How to GAN

Novel simulation methods for the LHC

Dissertation

Ramon Peter Winterhalder

Dissertation

submitted to the

Combined Faculty of Natural Sciences and Mathematics
of Heidelberg University, Germany

for the degree of

Doctor of Natural Sciences

Put forward by

Ramon Winterhalder

born in Titisee-Neustadt, Germany

Oral examination: November 4th, 2020

How to GAN

Novel simulation methods for the LHC

Referees: Prof. Dr. Tilman Plehn
Prof. Dr. Jan Pawlowski

Abstract

Various aspects of LHC simulations can be supplemented by generative networks. For event generation we show how a GAN can describe the full phase space structure of top-pair production including intermediate on-shell resonances and phase space boundaries. In order to resolve these sharp peaking features, we introduce the maximum mean discrepancy. Additionally, the architecture can be extended in a straightforward manner to improve the network performance and to handle weighted events in the training data. Furthermore, we employ GANs to generate new events which are distributed according to the sum or difference of the input data. We first show with the help of a toy example how such a network can beat the statistical limitations of bin-wise subtraction methods. Afterwards we demonstrate how this network can subtract background events or describe collinear subtraction events in next-to-leading order calculations.
Finally, we show how detector simulations can be inverted using GANs and INNs. They allow us to reconstruct parton level information from measured events. In detail, our results show how conditional generative networks can invert Monte Carlo simulations statistically. INNs even allow for a statistical interpretation of single-event unfolding and yield the possibility to unfold parton showering.

Zusammenfassung

Verschiedene Aspekte von LHC-Simulationen können durch generative Netzwerke ergänzt werden. Im Bereich von Eventgeneration zeigen wir, wie ein GAN die vollsändige Phasenraumstruktur für Top-Quark-Paarproduktionen beschreiben kann. Insbesondere zeigen wir, wie GANs auch Resonanzen und Phasenraumgrenzen beschreiben kann. Um diese lokal begrenzten Strukturen aufzulösen, führen wir die Maximum-Mean-Discrepany ein. Darüber hinaus kann die Netzwerkarchitektur auf einfache Art und Weise erweitert werden, um sowohl die Simulationsgenauigkeit zu verbessern als auch mit gewichteten Events in den Trainingsdaten umzugehen.
Des Weiteren nutzen wir GANs um neue Events zu generieren, die gemäß der Summe beziehungsweise der Differenz der Inputdaten verteilt sind. Wir zeigen zunächst anhand eines Beispiels wie solche Netzwerke die statistischen Limitationen von histogrammbasierten Subtraktionsmethoden lösen können. Anschließend demonstrieren wir wie diese Netzwerke sowohl Hintergrundevents subtrahieren als auch kollineare Subtraktionsevents, die für störungstheoretische Berechnungen höherer Ordnung benötigt werden, beschreiben kann.
Zum Schluss zeigen wir wie Detektorsimulationen mit Hilfe von GANs und INNs invertiert werden können. Diese machen es möglich Parton-Level-Informationen aus den gemessenen Daten zu rekonstruieren. Im Detail zeigen unsere Ergebnisse wie konditionale generative Netzwerke Monte-Carlo-Simulationen statistisch invertieren können. Mit INNs können wir sogar die Inversion von Einzelevents statistisch interpretieren und Parton-Shower auf den harten Streuprozess abbilden.

Contents

Preface

This thesis is based on research conducted between 2017 and 2020 at the Institute for Theoretical Physics at Heidelberg University. The presented results in Chapter 3–5 were achieved in collaboration with other researchers and have previously been published as:

[1] A. Butter, T. Plehn and R. Winterhalder,
How to GAN LHC Events,
SciPost Phys. 7, 075 (2019), arXiv:1907.03764,

[2] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn and R. Winterhalder,
How to GAN away Detector Effects,
SciPost Phys. 8, 070 (2020), arXiv:1912.00477,

[3] A. Butter, T. Plehn and R. Winterhalder,
How to GAN Event Subtraction,
to appear in SciPost Phys., arXiv:1912.08824,

[4] M. Bellagente, A. Butter, G. Kasieczka, A. Rousselot,
T. Plehn, R. Winterhalder, L. Ardizzone and U. Köthe,
Invertible Networks or Partons to Detector and Back Again,
to appear in SciPost Phys., arXiv:2006.06685,

Finally, the author is involved in ongoing projects that have not been ready for publication at the time of writing this thesis.

Introduction

Since time immemorial, we are interested in understanding what matter is composed of. In ancient history, philosophers coined the term *atom* to describe the undividable units making up matter. However, these ideas were rather motivated by philosophical and theological reasoning than by physics and experiments. It was not until the 19th century that the idea was finally refined by scientists when discoveries emerged only explainable by the concept of atoms. Since then much has been learned by more advanced experiments and the concept of atoms has been replaced by a more elaborate model describing the elementary constituents of matter and their interactions. This model is called the Standard Model of particle physics [5–11].

The Standard Model (SM) describes three of the four known fundamental forces of nature: electromagnetism, the weak interaction and the strong force. The matter content consists of spin-$\frac{1}{2}$ fermions and their interactions are described by the exchange of spin-1 force carriers called gauge bosons. The photon is responsible for the electromagnetic interactions, the massive $W^{\pm}$ and Z bosons are the carriers of the weak force, and eight gluons mediate the strong force.

The terms *matter*, *particle* and *force* are handy and appear to be vivid and tangible. Nevertheless, they are merely metaphoric expressions of mathematical quantities appearing in a sophisticated model. Consequently, it is better suited to describe the SM in the language of mathematics. In detail, the SM is formulated as a quantized field theory being symmetric under the Poincaré group and the $SU(3)_{\mathrm{C}} \times SU(2)_{\mathrm{W}} \times U(1)_{\mathrm{Y}}$ gauge group. In fact, the unitary irreducible representations of the proper and orthochronous Poincaré group are labeled by the squared mass m^2 and spin s which suggests the concept of particles.

Since the 1970s, the SM received great recognition by an enormous number of experiments showing remarkable agreement with the theoretical predictions. Most recently, the discovery of the Higgs boson in 2012 by the ATLAS and CMS collaborations [12–14] which was predicted by the SM as a consequence of the Higgs mechanism [15–17], was a milestone in particle physics for both experiment and theory.

The discovery of the Higgs boson would not have been possible without theory and experiment working hand in hand. In particular, analyzing the huge amount of recorded data in a proper and quantitative manner requires an equally large amount of synthetic data originating from complex simulations. Most commonly, these simulations are performed with Monte Carlo methods. In typical LHC analyses we need to employ a complex tool chain which combines several individual Monte Carlo simulations.

In the most simple scenario, we are only interested in the integrated cross section which can be compared to event rates. However, the calculation of hadronic scattering cross sections requires a numerical phase space integration [18–22] due to high-dimensional integrands, complicated matrix elements and the convolution with PDFs. In general, the integrated cross section is not sufficient to compare measured data with theoretical

models. Hence, in more elaborate analyses we usually need simulated events which were generated under the assumption of a specific model hypothesis [23, 24]. Furthermore, if we want to describe physical objects which can be actually detected in experimental setups, we need to translate parton level objects into particle level objects. Therefore, besides the simulation of hard matrix elements, we also need to add parton showering including hadronization and fragmentation [25, 26]. Finally, we need to parametrize the detector response and hence simulate the interactions of particle level objects with the detector [27, 28].

In order to push these theory predictions to high precision sophisticated methods are required. Therefore, most of the currently used tools either adopt self-adaptive sampling techniques like `Vegas` [29, 30] and multi-channel methods [31, 32], or put a lot of effort into hand-crafted phase space mappings [33, 34]. However, owing to the complexity of the tool chain described above, these methods are still computationally expensive and somewhat inefficient. For instance, to compare measured and simulated data in a simple but reasonable manner, we need to translate the weighted Monte Carlo events into unweighted events. However, the unweighting efficiencies for typical LHC processes are successively decreasing [35, 36] and thus limit the statistical significance of the analyses.

Some of these shortcomings might be diminished when we employ modern machine learning techniques. Boosted decision trees and neural networks have been proposed to improve phase space sampling [37–41] and aim to increase the unweighting efficiency. On the other hand, generative networks such as generative adversarial networks (GANs) [42] and invertible neural networks (INNs) [43–45] are promising attempts to describe various aspects of LHC simulations. For instance, generative networks have already been used for detector simulations [46–51], the description of parton showers [52–56], event generation [57–59], or searches for physics beyond the Standard Model [60].

In this thesis we propose to use GANs and INNs for three different tasks: event generation, sample-based subtraction of distributions, and detector unfolding. For event generation we consider top-pair production

$$pp \to t\bar{t} \to (bq_1\bar{q}_1') \, (\bar{b}\bar{q}_2 q_2'),$$

with sharp phase space structures originating from intermediate resonances. In order to resolve these resonances, we follow an entirely novel approach by adding a maximum mean discrepancy (MMD) [61] term to the generator loss. This enables us to extract the peak structure in a completely dynamical manner, not requiring further physical input such as the mass or width of the on-shell resonance.

Furthermore, we consider the $W + 2$ jet production process

$$pp \to W^+ jj$$

and investigate how more involved network architectures can increase the precision of our GAN based event generation. For instance, we require the network to obey 4-momentum conservation and on-shell conditions for the final-state particles by generating only the necessary degrees of freedom. We show how simple and generalizable modifications can lead to a significant improvement in the network performance.

Additionally, we also examine the Drell–Yan production process

$$pp \to \mu^- \mu^+,$$

which is now represented by a set of weighted events. Therefore, we derive a more generalized GAN which can handle weighted events as training data while still producing

unweighted events. This property yields a new contribution to the machine learning tool box and extends the abilities of standard GAN frameworks used for event generation.

Since experiments at the LHC become more and more precise, theory needs to keep up with this precision. Thus, the event samples which are used for data-to data comparisons have to be generated beyond leading order in perturbation theory. However, modern simulation techniques require the inclusion of subtraction terms in order to make the entire prediction finite. Typically, these subtraction terms appear in fixed-order real emission calculations [62–66], multi-jet merging including parton showering [67, 68], on-shell subtraction [69], or the subtraction of known backgrounds [70].

As neural networks possess excellent interpolation properties, we employ GANs to generate events representing either the subtraction or addition of two distributions. As a physical application we consider the subtraction of the photon-continuum contribution from the full e^+e^- production process in Drell–Yan scattering, and the subtraction of collinear gluon radiation in Z+jet production to obtain finite real emission contributions.

As we have already pointed out, a standard LHC analysis requires us to perform the entire event generation tool chain from the hard matrix element to the complex detector simulation for every new model hypothesis. This approach is extremely inefficient. Consequently, we want to invert parts of the simulation chain such that we can compare the measured data and the theoretical hypothesis at the level of the hard scattering process. To this account, we propose GANs and INNs to invert the detector simulation for the process

$$pp \to ZW^+ \to (\ell^-\ell^+)\,(jj). \tag{0.1}$$

In particular, we show how a fully conditional GAN [71] (FCGAN) and conditional INN (cINN) [72, 73] learn to map detector-level events onto parton-level events in a structured and statistically meaningful manner. We even show, that for a single detector-level event the cINN generates probability distributions in the parton-level phase space.

In detail, the thesis is organized as follows:
In Chapter 1 we give a brief overview of the fundamental ingredients of modern high-energy physics research. In particular, we shortly introduce the Standard Model Lagrangian as the state-of-the-art mathematical description of nature. Afterwards, we give a proper description of common Monte Carlo methods and how they can be used for event generation. We also highlight shortcomings of standard techniques and emphasize the need for modern machine learning approaches. Consequently, in Chapter 2 we give a short introduction into neural networks and explain their basic building blocks as well as typical training algorithms. In the end, we describe the basic elements of GANs and INNs, as they are needed to understand the following chapters. Subsequently, in Chapters 3–5 we present the results which have already been published in Refs. [1–4] as well as the latest developments we are currently working on. In detail, in Chapter 3, we show how GANs can be used to generate events for top-pair production at the LHC [1] and afterwards we show how this method can be extended to obtain higher precision and also cover possible weighted events in the training data. Furthermore, in Chapter 4, we utilize the excellent interpolation properties of neural networks and employ GANs for sample-based subtraction of distributions [3]. Then, in Chapter 5, we show how GANs and INNs can be used to invert detector simulations for a typical LHC process [2, 4]. Finally, we conclude this thesis by giving a short outlook of possible future applications and consequences.

Chapter 1

The HEP Trinity

The two fundamental pillars of high energy physics research are theory and experiment. In theory, we rely on predictions based on a fundamental Langrangian and observables calculated within the framework of a perturbative quantum field theory (QFT). On the experimental side we build large colliders and detectors to measure particle interactions and collect huge amounts of data. In order to analyze experimental measurements and link it with theoretical predictions we need to translate the mathematical description into objects which can be compared to the recorded data in a simple but quantitative manner. Most commonly, the translation or link between theory and experiment is performed with Monte Carlo simulations. For instance, Monte Carlo simulations are used for event generation [23, 24], phase space integration [18–22, 34], parton shower [25, 26] and detector simulations [27, 28].

In this chapter we first discuss the basics of the Standard Model (SM) of particle physics and shortly introduce the SM Lagrangrian in Section 1.1. In our analyses we consider processes and measurements at the Large Hadron Collider (LHC) and hence we need to introduce the parton model and the idea of parton distribution function (PDF) Section 1.2. Afterwards, in Section 1.3 we introduce the differential and total cross section and derive a possible phase space parametrization as this is crucial to understand why and how these quantities are computed numerically. In Section 1.4 we dive into the basics of Monte Carlo simulations which are represent the gold standard for numerical methods. In particular, we explain common integration and sampling procedures, which are needed to understand why we put so much effort into improving these techniques in the rest of our thesis.

1.1 The Standard Model of particle physics

The Standard Model (SM) describes three of the four known fundamental forces of nature: electromagnetism, the weak interaction and the strong force. It is formulated as a consistent quantum field theory (QFT) being symmetric under the Poincaré group and the

$$\mathrm{SU(3)_C} \times \mathrm{SU(2)_W} \times \mathrm{U(1)_Y} \tag{1.1}$$

gauge group. In the following the full Lagrangian of the Standard Model of particle physics will be constructed. A more extended treatment can be found in [74–76].

1.1.1 The Standard Model Lagrangian

The electroweak (EW) sector of the Standard Model is described by the Glashow-Salam-Weinberg (GSW) model [5–7] for electroweak interactions. The underlying gauge group

$\mathrm{SU(2)_W} \times \mathrm{U(1)_Y}$ is spontaneously broken to the electromagnetic $\mathrm{U(1)_Q}$ symmetry. The strong interaction couples to the color quantum numbers and is described by quantum chromodynamics (QCD) [8–11] with the associated gauge group $\mathrm{SU(3)_C}$. The SM Lagrangian can be divided into the following parts,

$$\mathcal{L}_{\mathrm{SM}} = \mathcal{L}_{\mathrm{Gauge}} + \mathcal{L}_{\mathrm{Fermion}} + \mathcal{L}_{\mathrm{Higgs}} + \mathcal{L}_{\mathrm{Yukawa}}, \tag{1.2}$$

which we discuss in the following.

Gauge part

The SM has the underlying gauge group $\mathrm{SU(3)_C} \times \mathrm{SU(2)_W} \times \mathrm{U(1)_Y}$, with gauge fields

$$\mathrm{SU(3)_C}: \quad G_\mu^a, \quad a = 1, \ldots, 8, \tag{1.3a}$$
$$\mathrm{SU(2)_W}: \quad W_\mu^i, \quad i = 1, 2, 3, \tag{1.3b}$$
$$\mathrm{U(1)_Y}: \quad B_\mu. \tag{1.3c}$$

In the following, we discuss the subgroups in more detail:

- $\mathrm{SU(3)_C}$: This group corresponds to the strong interaction, and the label C denotes the color. The color quantum numbers are denoted as red, green, and blue. The quark matter fields are arranged in triplets and transform under the fundamental representation of $\mathrm{SU(3)_C}$. The generators are given by

  $$T_{\mathrm{c}}^a = \frac{\lambda^a}{2} \equiv t^a, \tag{1.4}$$

  where the λ^a are the Gell-Mann matrices. The associated gauge fields are called gluons and denoted as G_μ^a.

- $\mathrm{SU(2)_W}$: The label W denotes the weak isospin. Since the weak interaction violates parity, a separate treatment of left- and right-handed fermions is needed. They can be defined as

  $$\psi_{\mathrm{L}} = \omega_- \psi, \quad \psi_{\mathrm{R}} = \omega_+ \psi, \quad \psi = \psi_{\mathrm{L}} + \psi_{\mathrm{R}}, \tag{1.5}$$

  with the projection operators

  $$\omega_\pm = \frac{1 \pm \gamma^5}{2}, \qquad (\omega_\pm)^2 = \omega_\pm. \tag{1.6}$$

  As only left-(right-)handed (anti-)fermions interact with the charged W boson, the left-handed fermions are $\mathrm{SU(2)_W}$ doublets, while the right-handed fermions are singlets. Since neutrinos are massless in the SM, the right-handed neutrinos $\nu_{i,\mathrm{R}}$ can be left out. The generators for the lef-handed fermions are given by

  $$T_{\mathrm{w}}^i = \frac{\sigma^i}{2} \equiv I_{\mathrm{w}}^i, \tag{1.7}$$

  where σ^i are the Pauli matrices. The associated gauge bosons are denoted as W_μ^i.

- $\mathrm{U(1)_Y}$: The hypercharge phase symmetry is needed for constructing the unbroken electric charge operator Q, which is a linear combination of Y and the third component of the weak isospin I_{w}^3. Since the charge needs to be the same for left- and right-handed fermions, they need to have different hypercharges Y. The associated gauge field is denoted as B_μ.

In order to incorporate dynamics for the gauge fields into the theory, a gauge-invariant kinetic term is added, given by the gauge part

$$\mathcal{L}_{\text{Gauge}} = -\frac{1}{4}G^a_{\mu\nu}G^{a\mu\nu} - \frac{1}{4}W^i_{\mu\nu}W^{i\mu\nu} - \frac{1}{4}B_{\mu\nu}B^{\mu\nu}, \tag{1.8}$$

with field-strength tensors defined as

$$\begin{aligned} G^a_{\mu\nu} &= \partial_\mu G^a_\nu - \partial_\nu G^a_\mu - g_s f^{abc} G^b_\mu G^c_\nu, \\ W^i_{\mu\nu} &= \partial_\mu W^i_\nu - \partial_\nu W^i_\mu + g\varepsilon^{ijk} W^j_\mu W^k_\nu, \\ B_{\mu\nu} &= \partial_\mu B_\nu - \partial_\nu B_\mu. \end{aligned} \tag{1.9}$$

Here, f^{abc} and ε^{ijk} are the structure constants of the SU(3) and SU(2) algebras, respectively. The coupling constant of the $\mathrm{U(1)_Y}$ subgroup is denoted as g'. As the $\mathrm{U(1)_Y}$ subgroup is Abelian, the coupling only appears in the definition of the covariant derivative.

Fermionic part

The matter content of the SM is built up from fermions, which can be further subdivided into leptons (ν_i,ℓ_i), only interacting electroweakly, and quarks (u_i,d_i), being also part of the strong interaction. The fermions appear in three different generations, differing only in their mass and flavor. The three generations of leptons and fermions are given in Table 1.1 with their representation under the SM gauge group (1.1) and EW quantum numbers.

fields		generation			representation	charges		
		1st	2nd	3rd		I^3_{w}	Y	Q
leptons	$L'_{i,\mathrm{L}}$	$\begin{pmatrix}\nu'_e\\ e'\end{pmatrix}_{\mathrm{L}}$	$\begin{pmatrix}\nu'_\mu\\ \mu'\end{pmatrix}_{\mathrm{L}}$	$\begin{pmatrix}\nu'_\tau\\ \tau'\end{pmatrix}_{\mathrm{L}}$	$(\mathbf{1},\mathbf{2})_{-1}$	$\frac{1}{2}$ $-\frac{1}{2}$	-1 -1	0 -1
	$\ell_{i,\mathrm{R}}$	e'_{R}	μ'_{R}	τ'_{R}	$(\mathbf{1},\mathbf{1})_{-2}$	0	-2	-1
quarks	$Q'_{i,\mathrm{L}}$	$\begin{pmatrix}u'\\ d'\end{pmatrix}_{\mathrm{L}}$	$\begin{pmatrix}c'\\ s'\end{pmatrix}_{\mathrm{L}}$	$\begin{pmatrix}t'\\ b'\end{pmatrix}_{\mathrm{L}}$	$(\mathbf{3},\mathbf{2})_{\frac{1}{3}}$	$\frac{1}{2}$ $-\frac{1}{2}$	$\frac{1}{3}$ $\frac{1}{3}$	$\frac{2}{3}$ $-\frac{1}{3}$
	$u_{i,\mathrm{R}}$	u'_{R}	c'_{R}	t'_{R}	$(\mathbf{3},\mathbf{1})_{\frac{4}{3}}$	0	$\frac{4}{3}$	$\frac{2}{3}$
	$d_{i,\mathrm{R}}$	d'_{R}	s'_{R}	b'_{R}	$(\mathbf{3},\mathbf{1})_{-\frac{2}{3}}$	0	$-\frac{2}{3}$	$-\frac{1}{3}$

Table 1.1: The matter content of the SM for all $i = 1, 2, 3$ generations of the leptons and quarks. The representation assignments of the fields with respect to the SM gauge group is denoted by $(\mathrm{SU(3)_C}, \mathrm{SU(2)_W})_{\mathrm{U(1)_Y}}$. I^3_{w} and Ydenote the third component of the weak isospin and the hypercharge, respectively. The electric charge Q is related to the above via Eq. (1.13).

Since the left- and right-handed fermions have different transformation properties under the $\mathrm{SU(2)_W} \times \mathrm{U(1)_Y}$ group, naive mass terms like

$$-m\bar{\psi}\psi = -m\left(\bar{\psi}_{\mathrm{L}}\psi_{\mathrm{R}} + \bar{\psi}_{\mathrm{R}}\psi_{\mathrm{L}}\right) \tag{1.10}$$

are forbidden. For including fermion masses Yukawa couplings need to be introduced which will be discussed in a separate section below. Without the mass term, the fermionic part of the Lagrangian is defined by

$$\mathcal{L}_{\text{Fermion}} = \sum_i \left(\bar{L}'_{i,\text{L}} \not{D} L'_{i,\text{L}} + \bar{Q}'_{i,\text{L}} \not{D} Q'_{i,\text{L}} + \bar{\ell}'_{i,\text{R}} \not{D} l'_{\ell,\text{R}} + \bar{u}'_{i,\text{R}} \not{D} u'_{i,\text{R}} + \bar{d}'_{i,\text{R}} \not{D} d'_{i,\text{R}} \right), \quad (1.11)$$

with the covariant derivative

$$D_\mu = \partial_\mu + \mathrm{i} g_s t^a G^a_\mu - \mathrm{i} g I^i_{\text{w}} W^i_\mu + \mathrm{i} g' \frac{Y}{2} B_\mu, \quad (1.12)$$

where the representations of the $\text{SU}(3)_\text{C} \times \text{SU}(2)_\text{W} \times \text{U}(1)_\text{Y}$ gauge group need to be considered for each fermion field individually. The hypercharge Y and the third component of the weak isospin I^3_w are related to the electric charge Q via the Gell-Mann–Nishijima relation [77, 78]

$$Q = I^3_\text{w} + \frac{Y}{2}. \quad (1.13)$$

It is constructed in such a way that the electric charge of the left- and right-handed fermions are equal. We should also pay attention to the primes on the fermionic field. The primes indicate that the fields are eigenstates of the EW interaction. In contrast, unprimed fields denote mass eigenstates. In general those states are different, which has also been observed in experiment. How they can be related to each other will be discussed in the Yukawa part of the SM.

Spontaneous symmetry breaking and the Higgs mechanism

As the introduction of naive mass terms for the gauge bosons would spoil gauge invariance we need to employ spontanous symmetry breaking (SSB) of a local gauge theory. In the presence of SSB, the full Lagrangian is assumed to be invariant under a symmetry group $\mathcal{G}$, while the vacuum of the theory only respects the symmetry of a subgroup $\mathcal{H} \subset \mathcal{G}$, denoted as little group or stability group. After SSB the gauge fields associated with the unbroken little group remain massless whereas the other gauge fields become massive. This mechanism is called Brout-Englert-Higgs mechanism [15–17]. In the SM a scalar $\text{SU}(2)_\text{W}$ doublet is introduced which is conventionally parametrized as follows,

$$\Phi(x) = \begin{pmatrix} \phi^+(x) \\ \phi^0(x) \end{pmatrix}, \quad (1.14)$$

with charged complex scalar fields ϕ^+ and ϕ^0, transforming in the $(\mathbf{1}, \mathbf{2})_1$ representation. The most general Higgs Lagrangrian then takes the form

$$\mathcal{L}_{\text{Higgs}} = (D_\mu \Phi)^\dagger (D^\mu \Phi) + \mu^2 (\Phi^\dagger \Phi) - \frac{\lambda}{4} (\Phi^\dagger \Phi)^2, \quad (1.15)$$

where the parameters λ and μ^2 are real since $\mathcal{L} = \mathcal{L}^\dagger$. Stability of the potential requires $\lambda > 0$ and for SSB we need $\mu^2 > 0$. For the chosen potential, the vacuum expectation value Φ_0 must satisfy

$$|\Phi_0|^2 = \Phi_0^\dagger \Phi_0 = \frac{2\mu^2}{\lambda} = \frac{v^2}{2}. \quad (1.16)$$

When switching into the unitary gauge the would-be Goldstone fields [79,80] vanish and the Higgs doublet can be parametrized as

$$\Phi = \frac{1}{\sqrt{2}} \begin{pmatrix} 0 \\ v + H(x) \end{pmatrix}. \tag{1.17}$$

In order to work out the gauge boson masses we first define the linear combinations of the fields

$$W_\mu^\pm = \frac{1}{\sqrt{2}}(W_\mu^1 \mp \mathrm{i} W_\mu^2), \tag{1.18}$$

which correspond to the electric charge eigenstates

$$Q W_\mu^\pm = I_\mathrm{w}^3 W_\mu^\pm = \pm W_\mu^\pm, \tag{1.19}$$

where the electric charge Q is the unbroken generator of the stability subgroup $\mathcal{H}$ given by

$$Q \equiv T_\mathrm{unbroken} = I_\mathrm{w}^3 + \frac{Y}{2}. \tag{1.20}$$

The mass matrix that is generated from the kinetic term in Eq. (1.15) for the B_μ and W_μ^3 fields, will be diagonalized by

$$\begin{pmatrix} A_\mu \\ Z_\mu \end{pmatrix} = \begin{pmatrix} \cos\theta_\mathrm{w} & -\sin\theta_\mathrm{w} \\ \sin\theta_\mathrm{w} & \cos\theta_\mathrm{w} \end{pmatrix} \begin{pmatrix} B_\mu \\ W_\mu^3 \end{pmatrix}, \tag{1.21}$$

with the weak mixing angle or Weinberg angle θ_w given by

$$\cos\theta_\mathrm{w} \equiv c_\mathrm{w} = \frac{g}{\sqrt{g^2 + (g')^2}}, \qquad \sin\theta_\mathrm{w} \equiv s_\mathrm{w} = \frac{g'}{\sqrt{g^2 + (g')^2}}. \tag{1.22}$$

The electric unit charge e and hence the fine-structure constant α are fixed by the couplings g and g' via the relations

$$e = \frac{g g'}{\sqrt{g^2 + (g')^2}}, \qquad \alpha \equiv \frac{e^2}{4\pi}. \tag{1.23}$$

Considering only the bilinear terms in the Higgs lagrangian we can read off the boson masses

$$M_W = \frac{vg}{2} = \frac{ve}{2s_\mathrm{w}}, \qquad M_Z = \frac{M_W}{c_\mathrm{w}}, \qquad M_H = \sqrt{2}\mu. \tag{1.24}$$

The first line in the Lagrangian corresponds to a neutral scalar particle H being called Higgs boson. The other lines correspond to the charged W bosons $W_\mu^\pm$ and the neutral Z boson Z_μ. The A_μ field associated with the unbroken generator Q represents the photon γ and remains massless.

Fermion masses and Yukawa couplings

As the different transformation properties of the left- and right-handed fermions forbid naive mass terms, the fermion masses are introduced by means of Yukawa couplings of

the Higgs doublet Φ to all fermions. Therefore, we define the charge conjugate of the Higgs doublet

$$\Phi^{\mathrm{c}} = \mathrm{i}\sigma^2\Phi^\dagger = ((\phi^0)^*, -\phi^-) \tag{1.25}$$

The Lagrangian takes the form

$$\mathcal{L}_{\mathrm{Yukawa}} = -\sum_{i,j=1}^{3}\left(\bar{L}'_{i,\mathrm{L}}G^{\ell}_{ij}\ell'_{j,\mathrm{R}}\Phi + \bar{Q}'_{i,\mathrm{L}}G^{u}_{ij}u'_{j,\mathrm{R}}\Phi^{\mathrm{c}} + \bar{Q}'_{i,\mathrm{L}}G^{d}_{ij}d'_{j,\mathrm{R}}\Phi + \mathrm{h.c.}\right). \tag{1.26}$$

In general, the Yukawa couplings G^f are complex 3×3 matrices, giving rise to a large number of free parameters. However, it turns out that most of them are not physically relevant. The vacuum expectation value (vev) part of the Higgs doublet generates mass terms of the fermions that can be diagonalized by two unitary matrices U^f_{L} and U^f_{R}

$$m_{f,i}\delta_{ij} = \frac{v}{\sqrt{2}}\sum_{k,l=1}^{3} U^f_{\mathrm{L},ik}G^f_{kl}U^{f\dagger}_{\mathrm{R},lj}, \tag{1.27}$$

relating the primed EW eigenstates and the unprimed mass eigenstates

$$f_{i,\mathrm{L}} = \sum_{j=1}^{3} U^f_{\mathrm{L},ij}\, f'_{j,\mathrm{L}}, \qquad f_{i,\mathrm{R}} = \sum_{j=1}^{3} U^f_{\mathrm{R},ij}\, f'_{j,\mathrm{R}}. \tag{1.28}$$

In the original formulation of the SM all neutrinos ν_i are massless[1]. Thus, we choose the mass eigenstates to be the same as the EW eigenstates by choosing the $\nu'_{i,\mathrm{L}}$ to transform in the same way as the $\ell'_{i,\mathrm{L}}$

$$\nu_{i,\mathrm{L}} = \sum_{j=1}^{3} U^{\ell}_{\mathrm{L},ij}\nu'_{j,\mathrm{L}}. \tag{1.29}$$

Hence all $U^{\ell}_{\mathrm{L},ij}$ in the Lagrangian cancel in the transition to mass eigenstates. However, the situation is different for the quarks. Here, there are masses for both u_i and d_i species, and we would like do diagonalize both. This means that the transformations U^u_{L} and U^d_{L} are independent of each other, giving no coherent transformation of $Q'_{i,\mathrm{L}}$ as a whole. Then the charged-current (cc) interaction terms with the $W^{\pm}_{\mu}$ are not invariant and the matrices do not cancel. The cc interaction terms are

$$\mathcal{L}_{\mathrm{Yukawa,cc}} = \frac{g}{\sqrt{2}}\sum_{i,j=1}^{3}\left(\bar{u}_{i,\mathrm{L}}\gamma^{\mu}\underbrace{(U^u_{\mathrm{L}}U^{d\dagger}_{\mathrm{L}})_{ij}}_{=V_{\mathrm{CKM}}}d_{j,\mathrm{L}}W^{+}_{\mu} + \bar{d}_{i,\mathrm{L}}\gamma^{\mu}\underbrace{(U^d_{\mathrm{L}}U^{u\dagger}_{\mathrm{L}})_{ij}}_{=V^{\dagger}_{\mathrm{CKM}}}u_{j,\mathrm{L}}W^{-}_{\mu}\right), \tag{1.30}$$

where the quark-mixing matrix

$$U^u_{\mathrm{L}}U^{d\dagger}_{\mathrm{L}} \equiv V_{\mathrm{CKM}} = \begin{pmatrix} V_{ud} & V_{us} & V_{ub} \\ V_{cd} & V_{cs} & V_{cb} \\ V_{td} & V_{ts} & V_{tb} \end{pmatrix} \tag{1.31}$$

is called the Cabibbo-Kobayashi-Maskawa (CKM) [82, 83] matrix being responsible for flavor transitions in the charged weak interactions. The CKM matrix can be parametrized in terms of 4 independent parameters: three angles and one complex phase, where the latter is the only source of $\mathcal{CP}$ violation in the SM.

[1] Indeed, after the observation of neutrino oscillations we know that neutrinos need to be massive [81].

1.2 The parton model

When considering scattering processes at the LHC we always consider proton-proton collisions. However, from deep inelastic scattering (DIS) [84] of electrons off nuclei we know that protons are not fundamental particles. In fact they have a rather complex substructure. Hence, if we want to investigate LHC processes in our analyses we need to take this into account and find an appropriate description. The first experiments leading to a simplified description of the dynamics have been realized at Stanford Linear Accelerator Center (SLAC) in the 1960s and in the 1970s. There, electrons with up to 25 GeV have been scattered off nuclei in a linear collider. Further DIS measurements have been performed with very high precision at the HERA collider at Deutsches Elektronen-Synchrotron (DESY) in Hamburg by the H1 [85] and ZEUS [86, 87] collaborations.

1.2.1 Bjorken scaling and naive parton model

When considering inelastic electron-nuclei scattering, the dynamics of the nuclei be be described by structure functions W_1 and W_2. Using these structure funcitons, the differential cross section for unpolarized electrons can be written as [88]

$$\frac{\mathrm{d}^2\sigma}{\mathrm{d}\Omega \mathrm{d}E'} = \frac{Z^2\alpha^2}{4E^2 \sin^4\frac{\theta}{2}} \left[W_2(Q^2,\nu)\, \cos^2\frac{\theta}{2} + 2W_1(Q^2,\nu)\, \sin^2\frac{\theta}{2} \right], \tag{1.32}$$

where W_1 and W_2 are functions of the two invariants

$$Q^2 = -q^2, \qquad \nu = \frac{Pq}{M} = E - E', \tag{1.33}$$

where q and P are the momenta of the photon and proton, M is the proton mass, and E and E' are the energies of the electron before and after the scattering, respectively. For large values of Q^2 and ν it has been seen in experiment that W_1 and νW_2 only depend on

$$x = \frac{Q^2}{2Pq} = \frac{Q^2}{2M\nu}, \tag{1.34}$$

The structure functions $W_1(Q^2,\nu)$ and $W_2(Q^2,\nu)$ are then denoted as $F_1(x)$ and $F_2(x)$, depending only on x:

$$\begin{aligned} \lim_{Q^2,\nu\to\infty} W_1(Q^2,\nu) &= F_1(x), \\ \lim_{Q^2,\nu\to\infty} \nu W_2(Q^2,\nu) &= F_2(x). \end{aligned} \tag{1.35}$$

This effect is called Bjorken scaling [89]. The variable x is also denoted as Bjorken variable. Richard Feynman suggested a vivid model for deep inelastic scattering: the parton model. There the proton is considered in an infinite momentum frame, in which the transverse momenta and rest masses of the constituents can be neglected. These constituents are called partons. Each parton a carries the fraction x_a ($0 \leq x_a \leq 1$) of the total proton momentum P,

$$p_a = x_a P, \qquad \text{with} \sum_k x_k = 1. \tag{1.36}$$

The momentum fraction x_a can be identified with the Bjorken variable x, where x is the momentum fraction the lepton is scattered off. The number density to find a parton a

with the momentum fraction x_a in a hadron A is described by the parton distribution function $f_{a|A}$. The structure function F_2 is connected to the PDFs $f_{a|A}$ and the other structure function F_1 via the charges q_a of the partons and the Callan-Gross relation[2], respectively:

$$\begin{aligned} F_2(x) &= \sum_a q_a^2 \, f_{a|A}(x), \\ F_2(x) &= 2xF_1(x), \end{aligned} \tag{1.37}$$

where a runs over all possible partons.

Hadronice scattering process

As an example relevant for our later purposes we now consider the scattering process of two hadrons A and B,

$$A + B \to C + X, \tag{1.38}$$

where C denotes an arbitrary final state we are interested in, and X represents the remnants of the incoming hadrons after the hard scattering. For calculating the hadronic cross section for this scattering process we have to take the sum over all possible partonic scattering reactions weighted by their respective PDFs,

$$\sigma_{AB} = \sum_{a,b} \int_0^1 \mathrm{d}x_a \int_0^1 \mathrm{d}x_b \, f_{a|A}(x_a) f_{b|B}(x_b) \hat{\sigma}_{ab}(x_a, x_b), \tag{1.39}$$

where $\hat{\sigma}_{ab}(x_a, x_b)$ denotes the partonic cross section of partons a and b contributing to the production of the final state C. A graphical representation of Eq. (1.39) is illustrated in Fig. 1.1.

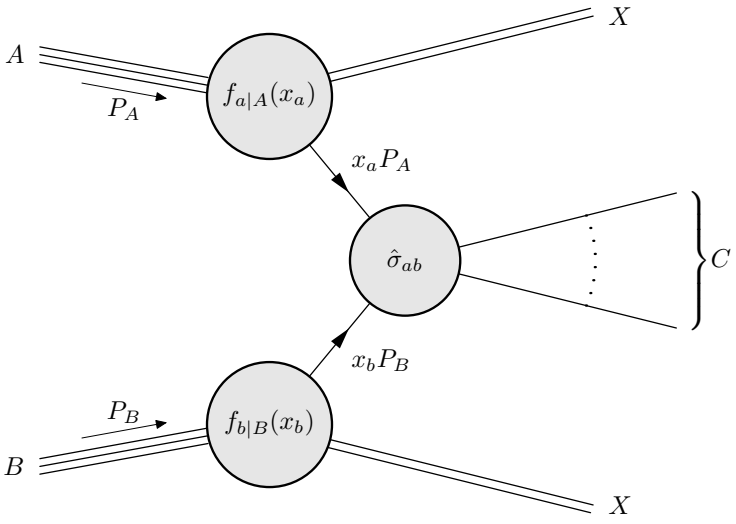

Figure 1.1: Schematic representation of the hadronic scattering process $A+B \to C+X$, where C denotes an arbitrary final state we are interested in and X represents any additional hadronic activity.

[2]The Callan-Gross relation reflects the spin-$\frac{1}{2}$ nature of the partons. For scalar particles we would get $F_1 = 0$. In fact, gluons are also partons but are spin-1 particles. However, this can only be understood within the framework of QCD and the QCD-improved parton model [90].

As PDFs cannot be obtained perturbatively they have to be extracted from experiments. However, they are universal in the sense that they encode the structure of the hadron, but are not dependent on the actual scattering process. Thus, they be can extracted, for instance from deep inelastic e^-p scattering, and used to calculate cross sections for proton-proton collisions.

1.2.2 The QCD improved parton model

In Fig. 1.2 we show, that the gluon and quark PDFs strongly depend on the Bjorken variable x. A more detailed PDF analysis shows, that they also depend on the factorization scale. In fact the statement of Bjorken scaling is only approximately true for specific domains of Q^2.

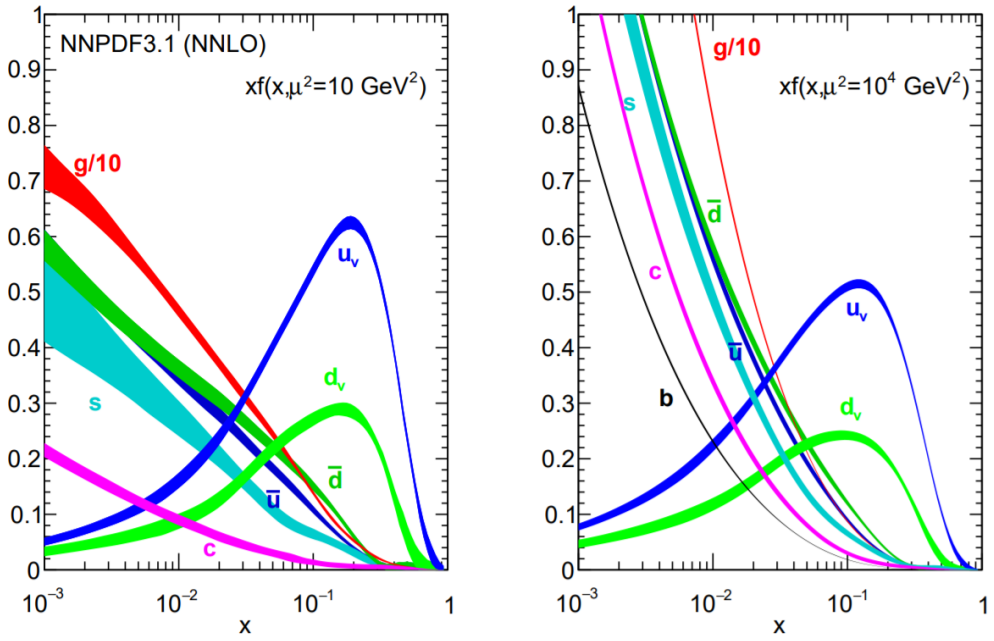

Figure 1.2: PDFs for different partons depending on the Bjorken variable x and the factorisation scale at $\mu_F = 10\,\mathrm{GeV}^2$ (left) and $\mu_F = 10^4\,\mathrm{GeV}^2$ (right) [91].

The naive parton model of Feynman is still the basis for investigating inelastic nuclei scattering. A more advanced description of the proton is given by quantum chromodynamics. QCD extends the naive parton model such that interactions between partons are allowed. This is also called QCD-improved parton model [90], which extends the concept of PDF to gluons and even photons [92, 93].

Using the DGLAP equations (for Dokshitzer, Gribov, Lipatov, Altarelli and Parisi) [94] the evolution of the PDFs, which were determined for a specific energy Q^2, can be calculated to arbitrary factorization scales μ_F. The factorization scale is an almost arbitrary scale where hard processes (partonic cross section) and soft processes (non-pertubative effects) can be factorized. This is the basis of the QCD-improved parton model and generalizes Eq. (1.39) to

$$\sigma_{AB} = \sum_{a,b} \int_0^1 \mathrm{d}x_a \int_0^1 \mathrm{d}x_b \, f_{a|A}(x_a, \mu_F^2) f_{b|B}(x_b, \mu_F^2) \hat{\sigma}_{ab}(x_a, x_b, \mu_F^2). \tag{1.40}$$

The factorization scale μ_F can also be interpreted as the scale that separates the long-distance from the short-distance physics.

1.3 Cross section and kinematics

A crucial aspect of LHC analyses and hence our research is the calculation of observables which can be compared to experimental data. An observable which can be directly linked to event rates and thus detector measurements is the differential or total cross section. In order to understand, how these quantities can be calculated, we first consider the kinematics of scattering processes and the relation between partonic and hadronic processes. Afterwards, we derive the phase space for a general $2 \to n$ scattering process.

1.3.1 Kinematics

The 4-momentum of a particle with mass m is given by

$$p^\mu = (E, \mathbf{p}), \qquad E = \sqrt{|\mathbf{p}|^2 + m^2}, \tag{1.41}$$

and the 3-momentum $\mathbf{p} = (p_x, p_y, p_z)$ can be parametrized as

$$\mathbf{p} = |\mathbf{p}| \, (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta), \tag{1.42}$$

where θ and ϕ denote the polar and azimuth angles in polar coordinates. Since the partonic cross section has to be convoluted with the PDFs, the partonic centre-of-mass system (CMS) is not equal to the hadronic CMS, which we denote as the laboratory (LAB) frame. However, both frames are connected to each other by a Lorentz boost along the beam axis. The phase space is usually parametrized in the partonic CMS. Thus, it s convenient to define variables which have simple transformation property under a boost along the z-direction. Note, that all variables denoted with a hat are defined in the partonic CMS (e.g. $\hat{y}$, $\hat{\theta}$), whereas the observables without a hat are those within the LAB frame. We introduce the following kinematic variables:

$$\text{transverse momentum:} \qquad p_\mathrm{T} = \sqrt{p_x^2 + p_y^2}, \tag{1.43a}$$

$$\text{transverse mass:} \qquad m_\mathrm{T} = \sqrt{p_\mathrm{T}^2 + m^2} \xrightarrow[m\to 0]{} p_\mathrm{T}, \tag{1.43b}$$

$$\text{rapidity:} \qquad y = \frac{1}{2} \ln\left(\frac{E + p_z}{E - p_z}\right), \tag{1.43c}$$

$$\text{pseudorapidity:} \qquad \eta = \frac{1}{2} \ln\left(\frac{|\mathbf{p}| + p_z}{|\mathbf{p}| - p_z}\right) = -\ln\left(\tan\frac{\theta}{2}\right) \xrightarrow[m\to 0]{} y. \tag{1.43d}$$

Using these variables we can also write the 4-momentum as

$$p^\mu = \begin{pmatrix} m_\mathrm{T}\cosh y \\ p_\mathrm{T}\cos\phi \\ p_\mathrm{T}\sin\phi \\ m_\mathrm{T}\sinh y \end{pmatrix} \xrightarrow[m\to 0]{} p_\mathrm{T} \begin{pmatrix} \cosh\eta \\ \cos\phi \\ \sin\phi \\ \sinh\eta \end{pmatrix}. \tag{1.44}$$

Lorentz transformation

The Lorentz boost Λ along the z-axis can be parametrized as

$$\Lambda^\mu{}_\nu(\xi) = \begin{pmatrix} \cosh\xi & 0 & 0 & \sinh\xi \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \sinh\xi & 0 & 0 & \cosh\xi \end{pmatrix}, \tag{1.45}$$

with rapidity ξ . Under this boost the kinematic variables in (1.43) have very simple transformation properties

$$p_{\mathrm{T}} \to p_{\mathrm{T}}, \qquad m_{\mathrm{T}} \to m_{\mathrm{T}}, \qquad y \to y + \xi. \tag{1.46}$$

Note that the pseudorapidity η transforms less simple. In the LAB frame, the 4-momenta of the incoming protons A and B are given by

$$P_A^\mu = (E_{\mathrm{B}}, 0, 0, E_{\mathrm{B}}), \qquad P_B^\mu = (E_{\mathrm{B}}, 0, 0, -E_{\mathrm{B}}) \tag{1.47}$$

where E_{B} is the beam energy and the masses of the protons have been neglected as we consider high energy scattering. Then the centre-of-mass energy is given by

$$E_{\mathrm{CM}} = \sqrt{s} = \sqrt{(P_A + P_B)^2} = 2E_{\mathrm{B}}. \tag{1.48}$$

The 4-momenta of the partons a and b coming from the protons are then obtained by

$$p_a^\mu = x_a P_A^\mu = x_a(E_{\mathrm{B}}, 0, 0, E_{\mathrm{B}}), \qquad p_b^\mu = x_b P_B^\mu = x_b(E_{\mathrm{B}}, 0, 0, -E_{\mathrm{B}}) \tag{1.49}$$

with momentum fractions x_a and x_b. In the partonic CMS the momenta of the incoming partons a and b have the form

$$\hat{p}_a^\mu = (\hat{E}_{\mathrm{B}}, 0, 0, \hat{E}_{\mathrm{B}}), \qquad \hat{p}_b^\mu = (\hat{E}_{\mathrm{B}}, 0, 0, -\hat{E}_{\mathrm{B}}) \tag{1.50}$$

where $\hat{E}_{\mathrm{B}}$ denotes the partonic beam energy. Thus, the squared centre-of-mass (CM) energy of the partonic subprocess is related to the hadronic system with

$$\sqrt{\hat{s}} = \hat{E}_{\mathrm{CM}} = 2\hat{E}_{\mathrm{B}} = \sqrt{(\hat{p}_a + \hat{p}_b)^2} = \sqrt{(p_a + p_b)^2} = \sqrt{x_a x_b s}. \tag{1.51}$$

Thus, the boost parameter ξ in (1.45) is given by

$$\xi = \frac{1}{2} \ln \frac{x_a}{x_b}. \tag{1.52}$$

1.3.2 Phase space parametrization

Let us consider two particles a and b with 4-momenta p_a and p_b being scattered off each other and producing n particles with 4-momenta $\{p_i\}_{i=1}^n$:

$$p_a + p_b \to p_1 + p_2 + \cdots + p_n. \tag{1.53}$$

The differential cross section is then given by [95]:

$$\mathrm{d}\sigma = \frac{\langle |\mathcal{M}(p_1, \ldots, p_n | p_a, p_b)|^2 \rangle}{F(p_a, p_b)} \frac{1}{S_{\{n\}}} \mathrm{d}\Phi_n(p_1, \ldots, p_n | \underbrace{p_a + p_b}_{=:Q}), \tag{1.54}$$

where the different factors are:

- The squared matrix element $\langle |\mathcal{M}|^2 \rangle$: The angular brackets $\langle \cdots \rangle$ denote possible averaging/summation over initial-/final-state degrees of freedom such as spin, helicity and color.

