final adaptive grid produced by VEGAS overlayed. Every 50th grid line is depicted.

There are various methods for generating the proposals mentioned in 3.1.1 and 3.2.1. VEGAS[20] is the most commonly used adaption algorithm in particle physics using importance sampling to adapt multi-dimensional targets. That is why it will be used as a baseline for benchmarks of the machine learning enhanced technique. VEGAS works by approximating the target distribution using step functions in each dimension through an adaptive multidimensional grid. First, a grid of equally spaced bins in each dimension is created. The grid is iteratively optimized by performing the following actions in each iteration:

- Generate random samples in each bin and evaluate the target for each sample.
- Calculate the variance for each bin.
- Shrink bins with high variance, enlarge bins with low variance while keeping the total number of bins constant.

This aims to reduce the total variance and adapt the grid to the target distribution. These actions are repeated for each adaption step until a stopping criterion is reached (e.g., max number of iterations, variance threshold).

If VEGAS achieves a correct adaption of the target, many bins will be concentrated in areas where the target function has high values, and few bins in regions with low values. At the end, the grid can be used to generate samples, where it acts as a step-wise approximation.

In Fig. 3.2, the VEGAS adaption is illustrated for a 2-dimensional Gaussian. Instead of all grid lines used by VEGAS, a reduced number of lines is depicted to improve visibility. Evidently, VEGAS achieved a precise approximation of the target by concentrating the bins around $(0.0, 0.0)$ where the Gaussian has its maximum. This shows how VEGAS can work well with suitable targets, i.e., distributions that factorize into one-dimensional functions, as is the case for the 2d single Gaussian.

The addition of just one more feature to the target can already lead to an insufficient adaption in VEGAS. Fig. 3.3 shows the adaption of two 2-dimensional Gaussians located at $(0.25, 0.25)$ $(0.75, 0.75)$ with equal variance. VEGAS correctly adapted the two features of the Gaussians, but it also adapted two additional features in the opposite corners at $(0.75, 0.25)$ and $(0.25, 0.75)$, where the target distribution shows no features. These incorrectly adapted features will be referred to as 'ghost features' or 'ghost peaks' in the following. The adaption of these ghost features is caused by the projection onto one dimension done by VEGAS. If the target distribution does not factorize into a product of one-dimensional distributions, VEGAS is not suited to generate a convenient proposal for importance sampling. In the case above, about half of the generated samples from this proposal would be rejected (more on the sampling efficiency in Sec. 5). For higher dimensions and more features, this efficiency loss is even more severe as will be shown in Sec. 5. Differential cross sections often feature multiple features in high dimensions, so an alternative to the VEGAS algorithm is desirable.

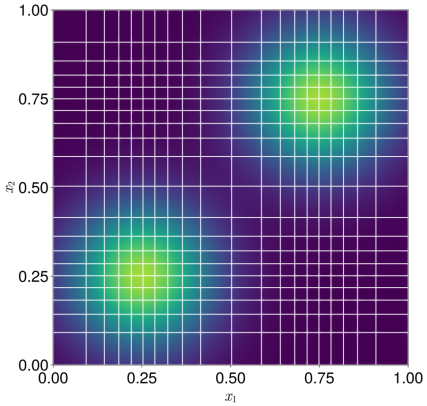


Figure 3.3: The two-dimensional double Gaussian distribution as a function of x_1 and x_2 with the final adaptive grid of VEGAS overlayed. VEGAS was run with default parameters, every 50th grid line is depicted.

4 Enhancement via machine learning

As discussed in Sec. 3, there is a need for accurate proposals to enhance the efficiency of importance sampling for differential cross sections. Since VEGAS suffers from low unweighting efficiencies through the sampling of ghost features, a different approach is required. This section will discuss an alternative method for generating such proposals using machine learning.

4.1 Basics of Artificial Neural Networks

Within this section, we give a brief introduction to the concepts and notations regarding artificial neural networks. A more detailed overview can be found in [21]. Machine learning is a wide field of computer science dealing with algorithms that are able to learn from data. Although even simple algorithms like linear regression can be considered machine learning, the term is usually used to describe more complex algorithms like artificial neural networks. Inspired by the human brain, artificial neural networks use a network of digital neurons, which are interconnected in different ways to process data. The basic principle is illustrated in

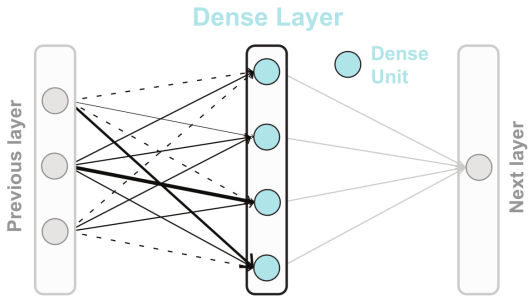


Figure 4.1: A neural network can consist of a chain of Dense Layers where each node is connected to each node of the previous layer. This picture is taken from [22].

Fig.4.1, where a simple feed-forward network[1] of three layers is depicted with one input layer, one hidden layer[2] and one output layer. The term deep learning or

[1] Feed forward neural networks do not contain loops and data flows in one direction, in contrast to recurrent neural networks

[2] All in-between layers which are not the input or output layer are called hidden layers.

deep neural networks is often used to describe such networks depending on the number of hidden layers. While there is no exact definition of how many layers are required, typically, networks with at least two hidden layers are called deep [23]. Each input node takes one scalar value as input and passes it to the next layer. Layers inside the network are characterized by their connection to adjacent layers. In Fig. 4.1, every node in the hidden layer is connected to every node of the input layer. Such layers are called dense layers and are one of the most commonly used types of layers. They will also be the only type of layer of interest in this work. The value of each node in the hidden layer is calculated by multiplying all the values of the previous layer by certain values, so-called weights W[3], summing them up and adding another scalar (bias). Afterward, a non-linear activation function is applied to the result to make an adaption of non-linear functions possible. More dense layers or other types of layers can be added to allow the approximation of complicated functions. Without the use of such an activation function, applying multiple dense layers would be equivalent to applying just one dense layer.

The operations for all nodes in the hidden layer can be written as one matrix multiplication, one addition of the bias vector and one piecewise application of the activation function:

$$y = \mathrm{act}(W \cdot x + b)\,, \tag{4.1}$$

where y is the output vector of the hidden layer, W is the weight matrix, x is the input vector, b is the bias vector, and act is the activation function that is applied element-wise to the input.

One advantage of this setup is that all involved operations are embarrassingly parallel[4], which enables the use of graphics processing units (GPUs) to speed up the calculations significantly. The network is not just evaluated on a single input vector (sample) but on a batch of input vectors to enable a full utilization of GPUs. x is thus a matrix where each column represents one input sample, and the network returns a matrix y where each column is the corresponding output. Using larger batches requires more memory, so the optimal batch size has to be chosen depending on the target function, the network architecture and the used hardware.

There are various other kinds of layers apart from dense layers, which can be used to build networks. We just want to mention one other type of layer which is used in the present thesis. Batch normalization layers[25] are used to normalize the output of a layer to have zero mean and unit variance. This can often speed up the training and increase training stability. However, the use of batch normalization also introduces a correlation between the samples in a batch, which is often undesired.

[3]The phrase *weight* in the context of machine learning has, therefore, a completely different meaning compared to the weights in the context of Monte Carlo event generation. To avoid confusion, we will use the term *network parameters* to refer to the weights and biases in machine learning.

[4]A workload that is separable into parallel tasks with minimal effort is called embarrassingly parallel [24].

4.2 Training of Neural Networks

After initialization, a neural network will produce random data instead of the desired output, which is referred to as the ground truth. The purpose of training is to tweak the parameters of the network in such a way that it produces data that is as close to the desired output as possible. To use the analogy to biology again: this process can be compared to learning in animals, where the connections between neurons are strengthened or weakened depending on the required action. The process usually requires the observation of many examples, which is also the case for artificial neural networks.

A metric is required to compare the output of the network to the desired output. This metric is called the loss function. Depending on the problem, different loss functions can be used. Common loss functions are the mean squared error, the mean absolute error, or the cross entropy loss. The specific loss function used in this work will be discussed in Sec. 4.3.

The loss needs to be minimized by changing the network parameters to improve the output of the network. This is done by calculating the gradient of the loss function with respect to the network parameters. There are different ways to use this gradient to update the parameters. These methods are called optimizers [26]. The simplest optimizer, gradient descent, multiplies the gradient with a certain step size, called learning rate, and moves down the gradient in steps to find the minimum of the loss function. However, the high dimensional loss function usually has not just one local minimum but many local minima and saddle points where gradient descent can get stuck. To overcome this problem, more sophisticated optimizers like Adam [27] have been developed. Adam works similarly to gradient descent, but it uses separate learning rates for each parameter and uses a momentum, obtained as the moving average of the gradient of past training steps, to adapt the learning rate. It has been shown to work well for many applications, including the one in this work. One training step consists of the following operations:

1. Forward pass: The network is evaluated on an input.
2. Loss evaluation: The loss function is evaluated using the output of the network and the desired output.
3. Backward pass: The gradient of the loss with respect to the network parameters is calculated.
4. Network update: The optimizer uses the gradient to update the network parameters.

This process is repeated for many training steps until a stop criterion is reached, e.g., if the maximum number of steps is reached or the optimizer terminates by finding a minimum. The training is performed on batches of input data instead of single inputs because processing a batch in parallel is as fast as processing a single

sample. Additionally, training on bigger samples decreases the total training time. For applications where a neural network is trained on a fixed dataset (which could be observed or synthetically generated data for training), this training dataset is chunked into smaller batches. Then, the network is trained for an arbitrary amount of epochs where the full dataset is used once per epoch. This means that in each training step/iteration, the network processes a batch of data, and the network parameters are updated; this is repeated multiple times in one epoch. For example, with a training set size of 128 and a batch size of 32, 4 iterations will be performed per epoch. With 100 epochs, this will lead to a total of 400 iterations. As we will see later, this is not the case for the network architecture used in the present thesis. In our case, the network is not trained on a fixed dataset. In fact, the model is not trained on any data points that are related to the target function. Instead, the network is trained by passing in uniformly distributed random numbers and comparing the output with the target function. This means that no chunking of training data is required. It would be possible to have a pre-generated set of random numbers, chunk it and use this as a training data set, which is reused every epoch. However, this would not be desirable because the limitation of training data sets limits the ability of networks to generalize to unseen data. In the ideal case, training datasets would be infinitely large so that neural networks do not get a chance to memorize the repeating training data instead of generalizing the problem (called overfitting). For real-world data, this is not possible. However, since the input for our model is generated randomly and at a low computational cost, it is an ideal choice to generate new random numbers for each training step. Because of that, there is just one training step per epoch, and the terms iterations, steps, and epochs can be used interchangeably in this work.

4.3 Neural Importance Sampling

An architecture that can learn distributions is required to aid in the sampling of differential cross sections. In particular, it needs to be able to learn distributions without specific knowledge about their features. With such knowledge, transformations can be applied to suppress the adaption of ghost features in VEGAS, which is done, for instance, by using multi-channel Monte Carlo[28]. However, differential cross sections in strong-field QED are of interest in this work, and these dynamically depend on the properties of the background field. Also, transformations that include the structure of this field can be very complicated and, therefore, also computationally expensive. Thus, the approach for generating proposals for these target distributions needs to be more general.

One kind of architecture which is commonly used to learn distributions, is flow-based generative models, also called normalizing flows [29]. It transforms a simple distribution, like a Gaussian, into an approximation of the desired distribution by applying a series of invertible transformations. When using the generated proposal

for Monte Carlo methods, special properties are required, which are not fulfilled by all normalizing flows:

1. Good approximator: The proposal needs to be a close approximation of the target distribution. (as explained in Sec. 3)

2. Invertible: For each generated sample x_i, the propability density $p(x_i)$ of the proposal needs to be known. ((3.9), (3.14))

3. Fast: To gain a speedup, the generation of samples needs to be faster than the evaluation of the target function.

These requirements are fulfilled by the **Neural Importance Sampling** model [30], also referred to as **NIS**, and it has been shown in [31] that it is especially well suited for the sampling of differential cross sections. In the following three subsections, we introduce the concept of NIS, where we mostly reproduce the explanations from [30, 31].

NIS extends on work done by Dinh et al.[32, 33] on 'NICE' (non-linear independent components estimation), which introduced the concept of coupling layers to learn a mapping from a simple distribution to complicated high-dimensional targets. In [30], this approach is extended by introducing new types of coupling layers, which use piece-wise polynomial functions and significantly increase the performance of the model. NIS represents a kind of normalizing flow, although there are some differences to other commonly used normalizing flows, which perform a forward flow and a backward flow. In NIS, only the forward flow is explicitly performed in training/sampling,

Let $h = h_L \circ ... \circ h_2 \circ h_1 : X \to Y, x \mapsto y$ be the invertible function that transforms points from a simple distribution, in this case uniform random numbers[5], to the target distribution. Each h_i performs a simple part of the total transformation and needs to be invertible. For input points $x \sim p_X(x)$, let the output of the transformation be $y = h(x)$ whose distribution is given by $p_Y(y)$. This PDF[6] can be obtained by the change of variables formula

$$p_Y(y) = p_X(x) \left| \det \left(\frac{\partial h(x)}{\partial x^T} \right) \right|^{-1} . \tag{4.2}$$

with the Jacobian $\frac{\partial h(x)}{\partial x^T}$ of the transformation h for x.

For efficient evaluation, the determinant $\frac{\partial h(x)}{\partial x^T}$ needs to be computed rapidly. In general, this is not the case, especially for high-dimensional Jacobians. For instance, for an nxn matrix, calculating the determinant using LU-decomposition requires n^3 multiplications. The current record needs $n^{2.373}$ multiplications [34].

[5] For the NICE model, a Gaussian distribution was used. But NIS uses uniform numbers instead.

[6] A probability density function (PDF) is defined as a function $f : V \subseteq \mathbb{R}^n \mapsto [0, 1]$ with $\int_V d^n f(x) = 1$.

Since applications on high-dimensional differential cross sections are of interest, y (and x, which has the same dimensionality) will be high dimensional, and so will the Jacobian. Thus, the computation is not feasible for all possible h. Thus, in [32, 33], a special kind of mapping is used for the h_i: the coupling layers.

