entropy

MDPI

Article

Introduction and Analysis of a Method for the Investigation of QCD-like Tree Data

Marko Jercic *, Ivan Jercic and Nikola Poljak

Department of Physics, Faculty of Science, University of Zagreb, 10000 Zagreb, Croatia; mjercic.phy@pmf.hr (I.J.); npoljak.phy@pmf.hr (N.P.)
* Correspondence: mjercic@phy.hr

Abstract: The properties of decays that take place during jet formation cannot be easily deduced from the final distribution of particles in a detector. In this work, we first simulate a system of particles with well-defined masses, decay channels, and decay probabilities. This presents the "true system" for which we want to reproduce the decay probability distributions. Assuming we only have the data that this system produces in the detector, we decided to employ an iterative method which uses a neural network as a classifier between events produced in the detector by the "true system" and some arbitrary "test system". In the end, we compare the distributions obtained with the iterative method to the "true" distributions.

Keywords: quantum chromodynamics; network model; data analysis; interpretability

check for updates

Citation: Jercic, M.; Jercic, I.; Poljak, N. Introduction and Analysis of a Method for the Investigation of QCD-like Tree Data. *Entropy* **2022**, *24*, 104. https://doi.org/10.3390/e24010104

Academic Editor: Rosa M. Benito

Received: 30 November 2021
Accepted: 6 January 2022
Published: 9 January 2022

1. Introduction

The properties of particle interactions determine the evolution of a quantum chromodynamical (QCD) system. Thorough understanding of these properties can help answer many fundamental questions in physics, such as the origin of the Universe or the unification of forces. This is one of the important reasons to collect data with particle accelerators, such as the Large Hadron Collider (LHC) at CERN. However, when collecting this data, we only register complex signals of high dimensionality which we can later interpret as signatures of final particles in the detectors. This interpretation stems from the fact that we, more or less, understand the underlying processes that produce the final particles.

In essence, from the all the particles produced in a collision at the accelerator, only the electron, the proton, the photon, and the neutrinos are stable and can be reconstructed with certainty, given that you have the proper detector. Other particles are sometimes also directly detected, given that they reach the active volume of the detector without first decaying. These include muons, neutrons, charged pions, and charged kaons. On the other hand, short-lived particles will almost surely decay before reaching the detector, and we can only register the particles they decay into.

A similar situation arises with quarks, antiquarks, and gluons, the building blocks of colliding nuclei. When a high-energy collision happens, a quark within a nucleus behaves almost as if it does not interact with neighboring particles, because of a property called asymptotic freedom. If it is struck with a particle from the other nucleus, it can be given sufficient momentum pointing outwards from the parent nucleus. However, we know that there are no free quarks in nature and that this quark needs to undergo a process called administration. This is a process in which quark–antiquark pairs are generated such that they form hadrons. Most of the hadrons are short-lived and they decay into other, more stable, hadrons. The end result of this process is a jet of particles whose average momentum points in the direction of the original outgoing quark. Unfortunately, we don't know the exact quark, nor gluon, decay properties, which serves as a motivation for this work.

The determination of these properties is a long standing problem in particle physics. To determine some of their properties, we turn to already produced data and try to fit

decay models onto them. Even though during the early phases of high-energy collisions the standard QCD parton processes are mainly of the $2 \to 2$ and $2 \to 3$ classes, here we present decay processes ($1 \to 1$ and $1 \to 2$). These are dominant after hadronization takes place and, are simpler to model and computationally less demanding and the model results are more clearly interpretable. Furthermore, these types of processes are used in classical jet reconstruction algorithms [1], which use $2 \to 1$ particle recombinations, further justifying their use. The $2 \to 1$ particle recombinations are in fact time reversed $1 \to 2$ decay chains, meaning that both the recombination algorithms, as well as the described decay chains, follow the same jet model. In the future, we plan to incorporate the $2 \to 2$ and $2 \to 3$ types of processes in our analysis; however, in this paper, we only wanted to present a proof-of-principle calculation.

With every new set of data our understanding changes. This is evident from the fact that we want to simulate a collision event, we can obtain, on average, slightly different results with different versions of the same tool [2]. Therefore, even though simulation tools are regularly reinforced with new observations from data, we can not expect the complete physical truth from them. Instead of trying to perform direct fits to data, we propose the use of machine learning methods to determine the decay properties. In fact, the onset of these methods is already hinted in the traditional approach, as a multivariate fit of decay models to data is already a form of a machine learning technique. It is only natural to extend the existing methods since we can't rely entirely on simulated data. Applications of machine learning techniques to particle physics have already been made, with a recent review given in [3]. An interesting example of such applications include a framework for unsupervised learning without reference to pre-established labels. However, to obtain results, in this case the unsupervised network had to be structured intelligently, based on a qualitative understanding of the data [4]. In contrast, in this work, we present a model based on minimal assumptions and try to recover patterns hidden in the final-state detector data.

To do so, we develop an interpretable model by first simulating a system of particles with well-defined masses, decay channels, and decay probabilities. We take this to be the "true system", whose decay properties we pretend not to know and want to reproduce. Mimicking the real world, we assume to only have the data that this system produces in the detector. Next, we employ an iterative method which uses a neural network as a classifier between events produced in the detector by the "true system" and some arbitrary "test system". In the end, we compare the distributions obtained with the iterative method to the "true" distributions.

This paper is organized as follows. In the materials and methods section we describe the developed artificial physical system and the algorithm used to recover underlying probability distributions of the system. Furthermore, we present in detail the methodology used to obtain the presented results. In the results section, we present our findings to see whether our hypothesis holds true. We conclude the paper with the discussion section.

2. Materials and Methods

The code used for the development the particle generator, the neural network models, and the calculations is written in the the Python programming language using the Keras module with the TensorFlow2 backend [5] and *Numpy* modules. The calculations were performed using a standardized PC setup equipped with an NVIDIA Quadro P6000 graphics processing unit.

2.1. The Physical System

In particle physics, jets are detected as collimated streams of particles. The jet production mechanism is in essence clear: partons from the initial hard process undergo the fragmentation and hadronization processes. In this work, we develop a simplified physical model in which the fragmentation process is modeled as cascaded $1 \to 2$ independent decays of partons with a constant number of decays. We represent each decay of a mother parton of mass M by four real numbers $(\frac{m_1}{M}, \frac{m_2}{M}, \theta, \phi)$, where m_1 and m_2 are the masses of

the daughter particles, and θ and ϕ are the polar and azimuthal angle of the lighter particle, as measured from the rest frame of the mother particle. For simplicity we make all the decays isotropic, which is not necessarily true in real processes. Using conservation laws, the energies and the momenta of the daughter particles for a given mother particle's mass M can be calculated in the rest frame of the mother particle in the following way:

$$E_1 = \frac{1}{2M}(M^2 + m_1^2 - m_2^2)\,, \qquad E_2 = \frac{1}{2M}(M^2 + m_2^2 - m_1^2)\,, \tag{1}$$

$$p_1 = \sqrt{E_1^2 - m_1^2}\,, \qquad p_2 = \sqrt{E_2^2 - m_2^2}\,, \tag{2}$$

$$p_{1x} = -p_{2x} = p_1 \sin\theta\cos\phi, \quad p_{1y} = -p_{2y} = p_1 \sin\theta\sin\phi, \quad p_{1z} = -p_{2z} = p_1\cos\theta\,. \tag{3}$$

The four-momenta of the daughter particles in the laboratory frame are obtained by performing a Lorentz transformation from the rest frame of the mother particle.

We note that this setup also covers the case in which a particle does not decay by setting m_1 equal to zero and $m_2 = M$. This produces a daughter particle with zero energy and a daughter particle with the same four-momentum as the mother particle. Physically, this corresponds to a non-decayed mother particle. Furthermore, we observe that for any pair of daughter masses in which $m_1 + m_2 < M$, some mass has converted to energy. The calculations described here are performed for each decay. This way, in each step we obtain the daughter particles' four-momenta and use them as mother particles' four-momenta in the next step. When this procedure is repeated N times, after $2^N - 1$ decays, we obtain a jet with 2^N particles in the final state (some of which may have mass zero). Assuming that the initial particle's properties are fixed, this means any single jet is described by $4 \times (2^N - 1)$ parameters. We call these parameters the degrees of freedom of a jet, and can sample them from some probability distribution.

