Eur. Phys. J. C (2019) 79:102
https://doi.org/10.1140/epjc/s10052-019-6607-9

THE EUROPEAN PHYSICAL JOURNAL C

Regular Article - Theoretical Physics

JUNIPR: a framework for unsupervised machine learning in particle physics

Anders Andreassen[1,a], **Ilya Feige**[2,b], **Christopher Frye**[1,c], **Matthew D. Schwartz**[1,d]

[1] Department of Physics, Harvard University, Cambridge, MA 02138, USA
[2] ASI Data Science, 54 Welbeck Street, London W1G 9XS, UK

Received: 19 September 2018 / Accepted: 17 January 2019 / Published online: 1 February 2019

Abstract In applications of machine learning to particle physics, a persistent challenge is how to go beyond discrimination to learn about the underlying physics. To this end, a powerful tool would be a framework for unsupervised learning, where the machine learns the intricate high-dimensional contours of the data upon which it is trained, without reference to pre-established labels. In order to approach such a complex task, an unsupervised network must be structured intelligently, based on a qualitative understanding of the data. In this paper, we scaffold the neural network's architecture around a leading-order model of the physics underlying the data. In addition to making unsupervised learning tractable, this design actually alleviates existing tensions between performance and interpretability. We call the framework JUNIPR: "Jets from UNsupervised Interpretable PRobabilistic models". In this approach, the set of particle momenta composing a jet are clustered into a binary tree that the neural network examines sequentially. Training is unsupervised and unrestricted: the network could decide that the data bears little correspondence to the chosen tree structure. However, when there is a correspondence, the network's output along the tree has a direct physical interpretation. JUNIPR models can perform discrimination tasks, through the statistically optimal likelihood-ratio test, and they permit visualizations of discrimination power at each branching in a jet's tree. Additionally, JUNIPR models provide a probability distribution from which events can be drawn, providing a data-driven Monte Carlo generator. As a third application, JUNIPR models can reweight events from one (e.g. simulated) data set to agree with distributions from another (e.g. experimental) data set.

[a] e-mail: anders@physics.harvard.edu
[b] e-mail: ilya@asidatascience.com
[c] e-mail: frye@physics.harvard.edu
[d] e-mail: schwartz@physics.harvard.edu

Contents

1 Introduction

Machine learning models based on deep neural networks have revolutionized information processing over the last decade. Such models can recognize objects in images [1–3], perform language translation [4,5], transcribe spoken language [6], and even speak written text [7] at approaching human level. The truly revolutionary aspect of this progress is the generality of deep neural networks: a broad diversity of network architectures can be created from basic building blocks that allow for efficient calculation of gradients via back propagation, and thus efficient optimization through stochastic gradient descent [8]. These methods are arbitrarily expressive and can model extremely high dimensional data.

The architecture of a neural network should be designed to process information efficiently, from the input data all the way through to the network's final output. Indeed, it empirically seems to be the case that networks that process information evenly layer-by-layer perform very well. One example of this empirical result is that deep convolutional networks for image processing seem to perform sequentially more abstract operations as a function of depth [1]. Similarly, recurrent networks perform well on time series data, as their recurrent layers naturally describe step-by-step evolution in time [9].

The power and generality of deep neural networks has been leveraged across the sciences, and in particular in particle physics. The simplest architecture explored has been the fully-connected network, which has successfully been applied in a wide variety of contexts, such as in identifying and splitting clusters from multiple particles in the pixel detector [10], in b-tagging [11], and in τ-identification [12]. In these basic applications, the neural network optimizes its use of some finite number of relevant physical observables for the task at hand.[1] One drawback of such an approach is that the neural network is limited by the observables it is given. In fact, for these applications, other multivariate methods such as boosted decision trees often have comparable performance using the same inputs, but train faster and can be less sensitive to noise [17,18].

As an alternative to feeding a neural network a set of motivated observables, one can feed it raw information. By doing so, one allows the network to take advantage of useful features that physicists have yet to discover. One way of preprocessing the raw data in a fairly unbiased way is through the use of jet images, which contain as pixel intensities the energy deposited by jet constituents in calorimeter cells [19]. Jet images invite the use of techniques from image recognition to discriminate jets of different origins. In [19], the pixel intensities in the two-dimensional jet image were combined into a vector, and a Fisher linear discriminant was then used to find a plane in the high-dimensional space that maximally separates two different jet classes. Treating a two-dimensional jet image as an unstructured collection of pixel intensities, however, ignores the spatial locality of the problem, i.e. that neighboring pixels should have related intensities. Convolutional neural networks (CNNs), which boast reduced complexity by leveraging this spatially local structure, have since been adopted instead, and they generally outperform fully-connected networks due to their efficient feature detection. In the first applications of CNNs to jet images, on boosted W detection [20] and quark/gluon discrimination [21], it was indeed found that simple CNNs could generally outperform previous techniques. Since then, a number of studies have aimed to optimize various discrimination tasks using CNNs [22–27].

While the two-dimensional detector image acts as a natural representation of a jet, especially from an experimental standpoint, the 4-momenta of individual jet constituents provide a more fundamental representation for the input to a neural network. One complication in transitioning from the jet image to its list of momenta is that, while the image is a fixed-size representation, the list of momenta will have different sizes for different jets. To avoid this problem, one could truncate the list of momenta in the jet to a fixed size, and zero-pad jets smaller than this size [28]. Alternatively, there are network architectures, namely recursive (RecNNs) and recurrent neural networks (RNNs), that handle variable length inputs naturally. With such methods, one also has the freedom to choose the order in which constituent momenta are fed into the network. In [29], a RecNN was used to build a fixed-size representation of the jet, and the authors explored various ways of ordering the momenta as input to the network: by jet clustering algorithms, by transverse momentum, and randomly. The resulting representation of the jet was then fed to a fully-connected neural network for boosted W tagging. RecNNs and RNNs have also been used in similar ways for quark/gluon discrimination [30], top tagging [31], and jet charge [32]. See also [33,34] for jet flavor classification using tracks.

To date, the majority of applications of machine learning to particle physics employ supervised machine learning techniques. Supervised learning is the optimization of a model to map input to output based on labeled input-output pairs in the training data. These training examples are typically simulated by Monte Carlo generators, in which case the labels come from the underlying physical processes being generated. Most of the classification studies mentioned above employ this style of supervised learning, and similar techniques have also been utilized for regression tasks such as pileup subtraction [22]. Alternatively, training data can be organized in mixed samples, each containing different proportions of the different underlying processes. In this case, labels correspond to the mixed samples, and learning is referred to as weakly supervised. While full and weak supervision are very similar as computational techniques, the distinction is exceptionally important in particle physics, where the underlying physical processes are unobservable in real collider data. Early studies of weakly supervised learning in particle physics show very promising results: performance comparable to fully supervised methods was found both with low-dimensional inputs [35,36] (a few physical observables) and with very high-dimensional inputs [37] (jet images).

With supervised learning, there is a notion of absolute accuracy: since every training example is labeled with the desired output, the network predicts this output either correctly or incorrectly. This is in contrast to *unsupervised learn-*

[1] For recent work on constructing a basis for neural network inputs, see [13–15], and see [16] for a linear approach that does not require neural network methods.

ing, where the machine learns underlying structure that is unlabeled in the training data. Without output-labeled training examples, there is no notion of absolute accuracy. Several recent studies have employed unsupervised learning techniques in particle physics. In [38], borrowing concepts from topic modelling in text documents, the authors extract observable distributions of underlying quark and gluon jets from two mixed samples. In [39–41], generative adversarial networks (GANs) are used to efficiently generate realistic jet images and calorimeter showers.

In this work, we explore another approach to unsupervised machine learning in particle physics, in which a deep neural network learns to compute the relative differential cross section of each data point under consideration, or equivalently, the probability distribution generating the data. The power of having access to the probability distribution underlying the data should not be underestimated. For example, likelihood ratios would provide optimal discriminants [42], and sampling from the probability distribution would provide completely data-driven simulations.

In this paper, we introduce a framework named JUNIPR: "Jets from UNsupervised Interpretable PRobabilistic models". We also present a basic implementation of this framework using a deep neural network. This network directly computes the general probability distribution underlying particle collider data using unsupervised learning.

The task of learning the probability distribution underlying collider data comes with challenges due to the complexity of the data. Some past studies have aimed to process collider information efficiently by using neural network architectures inspired by physics techniques already in use [29–33,43]. In this paper, we take this idea one step further. We scaffold the neural network architecture around a leading-order description of the physics underlying the data, from first input all the way to final output. Specifically, we base the JUNIPR framework on algorithmic jet clustering trees. The tree structure is used, both in processing input information, and in decomposing the network's output. In particular, JUNIPR's output is organized into meaningful probabilities attached to individual nodes in a jet's clustering tree. In addition to reducing the complexity and increasing the efficiency of the corresponding neural network, this approach also forces the machine to speak a language familiar to physicists, thus enabling its users to interpret the underlying physics it has learned. Indeed, one common downside associated with machine learning techniques in physics is that, though they provide powerful methods to accomplish the tasks learned in training, they do little to clarify the underlying physics that underpins their success. Our approach minimizes this downside.

Let us elaborate on the tree-based architecture used for JUNIPR's implementation. In particle physics, events at colliders are dominated by the production of collimated collections of particles known as jets. The origin of jets and many of their properties can be understood through the fundamental theory of strong interactions, quantum chromodynamics (QCD). One insight from QCD is that jets have an inherently fractal structure, inherited from the approximate scale invariance of the fundamental theory. The fractal structure is made precise through the notion of factorization, which states that the dynamics in QCD stratify according to soft, collinear, and hard physics [44–48], with each sector being separately scale invariant. To capture this structure efficiently in JUNIPR, we use a kind of factorized architecture, with a dense network to describe local branchings (well-suited for collinear factorization), and a global RNN superstructure general enough to encode soft coherence and any factorization-violating effects.

One might naively expect this setup to require knowledge of the sequence of splittings that created the jet. Although there is a sequence of splittings in parton-shower simulations, the splittings are only a semi-classical approximation used to model the intensely complex and essentially incalculable distribution of final state particles. Real data is not labelled with any such sequence. In fact, there are many possible sequences which could produce the same event, and the cross section for the event is given by the square of the quantum mechanical sum of all such amplitudes, including effects of virtual particles. A proxy for this fictitious splitting history is a clustering history that can be constructed in a deterministic way using a jet-clustering algorithm, such as the k_t algorithm [49,50] or the Cambridge/Aachen (C/A) algorithm [51,52]. There is no *correct* algorithm: each is just a different way to process the momenta in an event. Indeed, there seems to be useful information in the multiple different ways that the same event can be clustered [53–55]. Any of these algorithms, or any algorithm at all that encodes the momenta of an event into a binary tree, can be used to scaffold a neural network in the JUNIPR approach.

For practical purposes, JUNIPR is implemented with respect to a fixed jet clustering algorithm. Without a fixed algorithm, the probability of the final-state particles constructed through $1 \to 2$ branchings would require marginalization over all possible clustering histories – an extremely onerous computational task. In principle, fixing the algorithm used to implement JUNIPR should be inconsequential for its output, namely the probability distribution over final-state momenta, as these momenta are independent of clustering algorithm. To reiterate, the JUNIPR approach does not require the chosen clustering algorithm to agree with the underlying data-generation process; this is demonstrated in Sects. 5.2 and 5.3 below. On the other hand, the *sequence* of probabilities assigned to each branching in a clustering tree certainly depends on the algorithm used to define the tree. For example, the same final probability $P = 10^{-22}$ could be reached with one clustering algorithm through the sequence $P = 10^{-5} \cdot 10^{-6} \cdot 10^{-8} \cdot 10^{-3}$, or with another algorithm

through $P = 10^{-15} \cdot 10^{-2} \cdot 10^{-1} \cdot 10^{-4}$. The key idea is that, if an algorithm is chosen which does correspond to a semi-classical parton shower, the resulting sequence of probabilities may be understandable. This provides avenues for users to interpret what physics the machine learns, and we expect that dissecting JUNIPR will be useful in such cases. We will demonstrate this throughout the paper.

It is worth emphasizing one fundamental aspect of our approach for clarity. The JUNIPR framework yields a *probabilistic model*, not a generative model. The probabilistic model allows us to directly compute the probability density of an individual jet, as defined by its set of constituent particle momenta. To be precise, this is the probability density for those particular momenta to arise in an event, conditioned on the event selection criteria used to select the training data. As a complementary example of this, shower deconstruction [56,57] provides a theory-driven approach to probabilistic modeling in particle physics, in which probabilities are calculated using QCD rather than a neural network. In contrast, a generative model would output an example jet, taking random noise as input to seed the generation process. Given a distribution of input seeds, the jets output from a generative model should follow the same distribution as the training data. While this means that the probability distribution underlying the data is internally encoded in a generative model, this underlying distribution is hidden from the user. Examples of generative models in particle physics include Monte Carlo event generators and, more recently, GANs used to generate jet images and detector simulations [39–41].

The direct access to the probability distribution that is enabled by a probabilistic model comes with several advantages. If two different probabilistic models are trained on two different samples of jets, they can be used to compute likelihood ratios that distinguish between the two samples. Likelihood ratios provide theoretically optimal discriminants [42], which is indeed a major motivation for JUNIPR's probabilistic approach. One can also sample from a probabilistic model in order to generate events, though generative models are better-suited for this application [39–41]. In addition, one can use a probabilistic model to reweight events generated by an imperfect simulator, so that the reweighted events properly agree with data.

In this paper, as a proof-of-concept, we use simulated e^+e^- data to train a basic implementation of the JUNIPR framework described above. We have not yet attempted to optimize all of this implementation's hyperparameters; however, we do find that a very simple architecture with no fine tuning is adequate. This is confirmed by its impressive discrimination power and its effective predictivity for a broad class of observables, but more rigorous testing is needed to determine whether this approach can provide state-of-the-art results on the most pressing physics problems.

The general probabilistic model, its motivation, and a specific neural network implementation of it are discussed in Sect. 2. A comprehensive discussion of training the model, including the data used and potential subtleties in extending the model are covered in Sect. 3. Results on discrimination, generation, and reweighting are presented in Sect. 4. We provide robustness tests and some conceptually interesting results related to factorization in Sect. 5, including the counterintuitive anti-k_t shower generator. There are many ways to generalize our approach, as well as many applications that we do not fully explore in this work. We leave a discussion of some of these possible extensions to Sect. 6, where we conclude.

2 Unsupervised learning in jet physics

To establish the framework clearly and generally, Sect. 2.1 begins by describing JUNIPR as a general probabilistic model, independent of the specific parametric form taken by the various functions it involves. From this perspective, such a probabilistic model could be implemented in many different ways. Section 2.2 then describes the particular neural network implementation of JUNIPR used in this paper, which has a simple but QCD-customized architecture and minimal hyperparameter tuning.