4.3.1 Coupling Layers

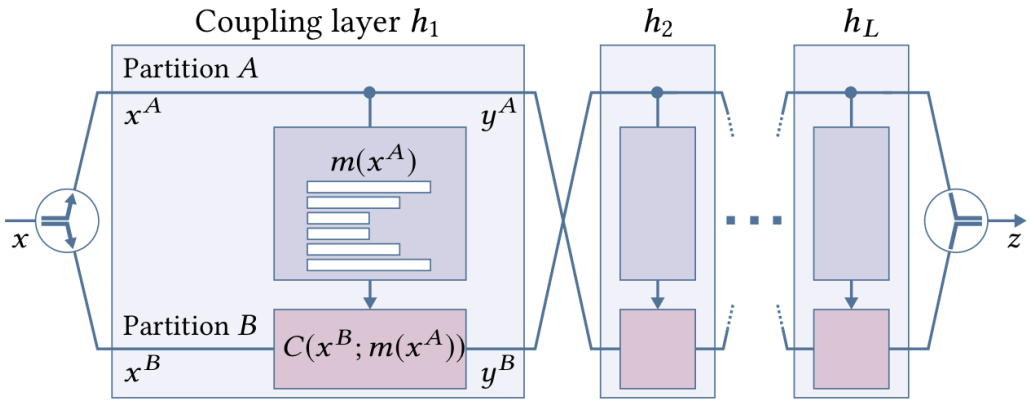


Figure 4.2: Architecture of a NIS model using multiple coupling layers. This picture is taken from [30].

Coupling layers are a special kind of network architecture whose internal structure is schematically depicted as part of a full NIS model in Fig. 4.2. Let $x \in \mathbb{R}^d$ be the d-dimensional input of the coupling layer. A disjoint partition $\{A, B\}$ of the input dimension is used to split the input such that $x = (x^A, x^B)$. The output $h(x) = y = (y^A, y^B)$ of the coupling layer is given by

$$y^A = x^A \tag{4.3}$$

$$y^B = C(x^b; m(x^A))\,, \tag{4.4}$$

where C is a separable invertible map called coupling transform, and m is a neural network (cf. Sec. 4.1). Invertibility means

$$C(x^b; m(x^A)) = \left(C_1(x_1^B; m(x^A)), ..., C_{|B|}(x_{|B|}^B; m(x^A))\right)^T . \tag{4.5}$$

This also implies that the Jacobian has a triangular form:

$$\frac{y(x)}{x^T} = \begin{pmatrix} \mathbb{1} & 0 \\ \dfrac{\partial C(x_B; m(x^A))}{\partial (x^A)^T} & \dfrac{\partial C(x_B; m(x^A))}{\partial (x^B)^T} \end{pmatrix} \tag{4.6}$$

and the determinant can be obtained from the product of the diagonal elements

$$\det\left(\frac{\partial y(x)}{\partial x^T}\right) = \prod_{i=1}^{|B|} \frac{\partial C_i(x^B; m(x^A))}{\partial (x_B)^T}\,. \tag{4.7}$$

With the assumed structure (4.5), the determinant scales linearly with the dimension and allows efficient computation for high-dimensional problems.

A coupling layer effectively splits the input into two parts, transforms one part, and leaves the other part unchanged. In the following coupling layer, the partition is changed so that a different part of the input is transformed. This allows each coupling layer to learn different lower dimensional features of the high dimensional target distribution. Stacking multiple coupling layers allows the model to learn the full distribution.

Dimension	0	1	2	3	4	5	6	7
Transformations 1, 2	0	1	0	1	0	1	0	1
Transformations 3, 4	0	0	1	1	0	0	1	1
Transformations 5, 6	0	0	0	0	1	1	1	1

Table 4.1: Transformations required to capture all correlations for an eight-dimensional input. This table is taken from [35].

The number of required coupling layers depends on the dimensionality and complexity of the target distribution. The minimum number of coupling layers is two, as x_A cannot be empty, and a single coupling layer, therefore, always leaves some part of the input unchanged. The number of coupling layers required to learn all possible correlations between every dimension of the input of the target distribution is d for $d \leq 5$ and $2\lceil \log_2 d \rceil$ for $d > 5$, as was shown in [35]. For the case of five or fewer dimensions, a model could be constructed where each coupling layer transforms one dimension. An example for eight dimensions is shown in table 4.1, where the required transformations are constructed the following way:

1. Reindex the dimension numbering from $[1, d]$ to $[0, d-1]$

2. Convert each dimension number to binary using the minimum required bits to represent $d-1$

3. For each n-th bit, construct a coupling layer that transforms each dimension where the n-th bit is one and a second coupling layer that transforms each dimension where the n-th bit is zero.

4. Repeat the previous step for all bits.

In this example, the first coupling layer would transform dimensions 1, 3, 5, and 7, and the second coupling layer would transform dimensions 0, 2, 4, and 6, where the $[0, d-1]$ notation is used for the dimensions. As $\lceil \log_2 d \rceil$ bits are required to represent the number $d-1$, and two transformations are required for each bit, the number of coupling layers required to adapt all possible correlations in d dimensions is $2\lceil \log_2 d \rceil$.

4.3.2 Piecewise-Quadratic Coupling Transform

Different choices for the coupling transform C are possible. The original NICE model used an additive coupling transform [32], though in [30], it was shown that a piecewise-polynomial coupling, and a piecewise-quadratic coupling in particular, perform particularly well. In this work, the piecewise-quadratic coupling transform will be used for C.

x and y are chosen to be vectors in the d-dimensional hypercube $[0,1]^d$. This will allow C_i to be interpreted as a cumulative distribution function (CDF)[7] for each dimension. For the construction of the CDFs, the network m returns parameters that are used to build unnormalized PDFs q_i through interpolation, which are then integrated to obtain the C_i, as illustrated in Fig. 4.3. The PDFs q_i are

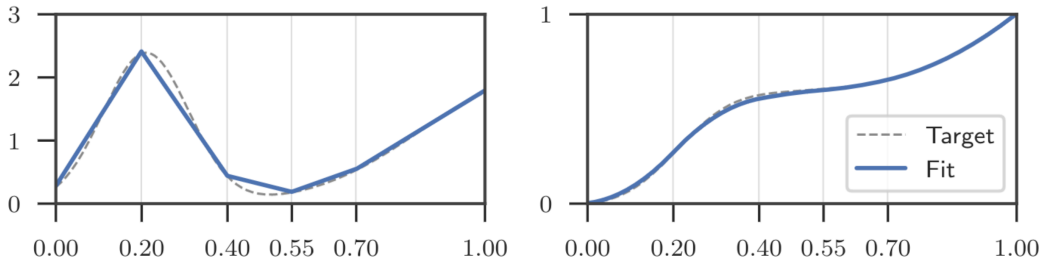


Figure 4.3: PDF predicted by the network (left) and the resulting CDF (right) with five bins for an example target distribution (dashed). Picture taken from [30].

constructed using piecewise linear functions in K bins with $K+1$ edges. Two vectors are required to construct each q_i: one of length $K+1$ containing the height at each vertex and one of length K containing the bin widths. For $|B|$ dimensions of the transformed partition, the $(K+1)\times|B|$ matrix V contains the heights, and the $K\times|B|$ matrix W contains the bin widths.[8] The output matrices $\hat{V}$ and $\hat{W}$ of the neural network m will be used for this construction, but since a neural network generally does not ensure normalization, the output needs to be normalized first. The widths of all bins should sum up to one, so each column of $\hat{W}$ will be normalized using the softmax function

$$W = \mathrm{softmax}(\hat{W})\,. \tag{4.8}$$

For $\hat{V}$, the columns need to be normalized in such a way, that the constructed q_i

[7] A cumulative distribution function (CDF) is defined as a right-continuous monotone increasing function $f : \mathbb{R} \mapsto [0,1]$ with $\lim_{x\to-\infty} f(x) = 0$ and $\lim_{x\to+\infty} f(x) = 1$.

[8] Note that rows and columns are swapped compared to the notation used in [30] and [31]. Our code was written to run on GPUs using the Flux.jl machine learning package, and with this setup, the transpose form was easier to implement.

will be PDFs. This is achieved by:

$$V_{i,j} = \frac{\exp\left(\hat{V}_{i,j}\right)}{\sum_{k=1}^{K} \frac{1}{2}\left(\exp\left(\hat{V}_{k,j}\right) + \exp\left(\hat{V}_{k+1,j}\right)\right) W_{k,j}} \,. \tag{4.9}$$

Now, the PDF can be constructed as

$$q_i(x_i^B) = V_{b,j} + \alpha(V_{b+1,j} - V_{b,j}) \,, \tag{4.10}$$

where b is the bin containing x_i^B and α is the relative position of x_i^B inside that bin.[9]

The coupling transform C_i is then obtained by integrating q_i:

$$\begin{aligned} C_i(x_i^B) &= \int_0^{x_i^B} q_i(u)du \\ &= \frac{\alpha^2}{2}(V_{b+1,j} - V_{b,j})W_{b,j} + \alpha V_{b,j} W_{b,j} + \sum_{k=1}^{b-1} \frac{V_{k,j} + V_{k+1,j}}{2} W_{k,j} \end{aligned} \tag{4.11}$$

and the Jacobian determinant (4.7) is

$$\det\left(\frac{\partial y(x)}{\partial x^T}\right) = \prod_{i=1}^{|B|} q_i(x_i^B) \,. \tag{4.12}$$

4.3.3 Loss

Subsequent to the definition of the NIS model, the training process will be posed in this section. The goal of the training is to optimize the parameters of the neural subnets m in each coupling layer in such a way that the output of the model follows the target distribution as closely as possible. Thus, a metric is required to measure the distance between the learned distribution of the model and the target distribution. Let $x \in [0,1]^d$ be the input samples and $y \in [0,1]^d$ be the corresponding output samples of the model. Following (3.9) and (3.10), the target distribution needs to be evaluated at each output sample y_i of the model. However, the output of the network is currently a vector in the d-dimensional unit hypercube, while the target function generally expects values in different intervals. Therefore, a channel mapping $y \mapsto z$ is introduced, which transforms the model output to the domain space of the target distribution.

As explained in Sec. 3.2.1, the weight of a generated sample z is given by the Jacobian of the transformation multiplied by the target distribution f evaluated at z:

$$w = \left|\det\left(\frac{\partial y(x)}{\partial x^T}\right)\right| \left|\det\left(\frac{\partial z(y)}{\partial y^T}\right)\right| f(z) \,, \tag{4.13}$$

[9] We mention, to use these functions efficiently in our code, we must implement them in a way compatible with the GPU programming paradigms (cf. [36, 37]), and in an auto-differentiable way (see, e.g., [38] for details.)

Note that this is the explicit realization of equation (3.7).

Following [31] and [30], the Pearson χ^2-divergence will be the loss function of choice, as minimizing it will lead to a minimal variance of the Monte Carlo estimate. The Pearson χ^2-divergence is a measure for the distance between two distributions for a set of n samples

$$D_{\chi^2} = \frac{1}{n} \sum_{i=1}^{n} \frac{\left(f(z_i) - g(z_i)\right)^2}{g(z_i)} \,, \tag{4.14}$$

where g is the output sample distribution and can be obtained from the Jacobian of the model:

$$g(z_i) = \frac{1}{\left|\det\left(\frac{\partial y(x)}{\partial x^T}\right)\right| \left|\det\left(\frac{\partial z(y)}{\partial y^T}\right)\right|} \,. \tag{4.15}$$

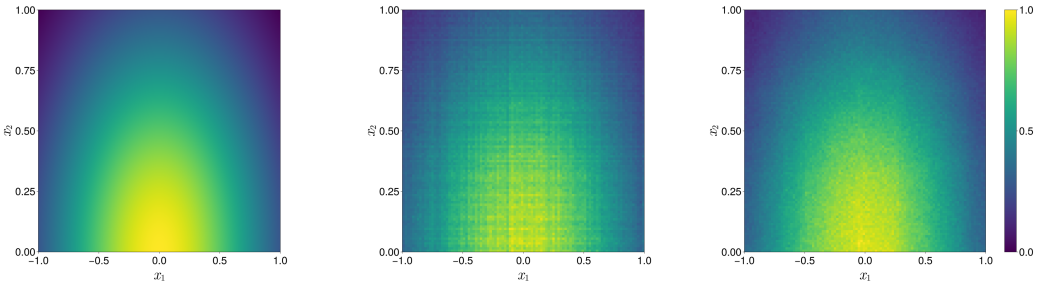
Note that the presented loss function technically requires the target distribution f to be normalized. However, this requirement can be bypassed by using optimizers like Adam that cancel out the integral during training, as is explained in [30].

5 Numerical experiments

5.1 Preliminary investigation

During the implementation of NIS using the Julia programming language and the subsequent vectorization of the code, simple examples have been used to test the correctness of the implementation at each step and evaluate different hyperparameters depending on the problem's dimensionality.

5.1.1 Single Gaussian Distribution



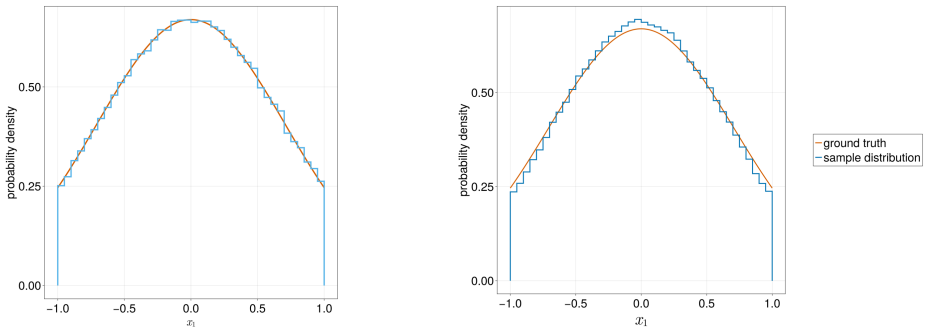
(a) Ground truth (b) VEGAS (c) NIS

Figure 5.1: Comparison of samples generated from the VEGAS and NIS proposals for the 2d single Gaussian distribution. For VEGAS, the default parameters were used: `maxiter=10`, `nbins=1000`, `ncalls=10000`, `rtol=1e-4`, `atol=1e-4`, `alpha=1.5` and for NIS, the following parameters were found to work well for this problem: `coupling_layers=2`, `subnet_depth=3`, `subnet_width=8`, `activation=relu`, `bins=10`, `learning_rate = 0.01`, `batch_size = 16384`, `epochs=80`, `iterations/epoch=1`.