To fully define our physical system we set a decay probability distribution function $p(m_1, m_2|M)$, the details of which are given in the following subsection. The aim of our proposed algorithm is to recover these underlying probability distributions, assuming we have no information on them, using only a dataset consisting of jets described with final particles' four-momenta, as one would get from a detector.

2.2. Particle Generator

To generate the jets, we developed an algorithm where we take a particle of known mass that undergoes three successive decays. We consider only the possibility of discrete decays, in the sense that the decay product masses and decay probabilities are well defined. We consider a total of 10 types of particles, labeled A–J, which can only decay into each other. The masses and the decay probabilities of these particles are given in Table 1. In this scenario, the "decay probabilities" p are given by the ratios of decay amplitudes. Thus, the total sum of the probabilities for a given particle to decay into others has to be one, and the probabilities describe the number of produced daughters per N decays, scaled by $1/N$.

Particles A–E are set to be long lived and can thus be detected in the detector, which only sees the decay products after several decays. This can be seen in Table 1 as a probability for a particle to decay into itself. In this way, we assure two things: first, that we have stable particles and second, that each decay in the binary tree is recorded, even if it is represented by a particle decaying into itself. Particles A and C are completely stable, as they only have one "decay" channel, in which they decay back into themselves. On the other hand, particles F–I are hidden resonances: if one of them appears in the i-th step of the decay chain, it will surely decay into other particles in the next, (i + 1)-th step of the chain.

Table 1. Allowed particle decays in the discrete model. The designation p/channel shows the probability that a mother particle will decay into specific daughters.

Particle	**A**		**B**		**C**		**D**		**E**	
mass	0.1		0.6		1.3		1.9		4.4	
p/channel	1	A	0.7	B	1	C	0.3	A + C	0.6	C + C
			0.3	A + A			0.3	A + A	0.4	E
							0.4	D		
particle	F		G		H		I		J	
mass	6.1		8.4		14.2		18.1		25	
p/channel	0.5	A + A	0.9	B + B	0.6	D + D	1	F + G	0.5	F + I
	0.5	B + C	0.1	A + F	0.25	D + E			0.4	G + H
					0.15	E + F			0.1	E + E

To create a jet, we start with particle J, which we call the mother particle, and allow it to decay in one of the decay channels. Each of the daughter particles then decays according to their decay channels, and this procedure repeats a total of 3 times. In the end, we obtain a maximum of 8 particles from the set A–E, with known momenta measured from the rest frame of the mother particle. An example of a generated jet is given in Figure 1.

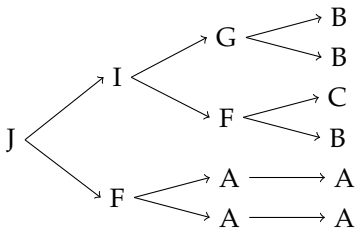


Figure 1. An example of the operation of the discrete jet generator. The mother particle J decays into particles I and F. According to decay probabilities, this happens in half the generated jets. The daughter particles subsequently decay two more times, leaving only stable, detectable particles in the final state.

2.3. Introduction to the Algorithm

Let us assume we have two distinct datasets: one that consists of samples from a random variable X distributed with an unknown probability density $p(x)$, which we call the "real" dataset, and the other, which consists of samples from a random variable Y distributed with a known probability density $q(x)$, which we call the "test" dataset. We would like to perform a hypothesis test between $H_0 : p = p(x)$ and $H_1 : p = q(x)$ using a likelihood-ratio test. The approach we use follows earlier work and employs the Neyman–Pearson lemma [6–8]. This lemma states that the likelihood ratio, Λ, given by

$$\Lambda(p \mid q) \equiv \frac{\mathcal{L}(x \mid real)}{\mathcal{L}(x \mid test)} = \frac{p(x)}{q(x)} \tag{4}$$

is the most powerful test at the given significance level [9].

We can obtain an approximate likelihood ratio Λ by transforming the output of a classifier used to discriminate between the two datasets. Assume that the classifier is a neural network optimized by minimizing the *crossentropy* loss. In this case, the network output gives the probability of x being a part of the real dataset $C_{\mathrm{NN}}(x) = p(real \mid x)$ [10]. If the datasets consist of the same number of samples, we can employ the Bayes' theorem in a simple manner:

$$p(real \mid x) = \frac{p(x \mid real)p(real)}{p(x \mid real)p(real) + p(x \mid test)p(test)} = \frac{p(x \mid p_{\text{real}})}{p(x \mid real) + p(x \mid test)} = \frac{\Lambda}{\Lambda + 1}. \tag{5}$$

A simple inversion of Equation (5) gives:

$$\Lambda = \frac{p(x)}{q(x)} = \frac{C_{\text{NN}}(x)}{1 - C_{\text{NN}}(x)}, \tag{6}$$

$$p(x) = \frac{C_{\text{NN}}(x)}{1 - C_{\text{NN}}(x)} q(x). \tag{7}$$

Therefore, in ideal conditions, the unknown probability density $p(x)$ describing the real dataset can be recovered with the help of the known probability density $q(x)$ and a classifier, using (7). It must be noted that (7) is strictly correct only for optimal classifiers, which are unattainable. In our case, the classifier is optimized by minimizing the *crossentropy* loss defined by

$$L = -\frac{1}{n} \sum_{i=1}^{n} [y(x_i) \ln C_{\text{NN}}(x_i) + (1 - y(x_i)) \ln(1 - C_{\text{NN}}(x_i))], \tag{8}$$

where $y(x_i)$ is 1 if x_i is a part of the real dataset, and 0 if x_i is a part of the test dataset. We can safely assume that the final value of loss of the suboptimal classifier is greater than the final value of loss of the optimal classifier:

$$L_{\text{optimal}} < L < \ln 2. \tag{9}$$

The value of $\ln 2$ is obtained under the assumption of the *worst* possible classifier. To prove our findings, in the next step we regroup the sums in the loss function in two parts, corresponding to the real and the test distributions:

$$-\frac{1}{n} \sum_{i \in real} \ln C_{\text{NN}}^{\text{optimal}}(x_i) < -\frac{1}{n} \sum_{i \in real} \ln C_{\text{NN}}(x_i) < -\frac{1}{n} \sum_{i \in real} \ln \frac{1}{2}, \tag{10}$$

$$-\frac{1}{n} \sum_{i \in test} \ln \left[1 - C_{\text{NN}}^{\text{optimal}}(x_i)\right] < -\frac{1}{n} \sum_{i \in test} \ln[1 - C_{\text{NN}}(x_i)] < -\frac{1}{n} \sum_{i \in test} \ln \frac{1}{2}. \tag{11}$$

After expanding inequality (10) we obtain

$$-\frac{1}{n} \sum_{i \in real} \ln \left[\frac{C_{\text{NN}}^{\text{optimal}}(x_i)}{1 - C_{\text{NN}}^{\text{optimal}}(x_i)}\right] < -\frac{1}{n} \sum_{i \in real} \ln \left[\frac{C_{\text{NN}}(x_i)}{1 - C_{\text{NN}}(x_i)}\right] < -\frac{1}{n} \sum_{i \in real} \ln 1. \tag{12}$$

According to Equation (7), we can recover the real probability density $p(x)$ when using the optimal classifier. However, if one uses a suboptimal classifier, a slightly different probability density $p'(x)$ will be calculated. As the ratios that appear as arguments of the logarithms in Equation (12) correspond to distribution ratios, it follows that

$$-\frac{1}{n} \sum_{i \in real} \ln \left[\frac{p(x_i)}{q(x_i)}\right] < -\frac{1}{n} \sum_{i \in real} \ln \left[\frac{p'(x_i)}{q(x_i)}\right] < -\frac{1}{n} \sum_{i \in real} \ln 1. \tag{13}$$

After some simplification this becomes

$$\sum_{i \in real} \ln p(x_i) > \sum_{i \in real} \ln p'(x_i) > \sum_{i \in real} \ln q(x_i). \tag{14}$$

If an analogous analysis is carried out for inequality (11) we get

$$\sum_{i \in test} \ln p(x_i) < \sum_{i \in test} \ln p'(x_i) < \sum_{i \in test} \ln q(x_i). \tag{15}$$