- The flux factor $F(p_a, p_b)$:

$$F(p_a, p_b) = 4\sqrt{(p_a \cdot p_b)^2 - (m_a m_b)^2}. \tag{1.55}$$

  For massless initial states $p_{a,b}^2 = 0$ the flux factor simplifies to

$$F(p_a, p_b) = 4\,(p_a \cdot p_b) = 2s, \tag{1.56}$$

  where s is the squared CM energy defined by $s = (p_a + p_b)^2 = Q^2$.

- The symmetry factor $S_{\{n\}}$: For each set of n_j identical particles in the final state we get a factor $n_j!$ correcting for double counting,

$$S_{\{n\}} = \prod_j n_j!. \tag{1.57}$$

- The Lorentz invariant n-particle phase space density $\mathrm{d}\Phi_n(p_1, \ldots, p_n|Q)$:

$$\begin{aligned} \mathrm{d}\Phi_n(p_1, \ldots, p_n|Q) &= \prod_{i=1}^{n} \left[\frac{\mathrm{d}^4 p_i}{(2\pi)^3} \delta(p_i^2 - m_i^2)\Theta(p_i^0) \right] (2\pi)^4 \delta^{(4)} \left(\sum_{i=1}^{n} p_i - Q \right) \\ &= \prod_{i=1}^{n} \left[\frac{\mathrm{d}^3 \mathbf{p}_i}{(2\pi)^3 2E_i} \right] (2\pi)^4 \delta^{(4)} \left(\sum_{i=1}^{n} p_i - Q \right), \end{aligned} \tag{1.58}$$

  with $E_i = \sqrt{\mathbf{p}_i^2 + m_i^2}$. The Dirac delta- and the Heaviside step functions implement three kinds of kinematical constraints:

  1. Energy and momenta need to be conserved: $p_a + p_b = \sum p_i$.
  2. Each external particle has to be on its mass shell: $p_i^2 = m_i^2$.
  3. The energy must always be positve: $p_i^0 = E_i > 0$.

1.4 Monte Carlo integration

In order to calculate total and differential cross sections the squared amplitude has to be integrated over the full phase space region or, depending on the observable, over a subspace defined by kinematic cuts.

However, owing to the high dimensionality of the integrand, complicated matrix elements, arbitrary phase space cuts, and the convolution with PDFs, the integration can only be done numerically. These numerical integrations are commonly performed with Monte Carlo methods. The Monte Carlo simulations naturally provide us with the possibility to generate unweighted events which are directly comparable to LHC data.

In the following, we briefly introduce the basic concepts of numerical integration using the RAMBO on diet [96] algorithm as an example of plain Monte Carlo. Afterwards, we discuss the advantageous of more sophisticated versions relying on the idea of importance sampling. To this end we present the idea of multi-channel Monte Carlo [19, 31, 32].

We finish this section by taking the step towards actual event generation as this will serve as a basis for our neural network based method introduced in Chapter 3. We will also define the unweighting efficiency as a quality measure to benchmark the perfomance of our approach in Sec. 3.4.

1.4.1 Basic definitions and plain Monte Carlo

Consider we have a function f defined on the d-dimensional domain $\Omega \subseteq \mathbb{R}^d$. We are now interested to perform the integral

$$I = \int_\Omega \mathrm{d}^d\mathbf{x}\, f(\mathbf{x}). \tag{1.59}$$

but we are not able to solve this integration analytically, either because this is unpractical or simply not feasible. Further, we are only given N points $\{\mathbf{x}_i\}_{i=1}^N$ chosen randomly from Ω, which are distributed according to a function $\rho(\mathbf{x})$. Thus, the integral can be approximated by

$$\bar{I}_N = \frac{1}{N}\sum_{i=1}^{N} \frac{f(\mathbf{x}_i)}{\rho(\mathbf{x}_i)}, \tag{1.60}$$

where $\rho(\mathbf{x})$ obeys the normalization equation

$$\int_\Omega \mathrm{d}^d\mathbf{x}\, \rho(\mathbf{x}) = 1. \tag{1.61}$$

Therefore ρ is a probability density function. In the limit of large numbers the estimation converges to the integral I:

$$\lim_{N\to\infty} \bar{I}_N = I. \tag{1.62}$$

For sufficiently large N the variance σ^2 can be approximated by

$$\sigma^2 \simeq \frac{\bar{S}_N - \bar{I}_N^2}{N-1}, \tag{1.63}$$

with

$$\bar{S}_N = \frac{1}{N}\sum_{i=1}^{N} \left(\frac{f(\mathbf{x}_i)}{\rho(\mathbf{x}_i)}\right)^2. \tag{1.64}$$

In the most simple case $\rho(\mathbf{x})$ is constant and hence the integral is evaluated with uniformly drawn points $\{\mathbf{x}_i\}_{i=1}^N \in \Omega$. Let further V be the volume of the domain Ω. In this case $\rho(\mathbf{x})$ is given by

$$\rho(\mathbf{x}) = \frac{1}{V}, \qquad V \equiv \int_\Omega \mathrm{d}^d\mathbf{x}. \tag{1.65}$$

The estimation for the integral is then simply given by

$$\bar{I}_N = \frac{V}{N}\sum_{i=1}^{N} f(\mathbf{x}_i). \tag{1.66}$$

RAMBO on diet

We start by considering a generic change of integration variables $\mathbf{x} \to \mathbf{y} = \mathbf{y}(\mathbf{x})$ and thus we rewrite the integral as

$$I = \int_\Omega \mathrm{d}^d\mathbf{x}\, f(\mathbf{x}) = \int_\Xi \mathrm{d}^d\mathbf{y} \left|\frac{\partial x}{\partial y}\right| f(\mathbf{y}), \tag{1.67}$$

with a function $\mathbf{y} : \Omega \to \Xi \subseteq \mathbb{R}^d$. In general, the Jacobian determinant $\left|\frac{\partial x}{\partial y}\right|$ or weight W is dependent on the specific mapping $\mathbf{y}(\mathbf{x})$ and thus on the new coordinates. However, we are now only interested in mappings in which the determinant or weight is constant.

In particular, in the RAMBO on diet [96] algorithm we try to find a mapping $\mathbf{r} : \Phi_n \to U_n = [0,1]^{3n-4}$ such that the n-particle phase space weight W_{Φ_n} is constant for a given total momentum Q

$$\left|\frac{\partial p}{\partial r}\right| = W_{\Phi_n} = \text{const.} \tag{1.68}$$

and thus resulting in a flat phase space. We recast our result from (1.58) and write the phase space density for massless final-state particles as

$$\mathrm{d}\Phi_n(p_1,\ldots,p_n|Q) = (2\pi)^{4-3n} \prod_{i=1}^{n} \left[\mathrm{d}^4 p_i \delta(p_i^2)\theta(p_i^0)\right] \delta\left(\sum_{i=i}^{n} p_i - Q\right), \tag{1.69}$$

The RAMBO on diet algorithm generates n massless momenta according to (1.69) using $3n-4$ random numbers $r_i \in [0,1]$. Note, that in the original version of RAMBO [97] the phase space was generated from $4n$ random numbers. Hence, the inversion of the phase space generation algorithm is only possible in the modified RAMBO on diet algorithm. In the following, we only present the final results and algorithms. The derivation and proofs are given in Ref. [96][3].

Algorithm 1: The massless RAMBO on diet algorithm.

$Q_1 \leftarrow Q$, $M_1 \leftarrow \sqrt{Q^2}$, $M_n \leftarrow 0$
for $i = 2,\ldots,n$ **do**
  if $i \neq n$ **then**
    solve $r_{i-1} = (n+1-i)u_i^{2(n-i)} - (n-i)u_i^{2(n+1-i)}$ for u_i
    $M_i \leftarrow u_2 \ldots u_i \sqrt{Q^2}$
  end if
  $\cos\theta_i \leftarrow 2r_{n-5+2i} - 1$, $\sin\theta_i \leftarrow \sqrt{1-\cos^2\theta_i}$, $\phi_i \leftarrow 2\pi r_{n-4+2i}$
  $q_i \leftarrow 4M_{i-1}\rho(M_{i-1}, M_i, 0)$
  $\mathbf{p}_{i-1} \leftarrow q_i(\cos\phi_i \sin\theta_1, \sin\phi_i \sin\theta_i, \cos\theta_i)$
  $p_{i-1} \leftarrow (q_i, -\mathbf{p}_{i-1})$, $Q_i \leftarrow (\sqrt{q_i^2 + M_i^2}, -\mathbf{p}_{i-1})$
  boost p_{i-1} and Q_i by $\mathbf{Q}_{i-1}/Q_{i-1}^0$
end for
$p_n \leftarrow Q_n$
return $\{p_1,\ldots,p_n\}$ with weight W_{Φ_n}

As the total phase space volume is given by

$$\begin{aligned} V_{\Phi_n} &= \int_{\Phi_n} \mathrm{d}\Phi_n(p_1,\ldots,p_n|Q) \\ &= \int_{U_n} \mathrm{d}^{3n-4}\mathbf{r} \left|\frac{\partial p}{\partial r}\right| = \left(\frac{\pi}{2}\right)^{n-1} \frac{(Q^2)^{n-2}}{(n-1)!\,(n-2)!}, \end{aligned} \tag{1.70}$$

the phase space weight W_{Φ_n} for the flat phase space is indeed constant

$$W_{\Phi_n} \equiv W_{\Phi_n}(Q) = \left|\frac{\partial p}{\partial r}\right| = \left(\frac{\pi}{2}\right)^{n-1} \frac{(Q^2)^{n-2}}{(n-1)!\,(n-2)!}. \tag{1.71}$$

[3] Note that there is an error in the original paper in the transformation from Eq. (8) to Eq. (9). Thus, the algorithm has some missing squares in the algorithm when solving for the variable u_i.

The mapping between $3n-4$ random numbers r_i and n 4-momenta is given in algorithm 1, and its inverse in algorithm 2.

Algorithm 2: The inverse massless RAMBO on diet algorithm.

$M_1 \leftarrow \sqrt{\left(\sum_{j=1}^{n} p_j\right)^2}$, $Q_n \leftarrow p_n$
for $i = n, \ldots, 2$ **do**
  $M_i \leftarrow \sqrt{\left(\sum_{j=i}^{n} p_j\right)^2}$
  if $i \neq n$ **then**
    $u_i \leftarrow M_i / M_{i-1}$
    $r_{i-1} \leftarrow (n+1-i)u_i^{2(n-i)} - (n-i)u_i^{2(n+1-i)}$
  end if
  $Q_{i-1} \leftarrow Q_i + p_{i-1}$
  boost p_{i-1} by $-\mathbf{Q}_{i-1}/Q^0_{i-1}$
  $r_{n-5+2i} \leftarrow (p^z_{i-1}/\left|\mathbf{p}_{i-1}\right| + 1)/2$, $\phi = \arctan(p^y_{i-1}/p^x_{i-1})$, $r_{n-4+2i} = \phi/2\pi + \Theta(-\phi)$
end for
return $\{r_1, \ldots, r_{3n-4}\}$

To solve the equation in line 4 of algorithm (1) we use Brent's method [98]. The function $\rho(M_{i-1}, M_i, m_{i-1})$ in line 8 corresponds to the two-body decay factor defined as

$$\rho(M_{i-1}, M_i, m_{i-1}) = \frac{1}{8M_{i-1}^2}\sqrt{(M_{i-1}^2 - (M_i + m_{i-1})^2)(M_{i-1}^2 - (M_i - m_{i-1})^2)},$$

which simplifies in the massless case to

$$\rho(M_{i-1}, M_i, 0) = \frac{M_{i-1}^2 - M_i^2}{8M_{i-1}^2}.$$

1.4.2 Importance sampling and change of variables

In principle a plain Monte Carlo integration is able to approximate even high-dimensional integrals. In practice, however, the variance σ^2 often becomes large and limits the reliability of the integral estimation. To reduce the error we have two options: We accumulate more statistics which inevitably requires more computational power, or we rewrite the integrand in such a way that the integral is unchanged and the variance is reduced. In other words, with an appropriate choice of $\rho(x)$, the variance σ^2 can be minimized. The idea is to modify the probability distribution function ρ such a way that points where f is large have high probability.

Consider a function g which is defined on Ω and is similar to f, in the sense that $g \approx C \cdot f$. In fact g can be defined in such a way that g is properly normalized and can be interpreted as a probability distribution function. Then g has the same meaning as ρ before. Hence, the integral of f over Ω can be written as,

$$I = \int_\Omega \mathrm{d}^d\mathbf{x}\, f(\mathbf{x}) = \int_\Omega \mathrm{d}^d\mathbf{x}\, g(\mathbf{x})\frac{f(\mathbf{x})}{g(\mathbf{x})} = \int_\Xi \mathrm{d}^d\mathbf{y} \left|\frac{\partial x}{\partial y}\right| g(\mathbf{x}) \left.\frac{f(\mathbf{x})}{g(\mathbf{x})}\right|_{\mathbf{x}\equiv\mathbf{x}(\mathbf{y})}, \tag{1.72}$$

with a function $\mathbf{y} : \Omega \to \Xi \subseteq \mathbb{R}^d$. Requiring

$$g(\mathbf{x})\left|\frac{\partial x}{\partial y}\right| = 1 \quad \Leftrightarrow \quad \left|\frac{\partial y}{\partial x}\right| = g(\mathbf{x}), \tag{1.73}$$

determines the mapping $\mathbf{y}$. This simplifies the integral

$$I = \int_\Xi \mathrm{d}^d\mathbf{y}\ \left.\frac{f(\mathbf{x})}{g(\mathbf{x})}\right|_{\mathbf{x}\equiv\mathbf{x}(\mathbf{y})} \tag{1.74}$$

and reduces the error, since the integrand is almost constant $C \approx f/g$. If a function $g(\mathbf{x})$ is known, the mapping $\mathbf{y}$ can be calculated and inverted to find $\mathbf{x} = \mathbf{x}(\mathbf{y})$, which is called the quantile function. In general, this is highly non-trivial. This method of choosing an adequate density is called importance sampling. There are several implementations available, one of the most frequently used is `Vegas` [29, 30] which assumes that $g(\mathbf{x})$ factorizes, *i.e.*

$$g(\mathbf{x}) = \prod_{k=1}^{d} g_k(x_k). \tag{1.75}$$

Example: Normal distribution

Let us consider a simple example: Suppose we are interested to integrate the following function over the domain $\Omega = \mathbb{R}$:

$$f(x) = \frac{1}{\sqrt{\pi}} \exp(-x^2). \tag{1.76}$$

Assuming we do not know the integral, a suitable mapping is the Cauchy distribution

$$g(x) = \frac{1}{\pi}\frac{1}{1+x^2}, \tag{1.77}$$

as the first terms of their Taylor expansion agree up to a factor which is irrelevant for importance sampling

$$f(x) = \frac{1}{\sqrt{\pi}}\left(1 - x^2 + \mathcal{O}(x^4)\right), \tag{1.78a}$$

$$g(x) = \frac{1}{\pi}\left(1 - x^2 + \mathcal{O}(x^4)\right). \tag{1.78b}$$

As we already know that $g(x)$ is the Cauchy distribution we also know its quantile function which is given by

$$x = \tan\left(\pi(y - 1/2)\right). \tag{1.79}$$

Using this, a better estimation of the integral can then be obtained by

$$I \approx \bar{I}_N = \frac{1}{N}\sum_{i=1}^{N} \frac{\frac{1}{\sqrt{\pi}}\exp\left(-\tan^2\left(\pi(y_i - 1/2)\right)\right)}{\frac{1}{\pi}\frac{1}{1+\tan^2(\pi(y_i-1/2))}}, \tag{1.80}$$

where $\{y_i\}_{i=1}^N$ are uniformly distributed numbers.

1.4.3 Adaptive multi-channel method

For importance sampling the knowledge of a function g which is similar to f is required. This function g also needs to be simple, so that a change of variables can be performed analytically. In general, it is not possible to find one function g that satisfies all these conditions. However, it is possible that a set of functions g_j approximate the integrand

well in some specific regions possessing all necessary properties. Then a combined probability distribution function can be defined as

$$g(\mathbf{x}) = \sum_{j=1}^{M} \alpha_j g_j(\mathbf{x}), \tag{1.81}$$

where the weights α_j and the individual g_j obey the additional conditions

$$\sum_{j=1}^{M} \alpha_j = 1, \tag{1.82a}$$

$$\int_\Omega \mathrm{d}^d\mathbf{x}\, g_j(\mathbf{x}) = 1. \tag{1.82b}$$

Hence, g is normalized as well. If the individual g_j are sufficiently small in the regions in which they approximate the integrand poorly, the combined function g approximates f well. Thus, the integral I can be rewritten as

$$I = \sum_{j=1}^{M} \alpha_j \int_\Omega \mathrm{d}^d\mathbf{x}\, g_j(\mathbf{x}) \frac{f(\mathbf{x})}{g(\mathbf{x})}. \tag{1.83}$$

Performing the change of variables $\mathbf{y}_j : \Omega \to \Xi_j$ for each g_j individually yields

$$I = \sum_{j=1}^{M} \alpha_j \int_{\Xi_j} \mathrm{d}^d\mathbf{y}_j \left.\frac{f(\mathbf{x})}{g(\mathbf{x})}\right|_{\mathbf{x}\equiv\mathbf{x}_j(\mathbf{y}_j)}, \tag{1.84}$$

where the coordinates $\mathbf{x}_j(\mathbf{y}_j)$ are set by the function g_j. To avoid explicit evaluation of the sum it is convenient to replace it by a Monte Carlo sum

$$I = \int \mathrm{d}j \int_{\Xi_j} \mathrm{d}^d\mathbf{x}_j \left.\frac{f(\mathbf{x})}{g(\mathbf{x})}\right|_{\mathbf{x}\equiv\mathbf{x}_j(\mathbf{y}_j)}, \quad j \sim \{\alpha_1, \alpha_2, \ldots, \alpha_M\}, \tag{1.85}$$

where the index j is randomly generated according to its weight α_j. The index j is called channel and the quantile functions $\mathbf{x}_j$ are the corresponding mappings. The weights α_j have to be set before the contributions of each channel are known. Therefore, a priori, the weights are set to $\alpha_j = \frac{1}{M}$ generating the channels uniformly. If, after a certain number of integrand evaluations, a specific channel j_1 gives large contributions to the integral its corresponding weight α_{j_1} will be enlarged [32], whereas the weight of channels with smaller contributions will be decreased. This method decreases the variance of the integral for a fixed number N of points $\{\mathbf{x}_i\}_{i=1}^N$, which is effectively importance sampling for the weights, called adaptive weight optimization. The multi-channel approach is also adopted by `Madgraph5` [23] and `Sherpa` [24] which are used in the projects presented in Chapter 3–5.

1.5 Event generation

In the previous sections we illustrated the relevance of appropriate sampling methods when perfoming Monte Carlo integrations. This sampling of random numbers and hence phase space points can also be interpreted as event simulation. These events naturally come with weights which correspond to the integrand. In experiment, however, we only obtain unweighted events. Consequently, if we want to compare our simulated events with the experimental measurements we first need to translate the weighted events into

unweighted events. Usually, this this unweighting of events is inefficient and computationally expensive.

In the following, we first take the step from the numerical integration towards weighted event generation. Afterwards, we emphasize the importance of unweighted events for actual LHC analyses in more detail. We will describe the most common unweighting method and explain its shortcomings. Finally, we define the unweighting efficiency as a quality metric for the simulation and sampling method.

1.5.1 Weighted events

We consider again the integrated cross section for a $2 \to n$ process being now parametrized as

$$\sigma = \int_\Omega \mathrm{d}^m \mathbf{x} \, \frac{\mathrm{d}\sigma}{\mathrm{d}\mathbf{x}}, \tag{1.86}$$

where $\mathrm{d}\sigma/\mathrm{d}\mathbf{x}$ is the differential cross section with respect to the observables x_i. We can now think of an event $\mathbf{x} = (x_1, \ldots, x_m)$ as a set of m observables which describe the scattering process we are interested in. If we want to encode all physics information including all correlations we need to have $m \geq 3n - 4$. When performing this integral numerically we draw N phase space points or events $\{\mathbf{x}_i\}_{i=1}^N$ and sum over their differential cross sections

$$\sigma \approx \frac{1}{N} \sum_{i=1}^{N} \frac{\mathrm{d}\sigma}{\mathrm{d}\mathbf{x}_i}. \tag{1.87}$$

The differential cross section or event weight

$$w_i := \frac{\mathrm{d}\sigma}{\mathrm{d}\mathbf{x}_i} \tag{1.88}$$

describes the probability that the event $\mathbf{x}_i$ occurs. Thus, the after sampling N phase space points $\{\mathbf{x}_i\}_{i=1}^N$ and evaluating their event weights $\{w_i\}_{i=1}^N$ we have effectively generated N weighted events as the combined sample $\{\mathbf{x}_i, w_i\}_{i=1}^N$.

1.5.2 Unweighted events

When we think of events in the sense of a measurement at the LHC we still use a set of observables to describe it. In nature, however, the events do not come with probabilities or event weights as this quantum mechanical information is lost during the measurement. You can however measure event rates and thus deduce the cross section statistically. For this, we usually fill histograms of certain kinematic observables to obtain their differential cross sections approximately.

When performing LHC analyses we would like to treat the measured and generated events in the same way to make their comparison as simple as possible. Therefore, we need to find a procedure to unweight the generated events. To be precise, we want to transform a set of N weighted events $\{\mathbf{x}_i, w_i\}_{i=1}^N$ into a set of M unweighted events $\{\mathbf{x}_i\}_{i=1}^M$, where $M \leq N$ ($N = M + l$) in general. The most common unweighting procedure is the hit-or-miss method which we describe in the following.

Hit-or-miss method

The basic idea of this method is to translate the weight w_i into a probability to keep or reject the event $\mathbf{x}_i$. For this we determine the maximum weight $w_{\max}$ and define the relative weight

$$w_{i,\text{rel}} = \frac{w_i}{w_{\max}}. \tag{1.89}$$

Then, we a draw random number $R_i \in [0,1]$ for every event $\mathbf{x}_i$ and keep it only if

$$w_{i,\text{rel}} > R_i \tag{1.90}$$

is satisfied, otherwise we reject it. An obvious shortcoming of this unweighting method is that we lose a lot of events in general.

Unweighting efficiency and improved event generation

As we have already mentioned, with the hit-or-miss method we will inevitably lose a lot of events. We can quantify the efficiency of the unweighting method with the unweighting efficiency ϵ_{uw} being defined as [41]

$$\epsilon_{\text{uw}} = \frac{\mathbb{E}[w]}{w_{\max}} \leq 1, \quad \text{with} \quad \mathbb{E}[w] = \frac{1}{N}\sum_{i=1}^{N} w_i \tag{1.91}$$

This implies that if the integrand and hence the weights are strongly varying, *i.e.* $\mathbb{E}[w] \ll w_{\max}$, we get very low unweighting efficiencies. In consequence, if we are interested to do proper LHC analyses we need to generate a lot of weighted events in the first place which is computationally expensive.

However, having more advanced sampling methods at hand, we can rewrite the original integral (1.86) and thus redefine the event weights in order to obtain a larger unweighting efficiency. For this, we shortly reinvestigate the role of importance sampling in the context of event generation. Using the same reparametrization trick as in (1.72) and with $f(\mathbf{x}) \equiv w(\mathbf{x}) = \mathrm{d}\sigma/\mathrm{d}\mathbf{x}$ the integrated cross section reads

$$\sigma = \int_\Xi \mathrm{d}^m\mathbf{y}\, \frac{w(\mathbf{x})}{g(\mathbf{x})}\bigg|_{\mathbf{x}\equiv\mathbf{x}(\mathbf{y})} \tag{1.92}$$

where we have introduced a density function $g \approx C \cdot w$ obeying the condition (1.73). Now, in event generation this translates into events $\mathbf{x}(\mathbf{y})_i$ with new weights

$$\tilde{w}_i = \frac{w_i}{g} \approx C = \text{const.} \tag{1.93}$$

As all weights $\tilde{w}_i$ are nearly constant, the unweighting procedure is much more effective

$$\epsilon_{\text{uw}} = \frac{\mathbb{E}[\tilde{w}_i]}{\tilde{w}_{\max}} \approx \frac{\mathbb{E}[C]}{C} = 1. \tag{1.94}$$

We can conclude that an appropriate phase space sampling is crucial not only for numerical integration but also for event generation. As we already pointed out by introducing importance sampling and the adaptive multi-channel method, a lot of effort has already been put into improving phase space sampling. However, typical unweighting efficiencies for state-of-the-art LHC analyses are gradually decreasing [35, 36]. Hence, modern machine learning approaches like boosted decision trees or neural networks [37–41] have been recently proposed to overcome current limitations. We will propose another method based on generative neural networks in Chapter 3–5 to diminish the shortcomings of standard Monte Carlo methods. The fundamentals of generative neural networks are therefore introduced in the following Chapter 2

Chapter 2

Neural Networks and Deep Learning

Deep learning or more specifically neural networks are a special kind of machine learning (ML). They are inspired by the human brain and are used as an automatic learning algorithm. Typical tasks for neural networks involve regression and classification problems or more involved challenges like image generation or density estimation. In recent years, the application of ML techniques received more and more attention among physicists. In LHC physics the simulation of events, parton showering or detector effects is most commonly performed with sophisticated Monte Carlo methods. However, these methods face many challenges, such as high-dimensional phase-space integrals, event unweighting, or complex and thus slow detector simulations. With the application of ML methods some of these bottlenecks might be circumvented or at least diminished.

In this chapter, we first give a short introduction into the basic concepts and building blocks of neural networks in Section 2.1. Afterwards, in Section 2.2 we explain the most common training algorithms and optimization techniques and explain the meaning of overfitting. Finally, in Section 2.3 and Section 2.4 we introduce generative adversarial networks (GANs) [42] and invertible neural networks (INNs) [43], respectively, as they are the most promising attempts to tackle the simulation and generation tasks mentioned above. Some displayed figures and parts of the text in Sec. 2.3 are identical to the content of Ref. [1].

2.1 An introduction to neural networks

Artificial neural networks (ANNs) are composed of neurons which are loosely modeled after the human brain. Each of these neurons represent a simple mathematical function and are connected to each other an entire network. These (deep) neural networks have the necessary capacity to learn the complex tasks we are interested in.

2.1.1 Structure of an artificial neuron

In the most simple setup, the neural network consists of exactly one neuron. In general, this neuron gets an arbitray number of inputs x_i and has a single output y. In order to get the output, the neuron performs the transformation

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{2.1}$$

where f is the activation function, and w_i and b denote the trainable weights and bias, respectively. The structure of an artificial neuron is depicted in Figure 2.1.
Having artificial neurons at hand, we can now construct deep neural networks which are capable of solving real world problems.

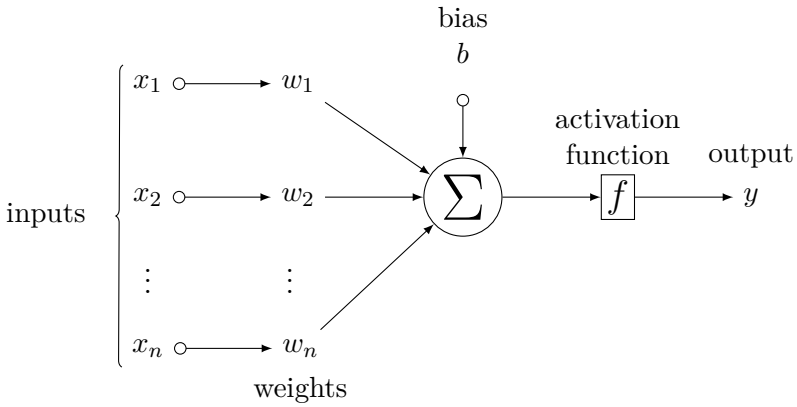

Figure 2.1: Structure of a neuron. The input is multiplied with weights w_i, summed up with an additional bias and an activation function is applied.

2.1.2 Deep neural networks

In general, a neural network consists of several neurons, which are typically organized in layers. The size of a layer is usually referred to as the width of the network. These layers can further be stacked one after another and represents the depth. Depending on the position in the network, we distinguish three kinds of layers: the input layer at the beginning, the output layer at the end and the hidden layers in between.

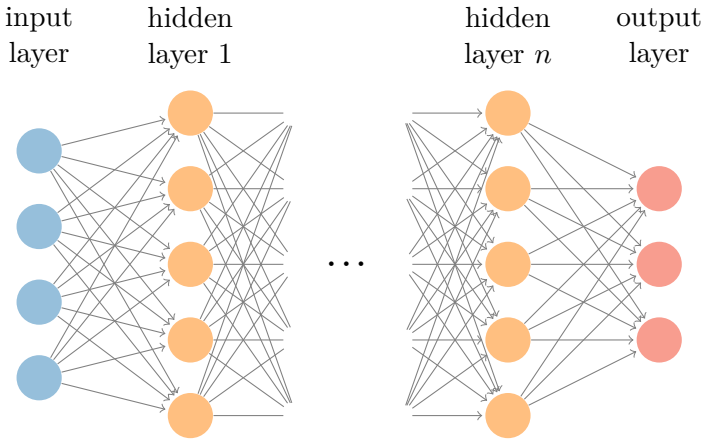

Figure 2.2: Generic structure of a neural network. Each circle illustrates one neuron. The blue layer on the left side is called the input layer, the orange layers are the hidden layers and the red layer is the output layer. The arrows indicate the connections between the neurons which are weighted by trainable weights w.

As depicted in Figure 2.2, we restrict ourselves to feedforward neural networks in which only connections to neurons in the next layer are possible. There are no cycles or loops in the network. These networks are generally easier to train than networks with loops, such as recurrant neural networks (RNNs), and are sufficient for to solve the problems we are facing in our applications [99].

At this stage we need to specify the role of the weights w_i, the bias b and the activation function f. The weights w_i and bias b are the heart of every neural network as they represent the tunable parameters being adjusted by the learning algorithm. Each

connection between neurons, depicted as arrows in Figure 2.2, is weighted by a weight and thus influences the importance of the inputs. On top of that, each neuron carries a bias b which allows for a shift of the summed input. Up to now, all transformations of the neural network are purely linear. Therefore, stacking layers and hence combining multiple linear transformations will result in another linear transformation. Then, the deep neural network is simply a linear transformation. In order to gain all benefits and performance neural networks are known for we need to apply a activation function which is non-linear. Only then, the neural network is also capable to learn non-linear and hence more complex structures

2.1.3 Activation functions

As we have already discussed, we need to have an activation function which is non-linear. There are many non-linear activation function which are suitable for neural networks [100]. In the following, we only describe those which are used in our applications in Chapter 3–5, as shown in Figure 2.3 together with their derivatives.

The sigmoid function

The sigmoid function or more specifically the logistic function is defined as

$$\text{sigmoid}(x) \equiv \sigma(x) := \frac{1}{1+e^{-x}}. \tag{2.2}$$

Usually, the sigmoid function is used in the output layer of neural networks, for instance representing probabilities in binary classification tasks. A nice feature of the sigmoid is its simple derivative

$$\sigma'(x) = \sigma(x)\,(1+\sigma(x)). \tag{2.3}$$

which is beneficial for calculating the gradients in the training algorithm. However, for large input values x the sigmoid is approximately flat and hence leads to small and vanishing gradients [100].

The rectified linear unit function

The ReLU (**Re**ctified **L**inear **U**nit) function is the most used activation function since its proposal in 2010 [101]. It is defined as

$$\text{ReLU}(x) := \max\{0, x\} = \begin{cases} x & x > 0, \\ 0 & x \leq 0. \end{cases} \tag{2.4}$$

This function is linear for positive inputs and zero for negative values. Therefore, the evaluation of both the function itself and its derivative is numerically efficient. A downside of this activation function is known as the dying neuron problem [102]. In order to overcome this issue the leaky ReLU function was suggested.

The leaky ReLU function

The leaky ReLU function is identical to the ReLU function for positive inputs but non-zero for negative values. In particular, a hyperparameter $\alpha \neq 0$ was introduced to solve the dying neuron problem such that no zero gradient occurs [103]:

$$\text{LeakyReLU}(x) := \begin{cases} x & x > 0, \\ \alpha x & x \leq 0. \end{cases} \tag{2.5}$$

where α is usually set to 0.3.

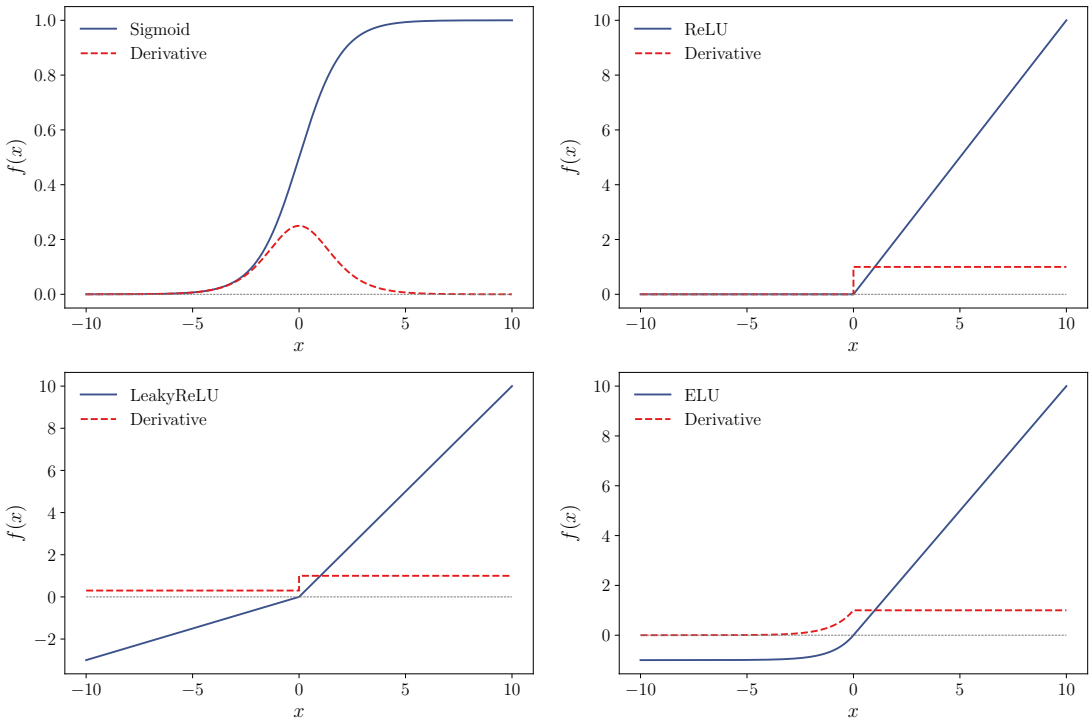

Figure 2.3: Plots of activation functions and their derivatives. On the upper left the ReLU activation function, on the upper right the LeakyReLU activation function with $\alpha = 0.3$, on the lower left the ELU activation function with $\alpha = 1$, and on the lower right the sigmoid activation function.

The exponential linear unit function

The ELU (**E**xponential **L**inear **U**nit) is also very similar to the ReLU activation function and was suggested in 2015 [104]. For negative value it has an exponential shape, resulting in a smoother activation function

$$\mathrm{ELU}(x) := \begin{cases} x & x > 0, \\ \alpha(e^x - 1) & x \leq 0. \end{cases} \tag{2.6}$$

where the hyperparameter α controls the saturation point for negative inputs and is usually set to one [100].

The softmax function

The softmax function [102] is given by

$$\mathrm{softmax}(x_i) := \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \tag{2.7}$$

where the layer this function is applied to has n neurons and x_i corresponds to the ith neuron output. The output of this function is in the range $[0, 1]$ and normalized such that

$$\sum_{i=1}^{n} \mathrm{softmax}(x_i) = 1. \tag{2.8}$$

Thus, it is used for creating the class probability output in the final layer of a model trained on a multi-class classification problem.

2.2 Tranining algorithms and loss functions

Up to now, we only introduced the basic ingredients to construct a neural network. However, we also need to have a recipe to adjust the trainable weights and hence train the network properly. For this we need two other component: a training goal and an algorithm that adjusts the weights such that this goal is achieved. In the following, we explain theses with the help of a simple example. Thereafter, we emphasize common problems in the training of neural networks and describe ways to avoid these problems.

2.2.1 Backpropagation

In order to train a neural network we need to define a training goal first. For this, we introduce a function that compares the network output with the training data in a quantitative manner. This function is called cost function or loss function. Usually, the goal of the training is such that this loss function is minimized, *i.e.* the smaller the loss value the better is the network performance. For instance, consider a regression task in which the network tries to match target values $\hat{y}_i$ for given inputs x_i. Let us further denote the composed function the network is representing as f_θ, where θ denotes the trainable network parameters, *i.e.* the weights w and the biases b. Thus, the network outputs are given by

$$y_i = f_\theta(x_i). \tag{2.9}$$

One possible loss function to tackle this regression task is the mean squared error (MSE)

$$L_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(f_\theta(x_i) - \hat{y}_i)^2. \tag{2.10}$$

As the input and target values are fixed by the data, the loss function is only a function of the neural network parameters $L = L(\theta)$. Therefore, the goal of the training algorithm is to find the parameter configuration θ^* that minimizes the loss function

$$\theta^* = \arg\min_\theta L(\theta). \tag{2.11}$$

If we neglect boundary effects on the parameter hypersurface[4], a necessary condition to find the global minimum of the loss function is

$$\nabla_\theta L(\theta) = 0. \tag{2.12}$$

In general, this problem can not be solved analytically and hence we need to use a numerical algorithm. The standard numerical method is stochastic gradient descent.

Stochastic gradient descent

In stochastic gradient descent (SGD) the parameters are updated after every single prediction according to

$$\theta_{t+1} = \theta_t - \eta\nabla_{\theta_t} L(\theta), \tag{2.13}$$

where η is the learning rate. In order to calculate the complete gradient of the loss function the gradients have to be propagated back through the network to the inputs [102]. As a consequence, every part of the neural network must be differentiable. Therefore, it is especially important to have activation functions which are not only differentiable but also simple and efficient to compute.

[4] In general, it is possible that the parameter configuration θ^* that minimizes the loss is not a local mimimum. Instead the parameter configuration lies on the edge of the parameter hypersurface. However it turns out, that most of the time this is not a problem [105].

2.2.2 Optimizer

Unfortunately, using the ordinary SGD algorithm often results in unstable training or even prevents convergence. In general, minimizing the loss function comes with several challenges. For instance, the loss function can have multiple local minima in the parameter hypersurface. However, the goal of the optimazation is to find a global minimum[5]. The ability of the training algorithm to find other minima is called exploration whereas the fast and precise descent into such a minimum is called exploitation. Typically, there exists a trade-off between exploration and exploitation, *i.e.* improving the first usually worsens the other one and vice-versa. A visualization of the exploration-exploitation dilemma is shown in Figure 2.4. Therefore, more advanced optimizers have to be used do not only calculate the parameter updates based on the gradient but also on other quantities [106].

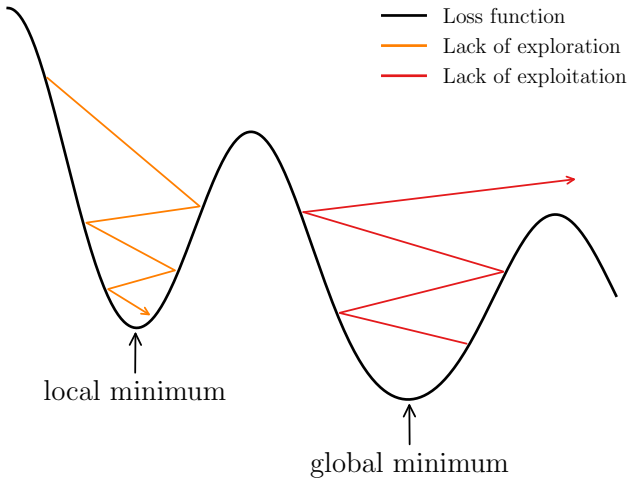

Figure 2.4: A visualization of the exploration-exploitation trade-off. The black line shows a generic loss function in parameter space. The orange line indicates that the optimizer has insufficient exploration capability and thus cannot find the global minimum. On the other hand, the red line shows an optimizer being unable to descend further into the global minimum, as it has a lack of exploitation potential.

Momentum

The implementation of a momentum m_t tries to smooth out the updates by adding a fraction β of the previous update to the next one [107]

$$\begin{aligned} m_{t+1} &= \beta m_t + \eta \nabla_{\theta_t} L, \\ \theta_{t+1} &= \theta_t - m_{t+1}. \end{aligned} \tag{2.14}$$

This additional term helps to avoid getting stuck in a local minimum. Further, neural networks without momentum tend to take a lot of sidesteps instead of going straight to the minimum. A drawback of momentum is that we might overshoot the minimum.

[5] Note that we did not write *the* global minimum as there generally exist several parameter configurations which lead to the same loss value [105].

RMSprop

The RMSprop optimizer is based on the idea that it is insufficient to have a global learning rate. Instead a separate learning rate for each parameter is required. Therefore, RMSProp keeps track of the moving average of the squared gradient updates. If a certain parameter is constantly updated with a small gradient, the root of the mean gradient squares (RMS) will be small as well. Thus, we divide by the RMS to adjust the gradient updates for each parameter

$$\begin{aligned} v_{t+1} &= \beta v_t + (1-\beta)(\nabla_{\theta_t} L)^2, \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_{t+1}} + \epsilon} \nabla_{\theta_t} L, \end{aligned} \tag{2.15}$$

where ϵ is just a regularization parameter.

Adam

The Adam optimizer combines momentum and RMSProp to **ada**ptively estimate the **m**oment of each gradient [108]

$$\begin{aligned} m_{t+1} &= \beta_1 m_t + (1-\beta_1)\nabla_{\theta_t} L, \\ v_{t+1} &= \beta_2 v_t + (1-\beta_2)(\nabla_{\theta_t} L)^2, \\ \hat{m}_{t+1} &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v}_{t+1} &= \frac{v_t}{1-\beta_2^t}, \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_{t+1}} + \epsilon} \hat{m}_{t+1}. \end{aligned} \tag{2.16}$$

The hyperparameters β_1 and β_2 are usually set to 0.9 and 0.999, respectively. Later in all our applications Adam is the optimizer of our choice.

Learning rate decay

On top of this, it is sometimes beneficial to also decrease the learning rate during the training. For instance, if we are close to the global minimum we want to have a smaller learning rate and thus smaller steps to have a higher accuracy. There are several ways to implement a decay. In all our projects we use the following learning rate schedule

$$\eta(k) = \frac{\eta_0}{1 + k\,\gamma}, \tag{2.17}$$

where k denotes the training epoch, η_0 the learning rate in the beginning, and γ is the decay hyperparameter.

2.2.3 Overfitting

A problem that can occur with neural networks is overfitting [109]. As we usually perform more parameter updates than we have training data available we inevitably do several training iterations on the same data points. Thus, we risk overspecialisation or overfitting which means that the network learns details which are specific to the training data, as shown in Figure 2.5. To be more precise, this means that the network is not just learning the underlying structures but rather memorizes every single data point and its corresponding target value. A standard method to check whether the neural network is

overfitting or not is by splitting the data into training data and test data. If the loss on the training data is significantly smaller than on the test data the network is overfitting. In general, the bigger the neural network the easier it can overfit.

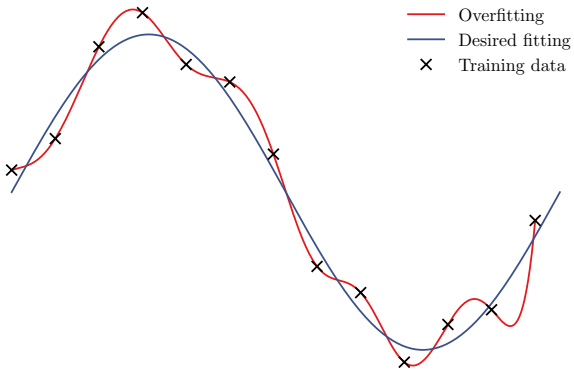

Figure 2.5: Overfitting of a neural network shown in a graphic example. The underlying structure is a sinus function

There exist various attempts to avoid overfitting when training neural networks. The most prominent methods are dropout [110], weight regularization [111, 112], or an early stopping of the training process [113].