2.1 General probabilistic model

Consider a set of final-state 4-momenta $p_1, \ldots, p_n$ that we hereafter refer to as "the jet". JUNIPR computes the probability density $P_{\text{jet}}(\{p_1, \ldots, p_n\})$ of this set of momenta arising in an event, assuming the event selection criteria used to select the training data. This probability distribution is normalized so that, abstractly,

$$\sum_{n=1}^{\infty} \int d^4p_1 \cdots d^4p_n \, P_{\text{jet}}(\{p_1, \ldots p_n\}) = 1, \tag{2.1}$$

where the integral extends over the physical region of phase space. (In practice, in implementing JUNIPR we discretized the phase space into cells and assigned a measure of unity to each discrete cell. This results in P_{jet} being a discrete cell-size-dependent probability distribution, but this choice is conceptually unimportant here.) A high-level schematic of JUNIPR is shown in Fig. 1, which emphasizes that the model does not attempt to learn the quantum-mechanical evolution that created the jet, but only meaningfully predicts the likelihood of its final-state momenta.

An unstructured model of the above form would ignore the fact that we know jet evolution is well-described by a semi-classical sequence of $1 \to 2$ splittings, due to factor-

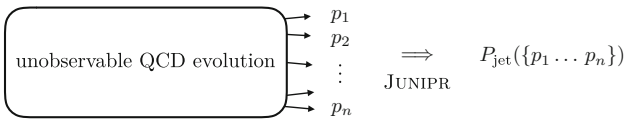

Fig. 1 JUNIPR predicts the probability density $P_{\text{jet}}(\{p_1, \dots, p_n\})$ of finding a given set of momenta $\{p_1, \dots, p_n\}$ in a jet, conditioned on the jet selection criteria used to select the training data. No assumptions are made about the underlying quantum-mechanical processes that generated the jet

ization theorems [44–48]. A model that ignores factorization would be much more opaque to interpretation, and have many more parameters than needed due to its unnecessary neutrality. Thus, we propose a model that describes a given configuration of final-state momenta using sequential $1 \to 2$ splittings. Such a sequence is defined by a jet clustering algorithm, which assigns a clustering tree to any set of final-state momenta, so that a sequential decomposition of the probability distribution can be performed without loss of generality. We imagine fixing a specific algorithm to define the trees, so that there is no need to marginalize over all possible trees in computing a probability, a computation that would be intractable. While a deterministic clustering algorithm cannot directly describe the underlying quantum-mechanical parton evolution, that is not the goal for this model. With the algorithm set, the model as shown in Fig. 1 becomes that shown in Fig. 2.

We will now formalize this discussion into explicit equations. For the rest of this section we assume that the clustering tree is determined by a fixed jet algorithm (e.g. any of the generalized k_t algorithms [58,59]). The particular algorithm chosen is theoretically inconsequential to the model, as the same probability distribution over final states will be learned for any choice. Practically speaking, however, certain algorithms may have advantages over others. We will discuss the choice of clustering algorithm further in Sects. 5.2 and 5.3.

The application of a clustering algorithm on the jet constituents $p_1, \dots, p_n$ defines a sequence of "intermediate states" $k_1^{(t)}, \dots, k_t^{(t)}$. Here the superscript $t = 1, \dots, n$ labels the intermediate state after the $(t-1)$th branching in the tree (where counting starts at 1) and the subscript $i = 1, \dots, n$ enumerates momenta in that state. To be explicit,

- the "initial state" consists of a single momentum: $k_1^{(1)} = p_1 + \cdots + p_n$;
- at subsequent steps $\{k_1^{(t)}, \dots, k_t^{(t)}\}$ is gotten from $\{k_1^{(t-1)}, \dots, k_{t-1}^{(t-1)}\}$ by a single momentum-conserving $1 \to 2$ branching;
- after the final branching, the state is the physical jet: $\{k_1^{(n)}, \dots, k_n^{(n)}\} = \{p_1, \dots, p_n\}$.

In this notation, the probability of the jet (as shown in Fig. 2) can be written as

$$P_{\text{jet}}(\{p_1, \dots p_n\}) = \left[\prod_{t=1}^{n-1} P_t\left(k_1^{(t+1)}, \dots, k_{t+1}^{(t+1)} \middle| k_1^{(t)}, \dots, k_t^{(t)}\right)\right] \times P_n\left(\text{end} \middle| k_1^{(n)}, \dots, k_n^{(n)}\right). \tag{2.2}$$

Equation (2.2) allows for a natural, sequential description of the jet. However, it obscures the factorization of QCD which predicts an approximately self-similar splitting evolution. Thus we decompose the model further, so that each P_t in Eq. (2.2) is described by a $1 \to 2$ branching function that only indirectly receives information about the rest of the jet. The latter is achieved via an unobserved representation vector $h^{(t)}$ of the global state of the jet at step t. To be explicit, let $k_m^{(t)} \to k_{d_1}^{(t+1)} k_{d_2}^{(t+1)}$ denote the branching of a mother into daughters that achieves the transition from $k_1^{(t)}, \dots, k_t^{(t)}$ to $k_1^{(t+1)}, \dots, k_{t+1}^{(t+1)}$ in the clustering tree. Then we can write

$$\begin{aligned} P_t\left(k_1^{(t+1)}, \dots \middle| k_1^{(t)}, \dots\right) &= P_{\text{end}}\left(0 \middle| h^{(t)}\right) P_{\text{mother}}\left(m^{(t)} \middle| h^{(t)}\right) \\ &\quad \times P_{\text{branch}}\left(k_{d_1}^{(t+1)}, k_{d_2}^{(t+1)} \middle| k_m^{(t)}, h^{(t)}\right) \\ P_n\left(\text{end} \middle| k_1^{(n)}, \dots\right) &= P_{\text{end}}\left(1 \middle| h^{(n)}\right). \end{aligned} \tag{2.3}$$

where $m^{(t)}$ is the mother's discrete index in the tth intermediate state. We thus have a sequential model that at each t step predicts

- $P_{\text{end}}\left(0 \middle| h^{(t)}\right)$: probability over binary values for whether or not the tree ends;
- $P_{\text{mother}}\left(m^{(t)} \middle| h^{(t)}\right)$: probability over $m \in \{1, \dots, t\}$ indexing candidate mother momenta;
- $P_{\text{branch}}\left(k_{d_1}^{(t+1)}, k_{d_2}^{(t+1)} \middle| k_m^{(t)}, h^{(t)}\right)$: probability over possible $k_m \to k_{d_1}, k_{d_2}$ branchings.

Note that we have left the conditioning on end $= 0$ implicit in P_{mother} and P_{branch}, since we will never need to use these

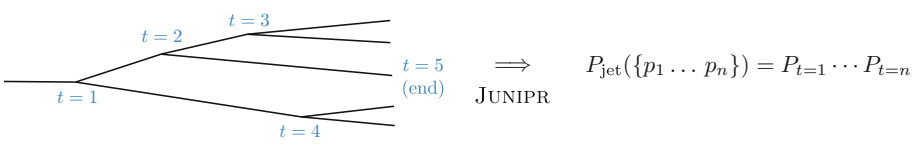

Fig. 2 With any fixed clustering algorithm, the probability distribution over final-state momenta can be decomposed into a product of distributions. Each factor in the product corresponds to a different step in the clustering tree. Subsequent probabilities are conditioned on the outcomes from previous steps, so this decomposition entails no loss of generality

functions when end $= 1$. In the product of Eq. (2.3), each subsequent factor is thus conditioned on the outcomes of previous factors, so that breaking up P_{jet} in this way is without loss of generality. In particular, no assumption has been made about the underlying physical processes that generate the data.

With these choices, we force the hidden representation $h^{(t)}$ to encode all global information about the tree, since it must predict whether the tree ends, which momentum branches next, and the branching pattern. In fact, providing P_{branch} with the momenta that directly participate in the $1 \to 2$ branching means that $h^{(t)}$ only needs to encode global information. We show that the global structure stored in $h^{(t)}$ is crucial for the model to predict the correct branching patterns in Sect. 5.1.

2.2 Neural network implementation

For a neural network based implementation of the model defined by Eqs. (2.2) and (2.3), we use an RNN with hidden state $h^{(t)}$ augmented by dense neural networks for each of the three probability distributions in Eq. (2.3). The recurrent structure of this implementation is shown in Fig. 3, which emphasizes how the RNN's hidden representation $h^{(t)}$ keeps track of the global state of the jet, by sequentially reading in the momenta that branched most recently.

The fact that $h^{(t)}$ learns and remembers the full jet, despite only being shown the two new momenta at step t, is ensured by the tasks for which $h^{(t)}$ is responsible. These are shown in the detailed network diagram of Fig. 4. There one can see that $h^{(t)}$ is the only input into the components of the model that predict when the tree ends and which momentum is next to branch. The domains of the three probability functions in Eq. (2.3) are shown in Fig. 4 as well: P_{end} is defined over the binary set $\mathbb{Z}_2$ corresponding to "end" or "not"; P_{mother} is multinomial over the set $\mathbb{Z}_t$ of candidate mothers; and P_{branch} is defined on the space of possible $1 \to 2$ branchings, which is (a subset of) $\mathbb{R}^4$ by momentum conservation. At each step, the model outputs the full probability distributions, which in mathematical notation are $P_{\text{end}}(\mathbb{Z}_2|h^{(t)})$, $P_{\text{mother}}(\mathbb{Z}_t|h^{(t)})$, and $P_{\text{branch}}(\mathbb{R}^4|k_m^{(t)}, h^{(t)})$.

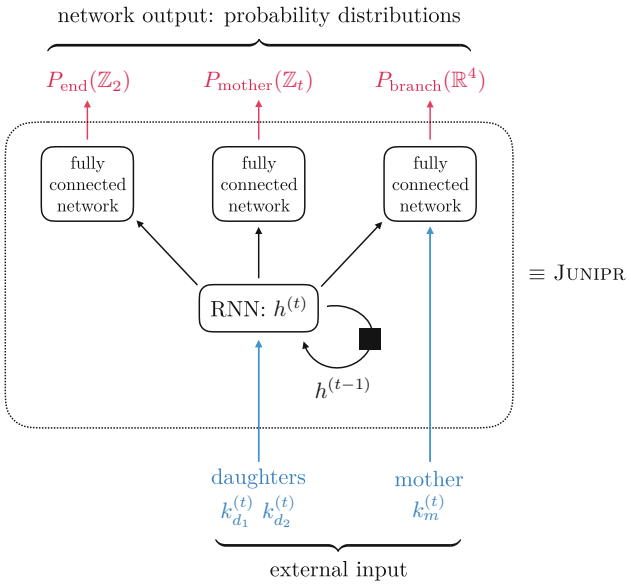

Fig. 4 Neural network implementation of the general probabilistic model proposed in Eqs. (2.2) and (2.3). The network takes as external inputs two daughter momenta and one mother momentum. The global RNN then passes only its representation vector $h^{(t)}$ to each of the dense networks shown. The networks output three full probability distributions, which predict the end of the tree, the next mother to branch, and its daughter momenta

Figures 3 and 4 show how JUNIPR provides a probability distribution at each step t given the momenta emerging from the preceding branching. For clarity, Fig. 5 separately shows how JUNIPR is used to evaluate the full probability density $P_{\text{jet}}(\{p_1, \ldots, p_n\})$ over final-state momenta in a jet. At each step t, the point in $\mathbb{Z}_2$ representing whether the tree ends, the point in $\mathbb{Z}_t$ representing which mother momentum branches, and the point in $\mathbb{R}^4$ representing its daughters are plugged into the probability distributions to obtain the probabilities that should be assigned to the jet under consideration. The product of these three probabilities, taken over all t steps, leads to $P_{\text{jet}}(\{p_1, \ldots, p_n\})$.

Let us now go into detail about the neural network architecture used. We use basic RNN cells [60] with tanh activation,

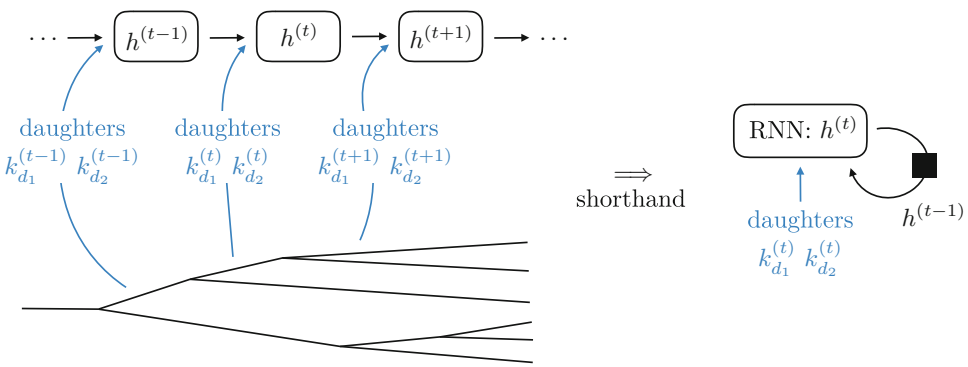

Fig. 3 Information about the clustering tree is embedded in the hidden state $h^{(t)}$ of the RNN. For brevity, this recurrent structure is simplified on the right using a shaded box to indicate stepping from $t-1$ to t. At each step, the next two daughter momenta emerging in the tree and the previous hidden state $h^{(t-1)}$ are inputs to the updated hidden state $h^{(t)}$

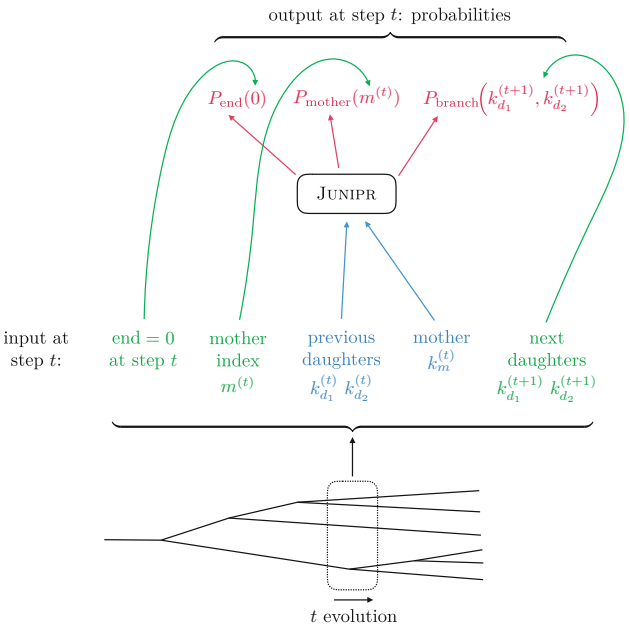

Fig. 5 Using JUNIPR to evaluate the probability density over final-state momenta in a jet. For a given jet and its particular clustering tree, the values associated with the tree ending, which momenta branch, and the emerging daughters are all known and plugged into the probability distributions directly. The probability density of the jet is then the product over the three distributions, over all splitting steps t

$$h^{(t)} = \tanh\left(W \cdot (k_{d_1}^{(t)}, k_{d_2}^{(t)}) + V \cdot h^{(t-1)} + b\right), \tag{2.4}$$

and found that a hidden representation vector $h^{(t)}$ of generic size 100 was sufficient for our needs. We found GRU [61] and LSTM [62] cells to be unnecessarily complex and high-capacity for the tasks carried out in this paper. This is in contrast to language modelling, for which basic RNN cells are underpowered. To see why this might heuristically be expected, note that a sentence containing 20 words is much more complex than a jet containing 20 momenta, because the words in the sentence are ordered, whereas the momenta in the jet are not. This introduces an additional factor of $20! \sim 10^{18}$ to the complexity of language modelling. It is thus reasonable to expect that jet physics will not require all the high-powered tools designed for natural language processing.