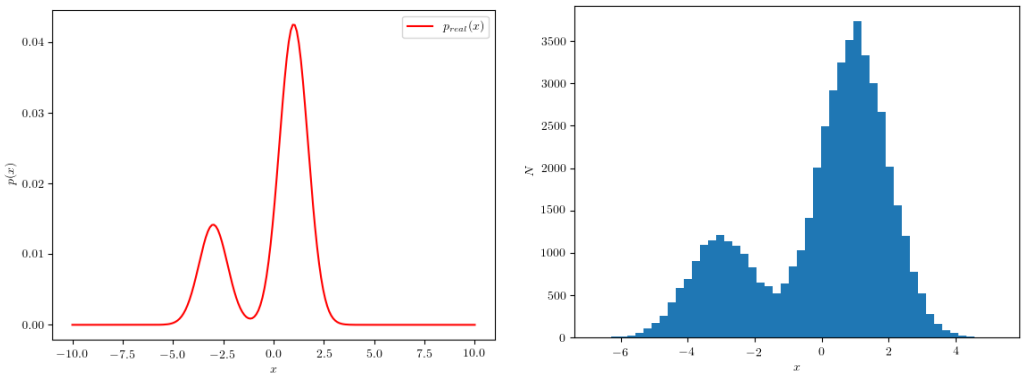
At this point, we recall that $p'(x_i)$ is obtained from $q(x_i)$. If $p(x_i)$ and $q(x_i)$ are assumed to be different, then we can see that for both the test and the real subsets, $\sum_{i \in test} \ln p'(x_i)$ falls in between the corresponding sums for p and q distributions. This means that the probability density $p'(x)$ moves from $q(x)$ towards the real probability density $p(x)$. In a realistic case, Equation (7) can't be used to completely recover the real probability density $p(x)$ within a single step. However, it can be used in an iterative method; starting with a known distribution $q(x)$, we can approach the real distribution more and more with each iteration step.

2.4. A Simple Example

Let us illustrate the recovery of an unknown probability density by using a classifier on a simple example. We start with a set of 50,000 real numbers generated from a random variable with a probability density given by

$$p_{\text{real}}(x) = \frac{1}{4}\mathcal{N}(-1, 1) + \frac{3}{4}\mathcal{N}(3, 1)\,, \tag{16}$$

where $\mathcal{N}(\mu, \sigma^2)$ denotes a normal distribution. A histogram of values in this set is shown in Figure 2. Let us now assume we do not know $p_{\text{real}}(x)$ and want to recover it using the procedure outlined in the previous subsection. This set will be denoted as the "real" dataset and the underlying probability density will be denoted as the "real" probability density.



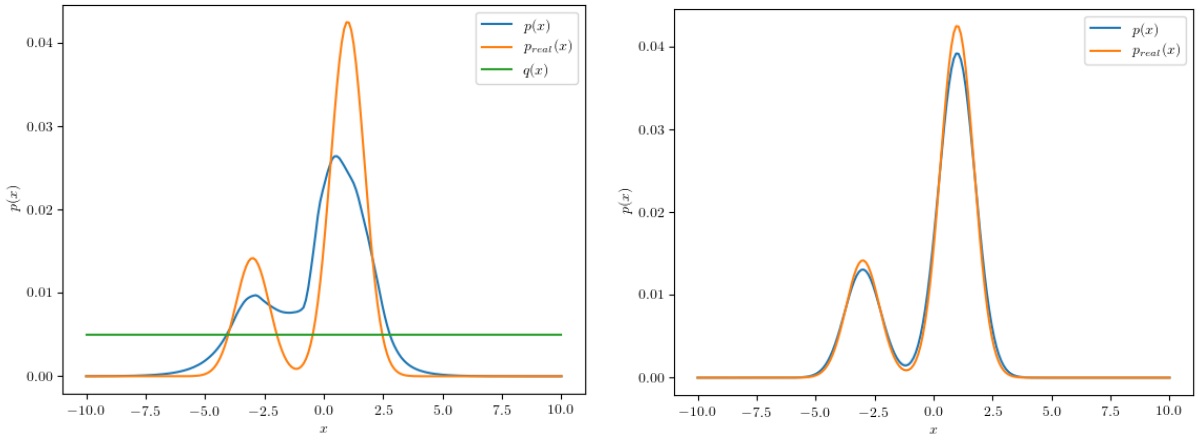
(**a**) (**b**)

Figure 2. (**a**) The normalized probability density for the example, given by Equation (16). (**b**) A histogram of values sampled from the set generated by the same equation.

To construct the "test" dataset, we generate values with a uniform probability density in the interval $[-10, 10]$. Finally, we construct a simple neural network which is used as a classifier that distinguishes the examples from the real dataset from examples from the test dataset. The classifier we use is a simple *feed-forward* neural network with 100 hidden units using a ReLU activation function. The activation function of the final neural network output is the *sigmoid* function, which we use to constrain the output values to the interval [0, 1]. After the classifier is trained to discriminate between the two datasets by minimizing the *binary cross-entropy* loss, we evaluate its output at 200 equidistant points between -10 and 10. Using Equation (7), the probability distribution $p_{\text{calculated}}(x)$ is calculated using the classifier outputs. The calculated $p_{\text{calculated}}(x)$ is compared to the real probability density $p_{\text{real}}(x)$ and is shown in Figure 3.

Although the resulting probability density differs from the real probability density due to the non-ideal classifier, we can conclude that the calculated $p_{\text{calculated}}(x)$ is considerably

closer to $p_{\text{real}}(x)$ than to uniform probability density $q(x)$ used to generate the test dataset. Now, if we use the calculated $p_{\text{calculated}}(x)$ to construct a new test dataset and repeat the same steps, we can improve the results even more. This procedure can therefore iteratively improve the resemblance of $p_{\text{calculated}}(x)$ to $p_{\text{real}}(x)$ to the point where the datasets are so similar that the classifier cannot distinguish between them. In this simple example convergence is reached after the 5th iteration, since no significant improvement is observed afterwards. The calculated probability density $p_{\text{calculated}}(x)$ after the final iteration is shown in Figure 3 compared to the real distribution $p_{\text{real}}(x)$. It is clear that in this case the procedure converges, and we could possibly obtain a better match between $p_{\text{calculated}}(x)$ and $p_{\text{real}}(x)$ if we used a more optimal classifier.



(**a**) (**b**)

Figure 3. Both panels: The calculated probability density $p_{\text{calculated}}(x)$ (blue line) compared to the real probability density p_{real}(x) (orange line). (**a**) The left panel shows the comparison after one iteration of the algorithm, alongside the starting "test" distribution (green line). (**b**) The right panel shows the comparison after the 5th iteration. In this panel we omit the starting distribution (green line) to more clearly show the comparison of the calculated and the real distributions.

In essence, a simple histogram could be used in this simple example to determine the underlying probability distribution instead of using the method described above. However, in case of multivariate probability distributions, which can be products of unknown probability distributions, a histogram approach would not prove useful.

2.5. Extension to Jets

We would now like to apply the described procedure on the datasets that contain jets. Every jet, represented by a binary tree of depth N, consists of $2^N - 1$ independent decays producing a maximum of 2^N particles in the final state. As all the decays are isotropic in space, a jet can be described with a $4 \times (2^N - 1)$-dimensional vector $\vec{x}$ and a probability distribution function given by

$$p(\vec{x}) = \prod_{i}^{2^N-1} p(m_1^i, m_2^i | M^i) p_\theta(\theta^i) p_\phi(\phi^i), \tag{17}$$

where i denotes the decay index, (m_1^i, m_2^i, θ^i, ϕ_i) are the components of the vector $\vec{x}$ and $p_\theta(\theta^i)$ and $p_\phi(\phi^i)$ are the angle probability distributions. The parameter M^i represent the mass of the mother particle in the i-th decay. As both angles are uniformly spatially distributed, they contribute to the probability with a simple constant factor. Therefore,

when plugging $p(\vec{x})$ from Equation (17) into Equation (7) we can omit angles, as the constant factors will cancel each other out:

$$\prod_{i}^{2^N-1} p(m_1^i, m_2^i|M^i) = \frac{C_{NN}(\vec{x})}{1-C_{NN}(\vec{x})} \prod_{i}^{2^N-1} q(m_1^i, m_2^i|M^i). \tag{18}$$

Taking the logarithm of both sides:

$$\sum_{i}^{2^N-1} \ln p(m_1^i, m_2^i|M^i) = \ln C_{NN}(\vec{x}) - \ln(1-C_{NN}(\vec{x})) + \sum_{i}^{2^N-1} \ln q(m_1^i, m_2^i|M^i). \tag{19}$$

Unfortunately, we cannot explicitly obtain the probability $p(m_1, m_2 \mid M)$ directly from Equation (19) without solving a linear system of equations. This task proves to be computationally exceptionally challenging due to the high dimensionality of the dataset. In order to avoid this obstacle, we introduce a neural network f to approximate $\ln p(m_1, m_2|M)$. We can optimize this neural network by minimizing the *mean squared error* applied to the two sides of Equation (19).