As a first example, a Gaussian distribution is used to confirm that NIS produces samples that are distributed according to the target distribution and ensure that NIS performs similarly to VEGAS. The investigated distribution is given by

$$f(x_1, x_2) = e^{-(x_1^2 + x_2^2)} \,. \tag{5.1}$$

It is centered around $(0, 0)$ and is evaluated in the range $x \in [-1, 1], y \in [0, 1]$. This domain is chosen because the same domain will be used later for the pulsed-perturbative Compton process. Thus, the mapping to the Compton domain can



(a) VEGAS (b) NIS

Figure 5.2: Histogram of $N = 10^7$ normalized proposal samples generated with VEGAS and NIS for the single Gaussian distribution as a projection onto the x_1-axis. Additionally, the normalized target distribution integrated over x_2 is depicted as a function of x_1.

also be tested in this example. The single Gaussian distribution and the respective adaptive grid from VEGAS were already shown in Fig. 3.2.

A comparison between the proposal samples generated by the two importance sampling methods is shown in Fig. 5.1 as a histogram of x_1 and x_2. 10^7 samples have been generated for each histogram. VEGAS was run with the default parameters; more iterations or function calls per iteration did not improve the results because VEGAS terminated before reaching the maximum values. NIS was trained using parameters that have been found to work well for this problem as given in Fig. 5.1. The produced sampling resembles the target distribution in both cases, no significant artifacts are present. However, a minor difference between the two methods can be observed: a grid structure is clearly present in the VEGAS proposal, while a binning is only slightly observable in the NIS proposal. This is due to the fact that VEGAS uses step functions to approximate the target, while a piecewise quadratic interpolation was chosen for NIS leading to a smoother approximation. We also mention that the number of bins differs significantly between the two methods (1000 for VEGAS and 10 for NIS[1]).

High-dimensional sampling can be directly compared with the analytical solution by using projections. Fig. 5.2 shows the generated samples accumulated onto one axis and normalized to obtain a 1d histogram, which is compared with the 1d-Gaussian function e^{-x^2}. The 1d-Gaussian has been normalized to the integral over the domain, thus the normalized histogram and the function are PDFs and can be compared directly. Note that, in general, comparing these projections of samples would require the integration of the high-dimensional target function over the other dimensions. But as shown in Sec. A.1.3, the 1d-Gaussian is identical to any integrated higher dimensional Gaussian after normalization. This identity holds true for all Gaussian examples investigated in the present thesis and is used

[1]The number of bins is always given as the number of bins per dimension in the present thesis.

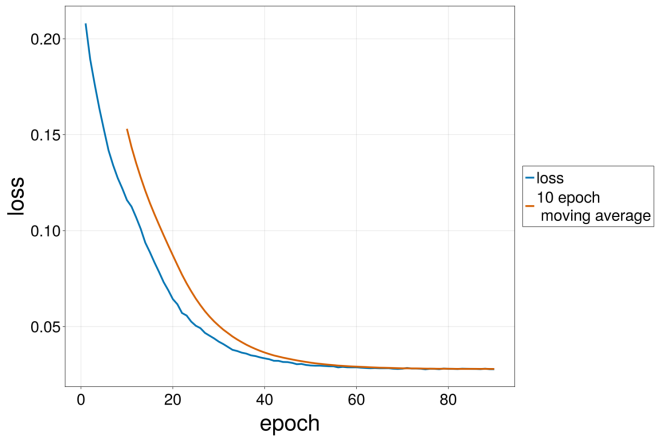


Figure 5.3: Training loss of the NIS model while training on the 2d single Gaussian steadily decreases without major fluctuations.

to obtain the projections used for comparisons of the two- and five-dimensional distributions. First, we observe in Fig. 5.2, that both methods are able to reproduce the 1d-Gaussian shape. The VEGAS proposal matches the ground truth over the whole domain with only minor deviations. This is the expected outcome as a single Gaussian factorizes into a product of one-dimensional functions and is therefore ideal for the VEGAS algorithm which exploits one-dimensional projections. The NIS proposal also matches the ground truth on average, but a bias can be observed. The proposal overestimates the target at the edges of the domain and underestimates it in the center. This behavior hints at a difficulty in adapting near domain boundaries for NIS which will be further investigated in the following sections. The training loss of the NIS model decreases steadily with increasing iteration without significant fluctuations, as shown in Fig. 5.3. This is an ideal behavior, which is generally not the case when training neural networks on complicated problems. It is an indication for the strong capability of the model to adapt a low-dimensional target without a complicated structure. To quantify this ability, the unweighting efficiency can be used as a measure of the proposal quality. As it equates to the ratio of samples that are accepted after unweighting, it should be as high as possible to ensure an efficient generation of unweighted samples. Therefore the weights of the generated samples are calculated as given by equation (4.13). For a maximum unweighting efficiency, all weights should be equal. If the target is a PDF, this value would be 1.0, as given by equation (3.13) and (3.15), where the ground truth $w = f$ and the proposal g would be identical PDFs. However, since one purpose of NIS is to aid in Monte Carlo integration to obtain the integral, the target is not necessarily normalized. Therefore, ideal weights will not concentrate around 1.0, but around the integral of the target. Therefore, the weight distributions will be normalized to the obtained Monte Carlo integral to achieve a target independet comparison of weight distributions. After normalization, a narrow distribution around 1.0 again indicates a good pro-

posal. A comparison of the weight distributions for VEGAS and NIS is shown in Fig. 5.4 where histograms of the sample weights are shown as a function of the weights. Both methods feature a weight distribution centered approximately around 1.0. The weight distribution of VEGAS is more smooth while the NIS proposal has a higher peak close to 1.0. For VEGAS, the normalized weights have a mean of 1.00 and a maximum 1.49, which results in an unweighting efficiency of 0.67. For the NIS proposal, the normalized weights have a mean of 1.00 and a maximum of 1.35, resulting in an unweighting efficiency of 0.74, slightly above VEGAS. Both methods perform similarly on this target distribution as measured by the unweighting efficiency. The Monte Carlo integral of the VEGAS proposal is 1.1151 with a low error of 0.0004, matching the analytical solution. For NIS, the interagral is 1.12 with an error 0.07. Due to the stronger bias in the NIS proposal, the error is higher than for VEGAS.

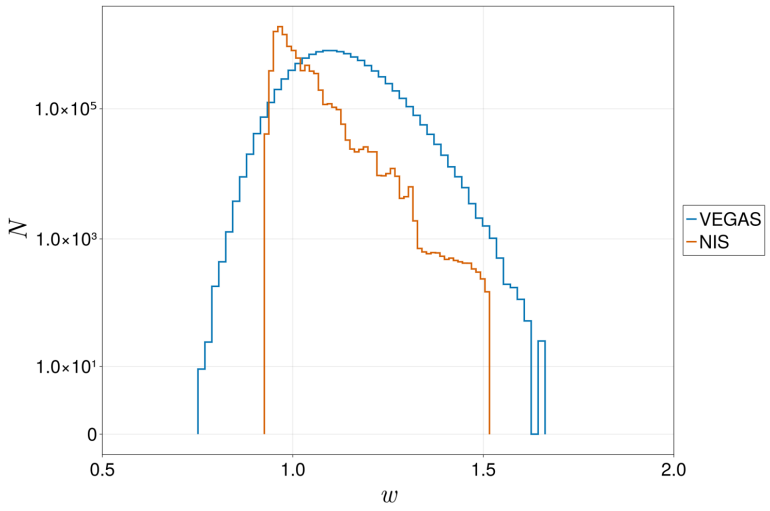


Figure 5.4: Comparison of the sample weight distributions of the VEGAS and NIS proposals for the 2d single Gaussian, $N = 10^7$, w normalized to the integral of the target.

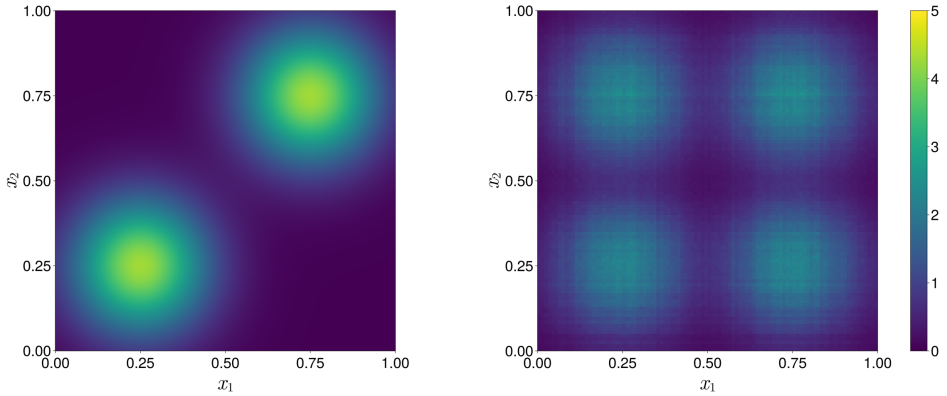
5.1.2 Double Gaussian

2d Double Gaussian

As a second example, a function that does not factorize into one-dimensional functions is used to test the ability of NIS to adapt multiple features without producing ghost features as VEGAS does (cf. Sec. 3.3). The target function is written in a dimension-independent way to allow the testing in different numbers of dimensions:

$$f(x) = e^{-\left(\sum_{i=1}^{d} \frac{(x_i-0.75)^2}{0.2^2}\right)} + e^{-\left(\sum_{i=1}^{d} \frac{(x_i-0.25)^2}{0.2^2}\right)}, \tag{5.2}$$

which will be referred to as the double Gaussian from now on. It represents two Gaussians located in two corners on the diagonal of the $[0, 1]^d$ hypercube. The Gaussians are centered around $(0.25)^d$ and $(0.75)^d$ respectively with a width of $\sqrt{2}/10$.

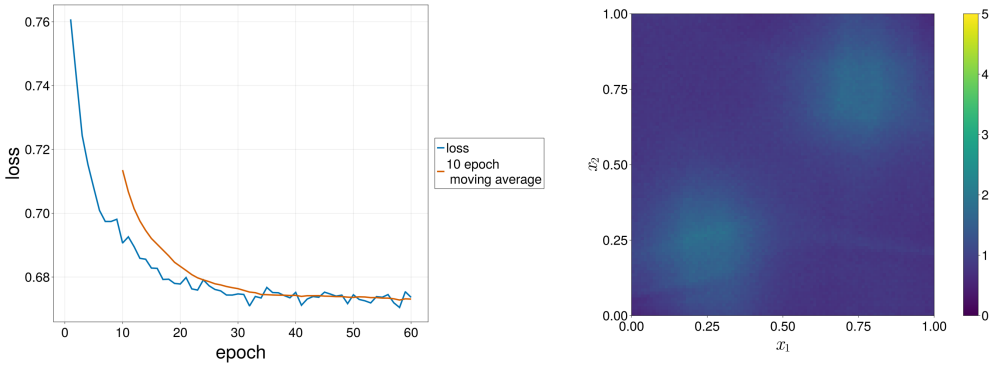


(a) Ground truth (b) VEGAS sampling

Figure 5.5: Comparison between the ground truth and proposal samples generated from the VEGAS proposal for the 2d double Gaussian distribution. VEGAS was run with the default parameters (see Fig.5.1), $N = 10^7$ samples generated.

As in the single Gaussian example, VEGAS is used as a benchmark and has been run with the default parameters given in the caption of Fig. 5.1. The resulting adaptive was already shown in Fig. 3.3, and the samples generated from the VEGAS proposal are compared with the ground truth in Fig. 5.5. It can be seen that VEGAS generates samples symmetrically in all four quadrants of the domain, whereas two of the four generated features should not be present. This is caused by the ambiguity of one-dimensional projections of the target. If the features of the target are aligned along one axis of the domain (e.g., at $(0.25, 0.25)$ and $(0.75, 0.25)$), the generated proposal would match the target distribution. But this example has been specifically chosen to be non-factorizing. Apart from the presence of the ghost features, the proposal Gaussians are shaped correctly, and no further artifacts are present. Since the samples are split about equally between the four features, the two correct features are half as intense as given by the target distribution.

The training loss and final sampling of NIS for the 2d double Gaussian are shown in Fig. 5.6. For the training parameters are used that have been found to work well for this problem. The training loss quickly decreases at the start of the training and reaches a plateau after about 40 epochs, where the 10-epoch moving average shows no further decrease. Note that the absolute value of the loss is not meaningful in most cases; the trend of the loss is more important. Therefore no specific values are given in the present thesis. Since no further improvement can



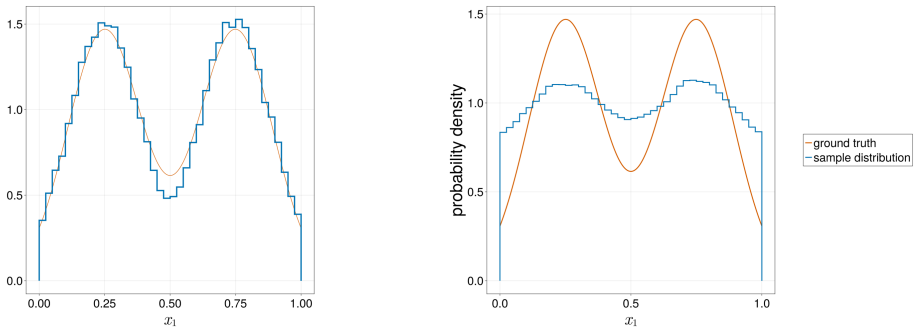
(a) Training loss (b) Generated samples

Figure 5.6: The NIS model has been trained on the 2d double Gaussian distribution using the following parameters: `coupling_layers=2`, `subnet_depth=3`, `subnet_width=8`, `activation=relu`, `bins=10`, `learning_rate=0.01`, `batch_size=16384`, `epochs=40`, `iterations/epoch=1`.

be observed, the performance of this model cannot be improved by increasing the training duration. Instead, changing the network architecture or hyperparameters would be required. Some limited fine-tuning has been performed to obtain the used settings for this example, but further investigation will instead be performed on the physical applications in the following section since the Gaussian examples only serve as a way to test the functionality of the implementation. In Fig. 5.6b, the sampling from the trained model is shown for 10^7 samples. The model was able to learn the location of both Gaussians correctly without creating ghost features in the opposite corners of the domain. However, the generated features are not perfectly symmetrical and a considerable amount of noise is present over the whole domain, which is only about an order of magnitude weaker than the actual features. Comparing regions in the domain with an almost vanishing target distribution, for example near $(0.5, 0.5)$, VEGAS performs better than NIS since it produces less noise.