2.3 Generative adversarial networks

When we are interested to employ neural networks for event generation we need to define a suitable network. One of the most promising architectures are generative adversarial networks (GANs) [42], which have already shown impressive results in tasks like the generation of images, videos or music. The defining structural elements of GANs are two competing neural networks, where a generator network G tries to mimic the data while a discriminator network D is trained to distinguish between generated and real data. These two networks play against each other until a Nash-equilibrium is reached in which neither the discriminator nor the generator can improve further.

As a side remark, another common type of neural networks used for generative problems are variational autoencoders (VAE) [114, 115]. While VAEs can be used to generate new data samples, a key component is the latent modeling and the marginalization of unnecessary variables, which is not a problem in generating LHC events. Further, as GANs are known to generate better samples than VAEs we only focus on GANs in the following.

2.3.1 Basic structure and loss functions

Let us denote the generator and discriminator weights as θ and φ, respectively. In general, the generator is a mapping $G : \mathcal{Z} \to \Phi$ from a latent space $\mathcal{Z}$ to the target or phase space Φ. Thus, the generator network induces a distribution

$$p_z(z) \xrightarrow{G_\theta} p_\theta(x) \equiv p_\theta(G_\theta(z)), \tag{2.18}$$

where $p_z(z)$ is the prior distribution the random numbers are drawn from. The discriminator network compares two distributions, the true data distribution $p_{\text{data}}(x)$ and the generated distribution $p_\theta(x)$. From each of the two distributions we provide batches of events $\{x_{\text{data}}\}$ and $\{x_G\}$ sampled from p_{data} or p_θ, respectively.

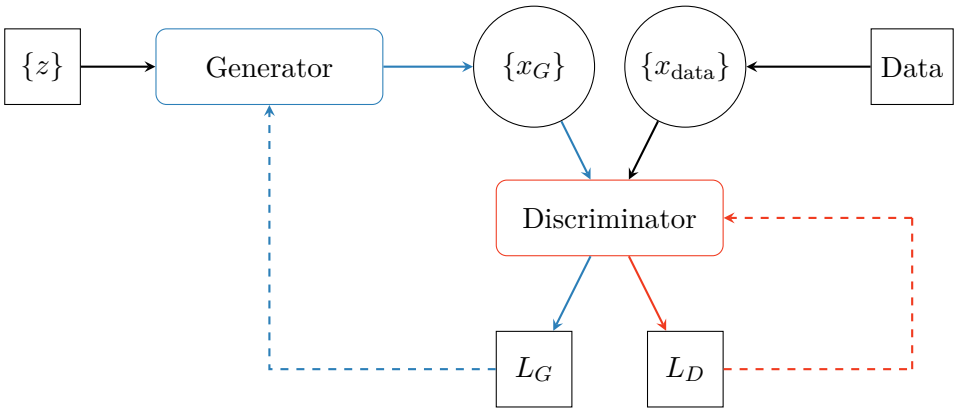

Figure 2.6: Schematic diagram of a GAN. The input $\{z\}$ describes a batch of random numbers and $\{x\}$ denotes a batch of events sampled either from p_θ or p_{data}. The blue (red) and arrows indicate which connections are used in the training of the generator (discriminator).

The discriminator $D(x) \in (0, 1)$ is trained to give $D = 1$ for each point in a true batch and $D = 0$ for the each point in the generated batch. The corresponding loss function is defined by

$$L(\theta, \phi) = \mathbb{E}_{x\sim p_{\text{data}}}[-\log D_\varphi(x)] + \mathbb{E}_{z\sim p_z}[-\log(1 - D_\varphi(G_\theta(z)))]. \tag{2.19}$$

The discriminator is trained to minimize this loss while the generator tries to maximize it. Our goal is to train the GAN such that it ends in a Nash-equilibrium. As the first term in the loss function is not dependent on the generator weights θ we can rephrase the task by writing it as two independent losses

$$\begin{aligned} L_D &= \mathbb{E}_{x\sim p_{\text{data}}}[-\log D_\varphi(x)] + \mathbb{E}_{z\sim p_z}[-\log(1 - D_\varphi(G_\theta(z)))], \\ L_G &= -\mathbb{E}_{z\sim p_z}[-\log(1 - D_\varphi(G_\theta(z))], \end{aligned} \tag{2.20}$$

where each need to be minimized by the corresponding network. Inspired by the original formulation of the loss function the associated GAN is usually denoted as minimax GAN. The evolution of the discriminator loss is illustrated in the left panel of Fig. 2.7. We can compute the discriminator loss in the limit where the generator has produced a perfect image of the true distribution. In this case the discriminator network will give $D = 0.5$ for each point x and the result becomes $L_D = -2 \log 0.5 \approx 1.4$.
In the orginal paper [42], they also proposed the non-saturating GAN, in which the generator loss function takes the form

$$L_G = -\mathbb{E}_{z\sim p_z}[\log(D_\varphi(G_\theta(z)))]. \tag{2.21}$$

It turns out that it is numerically more efficient to use the non-saturating GAN. In the right panel of Fig. 2.7 we see how this assignment leads to larger gradients away from the truth configuration.

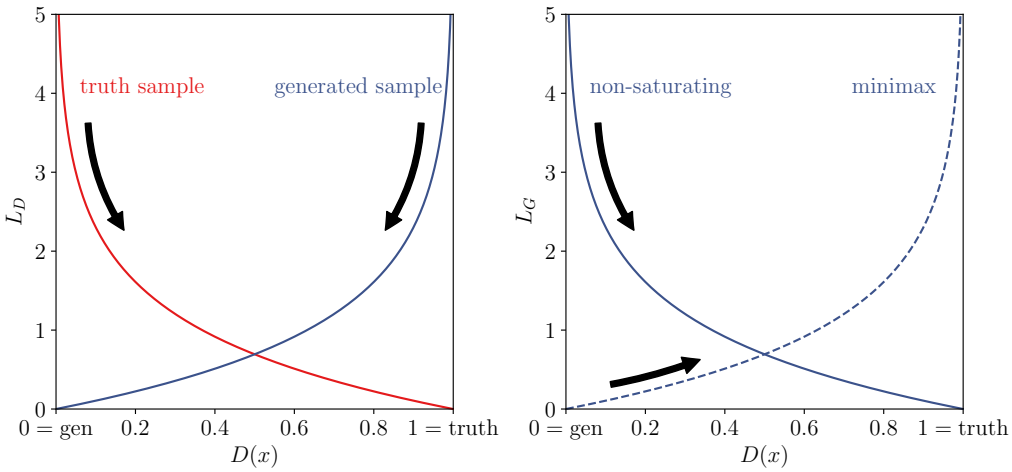

Figure 2.7: Discriminator and generator losses as a function of the value assigned by the discriminator. The red line indicates batches from the truth distribution, the blue lines batches from a generated distribution. The arrows indicate the direction of the training.

2.3.2 Training stability and regularization

The key to the GAN training is the alternating training of the generator and discriminator networks with their respective loss functions given in Eq.(2.20) and Eq.(2.21). Here, the balance between generator and discriminator is crucial. On the one hand, the generator can only be as good as the discriminator which defines the level of similarity between true and generated data. On the other hand, a perfect discriminator leads to a vanishing loss function, which reduces the gradient and slows down the training. This interplay of the two networks often leads to stability issues in the training [116].

Gradient penalty regularization

A common way to stabilize networks are noise-induced regularization methods, or equivalently including a penalty on the gradient for the discriminator variable $D(x)$ [117]. Specifically, we apply the gradient to the logit function

$$d(x) = \log\left(\frac{D(x)}{1-D(x)}\right) \qquad \Rightarrow \qquad \frac{\partial d}{\partial x} = \frac{1}{D(x)}\frac{1}{1-D(x)}\frac{\partial D}{\partial x} \tag{2.22}$$

enhancing its sensitivity in the regions $D \to 0$ or $D \to 1$. The penalty applies to regions where the discriminator loss leads to a wrong prediction, $D \approx 0$ for a true batch or $D \approx 1$ away from the truth. The gradient penalty term is defined by

$$L_{\text{GP}} = \mathbb{E}_{x\sim p_{\text{data}}}\left[(1-D(x))^2\,|\nabla d(x)|^2\right] + \mathbb{E}_{z\sim p_z}\left[D(G(z))^2\,|\nabla d(G(z))|^2\right], \tag{2.23}$$

where the pre-factors $(1-D)^2$ and D^2 ensure that for a properly trained discriminator this additional contribution vanishes. Thus, we add a term to the discriminator loss and obtain the regularized objective [117]:

$$L_D^{(\text{GP})} = L_D + \lambda_D L_{\text{GP}}, \tag{2.24}$$

with a properly chosen variable λ_D.

Spectral normalization

Another method of regularization can be obtained through spectral normalization (SN) of the weights [118]. In SN we want to replace every discriminator weight φ by

$$\varphi \to \varphi_{\text{SN}} = \frac{\varphi}{\zeta(\varphi)}, \tag{2.25}$$

where $\zeta(\varphi)$ is the spectral norm of the weight matrix φ defined by

$$\zeta(\varphi) := \max_{x \neq 0} \frac{||\varphi x||_2}{||x||_2} = \max_{||x||=1} ||\varphi x||_2 \,, \tag{2.26}$$

with layer in-and outputs x. If all weights φ are normalized according to (2.25) the discriminator is Lipschitz constraint [118] and bounded from above, *i.e.* $||D||_{\text{Lip}} \leq 1$. In general, the exact algorithm to calculate the spectral norm for each layer is computationally expensive. Therefore, we use the power iteration method to estimate $\zeta(\varphi)$ [119]. We start with random vectors u_0 for each weight φ. The power iteration update rule is then given by

$$v_{t+1} = \frac{\varphi^{\text{T}}\, u_t}{||\varphi^{\text{T}}\, u_t||}, \tag{2.27}$$

$$u_{t+1} = \frac{\varphi\, v_{t+1}}{||\varphi\, v_{t+1}||}. \tag{2.28}$$

Then the spectral norm can be approximated by

$$\zeta(\varphi) \approx u^{\text{T}} \varphi\, v. \tag{2.29}$$

When performing the weight updates with SGD or any other optimizer based on gradient descent it is sufficient to perform one round of power iteration to achieve a good performance [118].

Another method to avoid instabilities in the training of the GAN is to use the Wasserstein distance [120, 121] but our tests have shown that adding a gradient penalty term (2.24), or applying spectral normalization (2.25) works better in our case.

2.4 Invertible neural networks

Another generative network architecture we use in our projects is an invertible neural network (INN) [43–45]. They are specifically useful if we want to invert detector simulations as we will see in Chapter 5. INNs are an alternative class of generative networks, based on normalizing flows [122–125]. In particle physics such normalizing flow networks have proven useful for instance in phase-space generation [41], linking integration with generation [39, 40], or anomaly detection [126]. In order to understand the application of INNs we briefly introduce the basic concepts of INNs and explain its fundamental building block.

An invertible neural network learns a bijective mapping between the input and output space. This yields the possibility to evaluate the model in either direction and hence gain predictive capabilities in both directions. However, the usage of an INN is even benefical when we are not neccessarily interested in both mappings. For instance, if we simply use the architecture to define losses on both sides we can increase training stability and convergence speed [43]. In general, INNs are designed to achieve three objectives:

1. the mapping is bijective and thus invertible
2. the Jacobians for both directions are tractable
3. both directions can be evaluated efficiently. This third property goes beyond some other implementations of normalizing flows [122, 124].

2.4.1 Coupling blocks

The network architecture needs to be built such that it is easily invertible and satisfies the mentioned goals. This can be achieved with coupling blocks [44, 127]. Each of these coupling blocks is invertible and hence stacking multiple of these blocks results in an invertible model. We start by splitting the input vector u in two parts, u_1 and u_2. Using an element-wise multiplication $\odot$ and sum one could for instance define the output

$$v_1 = u_1 \odot s_1(u_2) + t_1(u_2) \qquad v_2 = u_2 \odot s_2(v_1) + t_2(v_1), \tag{2.30}$$

where s_1, s_2, t_1 and t_2 describe arbitrary transformations that are usually parametrized by neural networks. The inverse direction is then given by

$$u_2 = \frac{v_2 - t_2(v_1)}{s_2(v_1)} \qquad u_1 = \frac{v_1 - t_1(u_2)}{s_1(u_2)}. \tag{2.31}$$

For numerical reasons this gets modified to include an exponential,

$$v_1 = u_1 \odot e^{s_1(u_2)} + t_1(u_2), \qquad v_2 = u_2 \odot e^{s_2(v_1)} + t_2(v_1), \tag{2.32}$$

$$u_2 = (v_2 - t_2(v_1)) \odot e^{-s_2(v_1)}, \qquad u_1 = (v_1 - t_1(u_2)) \odot e^{-s_1(u_2)}. \tag{2.33}$$

By construction, this inversion works independent of the form of s_i and t_i. The structure of the coupling block is illustrated in Figure 2.8.

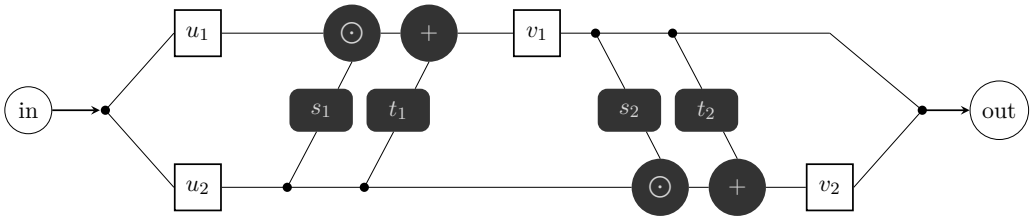

Figure 2.8: Structure of a coupling block. The input u is split in two parts u_1 and u_2. These are then transformed via Eq. (2.32) into v_1 and v_2 according to Eq. (2.32). In the end v_1 and v_2 are concatenated to form the complete output v.

With the coupling blocks we need to make sure that the Jacobian is tractable throughout the network. Following Eq. (2.32) we end up with an upper right triangular Jacobian J_1 after the first half of the forward pass [44]

$$J_1 = \begin{pmatrix} \mathrm{diag}(e^{s_1(u_2)}) & \frac{\partial v_1}{\partial u_2} \\ 0 & \mathbb{1} \end{pmatrix}. \tag{2.34}$$

Analogously, we obtain for the Jacobian J_2 of the second half of the coupling block

$$J_2 = \begin{pmatrix} \mathbb{1} & 0 \\ \frac{\partial v_2}{\partial v_1} & \mathrm{diag}(e^{s_2(v_1)}) \end{pmatrix}. \tag{2.35}$$

Hence, the Jacobian J of the entire coupling block is given by

$$J = J_2 \cdot J_1 = \begin{pmatrix} \operatorname{diag}(e^{s_1(u_2)}) & \frac{\partial v_1}{\partial u_2} \\ \operatorname{diag}(e^{s_1(u_2)}) \cdot \frac{\partial v_2}{\partial v_1} & \frac{\partial v_1}{\partial u_2} \cdot \frac{\partial v_2}{\partial v_1} + \operatorname{diag}(e^{s_2(v_1)}) \end{pmatrix} \tag{2.36}$$

Indeed, the Jacobian of the entire coupling block is not a triangular matrix. However, we are later only interested in the logarithmic Jacobian determinant which we can still calculate efficiently. Using the relation $\det(A \cdot B) = \det A \det B$ we obtain

$$\begin{aligned} \log(\det J) &= \log(\det J_1) + \log(\det J_2) \\ &= \log\left(\prod_{i=1}^{\dim u_2} e^{s_1(u_2)_i}\right) + \log\left(\prod_{i=1}^{\dim v_1} e^{s_2(v_1)_i}\right) \\ &= \sum_{i=1}^{\dim u_2} s_1(u_2)_i + \sum_{i=1}^{\dim v_1} s_2(v_1)_i. \end{aligned} \tag{2.37}$$

As we can use the same trick throughout multiple coupling blocks in the network and even in the inverse direction, the Jacobian of the entire network is tractable as required.

Chapter 3

Event Generation

The research presented in this Sec. 3.2 has been previously published in Ref. [1]. Most of the displayed figures and tables as well as the text are identical to the content of this article. Furthermore, the results shown and illustrated in Sec. 3.4 have also been presented in Ref. [128].

3.1 Introduction

First-principle simulations are a key ingredient to the ongoing success of the LHC, and they are crucial for further developing it into a precision experiment testing the structure of the Standard Model and its quantum field theory underpinnings. Such simulations of the hard scattering process, QCD activity, hadronization, and detector effects are universally based on Monte Carlo methods. Some of the shortcomings that come with these methods might be alleviated when we add a new direction, like machine learning techniques, to our tool box. While we should not expect them to magically solve all problems, we have seen that modern machine learning can trigger significant progress in LHC physics. The reason for our optimism related to event generation are generative adversarial networks or GANs [42], which have shown impressive performance in tasks like the generation of images, videos or music.

From the experimental side the detector simulation is the most time-consuming aspect of LHC simulations, and promising attempts exist for describing the behavior of the calorimeter with the help of generative networks [46–48, 50, 51, 129]. On the theory side, we know that the parton shower can be described by a neural network [52–55]. It has been shown that neural networks can help with phase space integration [37, 38] and with LHC event simulations [57–59]. One open question is why the GAN setup of Ref. [57] does not properly work and is replaced by a variational autoencoder with a density information buffer. Another challenge is how to replace the ad-hoc Z-constraint in the loss function of Ref. [58] by a generalizable approach to on-shell resonances. This problem of intermediate resonances is altogether avoided in Ref. [59]. It remains to be shown how GANs can actually describe realistic multi-particle matrix elements over a high-dimensional phase space in a flexible and generalizable manner.

Given this new piece of the event simulation puzzle through fast neural networks it should in principle be possible to add parton showers, possibly including hadronization, and detector effects to a full machine learning description of LHC events. Including higher-order corrections is possible and should lead to ever higher gains in computing time, assuming higher-orders are included in the training data. The interesting question then becomes where established methods might benefit from the fast and efficient machine learning input. Alternatively, we can replace the Monte Carlo event input and instead

generate reconstructed LHC events and use them to enhance analyses or to study features of the hard process. Indeed, the GAN approach also allows us to combine information from actual data with first-principles simulations in a completely flexible manner.

This chapter consists of three parts and we show how we can efficiently GAN[6] the simulation of three different LHC processes. In Section 3.2 we start by reviewing some of the features of phase space sampling with standard Monte Carlo methods and introducing GANs serving the same purpose. We then add the MMD and describe how its been used to describe intermediate resonances. We then apply the combined GAN-MMD network to the $2 \to 6$ particle production process

$$pp \to t\bar{t} \to (bq_1\bar{q}_1') \, (\bar{b}\bar{q}_2 q_2') \tag{3.1}$$

describing all intermediate on-shell states with Breit-Wigner propagators and typical width-to-mass ratios of few per-cent. We will focus on a reliable coverage of the full phase space, from simple momentum distributions to resonance peaks, strongly suppressed tails, and phase space boundaries.
Afterwards in Section 3.3 we impose full 4-momentum conservation and on-shell conditions for the external particles in a general way, by reducing the number of generated features to the degrees of freedom. As an example we consider the $W + 2$ jet production process

$$pp \to W^+ jj \tag{3.2}$$

and show how these modifications lead to a significant gain in precision.
Finally, in Section 3.4 we modify the discriminator loss function such that it can handle weighted events in the training data while the generator is still producing unweighted events. For this, we first explain these modifications using simple toy models and benchmark the GAN performance against th standard `Vegas` [29, 30] algorithm. Finally, we apply our unweighting GAN to weighted events stemming from Drell–Yan scattering process

$$pp \to \mu^- \mu^+. \tag{3.3}$$

3.2 Vanilla event generation

As a first benchmark example we consider top pair production with an intermediate decay of two W-bosons

$$pp \to t\bar{t} \to (bW^-) \, (\bar{b}W^+) \to (bq_1\bar{q}_1') \, (\bar{b}\bar{q}_2 q_2'). \tag{3.4}$$

illustrated in Fig. 3.1. If we assume that the masses of all final-state particles are known, as this can be extracted from the measurement, this leaves us with 18 degrees of freedom, which energy-momentum conservation reduces to a 14-dimensional phase space. In addition, our LHC simulation has to account for the 2-dimensional integration over the parton momentum fractions.
In this section we will briefly review how standard methods describe such a phase space, including the sharp features of the intermediate on-shell top quarks and W-boson. The relevant area in phase space is determined by the small physical particle widths and extends through a linearly dropping Breit-Wigner distribution, where it eventually needs to include off-shell effects. We will then show how a generative adversarial network can be constructed such that it can efficiently handle these features as well.

[6] From 'to GAN', in close analogy to the verbs taylor, google, and sommerfeld.

3.2.1 Standard Monte Carlo

For the hard partonic process we denote the incoming parton momenta as $p_{a,b}$ and the outgoing fermion momenta as p_i. Using the general definition in Eq. (1.58), we can parametrize the partonic cross section and the 14-dimensional phase space integration for six external particles as

$$\begin{aligned} \sigma &= \int \mathrm{d}\Phi_6 \, \frac{|\mathcal{M}(p_a, p_b; p_1, \ldots, p_6)|^2}{2\hat{s}}, \\ \text{with} \quad \mathrm{d}\Phi_6 &= \prod_{i=1}^{6} \left[\frac{\mathrm{d}^3 \mathbf{p}_i}{(2\pi)^3 2E_i} \right] (2\pi)^4 \delta^{(4)} \left(\sum_{i=1}^{6} p_i - Q \right). \end{aligned} \tag{3.5}$$

where $Q = p_a + p_b$ is the total momentum of the incoming partons. To cope with the high dimensionality of the integral we adopt advanced Monte Carlo techniques such as importance sampling which has been described in Section 1.4. There are several implementations available, one of the most frequently used is `Vegas` [29, 30].

A major challenge in particle physics applications is that multi-particle amplitudes in the presence of kinematic cuts typically have dramatic features. Our phase space sampling not only has to identify the regions of phase space with the leading contribution to the integral, but also map its features with high precision. For instance, the process illustrated in Fig. 3.1 includes narrow intermediate on-shell particles. Around a mass peak with $\Gamma \ll m$ they lead to a sharp Breit-Wigner shape of the transition amplitude. A standard way of improving the integration is to identify the invariant mass variable s where the resonance occurs and switch variables to

$$\int \mathrm{d}s \frac{F(s)}{(s-m^2)^2 + m^2\Gamma^2} = \frac{1}{m\Gamma} \int \mathrm{d}z \; F(s) \quad \text{with} \quad z = \arctan \frac{s-m^2}{m\Gamma} \, . \tag{3.6}$$

This example illustrates how phase space mappings, given some knowledge of the structure of the integrand, allow us to evaluate high-multiplicity scattering processes.

Finally, in LHC applications we are typically not interested in an integral like the one shown in Eq. (3.5). Instead, we want to simulate phase space configurations or events with a probability distribution corresponding to a given hard process, shower configuration, or detector smearing. This means we have to transfer the information included in the weights at a given phase space point to a phase space density of events with uniform weight. The corresponding unweighting procedure computes the ratio of a given event weight to the maximum event weights, probes this ratio with a random number, and in turn decides if a phase space point or event remains in the sample, now with weight one. This procedure is highly inefficient.

Summarizing, the challenge for a machine learning approach to phase space sampling is: mimic importance sampling, guarantee a precise mapping of narrow patterns, and avoid the limited unweighting efficiency.

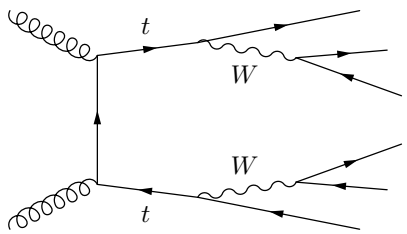

Figure 3.1: Sample Feynman diagram contributing to top-pair production, with intermediate on-shell particles labelled.

3.2.2 Network structure and resonances

Following Section 2.3, we denote the generator and discriminator weights as θ and φ, respectively. The generator induces a distribution $p_\theta(x)$ (2.18) of an event or phase space configuration x, typically organized with the same dimensionality as the (phase) space we want to generate. In order to train the GAN, we provide it with batches of phase space configurations $\{x_T\}$ and $\{x_G\}$, sampled from the true distribution p_T or p_θ, respectively. We can then parametrize the loss functions as

$$L_D = \mathbb{E}_{x\sim p_T}[-\log D_\varphi(x)] + \mathbb{E}_{x\sim p_\theta}[-\log(1-D_\varphi(x))], \tag{3.7}$$

$$L_G = \mathbb{E}_{x\sim p_\theta}\left[-\log D_\varphi(x)\right]. \tag{3.8}$$

As introduced in Eq. 2.23, we add a gradient penalty term [117] to the discriminator to avoid instabilities in the training [117]:

$$L_D^{(\text{GP})} = L_D + \lambda_D\, \mathbb{E}_{x\sim p_T}\left[(1-D_\varphi(x))^2\, |\nabla d_\varphi(x)|^2\right] \tag{3.9}$$

$$+ \lambda_D\, \mathbb{E}_{x\sim p_\theta}\left[D_\varphi(x)^2\, |\nabla d_\varphi(x)|^2\right], \tag{3.10}$$

where we have used

$$d_\varphi(x) = \log\left(\frac{D_\varphi(x)}{1-D_\varphi(x)}\right). \tag{3.11}$$

Maximum mean discrepancy

A particular challenge for our phase space GAN will be the reconstruction of the W and top masses from the final-state momenta. For instance, for the top mass the discriminator and generator have to probe a 9-dimensional part of the phase space, where each direction covers several 100 GeV to reproduce a top mass peak with a width of $\Gamma_t = 1.5$ GeV. Following the discussion of the Monte Carlo methods in Section 3.2.1 the question is how we can build an analogue to the phase space mappings for Monte Carlos. Assuming that we know which external momenta can form a resonance we explicitly construct the corresponding invariant masses and give them to the neural network to streamline the comparison between true and generated data. We emphasize that this is significantly less information than we use in Eq. (3.6), because the network still has to learn the intermediate particle mass, width, and shape of the resonance curve.

A suitable tool to focus on a low-dimensional part of the full phase space is the maximum mean discrepancy (MMD) [61]. The MMD is a kernel-based method to compare two samples drawn from different distributions. Consider two batches of points $\{x\}$ and $\{y\}$ drawn from a distribution p and q, respectively. The MMD then computes a distance between the distributions

$$\text{MMD}^2(p,q) = = \mathbb{E}_{x,x'\sim p}[k(x,x')] - 2\mathbb{E}_{x\sim p, y\sim q}[k(x,y)] + \mathbb{E}_{y,y'\sim q}[k(y,y')]. \tag{3.12}$$

where $k(x,y)$ can be any positive definite kernel function. Obviously, two identical distributions lead to $\text{MMD}(p,p) = 0$ in the limit of high statistics. Inversely, if $\text{MMD}(p,q) = 0$ for randomly sampled batches the two distributions have to be identical $p = q$. The shape of the kernels determines how local the comparison between the two distributions is evaluated. Two examples are Gaussian or Breit-Wigner kernels

$$k_{\text{Gauss}}(x,y) = \exp\left(-\frac{||x-y||_2^2}{2\sigma^2}\right), \qquad k_{\text{BW}}(x,y) = \frac{\sigma^2}{||x-y||_2^2 + \sigma^2}, \tag{3.13}$$

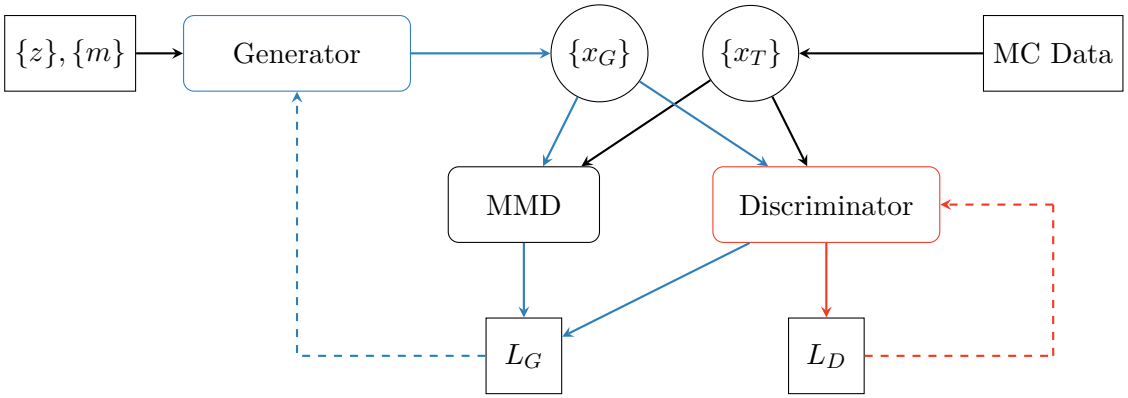

Figure 3.2: Schematic diagram of our GAN. The input $\{z\}$ and $\{m\}$ describe a batch of random numbers and the masses of the external particles, and $\{x\}$ denotes a batch of phase space points sampled either from the generator or the true data. The blue (red) lines and arrows indicate which connections are used in the training of the generator (discriminator).

where the hyperparameter σ determines the resolution. For an optimal performance it should be of the same order of magnitude as the width of the feature we are trying to learn. If the resonance and the kernel width become too narrow, we can improve convergence by including several kernels with increasing widths to the loss function. The shape of the kernel has nothing to do with the shape of the distributions we are comparing. Instead, the choice between the exponentially suppressed Gaussian and the quadratically suppressed Breit-Wigner determines how well the MMD accounts for the tails around the main feature. As a machine learning version of phase space mapping we add this MMD to the generator loss

$$L_G^{(\text{MMD})} = L_G + \lambda_G \, \text{MMD}^2, \tag{3.14}$$

with another properly chosen variable λ_G. Similar efforts in using the MMD to generate events have already been done in [130–132] and have also been extended to a adversarial MMD version or MMD-GAN [133–135], in which the MMD kernel is learned by another network.

In Fig. 3.2 we show the whole setup of our network. It works on batches of simulated parton-level events, or unweighted event configurations $\{x\}$. The input for the generator are batches of random numbers $\{z\}$ and the masses $\{m\}$ of the final state particles. Because of the random input a properly trained GAN will generate statistically independent events reflecting the learned patterns of the training data. For both the generator and the discriminator we use a 10-layer MLP with 512 units each combined with a leaky ReLU activation function in the hidden layers. The remaining network parameters are given in Tab. 3.1. The main structural feature of the competing networks is that the output of the discriminator, D, is computed from the combination of true and generated events and is needed by the generator network. The generator network combines the information from the discriminator and the MMD in its loss function, Eq. (3.14). The learning is done when the distribution of generated unweighted events $\{x_G\}$ and true Monte-Carlo events $\{x_T\}$ are essentially identical. We again emphasize that this construction does not (yet) involve weighted events. For the implementation of the GAN we have used `Keras` (v2.2.4) [136] with a `TensorFlow` (v1.14) backend [137].

3.2.3 Top-pair production

A sample Feynman diagram for our benchmark process

$$pp \to t\bar{t} \to (bq\bar{q}')\,(\bar{b}\bar{q}q') \tag{3.15}$$

is shown in Fig. 3.1. For our analysis we generate 1 million samples of the full $2 \to 6$ events as training data sample with `Madgraph5` [23] at a CM energy of 13 TeV. The intermediate tops and W-bosons allow us to reduce the number of Feynman diagrams by neglecting proper off-shell contributions and only including the approximate Breit-Wigner propagators. Our results can be directly extended to a proper off-shell description [138–140], which only changes the details of the subtle balance in probing small but sharp on-shell contributions and wide but flat off-shell contributions. Similarly, we do not employ any detector simulation, because this would just wash out the intermediate resonances and diminish our achievement unnecessarily.

Because we do not explicitly exploit momentum conservation our final state momenta are described by 24 degrees of freedom. Assuming full momentum conservation would for instance make it harder to include approximate detector effects. These 24 degrees of freedom can be reduced to 18 when we require the final-state particles to be on-shell. While it might be possible for a network to learn the on-shell conditions for external particles, we have found that learning constants like external masses is problematic for the GAN setup. Instead, we use on-shell relations for all final-state momenta in the generator network.

Combining the GAN with the MMD loss function of Eq. (3.14) requires us to organize the generator input in terms of momenta of final-state particles. With the help of a second input to the generator, namely a 6-dimensional vector of constant final-state masses, we enhance the 18-dimensional input to six 4-vectors. This way we describe all final-state particles, denoted as $\{x_G\}$ in Fig. 3.2, through an array

$$x = \{p_1, p_2, p_3, p_4, p_5, p_6\}\,, \tag{3.16}$$

where we fix the order of the particles within the events. This format corresponds to the generated unweighted truth events $\{x_T\}$ from standard LHC event simulators. In particular, we choose the momenta such that

$$p_{W^-} = p_1 + p_2\,, \quad p_{W^+} = p_4 + p_5\,, \quad p_{\bar{t}} = p_1 + p_2 + p_3\,, \quad p_t = p_4 + p_5 + p_6\,. \tag{3.17}$$

For the on-shell states we extract the resonances from the full phase space and use those to calculate the MMD between the true and the generated mass distributions.

Parameter	Value	Parameter	Value
Input dimension G	$18 + 6$	Input dimension D	24
Layers	10	Batch size	1024
Units per layer	512	Epochs	1000
Trainable weights G	2.3M	Iterations per epoch	1000
Trainable weights D	2.3M	Number of training events	1×10^6
λ_G	1		
λ_D	10^{-3}		

Table 3.1: Details for our GAN setup.

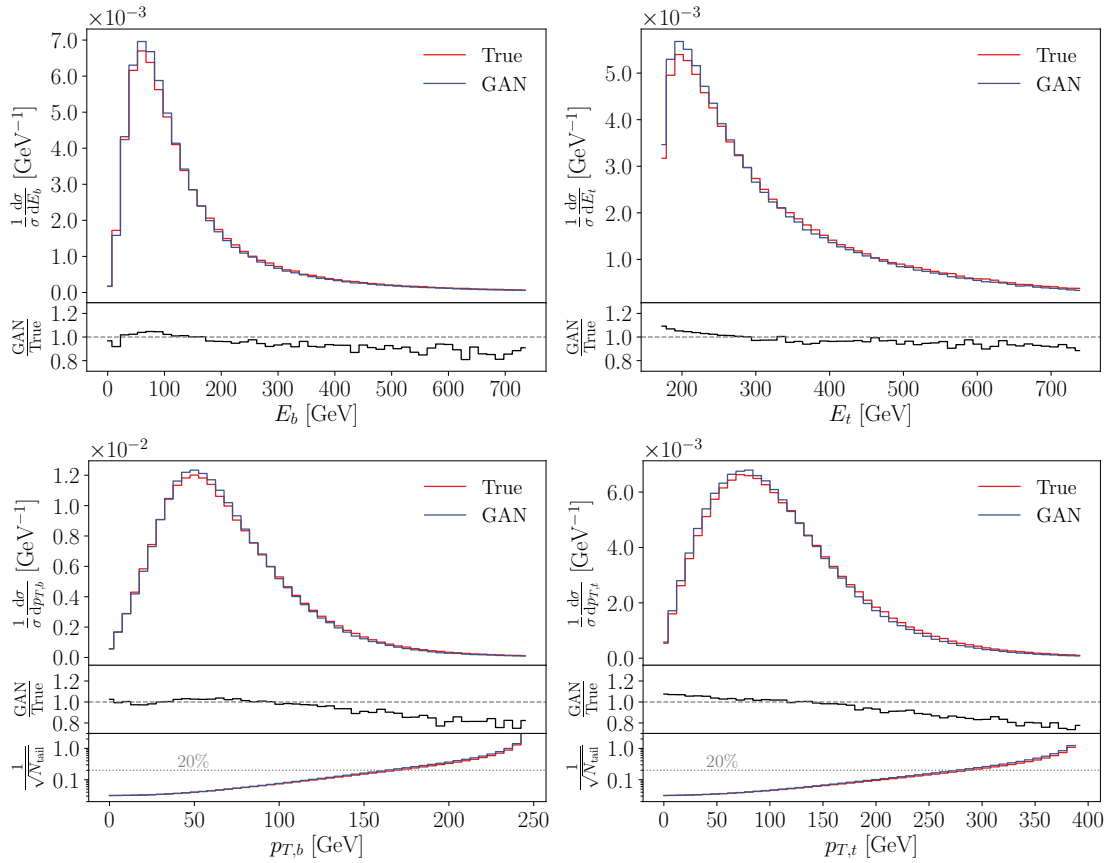

Figure 3.3: Energy (top) and transverse momentum (bottom) distributions of the final-state b-quark (left) and the decaying top quark (right) for MC truth (blue) and the GAN (red). The lower panels give the bin-wise ratio of MC truth to GAN distribution. For the p_T distributions we show the relative statistic uncertainty on the cumulative number of events in the tail of the distribution for our training batch size.

This additional loss is crucial to enhance the sensitivity in certain phase space regions allowing the GAN to learn even sharp feature structures.

Flat distributions

To begin with, relatively flat distributions like energies, transverse momenta, or angular correlations should not be hard to GAN [57–59]. As examples, we show transverse momentum and energy distributions of the final-state b-quarks and the intermediate top quarks in Fig. 3.3. The GAN reproduces the true distributions nicely even for the top quark, where the generator needs to correlate the four-vectors of three final-state particles.

To better judge the quality of the generator output, we show the ratio of the true and generated distributions in the lower panels of each plot, for instance $E_b^{(G)}/E_b^{(T)}$ where $E_b^{(G,T)}$ is computed from the generated and true events, respectively. The bin-wise difference of the two distributions increases to around 20% only in the high-p_T range where the GAN suffers from low statistics in the training sample. To understand this effect we also quantify the impact of the training statistics per batch for the two p_T-distributions. In the set of third panels we show the relative statistical uncertainty on

the number of events $N_{\text{tail}}(p_T)$ per batch b in the tail above the quoted p_T value

$$N_{\text{tail}}(p_T) = b\,(1 - N_{\text{cum}}(p_T)) = b\left(1 - \frac{1}{\sigma}\int_0^{p_T} \mathrm{d}p_T' \,\frac{\mathrm{d}\sigma}{\mathrm{d}p_T'}\right). \tag{3.18}$$

The relative statistical uncertainty on this number of events is generally given by $1/\sqrt{N_{\text{tail}}}$. For the $p_{T,b}$-distribution the GAN starts deviating at the 10% level around 150 GeV. Above this value we expect around 25 events per batch, leading to a relative statistical uncertainty of 20%. The top kinematics is harder to reconstruct, leading to a stronger impact from low statistics. Indeed, we find that the generated distribution deviates by 10% around $p_{T,t} \gtrsim 250$ GeV where the relative statistic uncertainty reaches 15%.

We emphasize that this limitation through training statistics is expected and can be easily corrected for instance by slicing the parameter in p_T and train the different phase space regions separately. Alternatively, we can train the GAN on events with a simple re-weighting, for example in p_T, but at the expense of requiring a final unweighting step.

Phase space coverage

To illustrate that the GAN populates the full phase space we can for instance look at the azimuthal coordinates of two final-state jets in Fig. 3.4. Indeed, the generated events follow the expected flat distribution and correctly match the true events.

Furthermore, we can use these angular correlations to illustrate how the GAN interpolates and generates events beyond the statistics of the training data. In Fig. 3.5 we show the 2-dimensional correlation between the azimuthal jet angles ϕ_{j_1} and ϕ_{j_2}. The upper-left panel includes 1 million training events, while the following three panels show an increasing number of GANed events, starting from 1 million events up to 50 million events. As expected, the GAN generates statistically independent events beyond the sample size of the training data and of course covers the entire phase space.

Resonance poles

From Ref. [38] we know that exactly mapping on-shell poles and tails of distributions is a challenge even for simple decay processes. Similar problems can be expected to arise for phase space boundaries, when they are not directly encoded as boundaries of the random number input to the generator. Specifically for our $t\bar{t}$ process, Ref. [57] finds that their GAN setup does not reproduce the phase space structure. The crucial task of this section is to show how well our network reproduces the resonance structures of the intermediate narrow resonances. In Fig. 3.6 we show the effect of the additional MMD loss on learning the invariant mass distributions of the intermediate W and top states. Without the MMD, the GAN barely learns the correct mass value, in complete agreement with the findings of Ref. [58]. Adding the MMD loss with default kernel widths of the Standard Model decay widths drastically improves the results, and the mass distribution almost perfectly matches the true distribution in the W-case. For the top mass the results are slightly worse, because its invariant mass needs to be reconstructed from three external particles and thus requires the generator to correlate more variables. This gets particularly tricky in our scenario, where the W-peak reconstruction directly affects the top peak. We can further improve the results by choosing a bigger batch size as this naturally enhances the power of the MMD loss. However, bigger batch sizes leads to longer training times and bigger memory consumption. In order to keep the training time on a responsible level, we limited our batch size to 1024 events per batch.

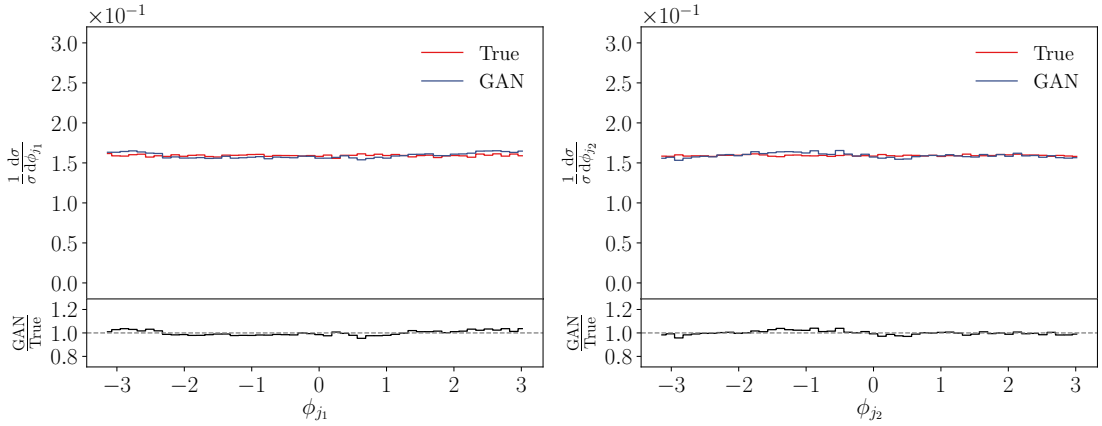

Figure 3.4: ϕ distributions of j_1 and j_2. The lower panels give the bin-wise ratio of MC truth to GAN distribution.

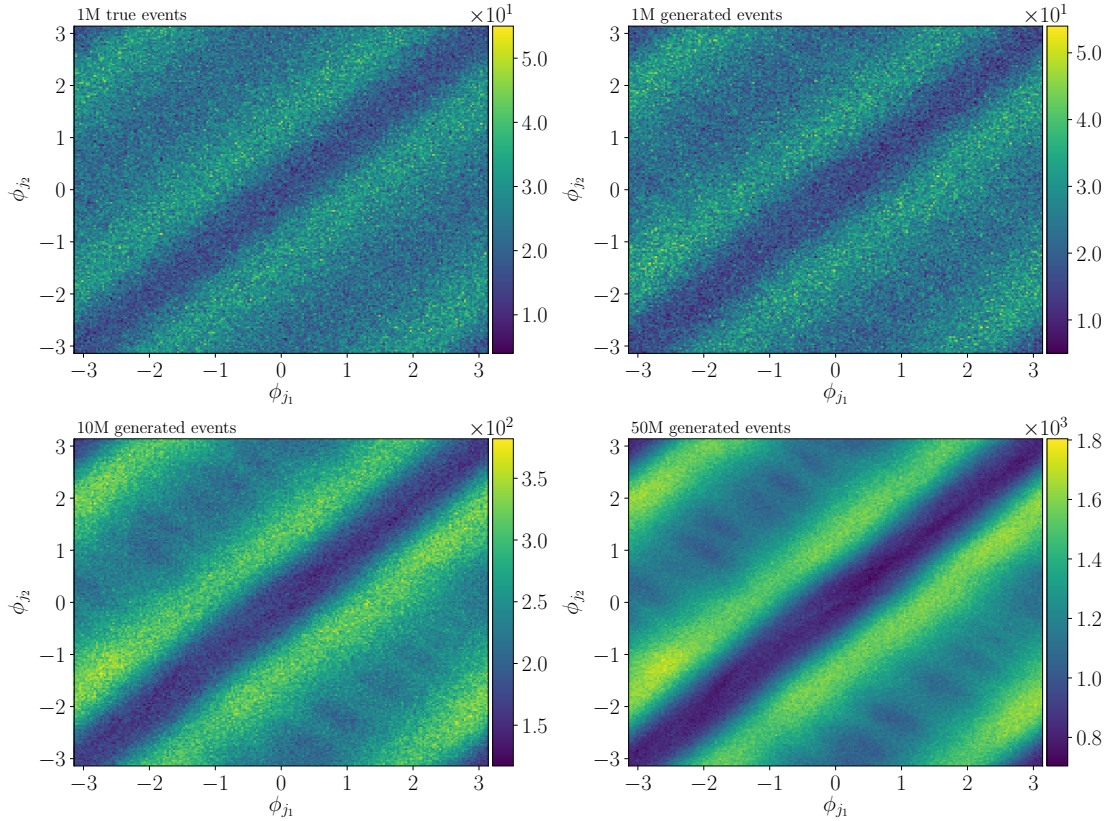

Figure 3.5: Correlation between ϕ_{j_1} and ϕ_{j_2} for 1 million true events (upper left) and for 1 million, 10 million, and 50 million GAN events.

As already pointed out, the results are not perfect in this scenario, especially for the top invariant mass, however, we can clearly see the advantages of adding the MMD loss.