2.6. The 2 Neural Networks (2NN) Algorithm

At this point, we are ready to recover the underlying probability distributions from an existing dataset that consists of jets described by the four-momenta of the final particles. We denote the jets from this dataset as "real". The building blocks of the full recovery algorithm are two independent neural networks; the aforementioned classifier C_{NN} and the neural network f. Based on the usage of two neural networks, we dubbed the algorithm *2NN*. The detailed architectures of both networks are given in Appendix A. The workflow of the *2NN* algorithm is simple: first we initialize the parameters of both neural networks. Then, we generate a test dataset using the neural network f. The test dataset and the real dataset are fed into the classifier network, which produces a set of linear equations in the form of Equation (19). We approximate the solution to these by fitting the neural network f, which in turn produces a new test dataset. The procedure is continued iteratively until there are no noticeable changes in the difference of the real and test distributions. More detailed descriptions of the individual steps are given in the next subsections.

2.6.1. Generating the Test Dataset

After the parameters of the neural network f are initialized, we need to generate a test dataset of jets with known decay probabilities $q(\vec{x})$. The input of the neural network f is a vector consisting of 3 real numbers: $a = m_1/M$, $b = m_2/M$ and M. We denote the output of the neural network with $f(a, b, M)$. Due to conservation laws, the sum $a + b$ needs to be strictly less or equal to 1. We can assume $a \leq b$ without any loss of generality. In order to manipulate with probabilities a partition function:

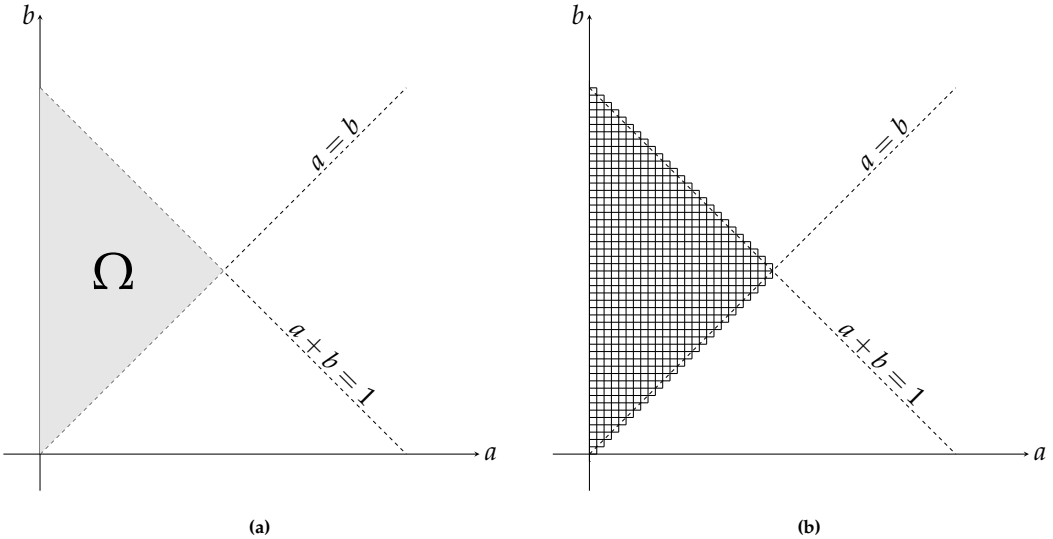
$$Z(M) = \int_{\Omega} e^{f(a,b,M)} \, \mathrm{d}a\mathrm{d}b \tag{20}$$

needs to be calculated. Here, Ω denotes the entire probability space and is shown as the gray area in the left panel of Figure 4. To calculate the integral in the above expression, the probability space is discretized into 650 equal areas shown in the right panel of Figure 4. These areas are obtained by discretizing the parameters a and b into equidistant segments of length 0.02. After the discretization, the partition function $Z(M)$ then becomes

$$Z(M) \approx \sum_{j} \sum_{k} e^{f(a_j, b_k, M)} \, . \tag{21}$$

The decay probability is then given by

$$q(m_1, m_2 \mid M) = \frac{e^{f(a,b,M)}}{Z(M)}\,. \tag{22}$$



(a) (b)

Figure 4. (**a**) The left panel shows the entire allowed probability space of the parameters a and b, designated by Ω. Due to conservation laws, $a + b \leq 1$ needs to hold true. To describe our system, we selected the case where $a \leq b$, which we can do without loss of generality. (**b**) The right panel shows the discretized space Ω, as used to evaluate the partition function.

This probability is then used to generate a single decay of a given mother particle of mass M. Each decay is generated by randomly sampling a pair of parameters (a, b) according to the calculated probability (Equation (22)) from 650 possible pairs which form the probability space. The kinematical parameters of the decay are calculated according to Equations (1)–(3) by setting $m_1 = aM$ and $m_2 = bM$, to obtain the four momenta of the daughter particles. To produce a full event, i.e., a jet, this procedure is done iteratively N times as described in Section 2.1. The four-momentum of the initial parton in its rest frame is sampled from the distribution of the total jet mass in the "real" dataset which is, in turn, obtained from the total jet mass histogram. It must be noted that the jets in the "real" dataset have to be preprocessed by applying suitable Lorentz transformations, so that the total momentum of each jet in its rest frame equals zero. After applying this procedure we have a test dataset in which each jet is represented as a list of 2^N particles and their four-momenta. For each jet in the test dataset we store $2^N - 1$ triplets (a, b, M) and their corresponding probabilities calculated by Equation (22) for each decay in the jet. As all of the decays are independent, the total probability of a jet appearing in the test dataset is a direct product of the probabilities for each particular decay. The value of N is a parameter of the algorithm which needs to be guessed, as we cannot foresee how many consecutive decays are needed to produce some a specific real dataset. As our kinematic setup allows non-decays, the value of N has to be larger or equal to the degree of the longest chain of consecutive decays in a physical system.

2.6.2. Optimizing the Classifier

The classifier used in this work is a convolutional neural network. The input to these type network are sets of images. For this purposes, all the jets are preprocessed by transforming the list of particles' four-momenta into jet images. Two 32×32 images are produces for a single jet. In both images the axes correspond to the decay angles θ and ϕ, while the pixel values are either the energy or the momentum of the particle found in that

particular pixel. If a pixel contains two or more particles, their energy and momenta are summed and stored as pixel values. The transformation of the jet representations is done on both the real and the test datasets. We label the "real" jet images with the digit 1 and "test" jet images with the digit 0. The classifier is then optimized by minimizing the *binary cross-entropy* loss between the real and the test datasets. The optimization is performed by ADAM algorithm [11]. It is important to note that the sizes of both datasets need to be the same.

2.6.3. Optimizing the Neural Network f

After the classifier is optimized, a new jet dataset is generated by using the neural network f as described in Section 2.6.1. Just as earlier, the generated jets are first transformed to jet images and then fed to the classifier. As we have access to each of the decay probabilities for each jet, the right side of Equation (19) can be easily calculated for all the jet vectors $\vec{x}$ in the dataset. This way we can obtain the desired log value of the total probability for each jet $p(\vec{x})$:

$$\ln p(\vec{x}) = \ln C_{NN}(\vec{x}) - \ln(1 - C_{NN}(\vec{x})) + \sum_{i}^{2^N-1} \ln q(m_1^i, m_2^i | M). \tag{23}$$

Finally, we update the parameters of the neural network f by minimizing the expression given by

$$L = \frac{1}{n} \sum_{i}^{n} \left[\sum_{j}^{2^N-1} f(a_j^i, b_j^i, M_j) - \ln p_i(\vec{x}) \right]^2, \tag{24}$$

where i denotes the jet index and j denotes the decay index in a particular jet. To this purpose we introduce a model which takes a sequence of $2^N - 1$ triplets (a^j, b^j, M_j) as an input, and simply gives the sum of neural network f outputs for each of the triplets. We store all the triplets during the generation of a dataset. After this step, the weights of the neural network are updated in such a way that the network output values $f(a, b, M)$ are on average closer to the real log value of $p(m_1, m_2 \mid M)$. The updated network f is used to generate the test dataset in the next iteration.