To investigate this behavior further, the samples are summed over x_2 and compared as a normalized histogram with the normalized ground truth in Fig. 5.7. The comparison for VEGAS shows a good adaption in this projection with only a slight bias. This is due to the fact that any non-factorizing property of the target is lost in this one-dimensional projection which masks the generated ghost features. Therefore it only serves as a means to validate the correct adaption of the shape of the Gaussians. NIS shows a strong bias in this projection. Due to the noise being present over the whole domain, there is an overestimation in all regions with a vanishing target distribution. This might be further amplified by the difficulty of NIS to adapt near domain boundaries as observed in the single Gaussian example. Due to these overestimations, the centers of the Gaussians are underestimated.

For a quantitative comparison of the proposal performances, the sample weight



(a) VEGAS (b) NIS

Figure 5.7: Histogram of $N = 10^7$ normalized proposal samples generated with VEGAS and NIS for the 2d double Gaussian distribution as a projection onto the x_1-axis. Additionally, the normalized target distribution integrated over x_2 is depicted as a function x_1.

distributions are shown in Fig. 5.8. The distribution of VEGAS features two peaks, one close to zero and one around 1.8. The first peak close to zero corresponds with the samples generated in the two ghost features. As the target distribution has a value of almost zero in the opposite corners, all samples generated there will have weights close to zero. The second peak corresponds with the samples generated in the true peak regions. Their weights are close to two, because these regions are undersampled and should have twice as many samples as are generated. For higher weights the distribution decreases exponentially. In contrast, the weight distribution for NIS is close to constant for weights between 0 and 2.6 with a sharp decrease after 2.6. There is also a peak close to zero, which corresponds with the noise samples generated in the whole domain. This peak is smaller than for VEGAS, which means NIS generates fewer noise samples than VEGAS produces samples in the ghost feature regions. The absence of a second peak indicates that VEGAS did not just undersample the two gaussians by the sample amount in the noise as VEGAS did but differs from the target in more than just a constant factor in those regions. This matches the observation made on Fig. 5.12b.

For VEGAS, the normalized weights have a mean of 1.00 and a maximum of 3.43, resulting in an unweighting efficiency of 0.29, which is approximately half of the unweighting efficiency for the single Gaussian case. A halfing in efficiency is expected, as about half of the samples generated in the ghost peak regions have weights close to 0. This decrease in efficiency will be exacerbated in higher dimensions. The VEGAS algorithm would try to change the adaptive grid to generate fewer samples in these regions with low weights (by increasing the bin widths and decreasing them in other regions with high weights), but since the Gaussians overlap with the opposite empty corners in 1d-projections, VEGAS fails. The normalized weights for NIS have a mean of 1.00 and a maximum of 2.60, resulting in an unweighting efficiency of 0.38, slightly above VEGAS. In this

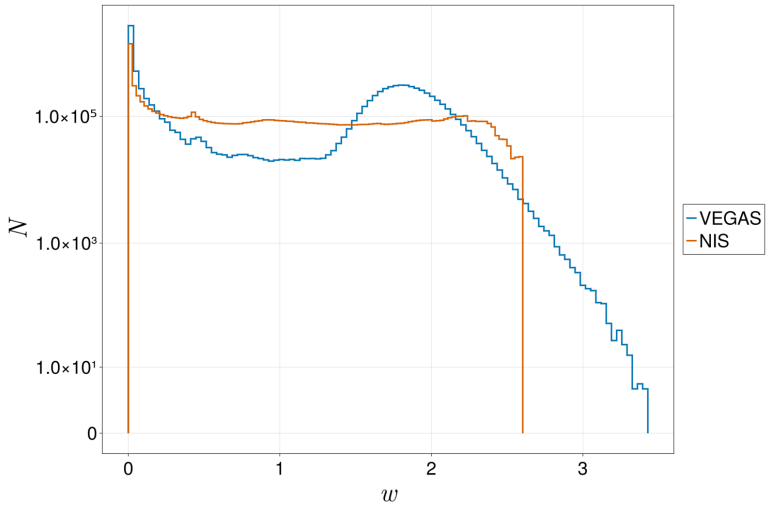


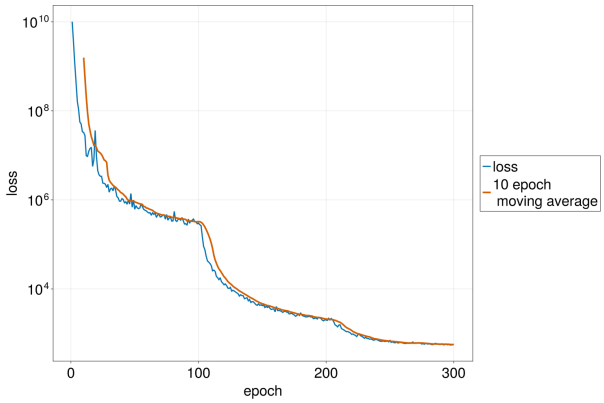
Figure 5.8: Comparison of the sample weight distribution of the VEGAS and NIS proposals for the 2d double Gaussian distribution, $N = 10^7$ samples generated, w normalized to the integral of the target.

example, the two importance sampling methods perform similarly again. But the reason for their efficiencies being well below 1.0 is different. The efficiency of VEGAS decreases due to the adaption of ghost features, while the efficiency of NIS decreases due to the the generation of significant noise. The VEGAS proposal samples result in an integral of 0.2327 with an error of 0.0007. For NIS, the integral is 0.23 with an error of 0.19, which is significantly higher than the error of VEGAS, which is unexpected given the similar unweighting efficiencies. However, taking the significant noise level into account, a higher error is reasonable.

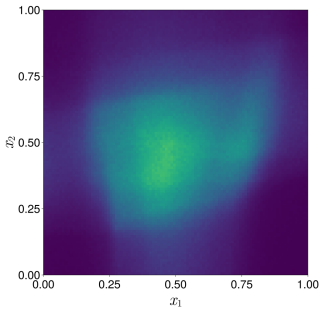
5d Double Gaussian

Since the double Gaussian target distribution is not limited to two dimensions, it can also be used to test the behavior of NIS and VEGAS in higher dimensions. We investigate the $d = 5$ case within this section because the trident process that shall be investigated later has a five-dimensional phase space. More coupling layers are now required to adapt the target as described in Sec. 4.3.1. We choose the parameters given in the caption of Fig. 5.9 to train the model.

Comparing the training loss for the 5d double Gaussian distribution training loss shown in Fig. 5.9a to the previous examples shown in Fig. 5.6a reveals a notable difference. An additional training strategy had to be employed to obtain more convenient results for the 5d double Gaussian. After about 100 epochs, the loss plateaus even though the proposal is far from optimal, as can be seen in the snapshot shown in Fig. 5.9b where the NIS model has not yet adapted the two Gaussians correctly. Further training did not improve the proposal in this case, so the training strategy was adjusted by decreasing the learning rate one every 100



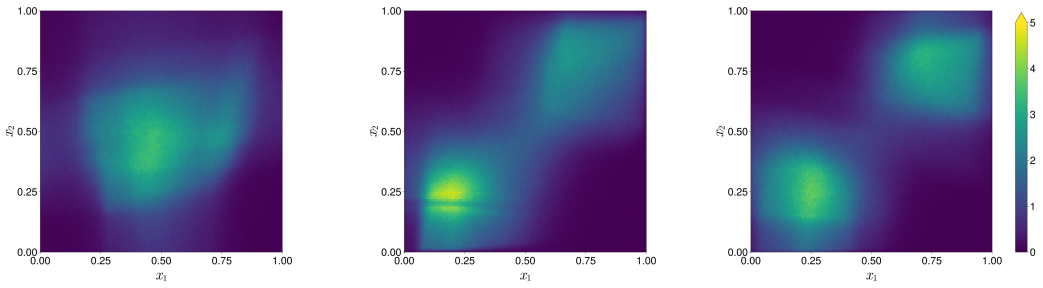
(a) Training loss



(b) Sampling after 100 epochs

Figure 5.9: Training loss and a snapshot of the proposal samples generated from the NIS for the the 5d double Gaussian distribution after 100 epochs of training. The training was performed using the following parameters: `coubling_layers=4`, `subnet_depth=4`, `subnet_width=16`, `activation=relu`, `bins=10`, `learning_rate=0.01`, `batch_size=16384`, `epochs=300`, `iterations/epoch=1`

epochs. With optimizers like gradient descent, this strategy can be used to get closer to the global minimum of the loss, as the optimizer can skip this minimum if the learning rate is too high [39]. However, the Adam optimizer is used in the present thesis as it has been found to work well in the investigated setup, and the Adam optimizer includes its own decaying version of a learning rate, called momentum, that updates with each training step. In theory, no learning rate decay is required for Adam. Updating the learning rate of Adam would overwrite the stored momentum so it should be counterproductive. However, updating the learning rate every few epochs significantly improves the training in our application. Finding the cause for this behavior leaves room for further investigation.



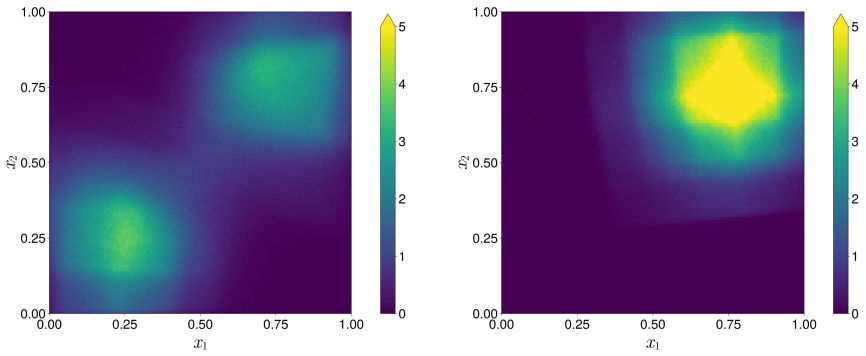
(a) After 100 epochs (b) After 200 epochs (c) After 300 epochs

Figure 5.10: Progression of the samples generated from the NIS proposal for the 5d double Gaussian distribution in steps of 100 epochs. $N = 10^7$ samples generated for each snapshot.

Fig. 5.10 shows the progression of the NIS proposal after 100, 200, and 300 epochs of training. For the first 100 epochs, the network has not yet been able to locate the two Gaussians correctly. After 200 epochs, the model is able to reproduce two separate features that are still connected to each other and show major artifacts. After 300 epochs, the two Gaussian features can be clearly identified, although they are still not perfectly symmetric and disconnected from each other. The learning rate was decreased once every 100 epochs by a factor of 0.7. Very frequent updating did not lead to meaningful improvements. The learning rate updates are strongly reflected in the loss shown in Fig. 5.9a. Every time the learning rate is updated, a rapid change in the loss can be observed. Earlier in the training, the updating causes the loss to decrease from its previous plateau immediately. In higher epochs, it causes the loss to spike up shortly before decreasing, and beyond 300 epochs, there is no more meaningful decrease in the loss can be observed which is a sign that no further improvement through training will be achieved.

Furthermore, a second significant difference to the 2-dimensional Gaussian example has been observed: The neural subnets of the coupling layers now require a batch normalization layer. Without batch normalization, the model only adapts one of the two Gaussians and does not adapt the second Gaussian at all. This can be observed in Fig.5.11, where the proposal samples are shown for two different NIS models that use the same parameters and architecture as given in the caption of Fig. 5.9 with the difference that the model used in Fig. 5.11a uses a batch normalization layer, and the model used in Fig. 5.11b does not. As NIS uses random input data instead of a training data set, the introduction of a correlation between the samples in a batch through batch normalization is not an issue. This is reflected by the enhanced performance of the model with batch normalization. However, the reason for the necessity of this additional layer leaves room for further investigation.

Fig. 5.12 shows a comparison between projections onto the first two dimensions of the samples generated from the VEGAS and NIS proposals and the target distribution as ground truth. Other projections are possible, but the sample distribution is similar in all possible projections as shown in appendix A.2. Comparing the sample distribution in Fig. 5.11a with the sampling for the 2d double Gaussian distribution shown in Fig. 5.6b reveals a significant decrease for the five-dimensional case. This is a surprising finding for a more complicated target. The behavior points at a property of the NIS model to be better suited for targets of dimensionality higher than two. Considering that for non-factorizing targets, VEGAS performs best in two dimensions and that its efficacy decreases for higher dimensions, this property of NIS is convenient. However, the explanation for this behavior leaves room for further investigation. Comparing the shape of each Gaussian, the increased difficulty in adapting higher-dimensional targets becomes apparent. In five dimensions, the sampled Gaussians are less symmetric than in the two-dimensional case. It seems like there are two competing effects for the efficiency of NIS in higher dimensions: a decrease in noise for higher dimensions



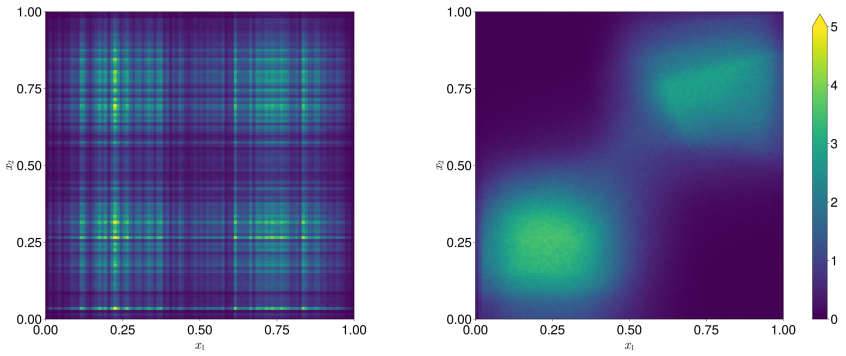
(a) With batchnorm (b) Without batchnorm

Figure 5.11: Comparison between the proposal samples generated by two different NIS models for the 5d double Gaussian distribution as projections onto the x_1-x_2-plane. The same parameters as given in the caption of Fig. 5.9 have been used with the exception of adding a batch normalization layer at the start of each subnet for Fig. (5.11a).

and a decrease in the correct adaption of the exact structure of features.