For P_{end} we use a fully-connected network with $h^{(t)}$ as input, a single hidden layer of size 100 with ReLU activation, and a sigmoid output layer. We use the same setup for P_{mother}, the only difference being that the output layer is a softmax over the t candidate mother momenta, ordered by energy. These choices are generic and not highly tuned. We found that JUNIPR works well for a very general set of architectures and sizes, so we stick with this simple setup.

For the branching function P_{branch} we must describe the probability distribution over all possible configurations of daughter momenta $k_{d_1}^{(t+1)}, k_{d_2}^{(t+1)}$ consistent with the mother momentum $k_m^{(t)}$. For this system, we use coordinates $x = (z, \theta, \phi, \delta)$ centered around the mother, where z is the energy fraction of the softer daughter, θ (δ) is the opening angle of the softer (harder) daughter, and ϕ specifies the plane in which the branching occurs. See Fig. 6 for a visualization of these coordinates.

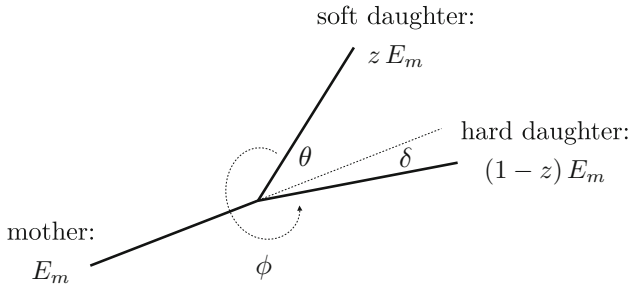

Fig. 6 Local coordinates $x = (z, \theta, \phi, \delta)$ that parameterize the momentum-conserving $1 \to 2$ branching at each step in the clustering tree of a jet

There are two separate approaches one could take to model the branching function P_{branch}. Firstly, the variables x could be treated as discrete, with P_{branch} outputting a softmax probability over discrete cells representing different x values. Secondly, one could treat x as a continuous variable and use an "energy model" of the form $P_{\text{branch}} \sim e^{E(x)}/Z$, where Z is a normalizing partition function. In this work we predominantly adopt the former approach, as it is much faster, and most distributions are insensitive to the discretization of x. However, we do train an energy model to show that models with continuous x are possible, which we discuss in Sect. 3.4.

In the discrete case, we bin the possible values of x into a four-dimensional grid with ten bins per dimension, so that the entire grid has 10^4 cells. For a given value of x, we place a 1 in the bin corresponding to that value, and we place 0's everywhere else. This 1-hot encoding of the possible values of x allows us to use a softmax function at the top layer of the neural network describing P_{branch} (see Fig. 4). Furthermore, we use a dense network with a single hidden layer of size 100 and ReLU activation for P_{branch}, just as we did for P_{end} and P_{mother}. The hidden units in this network receive $h^{(t)}$ as input, as well as the mother momentum $k_m^{(t)}$.

Thus we have a neural network implementation of Eqs. (2.2) and (2.3), with a representation of the evolving global jet state stored in $h^{(t)}$, and with fully-connected networks describing P_{end}, P_{mother}, and P_{branch}. As defined above, the model has a single 10^6 parameter matrix, mapping the branching function's 100 dimensional hidden layer to its 10^4 dimensional output layer, and has 6×10^4 parameters elsewhere. For future studies, we reccomend starting with the same architecture with all the hidden layers having 100 nodes each, and then performing vary the hyperparameters to optimize for the task at hand. One might refer to our

implementation as JUNIPR$_\emptyset$, as one can imagine many alternative implementations within the JUNIPR framework that may prove useful in future applications. We will continue to use the term JUNIPR for brevity, to refer both to the framework and to the basic implementation described here.

3 Training and validation

We now describe how to train the model outlined in Sect. 2.2. We begin by discussing the training data used, followed by our general approach to training and validation. Finally we discuss an alternative model choice that allows higher resolution on the particle momenta.

3.1 Training data

To enable proof-of-concept demonstrations of JUNIPR's various applications, we train the implementation described in Sect. 2.2 using jets simulated in PYTHIA v8.226 [63,64] and clustered using FASTJET v3.2.2 [59]. We simulated 600k hemisphere jets in PYTHIA using the process $e^+e^- \to q\bar{q}$ at a center-of-mass energy of 1 TeV, with hemispheres defined in FASTJET using the exclusive k_t algorithm [49,50], and with an energy window of 450–550 GeV imposed on the jets. To create the deterministic trees that JUNIPR requires, we reclustered the jets using the C/A clustering algorithm [51,52], with $E_{\text{sub}} = 1$ GeV and $R_{\text{sub}} = 0.1$. The nonzero values of E_{sub} and R_{sub} make the input to JUNIPR formally infrared-and-collinear safe, but this is by no means necessary. Furthermore, our approach is formally independent of the reclustering algorithm chosen. We demonstrate this by showing results using an absurd reclustering algorithm inspired by a 2D printer in Sect. 5.2, as well as for anti-k_t [58] reclustering in Sect. 5.3.

Thus we have 600k quark jets with $E_{\text{jet}} \sim 500$ GeV and $R_{\text{jet}} \sim \pi/2$. We use 500k of these jets for training, with 10k set aside as a test set to monitor overfitting, and we use the remaining validation set of 100k jets to make the plots in this paper.

In the applications of Sect. 4, we also make use of several other data sets produced according to the above specifications, with small but important changes. We list these modifications here for completeness. In one case, quark jets from $e^+e^- \to q\bar{q}$ were required to lie in a very tight mass window of 90.7–91.7 GeV. A sample of boosted Z jets from $e^+e^- \to ZZ$ events was also produced with the same mass cut. And finally, another sample of quark jets was produced, as detailed above, but with the value of $\alpha_s(m_Z)$ in the final state shower changed from PYTHIA's default value of 0.1365 to 0.11.

Before being fed to JUNIPR, jets in these data sets must be clustered, so that each jet becomes a tree of $1 \to 2$ branchings ending in the n final-state momenta of the jet:

$$\begin{array}{ccc} & & \begin{array}{cccc} (t=1) & (t=2) & \dots & (t=n-1) \quad (t=n) \end{array} \\ \text{jet} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_n \end{pmatrix} & \xrightarrow{\substack{\text{clustering} \\ \text{algorithm}}} & \begin{pmatrix} k_1^{(1)} & k_1^{(2)} & \dots & k_1^{(n-1)} & p_1 \\ & k_2^{(2)} & & k_2^{(n-1)} & p_2 \\ & & & \vdots & p_3 \\ & & & k_{n-1}^{(n-1)} & \vdots \\ & & & & p_n \end{pmatrix} \end{array} \tag{3.1}$$

where the momenta in one column are equal to those of the next column except for a single $1 \to 2$ branching. At each step t, only the momenta associated with this $1 \to 2$ branching are fed into JUNIPR, as detailed in Sect. 2. With this setup, JUNIPR requires minimal parameters; it learns to update $h^{(t)}$ as the tree evolves by focusing only on the step-by-step changes to the jet. Note also that jets of arbitrary length can be considered.

Note that in implementing JUNIPR, we do not directly evaluate the branching function $P_{\text{branch}}\big(k_{d_1}^{(t+1)}, k_{d_2}^{(t+1)} \big| k_m^{(t)}, h^{(t)}\big)$ on the momenta $k_{d_1}^{(t+1)}, k_{d_2}^{(t+1)}$ but instead use the parameterization $x = (z, \theta, \phi, \delta)$ shown in Fig. 6. In fact, we use a nonlinear transformation of this parameterization:

$$\begin{aligned} \tilde{z} &= \frac{\log z - \log \frac{E_{\text{sub}}}{E_{\text{jet}}}}{\log \frac{1}{2} - \log \frac{E_{\text{sub}}}{E_{\text{jet}}}} \\ \tilde{\theta} &= \frac{\log \theta - \log \frac{R_{\text{sub}}}{2}}{\log R_{\text{jet}} - \log \frac{R_{\text{sub}}}{2}} \\ \tilde{\phi} &= \frac{\phi}{2\pi} \\ \tilde{\delta} &= \frac{\log \delta - \log \frac{E_{\text{sub}} R_{\text{sub}}}{E_{\text{jet}}}}{\log \frac{R_{\text{jet}}}{2} - \log \frac{E_{\text{sub}} R_{\text{sub}}}{E_{\text{jet}}}} \end{aligned} \tag{3.2}$$

This invertible transformation simply maps the range of each coordinate onto [0, 1], which reduces the amount of global parametric shift required in optimization. Similarly, we perform a transformation on the components of $k_{d_1}^{(t)}, k_{d_2}^{(t)}$ before feeding them into the update rule for $h^{(t)}$ in Eq. (2.4); we do

the same for $k_m^{(t)}$, the input to the branching function P_{branch}. This is a technical point that is not conceptually important.

3.2 Approach to training

To train JUNIPR, we maximize the log likelihood over the full set of training data:

$$\text{log likelihood} = \sum_{\text{jet } i \text{ in data}} \log P_{\text{jet}}(\{p_1^{(i)}, \ldots, p_n^{(i)}\}). \quad (3.3)$$

For a particular jet with final-state momenta $p_1, \ldots, p_n$ we use Eqs. (2.2) and (2.3) to compute

$$\begin{aligned}\log P_{\text{jet}}(\{p_1, \ldots, p_n\}) = \sum_{t=1}^{n-1} \Big[& \log P_{\text{end}}\big(0 \big| h^{(t)}\big) \\ & + \log P_{\text{mother}}\big(m^{(t)} \big| h^{(t)}\big) \\ & + \log P_{\text{branch}}\big(k_{d_1}^{(t+1)}, k_{d_2}^{(t+1)} \big| k_m^{(t)}, h^{(t)}\big) \Big] + \log P_{\text{end}}\big(1 \big| h^{(n)}\big)\end{aligned} \quad (3.4)$$

where $m^{(t)}$ is the index of the mother momentum at step t in the training example and $k_{d_1}^{(t+1)}, k_{d_2}^{(t+1)}$ are its daughters. Maximizing the log likelihood in this way allows the model to learn each t step in parallel, providing computational efficiency and stability.

For all models presented in this paper, we use basic stochastic gradient descent with the following learning rate and batch size schedule, where training proceeds from left to right: We follow such a schedule to slowly increase the resolution and decrease the stochasticity of gradient descent throughout training. Decreasing the learning rate reduces the step size, thereby allowing finer details of the cost surface to be resolved. Increasing the batch size reduces the stochasticity by improving the sample estimates of the true gradients.

Schedule	5 epochs	5 epochs	5 epochs	5 epochs	5 epochs	5 epochs
Learning rate	10^{-2}	10^{-3}	10^{-4}	10^{-3}	10^{-4}	10^{-5}
Batch size	10	10	10	100	100	100

We wrote JUNIPR in Theano[2] [65] and trained it on 16-core CPU servers using the SherlockML technical data science platform. Training JUNIPR on 500k jets according to the above schedule took an average of 4 days.

[2] An open-source version of the code written in Keras will be released in the future.

3.3 Validation of model components

JUNIPR is constructed as a probabilistic model for jet physics by expanding P_{jet} as a product over steps t in the jet's clustering tree, as shown in Eq. (2.2). Each step involves three components: the probability P_{end} that the tree will end, the probability P_{mother} that a given momentum will be the next mother to branch, and the probability P_{branch} over the daughter momenta of the branching, as shown in Eq. (2.3). We now validate each of JUNIPR's components using our validation set of 100k previously unseen PYTHIA jets. In this section, we present histograms of actual outcomes in the PYTHIA validation set (i.e. frequency distributions) as well as JUNIPR's probabilistic output when evaluated on the jets in this data set (i.e. marginalized probability distributions) to check for agreement.

In Fig. 7 we show the probability P_{end} that the tree should end, as a function of both intermediate state length and maximum particle-to-jet-axis angle. In both cases we see excellent agreement with the validation data, demonstrating a good model fit with low underfitting and no overfitting. Note that Fig. 7 (left) is in one-to-one correspondence with the jet constituent multiplicity, and that the shape of Fig. 7 (right) is a direct consequence of C/A clustering with $R_{\text{sub}} = 0.1$. Indeed, if an opening angle near R_{sub} already exists in an angular-ordered tree, then there are likely no remaining branchings in the clustering tree.

In Fig. 8 we show the probability P_{mother} that a given candidate will be the next mother to branch in the clustering tree, as a function of both the candidate's index (which is sorted to be decreasing in energy) and the candidate's angle from the jet axis. The first of these results is shown in particular for the $t = 10$th step in the clustering trees. We observe again that the model fits the validation data well. Note from Fig. 8 (left) that the highest energy branches of the clustering tree are most likely to undergo subsequent branchings, in line with the expectation at leading logarithmic accuracy. Fig. 8 (right) shows consistent predictions, since the highest energy branches also lie at the narrowest angles to the jet axis.

In Fig. 9 we show the branching function P_{branch}, the component of the model that predicts how a mother momentum should split into a pair of daughter momenta. We show the branching function results for z and θ (i.e. with P_{branch} marginalized over the variables not shown) at the first step in the jet evolution $t = 1$, as well as at a later step $t = 10$. (See Fig. 6 for definitions of z and θ and Eq. (3.2) for their ranges in the data.) This shows the dependency of the branching function on the evolving jet representation $h^{(t)}$, which we will discuss in detail in Sect. 5.1. We see that for these direct predictions, JUNIPR fits the validation data almost perfectly. Note that in Fig. 9 (top) soft wide-angle emissions are the norm at the earliest t steps, as expected with the C/A clustering algorithm. In Fig. 9 (bottom) one can see that later

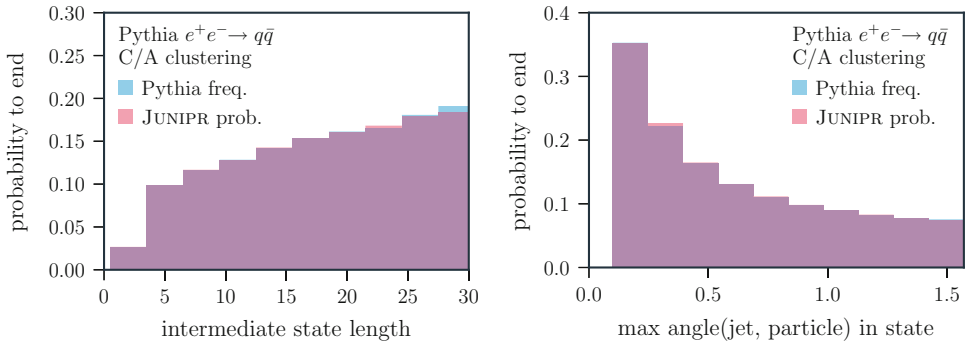

Fig. 7 Validation of P_{end}, the probability that the tree should end. Comparison is made between actual outcomes in the validation set of PYTHIA jets and JUNIPR's probabilistic predictions for these jets. (Left) P_{end} as a function of intermediate state length. (Right) P_{end} as a function of the maximum angle between the jet axis and momenta in the intermediate state

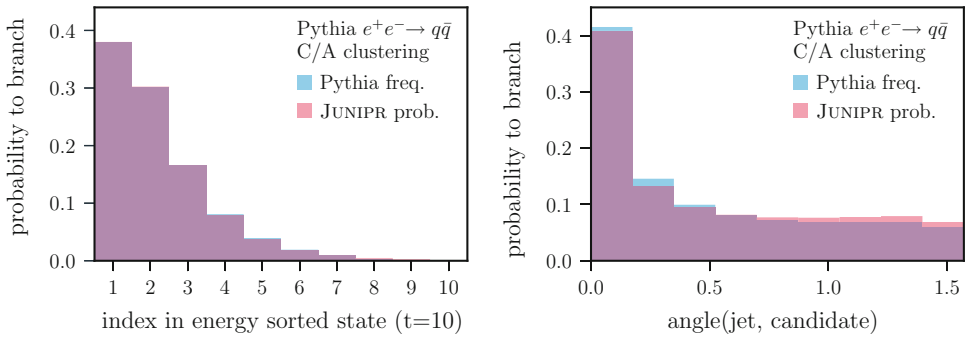

Fig. 8 Validation of P_{mother}, the probability that a given candidate will branch next in the clustering tree. Comparison is made between actual outcomes in the validation set of PYTHIA jets and JUNIPR's probabilistic predictions for these jets. (Left) P_{mother} at $t = 10$, as a function of a candidate's index in the energy ordered intermediate state. (Right) P_{mother} averaged over all t's, as a function of a candidate's angle relative to the jet axis

in the clustering trees, harder more-collinear branchings are commonplace. It bears repeating that these trends are highly dependent on the chosen clustering algorithm and have no precise connection to the underlying physical processes generating the data.