2.7. Evaluation of the 2NN Algorithm

Upon completion of each iteration of the algorithm, the underlying probability densities for a decay of the mother particle with a given mass M can be obtained from the output values of the neural network f according to Equation (22). In the results section, the 2NN algorithm is evaluated in terms of the Kullback–Leibler divergence (KL) in the following way [12]:

$$KL(M) = \sum_j \sum_k p_{\text{real}}(m_1^j, m_2^k \mid M) \left[\ln p_{\text{real}}(m_1^j, m_2^k \mid M) - f(a^j, b^k, M) + \ln Z(M) \right] \tag{25}$$

where the sum is performed over the whole probability space. The KL divergence is a non-negative measure of the difference between two probability densities defined on the same probability space. If the probability densities are identical, the KL divergence is zero. Note that the KL divergence defined in this way is dependent on the mass of the mother particle and has to be recalculated every time the mass of the mother particle changes.

3. Results

In this section, we present our findings after applying the $2NN$ algorithm on 500,000 jets created using the particle generator described in Section 2.2. In each iteration, the classifier is optimized using 50,000 randomly picked jets from the "real" dataset and 50,000 jets generated using the neural network f. To optimize the neural network f, we use 50,000 jets as well. The algorithm performed 800 iterations. In each step, a single epoch is used

when training the classifier and the neural network f in order to prevent overfitting on the small subsamples, which would slow down the algorithm. After the final iteration of the $2NN$ algorithm we obtain the calculated probability densities, which can be then used to generate samples of jets. First, we show the energy spectrum of the particles in the final state in jets generated by the calculated probabilities. This spectrum is directly compared to the energy spectrum of particles taken from jets belonging to the "real" dataset and shown on Figure 5.

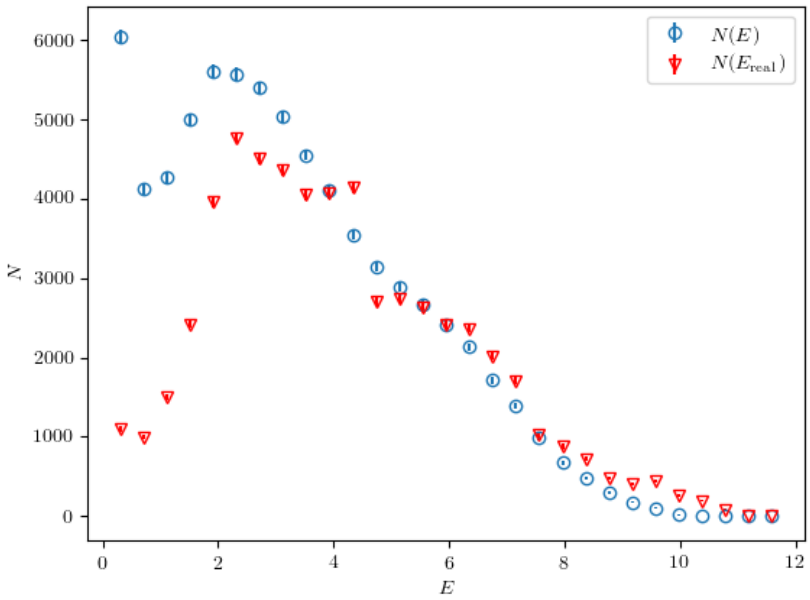
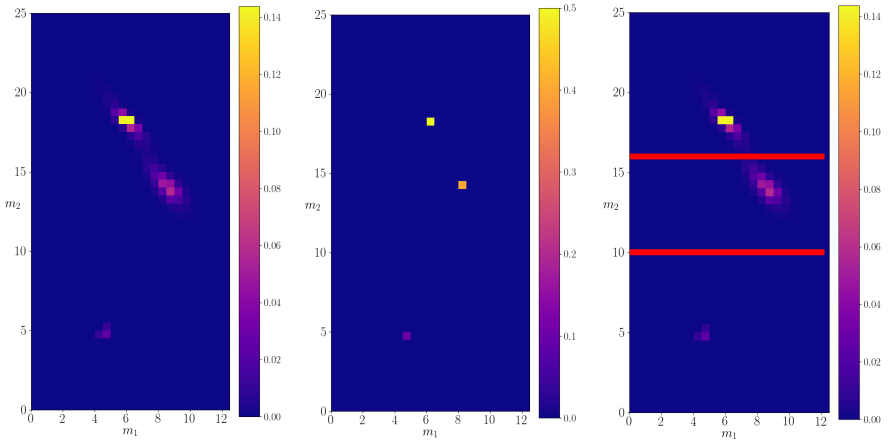


Figure 5. The energy spectrum of the particles in the final state in jets generated by the calculated probabilities, compared to the energy spectrum of particles taken from jets belonging to the "real" dataset.

The plotted spectra are obtained using 10,000 jets from each dataset. The error bars in the histogram are smaller than the marker size and are thus not visible. A resemblance between the two spectra is notable, especially at higher energies. This points to the fact that the calculated probabilities are approximately correct, so we can use them to generate samples of jets that resemble "real" jets. To further examine the calculated probability densities we need to reconstruct the hidden resonances which are not found in the final state. For this purpose, the calculated probability densities for mother particle masses of $M = 25.0$, $M = 18.1$, $M = 14.2$, and $M = 1.9$ are analyzed and compared to the real probability densities in the following subsections. These masses are chosen since they match the masses of the hidden resonances, as was introduced in Table 1.

3.1. Mother Particle with Mass M = 25.0

The calculated $2d$-probability density $p(m_1, m_2 \mid M)$ is shown in Figure 6, compared to the real probability density. A simple eye reveals that three possible decays of particle of mass $M = 25.0$ are recognized by the algorithm. After dividing the probability space as in Figure 6c with lines $m_2 > 16.0$ and $m_2 < 10.0$, we calculate the mean and the variance of the data on each subspace. As a result, we obtain $(m_1, m_2) = (18.1 \pm 0.5, 6.1 \pm 0.5)$ for $m_2 > 16.0$, $(m_1, m_2) = (14.0 \pm 0.7, 8.4 \pm 0.7)$ for $16.0 \leq m_2 > 10.0$ and $(m1, m2) = (4.8 \pm 0.2, 4.6 \pm 0.2)$ for $m_2 \leq 10.0$. These mean values closely agree with the masses of the resonances expected as the products of decays of the particle with mass $M = 25.0$. The calculated small variances indicate that the algorithm is very precise. The total decay probabilities for each of the subspaces are equal to $p_1 = 0.48$, $p_2 = 0.47$, $p_3 = 0.05$, which approximately agree with the probabilities of decay channels of the particle with mass $M = 25.0$, as defined in Table 1.



(**a**) (**b**) (**c**)

Figure 6. The calculated probability density for a decaying particle of mass $M = 25.0$. (**a**) The left panel shows the density evaluated on the entire discretized probability space. (**b**) The probability density of "real" data. (**c**) A division of the probability space into three subspaces, in order to isolate particular decays.

These results show that we can safely assume that the $2NN$ algorithm successfully recognizes all the decay modes of the particle that initiates a jet. To quantify the difference between the calculated probability density and the real probability density, we use the KL-divergence.

Figure 7 shows the dependence of the KL divergence on the iteration of the $2NN$ algorithm. First, we observe an initial steep decrease in the value of the divergence. Large variations in divergence value are observed later. This is an indicator that the approximate probability density is found relatively quickly—after a few hundred iterations. As the algorithm decreases the width of the peaks found in the probability distribution, the KL divergence becomes very sensitive to small variations in the location of these peaks and can therefore vary by a large relative amount.