As has been observed for the two-dimensional case, VEGAS also generates ghost features in the five-dimensional double Gaussian distribution. These ghost features are present in every possible two-dimensional projection as is shown in appendix A.1. Further, the Gaussians were only partially adapted in this example. It can be seen that every few bins, the VEGAS grid alternates between intense and weak sampling, showing that VEGAS was barely able to adapt the shape of each of the two Gaussians. On the other side, the NIS model was able to learn the location and general shape of both Gaussians. This can be verified in all two-dimensional projections as shown in appendix A.2. A model with six coupling layers has also been tested, which is the minimum number of coupling layers required correlations between all five dimensions as explained in Sec. 4.3.1. However, this larger model produced a proposal with a lower unweighting efficiency of 0.0033 compared to 0.011 of the model with five coupling layers. This shows that the model does not need to learn all correlations for simple targets like the double Gaussian and a smaller model actually leads to better training due to the reduced complexity of the model. The samples produced by the six-layer model are shown in appendix A.3.

The weights of the generated proposal samples will again be used again to quantify the proposal efficiency. Therefore, 10^7 samples have been generated, and the corresponding weights are shown in Fig. 5.13. For VEGAS, the normalized weights have a mean of 1.1 and the maximum of 3600, resulting in an unweighting efficiency of 0.00030. The normalized mean weight produced by the NIS proposal is 1.1 and the max weight is 95, resulting in an unweighting efficiency of 0.011. The weight distribution of NIS is centered around 2 while the weights distribution of VEGAS peaks peak at zero, which corresponds to samples in generated in the



(a) VEGAS (b) NIS

Figure 5.12: Comparison of samples generated from the VEGAS and NIS proposals for the 5d double Gaussian distribution. A projection onto the first two dimensions is shown for each. Note that the color bar is open ended as some values in the VEGAS result exceed the range of the color bar.

ghost features regions where. There, the contribution of the vanishing target leads to vanishing weights. In addition, the few outlying weights with values exceeding 1000 result in the low unweighting efficiency of VEGAS. Clearly, NIS performed better in this example, hinting at a better performance for higher dimensional non-factorizing targets in general. VEGAS resulted in an integral of 0.0088 with an error of 0.0002, while NIS resulted in an integral of 0.009 with an error of 0.005. Again, the error for NIS is higher than for VEGAS, even though the unweighting efficiency is also higher.

Models with more coupling layers, especially as described in Sec. 4.3.1, have been tested. However, their performance was below that of the discussed model with four coupling layers. Samples generated from the model using six layers are shown in appendix A.3.

5.2 Application to Physical Processes

After the functionality of NIS has been verified and knowledge on the suitable choice of hyperparameters and model architectures has been acquired, NIS can be applied to physical processes. First, the adaption of the differential cross section of the pulsed-perturbative Compton process will be investigated to understand the model's ability to learn the dynamic influence of the laser background field. This will be the physical application featuring a non-factorizing target distribution in a dimensionality that is favorable for VEGAS. As a second physical process, the trident process will be investigated to ascertain the suitability of NIS for higher-dimensional scattering processes where an alternative to VEGAS is desirable.

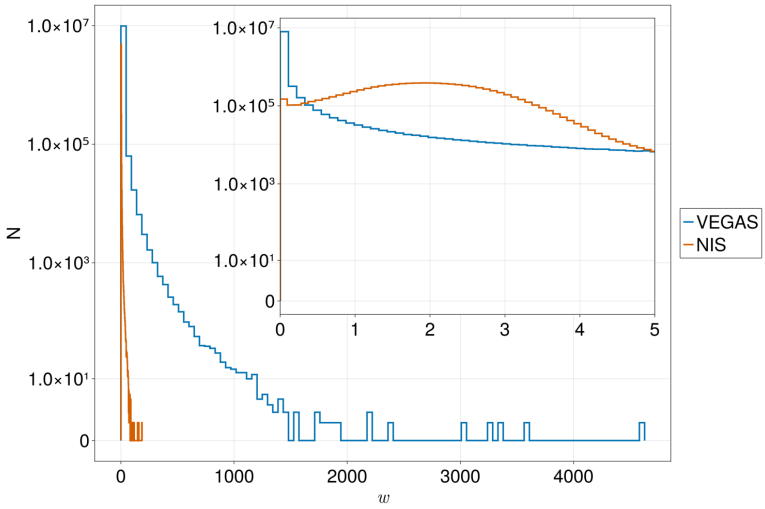
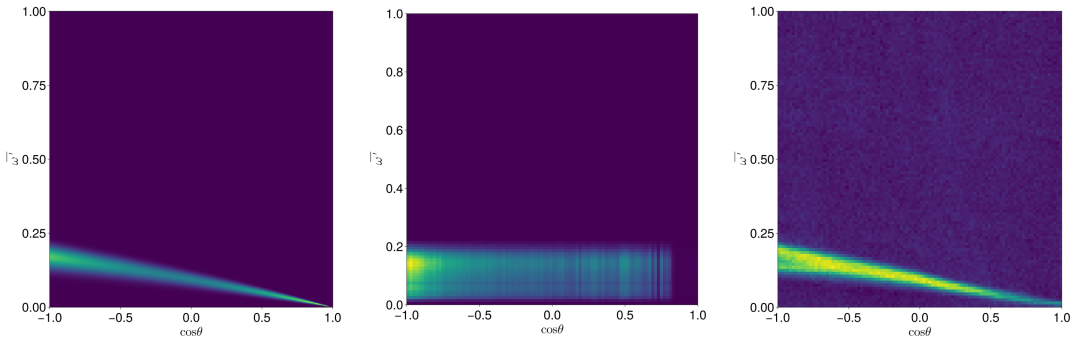


Figure 5.13: Comparison of the sample weight distribution of the VEGAS and NIS proposals for the 5d double Gaussian distribution, $N = 10^7$ samples, w normalized to the integral of the target.

5.2.1 Pulsed-Perturbative Compton Process

The target distribution for the pulsed-perturbative Compton process is given by equation (2.35). The same NIS training parameters are used as were used for the two-dimensional double Gaussian and are given in Fig. 5.6; the default parameters were used for VEGAS. For a photon energy of $\omega = 0.1\ m_e$ in the rest-frame of the electron and a $\cos^2$-pulse with $\Delta\phi = 10.0$ as envelope of the background field, the samples generated from the VEGAS and NIS proposals are shown in Fig. 5.14. As was presumed from the results of the double Gaussian examples, VEGAS produces a non-optimal adaption of the structure of this non-factorizing target. Instead of the true shape of the target distribution, a rectangular proposal is generated. A correct adaption of the peak at $\cos\theta = -1.0$ can be observed, but the peak at $\cos\theta = 1.0$ is not adapted. Moreover, due to the rectangular shape of the proposal, a large amount of samples are generated in regions where the target vanishes. For values larger than $\overline{\omega'} = 0.2$, almost no samples are generated, which is in accordance with the target distribution. VEGAS correctly minimized the proposal in this region because there, the target is almost flat, thus can be factorized.

The proposal samples of NIS are shown in Fig. 5.14c. In addition to the correct adaption of the peaks at both ends of the domain, the NIS model was also able to learn the approximate shape of the target in the whole domain. However, the exact structure of the target distribution within this non-vanishing region is not exactly reproduced by the NIS proposal. The poposal distribution peaks in the range $-0.8 < \cos\theta < 0.5$, while the target distributions peaks at $\cos\theta = -1.0$ and $\cos\theta = 1.0$. In addition to the samples generated in the non-vanishing region of the target, the NIS proposal also generates noise present in the whole domain.



(a) Ground truth (b) VEGAS (c) NIS

Figure 5.14: Comparison of samples generated from the VEGAS and NIS proposals for the pulsed-perturbative Compton differential cross section and the ground truth. For VEGAS the default parameters were used, which are given in the caption of Fig. 5.1. For NIS, the following parameters were used: `coupling_layers=2`, `subnet_depth=3`, `subnet_width=8`, `activation=relu`, `bins=20`, `learning_rate=0.01`, `batch_size=4096`, `epochs=100`, `iterations/epoch=1`.

This matches the behavior of the NIS model applied to the 2d double Gaussian example. Evidently the NIS model suffers from the generation of significant noise in general when applied to non-trivial two-dimensional targets.

For this set of physical parameters, the NIS model was able to adapt the structure of the target better than VEGAS did, but the NIS model suffers from the generation of noise in the whole domain, which VEGAS is able to avoid in some regions. Both methods did not exactly capture the structure of the target distribution.

To quantify the proposal performance, the weights of the samples generated from the VEGAS and NIS proposals are compared in Fig. 5.15. Since the shown weights are normalized to the integral of the target distribution, an ideal proposal will have a narrow distribution around 1.0. This is not the case for either of the methods. For both distributions, the majority of the weights are smaller than 50, but for VEGAS, there are a few outliers in the order of 10^4. Because the unweighting efficiency is given as the mean weight divided by the maximum weight, these outliers significantly decrease the unweighting efficiency.

Additionally, the samples have been filtered by a threshold value of 0.1 to gain insight into which parts of the weight distribution correspond with which samples. The resulting filtered sample distributions are shown in Fig. 5.16. The largest number of samples is generated with weights of almost 0 for both methods as can be seen in Fig 5.15. For VEGAS, this equates to samples generated in the ghost feature regions; for NIS, these samples represent the noise in the whole domain as can be seen in Fig. 5.16a. Also, an overestimation of the distribution at $\cos\theta \approx 1.0$ by the NIS model can be observed, which indicates that the model has difficulties

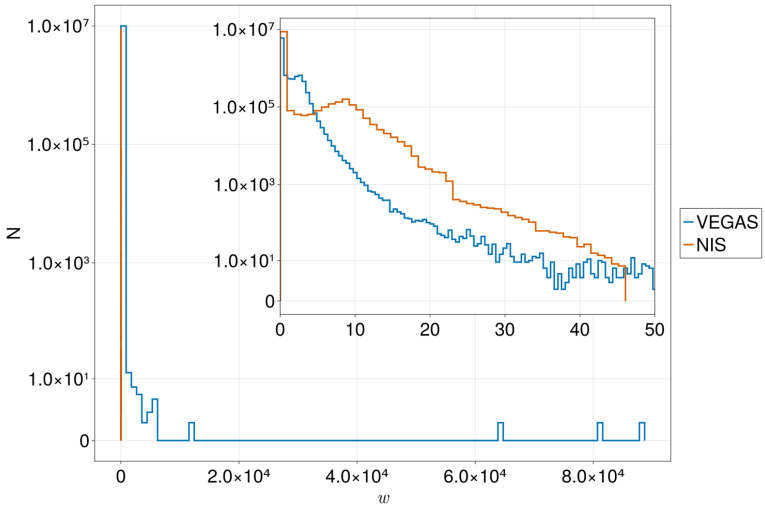
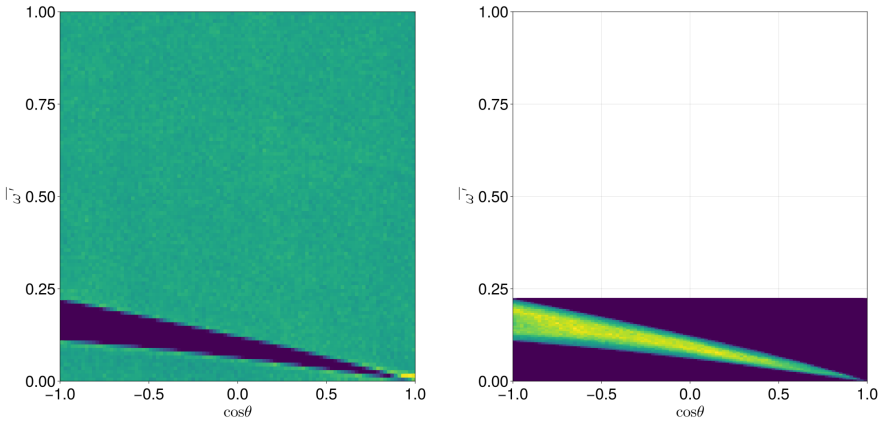


Figure 5.15: Comparison of the sample weight distributions of the VEGAS and NIS proposal samples for the pulsed-perturbative Compton differential cross section, $N = 10^7$ samples generated, w normalized to the integral of the target.

adapting very narrow features. The weight distribution of VEGAS has a second peak around $w = 2$. These are the samples correctly generated in the left peak of the target. VEGAS adapted that region well, as the corresponding weights are close to 1.0. For the weight distribution of NIS, there is a wider second peak centered around $w = 10$. These samples are located in the correctly region of non-vanishing target, as can be seen in Fig. 5.16b. Since the proposal does not exactly match the target structure in that region, the weights are larger than in the VEGAS case.

For VEGAS, the normalized weights have a mean of 0.039 and a maximum of 2358. This results in an unweighting efficiency of $1.66 \cdot 10^{-5}$. For NIS, the normalized weights have a mean of 0.039 and a maximum of 1.8. This results in an unweighting efficiency of 0.021. Although both methods have a similar mean weight, they result from different imperfections in the proposal of each method. Further, NIS is able to avoid the generation of samples with very large weights, since the model is not limited by any specific low dimensional projection and can therefore change its parameters during training in a way that prevents samples with such high weights from being generated. This is reflected in the drastic difference in maximum weights and, therefore, unweighting efficiency. For this set of physical parameters for the differential cross section of the pulsed-perturbative Compton process, NIS is able to provide an unweighting efficiency increase over VEGAS of 3 orders of magnitude.