3.4 Increasing the branching function resolution

In this section, we discuss increasing the resolution of the branching function

$$P(x) \equiv P_{\text{branch}}\left(k_{d_1}^{(t+1)}, k_{d_2}^{(t+1)} \middle| k_m^{(t)}, h^{(t)}\right) \tag{3.5}$$

including the case where $P(x)$ is an energy model over continuous $x = (z, \theta, \phi, \delta)$. (The x coordinates were defined in Fig. 6.) This technical section can easily be skipped without loss of the logical flow of the paper.

We begin by briefly discussing increasing the resolution of the branching function over discrete x, the case described in Sect. 2.2. The first thing to note is that with a softmax over four-dimensional x, the size of the matrix multiplication required in a dense network is quartic in the number of bins used for each dimension. We generically use ten bins for each of z, θ, ϕ, δ resulting in an output size of 10^4. (In fact we use ten linearly spaced bins in the transformed coordinates of Eq. (3.2), and this can be seen on the logarithmic axes of Fig. 9, but this detail is not conceptually important.) Given this quartic scaling, simply increasing the number of discrete x cells quickly becomes prohibitively computationally expensive. Potential solutions to this problem include: (i) using a hierarchical softmax [66,67], and (ii) simply interpolating between the discrete bins of the model.

In a hierarchical softmax, a low-resolution probability is predicted first, say with 5^4 cells, then another 5^4-celled distribution is predicted inside the chosen low-resolution cell. In principle, this gives 25^4 resolution at only twice the computational time required for 5^4 resolution. We briefly implemented the hierarchical softmax, and preliminary tests found it to work efficiently, but perhaps with a decrease in training stability. We chose not to pursue the hierarchical softmax further in this work, primarily because we have not seen the need for resolution much higher than 10^4 discrete x cells.

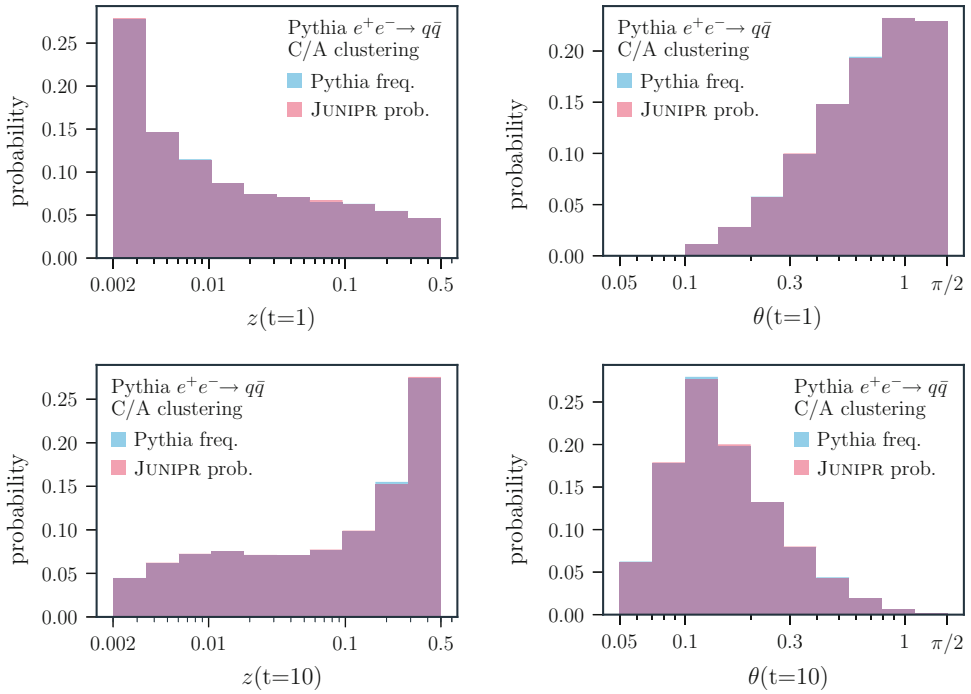

Fig. 9 Validation of P_{branch}, the four-dimensional probability distribution over $1 \to 2$ branchings. Comparison is made between actual outcomes in the validation set of PYTHIA jets and JUNIPR's probabilistic predictions for these jets. Results are shown for energy fraction z (left) and branching angle θ (right) as defined in Fig. 6. Evolution step $t = 1$ is shown (top) where soft wide-angle emissions are the norm, as expected in the C/A tree. Evolution step $t = 10$ (bottom) gives rise to harder more-collinear branchings

Due to its ease of use, we do employ linear interpolation between the discrete bins in our baseline model with resolution 10^4. This comes at no extra training cost, and removes most of the effects of discretization on the observable distributions generated by sampling from JUNIPR; see Sect. 4.2.

We now turn to the continuous version of JUNIPR in which the branching function $P(x)$ is given by an undirected energy model:

$$P(x) = \frac{e^{E(x)}}{Z}, \quad \text{where} \quad Z = \int dx\, e^{E(x)}. \tag{3.6}$$

To model $E(x)$, we again use a fully-connected network with hidden layer of size 100, as used everywhere else, except here the output layer is left to be linear. We perform the integral over Z using importance sampling:

$$\begin{aligned} Z &= \int dx\, q(x)\, \frac{e^{E(x)}}{q(x)} = \left\langle \frac{e^{E(x)}}{q(x)} \right\rangle_q \\ &\approx \frac{1}{|S|} \sum_{x_s \in S} \frac{e^{E(x_s)}}{q(x_s)} = \widehat{Z}(S) \end{aligned} \tag{3.7}$$

where S is the set of x_s's sampled from the importance distribution q.

Unlike the discrete-x version of JUNIPR, where training is relatively straightforward, the continuous-x version requires a non-standard technique in training the branching function $P(x)$. This is because, although Eq. (3.7) provides an unbiased approximation to Z,

$$\langle \widehat{Z} \rangle_{S \sim q} = Z, \tag{3.8}$$

this leads to a biased estimate of the log likelihood, since

$$\langle \log \widehat{Z} \rangle_{S \sim q} < \log \langle \widehat{Z} \rangle_{S \sim q} = \log Z \tag{3.9}$$

by Jensen's inequality. Thus, every gradient step taken is systematically different from the true gradient, and this bias derails training, especially near convergence when the true gradient becomes small.

To overcome this problem, we start by computing the sample variance on our estimate $\widehat{Z}(S)$, which is

$$\sigma(\widehat{Z})^2 = \frac{1}{|S| - 1} \sum_{x_s \in S} \left(\frac{e^{E(x_s)}}{q(x_s)} - \widehat{Z}(S) \right)^2. \tag{3.10}$$

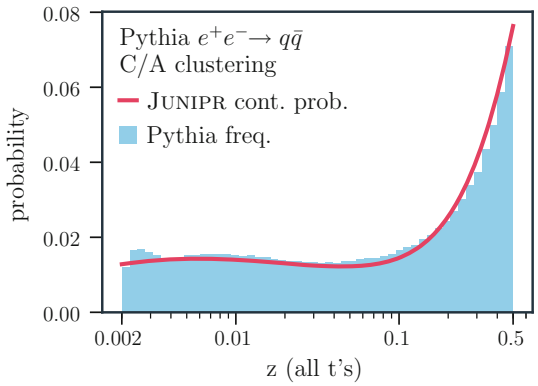

Fig. 10 Branching function modelled by a deep undirected energy model over continuous variables z, θ, ϕ, δ that parameterize the branching. Shown is the marginalized distribution over z, averaged over all t steps. Comparison is made between actual outcomes in the validation set of PYTHIA jets and JUNIPR's probabilistic predictions for these jets

Then the percent-error Δ in our biased estimate of the gradient is approximately

$$\Delta = \frac{1}{\sqrt{|S|}} \frac{\sigma(\widehat{Z})}{\widehat{Z}}. \tag{3.11}$$

This error propagates into the log likelihood, causing the bias in Eq. (3.9). To mitigate this, we adopt a policy of monitoring Δ during training, and whenever Δ increases above some value $\Delta_{\text{threshold}}$ (a hyperparameter that we set to 2%) we double the sample size $|S|$ used to compute $\widehat{Z}(S)$. This slows down training considerably, but it effectively reduces the bias in our gradient estimates. Note that while generic importance sampling typically fails in higher dimensions, our branching function lives in only four dimensions, so this approach is robust using any reasonable importance distribution q. Indeed, we found that a uniform distribution over the transformed coordinates of Eq. (3.2) is a fine choice for q.

In Fig. 10 we show results for JUNIPR trained with the continuous branching function as described above. In this case, we can use arbitrarily high-resolution binning, as JUNIPR has learned a fully continuous probability density. Figure 10 can be roughly compared to Fig. 9, where we were required to use 10 bins for each dimension of x.

To close this section, we note that in most cases, we expect the discretized branching function with ten bins per dimension of x to be sufficient, especially if one performs a linear interpolation on the output cells. This simple case is certainly faster to train and does not require the technique described here to avoid biased gradient estimates.

4 Applications and results

With JUNIPR trained and validated, we turn to some of the most interesting results it enables. Given a jet, JUNIPR can compute the probability density associated with the momenta inside the jet, conditioned on the criteria used to select the training data. To visualize this, we show a C/A-clustered PYTHIA jet in Fig. 11 with the JUNIPR-computed probability associated with each branching written near that node in the tree. Note that these are small discretized probabilities due to the discretized implementation of JUNIPR's branching function described in Sect. 2. This is shown primarily to conceptualize the model, which is constructed to be quite interpretable as it is broken down to compute the probability of each step in the clustering history of a jet.

A direct and powerful application of the JUNIPR framework, enabled by having access to separate probabilistic models of different data sources, is in discrimination based on likelihood ratios. We discuss discrimination in Sect. 4.1, along with a highly intuitive way of visualizing it. In contrast, an instinctive but indirect use of JUNIPR as a probabilistic model is in sampling new jets from it. We discuss the observable distributions generated through sampling in Sect. 4.2. However, sampling from a probabilistic model is often inefficient (e.g. slower than PYTHIA) compared to evaluating probabilities of jets directly. In Sect. 4.3 we discuss reweighting samples from one simulator to match those of another distribution. In principle, this could be used to tweak PYTHIA samples to match observed collider data simply by reweighting.

4.1 Likelihood ratio discrimination

We expect that one of the most exciting applications of JUNIPR will be in discriminating the underlying physics that could have created a jet.[3] For example, suppose we had two sets of jets, one set corresponding to decays of a boosted Z boson, the other set simply high-energy quarks. We could then train one copy of JUNIPR on just the boosted Z sample, giving the probability distribution P_Z, and another copy of JUNIPR on just the quark jets, giving P_q. Finally, for any new jet we could determine whether the jet was initiated by a boosted Z or by a high-energy quark by looking at the likelihood ratio:

$$\frac{P_Z(\text{jet})}{P_q(\text{jet})} > \text{threshold} \quad \Longrightarrow \quad \text{jet is boosted } Z \tag{4.1}$$

where the threshold is set according to the location on the ROC (receiver operating characteristic) curve desired for the discrimination task at hand. In contrast to approaches that try to compute likelihood ratios like this using QCD [56, 57], the JUNIPR approach can learn the separate probability distributions directly from samples of training data.

Discrimination based on the likelihood ratio theoretically provides the most statistically powerful discriminant

[3] We thank Kyle Cranmer for an early discussion on this topic.

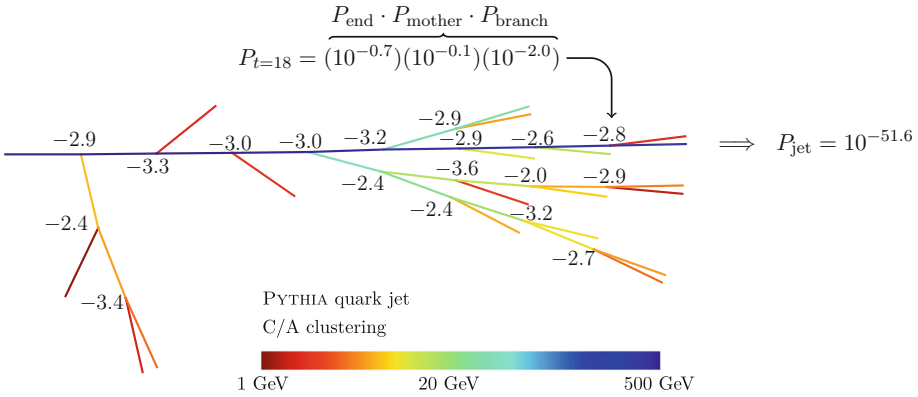

Fig. 11 JUNIPR-computed probability asigned to example PYTHIA jet and sequentially decomposed along its C/A clustering tree. Nodes are labeled with $\log_{10} P_t$, where $P_t = P_{\text{end}} \cdot P_{\text{mother}} \cdot P_{\text{branch}}$ includes the product of all three components of the probability at step t, as shown in Eq. (2.3). Color corresponds to energy and opening angle corresponds to three-dimensional branching angle. Probabilities are small and discrete due to the discretized branching function used in JUNIPR's implementation

between two hypotheses [42]. Moreover, our setup takes into account all the momenta that define a specific type of jet. Note also that for the task of pairwise discrimination between N jet types, this unsupervised approach requires training N probabilistic models, whereas a supervised learning approach would require training $N(N-1)/2$ classifiers. Thus, we expect likelihood-ratio discrimination using JUNIPR to provide a powerful tool.