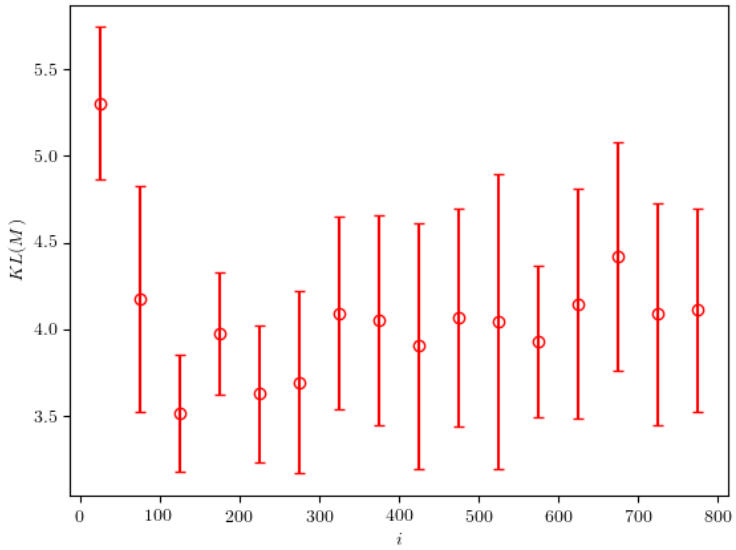
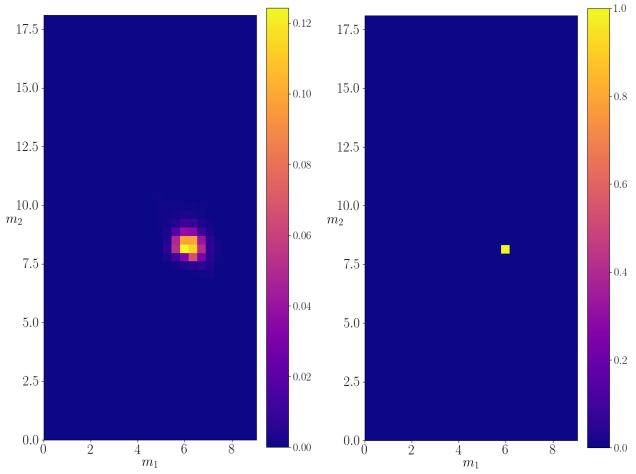


Figure 7. The KL divergence between the calculated and the real probability densities, evaluated in the case of particle of mass $M = 25.0$. The presented results are averaged over 50 iteration intervals. The error bars represent the standard deviation calculated on the same intervals.

3.2. Mother Particle with Mass M = 18.1

A similar analysis is performed for the particle with mass $M = 18.1$. The calculated probability density is shown in Figure 8 compared to the expected probability density. In this case, only one decay is allowed, so a division into probability subspaces is not necessary, as was for the case when $M = 25.0$. The calculated mean and the variance of the shown probability density are $(m_1, m_2) = (5.9 \pm 0.4, 8.2 \pm 0.6)$. In this case, just as in the former, the calculated values closely agree with the only possible decay, in which the mother particle decays into two particles of masses 6.1 and 8.4. Furthermore, just as in the previous subsection, the obtained result is very precise. Therefore, the algorithm can successfully find hidden resonances, as well as recognize the decay channels, without ever seeing them in the final state in the "real" dataset.

The calculated KL divergence in the case of particle with mass $M = 18.1$ decreases over time in a very smooth manner, as can be seen in Figure 9. We believe this could be due to the simpler expected probability density, which the algorithm manages to find very quickly.



(**a**) (**b**)

Figure 8. The calculated probability density for a decaying particle of mass $M = 18.1$. (**a**) The calculated density evaluated on the entire discretized probability space. (**b**) The probability density of "real" data.

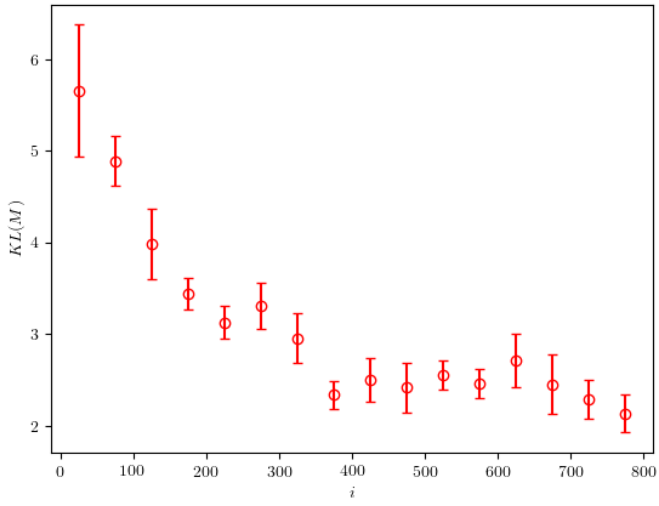
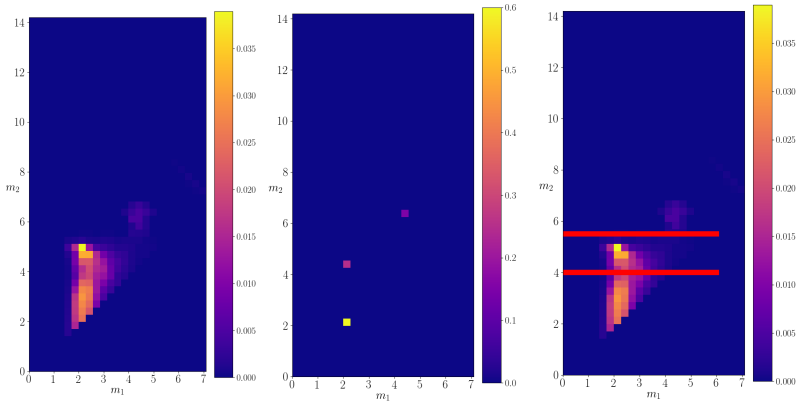


Figure 9. The KL-divergence between the calculated and the real probability densities, evaluated in the case of particle of mass $M = 18.1$. The presented results are averaged over 50-iteration intervals. The error bars represent the standard deviation calculated on the same intervals.

3.3. Mother Particle with Mass M = 14.2

Figure 10 shows the $2d$-probability density for the decaying particle of mass $M = 14.2$. In this case, we can identify three possible decay channels, which are not as clearly separated as the channels in the previous subsections. Similar to the case of decaying particle of mass $M = 25.0$, we divided the probability space into three subspaces, each of which covered one of the possible decays. In this case, the three subspaces cover areas where $m_2 \leq 4.0$, $4.0 < m_2 \leq 5.5$, and $m_2 > 5.5$. The mean values of the probability density on each of the subspaces are $(m_1, m2) = (2.4 \pm 0.5, 2.9 \pm 0.7)$, $(m_1, m_2) = (2.7 \pm 0.7, 4.3 \pm 0.3)$, and $(m_1, m_2) = (4.4 \pm 0.4, 6.2 \pm 0.3)$, respectively. The allowed decays of a mother particle with mass $M = 14.2$ in the "real" data are into channels with masses $(1.9, 1.9)$, $(1.9, 4.4)$, and $(4.4, 6.2)$, which agree with the calculated results. However, in this case the calculations show higher variance, especially for decays where one of the products is a particle with mass 1.9. The total probabilities of decay in each of the subspaces are 0.89, 0.05, and 0.06, respectively. The relative probabilities of decay channels into particles with masses (4.4, 6.1) and (1.9, 4.4) are approximately the same as expected. However, the algorithm predicts more decays in the channel (1.9, 1.9) than expected. The KL divergence shows a steady decrease with occasional spikes, as shown on Figure 11.



(**a**) (**b**) (**c**)

Figure 10. The calculated probability density for a decaying particle of mass $M = 14.2$. (**a**) The left panel shows the density evaluated on the entire discretized probability space. (**b**) The probability density of "real" data. (**c**) A division of the probability space into three subspaces, in order to isolate particular decays.

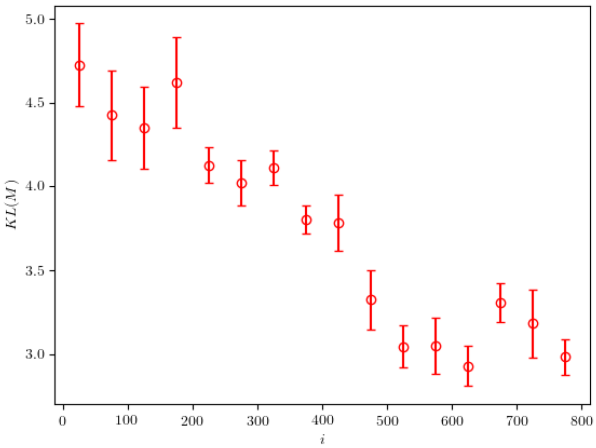


Figure 11. KL divergence between calculated and real probability density evaluated for the $M = 14.2$. Results are averaged over the intervals of 50 iteration. Error bars represent standard deviation on the same interval.