However, the training time and number of function evaluations also have to be considered when comparing the two methods. VEGAS was trained for ten iterations with 10000 function evaluations each, resulting in a total of $1 \cdot 10^5$.



(a) $w < 0.1$ (b) $w \geqslant 0.1$

Figure 5.16: Qualitative comparison of samples generated from the NIS proposal for the pulsed-perturbative Compton differential cross section filtered by their weights. $N = 10^7$ samples generated before filtering.

The training of NIS converged after about 100 epochs with a batch size of 4096, resulting in a total of $4 \cdot 10^5$ function calls, which is larger than for VEGAS, but it is important to consider how batch sizes are chosen in NIS. As long as the GPU can process the whole batch in parallel, different batch sizes have no significant impact on the runtime. But since larger batches result in fewer training iterations being required to achieve the same optimization, batch sizes are chosen as large as possible when training is performed on a GPU[2]. Consequently, the total number of single-threaded function evaluations is increased, but the total training time stays the same. The training took 25s on a single GPU[3], whereas the VEGAS algorithm took 1.1s to terminate and 5.0s to generate 10^7 samples on a single CPU core[4]. On the available GPU, a batch size of 4096 could be used before running into memory limitations, which is smaller by a factor of four than the batch size used for the Gaussian examples. This is due to the increased complexity of the target, which increases the complexity of its automatic differentiation and, therefore, memory requirements. In cases where the evaluation of the target takes significantly longer than the calculation of the model gradient, the total number of function calls should be used to compare the two methods. But in this case, the time needed to calculate the gradient outweighs the target evaluation time significantly. Therefore, training time is used as a metric to compare the methods

[2]In some applications, extremely large batch sizes can be detrimental to model training [40]. This has not been observed in the present thesis.

[3]NVIDIA Tesla V100, 32 GB VRAM, 14 TFLOPS single precision, 7 TFLOPS double precision

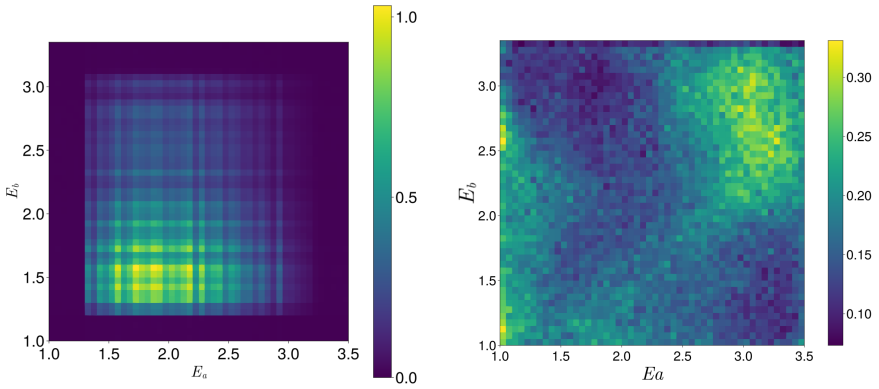
[4]Note that this implementation of VEGAS runs single-threaded on one CPU[5] core, so the runtime is not directly comparable to the vectorized GPU implementation of NIS. However, since there is no available vectorized implementation of VEGAS in Julia, only CPU times can be stated.

in the present thesis. Consequently, NIS took about 23 times longer to train than VEGAS. However, the speed to generate samples is about equal so if large amounts of samples needed to be generated and that sampling time outweighs the training time, then the unweighting efficiency is still the most important metric. And the NIS proposal is able to provide more efficient proposal, it still outperforms VEGAS for large sample sizes.

To summarize, for training time increase of one order of magnitude, NIS is able to provide a proposal that increases the unweighting efficiency by three orders of magnitude over VEGAS while not missing features of the target as VEGAS did and not generating ghost features. However, there is still room for improvement in the generated noise and adapted structure of the target distribution.

5.2.2 Trident

The trident process, as described in Sec. 2.3, will be investigated as a second physical process. A photon energy of $\omega = 5.12\ m_e$ in the rest-frame of the electron is chosen, as the resulting center-of-momentum energy $\sqrt{s} = 3.35$ is above the threshold of $\sqrt{s} = 3$ for the trident process. The threshold energy is reachable at the European XFEL combined with an electron beam of $E = 50$ MeV [16]. The differential cross section has been implemented in a vectorized way to allow efficient sampling and training on the process. For the same GPU as used in the previous examples, the evaluation of the differential cross section for 10^4 samples takes 98ms. For the same batch of samples, the calculation takes 2020ms on a single CPU core. A speedup of a factor of 20 is achieved by using the GPU for this implementation. However, the code is not yet optimized, and due to the incompatibility between different involved code packages, only a small fraction of the GPU could be utilized. Further, extensive efforts were dedicated to implement the differential cross section in a way that allows for automatic differentiation of the function, so that it can be used to train the NIS model. These efforts were partially successful, as the differential cross section can be auto-differentiated. However, due to further incompatibilities between multiple involved packages, the automatic differentiation only works on the CPU. See [41] for a discussion on one of the encountered incompatibilities. Therefore, the NIS model is trained on the CPU, which limits the number of samples that can be evaluated for training.



(a) VEGAS

(b) NIS

Figure 5.17: Comparison of the samples generated from the VEGAS and NIS proposals for the perturbative trident process. Projected onto the energy of the outgoing positron E_a and the energy of an outgoing electron E_b. For VEGAS, the default parameters were used. For NIS, the following parameters were used: `coupling_layers=4`, `subnet_depth=3`, `subnet_width=32`, `activation=relu`, `bins=20`, `learning_rate = 0.01`, `batch_size = 1024`, `epochs=600`, `iterations/epoch=1`.

The samples generated from the VEGAS and NIS proposals for the perturbative trident process are shown in Fig. 5.17 in a projection onto the energies of the positron and an electron. Evidently, VEGAS has identified a region of allowed phase space. The regions with no samples correspond with phase space points that are kinematically forbidden, as has been verified with [42]. However, the NIS model was not able to adapt the target distribution in a meaningful way. Different choices for hyperparameters and model architectures did not show an improvement. Apparently, the optimizer cannot find the minimum of the loss function in this case. A possible explanation is that the gradient in the forbidden phase space regions becomes zero and infinite at the boundaries of the allowed phase space. This might prevent the optimizer from finding the minimum of the loss function through the gradient. Further investigation is required to understand the behavior of the NIS model in this case.

6 Conclusion and Outlook

The aim of the present thesis was to investigate a machine learning enhanced method to generate proposal distributions for the use in Monte Carlo integration and Monte Carlo event generation. The performance of the NIS architecture was investigated on different non-physical and physical distributions and compared with the VEGAS algorithm. For both methods, the proposal distributions show different kinds of biases for every investigated application. It is therefore crucial to unweight the generated proposal samples before they can be used in simulations of physical processes. The weight distributions and unweighting efficiencies of the two methods were investigated as a measure of the quality of the proposal distributions. For the two-dimensional Gaussian distributions, both methods have shown a similar performance in terms of the unweighting efficiency. However, the two methods show different imperfections in the generated samples, which are reflected in the weight distributions. Whereas VEGAS generates ghost features in regions of a vanishing target distribution, NIS generates a considerable amount of noise in the two-dimensional application. For the five-dimensional double Gaussian distribution, a significant difference in the performance of the two methods was observed. Although the VEGAS algorithm is able to produce samples with a normalized mean weight close to one, few outliers with very large weights greatly reduce the unweighting efficiency. Compared to the lower dimensional cases, the NIS model shows a reduction in the generated noise in five dimensions. The combination of this weak noise and the ability to adapt non-factorizing targets without creating ghost features results in a higher unweighting efficiency of the NIS proposal compared to VEGAS.

For the pulsed-perturbative Compton process, a speedup of three orders of magnitude was observed for the NIS model over the VEGAS algorithm. This increase in performance can be explained by the ability of the NIS model to correctly adapt non-factorizing targets and, therefore, avoid the generation of samples with large weights. The increased training time of NIS is outweighed by the increase in unweighting efficiency over VEGAS.

Due to the ability of neural networks to approximate complicated functions, the NIS model is able to adapt non-factorizing target distribution better than the VEGAS algorithm. Therefore, it is a promising method for the importance sampling of differential cross sections with high-dimensional phase spaces.

For the perturbative trident process, the NIS model was not able to produce an efficient proposal distribution. It has been discovered that the architecture is not suitable to directly generate samples for distributions that feature kinematically forbidden regions. Therefore, an algorithm that can map the output of the NIS

model to kinematically allowed phase space points is crucial for the application of the NIS model to processes like the trident process. The 'RAMBO on diet' algorithm[43] provides such a method. However, this algorithm is not yet implemented in the Julia programming language. Further, it needs to be implemented in a vectorized and auto-differentiable way.

For the perturbative trident process, the NIS model was not able to adapt the target due the presumed inability to leave kinematically forbidden phase space regions. An additional algorithm is required that maps the output of the NIS model to allowed phase space points.

It has also become clear, that the vectorized and especially the auto-differentiable implementation of physical processes is crucial for the future possibility of using machine learning enhanced methods in combination with these processes. As has been shown in the present thesis, such methods can significantly increase the efficiency of event generation. However, these capabilities will only be able to be utilized for more complicated processes if special care is taken to ensure the compatibility of such process implementation with machine learning.

Interesting points of future investigation would be the cause for lower noise generation in higher dimensions with NIS, the improvement in training when discarding the momentum of the Adam optimizer, and the application of NIS to the trident process and other high-dimensional scattering processes once an algorithm for phase space mapping is implemented.

A Appendix

A.1 Derivations

A.1.1 Compton Scattering Equation

The equation for the energy of the outgoing photon in the Compton process can be obtained in the following way.

$$\begin{aligned}\vec{p'}^{\,2}c^2 &= \omega^2 + \omega'^2 - 2c^2\vec{k}\vec{k'} \\ &= \omega^2 + \omega'^2 - 2\omega\omega'\cos\theta\end{aligned}$$

with the scattering angel θ between $\vec{k}$ and $\vec{k'}$. Conservation of energy yields:

$$\begin{aligned}\omega + mc^2 &= \omega' + \sqrt{m^2c^4 + +\vec{p'}^{\,2}c^2} \\ \omega' &= \frac{\omega}{1 + \frac{\omega}{mc^2}(1-\cos\theta)}\end{aligned} \tag{A.1}$$

A.1.2 Jacobian determinants

Using equation (2.30), the jacobian determinant for the transformation from $\mathrm{d}\sigma/\mathrm{d}\ell\mathrm{d}\cos\theta$ to $\mathrm{d}\sigma/\mathrm{d}\omega'\mathrm{d}\cos\theta$ can be obtained using the following derivates:

$$\begin{aligned}\frac{\mathrm{d}\omega'}{\mathrm{d}\ell} &= \frac{\omega m^2}{(m+\omega\ell(1-c\theta))^2} \\ \frac{\mathrm{d}\omega'}{\mathrm{d}c\theta} &= \frac{\omega^2\ell^2 m}{(m+\omega\ell(1-c\theta))^2} \\ \frac{\mathrm{d}c\theta}{\mathrm{d}\ell} &= -\frac{m}{\omega\ell^2} \\ \frac{\mathrm{d}c\theta}{\mathrm{d}c\theta} &= 1\,.\end{aligned}$$

The Jacobian determinant follows as

$$\begin{aligned}\left|\frac{\mathrm{d}\omega'\mathrm{d}c\theta}{\mathrm{d}\ell\mathrm{d}c\theta}\right| &= \begin{vmatrix}\frac{\mathrm{d}\omega'}{\mathrm{d}\ell} & \frac{\mathrm{d}\omega'}{\mathrm{d}c\theta} \\ \frac{\mathrm{d}c\theta}{\mathrm{d}\ell} & \frac{\mathrm{d}c\theta}{\mathrm{d}c\theta}\end{vmatrix} \\ &= \frac{2\omega m^2}{(m+\omega\ell(1-c\theta))^2}\,.\end{aligned} \tag{A.2}$$

Using equations (2.30) and (2.33), the Jacobian determinant for the transformation from $\mathrm{d}\sigma/\mathrm{d}\omega'\mathrm{d}\cos\theta$ to $\mathrm{d}\sigma/\mathrm{d}\overline{\omega'}\mathrm{d}\cos\theta$ can be obtained using the following derivates:

$$\begin{aligned}
\frac{\mathrm{d}\overline{\omega'}}{\mathrm{d}\omega'} &= \frac{1-c\theta}{m} \\
\frac{\mathrm{d}c\theta}{\mathrm{d}\omega'} &= \frac{\mathrm{d}}{\mathrm{d}\omega'}\left(1-\frac{m}{\omega'}+\frac{m}{\omega\ell}\right) \\
&= \frac{m}{\omega'^2} \\
\frac{\mathrm{d}\overline{\omega'}}{\mathrm{d}c\theta} &= \frac{-\omega'}{m} \\
\frac{\mathrm{d}c\theta}{\mathrm{d}c\theta} &= 1\,.
\end{aligned}$$

The Jacobian determinant follows as

$$\begin{aligned}
\left|\frac{\mathrm{d}\overline{\omega'}\mathrm{d}c\theta}{\mathrm{d}\omega'\mathrm{d}c\theta}\right| &= \begin{vmatrix} \frac{\mathrm{d}\overline{\omega'}}{\mathrm{d}\omega'} & \frac{\mathrm{d}\overline{\omega'}}{\mathrm{d}c\theta} \\ \frac{\mathrm{d}c\theta}{\mathrm{d}\omega'} & \frac{\mathrm{d}c\theta}{\mathrm{d}c\theta} \end{vmatrix} \\
&= \frac{1-c\theta+\frac{m}{\omega'}}{m}\,. \qquad \text{(A.3)}
\end{aligned}$$