To check the sensitivity of the kernel width on the results, we vary it by factors of $\{1/4, 4\}$. As can be seen in the lower panels of both distributions, increasing the resolution of the kernel or decreasing the kernel width hardly affects the network performance. On the other hand, increasing the width decreases the resolution and leads to too broad

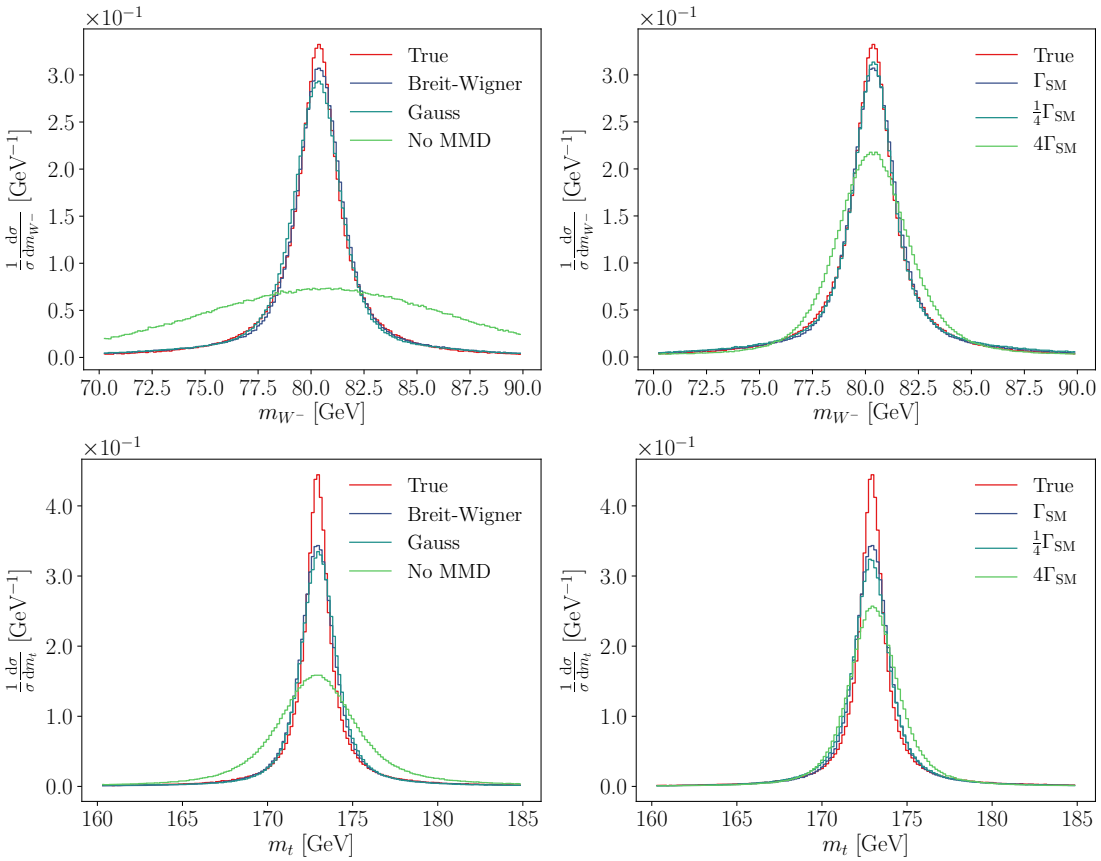

Figure 3.6: Comparison of different kernel functions (left) and varying widths (right) and their impact on the invariant mass of W boson (top) and top quark (bottom).

mass peaks. Similarly, if we switch from the default Breit-Wigner kernel to a Gaussian kernel with the same width we find identical results. This means that the only thing we need to ensure is that the kernel can resolve the widths of the analyzed features.

We emphasize again that we do not give the GAN the masses or even widths of the intermediate particles. This is different from Ref. [58], which tackles a similar problem for the $Z \to \ell\ell$ resonance structure and uses an explicit mass-related term in the loss function. We only specify the two final-state momenta for which the invariant mass can lead to a sharp phase space structure like a mass peak, define a kernel like those given in Eq. (3.13) with sufficient resolution and let the GAN do the rest. This approach is even more hands-off than typical phase space mappings employed by standard Monte Carlos.

Correlations

Now that we can individually GAN all relevant phase space structures in top pair production, it remains to be shown that the network also covers all correlations. A simple test is 4-momentum conservation, which is not guaranteed by the network setup. In Fig. 3.7, we show the sums of the transverse components of the final-state particle momenta divided by the sum of their absolute values. As we can see, momentum conservation at the GAN level is satisfied at the order of 2%.

Finally, in Fig. 3.8 we show 2-dimensional correlations between the transverse momenta of the outgoing b-quark and the intermediate top for the true (left) and GAN events (right). The phase space structure encoded in these two observables is clearly visible,

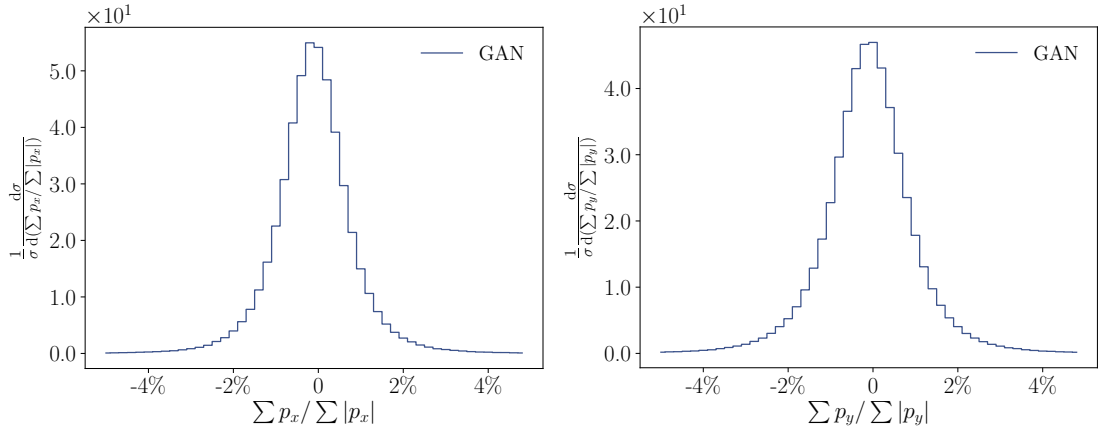

Figure 3.7: Sum of all p_x (p_y) momenta divided by the sum of the absolute values in the left (right) panel, testing how well the GAN learns momentum conservation.

and the GAN reproduces the peak in the low-p_T range, the plateau in the intermediate range, and the sharp boundary from momentum conservation in the high-p_T range. To allow for a quantitative comparison of true and generated events we show the bin-wise asymmetry in the lower left panel. Except for the phase space boundary the agreement is essentially perfect. The asymmetry we observe along the edge is a result from very small statistics. For an arbitrarily chosen p_T value of 100 GeV the deviations occur for $p_{T,b} \in [130, 140]$ GeV. We compare this region of statistical fluctuations in the asymmetry plot with a 1-dimensional slice of the correlation plot (lower right) for $p_{T,t} = 100 \pm 1$ GeV. The 1-dimensional distribution shows that in this range the normalized differential cross section has dropped below the visible range.

3.3 Precision event generation

So far, we have shown that it is possible to GAN the full phase space structure of a realistic LHC process, namely top pair production all the way down to the kinematics of the six final-state jets. Trained on a simulated set of unweighted events this allows us to generate any number of new events representing the same phase space information. With the help of an additional MMD kernel we described on-shell resonances. The only additional input was the final-state momenta related to on-shell resonances, and the rough phase space resolution of the on-shell pattern.

Until now, our comparison showed that relatively flat distributions can be reproduced up to a deviation of 5% in the bulk and 20% in the tails. The mass values defining intermediate resonance poles were also extracted from the dynamic GAN setup but also limited in precision.

Consequently, the question rises if a more profound approach possibly using more physical knowledge helps to improve the precision of the GANned events. Therefore, in a first step we compare two independent Monte Carlo samples. This will define an upper limit of performance and hence doing better than that is virtually impossible. One of the objective is to demonstrate that we can reach similar if not equal performances with a GAN , making the MC and the generated events practically indistinguishable.

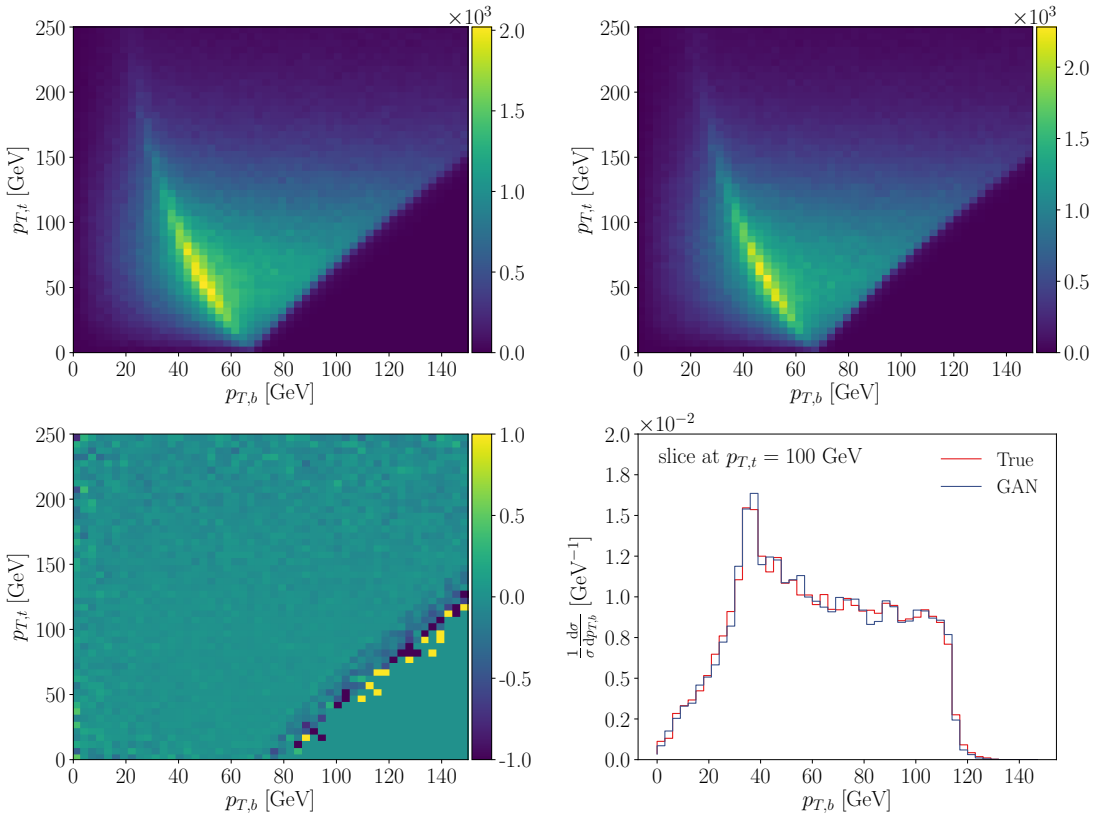

Figure 3.8: Correlation between $p_{T,t}$ and $p_{T,b}$ for the true data (upper left), GAN data (upper right) and the asymmetry between both (lower left). In addition, we show $p_{T,b}$ sliced at $p_{T,t} = 100 \pm 1$ GeV (lower right).

3.3.1 Performance benchmark

As a benchmark model we now consider $W + 2$ jet production

$$pp \to W^+ jj \tag{3.19}$$

illustrated in Fig. 3.9. With 4-momentum conservation and on-shell conditions for the final-state particles this leaves us with $3 \cdot 3 - 4 + 2 = 7$ degrees of freedom, taking the parton momentum fractions into account.

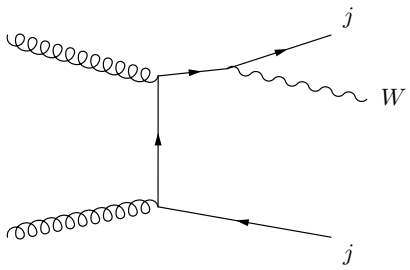

Figure 3.9: Sample Feynman diagram contributing to $W + 2$ jet production.

For the analysis we generate 3 million Monte Carlo events using `Madgraph5` [23] at a CM energy of 14 TeV. In order to avoid divergencies in the cross section, we apply the following simple phase cuts

$$p_{T,j} > 20\,\text{GeV}, \qquad \Delta R_{jj} = \sqrt{\Delta\phi_{jj}^2 + \Delta\eta_{jj}^2} > 0.4. \tag{3.20}$$

These events are split into 3 different sets: a training sample, a test sample and a benchmark sample with 1 million events each. To define a performance benchmark, we compare the test sample with the benchmark sample. For this we plot various kinematic observables O_i for both samples as well as their ratio and their absolute relative difference. As an example, in Fig. 3.10 we show the energy and transverse momentum distribution of the first jet in the upper row. In the lower row, we show the invariant mass distribution and the difference of the azimuthal angle of both jets.

As we can see from the lower panels in Fig. 3.10, the absolute relative difference and hence the deviation in this simple bin-wise comparison, is at the 1% level in the bulk and high populated regions, and decreases to around 10% in the tails where the phase space is sparsely populated.

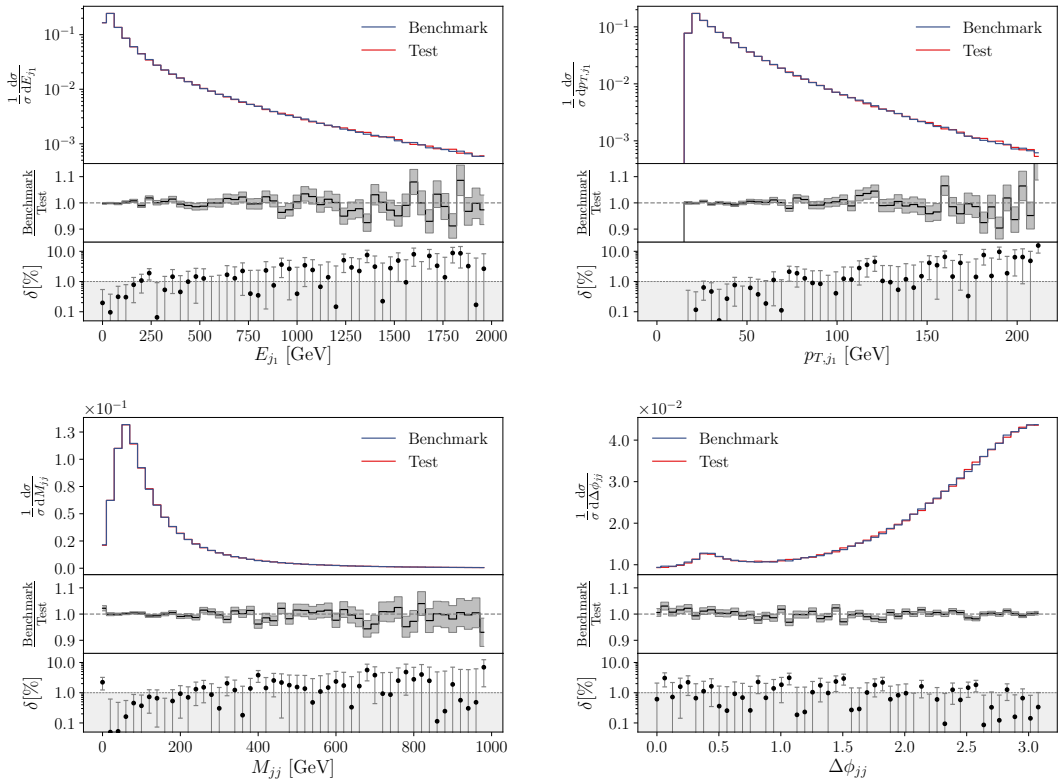

Figure 3.10: Energy (left) and transverse momentum (right) distribution of the first jet (upper row), and the invariant mass distribution (left) and the difference of the azimuthal angle (right) of both jets for the test (red) and benchmark (blue) Monte Carlo sample. In the middle and lower panel, the ratio and the absolute relative difference are shown.

3.3.2 W+2 jet production

The goal for our network is to generate events with high precision, *i.e.* having a deviation around 1% in the bin-wise comparison mentioned above. For this, we need to modify the neural network structure introduced in Section 3.2. In particular, we will use a more physics driven approach to define a generalizable network structure that is suitable to generate events in a more systematic and hence more precise manner. In detail, we want to satisfy the following criteria:

1. total 4-momentum conservation.
2. final state particles are on-shell.
3. proper description and parametrization of cuts and phase space boundaries.

Generator architecture

To begin with, we neglect the MMD term in the generator loss for now, as we do not have any intermediate resonances in the considered process. The MMD can of course always be added again if the considered process has strong varying features which need to be described properly. In order to obey the first two conditions most easily we change the number of generator outputs from $4n$ to $3n-2$, as the latter corresponds to the degrees of freedom in a hadronic scattering process of n particles. Any other observable can then easily be computed using momentum conservation and on-shell conditions.

In our example process (3.19) we have 3 final state particles and hence 7 degrees of freedom. We use this example to explain the parametrization in the generator. Let us assume, that the generator output is given by 7 real numbers $\{x_1, y_1, z_1, x_2, y_2, z_2, z_3\}$. We then parametrize the event in the following way

$$\begin{aligned} & & & \eta_1 = z_1, \\ p_{T,2} = p_{T,2}^{\min} + e^{x_2}, & & \phi_2 = y_2, & \eta_2 = z_2, \\ p_{T,3} = p_{T,3}^{\min} + e^{x_3}, & & \phi_3 = y_3, & \eta_3 = z_3, \end{aligned} \tag{3.21}$$

where the associated 4-momenta are connected to the final state particles by

$$p_1 = p_{W^+}, \qquad p_2 = p_{j_1}, \qquad p_3 = p_{j_2}. \tag{3.22}$$

Using these 7 degrees of freedom we can define the complete set of 4-momenta for the entire event satisfying momentum conservation and the on-shell conditions by construction. The choice of having a $p_T^{\min}$ is given by fact that events are generated with a p_T cut on the jets. Having this built-in cut helps to better reproduce the sharp phase space boundaries.

Discriminator architecture

For the discriminator we now employ spectral normalization [118] instead of a gradient penalty term (2.23) as this is numerically more efficient and hence saves computation time. Further, when using the spectral normalization instead of gradient penalty we do not have to fine-tune any hyperparameter to balance the importance of the penalty term.

In addition, we want to simplify the discriminator task in finding appropriate kinematic observables to distinguish the true and generated samples as well as to increase the sensitivity on those variables. For this, the discriminator does not only obtain the events paramatrized in the 7 variables used by the generator but also additional features. In particular, we implement two different layers between the generator and the discriminator. The first layer takes the minimal representation produced by the generator and transforms it into a complete set of 4-momenta using momentum conservation and on-shell conditions.

In the next layer, additional features such as ΔR_{ij}, M_{ij}, $\log E$ or

$$\mathcal{T}(p_T) = \log(p_T - p_T^{\text{cut}}) \tag{3.23}$$

are added to improve the discriminator sensitivity on phase space boundaries or possible sharp features. For instance, the observable $\mathcal{T}(p_T)$ maps the transverse momentum distribution with a sharp boundary and a rapidly dropping tail into a gaussian like distribution which is a much simpler discrimination variable. Indeed, we also take care that both the generated batches $\{x_G\}$ as well as the true batches $\{x_T\}$ are parametrized in the exact same way before feeding them into the discriminator.

Network performance

Using the mentioned ideas as a guideline, we figured out that we obtain the best results when parametrizing each individual particle as

$$\tilde{p}_i = \{p_{x,i}, p_{y,i}, p_{z,i}, \eta_i, \mathcal{T}(p_{T,i})\}, \tag{3.24}$$

and adding four additional features to complete the event representation

$$x = \{\tilde{p}_{W^+}, \tilde{p}_{j_1}, \tilde{p}_{j_2}, \Delta R_{jj}, M_{jj}, \Delta\phi_{jj}, \Delta\eta_{jj}\}. \tag{3.25}$$

For both the generator and the discriminator we use 6 layers with 128 units each combined with a leaky ReLU activation function in the hidden layers. Additionally, we use spectral normalization in the discriminator instead of a gradient penalty term. The other network parameters are given in Tab. 3.2. For the implementation we completely rely on the the `TensorFlow` (v2.1) framework [137].
In Fig. 3.11 we can see the results for 1 million GANed events compared to the truth (test) data sample. We again show the energy and transverse momentum distribution of the first jet as well as the invariant mass distribution and the difference in the azimuthal angle of both jets. In all distributions we only have a deviation of around 1% in the highly populated regions even compatible with 0 within the statistical error of the bin. In the high energy tails we observe the same behavior as in our benchmark comparison in Fig. 3.10, *i.e.* the deviation increases up to 10% which is, however, the performance limit set by statistics. In the p_T distribution we can see that due to the chosen parametrization in the generator the phase space cut at $p_T = 20\text{GeV}$ is exactly reproduced by the generated distribution. Furthermore, in the $\Delta\phi_{jj}$ distribution, we can see that the GAN is capable of reproducing the small peak around $\Delta\phi_{jj} \sim 0.4$. This peak rises from the $\Delta R > 0.4$ cut which was needed to obey a finite cross section. Using different parametrizations during the network hyperparameter tuning have shown that this peak is only correctly resolved if we feed $\Delta\phi$ into the discriminator. Thus, we can generally conclude that non-trivial features or peaks are best resolved if they are directly given to the discriminator.

Parameter	Value	Parameter	Value
Input dimension G	7	Input dimension D	19
Layers	6	Batch size	2048
Units per layer	128	Epochs	500
Trainable weights G	85k	Iterations per epoch	1000
Trainable weights D	85k	Number of training events	$1 \cdot 10^6$
Learning rate	$1 \cdot 10^{-4}$	Decay	0.1

Table 3.2: Details for our precision GAN setup.

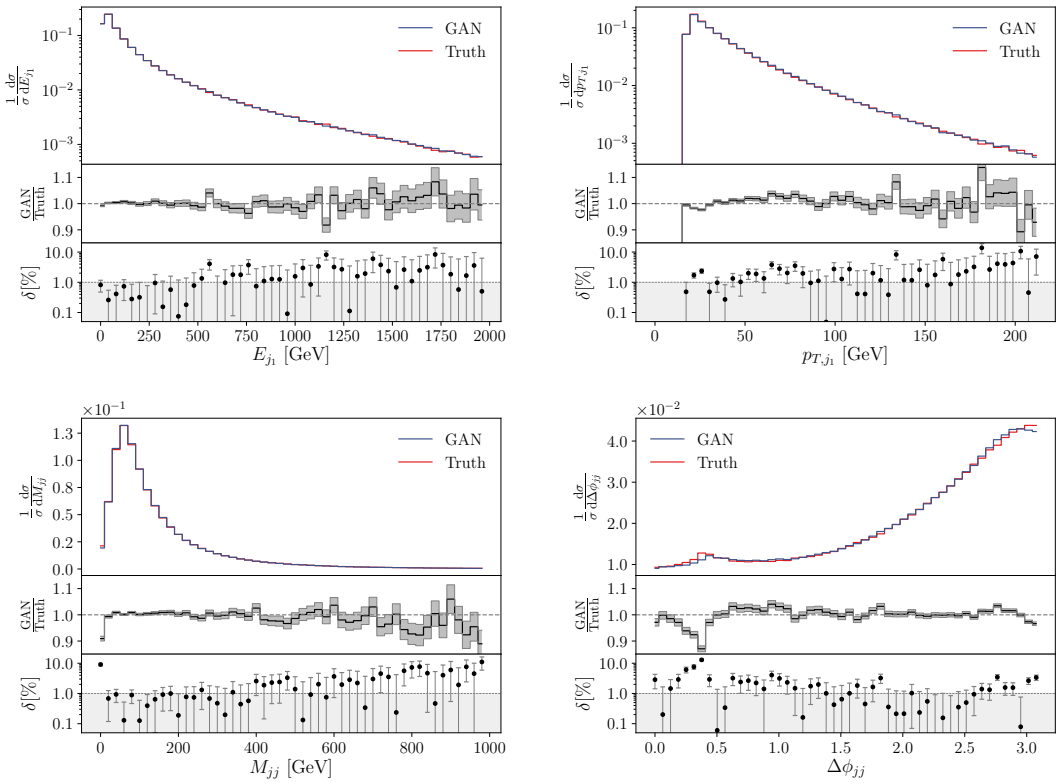

Figure 3.11: Energy (left) and transverse momentum (right) distribution of the first jet (upper row), and the invariant mass distribution (left) and the difference of the azimuthal angle (right) of both jets for the test (red) and GANned (blue) sample. In the middle and lower panel, the ratio and the absolute relative difference are shown.

3.4 Unweighting of weighted events

Inefficient unweighting procedures cause a main bottleneck for event generation as we have already pointed out in Section 1.5.2. Therefore, we could significantly improve the overall computation time of the simulation chain if we can train a GAN on possibly weighted events while still generating completely unweighted events. In this section, we first consider two simple toy examples: a one-dimensional camel distribution and a two-dimensional circle which are expressed in two sets of weighted events. Using the one-dimensional example we explain how the discriminator loss function needs to be modified to capture the weight information in the training data. Along with the previously defined unweighting efficiency (1.91) we can benchmark the performance of our unweighting GAN against a standard method like `Vegas` [29, 30]. Especially in the two-dimensional case we will benefit from the more flexible parametrization of neural networks. Finally, we will apply the derived method on the simple but well understood Drell–Yan process.

3.4.1 Toy examples

To start with, we consider a simple one-dimensional camel distribution given by

$$p_{\text{camel}}(x) = 0.3\,\mathcal{N}(x;\mu=0,\sigma=0.5) + 0.7\,\mathcal{N}(x;\mu=2,\sigma=0.5), \tag{3.26}$$

where $\mathcal{N}(x;\mu,\sigma)$ is the normal distribution. To explain how we treat possible event weights we define three sets of events. The first set of events

$$X_{\text{uw}} = (x_{\text{camel}}, w_{\text{unit}} = \text{const.}) \tag{3.27}$$

is a set of unweighted events which are distributed according to the camel function (3.26) but carrying a unit weight explicitly. The second set of events

$$X_{\text{w}} = (x_{\text{uniform}}, w_{\text{camel}}) \tag{3.28}$$

has elements x which are distributed uniformly and a weight $w_{\text{camel}} = p_{\text{camel}}$ carrying the information of the camel function. The unit weight w_{unit} in the first set is chosen such that the weighted histograms of both event sets look the same. Finally, we consider a third set

$$X_{\text{hybrid}} = (x_{q_1}, w_{q_2}), \tag{3.29}$$

where the probability information is spread across both the distribution of the elements and the weights, such that $p_{\text{camel}}(x) = N q_1(x) q_2(x)$, with some normalization factor N. In our example we chose $q_1(x) = \mathcal{N}(x;\mu=0,\sigma=1)$.

Standard GAN

First we consider the standard GAN application in which the full density information is encoded in the distribution of the events. In this case, we use the non-saturating GAN loss functions as introduced in (2.20) and (2.21)

$$\begin{aligned} L_D &= \mathbb{E}_{x\sim p_{\text{data}}}[-\log D_\varphi(x)] + \mathbb{E}_{x\sim p_\theta}[-\log(1 - D_\varphi(x))], \\ L_G &= -\mathbb{E}_{x\sim p_\theta}[\log(D_\varphi(x))]. \end{aligned} \tag{3.30}$$

To stabilize the training we again add the standard gradient penalty term L_{GP} as defined in (2.23). Both the generator and discriminator have 128 units in 4 layers. In the hidden layers, we employ the ELU and leaky ReLU activation function in the generator and discriminator, respectively. In order to compensate imbalance in the training the discriminator is updated 4 times as often as the generator. We use this architecture for all shown 1-dimensional examples.

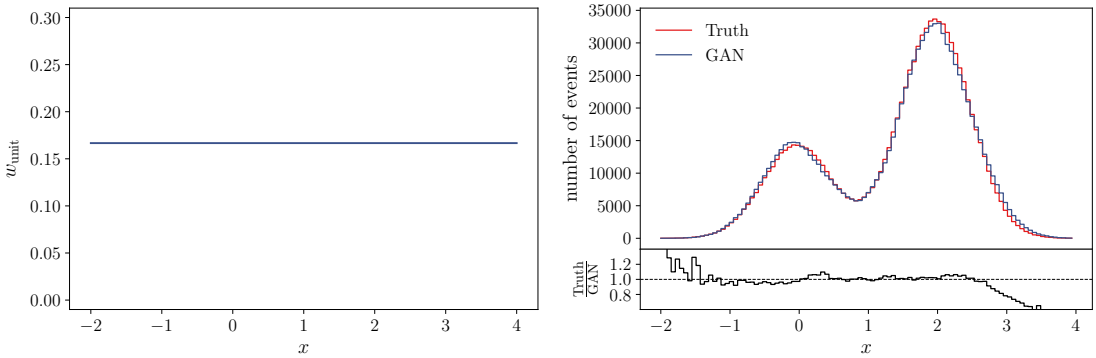

Figure 3.12: The distribution of the event weights w_{unit} (left) and the GAN and truth distribution of the combined X_{uw} sample (right). The lower panel of the right plot shows the bin-wise ratio of both distributions.

In Fig. 3.12 we can see that our GAN nicely reproduces the full one dimensional target distribution only being slightly off in the tails. This does not yield any news and only serves as a benchmark for the more interesting setups in which we need to modify the losses to handle weighted event sets.

Unweighting GAN

If we want to capture the event weights carried by each event we need to modify the loss function by replacing the normal expectation value by a weighted mean for batches coming from the weighted data set. This then gives

$$\begin{aligned} L_D^{(\text{uw})} &= \frac{\mathbb{E}_{x\sim\text{unif}}[-w_\text{camel}(x)\log D_\varphi(x)]}{\mathbb{E}_{x\sim\text{unif}}[w_\text{camel}(x)]} + \mathbb{E}_{x\sim p_\theta}[-\log(1-D_\varphi(x))], \\ L_G^{(\text{uw})} &= -\mathbb{E}_{x\sim p_\theta}[\log(D_\varphi(x))], \end{aligned} \tag{3.31}$$

where $w_\text{camel}(x) = p_\text{camel}(x)$ is the camel distribution. Further, we have assumed that $w_G(x) = \text{const.} = 1$ and thus the weighted mean simply is the standard expectation value for the generated batches. By this construction, the generator still produces unweighted events even though it is trained on weighted truth events. Consequently, our unweighting GAN (uwGAN) is effectively unweighting the data.

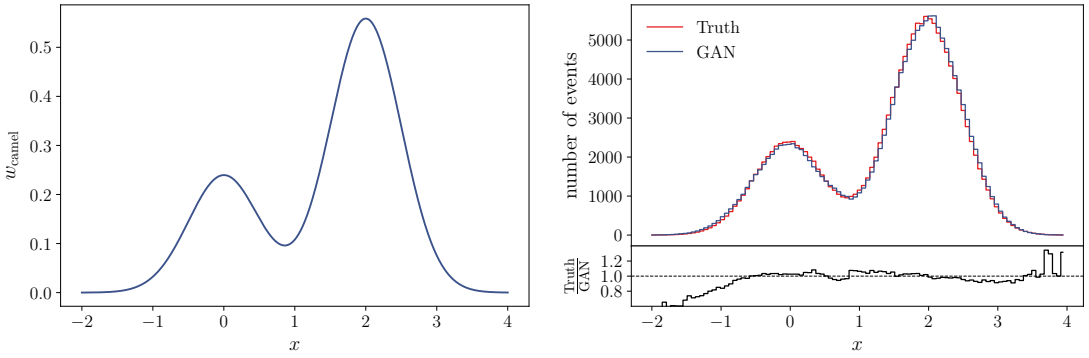

Figure 3.13: The distribution of the event weights w_camel (left) and the uwGAN and truth distribution of the combined X_w sample (right). The lower panel of the right plot shows the bin-wise ratio of truth to uwGAN distribution.

In detail, if we train on the weighted event set X_w, we have $w_T(x) = p_\text{camel}(x)$. Applying our modified uwGAN on the weighted event set X_w then results in generated unweighted events $X_{G,\text{uw}}$ which are distributed according to the camel distribution, as shown in Fig. 3.13.

We can now generalize this to the case in which the information is encoded in both, the weights and the distribution of the events, such that $p_\text{camel}(x) = Nq_1(x)w_{q_2}(x)$. In this case we can write the loss function as

$$\begin{aligned} L_D^{(\text{uw})} &= \frac{\mathbb{E}_{x\sim q_1}[-w_{q_2}(x)\log D_\varphi(x)]}{\mathbb{E}_{x\sim q_1}[w_{q_2}(x)]} + \mathbb{E}_{x\sim p_\theta}[-\log(1-D_\varphi(x))], \\ L_G^{(\text{uw})} &= -\mathbb{E}_{x\sim p_\theta}[\log(D_\varphi(x))], \end{aligned} \tag{3.32}$$

The results for the hybrid event sample X_hybrid is shown in Fig. 3.14. The uwGAN has no problems to extract the information from both the weights and the data simultanously and can translate this into unit weights and generates completely unweighted events.

Unweighting efficiency and weight distribution

As we have already introduced in Section 2.3.1, the generator is a mapping $G : \mathcal{Z} \to \Phi$ from a latent space $\mathcal{Z}$ to the target space Φ inducing a distribution

$$p_z(z) \xrightarrow{G_\theta} p_\theta(x) \equiv p_\theta(G_\theta(z)) \tag{3.33}$$

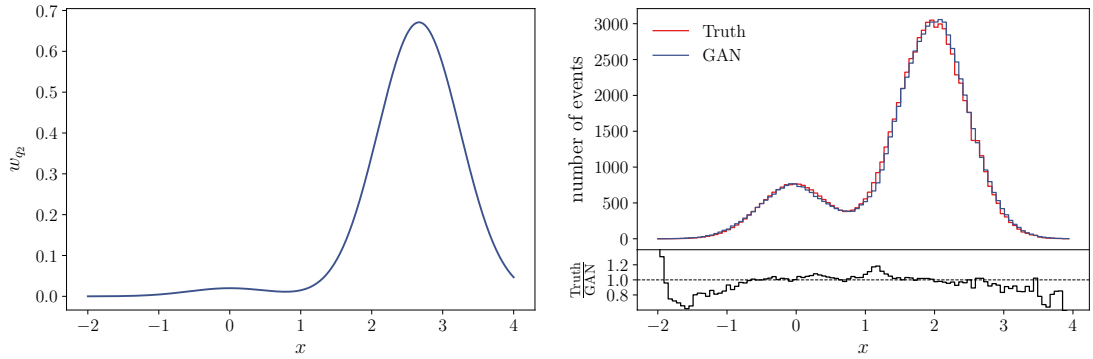

Figure 3.14: The distribution of the event weights w_{q_2} (left) and the uwGAN and truth distribution of the combined X_{hybrid} sample (right). The lower panel of the right plot shows the bin-wise ratio of truth to uwGAN distribution.

on the target space, where $p_z(z)$ is the prior distribution on the latent space. Consequently, following the idea of importance sampling in Section 1.4.2, we can understand this induced distribution p_θ as an estimator of the true distribution $p_\theta \approx p_{\text{data}}$. In this sense, the generator yields a better sampling of the target space and hence improves the unweighting efficiency. In order to actually calculate the unweighting efficiency we define

$$w_\theta(x) = \frac{p_{\text{data}}(x)}{p_\theta(x)}. \tag{3.34}$$

This exactly recasts the idea of importance sampling and introduces a new distribution or weight $w_\theta(x)$ which we use for the actual unweighting procedure. Intuitively, it also makes sense to define this quantity as the goal of the GAN is to achieve $p_\theta(x) = p_{\text{data}}(x)$ and hence $w_\theta(x) = 1$ leading to an unweighting efficiency of 1. We can now properly unweight the GANned events using the hit-or-miss method introduced in Section 1.5.2. In detail, this method now consists of the following steps

1. Generate data points $\{x_G\}$ and calculate the weights $w_\theta(x)$.

2. Determine the maximal weight

$$w_{\theta,\text{max}} = \max_{x \in \{x_G\}} w_\theta(x). \tag{3.35}$$

3. Define the relative weights

$$w_{\theta,\text{rel}}(x) = \frac{w_\theta(x)}{w_{\theta,\text{max}}}. \tag{3.36}$$

4. Draw a random number $R_i \in [0,1]$ for every event $x_i \in \{x_G\}$ and keep it only if

$$w_{\theta,\text{rel}}(x_i) > R_i \tag{3.37}$$

   is satisfied, otherwise reject it

In order to calculate the unweighting efficiency we use its definition in Eq. (1.91) and write it as

$$\epsilon_{\text{uw}} = \frac{\mathbb{E}_{x \in \{x_G\}}[w_\theta(x)]}{w_{\theta,\text{max}}}. \tag{3.38}$$

We can now calculate the unweighting efficiency of our uwGAN for the camel distribution function in Fig. 3.13 and obtain an unweighting efficiency of 40% to 50% over several runs. Although the distributions look very similar, there are problems in the tails. There, the weight w_θ reaches values up to $w_{\theta,\text{max}} \approx 2.5$. Since the efficiency calculation is strongly dependent on the maximum weight, we get a low unweighting efficiency.

If we restrict ourselves only to events which are not to far in the tails, *i.e.* $x_G \in [1.5, 3.5]$, we get an unweighting efficiency of 93%. Note that statistically 99.73% of the events should be in the restricted interval indicating that the GAN only fails in regions of low statistics.

In order to benchmark the performance of our uwGAN, we compare it to the results we obtain when using the `Vegas` [29, 30] algorithm instead. After some adaption steps we obtain an unweighting efficiency of 93%. However, after even more adaption steps the efficiency goes down to around 80%. The reason for that is that the `Vegas` alogrithm is designed to get good integration results by having a very tight grid in the bulk and a very wide grid in the tails. Consequently, the longer `Vegas` adapts its grid the more points are taken away from the tails which is fine for improving the integral estimate, but it decreases the unweighting efficiency. This can be seen in the ratio of the truth and `Vegas` distribution in the left plot of Fig. 3.15, where the ratio is fluctuating a lot.

Since the unweighting efficiency alone does not represent reliable quantity, we explicitly show the weight distributions w_θ and $w_{\texttt{Vegas}}$ for our uwGAN and `Vegas` in the right plot of Fig. 3.15. In a perfect scenario we would get a constant weight of one. We can see that in the one dimensional case the `Vegas` algorithm leads to much better results. This was expected as the `Vegas` algorithm is highly fine-tuned and designed for these kind of distributions.

Two dimensional example: circle

After we have explained how to modify the loss functions using the one-dimensional example we now want to consider a 2-dimensional example. If the chosen distribution does not factorize, we do not expect `Vegas` to outperform our uwGAN anymore. Therefore, we define a suitable 2-dimensional toy example: a circle in the x-y plane. In detail, the distribution function is given by

$$p_{\text{circle}}(x, y) = N_{\text{circle}}\, e^{-\frac{1}{2\sigma^2}\left(\sqrt{(x-x_0)^2+(y-y_0)^2}-r_0\right)^2}, \tag{3.39}$$

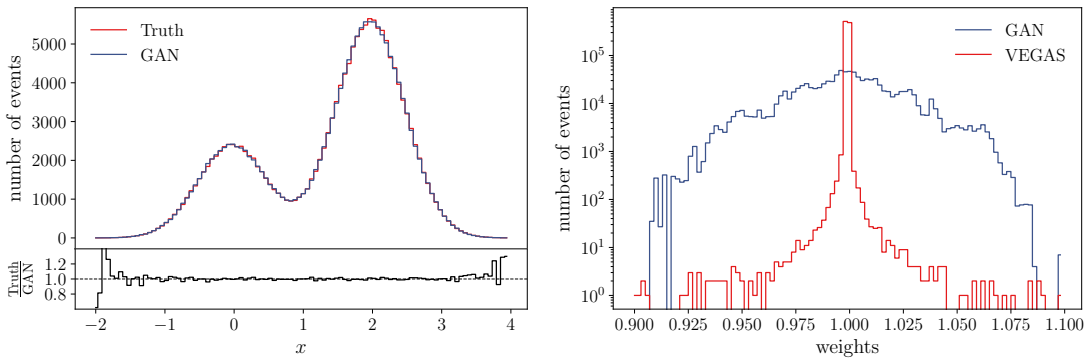

Figure 3.15: Desired and unweighted distribution with the VEGAS algorithm. In the lower plot again the ratio of the distributions.

where σ is the standard deviation and N_{circle} is a normalization factor. When choosing $x_0 = y_0 = 0.5$, $r_0 = 0.25$ and $\sigma = 0.05$ we obtain $N_{\text{circle}} \approx 5.079$.

In comparison to the 1-dimensional case we slightly modify our network architecture and replace the ELU activation function by the ReLU activation function in the generator. Furthermore, we now use 256 units within 8 layers in both the generator and the discriminator. The results of the uwGAN for the 2-dimensional example is shown in Fig. 3.16.
In Fig. 3.8 we show the results for the 2-dimensional example true (left) and GAN events (right). The circular structure is clearly visible, and the GAN reproduces distribution nicely. To allow for a more quantitative comparison of true and generated events we show the bin-wise asymmetry in the lower left panel. Except for the tails and sparsely populated regions the agreement is essentially perfect. In the lower right panel we show a slice at $x = 0.5$ and see that indeed the GAN nicely reproduces the truth distribution.

When we calculate the unweighting efficiency we obtain an efficiency of 45%. This is not as high as for the camel function. On one hand this is due to the higher dimension which generally makes the generator mapping more complicated. On the other hand the distribution is not just sparsely populated outside of the circle but also in the center. Therefore, we also undershoot the central region and hence end up with an efficiency of only 45%.

When applying the `Vegas` algorithm to the 2-dimensional case the algorithm cannot reproduce the right shape. As we can see in the left panel of Fig. Fig. 3.17 we obtain a quadratic distribution instead of the circle. This is mainly due to the fact that the circle

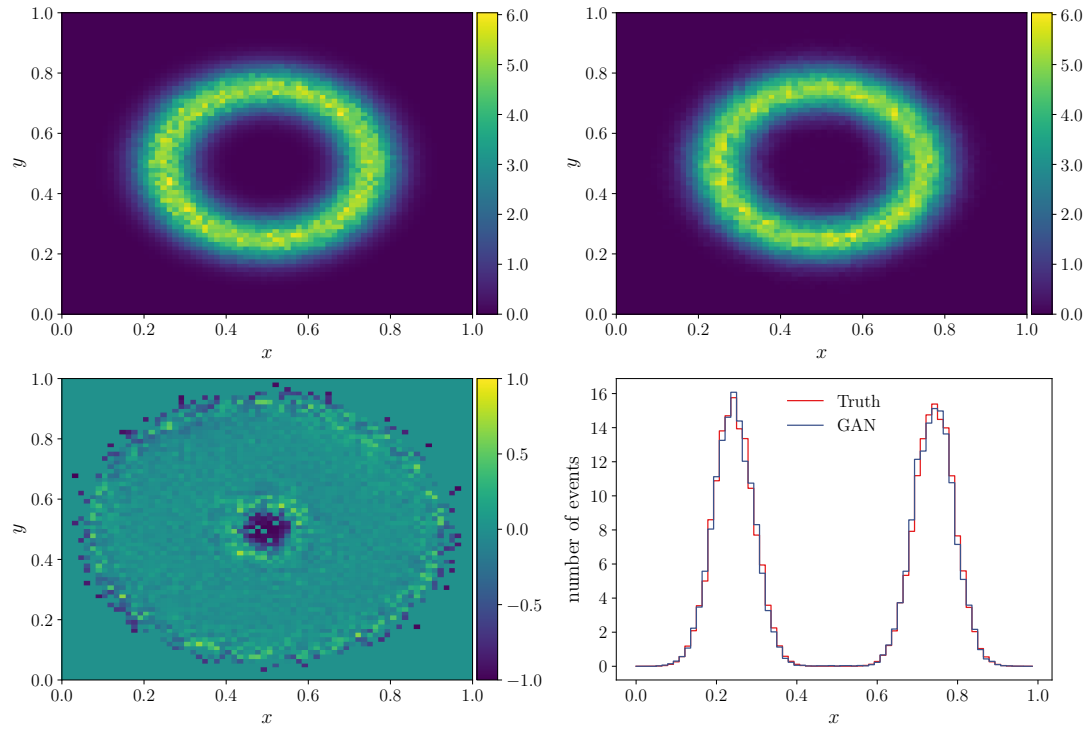

Figure 3.16: Results for the two dimensional case for the true data (upper left), uwGAN data (upper right) and the asymmetry between both (lower left). In addition, we show a one-dimensional slice at $y = 0.5$ (lower right).

distribution is not factorizable

$$p_{\text{cirle}}(x, y) \neq p_x(x)\, p_y(y). \tag{3.40}$$

and hence cannot be parametrized with a rectangular grid. Consequently, we only get an unweighting efficiency of 15% for the `Vegas` algorithm.

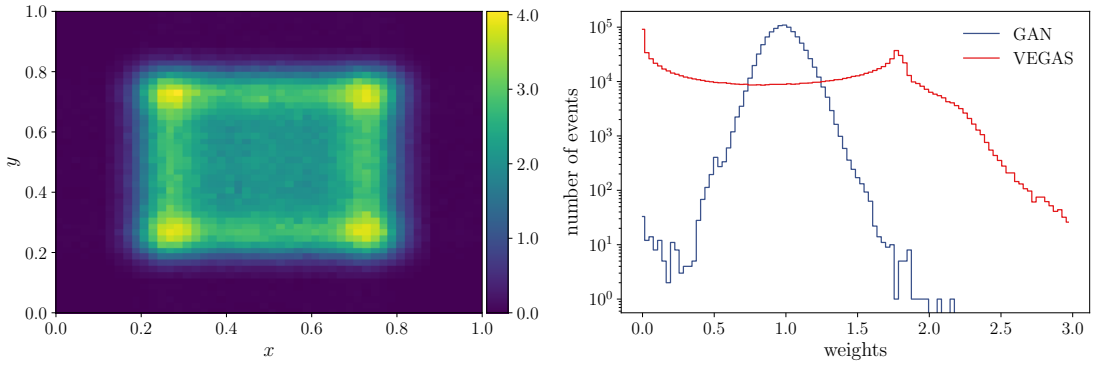

Figure 3.17: Result for the unweighting procedure with VEGAS for the two dimensional toy model.

As for the 1-dimensional, we show the weights w_θ of $w_{\texttt{Vegas}}$ in the right panel of Fig. 3.17. We can see that we get a gaussian-like result for our uwGAN which is peaked around 1. However, as already anticipated, the `Vegas` weight distribution is rather flat and does not indicate a peak around 1. This shows, that our uwGAN is already clearly outperforming `Vegas` in this 2-dimensional toy example. We expect `Vegas` to perform even worse if the distribution is higher-dimensional and more complex.