3.4. Mother Particle with Mass M = 1.9

The last probability density we analyze is the probability density for the mother particle with mass $M = 1.9$. Figure 12 shows the calculated probability density. It can be seen that one of the decay modes present in the "real" data, namely, when the particle decays in the $(0.1, 0.1)$ channel, is not recognized by the algorithm, but the decay mode when the particle decays in the $(0.1, 1.3)$ channel is visible. If we isolate given decay as shown in the right panel of Figure 12, we get a mean value of $(m_1, m_2) = (0.14 \pm 0.09, 1.27 \pm 0.09)$, which agrees with the expected decay. We also observe significant decay probabilities along the line $m_1 + m_2 = 1.9$. The decays that correspond to the points on this line in effect create particles with zero momentum in the rest frame of the mother particle. In the lab frame this corresponds to the daughter particles flying off in the same direction as the mother particle. As they reach the detector in the same time, they are registered as one particle of total mass $M = 1.9$. Thus, we can conclude that the probabilities on this line have to add up to the total probability of the mother particle not decaying. The calculated probabilities in the case of no decay and in the case when decaying into particles with masses $(0.1, 1.3)$ are 0.71 and 0.29, respectively. We note that relative probabilities are not correct, but two of the three decay modes are still recognized by the algorithm. The KL-divergence in this case cannot produce reasonable results, simply because of multiple points in the (m_1, m_2) phase space which produce the same decay and is therefore omitted from the analysis. We summarize the obtained results for different masses in Table 2.

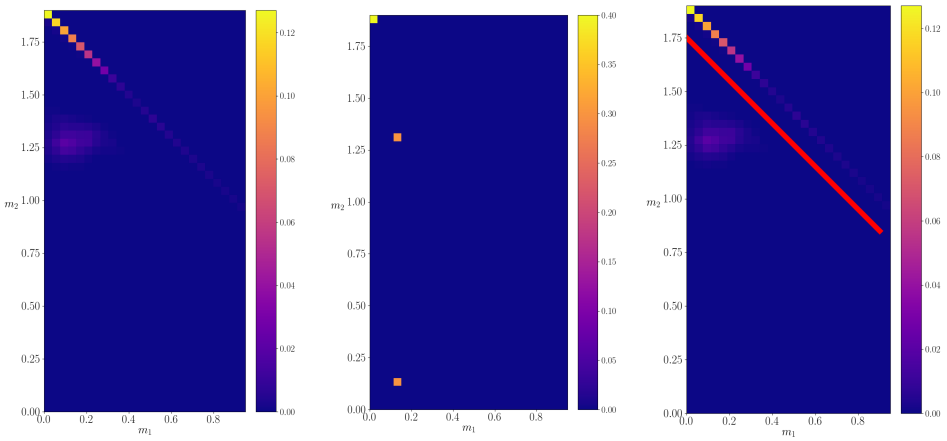


Figure 12. The calculated probability density for a decaying particle of mass $M = 1.9$. (**a**) The left panel shows the density evaluated on the entire discretized probability space. (**b**) The probability density of "real" data. (**c**) A division of the probability space into three subspaces, in order to isolate particular decays.

3.5. The Accuracy of the Classifier

The accuracy of the classifier is defined as the fraction of correctly "guessed" samples on a given dataset. The criterion used for guessing is checking whether the output of the classifier, C_{NN}, is greater than 0.5. The accuracy can indirectly indicate how distinguishable are some two datasets. In our algorithm, after starting from a test probability density, we approach the real probability density with increasing iteration number, so we can expect that the two jet datasets, the "real" and the "test" dataset, are less and less distinguishable over time. In Figure 13, we show the accuracy of the classifier in dependence on the iteration number.

Table 2. The characteristics of the reconstructed decay channels for decaying particles with masses M = 25, 14.2 and 1.9. The decay channel for the particle with mass 1.9 which was not reconstructed is not given here.

particle mass	25		25		25	
decay masses	18.1	6.1	14.2	8.4	4.4	4.4
reconstructed masses	18.1 ± 0.5	6.1 ± 0.5	14.0 ± 0.7	8.4 ± 0.7	4.8 ± 0.2	4.6 ± 0.2
channel probability	0.5		0.4		0.1	
reconstructed probability	0.48		0.47		0.05	
particle mass	18.1		14.2		14.2	
decay masses	8.4	6.1	6.1	4.4	4.4	1.9
reconstructed masses	8.2 ± 0.6	5.9 ± 0.4	6.2 ± 0.3	4.4 ± 0.4	4.3 ± 0.3	2.7 ± 0.7
channel probability	1		0.15		0.25	
reconstructed probability	1		0.05		0.06	
particle mass	14.2		1.9		1.9	
decay masses	1.9	1.9	1.3	0.1	no decay	
reconstructed masses	2.4 ± 0.5	2.9 ± 0.7	1.27 ± 0.09	0.14 ± 0.09	no decay	
channel probability	0.6		0.3		0.4	
reconstructed probability	0.89		0.29		0.71	

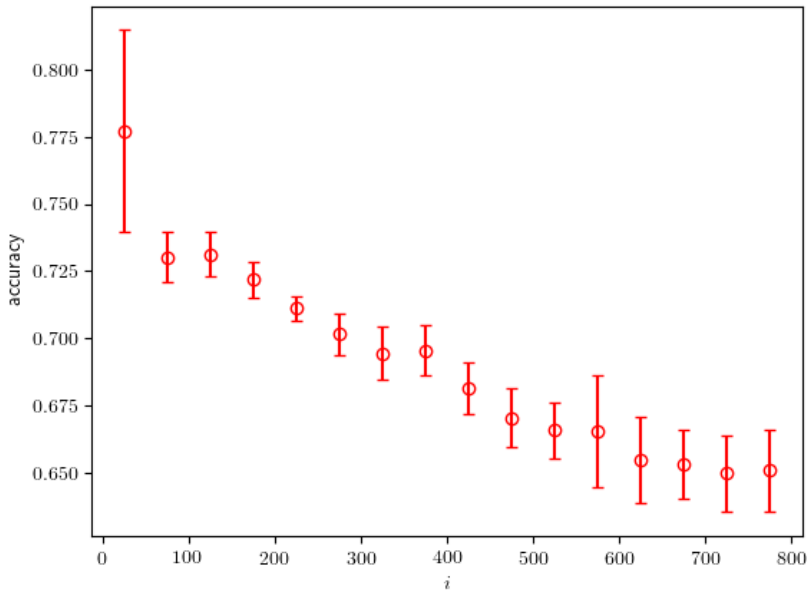


Figure 13. The calculated accuracy of the classifier in dependence on the iteration number.

After an initially high value, the accuracy decreases with growing iteration number, which demonstrates that the test dataset becomes increasingly similar to the real dataset. Ideally, the datasets are no longer distinguishable by a given classifier if the evaluated accuracy reaches 0.5. Therefore, we can use the evaluated accuracy of the classifier as a criterion for stopping the algorithm. Other measures can also be used as the stopping criterion such as the loss value of the classifier or the area under receiver operating characteristic (ROC) curve of the classifier. In this work, the algorithm is stopped after the accuracy reaches a value of 0.65, because we did not see any significant decrease in the accuracy once it reached this value. An accuracy value of 0.65 clearly shows that the classifier is capable of further discriminating between the two datasets. This is explained by the fact that the

neural network f and its hyperparameters are not fully optimized. For the algorithm to perform better, we need to optimize the neural network f and possibly improve the architecture for the selected task.