We note further that we do not even require pure samples of the two underlying processes between which we would like to discriminate [35]. Thus, it would be feasible to discriminate based solely on real collider data. In our Z/quark example above, we would simply train one copy of JUNIPR on a sample of *predominantly* boosted-Z jets, and train another copy on *predominantly* quark jets, and the likelihood ratio of those two models would still be theoretically optimal for Z/quark discrimination.

In order to get a first look at the potential of likelihood-ratio discrimination using JUNIPR, we continue with the Z/quark example discussed above. We use PYTHIA to simulate $e^+e^- \to q\bar{q}$ and $e^+e^- \to ZZ$ events at a center-of-mass energy of 1 TeV. We impose a very tight mass window, 90.7–91.7 GeV, on the jets in each data set, so that no discrimination power can be gleaned from the jet mass. More details on the generation of the data sets were given in Sect. 3.1. We admit that a more compelling example of discrimination power would be for quark and gluon jets at hadron colliders, but we leave a proper treatment of that important case to future work. The toy scenario studied here serves both to prove that the probabilities output by JUNIPR are meaningful, and that likelihood ratio discrimination using unsupervised probabilistic models is a promising application of the JUNIPR framework.

In Fig. 12 we show the Z/quark separation power achieved by JUNIPR, both in terms of full likelihood ratio distributions for validation sets of Z and quark jets, as well as the resulting ROC curve. For comparison, in Fig. 12 we also show the ROC curve achieved using a 2D likelihood ratio discriminant based on 2-subjettiness [68] and multiplicity. JUNIPR's likelihood-ratio discrimination is clearly superior to that based on combining the most natural observables: 2-subjettiness, multiplicity, (and keep in mind the tight mass cut). Of course, these observables do not provide state-of-the-art discrimination power even in this toy scenario, but we include the comparison in this proof-of-concept to provide a sense of scale on the plot.

By design, JUNIPR naturally processes the information in jets via a recurrent mechanism that tracks the evolution of their clustering trees, and this allows users to peer inside at this structure and access the probabilities at each branching. In particular, we can consider the likelihood ratio at each step in the clustering trees to understand which branchings give rise to the greatest discrimination power. We show this in Fig. 13, where it is clear that JUNIPR can extract useful discriminatory information at most branchings.

Indeed, visualizing jets as in Fig. 13 can provide a number of insights. Unsurprisingly, we see for the quark jet (on the top) that the likelihood ratio of the first branching is rather extreme, at $10^{-3.7}$, since it is unlike the energy-balanced first branching associated with boosted-Z jets. However, we also see that almost all subsequent branchings are also unlike those expected in boosted-Z jets, and they combine to provide comparable discrimination power to the first branching alone. Many effects probably contribute to this separation power at later branchings, including that quark jets often gain their mass throughout their evolution instead of solely at the first branching, and that the quark jet is color-connected to

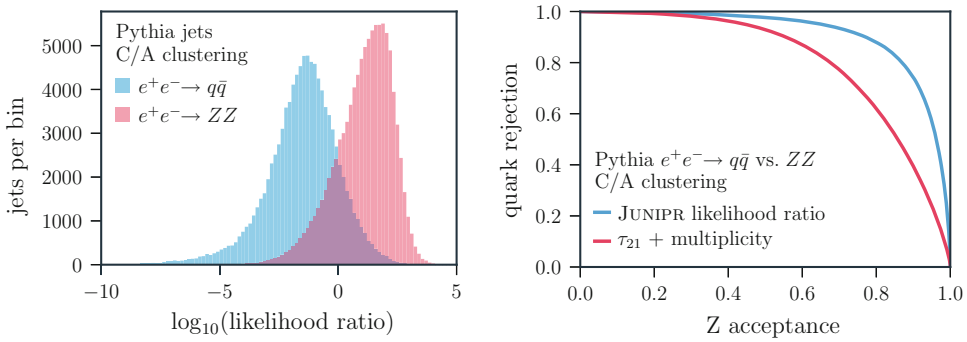

Fig. 12 (Left) Likelihood ratio $P_Z(\text{jet})/P_q(\text{jet})$ evaluated on PYTHIA jets in the validation set. (Right) ROC curve for discrimination based on JUNIPR's likelihood ratio, in comparison to the empirical 2D distribution using 2-subjettiness and constituent multiplicity. All jets used in this study have masses between 90.7 and 91.7 GeV

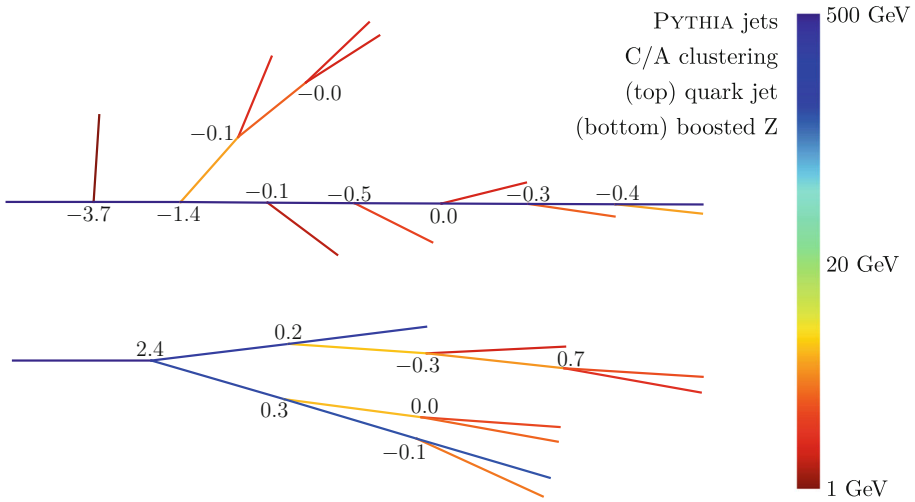

Fig. 13 JUNIPR trees for visualization of discrimination power at individual nodes in the clustering history. Each node is labeled with the component of $\log_{10} P_Z(\text{jet})/P_q(\text{jet})$ associated with that t step. Colors represent energies, and opening angles represent physical three-dimensional branching angles. The top figure is a quark jet generated using PYTHIA, with mass between between 90.7 and 91.7 GeV; the bottom figure is a boosted-Z jet. The role that the energy distribution, opening angles, multiplicity, and branching pattern play in high-performance discrimination can be understood from such pictures

other objects in the global event. Such effects have proven to be useful for discrimination in other contexts [69].

Similarly, considering the boosted-Z jet on the bottom of Fig. 13 shows that significant discrimination power comes not only from the first branching, but also from subsequent splittings, as the boosted-Z jet evolves as a color-singlet $q\bar{q}$ pair. Note the presence of the predictive secondary emissions sent from one quark-subject toward the other. This is reminiscent of the pull observable, which has proven useful for discrimination in other contexts [70]. More generally, the importance of the energy distribution, opening angles, multiplicity, and branching pattern in high-performance discrimination can be understood from such pictures.

We are very excited by the prospect of visualizing JUNIPR's discrimination power on jets, based on the likelihood ratio it assigns at each branching in their clustering trees, as in Fig. 13. Such visualizations could provide intuition that leads to the development of new, human-interpretable, perhaps calculable observables for discrimination in important contexts.

We would like to make one side note about discrimination, before moving on to the next application of JUNIPR. The statement that likelihood-ratio discrimination is optimal of course only applies in the limit of perfect models. Since this limit is never fully realized, one may worry that discrimination with JUNIPR may in fact be suboptimal. Since the two probabilistic models we use for discrimination are each trained individually to replicate a certain type of jet, they are not conditioned to focus on the differences between the two jet types, which may be very subtle in the case of a difficult discrimination task. In the realistic case of slightly imperfect models, it may be advantageous for discrimination purposes to instead train the two models to focus on the differences. To be specific, one could train the two models on the two data sets simultaneously, with the goal being to maximize the likelihood ratio

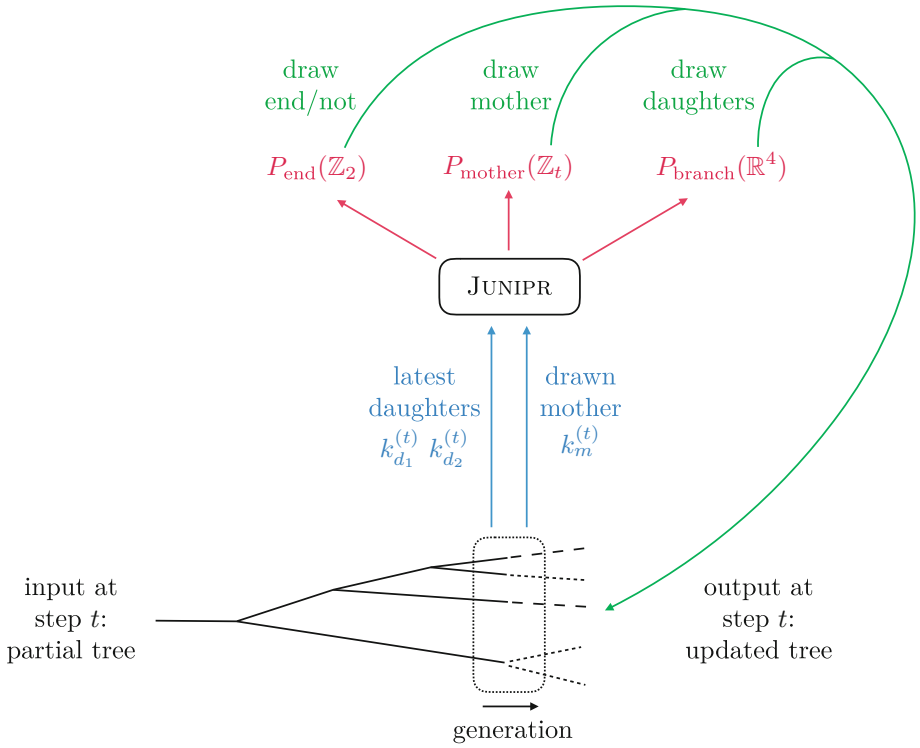

Fig. 14 Sampling from JUNIPR to generate jets. Draws from low-dimensional distributions at each step t are fed forward to subsequent steps to ultimately generate a full jet

on one data set and minimize it on the other. Following this method in the particular example of Z/quark discrimination used above, one would train the P_Z and P_q models on data sets D_Z and D_q to maximize the following quantity:

$$\sum_{\text{jet}\in D_Z} \log \frac{P_Z(\text{jet})}{P_q(\text{jet})} - \sum_{\text{jet}\in D_q} \log \frac{P_Z(\text{jet})}{P_q(\text{jet})}. \tag{4.2}$$

Compare this to the approach we have taken above, namely training P_Z and P_q to separately maximize the log likelihood of Eq. (3.3) on their corresponding sets of training data. This alternative training method would correspond to optimizing JUNIPR for the application of discrimination, leaving intact our ability to visualize discrimination power in clustering trees, but sacrificing the probabilistic interpretation of the model's output. We have not tested training with Eq. (4.2), and thus cannot attest to its practicality, but we suspect an approach along these lines may be useful in certain contexts.

4.2 Generation from JUNIPR

We now turn to a more familiar approach to jet physics, but a somewhat less appropriate usage of JUNIPR models: sampling new jets from the learned probability distribution to generate traditional observable distributions. We include this application here, not only to demonstrate this capability, but also to further validate the distribution learned by JUNIPR during unsupervised training.

Sampling from JUNIPR is relatively efficient; one simply samples from the low dimensional distributions at each step t and feeds those samples forward as input to subsequent steps. In this way, one generates a full jet in many steps, as detailed in Fig. 14.

We used the baseline implementation of JUNIPR trained on quark jets, as described in Sect. 3, to generate 100k jets in this way. The resulting jet mass and constituent multiplicity distributions are plotted in Fig. 15 where both distributions sampled from JUNIPR match those created from our validation set of 100k PYTHIA jets withheld from training. Reasonable agreement can also be seen in the 2D distributions of Fig. 16.

However, there are two reasons why we do not consider JUNIPR to be built for generation. (These drawbacks could be avoided with a generative model; see [39–41].) The first is simply that sampling from probability distributions is generally difficult. As we just showed, it turns out that JUNIPR is relatively easy to sample from, due to its sequential structure and the fact that distributions are low-dimensional at each t step. Despite this, sampling jets from JUNIPR is still much slower than generation with, for example, PYTHIA.

The second reason is more fundamental. With a sequential model structured as JUNIPR is, probability distributions at late t steps in generation are highly sensitive to the draws made at earlier t steps. Very small defects in the probability distributions at early steps cause feedback in the model that amplifies those errors. Furthermore, as a partially generated jet becomes more misrepresentative of the training data, the resulting probability distributions used at later steps are less

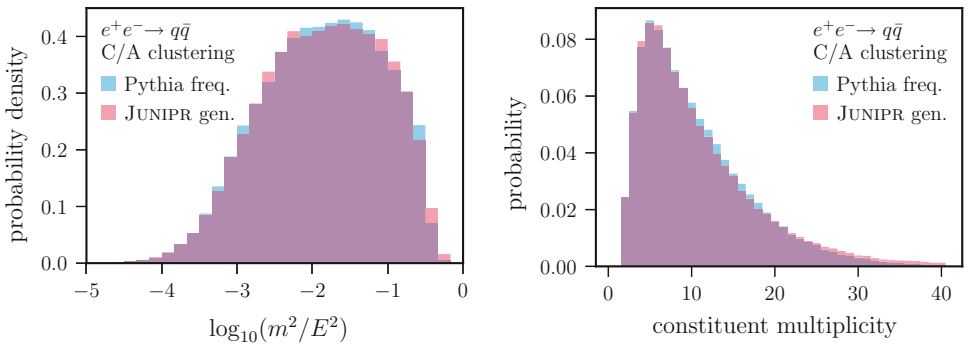

Fig. 15 Jet mass (left) and constituent multiplicity (right) distributions computed on jets sampled from JUNIPR and compared against PYTHIA jets in the validation set

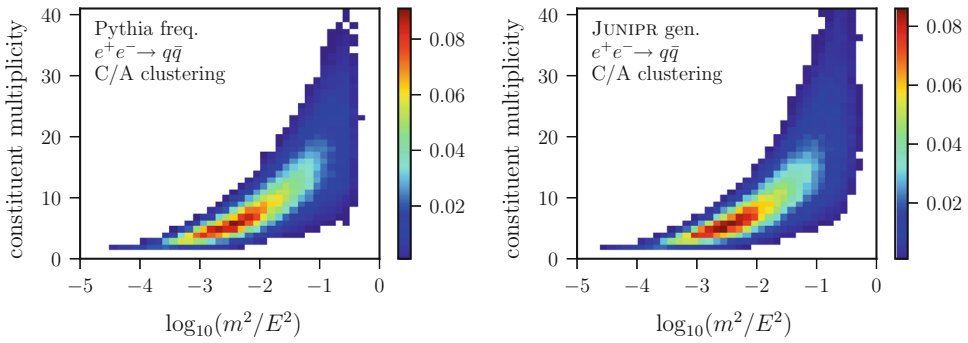

Fig. 16 Two-dimensional probability distributions with respect to jet mass and constituent multiplicity. (Left) Distribution computed using validation set of PYTHIA jets. (Right) Distribution computed using jets sampled from JUNIPR

trained, which can result in a run-away effect. All of this is to say that, for the purpose of generating jets, JUNIPR's accuracy at early t steps is disproportionately important. This is in tension with the training method undertaken in Sect. 3.2, namely the maximization of the log-likelihood, which prioritizes all branchings equally. Thus, we should expect that some observable distributions generated by sampling jets from JUNIPR might agree worse with the validation set of PYTHIA data than otherwise expected. We mention in passing that this second drawback could be mitigated by reweighting jets after generation, as detailed in Sect. 4.3 below.