3.4.2 Drell–Yan process

So far we have only considered toy examples to elaborate the details of our uwGAN and showed that it already outperforms classical methods in 2-dimensional case. Now, we want to apply the uwGAN on a realistic but simple LHC process. In particular, we consider the Drall–Yan process producing two muons

$$pp \to \mu^- \mu^+, \tag{3.41}$$

as illustrated in Fig. 3.18. For our analysis we generate 1 million weighted events with `Sherpa` [24] at a CM energy of 14 TeV. We are interested in the fiducial phase space defined by the following phase space cuts

$$p_{T,\mu} > 25 \text{ GeV} \qquad |\eta_\mu| < 2.5. \tag{3.42}$$

Owing to momentum and energy conservation this also implies a cut on the invariant mass

$$M_{\mu\mu} > 2 p_{T,\mu} = 50 \text{ GeV}. \tag{3.43}$$

For the generator architecture we employ the same ideas as presented in Section 3.3 and produce only the degrees of freedom of the process. By construction this again guarantees momentum conservation and on-shell conditions. Before passing to the discriminator both the generated batches $\{x_G\}$ and the truth batches $\{x_T\}$ are parametrized as

$$x = \{p_{\mu^-}, p_{\mu^+}, w\}, \tag{3.44}$$

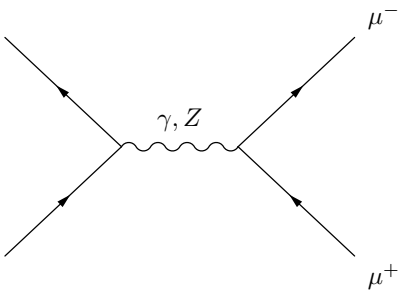

Figure 3.18: Sample Feynman diagram contributing to myon pair production.

where $p_{\mu^\pm}$ is the 4-momentum in cartesian coordinates, and w is the associated event weight. Since we want to produce unweighted events the generated event weight is simply given by a unit weight $w_G = 1$.

In Fig. 3.19 we show different kinematic distributions. As examples, we show the transverse momentum, the azimuthal angle and the pseudorapidity distribution of the μ^- as well as the invariant mass of the muon pair. We can see that all distributions nicely agree with the truth distribution. Only around the phase space cut in the transverse momentum distribution the GAN is overshooting the truth distribution. However, so far we did neither use an explicit cut parametrization in the generator nor did we feed the $\mathcal{T}(p_T)$ (3.23) variable to the discriminator to enhance the sensitivity in that region. Using the former one and adapting the parametrization in the generator will resolve the problem as shown in Sec. 3.3. Further, we can see that the invariant mass $M_{\mu\mu}$ distribution is extremely well reproduced even though we did not employ a MMD loss, as introduced in Section 3.2.

Note that we would also like to calculate the unweighting efficiency in this more realistic scenario. While the calculation of the induced distribution p_θ is straight forward the calculation of the truth event weight $w_T = p_{\text{data}}$ is generally non-trivial. The event weight of a LHC process is the combination of the squared matrix element, the PDFs and the phase space weight. In our case, the training data has been produced with `Sherpa` which uses a combination of different phase space remappings to improve the sampling efficiency. Unfortunately, once the events have been produced the internal mappings are discarded and hence the phase space weight cannot be evaluated for the GANed phase space points.

In order to solve this problem we could for instance employ the simple RAMBO algorithm introduced in Sec. 1.4.1 to generate phase space points uniformly. In this case the phase space factor or weight is only dependent on the total momentum Q (1.68) and hence easy to compute. Usually, RAMBO is not the best choice as a phase space generator because the generated phase space is flat and thus yields a bad sampling efficiency. However, as the uwGAN can extract all necessary information from the weights, the RAMBO algorithm does not limit the performance of our GAN.

3.5 Conclusion

We have shown that it is possible to GAN the full phase space structure of several realistic LHC processes, such as top pair production all the way down to the kinematics of the six top decay jets, $W + 2$ jet production and a simple Drell–Yan process. Trained on a simulated set of either unweighted or weighted events this allows us to generate any

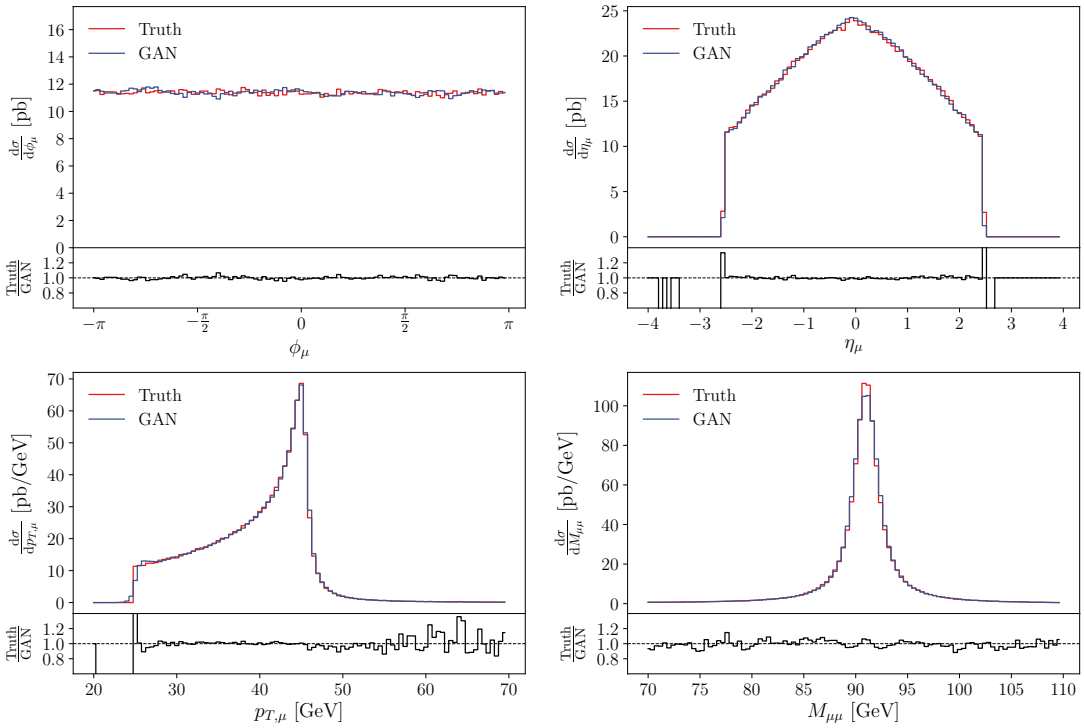

Figure 3.19: Azimuthal angle (top left), pseudorapidity (top right) and transverse momentum (bottom left) distribution of the μ^-, as well as the invariant mass distribution (bottom right) of both myons for the truth (red) and GAN (blue) events. The lower panels show the bin-wise ratio of MC truth to GAN distribution.

number of new unweighted events representing the same phase space information.
With the help of an additional MMD kernel we described on-shell resonances as well as tails of distributions. The only additional input was the final-state momenta related to on-shell resonances, and the rough phase space resolution of the on-shell pattern.
Simple modifications to both the generator and discriminator network significantly increases the performance of the GAN. Our detailed comparison showed that relatively flat distributions can be reproduced at arbitrary precision, limited only by the statistics of the training sample. Augmenting additional features to the discriminator input increases its sensitivity to certain phase space regions and hence improves the GAN precision.
Replacing the normal expectation value in the discriminator loss with a weighted mean the GAN can be trained on weighted data while still producing unweighted events. This will speed-up the generation of training data and consequently the entire tool chain will be computationally more efficient.

Chapter 4

Sample-Based Subtraction of Distributions

The research presented in this chapter has been previously published in Reference [3]. The displayed figures and tables as well as most the text are identical to the content of this article.

4.1 Introduction

Modern analyses of LHC data are increasingly based on a data-to-data comparison of measured and simulated events. The theoretical basis of this approach are generated samples of unweighted or weighted LHC events. To match the experimental precision such samples have to be generated beyond leading order in QCD. In modern approaches to perturbative QCD at the LHC such simulations include subtraction terms, leading to events with negative weights. Examples for such subtraction event samples are subtraction terms for fixed-order real emission [62–66], multi-jet merging including a parton shower [67, 68], on-shell subtraction [69], or the subtraction of precisely known backgrounds [70].

GANs [42] are neural networks which naturally lend themselves to operations on event samples, as we will show in this chapter. Such generative networks have been proposed for a wide range of tasks related to LHC event simulation and are expected to lead to significant progress once they become part of the standard tool box. This includes for instance phase space integration [37], event generation [1, 57–59], detector simulations [46–51, 129], unfolding [2, 4, 141], parton showers [52–56], or searches for physics beyond the Standard Model [60].

In this chapter, we show how GANs can perform simple operations on event samples, namely adding and subtracting existing samples. Such a network is trained to generate unweighted events with a phase space density corresponding to a sum or difference of two or more input samples. We will illustrate the idea behind a generative event sample subtraction and addition in Section 4.2. This example shows how generative networks can beat the statistical limitations of the training samples. Specifically, we produce events with statistical fluctuations which are significantly smaller than the corresponding statistical fluctuations of the training data. The feature behind this naively impossible improvement are the excellent interpolation properties of neural networks in a high-dimensional phase space.

In Section 4.3 we will then subtract unweighted 4-vector events for the LHC in two examples. First, we subtract the photon continuum from the complete Drell–Yan process and find the Z-pole and the known interference patterns. This can be seen as a toy example for a background subtraction at the level of parton-level event samples. For instance,

this setup could allow us to study the kinematics of four-body decay signals, simulated to high precision from observed background and signal-plus-background samples.

Finally, we combine a hard matrix element for jet radiation with collinear subtraction events. This gives us an event sample that follows the matrix element minus the subtraction term without any intermediate binning in the phase space. We show how this subtraction works even if we do not make use of the local structure of the subtraction terms. It illustrates how simulations in perturbative QCD might benefit from GANs, in soft-collinear subtraction, on-shell subtraction, or a veto-like combination of phase space and parton shower.

4.2 Toy example

The advantage of GANs learning how to subtract event samples can be seen easily from statistical uncertainties in event counts. Traditionally, we generate the two samples and combine them through some kind of histogram. If we start with $N+n$ events and subtract $N \gg n$ statistically independent events, the uncertainty on the combined events in one bin is given by

$$\Delta_n = \sqrt{\Delta_{N+n}^2 + \Delta_N^2} \approx \sqrt{2N} \gg \sqrt{n}. \tag{4.1}$$

In any bin-wise analysis the bin width has to be optimized. On the one hand larger bins with more events per bin minimize the relative statistical error, but on the other hand they reduce the resolution of features.
In our GAN approach we avoid defining such histograms and replace the explicit event subtraction by a subtraction of interpolated sample properties over phase space. We will first develop this approach in terms of a simple toy example and then show how it can be extended to unweighted 4-vector events as used in LHC simulations. Unfortunately, there does not (yet) exist a rigid description of statistical and systematic uncertainties associated with GANs, but we will show how the fluctuations we observe in our generated samples are visibly smaller than what we would expect from the input data and Eq.(4.1).

Note, that a more general treatment of uncertainties in GANs has been investigated in Ref. [142] and whether GANs can amplify the training statistics was examined in Ref. [143].

4.2.1 Single subtraction

We start with a simple 1-dimensional toy model, *i.e.* toy events which are described by a single real number x. We then define a base distribution P_B and a subtraction distribution P_S as

$$P_B(x) = \frac{1}{x} + 0.1, \qquad P_S(x) = \frac{1}{x}. \tag{4.2}$$

The target distribution for the subtraction is then

$$P_{B-S} = 0.1. \tag{4.3}$$

To produce unweighted subtracted events our GAN is trained to generate the event sets $\{x_B\}$ and $\{x_S\}$ simultaneously. It thereby learns the distribution P_{B-S} using the information encoded in the two input samples.
The corresponding GAN architecture is shown in Fig. 4.1 and consists of a generator and two independent discriminators, one for each dataset. The generator takes random

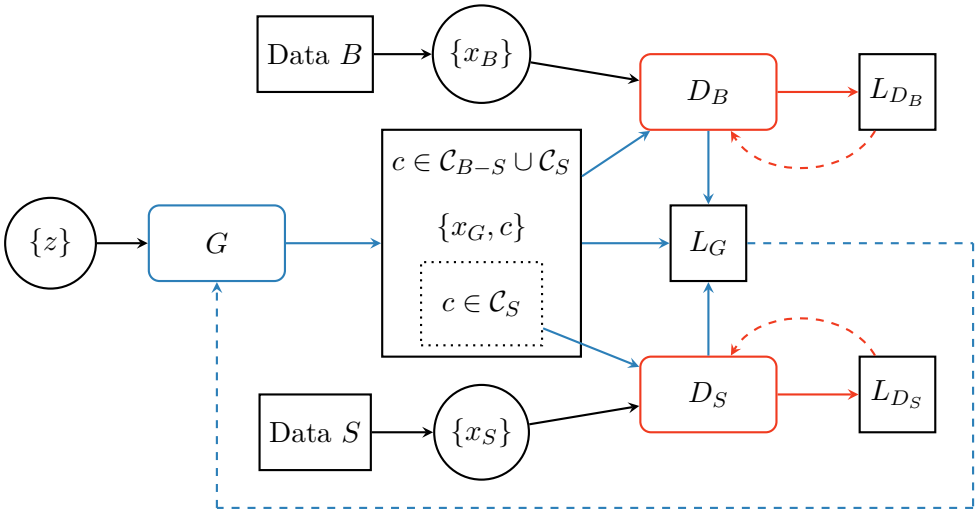

Figure 4.1: Structure of our subtraction GAN. The input $\{z\}$ describes a batch of random numbers and $\{x_{B,S}\}$ the true input data batches. The label c encodes the category of the generated events. Blue arrows indicate the generator training, red arrows the discriminators training.

noise $\{z\}$ as input and generates samples $\{x_G, c\}$, where x_G stands for an event and c for a label. The underlying idea is to start from an event sample which follows P_B and split it into two mutually exclusive samples following P_S and P_{B-S}, with class labels $\mathcal{C}_S$ or $\mathcal{C}_{B-S}$. During training we demand that the distribution over events from class $\mathcal{C}_S$ follow P_S while the full event sample follows P_B. After normalizing all samples correctly the events with class label $\mathcal{C}_{B-S}$ will then follow the distribution P_{B-S}.
Technically, the class label c attached to each event is a real 2-dimensional vector, such that it can be manipulated by the network. Through the softmax function (2.7) in the final generator layer the entries of c are forced into the interval $[0, 1]$ and sum up to 1. We then create a so-called one-hot encoding by mapping c to

$$c_i^{\text{one-hot}} = \begin{cases} 1 & \text{if } c_i = \max(c) \\ 0 & \text{else} \end{cases}. \tag{4.4}$$

This representation is two-dimensional binary and most convenient for manipulating the samples. We can use it to define the label classes via $\mathcal{C}_i = \{c \mid c_i^{\text{one-hot}} = 1\}$.

In Fig. 4.1 we see that for the class $\mathcal{C}_S$ and the union of $\mathcal{C}_S$ with $\mathcal{C}_{B-S}$ we train the discriminators to distinguish between events from the input samples and the generated events. The training of the discriminators D_i corresponding to the two input samples $\{x_S\}$ and $\{x_B\}$ uses the standard discriminator loss function (2.20)

$$L_{D_i} = \mathbb{E}_{x\sim p_{\text{data}_i}}[-\log D_{\varphi_i}(x)] + \mathbb{E}_{x\sim p_\theta}[-\log(1 - D_{\varphi_i}(x)]. \tag{4.5}$$

We also add the gradient penalty term (2.23) and obtain a regularized loss function

$$\begin{aligned} L_{D_i}^{(\text{GP})} = L_{D_i} &+ \lambda_{D_i} \mathbb{E}_{x\sim p_{\text{data}_i}} \left[(1 - D_{\varphi_i}(x))^2 \left|\nabla d_{\varphi_i}(x)\right|^2\right] \\ &+ \lambda_{D_i}\, \mathbb{E}_{x\sim p_\theta} \left[D_{\varphi_i}(x)^2 \left|\nabla d_{\varphi_i}(x)\right|^2\right], \end{aligned} \tag{4.6}$$

where we defined

$$d_{\varphi_i}(x) = \log \frac{D_{\varphi_i}(x)}{1 - D_{\varphi_i}(x)}. \tag{4.7}$$

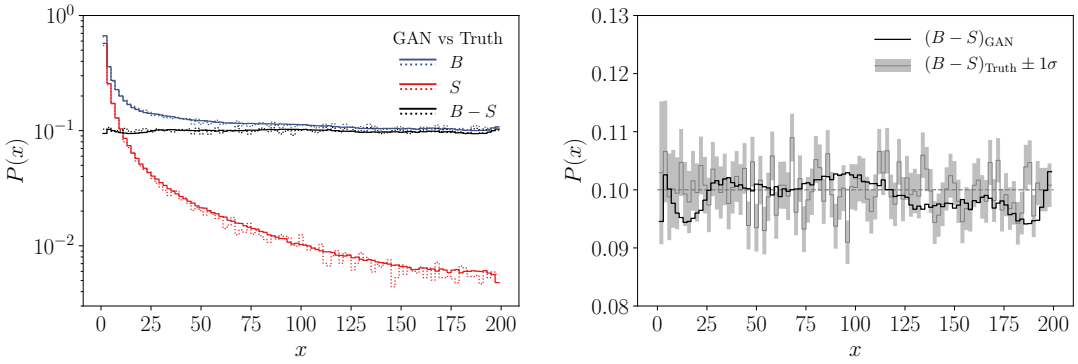

Figure 4.2: Left: Generated (solid) and true (dashed) events for the two input distributions and the subtracted output. Right: distribution of the subtracted events, true and generated, including the error envelope propagated from the input statistics.

In parallel, we train the generator to fool the discriminators by minimizing

$$L_G = \sum_i \mathbb{E}_{x\sim p_\theta}[-\log D_{\varphi_i}(x)]. \tag{4.8}$$

An additional aspect in manipulating samples is that we need to keep track of the normalization or number of events in each class. To generate a clear and differentiable assignment we introduce the function

$$f(c) = e^{-\alpha(\max(c)^2-1)^{2\beta}} \in [0,1] \qquad \text{for} \qquad 0 \leq c_i \leq 1. \tag{4.9}$$

Adapting α and β we can make the gradient around the maximum steeper and push $f(0) \to 0$. In that case $f(c) \approx 1$ only if one of the entries of $c_i \approx 1$. By adding

$$L_G^{(\text{class})} = \left(1 - \frac{1}{b} \sum_{c\in batch} f(c)\right)^2 \tag{4.10}$$

to the loss function we reward a clear assignment of each event to one class and generate a clear separation between classes. Finally, we use the counting function in combination with masking to fix the normalization of each sample with

$$L_G^{(\text{norm})} = \sum_i \left(\frac{\sum_{c\in\mathcal{C}_i} f(c)}{\sum_{c\in\mathcal{C}_B} f(c)} - \frac{\sigma_i}{\sigma_0}\right)^2 . \tag{4.11}$$

Adding these losses to the generator loss we get

$$L_G \to L_G^{(\text{full})} = L_G + \lambda_{\text{class}} L_G^{(\text{class})} + \lambda_{\text{norm}} L_G^{(\text{norm})} , \tag{4.12}$$

with properly chosen factors λ_{class} and λ_{norm}. In this chapter we always use $\lambda_{\text{class}} = \lambda_{\text{norm}} = 1$. For the denominator in Eq. (4.11) we always choose the approximation of the number of predicted events in the base class $\mathcal{C}_B = \mathcal{C}_{B-S} \cup \mathcal{C}_S$ as reference value. The integrated rates σ_i have to be given externally. For our toy model we can compute them analytically while for an LHC application they are given by the cross section from the Monte Carlo simulation.
Our GAN uses a vector of random numbers as input. The size of the vector has to be at least the number of degrees of freedom. For the implementation we have used

`Keras` 2.2.4 [136] with a `TensorFlow` 1.14 backend [137]. The discriminator and generator networks consist of 5 layers with 128 units per layer using the ELU activation function (2.6). With $\lambda_{D_i} = 5 \cdot 10^{-5}$ and a batch size of 1024 events, we run for 4000 epochs. Each epoch consists of one update of the generator and 20 updates of the discriminator. We found that the intense training of the discriminator is necessary to reach sufficiently precise results. To obtain a good separation of the classes with $f(c)$ we set $\alpha = 10$ and $\beta = 1$. Finally, using the adam optimizer [108] throughout this chapter, we choose a learning rate of $3 \cdot 10^{-4}$ for generator and discriminator and a large decay following Eq. (2.17) of the learning rate of $2 \cdot 10^{-2}$ for the discriminator which stabilizes the training. The decay for the generator is slightly smaller with $5 \cdot 10^{-3}$. Our training datasets consist of 10^5 samples for each dataset $\{x_S\}$ and $\{x_B\}$.

We show numerical results for a single GAN subtraction and analyze the size of the statistical fluctuations in Fig. 4.2. In the left panel we show the two input distributions defined in Eq. (4.2), as well as the true and generated subtracted distribution. The dotted lines illustrate the shape of the training dataset, while the full lines show the generated distribution using $5 \cdot 10^6$ events. The former two distributions only serve to confirm that the GAN learns the input information correctly. The generated subtracted events indeed follow the probability distribution in Eq. (4.3). Aside from the fact that all three distributions show excellent agreement between truth and GANned events, we see how the neural network interpolates especially in the tail of the distribution. In the right panel of Fig. 4.2 we zoom into the subtracted sample to compare the statistical uncertainties from the input data with the behavior of the GAN. The uncertainty is estimated from the number of events per bin in the base and subtraction histogram N_B and N_S, taking into account the corresponding normalization factors n_B and n_S. In analogy to Eq. (4.1) we compute it as

$$\begin{aligned} \Delta_{B-S} &= \Delta_{n_B N_B - n_S N_S} \\ &= \sqrt{\Delta^2_{n_B N_B} + \Delta^2_{n_S N_S}} \\ &= \sqrt{n_B^2 N_B + n_S^2 N_S}. \end{aligned} \tag{4.13}$$

As mentioned above, we expect the GAN to deliver more stable results than we could expect from the input sample, because the GAN interpolates all input distributions. This way we avoid a bin-by-bin statistical uncertainty of the subtracted sample. Indeed, our subtracted curve in the right panel of Fig. 4.2 lies safely within the 1σ region of the data. The statistical fluctuations of the GANned events are much smaller than the statistical fluctuations in the input data. On the other hand, the GANned distribution shows systematic deviations, but also at a visibly smaller level than the statistical fluctuation of the input data. While this observation does not imply a proof that GANs can beat the statistical limitations of the input data, they give a clear hint that the interpolation properties can balance statistics at some level.

4.2.2 Combined subtraction and addition

To show how our approach could be generalized to subtracting and adding any number of event samples we can extend our single subtraction toy model by a third sample to be added to the difference described in Eq. (4.3). We now consider three samples corresponding to the 1-dimensional distributions

$$P_B(x) = \frac{1}{x} + 0.1, \qquad P_S(x) = \frac{1}{x}, \qquad P_A(x) = \frac{m}{\pi} \frac{\gamma}{\gamma^2 + (x - x_0)^2}. \tag{4.14}$$

As a third input we add the Breit-Wigner distribution P_A, so our target distribution becomes

$$\begin{aligned} P_{B-S+A} &= \frac{m}{\pi} \frac{\gamma}{\gamma^2 + (x - x_0)^2} + 0.1 \\ &= \frac{5}{\pi} \frac{10}{100 + (x - 90)^2} + 0.1 \,, \end{aligned} \tag{4.15}$$

for the values $m = 5$, $\gamma = 10$, and $x_0 = 90$. We now sample $\{x_B\}$, $\{x_S\}$ and $\{x_A\}$ individually from the input distributions and want to learn the probability distribution P_{B-S+A}. The approach is the same as described before, but for three classes as shown in Tab. 4.1 and a three-dimensional class vector. Treating the subtraction exactly as before we obtain our target distribution P_{B-S+A} by adding the events with class $\mathcal{C}_A$.
Compared to the sample subtraction introduced before, adding samples is obviously not a big challenge. In principle, we could just add the unweighted event samples in the correct proportion, learn the phase space structure with a GAN, and then generate any number of events very efficiently. The reason why we discuss this aspect here is that it shows how our subtraction GAN can be generalized easily.
In Fig. 4.3 we show the numerical results of subtracting one distribution $\{x_S\}$ from the base distribution $\{x_B\}$ and adding a second distribution $\{x_A\}$ with a distinct feature. As before, this combination is learned from the three input distributions without binning the corresponding phase space. The hyperparameters are slightly modified with respect to the simple subtraction model. The networks now consist of 7 layers with 128 units which we train for 1000 epochs with 4 iterations. We fix the relative weight of the gradient penalty to $\lambda_{D_i} = 5 \cdot 10^{-5}$. The separation of the three classes is efficient for $\alpha = 5$ and $\beta = 1$. Finally, we set the learning rate to $8 \cdot 10^{-4}$ and its decay to $2 \cdot 10^{-2}$ for generator and discriminator. The remaining parameters are the same as for the pure subtraction case. In the left panel of Fig. 4.3 we confirm that the GAN indeed learns the three input structures correctly and interpolates each of them smoothly. We also see that the generated events follow the combination $B - S + A$ with its flat tails and the central Breit–Wigner shape. As for the pure subtraction in Fig. 4.2 we also compare the statistical fluctuation of the binned input data with the behavior of the GANned events. The GAN extracts the additional Breit–Wigner feature with high precision, but, as always, some systematic deviations arise in the tails of the distribution.

4.2.3 General setup

Finally, we note that our network setup is not limited to three classes. We can generalize it to a base distribution, M subtraction datasets, and N added datasets. The corresponding category assignment, generalized from Tab. 4.1, is given in Tab. 4.2 and

	$\mathcal{C}_{B-S}$	$\mathcal{C}_S$	$\mathcal{C}_A$
Data B	1	1	0
Data S	0	1	0
Data A	0	0	1
$B - S + A$	1	0	1

Table 4.1: Category assignment for a combined addition and subtraction of three samples.

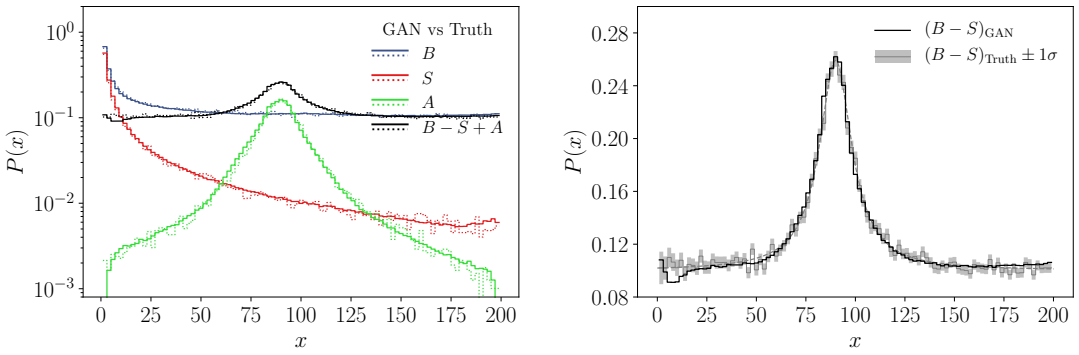

Figure 4.3: Left: Generated (solid) and true (dotted) events for the three input distributions and the combined output. Right: distribution of the combined events, true and generated, including the error envelope propagated from the input statistics.

encoded in an enlarged classification vector c. The base class is then defined as

$$\mathcal{C} = \bigcup_{i=0}^{M} \mathcal{C}_i \,. \tag{4.16}$$

In this case the network has to learn all $M+N+1$ input distributions through individual discriminators D_B, D_{S_i}, and D_{A_j} with $i \leq M$ and $j \leq N$. The rough structure of the network is given in Fig. 4.4. The training of the generator follows directly from the description above. While we do not benchmark this extended setup in this chapter, we expect it to be useful when a set of subtraction terms accounts for different features, and splitting them improves their simulation properties.
Until now we have always assumed that we can subtract a sample $\{x_S\}$ from a sample $\{x_B\}$ and find a well-behaved distribution for $B-S$. Specifically, the resulting probability P_{B-S} should be positive all over phase space. This is not always the case. First, we note that a global sign of the combination is not a problem, because we can always learn $S - B$ instead of $B - S$. Next, changing signs in the S or B contributions can be accommodated by splitting the respective sample according to the sign and applying the combined subtraction and addition described in Section 4.2.2. A phase space dependent

	$\mathcal{C}_0$	$\mathcal{C}_1$	$\mathcal{C}_2$	$\cdots$	$\mathcal{C}_M$	$\mathcal{C}_{M+1}$	$\cdots$	$\mathcal{C}_{M+N}$
Data B	1	1	1	$\cdots$	1	0	$\cdots$	0
Data S_1	0	1	0	$\cdots$	0	0	$\cdots$	0
Data S_2	0	0	1		0	0	$\cdots$	0
$\vdots$	$\vdots$	$\vdots$		$\ddots$		$\vdots$		$\vdots$
Data S_M	0	0	0		1	0	$\cdots$	0
Data A_1	0	0	0	$\cdots$	0	1		0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$		$\ddots$	
Data A_N	0	0	0	$\cdots$	0	0		1
Combination	1	0	0	$\cdots$	0	1	$\cdots$	1

Table 4.2: Details for the category selection in the general case.

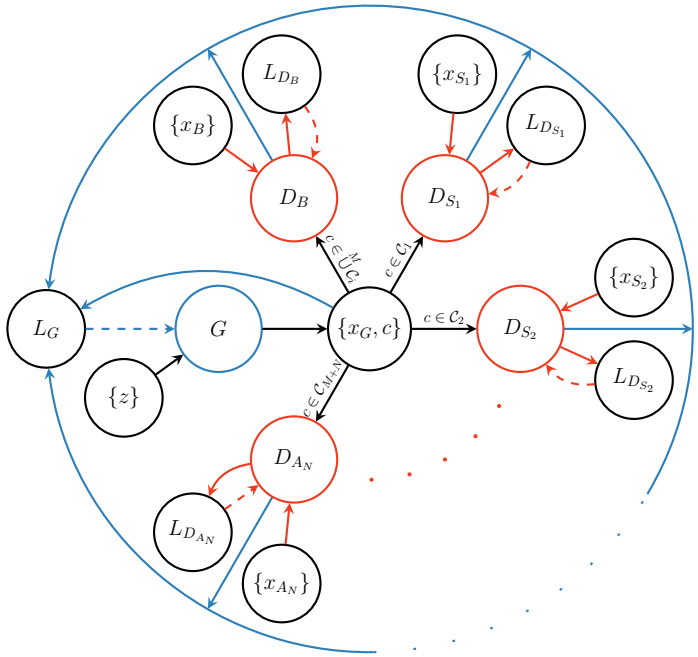

Figure 4.4: Structure of our general subtraction and addition GAN. The input $\{z\}$ describes a batch of random numbers and $\{x\}$ the true input data or generated batches. The label c encodes the category of the generated events. Blue arrows indicate the generator training, red arrows the discriminators training.

sign in $S-B$ could be most easily accommodated by adding a constant off-set either by hand or again using the combined subtraction and addition. A typical example would be to add the Born term to the virtual correction before subtracting the dipole.
In cases where this is not a suitable solution, we can replace the categories $\mathcal{C}_{B-S}$ and $\mathcal{C}_S$ by the three categories $\mathcal{C}_{B\cap S}$, $\mathcal{C}_{B\setminus S}$, and $\mathcal{C}_{S\setminus B}$. They indicate events corresponding to B and S, only B, or only S. The discriminator compares for instance the combination of $\mathcal{C}_{B\cap S}$ and $\mathcal{C}_{B\setminus S}$ with the B-data. The difference $B-S$ will be given by events with label $\mathcal{C}_{B\setminus S}$ in regions where $B > S$ and events with label $\mathcal{C}_{S\setminus B}$ in regions where $B < S$, the latter weighted with weight minus one. While this simple extension of the label vector is very straightforward, the third category induces an additional degree of freedom in the way the network can distribute events into different categories. This freedom needs to be constrained to prevent the network from simply assigning for instance all events into the categories $\mathcal{C}_{B\setminus S}$, and $\mathcal{C}_{S\setminus B}$. A possible solution would be to maximize the number of events in $\mathcal{C}_{B\cap S}$ via a term in the loss function and force the network to share as many events between the distributions as possible.

4.3 LHC events

After showing how it is possible to GAN-subtract 1-dimensional event samples from each other we want to show how such a tool can be applied in LHC physics. In this case the (unweighted) events are 4-momenta of external particles. We ignore all information on the particle identification, except for its mass, which allows us to reduce external 4-momenta to external 3-momenta [1, 2]. Because the input events might have been object to detector effects we do not assume energy-momentum conservation for the entire event. This means that the network has to learn the 4-dimensional energy-momentum

conservation and this subtraction of simple LHC events is inherently multi-dimensional. We will present two simple examples for LHC event subtraction, the separation of on-shell Z and photon contributions to the Drell-Yan process and the subtraction of collinear gluon radiation in Z+jet production.

4.3.1 Background subtraction

Our first example for event subtraction at the LHC is the Drell–Yan process, which receives contributions with distinct phase space features from the photon and from the Z-boson, as seen in Fig. 4.5. The specific question in our setup is if we can subtract a background-like photon continuum contribution from the full process and generate events only for the Z-exchange combined with the interference term,

$$\begin{aligned} B: &\qquad pp \to e^+e^- \\ S: &\qquad pp \to \gamma \to e^+e^- \; . \end{aligned} \tag{4.17}$$

We generate 1M events with `Madgraph5` [23] for an LHC energy of 13 TeV, applying minimal cuts on the outgoing electrons. We require a minimal p_T of 10 GeV, a maximal rapidity of 2.5 for each electron, and a minimal angular separation of 0.4. We do not apply a detector simulation at this stage, because our focus is on comparing the generated and true distributions, and we have already shown that detector simulations can be included trivially in our GAN setup [1, 2].

Aside from the increased dimension of the phase space the subtraction GAN has exactly the same structure as the toy example of Section 4.2. The hyperparameters have to be adjusted to the increased dimensionality of the phase space. We use a 16-dimensional latent space. The discriminator and generator networks consist of 8 layers with 80 and 160 units per layer, respectively. In this high-dimensional case we use the leaky ReLU activation function. Further, we choose $\lambda_{D_i} = 10^{-5}$ and a batch size of 1024 events and train for 1000 epochs. Each epoch consists of 5 iterations in which the discriminator gets updated twice as much as the generator. For a proper separation of the classes with $f(c)$ we set $\alpha = 5$ and $\beta = 1$. Finally, we choose a large decay of the learning rate of 10^{-2} which stabilizes the training and pick a learning rate at the beginning of 10^{-3}. Our training datasets consist of 10^5 samples for each dataset $\{x_B\}$ and $\{x_S\}$..

In Fig. 4.6 we show the performance of the LHC event subtraction for two example distributions. First, we clearly see the Z-mass peak in the lepton energy of the full sample, compared with the feature-less photon continuum in the subtraction sample. The subtracted curve is expected to describe the Z-contribution and the interference. It smoothly approaches zero for small lepton energies, where the interference is negligible.

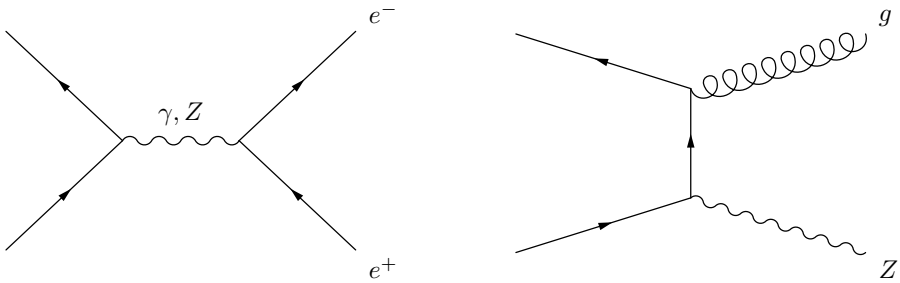

Figure 4.5: Sample Feynman diagrams for the background subtraction (left) and collinear subtraction (right) applications.

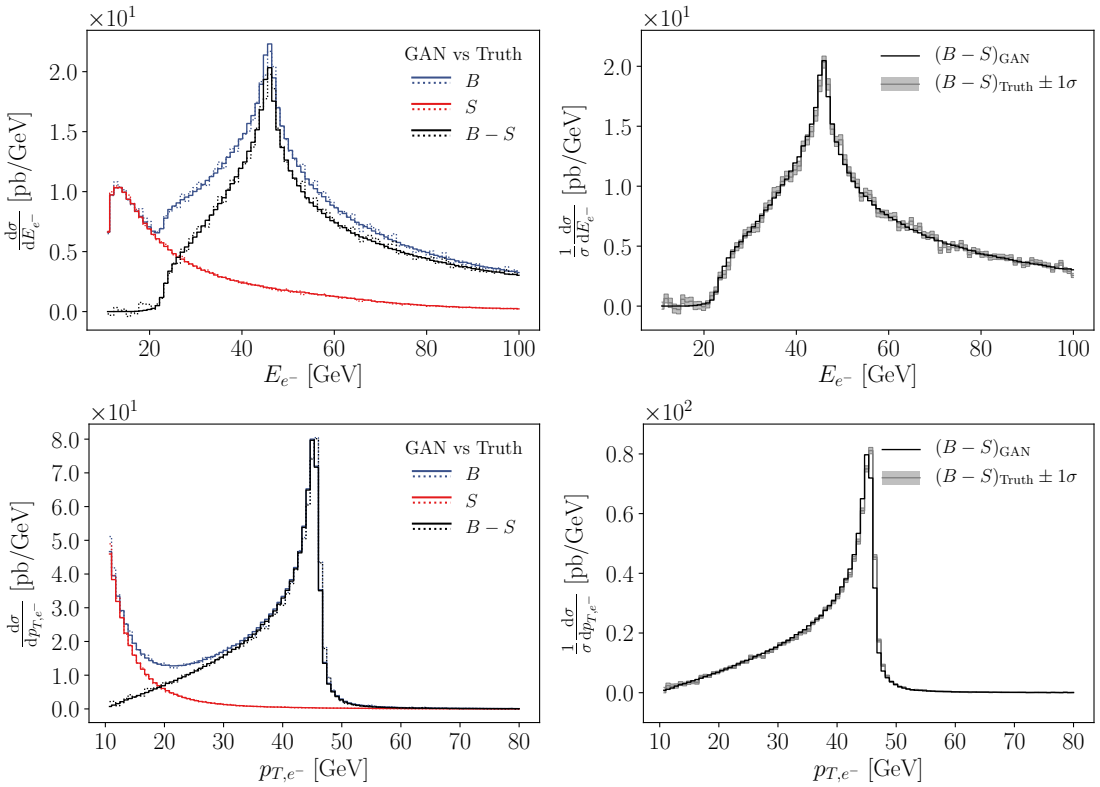

Figure 4.6: Left: Generated (solid) and true (dashed) e^+e^- events at the LHC for the two input distributions and the subtracted output. Right: distribution of the subtracted events, true and generated, including the error envelope propagated from the input statistics.

Above that we see the Jacobian peak from the on-shell decay, and for larger energies a small interference term enhancing the high-energy tail. In the right panel we show the subtracted curve including the statistical uncertainties from the input samples. As the second observable we show the transverse momentum of the electron. Here the Z-pole appears as a softened endpoint at $m_Z/2$. The photon continuum dominates the combined distribution for small transverse momenta. Indeed, the GAN-subtracted on-shell and interference contribution is localized around the endpoint, with a minor shift in the resolution at the edge.

Indeed, our subtraction of the background to a di-electron resonance is not a state-of-the-art problem in LHC physics. A more interesting application of our method could be four-body decays. We could start from a combined signal plus background sample of Higgs decays to four fermions, generate a background-only sample using control regions, and then GAN a set of signal events. While in a regular analysis the events we obtain from subtracting a background from the signal-plus-background sample do not reflect the signal properties, our GANned subtraction events should reflect all kinematic features of the signal events in the data.

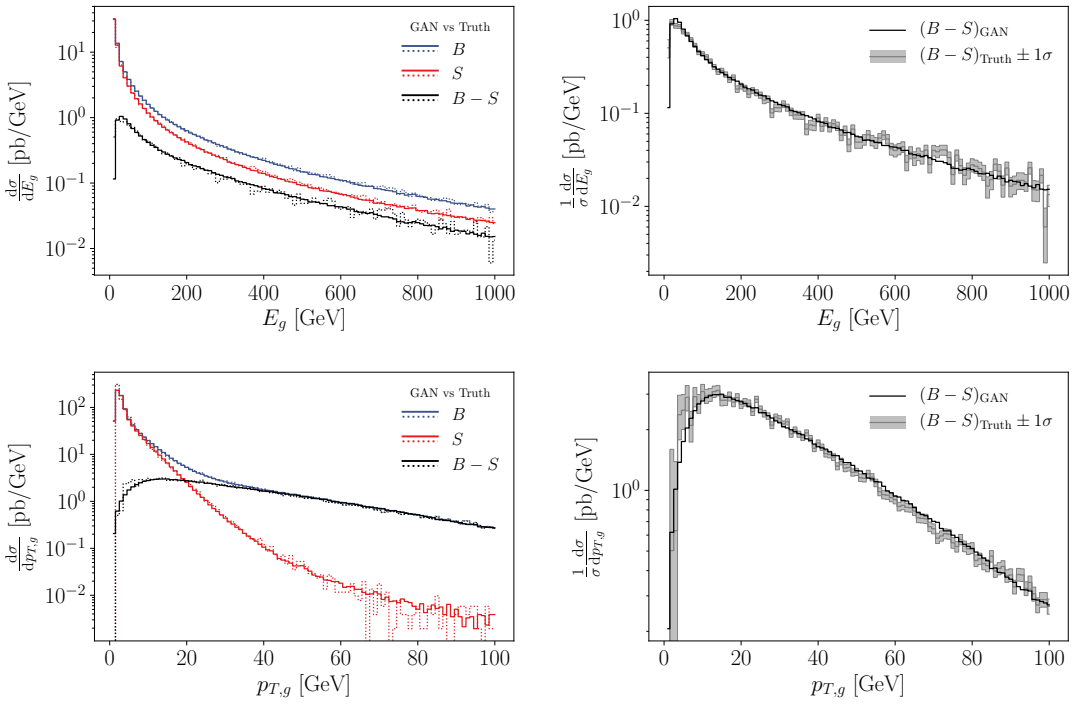

Figure 4.7: Left: Generated (solid) and true (dashed) Zg events at the LHC for the two input distributions and the subtracted output. Right: distribution of the subtracted events, true and generated, including the error envelope propagated from the input statistics.

4.3.2 Collinear subtraction

The second example for event subtraction at the LHC is collinear radiation off the initial state, for instance

$$\begin{aligned} B: &\quad pp \to Zg \quad \text{(matrix element)} \\ S: &\quad pp \to Zg \quad \text{(collinear approximation)} \end{aligned} \tag{4.18}$$

We generate 1M events for the hard process with `Sherpa` [24], where the Z-boson decays to electrons. For the network we combine the electron and positron momenta to a 4-momentum of the Z-boson, so we obtain a Breit–Wigner distribution with $m_{ee} = 66 \ldots 116$ GeV instead of an on-shell condition. We then subtract the corresponding Catani-Seymour dipoles [62] for the gluon radiation off each of the incoming quarks, based on 1M events each. The corresponding Feynman diagram is shown in the right panel of Fig. 4.5. To avoid the soft divergence we require $p_{T,g} > 1$ GeV in the training data, a smaller cutoff would be possible but increases the training time. We apply the same external cutoff to the GANned samples, aligning the phase space boundaries of the training and GANned data sets by hand.
The problem with this specific process is that the Catani-Seymour dipoles describe the full matrix element over a huge part of phase space [**?**] and the combination of hard matrix element and dipoles is typically tiny and negative. We discussed changing signs in probability distributions in Section 4.2.3. In addition, the one distribution a GAN can never generate is a probability distribution compatible with zero everywhere. In this case the GAN would either over-fit statistical fluctuations or become unstable. This is why in our toy application we shift the Catani-Seymour dipole by a constant such that

the cancellation of the divergent matrix element still works, but the combined result integrated over phase space remains finite.

Note that the kinematics of our subtraction terms are not the same as in fixed-order calculations, instead it is similar to the mapping in the modified subtraction method MC@NLO [144]. In this case the global efficiency of event generators at NLO accuracy is dominated by the efficiency of computing the subtracted real-emission corrections, which presents a major challenge for event simulation at the HL-LHC [36, 145].

The hyperparameters have to be modified with respect to the background subtraction example, due to the large cancellations in the low energy regime. Now, the discriminator and generator networks consist of 8 layers with 256 and 512 units per layer, respectively. In the generator we alternate leaky ReLU and tanh activation functions. We achieve the best and most stable results choosing $\lambda_{D_i} = 10^{-3}$ with a batch size of 1024 events. We train for 60000 epochs, where each epoch consists of 5 iterations in which the discriminator gets updated twice as much as the generator. In this example, the discriminator gets the events in the $\{E, p_T, \eta, \phi\}$ representation, which is better suited to resolve the p_T distribution. The other hyperparameters are kept the same as in the background subtraction.

We show the results from the collinear subtraction in Fig. 4.7. The GAN perfectly reconstructs the cancellation in the energy spectrum and the transverse momentum of the emitted gluon. The left panel shows the distribution of the real (B) and dipole (S) contributions to the process and their difference ($B - S$). With the logarithmic axis we see that the GAN smoothly interpolates over the entire energy range. For small gluon energies and momenta the GAN reproduces the rate increase towards the (enforced) phase space boundary, including the finite value of the subtracted combination $B - S$. Also in the high energy region, which suffers from low statistics, the GAN nicely matches the truth distributions. In the right panel we show the subtracted curve including the error envelope of the input data.

As before, we only use the established NLO dipole as a simple structure to illustrate the features of our subtraction GAN. Proper applications could be the more complicated subtraction terms beyond NLO or the subtraction of on-shell resonances [69]. The latter would combine aspects discussed in Section 4.3.1 and Section 4.3.2 and allow for a fully inclusive study of the kinematics in the off-shell process, without having to actually do a subtraction and deciding if a given event is more likely to be part of the on-shell or off-shell sample.