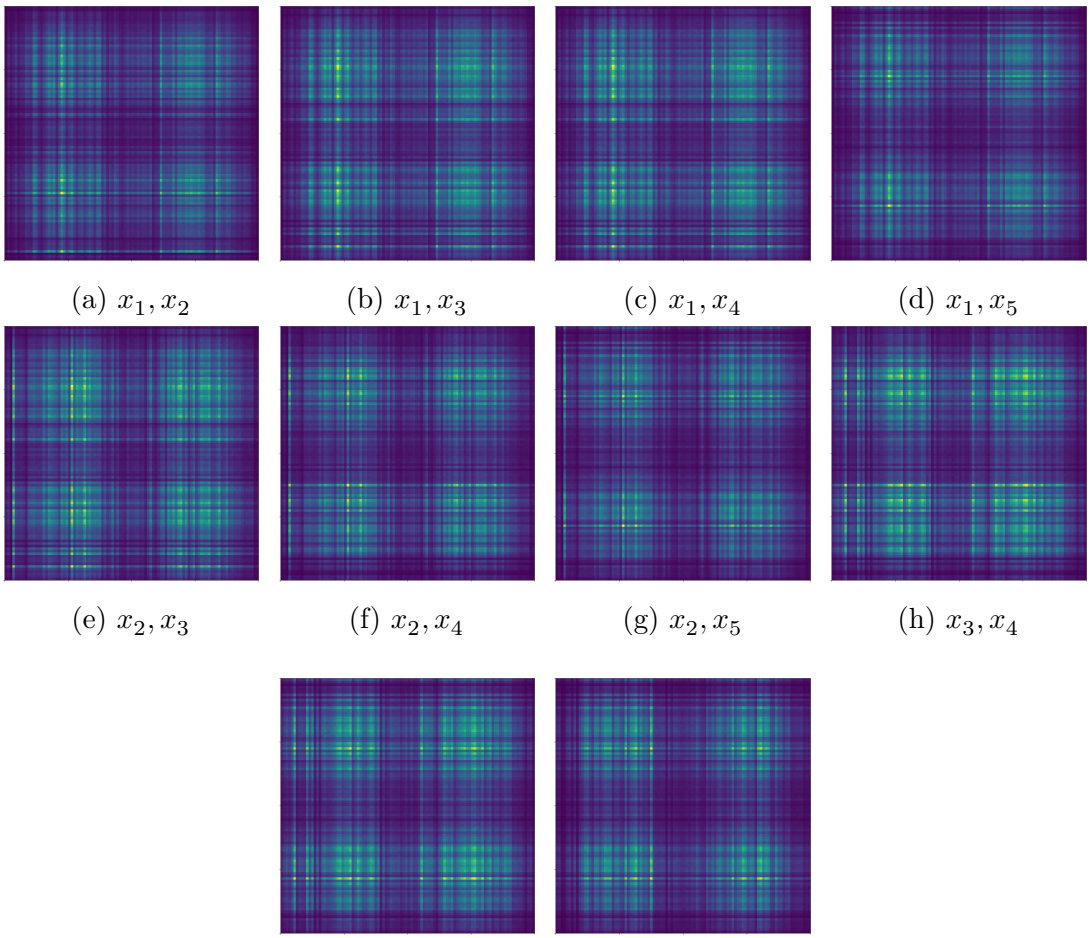
A.1.3 Gaussian projections

For normalized projections of the single or double Gaussian distributions used in Sec. 5, no summation/integration is necessary and the lower dimensional version of the distributions can be used instead. This can be seen by the following example for the projection of the 3d double Gaussian onto one dimension.

$$\begin{aligned}
\int \mathrm{d}x_2\mathrm{d}x_3 f(x_1,x_2,x_3) &= \int \mathrm{d}x_2\mathrm{d}x_3 e^{-\left(\sum_{i=1}^{3}\frac{(x_i-0.75)^2}{0.2^2}\right)} + e^{-\left(\sum_{i=1}^{3}\frac{(x_i-0.25)^2}{0.2^2}\right)} \\
&= \quad e^{-\frac{(x_1-0.75)^2}{0.2^2}} \int \mathrm{d}x_2\mathrm{d}x_3 e^{-\frac{(x_2-0.75)^2}{0.2^2}} e^{-\frac{(x_3-0.75)^2}{0.2^2}} \\
&\quad + e^{-\frac{(x_1-0.25)^2}{0.2^2}} \int \mathrm{d}x_2\mathrm{d}x_3 e^{-\frac{(x_2-0.25)^2}{0.2^2}} e^{-\frac{(x_3-0.25)^2}{0.2^2}} \\
&= Cf(x_1)
\end{aligned}$$

After dividing by the constant C, the projection is equivalent to the one-dimensional function. The same can be done for other projections.

A.2 Figures

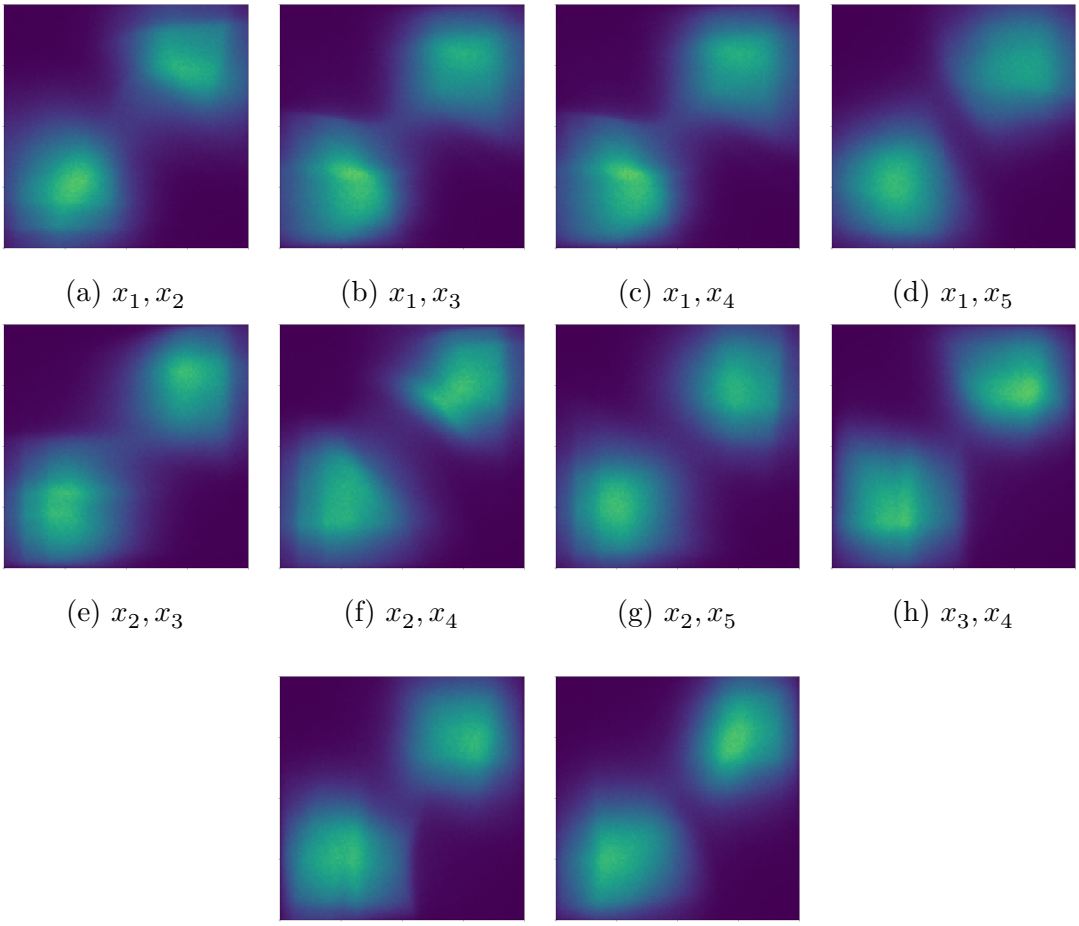


(a) x_1, x_2 (b) x_1, x_3 (c) x_1, x_4 (d) x_1, x_5

(e) x_2, x_3 (f) x_2, x_4 (g) x_2, x_5 (h) x_3, x_4

(i) x_3, x_5 (j) x_4, x_5

Figure A.1: Samples generated from the VEGAS proposal for the 5d double Gaussian distribution in different projections. $N = 10^7$ samples generated, axis ticks and heatmap colors are the same as in Fig. 5.12.

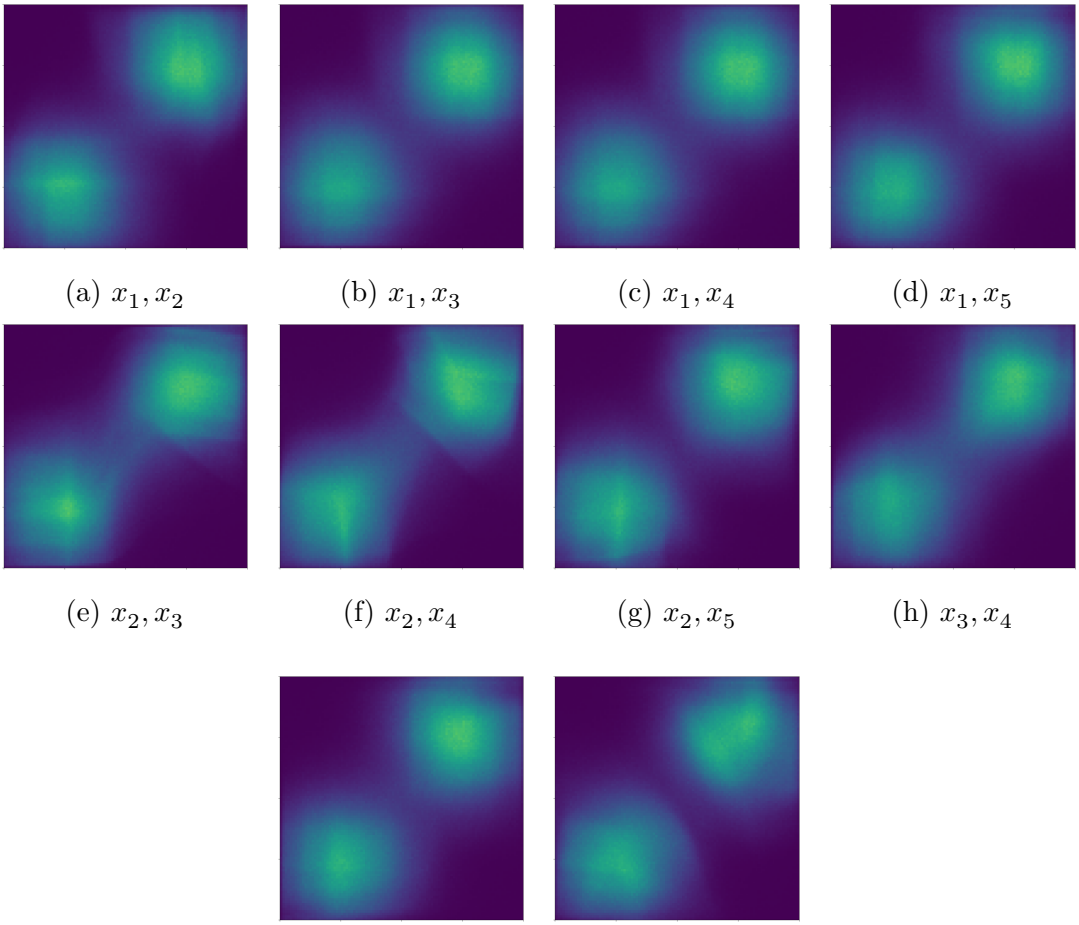


(a) x_1, x_2 (b) x_1, x_3 (c) x_1, x_4 (d) x_1, x_5

(e) x_2, x_3 (f) x_2, x_4 (g) x_2, x_5 (h) x_3, x_4

(i) x_3, x_5 (j) x_4, x_5

Figure A.2: Samples generated from the NIS proposal for the 5d double Gaussian distribution in different projections. $N = 10^7$ samples generated, axis ticks and heatmap colors are the same as in Fig. 5.12.

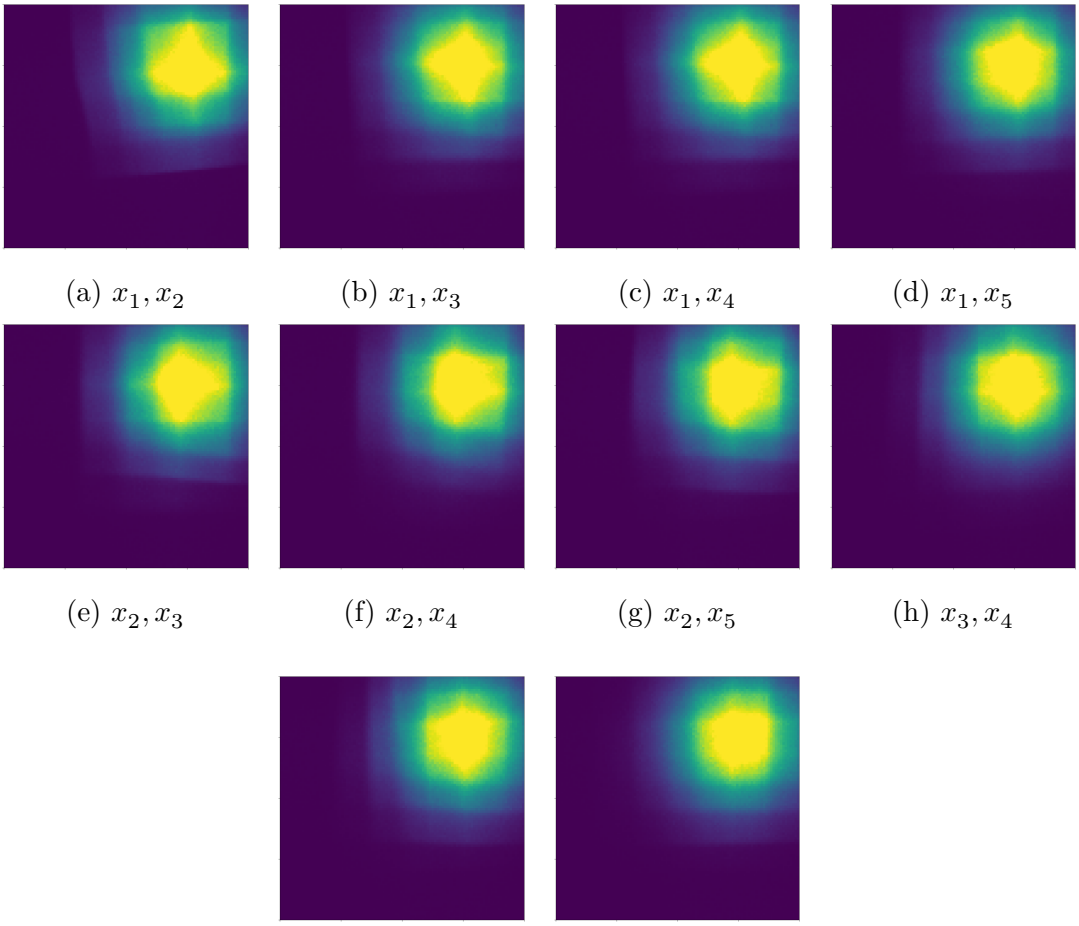


(a) x_1, x_2 (b) x_1, x_3 (c) x_1, x_4 (d) x_1, x_5

(e) x_2, x_3 (f) x_2, x_4 (g) x_2, x_5 (h) x_3, x_4

(i) x_3, x_5 (j) x_4, x_5

Figure A.3: Samples generated from the six coupling layer NIS proposal for the 5d double Gaussian distribution in different projections. $N = 10^7$ samples generated, axis ticks and heatmap colors are the same as in Fig. 5.12.