In fact, we have found empirically that the N-subjettiness ratio observables computed by sampling from JUNIPR do not match the held-out PYTHIA data perfectly. This can be seen in Fig. 17 with the 2-subjettiness distribution, where the difference between the two distributions is more significant.

We consider this disagreement to be both expected and non-diminishing of JUNIPR's potential. Indeed, in the next section we will show how to overcome this issue, by generating samples consistent with JUNIPR's learned probabilistic model, without ever sampling from it. In particular, the disagreement in Fig. 17 will be rectified in Fig. 18.

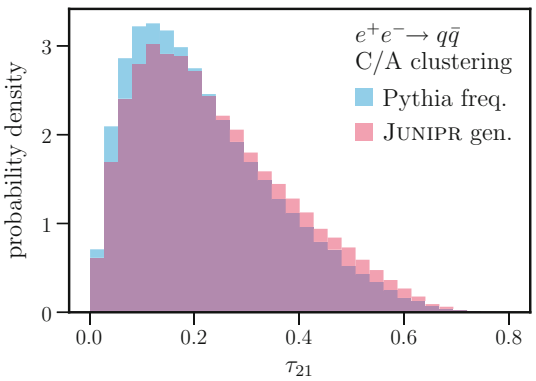

Fig. 17 2-Subjettiness ratio observable computed on jets sampled from JUNIPR. Disagreement with the distribution on PYTHIA jets, due to the feedback involved in sampling from JUNIPR, is visible. This disagreement is amended in Fig. 18

4.3 Reweighting Monte Carlo events

Another application of the JUNIPR framework is to reweight events. For example, suppose we trained JUNIPR on data from the Large Hadron Collider (LHC) to yield a probabilistic model P_{LHC}. Then one could generate a sample of new events using a relatively accurate Monte Carlo simulator, train another instance of JUNIPR on that sample to yield P_{sim}, and finally reweight the simulated events by $P_{\text{LHC}}/P_{\text{sim}}$ evaluated on an event-by-event basis. This process yields a sample of events that is theoretically equivalent to the LHC data used in training P_{LHC}. The advantage of such an approach is that JUNIPR can correct the simulated events on different levels, for example using the data reclustered in $R_{\text{sub}} = 0.1$ subjets as we have done in this paper. However, the full simulated event has the complete hadron distributions and can thereby be interfaced with a detector simulation. This is in many ways a simpler approach than trying to improve the

simulation directly through the dark art of Monte-Carlo tuning.

This reweighting is identical to importance sampling from a proposal distribution given by the simulated data distribution P_{sim}. For example, suppose one wanted to measure the distribution of an observable $\mathcal{O}(\text{jet})$ at the LHC, which is given by

$$\begin{aligned} P(\mathcal{O}) &= \int d[\text{jet}]\, P_{\text{LHC}}(\text{jet})\, \delta(\mathcal{O} - \mathcal{O}(\text{jet})) \\ &\approx \frac{1}{N} \sum_{\text{jet}\sim P_{\text{LHC}}} \delta(\mathcal{O} - \mathcal{O}(\text{jet})) \end{aligned} \tag{4.3}$$

where the last approximation is associated with collecting a finite amount N of LHC data in order to measure the distribution. (The reader can substitute discretized delta functions appropriate for histogramming if averse to the singular notation used in these equations.) Instead of using real data, if say a public version of P_{LHC} were available, then anyone could calculate this observable distribution using only simulated data sampled from P_{sim} as follows:

$$\begin{aligned} P(\mathcal{O}) &= \int d[\text{jet}]\, P_{\text{sim}}(\text{jet})\, \delta(\mathcal{O} - \mathcal{O}(\text{jet}))\, \frac{P_{\text{LHC}}(\text{jet})}{P_{\text{sim}}(\text{jet})} \\ &\approx \frac{1}{N} \sum_{\text{jet}\sim P_{\text{sim}}} \delta(\mathcal{O} - \mathcal{O}(\text{jet}))\, \frac{P_{\text{LHC}}(\text{jet})}{P_{\text{sim}}(\text{jet})}. \end{aligned} \tag{4.4}$$

In this way, one could efficiently obtain samples of arbitrary size from P_{LHC} by reweighting samples generated by an efficient simulator. The only limitation to this process is that the simulated data must be similar to the actual target data, so that they have overlapping regions of support (formal requirement) and the weights are not too far from unity (efficiency requirement).

As with the likelihood-ratio discrimination in Sect. 4.1, here we will show results in a toy scenario as a proof-of-principle. Ideally a model trained on LHC data, with all related complications, would be used to reweight Monte Carlo jets to make the simulated data indiscernible from LHC data; we leave a proper study of this to future work.

Instead, here we use two samples of jets generated using two different versions of Pythia. We reweight jets from one of the samples and demonstrate their agreement with the other sample. In particular, we use our baseline Junipr model trained on Pythia-generated quark jets as our "true distribution". For the moment, we will refer to this model as $P_{\alpha_s=0.1365}$, since its training data was generated using Pythia's default value of $\alpha_s(m_Z) = 0.1365$ in the final state shower. As our "simulated distribution" we will use $P_{\alpha_s=0.11}$, which was trained on quark jets generated with coupling parameter changed to $\alpha_s(m_Z) = 0.11$ in Pythia's final-state shower. (See Sect. 3.1 for a more in-depth description of the training data used.) Our goal is to show that reweighting jets from the "simulated distribution" according to the likelihood ratio $P_{\alpha_s=0.1365}/P_{\alpha_s=0.11}$ leads to observables in agreement with the "true distribution".

In Fig. 18 we demonstrate that this is indeed the case. We check this for both the 2-subjettiness and 3-subjettiness ratio observables, as well as the jet shape observable. On the left side of Fig. 18, one can see that in all cases, the $\alpha_s = 0.11$ distribution is clearly different from the $\alpha_s = 0.1365$ distribution. On the right side of Fig. 18, one finds that the two distributions come into relatively good agreement once the $\alpha_s = 0.11$ jets are reweighted by $P_{\alpha_s=0.1365}/P_{\alpha_s=0.11}$. This also provides further confirmation that Junipr learns subtle correlations between constituent momenta inside jets.

Note that it was the 2-subjettiness ratio observable that Junipr struggled to predict well through direct sampling (see Fig. 17), whereas when reweighting another set of samples, Junipr matches the data well on this observable (see top-right of Fig. 18). This corroborates the discussion in Sect. 4.2 concerning the difficulties in sampling directly from Junipr.

Before closing this section, let us reiterate one point mentioned above. For the procedure of reweighting events to be practical, the weights used should not be radically different from unity, meaning that the two distributions generating the two samples should not be too different. If this condition is not satisfied, then away from the limit of infinite statistics, a few events with very large weights could vastly overpower the rest of the events, leading to a choppy reweighted distribution with large statistical uncertainties. To avoid this problem in the toy scenario explored in this section, we found it necessary to discard roughly 0.1% of the jets in the $\alpha_s = 0.11$ sample which were outliers with $P_{\alpha_s=0.1365}/P_{\alpha_s=0.11} > 100$. These outliers were uncorrelated with the observables shown, and we believe they resulted from imperfections in the trained model. It is clear that much more needs to be understood about the application of reweighting, but this would perhaps be more effectively done in the context of a specific task of interest involving LHC data.

5 Factorization and JUNIPR

In the previous section, we showed some preliminary but very exciting results for likelihood-ratio discrimination and for the generation of observables by reweighting simulated jets. Both of these applications require access to an unsupervised probabilistic model. Next we discuss some of the more subtle internal workings of Junipr, which are intimately related to the underlying physics of factorization.

In particular, we show that the hidden representation $h^{(t)}$ indeed stores important global information about intermediate states of jets in Sect. 5.1. We then discuss the clustering-algorithm independence of Junipr by considering two dis-

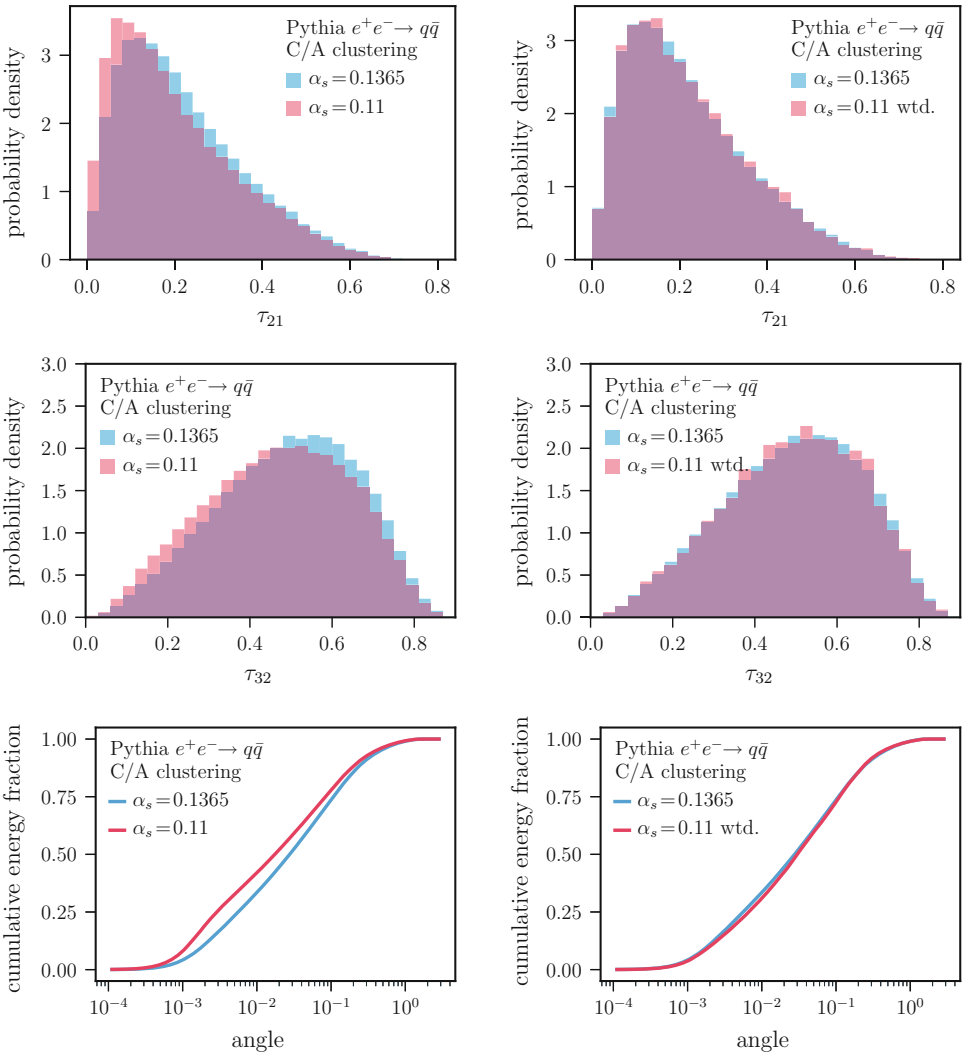

Fig. 18 (Left) Disagreement in observable distributions for two PYTHIA tunes of α_s. Observables are the 2-subjettiness and 3-subjettiness ratio observables and the jet shape, from top to bottom. (Right) Upon reweighting the $\alpha_s = 0.11$ jets by the ratio $P_{\alpha_s=0.1365}/P_{\alpha_s=0.11}$ of learned underlying probability distributions, observable distributions exhibit good agreement

tinct clustering algorithms: a "printer" algorithm in Sect. 5.2, where momenta are processed left-to-right and top-to-bottom as if by an inkjet printer; and the anti-k_t algorithm in Sect. 5.3, which allows us to present another counterintuitive result, the anti-k_t shower generator.

5.1 The encoding of global information

We have constructed JUNIPR so that all global information about the jet is contained in the RNN's hidden state $h^{(t)}$. Only the branching function P_{branch} receives the local $1 \to 2$ branching information in addition to $h^{(t)}$. This forces $h^{(t)}$ to contain all the information needed to predict when the shower should end, P_{end}, to predict which momentum should branch next, P_{mother}, and to inform the branching function P_{branch} of the relevant global structure. As the primary feature vector for all three of these distinct tasks, $h^{(t)}$ must learn an effective representation of the jet at evolution step t.

To explicitly show that $h^{(t)}$ stores important global information about the intermediate jet state at step t, we train a new model on our baseline quark jet data (see Sect. 3.1) with the difference that we remove $h^{(t)}$ as an input to the branching function P_{branch}. We expect that such a "local" branching model will not evolve correctly as the global jet structure evolves, since all global information is being withheld. This is indeed what we find, as can be seen in Fig. 19. On the left side of that figure, the evolution of the θ distribution (defined in Fig. 6) from $t = 1$ to $t = 2$ is shown using 100k PYTHIA jets from our held-out set of validation data. There we see the gradual decrease in angle as expected for C/A trees. On the right side of Fig. 19, the evolution of the branching function is shown for the "local" branching model, and the disagreement between this damaged model and PYTHIA is clear. Note that this prediction of incorrect distributions at intermediate branchings in the C/A tree will inevitably lead to an incorrect probability distribution $P_{\text{jet}}(\{p_1, \ldots, p_n\})$ over final-state momenta.

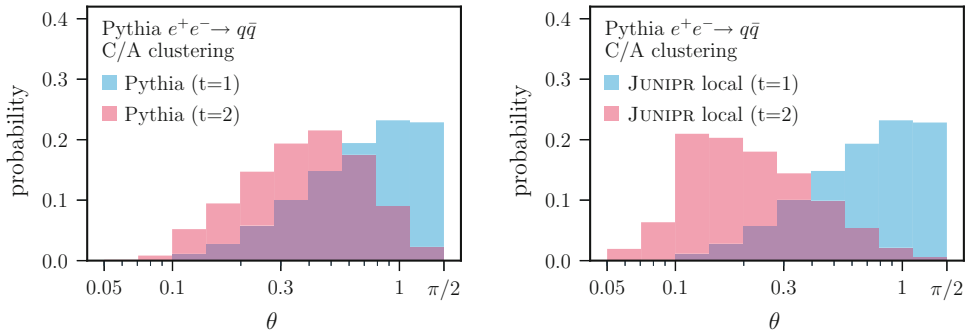

Fig. 19 (Left) Evolution of the θ distribution from $t = 1$ to $t = 2$ in the validation set of PYTHIA jets. (Right) Corresponding evolution of the branching function as predicted by a "local" branching model without access to the hidden representation $h^{(t)}$. Disagreement between PYTHIA and this local model is clear. Not shown is the result using our baseline (global) model, which agrees perfectly with PYTHIA, as expected from Fig. 9

While we do not show the corresponding results from our baseline (global) model in Fig. 19 to avoid clutter, the agreement with PYTHIA is essentially perfect, as one would expect from the similar check performed in Fig. 9. This confirms the success of the jet representation $h^{(t)}$ in supplying the branching function P_{branch} with important information about the global structure.