4.4 Conclusion

We have shown how to generate events representing the difference between two input distributions with a GAN. As a toy example we used events representing a 1-dimensional probability distribution. Because the GAN interpolates the input while learning the difference between the two distributions, it circumvents the statistical limitations of large cancellations. We have found that the GAN-subtracted events lead to a very stable phase space coverage and beat the statistical limitations of the input sample over the entire phase space.

For a slightly more realistic setup we have GANned background subtraction and collinear dipole subtraction for Drell–Yan production at the LHC. In the first case the network learned on-shell final state momenta to subtract the photon-induced continuum from the full e^+e^- production. It could serve as a test case for a background subtraction for

four-body decays, such that the GANned signal events reflect the kinematic correlations of the actual signal events hidden in the background.

In the second case we combined the hard matrix element with modified Catani-Seymour dipoles for gluon emission into a stable finite prediction of the real emission process. We are aware of the fact that our toy examples are not more than an illustration of what a subtraction GAN can achieve. However, we have shown how to use a GAN to manipulate event samples avoiding binning (at least in particle physics) and we hope that some of the people who do LHC event simulations for a living will find this technique useful.

Chapter 5

Unfolding of Detector Level Events

The research presented in this chapter has been previously published in Ref. [2] and [4]. The displayed figures and tables as well as most of the text are identical to the content of these articles.

5.1 Introduction

The unique feature of LHC physics from a data science perspective is the comparison of vast amounts of data with predictions based on first principles. This modular prediction starts with the Lagrangian describing the hard scattering, then adds perturbative QCD providing precision predictions, resummed QCD describing parton showers and fragmentation, hadronization, and finally a full detector simulation [146]. In this so-defined forward direction all simulation modules are based on Monte Carlo techniques, and in the ideal world we would just compare measured and simulated events and draw conclusions about the hard process. This hard process is where we expect to learn about new aspects of fundamental physics, for instance dark matter, extended gauge groups, or additional Higgs bosons.

Because our simulation chain works only in one direction, the typical LHC analysis starts with a new, theory-inspired hypothesis encoded in a Lagrangian as new particles and couplings. For every point in the new physics parameter space we simulate events, compare them to the measured data using likelihood methods, and discard the new physics hypothesis. This approach is extremely inefficient. First, we know that the best way to compare two hypotheses is the log-likelihood ratio based on new physics and Standard Model predictions for the hard process. Using this ratio in the analysis is the idea behind the matrix element method [147–152], but usually this information is not available. Second, any new physics hypothesis has free parameters like masses or couplings, and even if an analysis is independent of these model parameters we cannot avoid simulating events for each point in model space. Finally, it is impossible to derive competitive limits on a new model by recasting an existing analysis.

All these shortcomings of LHC analyses point into the same direction: we need to invert the simulation chain, apply this inversion to the measured data, and compare hypotheses at the level of the hard scattering. For hadronization and fragmentation an approximate inversion is standard in that we always apply jet algorithms to extract simple parton properties from the complex QCD jets. Obviously, this only works for analyses which do not benefit from subjet information. For the detector simulation either at the level of particles or at the level of jets this problem is usually referred to as detector unfolding. For instance in top physics we also unfold kinematic information to the level of the decaying top quarks, assuming that the top decays are correctly described

by the standard model [153,154]. If we assume that QCD jet radiation adds little to our new physics search, we should at some level be able to also unfold this last simulation step. This is the goal of this chapter.

Technically, we propose to use invertible networks (INNs) [43–45] or GANs [42] to invert parts of the LHC simulation chain. This application builds on a long list of one-directional applications of generative or similar networks to LHC simulations, including phase space integration [37, 38], amplitudes [155, 156], event generation [1, 57–59, 157], event subtraction [3], detector simulations [46–51, 129, 158, 159], unfolding [141] parton showers [52–55], or searches for physics beyond the Standard Model [60].

The question is if and how we can invert them. In Sec. 5.2 we shortly introduce the inverse problem and explain it with the help of a toy example. Afterwards, we start with a naive GAN and INN inversion and see how a mismatch between local structures at parton level and detector level leads to problems. We then introduce a fully conditional GAN [71] (FCGAN) and conditional INN (cINN) [72, 73] to invert a fast detector simulation [27] for the process

$$pp \to ZW^+ \to (\ell^-\ell^+)\,(jj), \tag{5.1}$$

as illustrated in Fig. 5.1. We will see how the fully conditional setup gives us all the required properties of an inverted detector simulation. The cINN adds even more sampling elements to the generation of unfolded configurations. For arbitrary kinematic distributions we can test the calibration of this generative network output using truth information and find that unlike GANs the cINN lives up to its generative promise: for a single detector-level event the cINN generates probability distributions in the multi-dimensional parton-level phase space.

Finally, we show in Sec. 5.5 how the inversion can link two phase spaces with different dimensions. This means that we can for instance unfold based on a model with a variable number of final state particles at the detector level. This is crucial when we include higher-order perturbative corrections, which come with an increased number of partons in the final state. The hard scattering process which we choose to unfold then includes a reduced number of relevant final-state particles. We show how the cINN can account for this QCD jet radiation and unfolds it together with the detector effects. In other words, the network distinguishes between jets from the hard process and jets from QCD radiation and it also unfolds the kinematic modifications from initial state radiation, to provide probability distributions in the parton-level phase space of a pre-defined hard process.

5.2 Unfolding basics

Unfolding particle physics events is a classic example for an inverse problem [160]. In the limit where detector effects can be described by Gaussian noise, it is similar to unblurring images. However, actual detector effects depend on the individual objects, the global energy deposition per event, and the proximity of objects, which means they are much more complicated than Gaussian noise. The situation gets more complicated when we add effects like QCD jet radiation, where the radiation pattern depends for instance on the quantum numbers of the incoming partons and on the energy scale of the hard process.
What we do know is that we can describe the measurement of phase space distributions $\mathrm{d}\sigma/\mathrm{d}x_\mathrm{d}$ as a random process, just as the detector effects or jet radiation can be simulated

by a set of random numbers describing a Markov process. This means that also the inversion or extraction of $\mathrm{d}\sigma/\mathrm{d}x_\mathrm{p}$ is a statistical problem.

5.2.1 Binned toy model and locality

As a one-dimensional toy example we can look at a binned (parton-level) distribution $\sigma_j^{(\mathrm{p})}$ which gets transformed into another binned (detector-level) distribution $\sigma_j^{(\mathrm{d})}$ by the kernel or response function g_{ij},

$$\sigma_i^{(\mathrm{d})} = g_{ij}\sigma_j^{(\mathrm{p})}. \tag{5.2}$$

We can postulate the existence of an inversion with the kernel $\bar{g}$ through the relation

$$\sigma_k^{(p)} = \sum_{i=1}^{N} \bar{g}_{ki}\sigma_i^{(d)} = \sum_{j=1}^{N}\left(\sum_{i=1}^{N}\bar{g}_{ki}g_{ij}\right)\sigma_j^{(p)} \quad \text{with} \quad \sum_{i=1}^{N}\bar{g}_{ki}g_{ij} = \delta_{kj}. \tag{5.3}$$

If we assume that we know the N^2 entries of the kernel g, this form gives us the N^2 conditions to compute its inverse $\bar{g}$. We illustrate this one-dimensional binned case with a semi-realistic smearing matrix

$$g = \begin{pmatrix} 1-\varepsilon & \varepsilon & 0 \\ \varepsilon & 1-2\varepsilon & \varepsilon \\ 0 & \varepsilon & 1-\varepsilon \end{pmatrix}, \tag{5.4}$$

where $\varepsilon \ll 1$ is a small migration or smearing parameter. We illustrate the smearing pattern with two input vectors, keeping in mind that in an unfolding problem we typically only have one kinematic distribution to determine the inverse matrix $\bar{g}$,

$$\begin{aligned} \sigma^{(p)} = n\begin{pmatrix}1\\1\\1\end{pmatrix} \quad &\Rightarrow \quad \sigma^{(d)} = \sigma^{(p)}, \\ \sigma^{(p)} = \begin{pmatrix}1\\n\\0\end{pmatrix} \quad &\Rightarrow \quad \sigma^{(d)} = \sigma^{(p)} + \varepsilon\begin{pmatrix}n-1\\-2n+1\\n\end{pmatrix}. \end{aligned} \tag{5.5}$$

The first example shows how for a symmetric smearing matrix a flat distribution removes all information about the detector effects. This implies that we might end up with a choice of reference process and phase space such that we cannot extract the detector effects from the available data. The second example illustrates that for bin migration from a dominant peak the information from the original $\sigma^{(p)}$ gets overwhelmed easily. We can also compute the inverse of the smearing matrix in Eq.(5.4) and find

$$\bar{g} \approx \frac{1}{1-4\varepsilon}\begin{pmatrix} 1-3\varepsilon & -\varepsilon & \varepsilon^2 \\ -\varepsilon & 1-2\varepsilon & -\varepsilon \\ \varepsilon^2 & -\varepsilon & 1-3\varepsilon \end{pmatrix}, \tag{5.6}$$

where we neglect the sub-leading ε^2-terms whenever there is a linear term as well. The unfolding matrix extends beyond the nearest neighbor bins, which means that local detector effects lead to a global unfolding matrix and unfolding only works well if we understand our entire data set. The reliance on useful kinematic distributions and the global dependence of the unfolding define the main challenges once we attempt to unfold the full phase space of an LHC process.

5.2.2 Bayes' theorem and model dependence

Over the continuous phase space a detector simulation can be written as

$$\frac{\mathrm{d}\sigma}{\mathrm{d}x_d} = \int \mathrm{d}x_p \; g(x_d, x_p) \; \frac{\mathrm{d}\sigma}{dx_p}, \tag{5.7}$$

where x_d is a kinematic variable at detector level, x_p the same variable at parton level, and g a kernel or transfer function which links these two arguments. We ignore efficiency factors for now, because they can be absorbed into the parton-level rate. To invert the detector simulation we define a second transfer function $\bar{g}$ such that [161–163]

$$\frac{\mathrm{d}\sigma}{\mathrm{d}x_p} = \int \mathrm{d}x_d \; \bar{g}(x_p, x_d) \; \frac{\mathrm{d}\sigma}{\mathrm{d}x_d} = \int \mathrm{d}x'_p \; \frac{\mathrm{d}\sigma}{\mathrm{d}x'_p} \int \mathrm{d}x_d \; \bar{g}(x_p, x_d) g(x_d, x'_p). \tag{5.8}$$

This inversion is fulfilled if we construct the inverse $\bar{g}$ of g defined by

$$\int \mathrm{d}x_d \; \bar{g}(x_p, x_d) g(x_d, x'_p) = \delta(x_p - x'_p), \tag{5.9}$$

all in complete analogy to the binned form above. The symmetric form of Eq. (5.7) and Eq. (5.8) indicates that g and $\bar{g}$ are both defined as distributions. In the g-direction we use Monte Carlo simulation and sample in x_p, while $\bar{g}$ needs to be sampled in $g(x_p)$ or x_d. In both directions this statistical nature implies that we should only attempt to unfold sufficiently large event samples.

The above definitions can be linked to Bayes' theorem if we identify the kernels with probabilities. We now look at $\bar{g}(x_d|x_p)$ in the slightly modified notation as the probability of observing x_d given the model prediction x_p and $g(x_p|x_d)$ gives the probability of the model x_p being true given the observation x_d [164, 165]. In this language Eq.(5.7) and (5.8) describe conditional probabilities, and we can write something analogous to Bayes' theorem,

$$\bar{g}(x_p|x_d) \; \frac{\mathrm{d}\sigma}{\mathrm{d}x_d} \sim g(x_d|x_p) \; \frac{\mathrm{d}\sigma}{\mathrm{d}x_p} \; . \tag{5.10}$$

In this form $\bar{g}(x_p|x_d)$ is the posterior, $g(x_d|x_p)$ as a function of x_p is the likelihood, $\mathrm{d}\sigma/\mathrm{d}x_p$ is the prior, and the model evidence $\mathrm{d}\sigma/\mathrm{d}x_d$ fixes the normalization of the posterior. From standard Bayesian analyses we know two things:

(i) the posterior will in general depend on the prior, in our case the kinematics of the underlying particle physics process or model.

(ii) when analyzing high-dimensional spaces the prior dependence will vanish when the likelihood develops a narrow global maximum.

If the posterior $\bar{g}(x_p|x_d)$ in general depends on the model $\mathrm{d}\sigma/\mathrm{d}x_p$, then Eq.(5.8) does not look useful. On the other hand, Bayesian statistics is based on the assumption that the prior dependence of the posterior defines an iterative process where we start from a very general prior and enter likelihood information step by step to finally converge on the posterior. The same approach can define a kinematic unfolding algorithm [166]. Will will not discuss these methods further, but come back to this model dependence throughout this chapter.

5.2.3 Reference process $pp \to ZW$

To provide a quantitative estimate of unfolding with neural networks we use the process

$$pp \to ZW^{\pm} \to (\ell^-\ell^+)\,(jj). \tag{5.11}$$

One of the contributing Feynman diagrams is shown in Fig. 5.1. With jets and leptons in the final state we can test the stability of the unfolding network for small and for large detector effects. We generate the ZW events using `Madgraph5` [23] without any generation cuts and then simulate parton showering with `Pythia8` [25] and the detector effects with `Delphes` [27] using the standard ATLAS card. For jet clustering we use the anti-k_T algorithm [167] with $R = 0.6$ implemented in `FastJet` [168]. All jets are required to have

$$p_{T,j} > 25 \text{ GeV} \qquad \text{and} \qquad |\eta_j| < 2.5\,. \tag{5.12}$$

For the hadronically decaying W-boson the limited calorimeter resolution will completely dominate over the parton-level Breit-Wigner distribution. After applying the cuts we have 320k events which we split into 90% training and 10% test data.

In a first step, we are only interested in inverting these detector effects. The results are shown in Sec. 5.3 and Sec. 5.4. For the simulation this implies that we switch off initial state radiation as well as underlying event and pile-up effects and require exactly two jets and a pair of same-flavor opposite-sign leptons. The jets and corresponding partons are separately ordered by p_T. The detector and parton level leptons are assigned by charge. This gives us two samples matched event by event, one at the parton level (x_p) and one including detector effects (x_d). Each of them is given as an unweighted set of four 4-vectors. These 4-vectors can be simplified if we assume all external particles at the parton level to be on-shell. Obviously, this method can be easily adapted to weighted events as we have already shown in Sec. 3.4.

In a second step we include initial state radiation and allow for additional jets in Sec. 5.5. We still require a pair of same-flavor opposite-sign leptons and at least two jets in agreement with the condition in Eq. (5.12). The four jets with highest p_T are then used as input to the network, ordered by p_T. Events with less than 4 jets are zero-padded. The second data set is only used for the conditional INN.

5.3 GAN unfolding detector effects

A standard method for fast detector simulation is smearing the outgoing particle momenta with a detector response function. This allows us to generate and sample from a

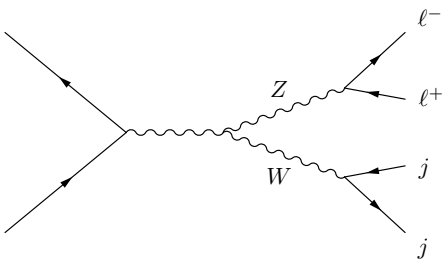

Figure 5.1: Sample Feynman diagram contributing to ZW production, with intermediate on-shell particles labelled.

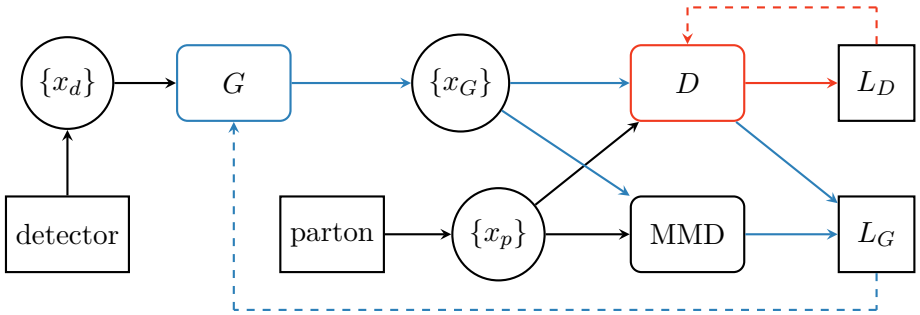

Figure 5.2: Structure of a naive unfolding GAN. The input $\{x_d\}$ describes a batch of events sampled at detector level and $\{x_{G,p}\}$ denotes events sampled from the generator or parton-level data. The blue (red) arrows indicate which connections are used in the training of the generator (discriminator).

probability distribution of smeared final-state momenta for a single parton-level event. For the inversion we need to rely on event samples, as we can see from a simple example: we start from a sharp Z-peak at the parton level and broaden it with detector effects. Now we look at a detector-level event in the tail and invert the detector simulations, for which we need to know in which direction in the invariant mass the preferred value m_Z lies. This implies that unfolding detector effects requires a model hypothesis, which can be thought of as a condition in a probability of the inversion from the detector level. The problem with this perspective is that the parton-level distribution of the invariant mass requires a dynamic reconstruction of the Breit-Wigner peak, which is not easily combined with a Markov process. From this argument it is clear that unfolding only makes sense at the level of large enough event samples.

From Sec. 1.3.2 we know how to set up a GAN to either generate detector-level events from parton-level events or vice versa. In our current setup the events are an unweighted set of four 4-vectors, two jets and two leptons. The final-state masses are fixed to the parton-level values. Our hadronic final state is defined at the level of jet 4-vectors. This does not mean that in a possible application we take a parton shower at face value. All we do is assume that there is a correspondence between a hard parton and its hadronic final state, and that the parton 4-momentum can be reconstructed with the help of a jet algorithm. The question if for instance an anti-k_T algorithm is an appropriate description of sub-jet physics does not arise as long as the jet algorithm reproduces the hard parton momentum.

5.3.1 Naive GAN

Following the notations in Sec. 2.3, our GAN comprises a generator network G_θ competing against a discriminator network D_φ in a min-max game, as illustrated in Fig. 5.2. For the implementation we use `Keras` (v2.2.4) [136] with a `TensorFlow` (v1.14) backend [137]. As the starting point, G_θ is randomly initialized to produce an output, typically with the same dimensionality as the target space. It induces a probability distribution $p_\theta(x)$ of a target space element x, in our case a parton-level event. To be precise, the generator obtains a batch of detector level events as input and generates a batch of parton level events as output, *i.e.* $G_\theta(\{x_d\}) = \{x_G\}$. The discriminator is given batches $\{x_G\}$ and $\{x_p\}$ sampled from p_θ and the parton-level target distribution p_p. It is trained as a binary classifier, such that $D_\varphi(x \in \{x_p\}) = 1$ and $D_\varphi(x) = 0$ otherwise. Following the

conventions of Sec. 2.3 the loss functions are given by

$$L_D = \mathbb{E}_{x\sim p_p}[-\log D_\varphi(x)] + \mathbb{E}_{x\sim p_\theta}[-\log(1 - D_\varphi(x))], \tag{5.13}$$
$$L_G = \mathbb{E}_{x\sim p_\theta}\left[-\log D_\varphi(x)\right]. \tag{5.14}$$

We again add gradient penalty term [117] to the discriminator to avoid instabilities in the training [117]:

$$L_D^{(\text{GP})} = L_D + \lambda_D \, \mathbb{E}_{x\sim p_p}\left[(1 - D_\varphi(x))^2 \, |\nabla d_\varphi(x)|^2\right] \tag{5.15}$$
$$+ \lambda_D \, \mathbb{E}_{x\sim p_\theta}\left[D_\varphi(x)^2 \, |\nabla d_\varphi(x)|^2\right], \tag{5.16}$$

with a properly chosen pre-factor λ_D and where we define

$$d_\varphi(x) = \log\left(\frac{D_\varphi(x)}{1 - D_\varphi(x)}\right). \tag{5.17}$$

If the training of the generator and the discriminator with their respective losses Eq. (5.16) and Eq. (5.14) is properly balanced, the distribution p_θ converges to the parton-level distribution p_p, while the optimized discriminator is unable to distinguish between real and generated samples.

If we want to describe phase space features, for instance at the LHC, it is useful to add a MMD [61] contribution to the loss function as we have already shown in Sec. 3.2. It allows us to compare pre-defined distributions, for instance the one-dimensional invariant mass of an intermediate particle. We add this MMD to the generator loss

$$L_G^{(\text{MMD})} = L_G + \lambda_G \, \text{MMD}, \tag{5.18}$$

with another properly chosen variable λ_G. Note that we now use MMD instead of MMD^2 to enhance the sensitivity of the model [130]. As a naive approach to GAN unfolding we use detector-level event samples as generator input. The network input is always a set of four 4-vectors, one for each particle in the final state, with their masses fixed [1]. In the GAN setup we train our network to map detector-level events to parton-level events. Both networks consist of 12 layers with 512 units per layer. With $\lambda_G = 1$, $\lambda_D = 10^{-3}$ and a batch size of 512 events, we run for 1200 epochs and 500 iterations per epoch.

In Fig. 5.3 we compare true parton-level events to the output from a GAN trained to unfold the detector effects. We run the unfolding GAN on a set of statistically independent, but simulation-wise identical sets of detector-level events. Both, the relatively flat p_{T,j_1} and the peaked m_{jj} distributions agree well between the true parton-level events and the GAN-inverted sample, indicating that the statistical inversion of the detector effect works well.

A great advantage of this GAN approach is that, strictly speaking, we do not need event-by-event matched samples before and after detector simulation. The entire training is based on batches of typically 512 events, and these batches are independently chosen from the parton-level and detector-level samples. Increasing the batch size within the range allowed by the memory size and hence reducing the impact of event-wise matching will actually improve the GAN training, because it reduces statistical uncertainties [1].

The big challenge arises when we want to unfold an event sample which is not statistically equivalent to the training data; in other words, the unfolding model is not exactly the same as the test data. As a simple example we train the GAN on data covering the

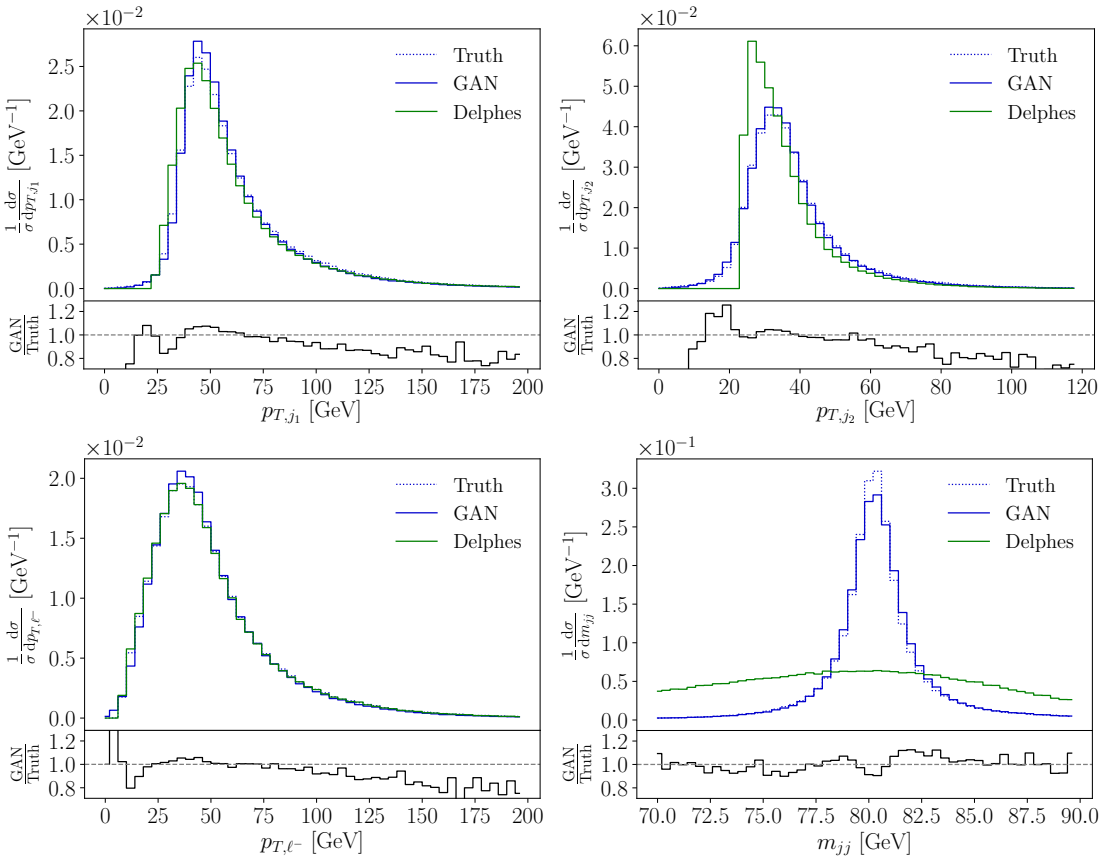

Figure 5.3: Example distributions for parton level truth, after detector simulation, and GANned back to parton level. The lower panels give the ratio of parton level truth and reconstructed parton level.

full phase space and then apply and test the GAN on data only covering part of the detector-level phase space. Specifically, we apply the two sets of jet cuts

$$\text{Cut I}: \quad p_{T,j_1} = 30 \ldots 100 \text{ GeV}, \tag{7}$$

$$\text{Cut II}: \quad p_{T,j_1} = 30 \ldots 60 \text{ GeV} \quad \text{and} \quad p_{T,j_2} = 30 \ldots 50 \text{ GeV}, \tag{8}$$

which leave us with 88% and 38% of events, respectively. This approach ensures that the training has access to the full information, while the test sample is a significantly reduced sub-set of the full sample.
In Fig. 5.4 we show a set of kinematic distributions, for which we GAN only part of the phase space. As before, we can compare the original parton-level shapes of the distributions with the results from GAN-inverting the fast detector simulation. We see that especially the GANned $p_{T,j}$ distribution is strongly sculpted by the phase space cuts. This indicates that the naive GAN approach to unfolding does not work once the training and test data sets are not statistically identical. In a realistic unfolding problem we cannot expect the training and test data sets to be arbitrarily similar, so we have to go beyond the naive GAN setup described in Fig. 5.2. The technical reason for this behavior is that events which are similar or, by some metric, close on detector level are not guaranteed to be mapped onto events which are also close on parton level. Looking at classification networks this is the motivation to apply variational methods, for instance upgrade autoencoders to variational autoencoders. For a GAN we discuss a standard solution in the next section.

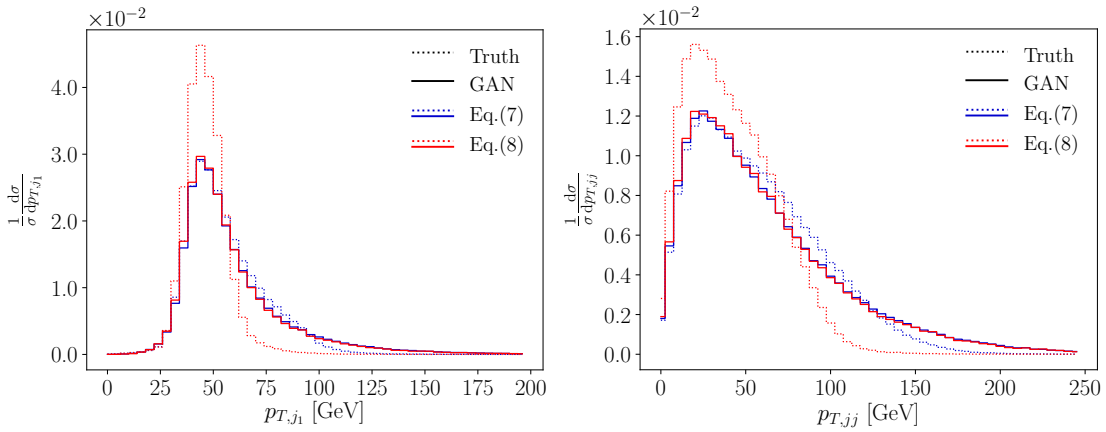

Figure 5.4: Parton level truth and GANned distributions when we train the GAN on the full data set but only unfold parts of phase space defined in Eq.(7) and Eq.(8).

5.3.2 Conditional GAN

The way out of the sculpting problem when looking at different phase space regions is to add a conditional structure to the GAN [71] shown in Fig. 5.2. The idea behind the conditional setup is not to learn a deterministic link between input and output samples, because we know that without an enforced structure in the weight or function space the generator does not benefit from the structured input. In other words, the network does not properly exploit the fact that the detector-level and parton-level data sets in the training sample are paired. A second, related problem of the naive GAN is that once trained the model is completely deterministic, so each detector-level event will always be mapped to the same parton-level events. This goes against the physical intuition that this entire mapping is statistical in nature.

In Fig. 5.5 we introduce a fully conditional GAN (FCGAN). It is identical to our naive network the way we train and use the generator and discriminator. However, the input to the generator are actual random numbers $\{z\}$, and the detector-level information $\{x_d\}$ is used as an event-by-event conditional input on the link between a set of random numbers and the parton-level output, *i.e.* $G_\theta(\{z\},\{x_d\}) = \{x_G\}$ and hence $p_\theta(x_G) \equiv p_\theta(x_G(z,x_d))$. This way the FCGAN can generate parton-level events from random noise but still using the detector-level information as input. To also condition the discriminator we modify its loss to

$$L_D \to L_D^{(\text{FC})} = \mathbb{E}_{x\sim p_p, y\sim p_d}[-\log D_\varphi(x,y)] + \mathbb{E}_{x\sim p_\theta, y\sim p_d}[-\log(1-D_\varphi(x,y))], \tag{5.19}$$

and the regularized loss function changes accordingly,

$$\begin{aligned} L_D^{(\text{GP})} \to L_D^{(\text{GP,FC})} = L_D^{(\text{FC})} &+ \lambda_D \, \mathbb{E}_{x\sim p_p, y\sim p_d} \left[(1-D_\varphi(x,y))^2 \, |\nabla d_\varphi(x,y)|^2\right] \\ &+ \lambda_D \, \mathbb{E}_{x\sim p_\theta, y\sim p_d} \left[D_\varphi(x,y)^2 \, |\nabla d_\varphi(x,y)|^2\right], \end{aligned} \tag{5.20}$$

The generator loss function now takes the form

$$L_G \to L_G^{(\text{FC})} = \mathbb{E}_{x\sim p_\theta, y\sim p_d} \left[-\log D_\varphi(x,y)\right]. \tag{5.21}$$

Note, that we do not build a conditional version of the MMD loss. The hyper-parameters of our FCGAN are summarized in Tab. 4.2. Changing from a naive GAN to a fully

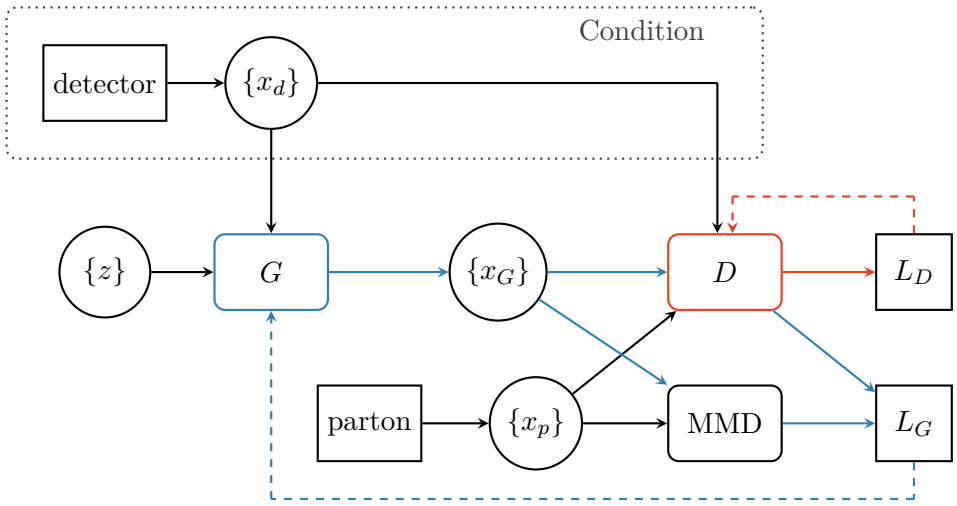

Figure 5.5: Structure of our fully conditional FCGAN. The input $\{z\}$ describes a batch of random numbers and $\{x_{G,d,p}\}$ denotes events sampled from the generator, detector-level data, or parton-level data. The blue (red) arrows indicate which connections are used in the training of the generator (discriminator).

conditional GAN we have to pay a price in the structure of the training sample. While the naive GAN only required event batches to be matched between parton level and detector level, the training of the FCGAN actually requires event-by-event matching.

In Fig. 5.6 we compare the truth and the FCGANned events, trained on and applied to events covering the full phase space. Compared to the naive GAN, inverting the detector effects now works even better. The systematic under-estimate of the GAN rate in the tails no longer occurs for the FCGAN. The reconstructed invariant W-mass forces the network to dynamically generate a very narrow physical width from a comparably broad Gaussian peak. Using our usual MMD loss developed in Chapter 3 and Ref. [1] we reproduce the peak position, width, and peak shape to about 90%. We emphasize that the MMD loss requires us to specify the relevant one-dimensional distribution, in this case m_{jj}, but it then extracts the on-shell mass or width dynamically.

As for our naive ansatz we now test what happens to the network when the training data and the test data do not cover the same phase space region. We train on the full set of events, to ensure that the full phase space information is accessible to the network, but we then only apply the network to the 88% and 38% of events passing the jet cuts I and II defined in Eq.(7) and Eq.(8). We show the results in Fig. 5.7. As

Parameter	Value	Parameter	Value
Layers	12	Batch size	512
Units per layer	512	Epochs	1200
Trainable weights G	3M	Iterations per epoch	500
Trainable weights D	3M	Number of training events	3×10^5
λ_G	1		
λ_D	10^{-3}		

Table 5.1: FCGAN setup.

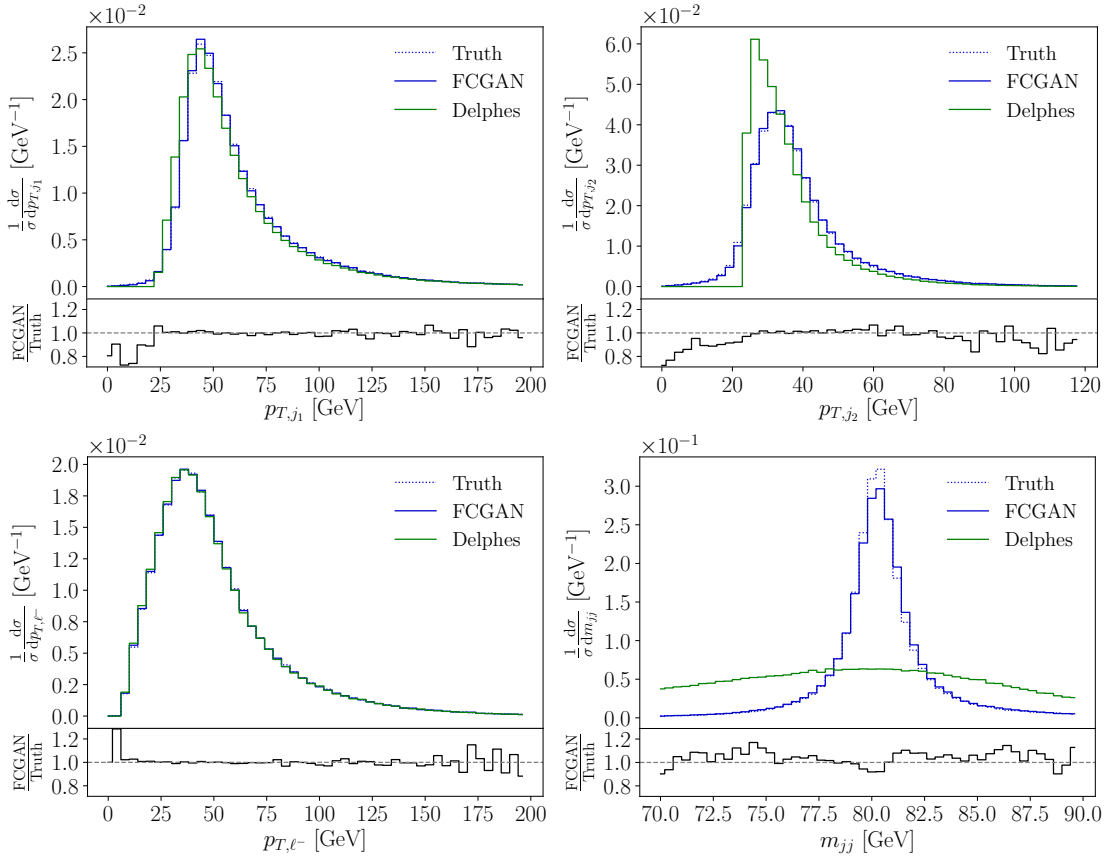

Figure 5.6: Example distributions for parton level truth, after detector simulation, and FCGANned back to parton level. The lower panels give the ratio of parton level truth and reconstructed parton level. To be compared with the naive GAN results in Fig. 5.3.

observed before, especially the jet cuts with only 40% survival probability shape our four example distributions. However, we see for example in the $p_{T,jj}$ distribution that the inverted detector-level sample reconstructs the patterns of the true parton-level events perfectly. This comparison indicates that the FCGAN approach deals with differences in the training and test samples very well.
Because we want to push our FCGAN to its limit we move on to harsher cuts on the inclusive event sample. We start with

$$\text{Cut III}: \quad p_{T,j_1} = 30 \ldots 50 \text{ GeV} \quad p_{T,j_2} = 30 \ldots 40 \text{ GeV} \quad p_{T,\ell^-} = 20 \ldots 50 \text{ GeV}, \tag{12}$$

which 14% of all events pass. In Fig. 5.8 we see that also for this reduced fraction of test events, the FCGAN inversion reproduces the true distributions extremely well, to a level where it appears not really relevant what fraction of the training and test data correspond to each other.

Finally, we apply a cut which not only removes a large fraction of events, but also cuts into the leading peak feature of the p_{T,j_1} distribution and removes one of the side bands needed for an interpolation,

$$\text{Cut IV}: \quad p_{T,j_1} > 60 \text{ GeV} . \tag{13}$$

For this choice 39% of all events pass, but we remove all events at low transverse momentum, as can be seen from Fig. 5.6. This kind of cut could therefore be expected to

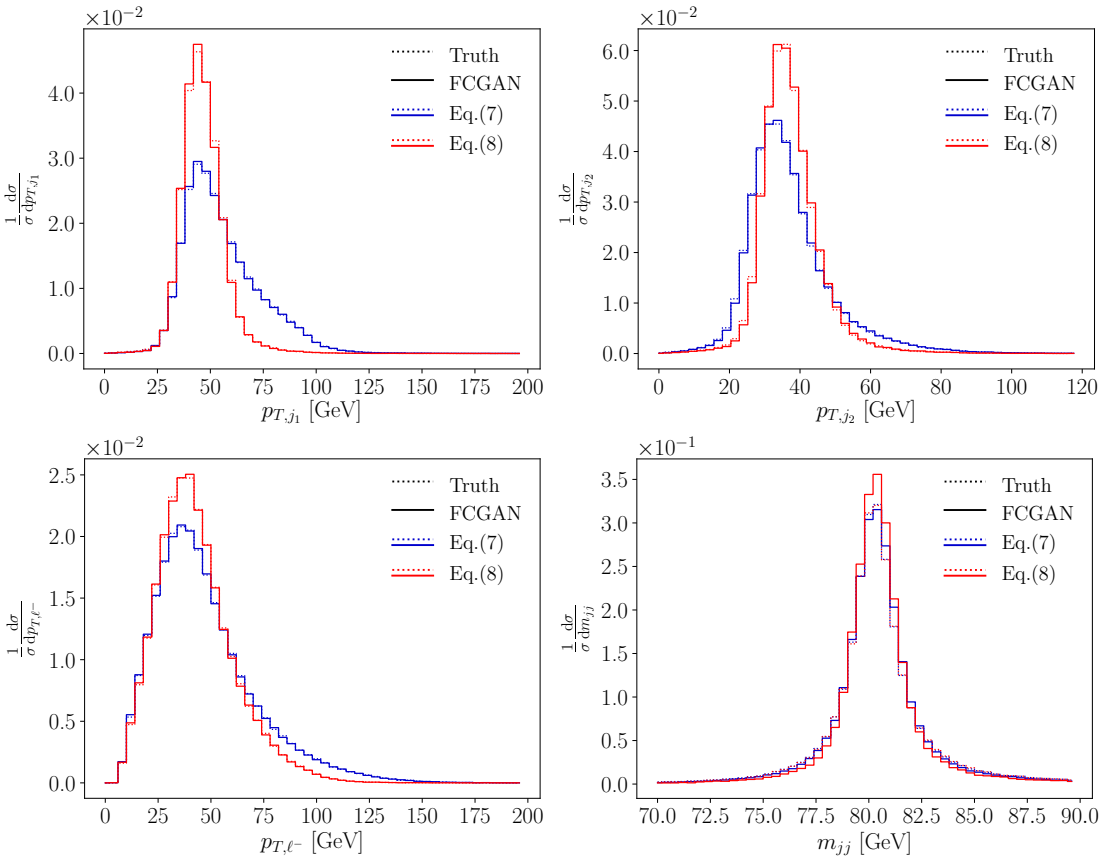

Figure 5.7: Parton level truth and FCGANned distributions when we train the GAN on the full data set but only unfold parts of phase space defined in Eq.(7) and Eq.(8). To be compared with the naive GAN results in Fig.5.4.

break the unfolding. Indeed, the red lines in Fig. 5.8 indicate that we have broken the m_{jj} reconstruction through the FCGAN. However, all other (shown) distributions still agree with the parton-level truth extremely well. The problem with the invariant mass distribution is that our implementation of the MMD loss is not actually conditional. This can be changed in principle, but standard implementations are fairly inefficient and the benefit is not clear at the moment. At this stage it means that, when pushed towards its limits, the network will first fail to reproduce the correct peak width in the m_{jj} distribution, while all other kinematic variables remain stable.

Finally, just like in Sec. 3.2 we show 2-dimensional correlations in Fig. 5.9. We stick to applying the network to the full phase space and show the parton level truth and the FCGAN-inverted events in the two upper panels. Again, we see that the FCGAN reproduces all features of the parton level truth with high precision. The bin-wise relative deviation between the two 2-dimensional distributions only becomes large for small values of E_{j_1}, where the number of training events is extremely small.

5.3.3 New physics injection

As discussed before, unfolding to a hard process is necessarily model-dependent. Until now, we have always assumed the Standard Model to correctly describe the parton-level and detector-level events. An obvious question is what happens if we train our FCGAN on Standard Model data, but apply it to a different hypothesis. This challenge becomes

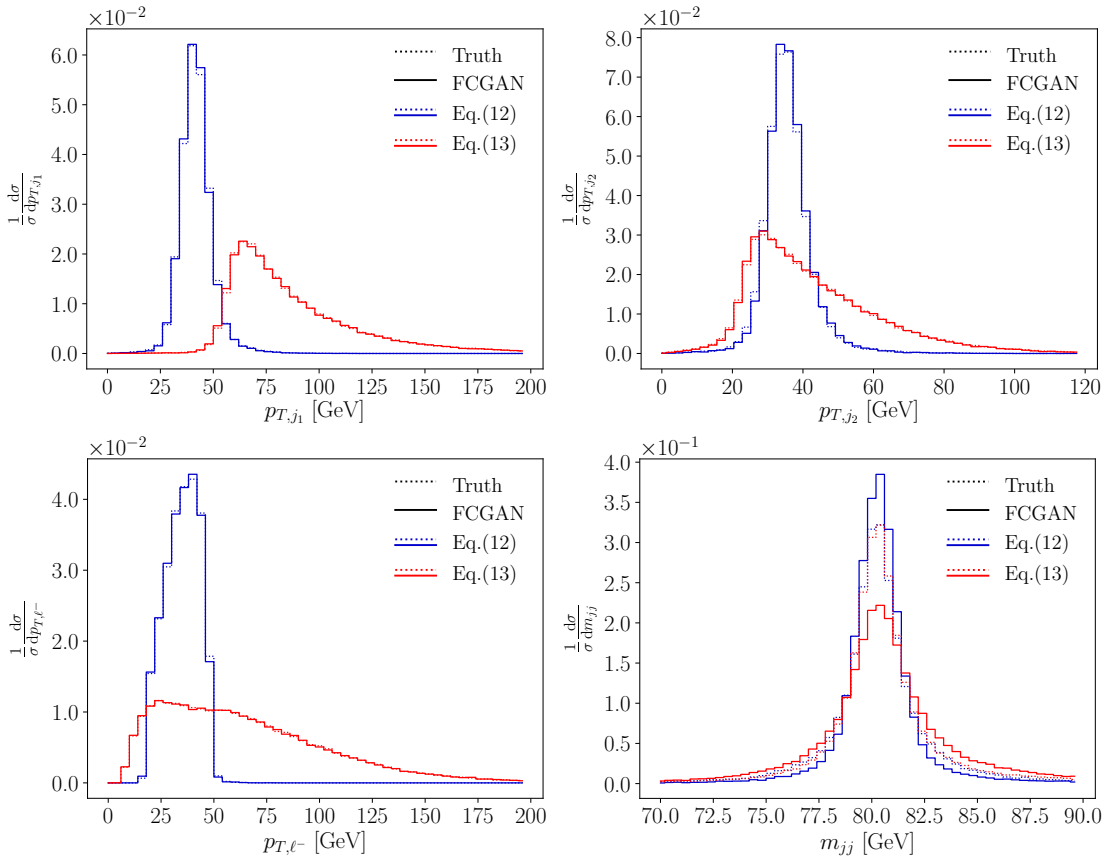

Figure 5.8: Parton level truth and FCGANned distributions when we train the GAN on the full data set but only unfold parts of phase space defined in Eqs.(12) and (13).

especially interesting if this alternative hypothesis differs from the Standard Model in a local phase space effect. It then allows us to test if the generator network maps the parton-level and detector-level phase spaces in a structured manner. Such features of neural networks are at the heart of all variational constructions, for instance variational autoencoders which are structurally close to GANs. Observing them for GAN unfolding could turn into a significant advantage over alternative unfolding methods.