(a) x_1, x_2 (b) x_1, x_3 (c) x_1, x_4 (d) x_1, x_5

(e) x_2, x_3 (f) x_2, x_4 (g) x_2, x_5 (h) x_3, x_4

(i) x_3, x_5 (j) x_4, x_5

Figure A.4: Samples generated from the NIS proposal without a batch normalization layer for the 5d double Gaussian distribution in different projections. $N = 10^7$ samples generated, axis ticks and heatmap colors are the same as in Fig. 5.12.

A.3 QED.jl

QED.jl is an open-source Julia ecosystem for the modeling of particle physics processes and laser-matter interactions. It serves fundamental data structures and interfaces for strong-field QED processes and event generation. For the implementation of the trident process in Julia as part of the present thesis, implementations for particle spinors, Lorentz vectors, and gamma matrices, as well as the operations defined on them, were imported from QED.jl and adapted to be auto-differentiable on GPUs. Also, QED.jl was enhanced via various code contributions, i.e., to the sampler interface of QEDevents.jl. The NIS sampler presented in this thesis will eventually be part of the event-generation workflow of QED.jl and was Therefore

also written in Julia.

A.4 Julia

Julia is a high-level programming language that combines the ease of use of dynamic languages like Python with the performance of statically typed languages like C. Various packages have been helpful in the implementation of NIS, most notably:

Flux.jl [44, 45] is a machine learning framework similar to the popular Python libraries PyTorch or TensorFlow. It provides various useful tools for the implementation and training of neural networks. Since there was no template for the special architecture of NIS, it had to be implemented from scratch. Flux.jl provided the basic functionality of dense layers and, most importantly, training functionality. With the help of the included automatic differentiation package Zygote.jl [46], the models were trained.

DrWatson [47] was used to organize the project and provide a reproducible workflow. Using DrWatson, the developed code can easily be compiled with all required dependencies and matching versions. Also, all produced results have been tagged with the respective commit hash of the code used to produce them. Therefore, all data presented in this thesis can be reproduced.

CUDA.jl [36, 37] provides an interface for CUDA programming in Julia. It allowed the utilization of Nvidia GPUs for highly parallel execution and fast model training.

Makie.jl [48] is the plotting package used to produce the plots of all results.

QEDbase.jl [17] is part of QED.jl [49] and provides functionality for basic implementations, like four-momenta and spinors, that were used to implement the trident process.

A.5 Implementation

In the present thesis, NIS had to be implemented fully from scratch in Julia. There is a Python implementation [50] available providing NIS functionality, which uses PyTorch as its backend. However, this code was neither ported nor wrapped because the different backend interfaces (Flux.jl vs PyTorch) would be cumbersome to match. Therefore, coupling layers, including the coupling transform, channel mappings, network architectures, loss functions, and a training workflow were implemented. Also, all analyzed distributions, such as the differential cross sections of the Compton and trident process needed to be implemented, vectorized, and tested. Further, a system to manage all generated results, store the used parameters, and save/load the model states were implemented to ensure reproducibility. The final code used to obtain the presented results can be found in the following repository: `https://github.com/tjungni/MasterThesisTomJungnickel`

A.6 GPU vectorization in Julia and auto differentializability

After the code had been tested thoroughly on the CPU, it had to be modified to run in parallel and run on GPUs. Vectorizing code can be trivial in Julia, thanks to the built-in broadcasting functionality. In principle, any complicated function that takes single Floats as input can be vectorized just by adding a broadcast dot to the function call. For the trident process differential cross sections, it would look like this

```
single threaded: dsigmadT(omega, Ea, ctha, phia, Eb, cthb)
multi threaded: dsigmadT.(omega_array, Ea_array, ctha_array, phia_array,
 Eb_array, cthb_array)
```

where the inputs in the second call are arrays of Floats instead of single Floats. This modification is often already sufficient when these functions just need to be evaluated However, for the training of the network, the target function, as well as the network, have to be autodifferentiated by Zygote. Also, to be able to run on GPUs, the code has to be compatible with CUDA. With these two requirements, there were a lot more restrictions on the code. For example, broadcasting the entire cross section function is not possible because Zygote cannot differentiate through it. Therefore, it had to be rewritten with broadcasts for all atomic operations inside as well as various tricks to circumvent different problems with CUDA and Zygote.[1]

Most importantly, ubiquitous operations like for-loops, if-else-statements, scalar indexing of arrays, and array mutation had to be entirely avoided, which required some creative workarounds. This is also the reason why the code is not as readable as it initially was for single threading.

[1] In one case, a bug emerged, where sums with too many elements would crash, which was the case in the summing over all spin/polarization combinations. In another case, the on-shell checking inside the spinor constructor had to be disabled because it had the possibility to print a string in case of off-shell values. And even if this would never be triggered, the existence of this string inside a to-be-broadcast function prevented the compilation with CUDA.jl.

Bibliography

[1] M. Altarelli, R. Brinkmann, and M. Chergui, "The european x-ray free-electron laser. technical design report," (2007).

[2] T. Heinzl and A. Ilderton, "A lorentz and gauge invariant measure of laser intensity," Optics communications **282**, 1879 (2009).

[3] A. Di Piazza, C. Müller, K. Hatsagortsyan, and C. H. Keitel, "Extremely high-intensity laser interactions with fundamental quantum systems," Reviews of Modern Physics **84**, 1177 (2012).

[4] A. Gonoskov, T. Blackburn, M. Marklund, and S. Bulanov, "Charged particle motion and radiation in strong electromagnetic fields," Reviews of Modern Physics **94**, 045001 (2022).

[5] A. Fedotov, A. Ilderton, F. Karbstein, B. King, D. Seipt, H. Taya, and G. Torgrimsson, "Advances in qed with intense background fields," Physics Reports **1010**, 1 (2023).

[6] U. Hernandez Acosta and B. Kämpfer, "Strong-field qed in the furry-picture momentum-space formulation: ward identities and feynman diagrams," Phys. Rev. D **108**, 016013 (2023).

[7] U. H. Acosta and B. Kämpfer, "Laser pulse-length effects in trident pair production," Plasma Physics and Controlled Fusion **61**, 084011 (2019).

[8] K. J. Mork, "Pair production by photons on electrons," Physical Review **160**, 1065 (1967).

[9] J. Joseph and F. Rohrlich, "Pair production and bremsstrahlung in the field of free and bound electrons," Reviews of Modern Physics **30**, 354 (1958).

[10] J. Motz, H. A. Olsen, and H. Koch, "Pair production by photons," Reviews of Modern Physics **41**, 581 (1969).

[11] T. Arber, K. Bennett, C. Brady, A. Lawrence-Douglas, M. Ramsay, N. J. Sircombe, P. Gillies, R. Evans, H. Schmitz, A. Bell, et al., "Contemporary particle-in-cell approach to laser-plasma modelling," Plasma Physics and Controlled Fusion **57**, 113001 (2015).

[12] G. Cybenko, "Approximation by superpositions of a sigmoidal function," Mathematics of control, signals and systems **2**, 303 (1989).

[13] C. Itzykson and J.-B. Zuber, *Quantum field theory* (Courier Corporation, 2012).

[14] L. H. Ryder, *Quantum field theory* (Cambridge university press, 1996).

[15] M. E. Peskin, *An introduction to quantum field theory* (CRC press, 2018).

[16] U. H. Acosta, "Pulsed-perturbative qed: a study of trident pair production in pulsed laser fields," PhD thesis (Technische Universität Dresden, 2020).

[17] U. H. Acosta, *Qedbase.jl*, version v0.1.2, `https://gitlab.hzdr.de/QEDjl/QEDbase.jl`, Oct. 2021.

[18] S. Weinzierl, "Introduction to monte carlo methods," arXiv preprint hep-ph/0006269 (2000).

[19] G. E. Forsythe, "Von neumann's comparison method for random sampling from the normal and other distributions," Mathematics of Computation **26**, 817 (1972).

[20] G. P. Lepage, "A new algorithm for adaptive multidimensional integration," Journal of Computational Physics **27**, 192 (1978).

[21] Z.-H. Zhou, *Machine learning* (Springer Nature, 2021).

[22] F. Malard, L. Danner, E. Rouzies, J. G. Meyer, E. Lescop, and S. Olivier-Van Stichelen, "Epynn: educational python for neural networks," SoftwareX **19**, 101140 (2022).

[23] Baeldung, *Neural network architecture: criteria for choosing the number and size of hidden layers*, `https://www.baeldung.com/cs/neural-networks-hidden-layers-criteria`, Accessed: 2024-01-24.

[24] M. Herlihy, N. Shavit, V. Luchangco, and M. Spear, *The art of multiprocessor programming* (Newnes, 2020).

[25] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in International conference on machine learning (pmlr, 2015), pp. 448–456.

[26] R. Zaheer and H. Shaziya, "A study of the optimization algorithms in deep learning," in 2019 third international conference on inventive systems and control (icisc) (IEEE, 2019), pp. 536–539.

[27] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," arXiv preprint arXiv:1412.6980 (2014).

[28] R. Kleiss and R. Pittau, "Weight optimization in multichannel monte carlo," Computer Physics Communications **83**, 141 (1994).

[29] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," J. Mach. Learn. Res. **22** (2021).

[30] T. Müller, B. Mcwilliams, F. Rousselle, M. Gross, and J. Novák, "Neural importance sampling," ACM Trans. Graph. **38**, `10.1145/3341156` (2019).

[31] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale, and S. Schumann, "Exploring phase space with neural importance sampling," SciPost Physics **8**, 069 (2020).

[32] L. Dinh, D. Krueger, and Y. Bengio, "Nice: non-linear independent components estimation," arXiv preprint arXiv:1410.8516 (2014).

[33] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," arXiv preprint arXiv:1605.08803 (2016).

[34] V. Fisikopoulos and L. Penaranda, "Faster geometric algorithms via dynamic determinant computation," Computational Geometry **54**, 1 (2016).

[35] C. Gao, J. Isaacson, and C. Krause, "I-flow: high-dimensional integration and sampling with normalizing flows," Machine Learning: Science and Technology **1**, 045023 (2020).

[36] T. Besard, C. Foket, and B. De Sutter, "Effective extensible programming: unleashing Julia on GPUs," IEEE Transactions on Parallel and Distributed Systems, 10.1109/TPDS.2018.2872064 (2018).

[37] T. Besard, V. Churavy, A. Edelman, and B. De Sutter, "Rapid software prototyping for heterogeneous and distributed platforms," Advances in Engineering Software **132**, 29 (2019).

[38] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," Journal of Marchine Learning Research **18**, 1 (2018).

[39] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," Neural networks **1**, 295 (1988).

[40] Y. You, Y. Wang, H. Zhang, Z. Zhang, J. Demmel, and C.-J. Hsieh, "The limit of the batch size," arXiv preprint arXiv:2006.08517 (2020).

[41] *Many basic cuda functions do not work with zygote*, https://github.com/FluxML/Zygote.jl/issues/730.

[42] E. Haug, "Bremsstrahlung and pair production in the field of free electrons," Zeitschrift für Naturforschung A **30**, 1099 (1975).

[43] S. Plätzer, "Rambo on diet," arXiv preprint arXiv:1308.2922 (2013).

[44] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah, "Fashionable modelling with flux," CoRR **abs/1811.01457** (2018).

[45] M. Innes, "Flux: elegant machine learning with julia," Journal of Open Source Software, 10.21105/joss.00602 (2018).

[46] M. Innes, "Don't unroll adjoint: differentiating ssa-form programs," CoRR **abs/-1810.07951** (2018).

[47] G. Datseris, J. Isensee, S. Pech, and T. Gál, "Drwatson: the perfect sidekick for your scientific inquiries," Journal of Open Source Software **5**, 2673 (2020).

[48] S. Danisch and J. Krumbiegel, "Makie.jl: flexible high-performance data visualization for Julia," Journal of Open Source Software **6**, 3349 (2021).

[49] U. H. Acosta, *Qed.jl*, https://github.com/QEDjl-project/QED.jl.

[50] N. Deutschmann and N. Götz, *Zünis*, https://github.com/ndeutschmann/zunis.

List of Figures

Acknowledgements

I want to thank my supervisor Dr. Uwe Hernandez Acosta for his support and guidance not only during the course of this work, but also my previous few years at HZDR. I am grateful for the time he invested in mentoring me and for the valuable feedback he provided. I also want to thank Prof. Thomas E. Cowan and Dr. Frank Siegert for agreeing to review this thesis and the various people at HZDR for the valuable discussions, special thanks goes to Simeon Ehrig for his technical support and Dr. Peter Steinbach for his great machine learning lecture. Thank you to Marius Melcher for proofreading, Wojciech Kolodziej for supporting me through tough times and a very big thank you to my cat Fussel for her loving support ♡

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit im Rahmen der Betreuung am Helmholtz-Zentrum Dresden-Rossendorf ohne unzulässige Hilfe Dritter verfasst habe und alle verwendeten Quellen als solche gekennzeichnet habe.

Ort, Datum Unterschrift