5.2 Clustering algorithm independence

Another subtle aspect of JUNIPR is its theoretical clustering algorithm independence. In principle, the model as described in Sect. 2.1 is indeed independent of the chosen algorithm, which is fixed simply to avoid a sum over all possible trees consistent with the final-state momenta. That is, for each clustering procedure chosen by the user, a different model is learned, but one that describes the same probability distribution over final-state momenta, at least formally.

However, it is not guaranteed that a given neural-network implementation of JUNIPR will work well for every clustering algorithm. We have chosen an architecture that stores the global jet physics in the RNN's hidden state $h^{(t)}$ and the local $1 \to 2$ branching physics in the branching function P_{branch}. This architecture is motivated by the factorizing structure of QCD, and thus JUNIPR will most easily learn jet trees that are most similar to QCD – our primary reason for predominantly using the C/A algorithm. Consequently, though the model described in Sect. 2.1 is formally independent of clustering algorithm, the particular implementation adopted in Sect. 2.2 may weakly depend on the chosen algorithm by virtue of the ease with which it can learn the data.

To put this to the test, we have introduced a jet clustering algorithm that is nothing like QCD, but more like a 2D printer.[4] The "printer" clustering algorithm scans the 2D jet image (i.e. the cross sectional image perpendicular to the jet axis) from right-to-left and bottom-to-top, clustering particles as it encounters them. Run in reverse (i.e. as a shower) particles are emitted from the jet core from left-to-right and top-to-bottom; this is how a jet image would be printed by an inkjet printer with a single printing tip. In Fig. 20 we show a single PYTHIA jet clustered using the printer algorithm. As can be seen in the jet image on the right side of Fig. 20, momenta are indeed emitted top-to-bottom. On the left side of Fig. 20, we see that any collinear branching structure is completely absent from the clustering tree; instead, particles are steadily emitted up-and-to-the-left.

Though JUNIPR's neural network architecture is not optimized for the informational structure of the printer algorithm, it is still able to learn the structure, by relying much more heavily on the the jet representation $h^{(t)}$. We demonstrate this by training JUNIPR on our data set of PYTHIA-generated quark jets (see Sect. 3.1) clustered with the printer algorithm, thus yielding the probabilistic model P_{printer}. Indeed, in Fig. 21 one can see a jet sampled from P_{printer}, which correctly follows the printer structure.

As expected, however, the distributions sampled from P_{printer} are not quite as good as our C/A results. On the left side of Fig. 22 we show the two-dimensional distribution over jet mass and constituent multiplicity generated using 100k jets sampled directly from P_{printer}. Comparing to the distribution generated by PYTHIA (see the left side of Fig. 16) this distribution matches well. However, for the 2-subjettiness ratio observable on the right side of Fig. 22 we get a somewhat worse match to the PYTHIA validation data; compare this to the results of the C/A model in Fig. 17. Of course, we discussed in Sect. 4.2 why we do not expect direct sampling from JUNIPR to be perfectly reliable (and we discussed a way around this in Sect. 4.3), but it is still clear that such distributions are comparably worse when using the printer clustering algorithm, instead of the more natural C/A algorithm.

[4] We thank Eric Metodiev for this suggestion.

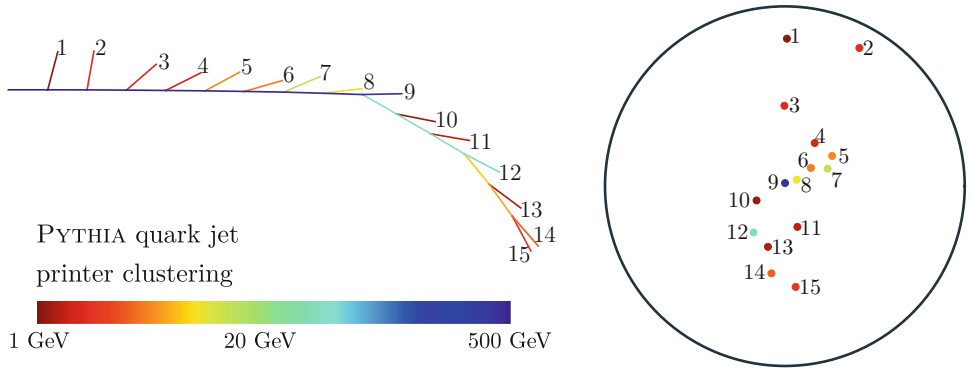

Fig. 20 A single PYTHIA jet clustered using the printer algorithm. Shown are its clustering tree (left) and jet image (right) in which colors correspond to energies and polar coordinates correspond to the θ and ϕ values of the momenta. Each momentum is labelled by its corresponding step t in the clustering tree

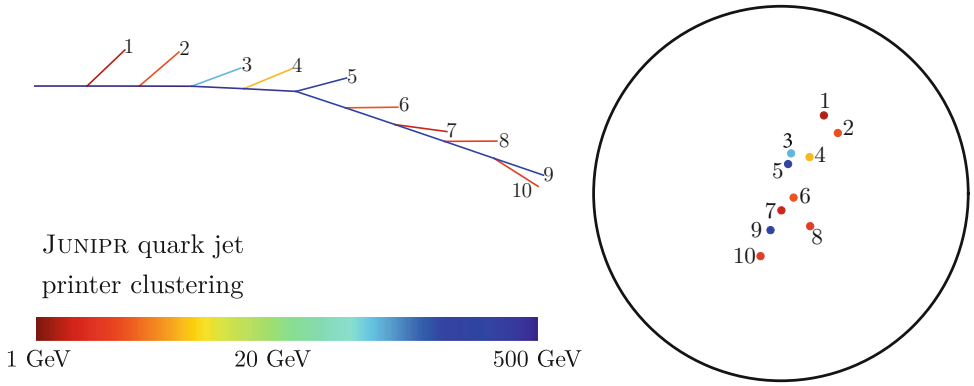

Fig. 21 A single jet sampled from JUNIPR, which was trained on PYTHIA-generated quark jets that were clustered using the printer algorithm. The sampled jet emits with the correct printer structure, as can be seen by its emission tree (left) and jet image (right). Each momentum is labelled by the step t at which it was emitted during generation from JUNIPR

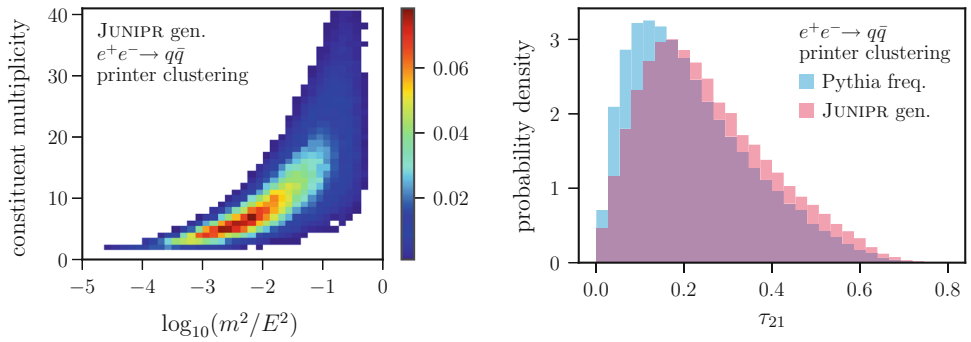

Fig. 22 (Left) 2-dimensional distribution with respect to jet mass and constituent multiplicity, calculated by sampling jets directly from P_{printer}, an instance of JUNIPR trained on jets clustered with the printer algorithm. (Right) 2-subjettiness ratio observable distribution generated using P_{printer} and compared to the corresponding distribution on PYTHIA jets in the validation set

5.3 Anti-k_t shower generator

Reassured by the results of the previous section, we next consider JUNIPR trained on PYTHIA jets reclustered with anti-k_t [58]. Like the printer algorithm, anti-k_t does not approximate the natural collinear structure of QCD. Unlike the printer algorithm, however, anti-k_t is a very commonly used tool. For the latter reason we explore anti-k_t jets here.

Perhaps the most interesting result associated with an anti-k_t version of JUNIPR is that it provides access to an anti-k_t shower generator. Generating an anti-k_t shower is counterintuitive, because the anti-k_t algorithm generally clusters soft emissions one-by-one with the hard jet core. Thus, a generator must remember where previous emissions landed in order to send subsequent emissions nearby. This is required to reproduce the correct collinear structure in the distribution of final-state of momenta. Said in another way, since the collinear factorization of QCD is not built into the anti-k_t clustering algorithm, a local (or factorized) anti-k_t generator could not produce emissions with the correct collinear distribution. Thus, we should expect that, in an anti-k_t version of JUNIPR, higher demands will be placed on the jet representation $h^{(t)}$ to monitor all the radiation in the jet. This is certainly possible, but not the task for which our neural network architecture is optimized.

To see to what extent an anti-k_t implementation of JUNIPR relies on the global information stored in $h^{(t)}$, we trained two models on PYTHIA-generated quark jets clustered with anti-k_t (see Sect. 3.1 for more details on the training data used). One model, P_{anti}, has the baseline architecture outlined in Sect. 2. The other, $P_{\text{anti-local}}$, is a local branching model like

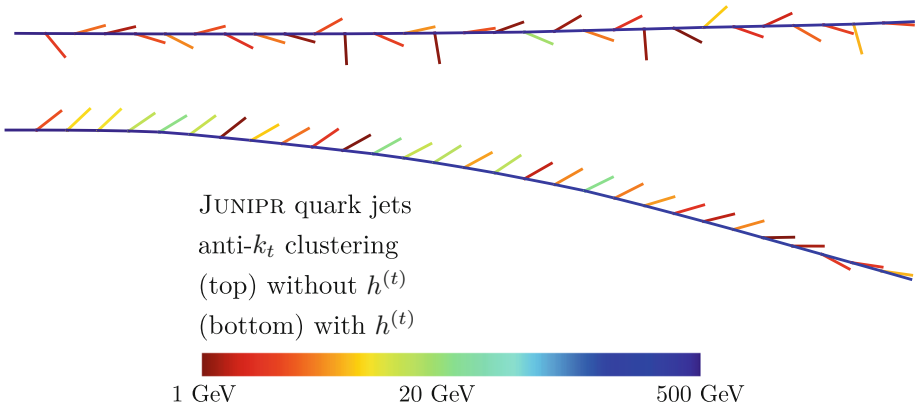

Fig. 23 (Top) Shower sampled from an anti-k_t version of JUNIPR, but one in which the global representation $h^{(t)}$ is withheld from the branching function. Correlation between emission directions is absent in this case. (Bottom) Shower sampled from an anti-k_t version of JUNIPR, using the standard architecture complete with $h^{(t)}$. Strong coherence in emission directions is clearly evident

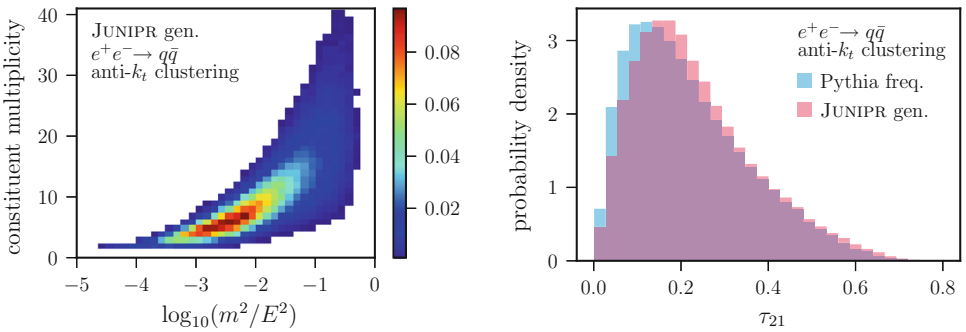

Fig. 24 (Left) 2-dimensional distribution over jet mass and constituent multiplicity, calculated by sampling jets directly from an anti-k_t implementation of JUNIPR. (Right) 2-subjettiness ratio observable distribution sampled from this model and compared to the distribution on PYTHIA jets in the validation set

the one used in Sect. 5.1, in which the global representation $h^{(t)}$ is withheld as input to the branching function.

In Fig. 23 (bottom) we show a jet sampled from P_{anti}. In this case, though the tree itself does not properly guide the collinear structure of emissions, one can see that the emission directions are highly correlated with one another, demonstrating the success of the jet representation $h^{(t)}$ in tracking the global branching pattern. In Fig. 23 (top) we show for comparison a jet sampled from $P_{\text{anti-local}}$, in which the branching function does not receive $h^{(t)}$. In the latter case, all correlation between the emission directions is lost. This shows that the global representation $h^{(t)}$ is crucial for a successful anti-k_t branching model.

In Fig. 24 we show the 2-dimensional distribution over jet mass and constituent multiplicity, as well as the 2-subjettiness distribution, generated with P_{anti}. One can see that the former distribution is consistent with the distribution generated by PYTHIA in Fig. 16. Mild disagreement between P_{anti}'s 2-subjettiness distribution and PYTHIA's can be seen on the right side of Fig. 24. This is on par with the agreement obtained by sampling from the C/A model in Fig. 17.

In Sect. 5.1 we saw that the RNN's hidden state $h^{(t)}$ manages the global information in JUNIPR's neural network architecture. This is an efficient and natural way to characterize QCD-like jets, and therefore also C/A clustering trees. Though JUNIPR is formally independent of jet algorithm (i.e. in the infinite-capacity and perfect-training limit), we might expect JUNIPR's performance to degrade somewhat when paired with clustering algorithms that require significantly more information to be stored in $h^{(t)}$. This was explored in Sects. 5.2 and 5.3 using two separate non-QCD-like clustering algorithms, namely the "printer" and anti-k_t algorithms. Despite these clustering algorithms being unnatural choices for JUNIPR, we were able to demonstrate conceptually interesting and novel results, such as the anti-k_t shower generator. This further demonstrates that JUNIPR can continue to function well, even when the clustering algorithm chosen for implementation bears little resemblance to the underlying physical processes that generate the data.

6 Conclusions and outlook

In this paper, we have introduced JUNIPR as a framework for unsupervised machine learning in particle physics. The framework calls for a neural network architecture designed to efficiently describe the leading-order physics of $1 \to 2$ splittings, alongside a representation of the global jet physics. This requires the momenta in a jet to be clustered into a binary tree. The choice of clustering algorithm is not essential to JUNIPR's performance, but choosing an algorithm that has some correspondence with an underlying physical model, such as the angular-ordered parton shower in quantum chromodynamics, gives improved performance and allows for intrerpretability of the network. At JUNIPR's core is a recurrent neural network with three interconnected components.