Therefore, we add a fraction of resonant W' events from a triplet extension of the Standard Model [169], representing the hard process

$$pp \to W'^* \to ZW^\pm \to (\ell^-\ell^+)\,(jj) \tag{5.22}$$

to the test data. We simulate these events with `Madgraph5` using the model implementation of Ref. [170] and denote the new massive charged vector boson with a mass of 1.3 TeV and a width of 15 GeV as W'. For the test sample we combine the usual Standard Model sample with the W'-sample in proportions $90\% - 10\%$. The other new particles do not appear in our process to leading order. Because we want to test how well the GAN maps local phase space structures onto each other, we deliberately choose a small width $\Gamma_{W'}/M_{W'} \sim 1\%$, not exactly typical for such strongly interacting triplet extensions.

The results for this test are shown in Fig. 5.10. First, we look at transverse momentum distribution of final-state particles, which are hardly affected by the new heavy

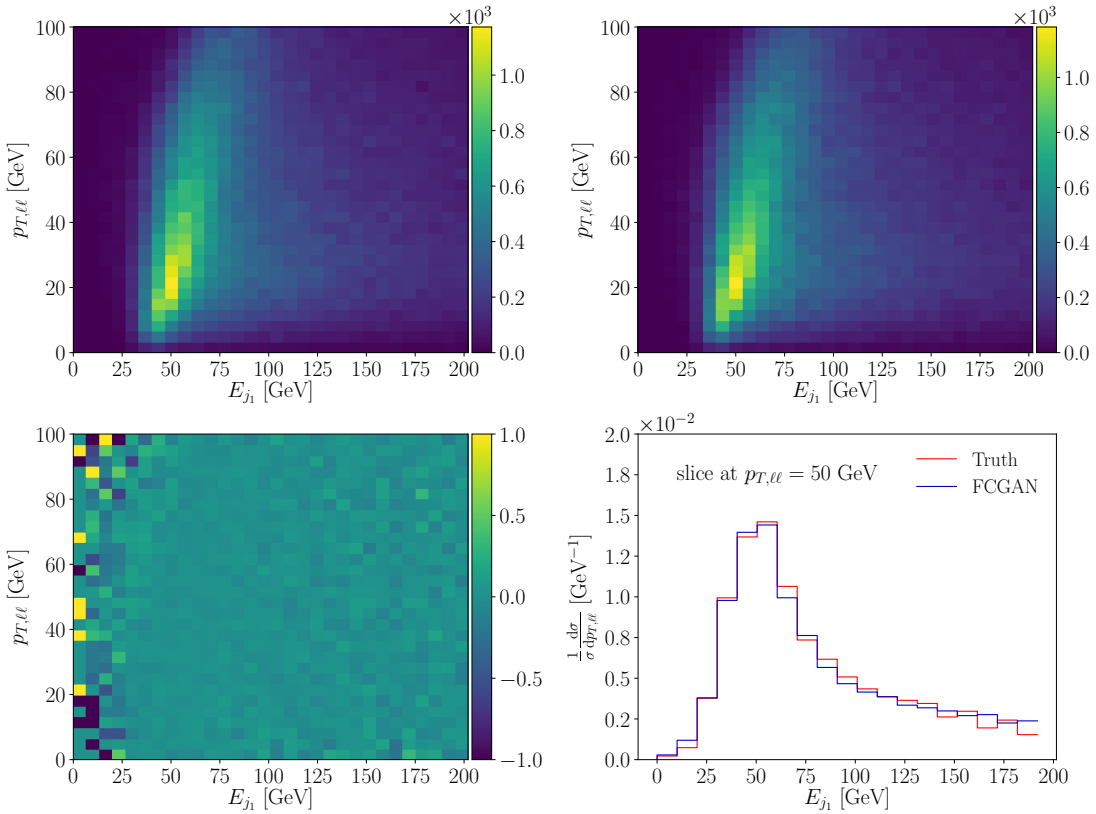

Figure 5.9: Two-dimensional parton level truth (upper left) and FCGANned (upper right) distributions when we train the GAN on the full data set and unfold over the full phase space. The lower panels show the relative deviation between truth and FCGANned and the one-dimensional E_{j_1} distribution along fixed $p_{T,\ell\ell}$.

resonance. Both, the leading jet and the lepton distributions are essentially identical for both truth levels and the FCGAN output. The same is true for the invariant mass of the hadronically decaying W-boson, which nevertheless provides a useful test of the stability of our training and testing.

Finally, we show the reconstructed W'-mass in the lower-right panel. Here we see the different (normalized) truth-level distributions for the Standard Model and the W'-injected sample. The FCGAN, trained on the Standard Model, keeps track of local phase space structures and reproduces the W' peak faithfully. It also learn the W'-mass as the central peak position very well. The only issue is the W'-width, which the network overestimates. However, we know already that dynamically generated width distributions are a challenge to GANs and require for instance an MMD loss. Nevertheless, Fig. 5.10 clearly shows that GAN unfolding shows a high degree of model independence, making use of local structures in the mapping between the two phase spaces. We emphasize that the additional mass peak in the FCGANned events is not a one-dimensional feature, but a localized structure in the full phase space. This local structure is a feature of neural networks which comes in addition to the known strengths in interpolation.

5.4 INN unfolding detector effects

So far we have shown how a GAN and more spefically a conditional GAN can be used to invert detector effects. As an alternative to our FCGAN we now study the usage of a conditional INN (cINN) for the same purpose. We introduce the cINN in two steps, starting with the non-conditional, standard setup. The construction of the INN we use in our analysis combines two goals [43]:

1. the mapping from input to output is invertible and the Jacobians for both directions are tractable;

2. both directions can be evaluated efficiently. This second property goes beyond some other implementations of normalizing flow [122, 124].

While the final aim is not actually to evaluate our INN in both directions, we will see that these networks can be extremely useful to invert a Markov process like detector smearing. Their bi-directional training makes them especially stable. Moreover, in Sec. 5.4.3 we will show how especially the conditional INN retains a proper statistical notion of such an inversion. For the implementation we use `Pytorch` (v1.2.0) [171].

5.4.1 Naive INN

While it is clear from our discussion in the previous section and Ref. [2] that a standard INN will not serve our purpose, we still describe it in some detail before we extend

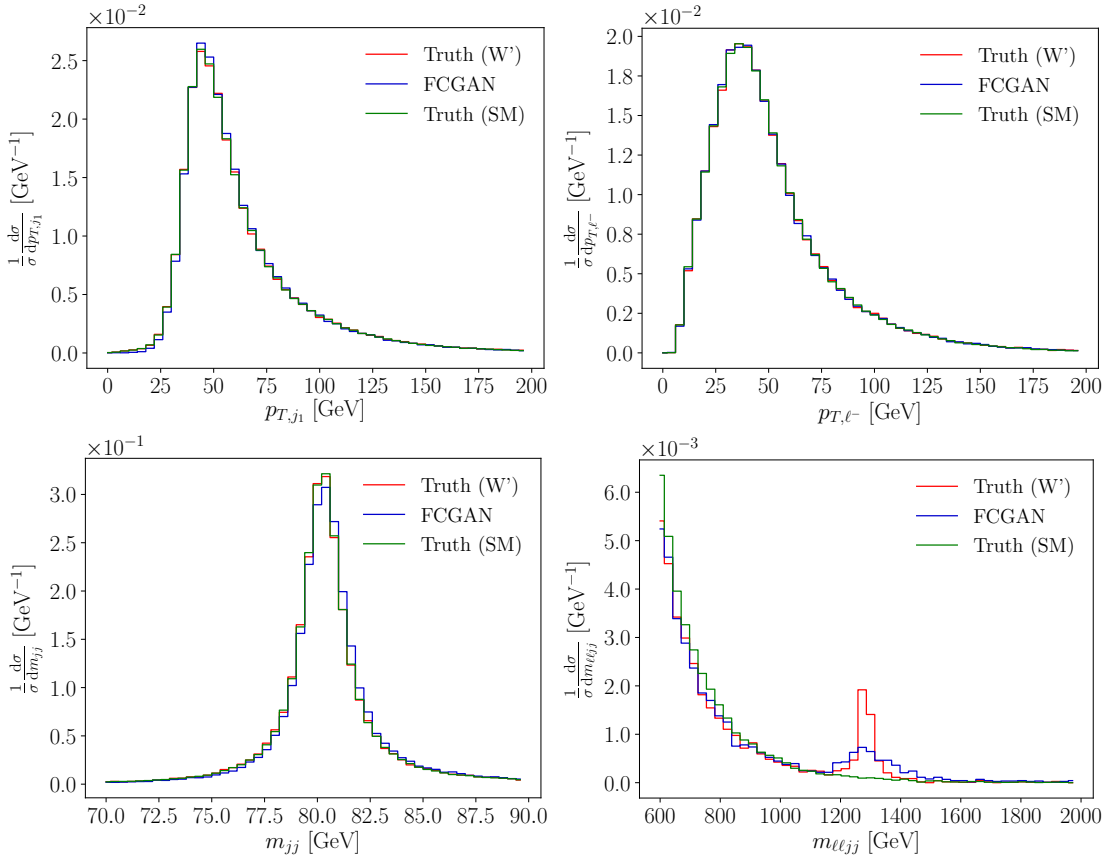

Figure 5.10: Parton level truth and FCGANned distributions when we train the network on the Standard Model only and unfold events with an injection of 10% W' events. The mass of the additional s-channel resonance is 1.3 TeV.

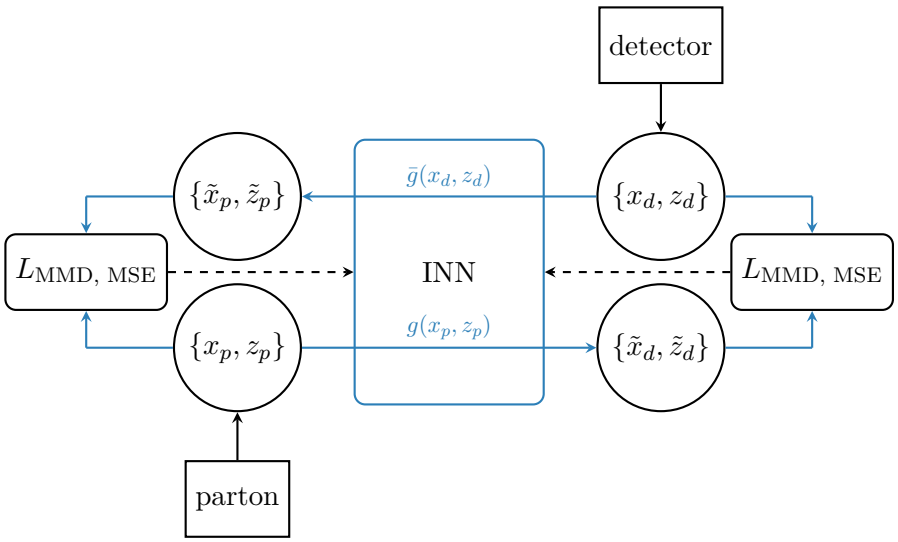

Figure 5.11: Structure of INN. The $\{x_{d,p}\}$ denote detector-level and parton-level events, $\{z_{d,p}\}$ are random numbers to match the phase space dimensionality. A tilde indicates the INN generation.

it to a conditional network. Following the conventions of our GAN analysis and in analogy to Eqs. (5.7) to (5.9) we define the network input as a vector of hard process information $x_p \in R^{D_p}$ and the output at detector level via the vector $x_d \in \mathbb{R}^{D_d}$. If the dimensionalities of the spaces are such that $D_p < D_d$ we add a noise vector z with dimension $D_d - D_p$ to define the bijective, invertible transformation,

$$\begin{pmatrix} x_p \\ z \end{pmatrix} \xleftrightarrow[\leftarrow \text{ unfolding:} \bar{g}]{\text{Pythia,Delphes:} g \rightarrow} x_d. \tag{5.23}$$

A correctly trained network g with the parameters θ then reproduces x_d from the combination x_p and z. Its inverse $\bar{g}$ instead reproduces the combination of x_p and z from x_d.

The defining feature of the INN illustrated in Fig. 5.11 is that it learns both directions of the bijective mapping in parallel and encodes them into one network. Such a simultaneous training of both directions is guaranteed by the building blocks of the network, the invertible coupling layers [44, 127] as introduced in Sec. 2.4.1. For notational purposes we ignore the random numbers in Eq.(5.23) and assume that this layer links an input vector x_p to an output vector x_d after splitting both of them in halves, $x_{p,i}$ and $x_{d,i}$ for $i = 1, 2$. The relation between input and output is given by a sub-network, which encodes arbitrary functions $s_{1,2}$ and $t_{1,2}$. Following the conventions Eq.(2.32) and Eq.(2.33) we can parametrize the mappings of g and $\tilde{g}$ as

$$g: \quad \begin{pmatrix} x_{d,1} \\ x_{d,2} \end{pmatrix} = \begin{pmatrix} x_{p,1} \odot e^{s_2(x_{p,2})} + t_2(x_{p,2}) \\ x_{p,2} \odot e^{s_1(x_{d,1})} + t_1(x_{d,1}) \end{pmatrix}, \tag{5.24}$$

$$\bar{g}: \quad \begin{pmatrix} x_{p,1} \\ x_{p,2} \end{pmatrix} = \begin{pmatrix} (x_{d,1} - t_2(x_{p,2})) \odot e^{-s_2(x_{p,2})} \\ (x_{d,2} - t_1(x_{d,1})) \odot e^{-s_1(x_{d,1})} \end{pmatrix}. \tag{5.25}$$

Again, this inversion works independent of the form of s and t. By construction, the Jacobian of the network function is tractable and hence its determinant is easy to compute,

as shown in Eq.(2.37). Such coupling layer transformations define a so-called normalizing flow, when we view it as transforming an initial probability density into a very general form of probability density through a series of invertible steps. We can relate the two probability densities as long as the Jacobians of the individual layers can be efficiently calculated.

Since the first use of the invertible coupling layer, much effort has gone into improving its efficiency. The All-in-One (AIO) coupling layer includes two features, introduced by Ref. [44] and Ref. [45]. The first modification replaces the transformation of $x_{p,2}$ by a permutation of the output of each layer. Due to the permutation each component still gets modified after passing through several layers. The second modification includes a global affine transformation to include a global bias and linear scaling that maps $x \to sx + b$. Finally, we apply a bijective soft clamping after the exponential function in Eq.(5.25) to prevent instabilities from diverging outputs.

The INN in our simplified example combines three contributions to the loss function. First, it tests if in the `Delphes` direction of Eq.(5.23) we indeed find $g(x_p) = x_d$ via the mean squared error (MSE) function. While this is theoretically sufficient to obtain the inverse function, also testing the inverse direction $\bar{g}(x_d) = x_p$ greatly improves the efficiency and stability of the training. Third, to resolve special sharp features like the invariant mass of intermediate particles we again use the MMD as introduced in Sec. 3.2.2.

In Sec. 3.2 and Sec. 5.3.2 we already compared common kernel choices, like Gaussian or Breit-Wigner kernels

$$k_{\text{Gauss}}(x,y) = \exp \frac{-(x-y)^2}{2\sigma^2} \qquad \text{or} \qquad k_{\text{BW}}(x,y) = \frac{\sigma^2}{(x-y)^2 + \sigma^2} \tag{5.26}$$

with a fixed or variable width σ [2]. Inside the INN architecture the Breit-Wigner kernel is the best choice to analyze the distribution of the random numbers as part of the loss function [43].

We now use the INN network to map parton-level events to detector-level events or vice-versa. In a statistical analysis we then use standard kinematic distributions and compare the respective truth and INN-inverted shapes for both directions. The left panels of Fig. 5.12 shows the transverse momentum distributions of the two jets and their invariant mass for both directions of the INN. The truth events at parton level and at detector level are marked as dashed lines. Starting from each of the truth events we can apply the INN describing the detector effects as $x_d = g(x_p)$ or unfolding the detector effects as $x_p = \bar{g}(x_d)$ in Eq.(5.23). The corresponding solid lines have to be compared to the dotted truth lines, where we need to keep in mind that at the parton level the relevant objects are quarks while at the detector level they are jets.

For the leading jet the truth and INNed detector-level agree very well, while for the second jet the naive INN fails to capture the hard cut imposed by the jet definition. For the invariant mass we find that the smearing due to the detector effects is reproduced well with some small deviations in the tails. In the unfolding direction both p_T distributions follow the parton level truth. The only difference is a systematic lack of events in the tail for the second quark. This is especially visible in the ratio of the INN-unfolded events and the parton-level truth, indicating that also at small p_T the network does not fill the phase space sufficiently. Combining both directions we see that in forward direction the INN produces a too broad p_T-distribution, the unfolding direction of the INN produces a too narrow distribution. The conceptual advantage of the INN actually implies a

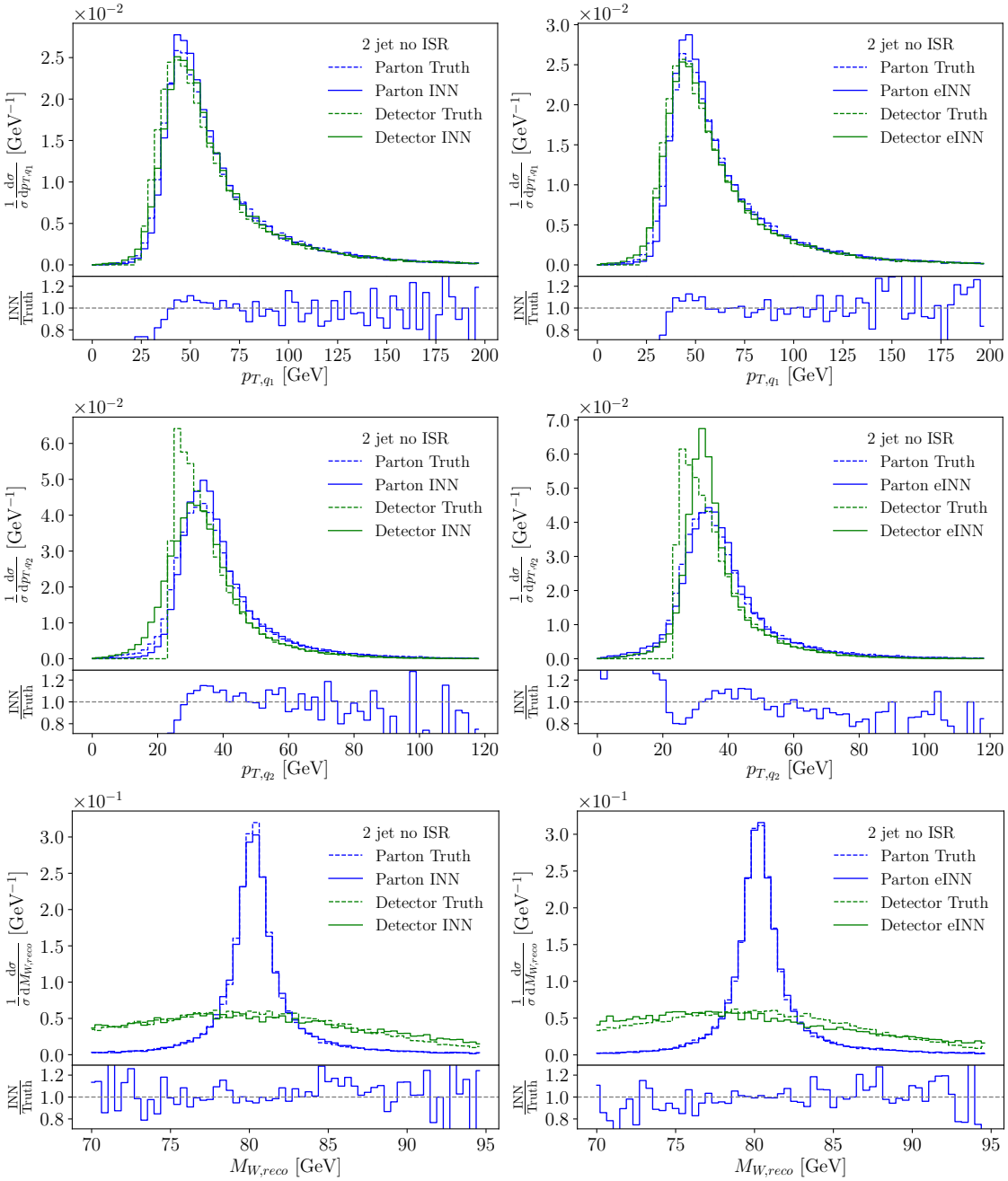

Figure 5.12: INNed $p_{T,q}$ and $M_{W,\text{reco}}$ distributions from a naive INN (left) and the noise-extended eINN (right). In green we compare the detector-level truth to INNed events transformed from parton level. In blue we compare the parton-level truth to INNed events transformed from detector level. The lower panels show the ratio of INNed events over parton-level truth.

disadvantage for the inversion of particular difficult features. Finally, the invariant mass of the W is reproduced perfectly without any systematic deviation.

5.4.2 Noise-extended INN

While our simplified example in the previous section shows some serious promise of INNs, it fails to incorporate key aspects of the physical process. First of all, the number

of degrees of freedom is not actually the same at parton level and at detector level. External partons are on their mass shell, while jets come with a range of jet masses. This mismatch becomes crucial when we include missing transverse momentum in the signature. We generally need fewer parameters to describe the partonic scattering than the detector-level process. For a fixed set of parton-level momenta we usually smear each momentum component to simulate the detector measurement. These additional degrees of freedom are of stochastic nature, so adding Gaussian random variable on the parton side of the INN could be a first step to address this problem.
To also account for potentially unobservable degrees of freedom at the parton level we extend each side of the INN by a random number vector. The mapping in Eq.(5.23) now includes two random number vectors with dimensions $D_{z_d} = D_p$ and $D_{z_p} = D_d$,

$$\begin{pmatrix} x_p \\ z_p \end{pmatrix} \xleftarrow[\leftarrow \text{ unfolding:}\bar{g}]{\text{PYTHIA,DELPHES:}g\rightarrow} \begin{pmatrix} x_d \\ z_d \end{pmatrix} . \tag{5.27}$$

In addition, a pure MSE loss can not capture the fact that the additional noise generates a distribution of detector-level events given fixed parton momenta. It would just predict a mean value of this distribution and minimize the effect of the noise. A better solution is an MMD loss for each degree of freedom in the event and the masses of intermediate particles, as well as the Gaussian random variables. On the side of the random numbers this MMD loss ensures that they really only encode noise. Again it is beneficial for the training to use the inverse direction and apply additional MMD losses to the parton level events as well as the corresponding Gaussian inputs. Finally we add a weak MSE loss on the four vectors of each side to stabilize the training.

In the right panels of Fig. 5.12 we show results for this noise-extended INN (eINN). The generated distributions are similar to the naive INN case and match the truth at the parton level. A notable difference appears in the second jet, the weak spot of the naive INN. The additional random numbers and MMDs provide more freedom to generate the peak in the forward direction and also improve the unfolding in the low-p_T and high-p_T regimes.

Aside from the better modeling, the noise extension allows for a statistic interpretation of the generated distributions and a test of the integrity of the INN-inverted distributions. In the left panel of Fig. 5.13 we illustrate the goal of the statistical treatment: we start from a single event at the detector level and generate a set of unfolded events. For each of them we evaluate for instance p_{T,q_1}. Already in this illustration we see that the GAN output is lacking a statistical behavior at the level of individual events, while the noise-extended eINN returns a reasonable distribution of unfolded events.

To see if the width of the noise-extended eINN output is correct we take 1500 parton-level and detector-level event pairs and unfold each event 60 times, sampling over the random variables. This gives us 1500 combinations like the one shown in the left panel of Fig. 5.13: a single parton-level truth configuration and a distribution of the eINNed configuration. To see if the central value and the width of the eINNed distribution can be interpreted statistically as a posterior probability distribution in parton phase space we analyze where the truth lies within the eINN distribution for each of the 1500 events. For a correctly calibrated curve we start for instance from the left of the kinematic distribution and expect 10% of the 1500 events in the 10% quantile of the respective probability distribution, 20% of events in the 20% quantile, etc. The corresponding calibration curves for the noise-extended eINN are shown in the right panel of Fig. 5.13. While they indicate that we can attempt a statistical interpretation of the INN unfolding,

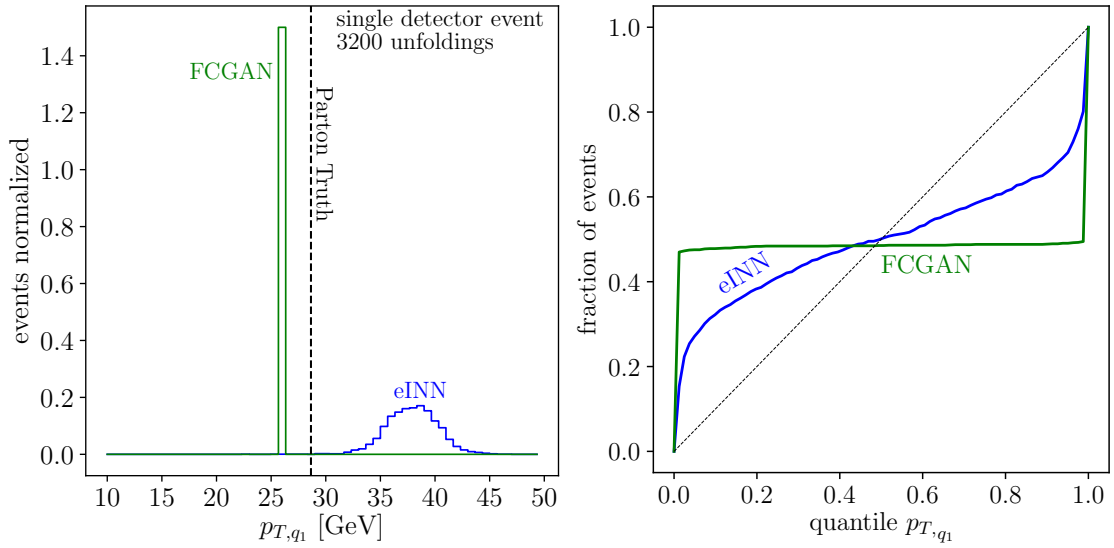

Figure 5.13: Left: illustration of the statistical interpretation of unfolded events for one event. Right: calibration curves for p_{T,q_1} extracted from the FCGAN and the noise-extended eINN.

the calibration is not (yet) perfect. A steep rise for the lower quantile indicates that too many events end up in the first 10% quantile. In other words, the distributions we obtain by sampling over the Gaussian noise for each event are too narrow.

While our noise-extended eINN takes several steps in the right direction, it still faces major challenges: the combination of many different loss functions is sensitive to their relative weights; the balance between MSE and MMD on event constituents has to be calibrated carefully to generate reasonable quantile distributions; when we want to extend the INN to include more detector-level information we have to include an equally large number of random variables on the parton level which makes the training very inefficient. This leads us again [2] to adopt a conditional setup.

Parameter	INN	eINN
Blocks	24	24
Layers per block	2	2
Units per layer	256	256
Trainable weights	$\sim$ 150k	$\sim$ 270k
Epochs	1000	1000
Learning rate	$8 \cdot 10^{-4}$	$8 \cdot 10^{-4}$
Batch size	512	512
Training/testing events	290k / 30k	290k / 30k
Kernel widths	$\sim 2, 8, 25, 67$	$\sim 2, 8, 25, 67$
$D_p + D_{r_p}$	$12 + 4$	$12 + 16$
$D_d + D_{r_d}$	$16 + 0$	$16 + 12$
λ_{MMD}	0.1 (masses only)	0.2
λ_{MMD} increase	-	-

Table 5.2: INN and noise-extended eINN setup and hyper-parameters.

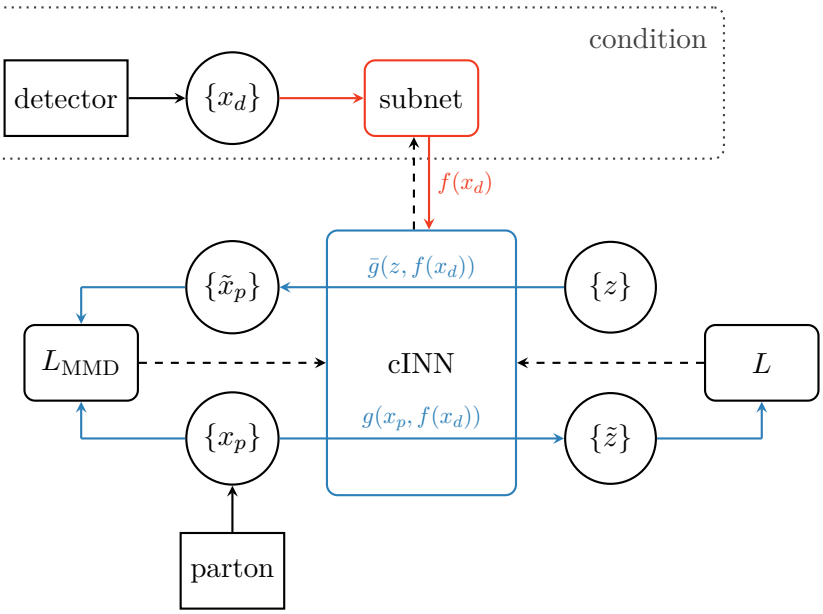

Figure 5.14: Structure of the conditional INN. The input are random numbers $\{z\}$ while $\{x_{d,p}\}$ denote detector-level and parton-level data. The latent dimension loss L follows Eq.(5.28), a tilde indicates the INN generation.

5.4.3 Conditional INN

If a distribution of parton-level events can be described by n degrees of freedom, we should be able to use normalizing flows or an INN to map a n-dimensional random number vector onto parton-level 4-momenta. To capture the information from the detector-level events we need to condition the INN on these events [2, 42, 71], so we link the parton-level data x_p to random noise z under the condition of x_d. Trained on a given process the network should now be able to generate probability distributions for parton-level configurations given a detector-level event and an unfolding model. We note that the cINN is still invertible in the sense that it includes a bi-directional training from Gaussian random numbers to parton-level events and back. While this bi-directional training does not represent the inversion of a detector simulation anymore, it does stabilize the training by requiring the noise to be Gaussian.

A graphic representation of this conditional INN or cINN is given in Fig. 5.14. We first process the detector-level data by a small subnet, *i.e.* $x_d \to f(x_d)$, to optimize its usability for the cINN [72]. The subnet is trained alongside the cINN and does not need to be reversed or adapted. We choose a shallow and wide architecture of two layers with a width of 1024 internally, because four layers degrade already the conditional information and allow the cINN to ignore it. When a deeper subnet is required we advertize to use an encoder, which is initialized by pre-training it as part of an autoencoder. We apply this technique when using the larger ISR input, where it leads to a more efficient training. After this preprocessing, the detector information is passed to the functions s_i and t_i in Eq.(5.25), which now depend on the input, the output, and on the fixed condition. Since the invertibility of the network is independent of the values of s_i and t_i, the network remains invertible between the parton-level events $\{x_p\}$ and the random variables $\{z\}$. This feature stabilizes the training. The cINN loss function is motivated by the simple argument that for the correct set of network parameters θ describing s_i

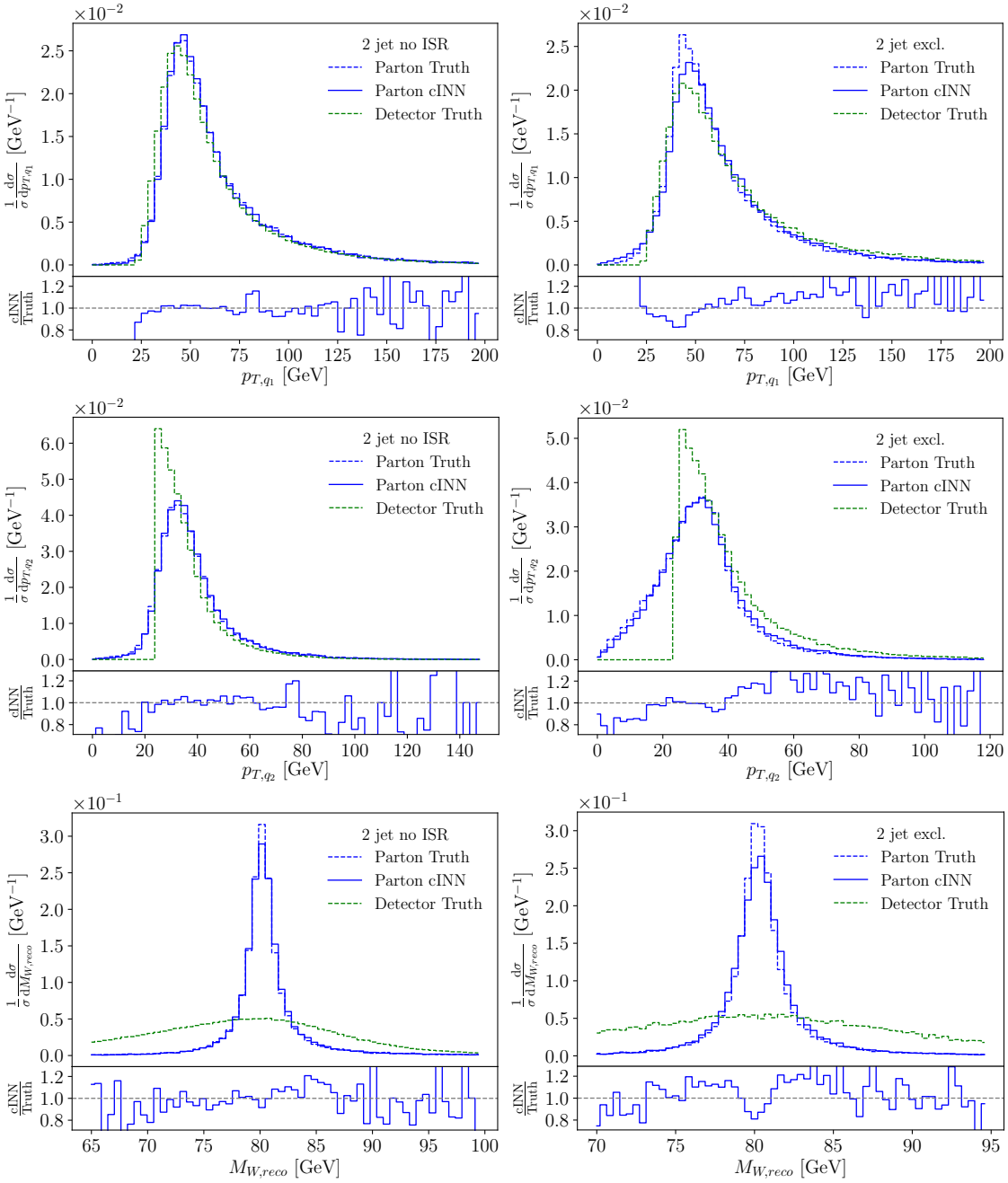

Figure 5.15: cINNed $p_{T,q}$ and $m_{W,\text{reco}}$ distributions. Training and testing events include exactly two jets. In the left panels we use a data set without ISR, while in the right panels we use the two-jet events in the full data set with ISR. The lower panels give the ratio of cINNed to parton-level truth.

and t_i we maximize the (posterior) probability $p(\theta|x_p, x_d)$ or minimize

$$\begin{aligned} L &= -\mathbb{E}_{x_p\sim p_p, x_d\sim p_d}\left[\log p(\theta|x_p, x_d)\right] \\ &= -\mathbb{E}_{x_p\sim p_p, x_d\sim p_d}\left[\log p(x_p|x_d, \theta)\right] - \log p(\theta) + \text{const.} \\ &= -\mathbb{E}_{x_p\sim p_p, x_d\sim p_d}\left[\log p(g(x_p, x_d)) + \log\left|\frac{\partial g(x_p, x_d)}{\partial x_p}\right|\right] - \log p(\theta) + \text{const.}\,, \end{aligned} \tag{5.28}$$

where we first use Bayes' theorem, then ignore all terms irrelevant for the minimization,

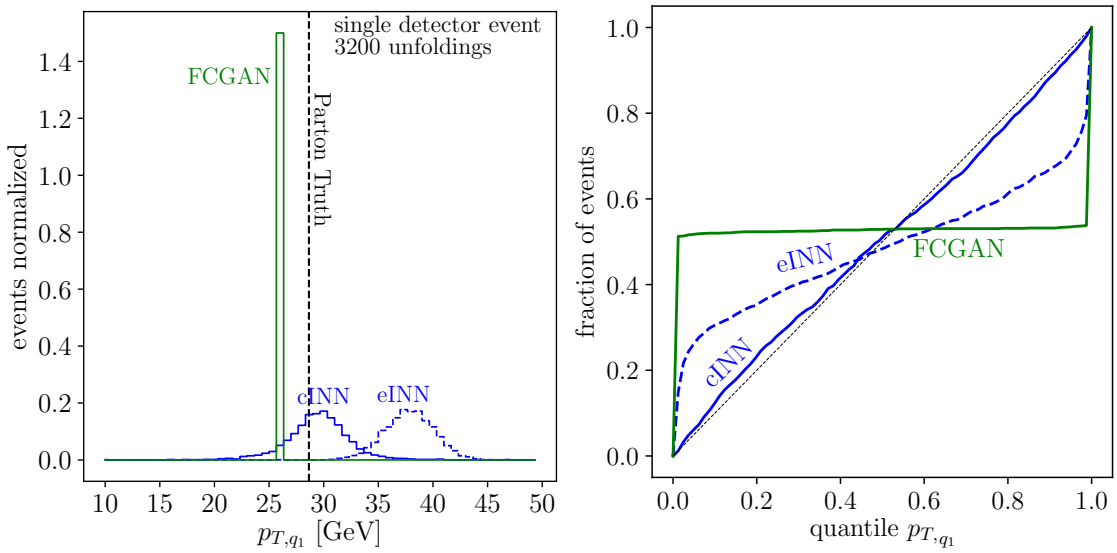

Figure 5.16: Left: illustration of the statistical interpretation of unfolded events for one event. Right: calibration curves for p_{T,q_1} extracted from the FCGAN and the noise-extended eINN, as shown in Fig. 5.13, and the cINN.

and finally apply a simple coordinate transformation for the bijective mapping. The last term is a simple weight regularization, while the first two terms are called the maximum likelihood loss. Since we impose the latent distribution of the random variable $p(g(x_p, x_d))$ to produce a normal distribution centered around zero and with width one, the first term becomes

$$\log p(g(x_p, x_d)) = -\frac{||g(x_p, x_d))||_2^2}{2} \,. \tag{5.29}$$

The parameters of our cINN are given in Tab. 5.3.
In the left panels of Fig. 5.15 we show the unfolding performance of the cINN, trained and tested on the same exclusive 2-jet events as the simpler INNs in Fig. 5.12. Unlike the naive and the noise-extended INNs we cannot evaluate the cINN in both directions, detector simulation and unfolding, so we focus on the detector unfolding. The agreement between parton-level truth and the INN-unfolded distribution is around 10% for the bulk of the p_T distributions, with the usual larger relative deviations in the tails. An interesting feature is still the cut $p_{T,j} > 20$ GeV at the detector level, because it leads to a slight shift in the peak of the p_{T,j_2} distribution. Finally, the reconstructed invariant W-mass and the physical W-width agree extremely well with the Monte Carlo truth owing to the MMD loss.

As in Fig. 5.13 we can interpret the unfolding output for a given detector-level event statistically. First, in the left panel of Fig. 5.16 we show a single event and how the FCGAN, INN, and cINN output is distributed in parton level phase space. The separation between truth and sampled distributions does not have any significance, but we see that the cINN inherits the beneficial features of the noise-extended eINN. In the right panel of Fig. 5.16 we again reconstruct the individual probability distribution from the unfolding numerically. We then determine the position of the parton-level truth in its respective probability distribution for the INN and the cINN. We expect a given percentage of the 1500 events to fall into the correct quantile of its respective probability distribution. The corresponding calibration curve for the cINN is added to the right

panel of Fig. 5.16, indicating that without additional calibration the output of the cINN unfolding can be interpreted as a probability distribution in parton-level phase space for a single detector-level event, as always assuming an unfolding model. Instead of the transverse momentum of the harder parton-level quark we could use any other kinematic distribution at parton level. This marks the final step for a statistically interpretable unfolding.

5.5 Unfolding parton showering

In the previous sections we have used a simplified data set to explore different possibilities to unfold detector level information with invertible networks and GANs. We limited the data to events with exactly two jets, by switching off initial state radiation (ISR). This guarantees that the two jets come from the W-decay, so the network does not have to learn this feature. In a realistic QCD environment we do not have that information, because additional QCD jets will be radiated off the initial and final state partons. In this section we demonstrate how we can unfold a sample of events including ISR and hence with a variable number of jets using a cINN. We know that with very few exceptions [172, 173] the radiation of QCD jets does not help to understand the nature of the hard process. The question is if the unfolding network can not only invert the detector effects, but also QCD jet radiation including kinematic modifications to the hard process in our case done by `Pythia8`. We note that this approach does not define a specific hard process. It allows us to define the relevant hard process with any number of external jets and other features, and to unfold the detector-level events to the parton-level phase space for this hard process.

Parameter	cINN no ISR	cINN ISR incl.
Blocks	24	24
Layers per block	2	3
Units per layer	256	256
Condition/encoder layers	2	8
Units per condition/encoder layer	1024	1024
Condition/encoder output dimension	256	256
Trainable weights	$\sim$ 2 M	$\sim$ 10 M
Encoder pre training epochs	-	300
Epochs	1000	900
Learning rate	$8 \cdot 10^{-4}$	$8 \cdot 10^{-4}$
Batch size	512	512
Training/testing events	290k / 30k	620k / 160k
Kernel widths	$\sim 2, 8, 25, 67$	$\sim 2, 8, 25, 67$
D_p	12	12
D_d	16	25
λ_{MMD}	0.5	0.04
λ_{MMD} increase	-	1.6 / 100 epochs

Table 5.3: cINN setup and hyper-parameters.

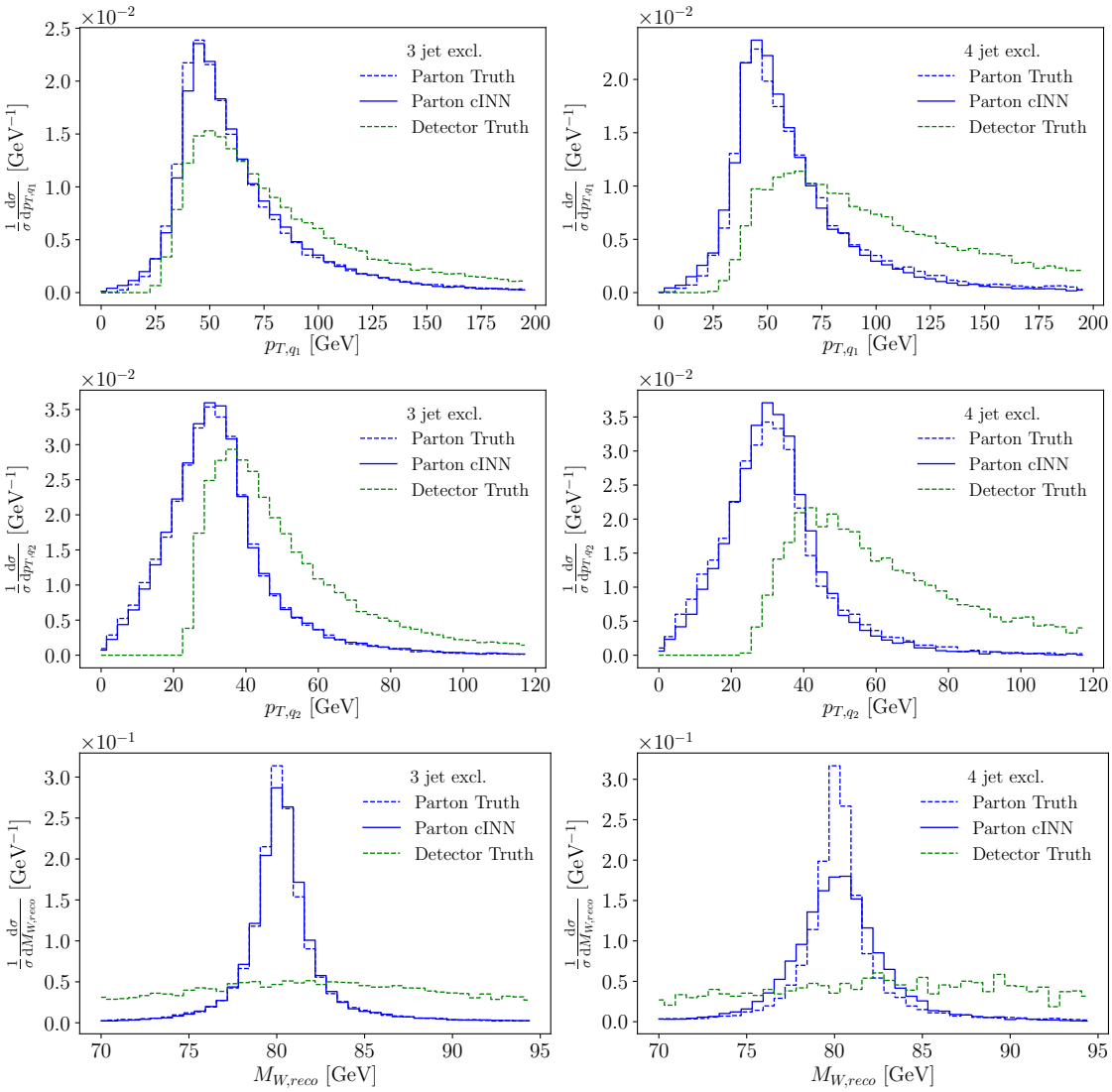

Figure 5.17: cINNed $p_{T,q}$ and $m_{W,\text{reco}}$ distributions. Training and testing events include exactly three (left) and four (right) jets from the data set including ISR.