4. Discussion

In this work, we propose a method for calculating underlying probability distributions in particle decays, using only the data that can be collected in a real-world physical system. First, we developed an artificial physical system based on a part of the QCD fragmentation process. Next, we present the core part of the method: the $2NN$ algorithm, which we described in detail. The algorithm performs very well when tested on the developed physical system. It accurately predicts most of the hidden resonant particles, as well as their decay channels, which can occur in the evolution of jets. The energy spectra of the particles in the final state can also be accurately reproduced. Although tested only on the developed artificial physical system, we believe that the method is general enough to be applicable to real-world physical systems, such as collisions of high-energy particles, with some modifications. For example, we hope that this method can in the future prove helpful in measuring the fragmentation functions of quarks and gluons. Furthermore, one could employ such a method in the search for supersymmetric particles of unknown masses, or in measuring the branching ratios of known decays.

The $2NN$ algorithm does not specify the exact architecture of the neural networks, nor the representation of the data used. Furthermore, the classifier does not need to be a neural network—it can be any machine learning technique which maximizes likelihood. Although the algorithm has a Generative Adversarial Netowrk (GAN)-like structure, it converges readily and does not show usual issues associated with GANs, such as mode collapse or vanishing gradients. The downside of the presented algorithm are high computational requirements. Continuous probability distributions, which we expect to occur in nature, are approximated by discrete probability distributions. In quest for higher precision and a better description of reality, one always aims to increase the resolution of discrete steps, but this carries a high computational cost. Furthermore, the used neural networks are not fully optimized, which slows down the convergence of the algorithm. In conclusion, in order to cut down computational costs, a more thorough analysis of convergence is needed to achieve better performance.

In future work we hope to make the method even more general and thus even more applicable to real-world physical systems. As it stands, the method approximates a part of the QCD decay tree, covering $1 \to 2$ decays. Even though we prove that some of the decay characteristics can be recovered from the final state without prior knowledge of the underlying processes, the method still doesn't lend itself for use on general QCD data. To remedy that, we want to introduce angle dependent probability distributions, which can be retrieved from some detector data. We would also like to investigate the possibility of including other decay modes, such as $1 \to 3$ type decays. Finally, we plan to include other processes that appear naturally in QCD, such as $2 \to 2$ and $2 \to 3$ type interactions, as well as allow for particle decay widths.

Author Contributions: Conceptualization, M.J., N.P. and I.J.; Investigation, N.P. and I.J.; Methodology, M.J.; Writing—original draft, M.J., N.P. and I.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Croatian science foundation grant IP-2018-01-4108 "Demystifying Two Particle Correlations in pp collisions with the upgraded Time Projection Chamber".

Data Availability Statement: Not applicable.

Acknowledgments: We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro P6000 graphics processing unit used for this research.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

QCD	Quantum Chromodynamics
LHC	Large Hadron Collider
CERN	Conseil Européen pour la Recherche Nucléaire
2NN	2 Neural Networks
ROC	Receiver Operating Characteristic
GAN	Generative Adversarial Network
CNN	Convolutional Neural Network

Appendix A. Description of the Neural Networks

The classifier used to recover the underlying probability distributions is a feed-forward convolutional neural network (CNN), already used in [8]. In that work, the classifier was used for a different purpose, but its architecture proved to be suitable for the task at hand. The architecture of the used CNN is schematically shown in Figure A1. It consists of a block of layers, repeated four times, followed by three dense layers consisting of 20, 10, and 1 unit, respectively. A ReLu activation function is used in each layer, except for the last one, where a sigmoid function is used. The layer block consists of a 2-dimensional convolutional layer (with 32 filters and a (3,3) kernel), a MaxPooling layer, a batch normalization layer and a dropout layer. The training of the classifier is performed by minimizing the binary cross entropy loss [13]. The AdaM optimizer is used to optimize the weights of the CNN [11].

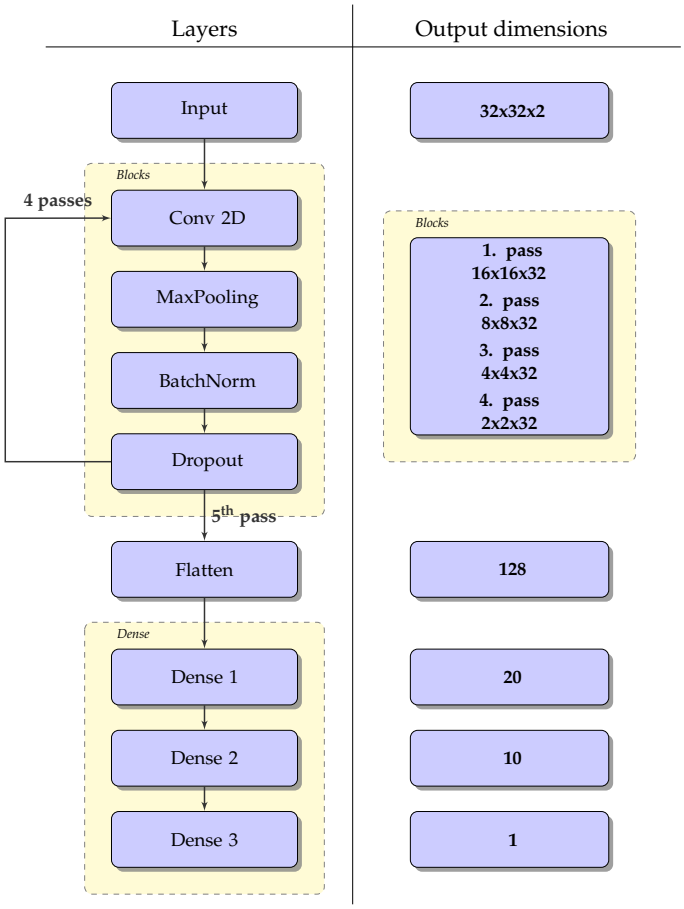


Figure A1. The architecture of the convolutional neural network as described in the text. The output dimensions of each layer are given on the right side of the panel. The Blocks layer goes through 4 passes.

The neural network f is a feed-forward neural network consisting of five independent completely connected layers. It takes on a vector of three values as the input and outputs a single value. Each of the hidden layers consist of 100 neurons apart from the last one, which is a single neuron. The activation function used in all the layers, apart from the last one, is a ReLu activation function. The last layer has no activation function. The network was optimized using the ADAM algorithm.

References

1. Cacciari, M.; Salam, G.P.; Soyez, G. The anti-kt jet clustering algorithm. *J. High Energy Phys.* **2008**, *4*, 63. [CrossRef]
2. Sjostrand, T.; Mrenna, S.; Skands, P. PYTHIA 6.4 Physics and Manual. *arXiv* **2006**, arXiv:hep-ph/0603175.
3. Guest, D.; Cranmer, K.; Whiteson, D. Deep Learning and Its Application to LHC Physics. *Annu. Rev. Nucl. Part. Sci.* **2018**, *68*, 161–181. [CrossRef]
4. Andreassen, A.; Feige, I.; Frye, C.; Schwartz, M.D. Junipr: A framework for unsupervised machine learning in particle physics. *Eur. Phys. J. C* **2019**, *79*, 1–24. [CrossRef]
5. Chollet, F. Keras. 2015. Available online: https://keras.io (accessed on 1 January 2021).
6. Streit, R.L. A neural network for optimum Neyman-Pearson classification. In Proceedings of the 1990 IJCNN International Joint Conference on Neural Networks, San Diego, CA, USA, 17–21 June 1990; Volume 1, pp. 685–690.
7. Tong, X.; Feng, Y.; Li, J.J. Neyman-Pearson classification algorithms and NP receiver operating characteristics. *Sci. Adv.* **2018**, *4*. [CrossRef] [PubMed]
8. Jercic, M.; Poljak, N. Exploring the Possibility of a Recovery of Physics Process Properties from a Neural Network Model. *Entropy* **2020**, *22*, 994. [CrossRef] [PubMed]
9. Neyman, J.; Pearson, E.S. On the problem of the most efficient tests of statistical hypotheses. *Phil. Trans. R. Soc. Lond. A.* **1933**, *231*, 694–706.
10. Bishop, C.M. *Neural Networks for Pattern Recognition*; Oxford University Press: Oxford, UK, 1995.
11. Kingma, D.; Ba, J. A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980.
12. Kullback, S.; Leibler, R.A. On Information and Sufficiency. *Ann. Math. Stat.* **1951**, *22*, 79–86. [CrossRef]
13. Nielsen, M.A. Neural Networks and Deep Learning. 2015. Available online: http://neuralnetworksanddeeplearning.com/ (accessed on 1 January 2021).