It moves along the jet's clustering tree, evaluating the likelihood of each branching. More generally, JUNIPR is a function that acts on a set of 4-momenta in an event to compute their relative differential cross section, i.e. the probability density for this event to occur, given the event selection criteria used to select the training sample. One of the appealing features of JUNIPR is its interpretability: it provides a desconstruction of the probability density into contributions from each point in the clustering history.

There are many promising applications of JUNIPR, and we have only been able to touch on a few proof-of-concept tests in this introductory work. One exciting use case is discrimination. In contrast to supervised models which directly learn to discriminate between two samples, JUNIPR learns the features of the samples separately. It then discriminates by comparing the likelihood of a given event with respect to alternative models of the underlying physics. The resulting likelihood ratio provides theoretically optimal statistical power. As an example, we showed that JUNIPR can discriminate between boosted Z bosons and quark jets (in a very tight mass window around m_Z) in e^+e^- events when trained on the two samples separately. With JUNIPR, it is not only possible to perform powerful discrimination using unsupervised learning, but the discrimination power can be visualized over the entire clustering tree of each jet, as in Fig. 13. This opens new avenues for physicists to gain intuition about the physics underlying high-performance discrimination. Such studies might even inspire the construction of new calculable observables.

Another exciting potential application of JUNIPR is the reweighting of Monte Carlo events, in order to improve agreement with real collider data. A proof-of-concept of this idea was given in Fig. 18, where jets generated with one PYTHIA tune were reweighted to match jets generated with another. The reason this application is important is that current Monte Carlo event generators do an excellent job of simulating events on average, but are limited by the models and parameters within them. It may be easier to correct for systematic bias in event generation by a small reweighting factor appropriate for a particular data sample, rather than by trying to isolate and improve faulty components of the model. In this context, JUNIPR can be thought of as providing small but highly granular tweaks to simulations in order to improve agreement with data.

The JUNIPR framework was used here to compute the likelihood that a given set of particle momenta will arise inside a jet. One can also imagine more general models that act on all the momenta in an entire event, including particle identification tags, or even low-level detector data. A particularly interesting direction would be to consider applying JUNIPR to heavy ion collisions, in which the medium where the jets are produced and evolve is not yet well understood. In this case, comparing the probabilities in data to those of simulation could give insights into how to improve the simulations, or more optimistically, to improve our understanding of the underlying physics.

Acknowledgements We benefited from interesting discussions with D. Barber, E. Bernton, A. Botev, Y.-T. Chien, K. Cranmer, R. Elayavalli, M. Freytsis, B. Gaujac, R. Habib, P. Komiske, E. Metodiev, B. Nachman, and J. Thaler. AA, CF, and MDS are supported in part by the Department of Energy under contract DE-SC0013607. Support for AA and CF was provided in part by the Harvard Data Science Initiative. All compute costs were covered by ASI Data Science through their machine learning platform SherlockML.

Data Availability Statement This manuscript has no associated data or the data will not be deposited. [Authors' comment: All our data is simulated, and it can be reproduced with the parameters provided in this paper.]

Funded by SCOAP[3].

References

1. A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems* (2012), pp. 1097–1105
2. K. He, X. Zhang, S. Ren, J. Sun, *Deep Residual Learning for Image Recognition* (2016), pp. 770–778. arXiv:1512.03385
3. G. Huang, Z. Liu, K.Q. Weinberger, *Densely Connected Convolutional Networks* (2017). arXiv:1608.06993
4. D. Bahdanau, K. Cho, Y. Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate* (2014). arXiv:1409.0473
5. Y. Wu, M. Schuster, Z. Chen, Q.V. Le, M. Norouzi, W. Macherey et al., *Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation*. arXiv:1609.08144
6. A. Graves, N. Jaitly, *Towards End-to-End Speech Recognition with Recurrent Neural Networks* (2014)
7. A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves et al., *Wavenet: A Generative Model for Raw Audio* (2016). arXiv:1609.03499
8. D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors. Nature **323**, 533 (1986)
9. T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, S. Khudanpur, Recurrent neural network based language model, in *Eleventh Annual Conference of the International Speech Communication Association* (2010)
10. ATLAS Collaboration, G. Aad et al., A neural network clustering algorithm for the ATLAS silicon pixel detector. JINST **9**, P09009 (2014). arXiv:1406.7690
11. ATLAS Collaboration, G. Aad et al., Performance of b-jet identification in the ATLAS experiment. JINST **11**, P04008 (2016). arXiv:1512.01094
12. CMS Collaboration, S. Chatrchyan et al., Performance of tau-lepton reconstruction and identification in CMS. JINST **7**, P01001 (2012). arXiv:1109.6034
13. K. Datta, A. Larkoski, How much information is in a jet? JHEP **06**, 073 (2017). arXiv:1704.08249
14. K. Datta, A.J. Larkoski, Novel jet observables from machine learning. JHEP **03**, 086 (2018). arXiv:1710.01305

15. H. Luo, M.-X. Luo, K. Wang, T. Xu, G. Zhu, Quark jet versus gluon jet: deep neural networks with high-level features. arXiv:1712.03634
16. P.T. Komiske, E.M. Metodiev, J. Thaler, Energy flow polynomials: a complete linear basis for jet substructure. arXiv:1712.07124
17. J. Gallicchio, J. Huth, M. Kagan, M.D. Schwartz, K. Black, B. Tweedie, Multivariate discrimination and the Higgs + W/Z search. JHEP **04**, 069 (2011). arXiv:1010.3698
18. ATLAS Collaboration, Identification of hadronically-decaying W bosons and top quarks using high-level features as input to boosted decision trees and deep neural networks in ATLAS at $\sqrt{s} = 13\ TeV$, in *Technical Report*, *ATL-PHYS-PUB-2017-004*. CERN, Geneva (2017)
19. J. Cogan, M. Kagan, E. Strauss, A. Schwartzman, Jet-Images: computer vision inspired techniques for jet tagging. JHEP **02**, 118 (2015). arXiv:1407.5675
20. L. de Oliveira, M. Kagan, L. Mackey, B. Nachman, A. Schwartzman, Jet-images deep learning edition. JHEP **07**, 069 (2016). arXiv:1511.05190
21. P.T. Komiske, E.M. Metodiev, M.D. Schwartz, Deep learning in color: towards automated quark/gluon jet discrimination. JHEP **01**, 110 (2017). arXiv:1612.01551
22. P.T. Komiske, E.M. Metodiev, B. Nachman, M.D. Schwartz, Pileup mitigation with machine learning (PUMML). JHEP **12**, 051 (2017). arXiv:1707.08600
23. G. Kasieczka, T. Plehn, M. Russell, T. Schell, Deep-learning top taggers or the end of QCD? JHEP **05**, 006 (2017). arXiv:1701.08784
24. W. Bhimji, S.A. Farrell, T. Kurth, M. Paganini, Prabhat, E. Racah, Deep neural networks for physics analysis on low-level whole-detector data at the LHC, in *18th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2017)* Seattle, WA, USA, August 21–25, 2017 (2017). arXiv:1711.03573
25. ATLAS Collaboration , Quark versus gluon jet tagging using jet images with the ATLAS detector, in *Technical Report ATL-PHYS-PUB-2017-017*. CERN, Geneva (2017)
26. S. Macaluso, D. Shih, *Pulling Out All the Tops with Computer Vision and Deep Learning*. arXiv:1803.00107
27. Y.-T. Chien, R. Kunnawalkam Elayavalli, Probing heavy ion collisions using quark and gluon jet substructure. arXiv:1803.03589
28. J. Pearkes, W. Fedorko, A. Lister, C. Gay, *Jet Constituents for Deep Neural Network Based Top Quark Tagging*. arXiv:1704.02124
29. G. Louppe, K. Cho, C. Becot, K. Cranmer, *QCD-Aware Recursive Neural Networks for Jet Physics*. arXiv:1702.00748
30. T. Cheng, *Recursive Neural Networks in Quark/Gluon Tagging*. arXiv:1711.02633
31. S. Egan, W. Fedorko, A. Lister, J. Pearkes, C. Gay, *Long Short-Term Memory (LSTM) Networks with Jet Constituents for Boosted Top Tagging at the LHC*. arXiv:1711.09059
32. K. Fraser, M.D. Schwartz, *Jet Charge and Machine Learning*. arXiv:1803.08066
33. D. Guest, J. Collado, P. Baldi, S.-C. Hsu, G. Urban, D. Whiteson, Jet flavor classification in high-energy physics with deep neural networks. Phys. Rev. D **94**, 112002 (2016). arXiv:1607.08633
34. ATLAS Collaboration, Identification of jets containing *b*-hadrons with recurrent neural networks at the ATLAS experiment, in *Technical Report ATL-PHYS-PUB-2017-003*. CERN, Geneva (2017)
35. E.M. Metodiev, B. Nachman, J. Thaler, Classification without labels: learning from mixed samples in high energy physics. JHEP **10**, 174 (2017). arXiv:1708.02949
36. T. Cohen, M. Freytsis, B. Ostdiek, (Machine) learning to do more with less. JHEP **02**, 034 (2018). arXiv:1706.09451
37. P.T. Komiske, E.M. Metodiev, B. Nachman, M.D. Schwartz, *Learning to Classify from Impure Samples*. arXiv:1801.10158
38. E.M. Metodiev, J. Thaler, *On the Topic of Jets*. arXiv:1802.00008
39. L. de Oliveira, M. Paganini, B. Nachman, Learning particle physics by example: location-aware generative adversarial networks for physics synthesis. Comput. Softw. Big Sci. **1**, 4 (2017). arXiv:1701.05927
40. M. Paganini, L. de Oliveira, B. Nachman, Accelerating science with generative adversarial networks: an application to 3D particle showers in multilayer calorimeters. Phys. Rev. Lett. **120**, 042003 (2018). arXiv:1705.02355
41. M. Paganini, L. de Oliveira, B. Nachman, CaloGAN : simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks. Phys. Rev. D **97**, 014021 (2018). arXiv:1712.10321
42. J. Neyman, E.S. Pearson, Ix. on the problem of the most efficient tests of statistical hypotheses. Philos. Trans. R. Soc. Lond. A Math. Phys. Eng. Sci. **231**, 289–337 (1933). http://rsta.royalsocietypublishing.org/content/231/694-706/289.full.pdf
43. A. Butter, G. Kasieczka, T. Plehn, M. Russell, Deep-learned top tagging with a Lorentz layer. arXiv:1707.08966
44. S. Coleman, R. Norton, Singularities in the physical region. Nuovo Cim. **38**, 438–442 (1965)
45. J.C. Collins, D.E. Soper, G.F. Sterman, Factorization for short distance hadron–hadron scattering. Nucl. Phys. B **261**, 104 (1985)
46. J.C. Collins, D.E. Soper, G.F. Sterman, Soft gluons and factorization. Nucl. Phys. B **308**, 833 (1988)
47. I. Feige, M.D. Schwartz, Hard-soft-collinear factorization to all orders. Phys. Rev. D **90**, 105020 (2014)
48. I. Feige, M.D. Schwartz, An on-shell approach to factorization. Phys. Rev. D **88**, 065021 (2013)
49. S. Catani, Y.L. Dokshitzer, M.H. Seymour, B.R. Webber, Longitudinally invariant K_t clustering algorithms for hadron hadron collisions. Nucl. Phys. B **406**, 187–224 (1993)
50. S.D. Ellis, D.E. Soper, Successive combination jet algorithm for hadron collisions. Phys. Rev. D **48**, 3160–3166 (1993). arXiv:hep-ph/9305266
51. Y.L. Dokshitzer, G.D. Leder, S. Moretti, B.R. Webber, Better jet clustering algorithms. JHEP **08**, 001 (1997). arXiv:hep-ph/9707323
52. M. Wobisch, T. Wengler, Hadronization corrections to jet cross-sections in deep inelastic scattering, in *Monte Carlo Generators for HERA Physics. Proceedings, Workshop, Hamburg, Germany 1998–1999*, pp. 270–279 (1998). arXiv:hep-ph/9907280
53. S.D. Ellis, A. Hornig, T.S. Roy, D. Krohn, M.D. Schwartz, Qjets: a non-deterministic approach to tree-based jet substructure. Phys. Rev. Lett. **108**, 182003 (2012). arXiv:1201.1914
54. D. Kahawala, D. Krohn, M.D. Schwartz, Jet sampling: improving event reconstruction through multiple interpretations. JHEP **06**, 006 (2013). arXiv:1304.2394
55. L. Mackey, B. Nachman, A. Schwartzman, C. Stansbury, Fuzzy jets. JHEP **06**, 010 (2016). arXiv:1509.02216
56. D.E. Soper, M. Spannowsky, Finding physics signals with shower deconstruction. Phys. Rev. D **84**, 074002 (2011). arXiv:1102.3480
57. D.E. Soper, M. Spannowsky, Finding physics signals with event deconstruction. Phys. Rev. D **89**, 094005 (2014). arXiv:1402.1189
58. M. Cacciari, G.P. Salam, G. Soyez, The anti-$k(t)$ jet clustering algorithm. JHEP **04**, 063 (2008). arXiv:0802.1189
59. M. Cacciari, G.P. Salam, G. Soyez, FastJet user manual. Eur. Phys. J. C **72**, 1896 (2012). arXiv:1111.6097
60. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, New York, 2016)
61. K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk et al., Learning phrase representations using RNN encoder–decoder for statistical machine translation, in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734. Association for Computational Linguistics (2014). https://doi.org/10.3115/v1/D14-1179

62. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Comput. **9**, 1735–1780 (1997)
63. T. Sjostrand, S. Mrenna, P.Z. Skands, PYTHIA 6.4 physics and manual. JHEP **05**, 026 (2006). arXiv:hep-ph/0603175
64. T. Sjöstrand, S. Ask, J.R. Christiansen, R. Corke, N. Desai, P. Ilten et al., An introduction to PYTHIA 8.2. Comput. Phys. Commun. **191**, 159–177 (2015). arXiv:1410.3012
65. Theano Development Team Collaboration, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas et al., Theano: a Python framework for fast computation of mathematical expressions. arXiv e-prints: arXiv:1605.02688 (2016)
66. F. Morin, Y. Bengio, Hierarchical probabilistic neural network language model, in *AISTATS'05*, pp. 246–252 (2005)
67. A. Mnih, G.E. Hinton, A scalable hierarchical distributed language model, in *Advances in Neural Information Processing Systems*, vol. 21, ed. by D. Koller, D. Schuurmans, Y. Bengio, L. Bottou (Curran Associates Inc., 2009), pp. 1081–1088
68. J. Thaler, K. Van Tilburg, Identifying boosted objects with *N*-subjettiness. JHEP **03**, 015 (2011). arXiv:1011.2268
69. Y.-T. Chien, A. Emerman, S.-C. Hsu, S. Meehan, Z. Montague, *Telescoping Jet Substructure*. arXiv:1711.11041
70. J. Gallicchio, M.D. Schwartz, Seeing in color: jet superstructure. Phys. Rev. Lett. **105**, 022001 (2010). arXiv:1001.5027