5.5.1 Individual n-jet samples

In Sec. 5.4.3 we have shown that our cINN can unfold detector effects for ZW-production at the LHC. The crucial new feature of the cINN is that it provides probability distribution in parton-level phase space for a given detector-level event. The actual unfolding results are illustrated in Fig. 5.15, focusing on the two critical distribution known from the corresponding FCGAN analysis. The event sample used throughout Sec. 2.3 and Sec. 2.4 includes exactly two partons from a W-decay with minimal phase space cuts on the corresponding jets. Strictly speaking, these phase space cuts are not necessary in this simulation. The correct definition of a process described by perturbative QCD includes a free number of additional jets,

$$pp \to ZW^\pm + \text{jets} \to (\ell^-\ell^+)\,(jj) + \text{jets}\,, \tag{5.30}$$

For the additional jets we need to include for instance a p_T cut to regularize the soft and collinear divergences at fixed-order perturbation theory. The proper way of generating events is therefore to allow for any number of additional jets and then cut on the number

of hard jets. Since ISR can lead to jets with larger p_T than the W-decay jets, an assignment of the hardest jets to hard partons does not work. We simply sort jets and partons by their respective p_T and let the network learn their relations. We limit the number of jets to four because larger jet numbers appear very rarely and would not give us enough training events.

Combining all jet multiplicities we use 780k events, out of which 530k include exactly two jets, 190k events include three jets and 60k have four or more jets. We split the data into 80% training data and 20% test data to produce the shown plots. For the network input we zero-pad the event-vector for events with less than four jets and add the number of jets as additional information. The training samples are then split by exclusive jet multiplicity, such that the cINN reconstructs the 2-quark parton-level kinematics from two, three, and four jets at the detector level.

As before, we can start with the sample including exactly two jets. The difference to the sample used before is that now one of the W-decay jets might not pass the jet p_T condition in Eq.(5.12), so it will be replaced by an ISR jet in the 2-jet sample. Going back to Fig. 5.15 we see in the right panel how these events are slightly different from the sample with the W-decay jet only. The main difference is in p_{T,q_2}, where the QCD radiation produces significantly more soft jets. Still, the network learns these features, and the unfolding for the sample without ISR and the 2-jet exclusive sample has a similar quality. In Fig. 5.17 we see the same distributions for the exclusive 3-jet and 4-jet samples. In this case we omit the secondary panels because they are dominated by the statistical uncertainties of the training sample. For these samples the network has to extract the parton-level kinematics with two jets only from up to four jets in the final state. In many cases this corresponds to just ignoring the two softest jets and mapping the two hardest jets on the two W-decay quarks, but from the p_{T,q_2} distributions in Fig. 5.15 we know that this is not always the correct solution. Especially in the critical m_{jj} peak reconstruction we see that the network feels the challenge, even though the other unfolded distributions look fine.

5.5.2 Combined n-jet sample

The obvious final question is if our INN can also reconstruct the hard scattering process with its two W-decay quarks from a sample with a variable number of jets. Instead of separate samples, as in Sec. 5.5.1, we now interpret the process in Eq.(5.30) as jet-inclusive. This means that the hard process includes only the two W-decay jets, and all additional jets are understood as jet radiation, described either by resummed ISR or by fixed-order QCD corrections. The training sample consists of the combination of the right panels in Fig. 5.15 and the two panels in Fig. 5.17. This means that the network has to deal with the different number of jets in the final state and how they can be related to the two hard jets of the partonic $ZW \to \ell\ell jj$ process. The number of jets in the final state is not given by individual hard partons, but by the jet algorithm and its R-separation.

In Fig. 5.18 we show a set of unfolded distributions. First, we see that the $p_{T,j}$ thresholds at the detector level are corrected to allow for $p_{T,q}$ values to zero. Next, we see that the comparably flat azimuthal angle difference at the parton level is reproduced to better than 10% over the entire range. Finally, the m_{jj} distribution with its MMD loss regenerates the W-mass peak at the parton level almost perfectly. The precision of this unfolding is not any worse than it is for the case where the number of hard partons and jets have to match and we only unfold the detector effects.

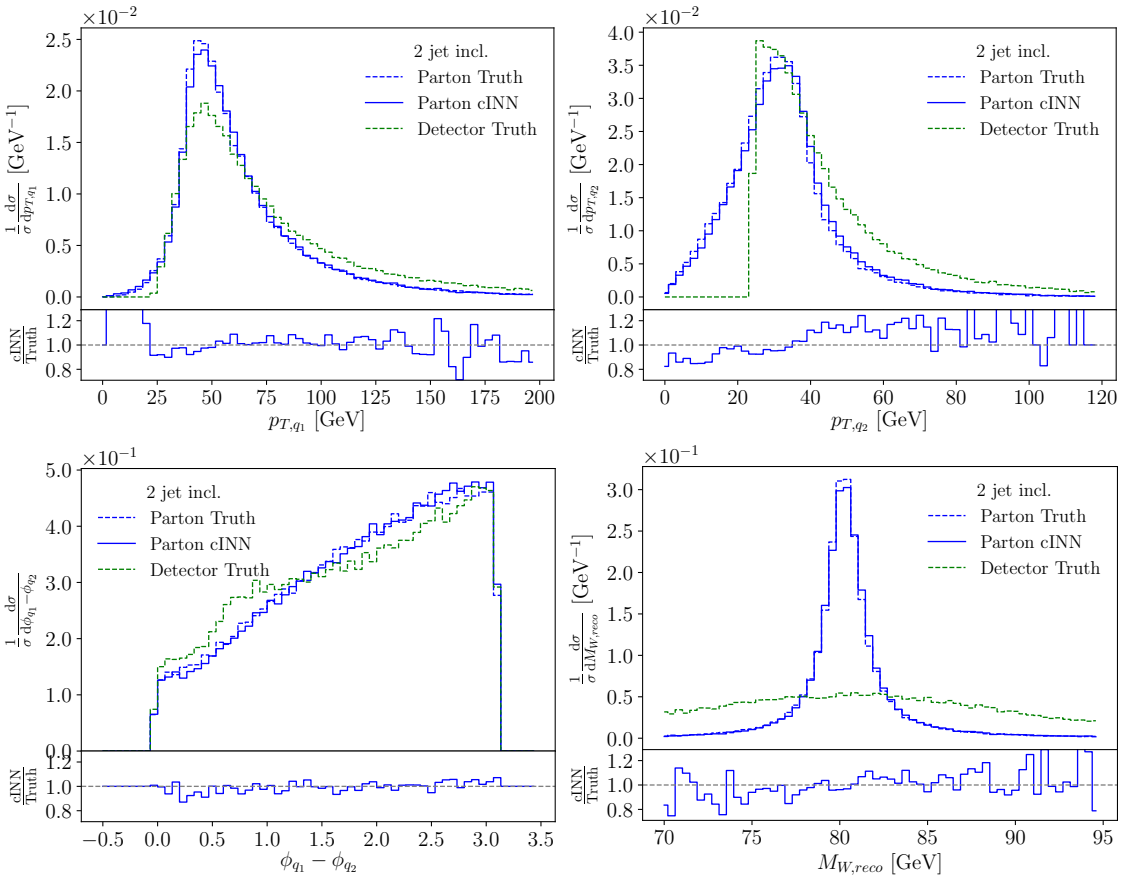

Figure 5.18: cINNed example distributions. Training and testing events include two to four jets, combining the samples from Fig. 5.15 and Fig. 5.17 in one network. At the parton level there exist only two W-decay quarks.

In Fig. 5.19 we split the unfolded distributions in Fig. 5.18 by the number of 2, 3, and 4 jets in the detector-level events. In the first two panels we see that the transverse momentum spectra of the hard partons are essentially independent of the QCD jet radiation. In the language of higher-order calculations this means that we can describe extra jet radiation with a constant K-factor, if necessary with the appropriate phase space mapping. Also the reconstruction of the W-mass is not affected by the extra jets, confirming that the neural network correctly identifies the W-decay jets and separates them from the ISR jets. Finally, we test the transverse momentum conservation at the unfolded parton level. Independent of the number of jets in the final state the energy and momentum for the pre-defined hard process is conserved at the 10^{-4} level. The kinematic modifications from the ISR simulation are unfolded correctly, so we can compute the matrix element for the hard process and use it for instance for inference.

5.6 Conclusion

We have shown that it is possible to invert a simple Monte Carlo simulation, like a fast detector simulation, with a fully conditional GAN. Our example process is $WZ \to (jj)(\ell\ell)$ at the LHC and we GAN away the effect of standard `Delphes`. A naive GAN approach works extremely well when the training sample and the test sample are very similar. In that case the GAN benefits from the fact that we do not actually need an

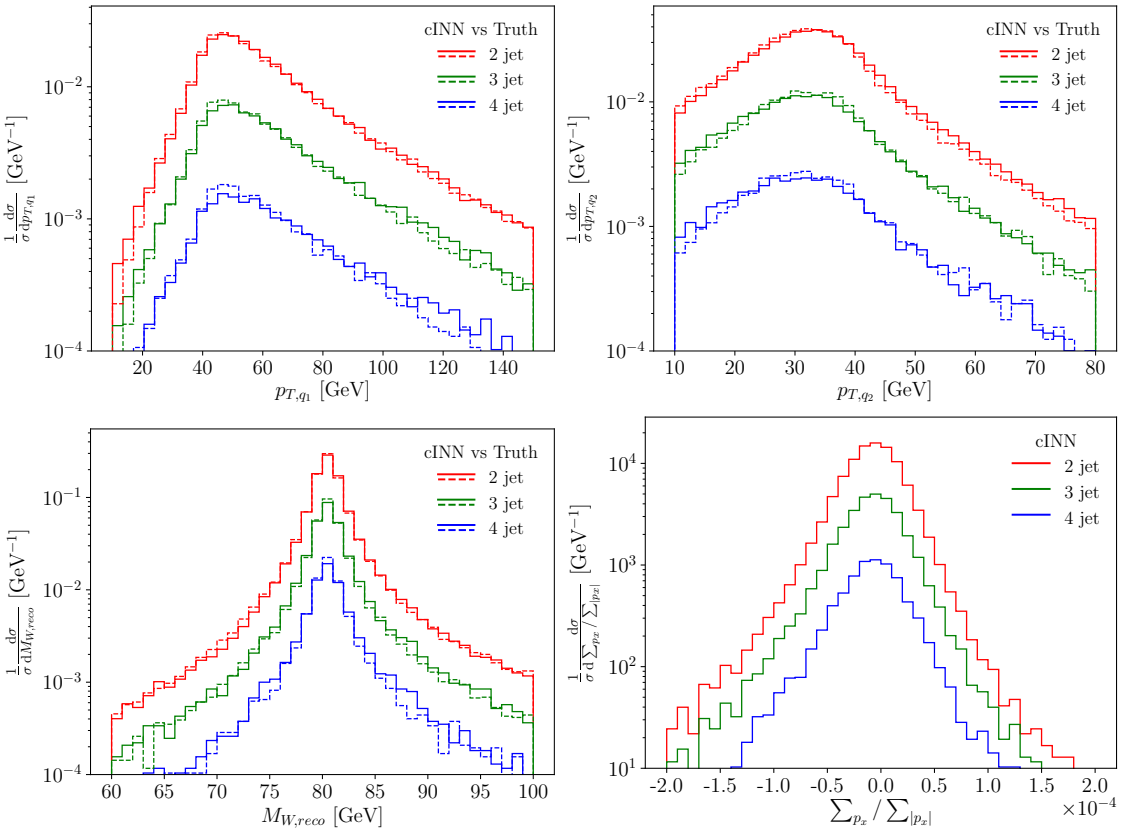

Figure 5.19: cINNed example distributions. Training and testing events include two to four events, combining the samples from Fig. 5.15 and Fig. 5.17 in one network. The parton-level events are stacked by number of jets at detector level.

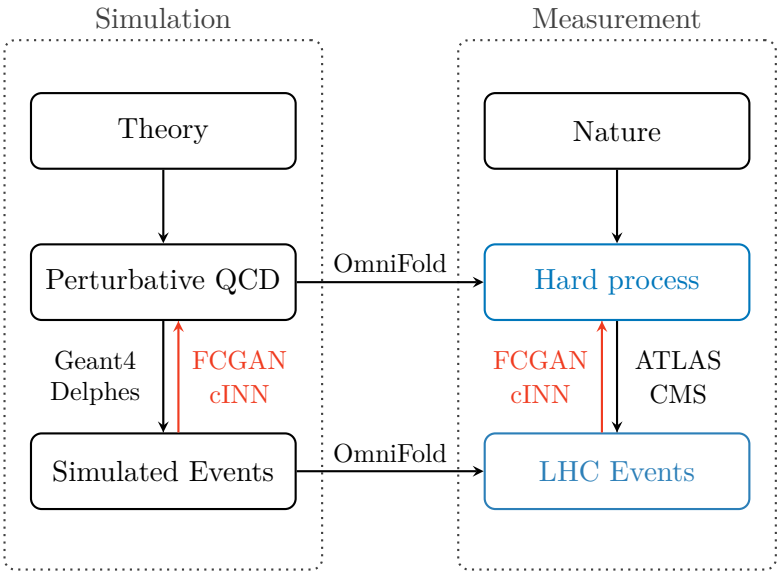

Figure 5.20: Illustration of the complementary FCGAN/cINN and OmniFold [174] approaches.

event-by-event matching of the parton-level and detector-level samples.
If the training and test samples become significantly different we need a fully conditional GAN to invert the detector effects. It maps random noise parton-level events with conditional, event-by-event detector-level input and learns to generate parton-level events from detector-level events. First, we noticed that the FCGAN with its structured mapping provides much more stable predictions in the tails of distributions, where the training sample is statistically limited. Then, we have shown that a network trained on the full phase space can be applied to much smaller parts of phase space, even including cuts in the main kinematic features. The FCGAN successfully maintains a notion of events close to each other at detector level and at parton level and maps them onto each other. This approach only breaks eventually because the MMD loss needed to map narrow Breit-Wigner propagators is not (yet) conditional in our specific setup.

Finally, we have seen that the network reproduces an injected new physics signal as a local structure in phase space. This large degree of model independence reflects another beneficial feature of neural networks, namely the structured mapping of the linked phase spaces.

Afterwards, we have shown how an invertible network (INN) and in particular a conditional INN can also be used to unfold detector effects for the same process. The cINN is not only able to unfold the process over the entire phase space, it also gives correctly calibrated posterior probability distributions over parton-level phase space for given detector-level events. This feature is new even for neural network unfolding.

Next, we have extended the unfolding to a variable number of jets in the final state. This situation will automatically appear whenever we include higher-order corrections in perturbative QCD for a given hard process. The hard process at parton level is defined at the training level. We find that the cINN also unfolds QCD jet radiation in the sense that it identifies the ISR jets and corrects the kinematics of the hard process to ensure energy-momentum conservation in the hard scattering.

In combination, these features should enable analysis techniques like the matrix element method and efficient ways to communicate analysis results including multi-dimensional kinematic distributions.

Note that while finishing the projects presented in this chapter, the OmniFold approach appeared [174]. It aims at the same problem as our FCGAN and cINN, but as illustrated in Fig. 5.20 it is completely complementary. Our FCGAN or cINN uses the simulation based on `Delphes` to train a generative network, which we can apply to LHC events to generate events describing the hard process. The OmniFold approach also starts from matched simulated events, but instead of inverting the detector simulation it uses machine learning to iteratively translate each side of this link to the measured events. This way both approaches should be able to extract hard process information from LHC events, assuming that we understand the relation between perturbative QCD predictions and Monte Carlo events.

Summary and Outlook

In this thesis we have presented three different applications for generative networks: event generation, sample-based subtraction of distributions, and detector unfolding. For this we employed generative adversarial networks (GANs) and invertible neural networks (INNs) as machine learning based methods to perform LHC simulations. We have shown, how these more advanced approches can circumvent or at least alleviate shortcomings of regular simulation methods.

In detail, in Chapter 3 we constructed a GAN which is capable of reproducing the full phase space structure of a realistic LHC process, namely top-pair production all the way down to its decay products. We have seen that using a simple feed-forward neural network in both the generator and discriminator network we can nicely reproduce flat observables such as the transverse momentum up to a deviation of 10%, as shown in Fig. 3.3. We have further shown in Fig. 3.5 and Fig. 3.8 that even 2-dimensional correlations are perfectly reproduced. Furthermore, by including an additional MMD term to the generator loss, our GAN was even capable of resolving sharp phase space structures originating from intermediate on-shell resonances, as shown in Fig. 3.6. A notable feature of this MMD term is that no extra mass or width information about the resonance is needed and that the peak structure is extracted completely dynamical. We only need to know which final-state 4-momentum combination encodes the intermediate resonance. Hence, this MMD term represents a novel approach to map out resonances appropriately.
Even though this setup has already shown remarkable results, we studied this GAN approach in more detail since we wanted to push the precision to a level being comparable to state-of-the-art simulation methods. Therefore, in Sec. 3.3 we modified the generator and discriminator architecture in such a way that the inputs and outputs are more physically motivated as well as more generalizable to other processes. For this, we considered $W + 2$ jet production and investigated how much further we can push the precision of our GAN. We found that applying our modifications, we can improve our GAN performance to decrease the overall deviation to the 1% level in the bulk and 10% in the sparsely populated phase space regions, as shown in Fig. 3.11. In order to benchmark this performance we compared it to two independent Monte Carlo samples of the same size in Fig. 3.10. We can see that our GAN already reaches similar if not equal precision.
Another question which has risen while working on the projects mentioned above was how we can deal with training data which is not yet unweighted but contains non-unit weights. In detail, we were interested to train a GAN on these weighted events while still producing unweighted events as before. Therefore, we have modified the discriminator loss function such that the event weights in the training data are taken into account. More specifically, we replaced the standard expectation value by a weighted mean using the event weights. As we did not modify the loss terms corresponding to the generated events we have restricted our generator to produce unweighted events only. In order to check whether these modifications are correct, we have considered two toy examples

shown in Fig. 3.13 and Fig. 3.16. Besides, showing the distributions we also calculated the unweighting efficiency defined in Sec. 1.5.2 for both examples and obtained efficiencies of 50% and 45% for an 1-dimensional and 2-dimensional example, respectively. We compared these with unweighting efficiencies obtained by the `Vegas` algorithm. With `Vegas` we got efficiencies of 93% and 15% for the 1-dimensional and 2-dimensional case, respectively. From this we can conclude that the neural network approach is outperforming standard techniques if the considered distribution is non-factorizable and more complex. As a physics application we considered muon pair production via the Drell–Yan scattering. We can see in Fig. 3.19 that our GAN can still nicely reproduce the correct distributions when trained on weighted data.

In Chapter 4, we employed GANs to subtract and add distributions based on samples to avoid current statistical limitations of bin-wise methods. For this, we extended our GAN architecture by adding a discriminator for each available dataset we wanted to train on. For instance, we considered two simple 1-dimensional distributions represented as event samples, and we were interested in the difference of both distributions. We further wanted to generate new samples distributed according to this difference. In order to do so, we employed two discriminator networks, where one discriminator was trying to distinguish generated events from true events being drawn from either of the two distributions, and the other one only classified events which were either fake or true following one of the two distributions. By supplementing each event with a class label we were able to directly generate evens which were distributed according to the difference of both distributions, as shown for a toy example in Fig. 4.2. Furthermore, we considered two different LHC applications: background subtraction and collinear dipole subtraction for Drell–Yan scattering.
In the first example, the network was trained to subtract the photon-induced contribution from the full e^+e^- production at LO. Even though this does not yield a state-of-the-art problem for LHC analyses we could further employ it for more involved background subtractions in four body-decays while still preserving all kinematic correlations. In the second example, we combined the full LO matrix element for $Z + g$ production and the collinear approximated contribution expressed as modified Catani-Seymour dipole to obtain events following the finite contribution from real gluon emission.

Finally, in Chapter 5, we considered another application of GANs and also INNs. There, we were interested in unfolding detector effects. As a naive ansatz we used both a standard GAN and INN to directly map from the detector level to the parton level. As we can see in Fig. 5.3 and Fig. 5.12 this works fine if we unfold the entire data at once. However, once we try to cut on detector level events and only unfold a part of the data this procedure fails. The reason for this is that two events which might be close on detector level are not mapped onto events which are also close on parton level. In other words, the network does not learn a mapping between both spaces in a structured manner. In order to solve this problem, we introduced a conditional setup in which the network tries to map random numbers onto the parton level but being conditioned on detector level events. Using this conditioning the slicing of detector level input does not brake the unfolding procedure and we obtain the correct parton level distributions, as shown in Fig. 5.6. A nice feature which only comes with INNs is that they are also capable to give a correctly calibrated posterior probability distribution over parton-level phase space for a single detector-level event. This is new and unique even for neural network unfolding.
Additionally, we have shown that the GAN network also reproduces an possible new resonance as a local structure in phase space even though it was trained on SM events only, as illustrated in Fig. 5.10. From this we can conclude that the neural network preserves

local structures in the mapping between the input and target space.Furthermore, we have also considered unfolding of a variable number of jets and inverted parton showering. We found that the conditional INN can also unfold QCD jet radiation and identifies the ISR jets correctly. It also preserves 4-momentum conservation in the hard scattering process.

Putting everything together, we can conclude that machine learning and especially neural networks can be used for various tasks to supplement standard LHC analyses. While our applications already serve as good examples on how to use generative networks in particle physics there is still more of the story to tell. Undoubtedly, the application of machine learning and neural networks will be beneficial in many other areas of theoretical physics. For instance, neural networks could also be used to speed-up the evaluation of complicated higher-order matrix elements. In either case, we should expect that machine learning will have a huge impact on theoretical and experimental high-energy physics research in the future.

Acknowledgments

First of all I would like to express my gratitude to my supervisor Tilman Plehn who has provided me with many interesting ideas on how to GAN better LHC simulations. He was also keen enough to steal me from Freiburg and heartily welcomed me in Heidelberg within his Pheno group. It has been a pleasure to work with him on various projects.

Furthermore, I would like to thank Jan Pawlowski for refereeing this thesis and Monica Dunford as well as Fred Hamprecht for completing my examination committee.

I gratefully acknowledge support the International Max Planck Research School "Precision Tests of Fundamental Symmetries".

My special thanks goes to the people who have worked with me on a collaboration. Without them, most of the results shown in this thesis would not have been possible. I am indebted to my colleagues Anja Butter, Marco Bellagente and Mathias Backes as well as to my collaborators Gregor Kasieczka, Lynton Ardizzone and Ullrich Köthe.

I would also like to thank the proof-readers Marco Bellagente, Anja Butter, Luca Mantani, Sebastian Schenk, Tabea Nimz, Coralie Schneider and especially Anna Schultz-Coulon. Without them, I would not have been able to turn this piece of sh** into a proper and acceptable thesis.

Indeed, I would also like to thank all the people from Philisophenweg which have somehow influenced or accompanied me throughout my PhD. I would like to thank in random order Anja Butter, Marco Bellagente and Luca Mantani for working with me on so many projects and being part of the PIML as well as Mathias Backes for being my first Bachelor student making my life really easy as a supervisor. Likewise I would also like to thank Michel Luchman, who is also a member of the PIML as well as of the Buffalo club and who made me a member of the latter one, for funny tilted-head moments on the way to Marstall. Moreover, I want to thank my former group members such as Patrick Foldenauer for nice race bike tours through Odenwald and for bringing me to bed, Sebastian Schenk for great discussions on physics and introducing me into the fun sport Boßeln, and Anke Biekötter for nice runs along the Neckar as well as not punishing me for always destroying her interior. I would also like to thank Nastya Filimonova for always being interested in so many things outside of physics and being such a kind person. I will definitely visit you in Amsterdam. Indeed, I also want to thank my long-term office buddy and partner in crime Peter Reimitz aka. *Prollinho* for always cheering me up with one or too many beers and great nights in Untere. I wish you all the best for your upcoming Post-Doc in Brazil. May the birds (thanks to Michael Russel at this point for being a member of the Prolls too) be with you! Obviously, I am also thankful for all other current and former members of the Pheno group including the folks and peeps from Susanne Westhoff and Jörg "Who?" Jäckel who I did not mention personally yet: Susanne Westhoff, Jörg Jäckel, Lennert "Ginger" Thormählen, Ilaria Brivio, Sebastian "Hopp Schwiz Hopp" Bruggisser, Emma Geoffray, Ruth "is on fire" Schäfer, Theo "the machine" Heimel, Elias Bernreuther, Fabian Keilbach, Sascha Diefenbacher, Lukas

Blecher, Nicolas Kiefer, Christopher Lüken-Winkels, Gonzalo Alonso (not the formula 1 driver) Álvarez and anyone else I unfortunately forgot to mention.

Further, I would also like to thank Anna "Schocoholic" Schultz-Coulon who I just got to know at the Graduate Days last year but became such a great part of my life since then. I am always thankful when you remind me to stay positive and that life is more than just physics and fulfilling deadlines.

After all this years, I would also like to take this chance to say thank you to my former math and physics teachers Peter Beck and Andre Hediger who have always encouraged me throughout my school years. Without them I would have probably either ended up as an engineer (I am glad I didn't) or have quit school because of boredom in the other subjects (Actually sports was also fine but I would not have had the talent to earn money with it).

Finally, I would like to say a big thank you to some special friends such as Simon Riedinger for being on this journey called life together since almost 20 years and for all the great experiences we shared in the past and will hopefully share in the future, Christoph Beha for always being such a good friend even though we do not see each other very often, Viktoria Ehret who became a really good friend over the last years and that she is always telling me what she thinks even when I don't wanna hear it, Timon "Unicorn" Eichhorn for being such a great FP I/II partner by taking care of all this experimental stuff I never really cared about and for great conversations about life in general, David Schwarz for being the best flat-mate I ever had and being so competitive when we play squash together as well as always taking his time when I need someone to talk, and Frederic "Freddy" Zeiler who always has an opinion about the things in life that matters the most and for all the great Heineken nights. I am also grateful about all other friends and people who have been part of my life so far and who have helped me to become a grown-up.

Last but not least, I want to say thank you to my family and especially to my parents and my brother Kolja, who have always give me a place to feel safe and home whenever I need it. I am really happy to have you.

Bibliography

[1] A. Butter, T. Plehn, and R. Winterhalder, SciPost Phys. **7** (2019) 6, 075, arXiv:1907.03764 [hep-ph].

[2] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn, and R. Winterhalder, SciPost Phys. **8** (2020) 4, 070, arXiv:1912.00477 [hep-ph].

[3] A. Butter, T. Plehn, and R. Winterhalder, arXiv:1912.08824 [hep-ph].

[4] M. Bellagente, A. Butter, G. Kasieczka, T. Plehn, A. Rousselot, R. Winterhalder, L. Ardizzone, and U. Köthe, arXiv:2006.06685 [hep-ph].

[5] S. Weinberg, Phys. Rev. Lett **19** (1967) 1264.

[6] A. Salam, Conf. Proc. **C680519** (1968) 367.

[7] S. L. Glashow, Nuclear Physics **22** (1961) 4, 579.

[8] D. J. Gross and F. Wilczek, Phys. Rev. Lett. **30** (Jun, 1973) 1343.

[9] H. D. Politzer, Phys. Rev. Lett. **30** (Jun, 1973) 1346.

[10] H. Fritzsch and M. Gell-Mann, eConf **C720906V2** (1972) 135, arXiv:hep-ph/0208010.

[11] H. Fritzsch, M. Gell-Mann, and H. Leutwyler, Phys. Lett. B **47** (1973) 4, 365.

[12] ATLAS Collaboration, Phys. Lett. **B716** (2012) 1.

[13] CMS Collaboration, Phys. Lett. **B716** (2012) 30.

[14] ATLAS Collaboration, Phys. Rev. Lett. **114** (2015) 191803.

[15] F. Englert and R. Brout, Phys. Rev. Lett. **13** (1964) 321.

[16] P. W. Higgs, Phys. Lett. **12** (1964) 132.

[17] P. W. Higgs, Phys. Rev. Lett. **13** (1964) 508.

[18] J. Hilgart, R. Kleiss, and F. L. Diberder, Computer Physics Communications **75** (1993) 1, 191.

[19] F. Berends, R. Pittau, and R. Kleiss, Computer Physics Communications **85** (1995) 3, 437.

[20] A. Denner, S. Dittmaier, M. Roth, and D. Wackeroth, Comput. Phys. Commun. **153** (2003) 462, arXiv:hep-ph/0209330.

[21] K. Arnold *et al.*, Computer Physics Communications **180** (2009) 9, 1661.

[22] S. Carrazza and J. M. Cruz-Martinez, Comput. Phys. Commun. **254** (2020) 107376, arXiv:2002.12921 [physics.comp-ph].

[23] J. Alwall *et al.*, JHEP **07** (2014) 079, arXiv:1405.0301 [hep-ph].

[24] E. Bothmann *et al.*, SciPost Phys. **7** (2019) 3, 034, arXiv:1905.09127 [hep-ph].

[25] T. Sjöstrand, S. Ask, J. R. Christiansen, R. Corke, N. Desai, P. Ilten, S. Mrenna, S. Prestel, C. O. Rasmussen, and P. Z. Skands, Comput. Phys. Commun. **191** (2015) 159, arXiv:1410.3012 [hep-ph].

[26] J. Bellm *et al.*, Eur. Phys. J. C **76** (2016) 4, 196, arXiv:1512.01178 [hep-ph].

[27] DELPHES 3, J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, and M. Selvaggi, JHEP **02** (2014) 057, arXiv:1307.6346 [hep-ex].

[28] J. Allison and other, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **835** (2016) 186 .

[29] G. P. Lepage, Journal of Computational Physics **27** (1978) 2, 192 .

[30] G. P. Lepage, Cornell Preprint **CLNS-447** (1980) .

[31] F. Berends, R. Pittau, and R. Kleiss, Nuclear Physics B **424** (1994) 2, 308.

[32] R. Kleiss and R. Pittau, Computer Physics Communications **83** (1994) 2, 141.

[33] M. Roth, *Precise predictions for four fermion production in electron positron annihilation.* PhD thesis, Zürich, ETH, 1999.

[34] S. Dittmaier and M. Roth, Nucl. Phys. B **642** (2002) 307, arXiv:hep-ph/0206070.

[35] S. Höche, S. Prestel, and H. Schulz, Phys. Rev. D **100** (2019) 1, 014024, arXiv:1905.05120 [hep-ph].

[36] A. Buckley, arXiv:1908.00167 [hep-ph].

[37] J. Bendavid, arXiv:1707.00028 [hep-ph].

[38] M. D. Klimek and M. Perelstein, arXiv:1810.11509 [hep-ph].

[39] C. Gao, J. Isaacson, and C. Krause, arXiv:2001.05486 [physics.comp-ph].

[40] C. Gao, S. Höche, J. Isaacson, C. Krause, and H. Schulz, Phys. Rev. D **101** (2020) 7, 076002, arXiv:2001.10028 [hep-ph].

[41] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale, and S. Schumann, arXiv:2001.05478 [hep-ph].

[42] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, arXiv:1406.2661 [stat.ML].

[43] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother, and U. Köthe, arXiv:1808.04730 [cs.LG].

[44] L. Dinh, J. Sohl-Dickstein, and S. Bengio, arXiv:1605.08803 [cs.LG].

[45] D. P. Kingma and P. Dhariwal, arXiv:1807.03039 [stat.ML].

[46] M. Paganini, L. de Oliveira, and B. Nachman, Phys. Rev. Lett. **120** (2018) 4, 042003, arXiv:1705.02355 [hep-ex].

[47] M. Paganini, L. de Oliveira, and B. Nachman, Phys. Rev. **D97** (2018) 1, 014021, arXiv:1712.10321 [hep-ex].

[48] P. Musella and F. Pandolfi, Comput. Softw. Big Sci. **2** (2018) 1, 8, arXiv:1805.00850 [hep-ex].

[49] M. Erdmann, L. Geiger, J. Glombitza, and D. Schmidt, Comput. Softw. Big Sci. **2** (2018) 1, 4, arXiv:1802.03325 [astro-ph.IM].

[50] ATLAS Collaboration, Tech. Rep. ATL-SOFT-PUB-2018-001, CERN, Geneva, Jul, 2018.

[51] ATLAS Collaboration, A. Ghosh, Tech. Rep. ATL-SOFT-PUB-2019-007, CERN, Geneva, Jun, 2019.

[52] E. Bothmann and L. Debbio, JHEP **01** (2019) 033, arXiv:1808.07802 [hep-ph].

[53] L. de Oliveira, M. Paganini, and B. Nachman, Comput. Softw. Big Sci. **1** (2017) 1, 4, arXiv:1701.05927 [stat.ML].

[54] J. W. Monk, JHEP **12** (2018) 021, arXiv:1807.03685 [hep-ph].

[55] A. Andreassen, I. Feige, C. Frye, and M. D. Schwartz, Eur. Phys. J. **C79** (2019) 2, 102, arXiv:1804.09720 [hep-ph].

[56] S. Carrazza and F. A. Dreyer, Eur. Phys. J. **C79** (2019) 11, 979, arXiv:1909.01359 [hep-ph].

[57] S. Otten *et al.*, arXiv:1901.00875 [hep-ph].

[58] B. Hashemi, N. Amin, K. Datta, D. Olivito, and M. Pierini, arXiv:1901.05282 [hep-ex].

[59] R. Di Sipio, M. Faucci Giannelli, S. Ketabchi Haghighat, and S. Palazzo, JHEP **08** (2020) 110, arXiv:1903.02433 [hep-ex].

[60] J. Lin, W. Bhimji, and B. Nachman, JHEP **05** (2019) 181, arXiv:1903.02556 [hep-ph].

[61] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. J. Smola, CoRR (2008) , arXiv:0805.2368 [cs.LG].

[62] S. Catani and M. H. Seymour, Phys. Lett. **B378** (1996) 287, arXiv:hep-ph/9602277 [hep-ph].

[63] S. Höche, S. Liebschner, and F. Siegert, Eur. Phys. J. **C79** (2019) 9, 728, arXiv:1807.04348 [hep-ph].

[64] A. Gehrmann-De Ridder, T. Gehrmann, and E. W. N. Glover, JHEP **09** (2005) 056, arXiv:hep-ph/0505111 [hep-ph].

[65] R. Frederix, T. Gehrmann, and N. Greiner, JHEP **09** (2008) 122, arXiv:0808.2128 [hep-ph].

[66] J. Currie, E. W. N. Glover, and S. Wells, JHEP **04** (2013) 066, arXiv:1301.4693 [hep-ph].

[67] S. Catani, F. Krauss, R. Kuhn, and B. R. Webber, JHEP **11** (2001) 063, arXiv:hep-ph/0109231 [hep-ph].

[68] M. L. Mangano, M. Moretti, F. Piccinini, R. Pittau, and A. D. Polosa, JHEP **07** (2003) 001, arXiv:hep-ph/0206293 [hep-ph].

[69] D. Gonçalves-Netto, D. López-Val, K. Mawatari, T. Plehn, and I. Wigmore, Phys. Rev. **D87** (2013) 1, 014002, arXiv:1211.0286 [hep-ph].

[70] T. Plehn and M. Takeuchi, J. Phys. **G38** (2011) 095006, arXiv:1104.4087 [hep-ph].

[71] M. Mirza and S. Osindero, arXiv:1411.1784 [cs.LG].

[72] L. Ardizzone, C. Lüth, J. Kruse, C. Rother, and U. Köthe, arXiv:1907.02392 [cs.CV].

[73] C. Winkler, D. Worrall, E. Hoogeboom, and M. Welling, arXiv:1912.00042 [cs.LG].

[74] M. Böhm, A. Denner, and H. Joos, *Gauge theories of the strong and electroweak interaction*, vol. 3. B. G. Teubner, Stuttgart, 2001.

[75] M. E. Peskin and D. V. Schroeder, *An Introduction to Quantum Field Theory; 1995 ed.* Westview, Boulder, CO, 1995.

[76] E. S. Abers and B. W. Lee, Phys. Rept. **9** (1973) 1.

[77] T. Nakano and K. Nishijima, Progress of Theoretical Physics **10** (11, 1953) 581.

[78] M. Gell-Mann, Nuovo Cim. **4** (1956) S2, 848.

[79] Y. Nambu, Phys. Rev. **117** (Feb, 1960) 648.

[80] J. Goldstone, Nuovo Cim. **19** (1961) 154.

[81] S. M. Bilenky and S. T. Petcov, Rev. Mod. Phys. **59** (Jul, 1987) 671.

[82] N. Cabibbo, Phys. Rev. Lett. **10** (1963) 531.

[83] M. Kobayashi and T. Maskawa, Prog. Theor. Phys. **49** (1973) 652.

[84] J. I. Friedman *et al.*, Annual Review of Nuclear Science **22** (1972) 1, 203.

[85] H1 Collaboration, Eur. Phys. J. **C64** (2009) 561.

[86] H. Abramowicz *et al.*, Eur. Phys. J. **C61** (2009) 2, 223.

[87] H. Abramowicz *et al.*, Eur. Phys. J. **C70** (2010) 4, 945.

[88] J. D. Bjorken and E. A. Paschos, Phys. Rev. **185** (1969) 1975.

[89] J. D. Bjorken, Phys.Rev. **179** (1969) 1547.

[90] J. C. Collins, D. E. Soper, and G. F. Sterman, Adv. Ser. Direct. High Energy Phys. **5** (1989) 1.

[91] NNPDF, R. D. Ball *et al.*, arXiv:1706.00428.

[92] A. D. Martin, R. G. Roberts, W. J. Stirling, and R. S. Thorne, The European Physical Journal C - Particles and Fields **39** (2005) 2, 155.

[93] NNPDF, R. D. Ball, V. Bertone, S. Carrazza, L. Del Debbio, S. Forte, A. Guffanti, N. P. Hartland, and J. Rojo, Nucl. Phys. **B877** (2013) 290, arXiv:1308.0598 [hep-ph].

[94] Y. L. Dokshitzer, Sov. Phys. JETP **46** (1977) 641.

[95] D. J. Griffiths, *Introduction to elementary particles; 2nd rev. version.* Physics textbook. Wiley, New York, NY, 2008.

[96] S. Plätzer, arXiv:1308.2922 [hep-ph].

[97] R. Kleiss, W. Stirling, and S. Ellis, Computer Physics Communications **40** (1986) 2, 359 .

[98] R. Brent, Englewood Cliffs, Prentice Hall **19** (01, 2002) .

[99] J. Schmidhuber, Neural Networks **61** (Jan, 2015) 85–117.

[100] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, arXiv:1811.03378 [cs.LG].

[101] V. Nair and G. E. Hinton in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10. Omnipress, Madison, WI, USA, 2010.

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

[103] A. L. Maas, A. Y. Hannun, and A. Y. Ng in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing.* 2013.

[104] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, arXiv:1511.07289 [cs.LG].

[105] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai, arXiv:1811.03804 [cs.LG].

[106] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, *On empirical comparisons of optimizers for deep learning*, 2019.

[107] N. Qian, Neural Networks **12** (1999) 1, 145 .

[108] D. P. Kingma and J. Ba, arXiv:1412.6980 [cs.LG].

[109] S. Lawrence and C. L. Giles in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium.* 2000.

[110] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, arXiv:1207.0580 [cs.NE].

[111] J. E. Moody in *Neural Networks for Signal Processing Proceedings of the 1991 IEEE Workshop.* 1991.

[112] A. Krogh and J. A. Hertz in *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1991.

[113] H. Song, M. Kim, D. Park, and J.-G. Lee, arXiv:1911.08059 [cs.LG].

[114] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, Journal of the American Statistical Association **112** (Feb, 2017) 859–877.

[115] C. Doersch, arXiv:1606.05908 [stat.ML].

[116] L. M. Mescheder, CoRR (2018) , arXiv:1801.04406 [cs.LG].

[117] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann, CoRR (2017) , arXiv:1705.09367 [cs.LG].

[118] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, arXiv:1802.05957 [cs.LG].

[119] Y. Yoshida and T. Miyato, arXiv:1705.10941 [stat.ML].

[120] M. Arjovsky, S. Chintala, and L. Bottou, arXiv:1701.07875 [stat.ML].

[121] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, arXiv:1704.00028 [cs.LG].

[122] D. J. Rezende and S. Mohamed, arXiv:1505.05770 [stat.ML].

[123] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, arXiv:1912.02762 [stat.ML].

[124] I. Kobyzev, S. Prince, and M. A. Brubaker, arXiv:1908.09257 [stat.ML].

[125] T. Müller, B. McWilliams, F. Rousselle, M. Gross, and J. Novák, arXiv:1808.03856 [cs.LG].

[126] B. Nachman and D. Shih, arXiv:2001.04990 [hep-ph].

[127] L. Dinh, D. Krueger, and Y. Bengio, arXiv:1410.8516 [cs.LG].

[128] M. Backes, *How to unweight with GANs.* Bachelor's Thesis, Heidelberg University, 2020.

[129] M. Erdmann, J. Glombitza, and T. Quast, Comput. Softw. Big Sci. **3** (2019) 4, arXiv:1807.01954 [physics.ins-det].

[130] Y. Li, K. Swersky, and R. Zemel, arXiv:1502.02761 [cs.LG].

[131] S. Ravuri, S. Mohamed, M. Rosca, and O. Vinyals, arXiv:1806.11006 [cs.LG].

[132] G. Karolina Dziugaite, D. M. Roy, and Z. Ghahramani, arXiv:1505.03906 [stat.ML].

[133] C.-L. Li, W.-C. Chang, Y. Cheng, Y. Yang, and B. Póczos, arXiv:1705.08584 [cs.LG].

[134] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton, arXiv:1801.01401 [stat.ML].

[135] C.-L. Li, W.-C. Chang, Y. Mroueh, Y. Yang, and B. Póczos, arXiv:1902.10214 [stat.ML].

[136] F. Chollet. https://github.com/fchollet/keras, 2015.

[137] M. Abadi *et al.*, CoRR (2016) , arXiv:1605.08695 [cs.DC].

[138] G. Bevilacqua, M. Czakon, A. van Hameren, C. G. Papadopoulos, and M. Worek, JHEP **02** (2011) 083, arXiv:1012.4230 [hep-ph].

[139] G. Heinrich, A. Maier, R. Nisius, J. Schlenk, and J. Winter, JHEP **06** (2014) 158, arXiv:1312.6659 [hep-ph].

[140] A. Denner and M. Pellen, JHEP **02** (2018) 013, arXiv:1711.10359 [hep-ph].

[141] K. Datta, D. Kar, and D. Roy, arXiv:1806.00433 [physics.data-an].

[142] K. T. Matchev and P. Shyamsundar, arXiv:2002.06307 [hep-ph].

[143] A. Butter, S. Diefenbacher, G. Kasieczka, B. Nachman, and T. Plehn, arXiv:2008.06545 [hep-ph].

[144] S. Frixione and B. R. Webber, JHEP **06** (2002) 029, arXiv:hep-ph/0204244 [hep-ph].

[145] HEP Software Foundation, J. Albrecht *et al.*, Comput. Softw. Big Sci. **3** (2019) 1, 7, arXiv:1712.06982 [physics.comp-ph].

[146] J. Campbell, J. Huston, and F. Krauss, *The Black Book of Quantum Chromodynamics*. Oxford University Press, 2017.

[147] K. Kondo, J. Phys. Soc. Jap. **57** (1988) 4126.

[148] T. Martini and P. Uwer, JHEP **09** (2015) 083, arXiv:1506.08798 [hep-ph].

[149] A. V. Gritsan, R. Röntsch, M. Schulze, and M. Xiao, Phys. Rev. **D94** (2016) 5, 055023, arXiv:1606.03107 [hep-ph].

[150] T. Martini and P. Uwer, JHEP **05** (2018) 141, arXiv:1712.04527 [hep-ph].

[151] M. Kraus, T. Martini, and P. Uwer, Phys. Rev. **D100** (2019) 7, 076010, arXiv:1901.08008 [hep-ph].

[152] S. Prestel and M. Spannowsky, Eur. Phys. J. **C79** (2019) 7, 546, arXiv:1901.11035 [hep-ph].

[153] CMS Collaboration, Eur. Phys. J. **C75** (2015) 11, 542, arXiv:1505.04480 [hep-ex].

[154] ATLAS Collaboration, Eur. Phys. J. **C76** (2016) 10, 538, arXiv:1511.04716 [hep-ex].

[155] F. Bishara and M. Montull, arXiv:1912.11055 [hep-ph].

[156] S. Badger and J. Bullock, arXiv:2002.07516 [hep-ph].

[157] Y. Alanazi, N. Sato, T. Liu, W. Melnitchouk, M. P. Kuchera, E. Pritchard, M. Robertson, R. Strauss, L. Velasco, and Y. Li, arXiv:2001.11103 [hep-ph].

[158] D. Belayneh *et al.*, arXiv:1912.06794 [physics.ins-det].

[159] E. Buhmann, S. Diefenbacher, E. Eren, F. Gaede, G. Kasieczka, A. Korol, and K. Krüger, arXiv:2005.05334 [physics.ins-det].

[160] F. Spano, EPJ Web Conf. **55** (2013) 03002.

[161] G. Cowan, Conf. Proc. C **0203181** (2002) 248.

[162] V. Blobel, *Unfolding Methods in Particle Physics*. Jan, 2011.

[163] R. Balasubramanian, L. Brenner, C. Burgard, G. Cowan, V. Croft, W. Verkerke, and P. Verschuuren, arXiv:1910.14654 [physics.data-an].

[164] L. B. Lucy, Astronomical Journal **79(6)** (1974) 745.

[165] G. Zech, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **716** (Jul, 2013) 1–9, arXiv:1210.5177 [physics.data-an].

[166] G. D'Agostini, Nucl. Instrum. Meth. **A362** (1995) 487.

[167] M. Cacciari, G. P. Salam, and G. Soyez, JHEP **04** (2008) 063, arXiv:0802.1189 [hep-ph].

[168] M. Cacciari, G. P. Salam, and G. Soyez, Eur. Phys. J. C **72** (2012) 1896, arXiv:1111.6097 [hep-ph].

[169] A. Biekötter, A. Knochel, M. Krämer, D. Liu, and F. Riva, Phys. Rev. **D91** (2015) 055029, arXiv:1406.7320 [hep-ph].

[170] J. Brehmer, A. Freitas, D. Lopez-Val, and T. Plehn, Phys. Rev. **D93** (2016) 7, 075014, arXiv:1510.03443 [hep-ph].

[171] A. Paszke *et al.*, arXiv:1912.01703 [cs.LG].

[172] T. Plehn, D. L. Rainwater, and D. Zeppenfeld, Phys. Rev. Lett. **88** (2002) 051801, arXiv:hep-ph/0105325.

[173] M. R. Buckley, T. Plehn, and M. J. Ramsey-Musolf, Phys. Rev. D **90** (2014) 1, 014046, arXiv:1403.2726 [hep-ph].

[174] A. Andreassen, P. T. Komiske, E. M. Metodiev, B. Nachman, and J. Thaler, arXiv:1911.09107 [hep-ph].