Jet Flavor Classification in High-Energy Physics with Deep Neural Networks

Daniel Guest*,[1] Julian Collado*,[2] Pierre Baldi,[2] Shih-Chieh Hsu,[3] Gregor Urban,[2] and Daniel Whiteson[1]

[1]*Department of Physics and Astronomy, University of California, Irvine, CA 92697*
[2]*Department of Computer Science, University of California, Irvine, CA 92697*
[3]*Department of Physics, University of Washington, Seattle, WA 98195*

(Dated: September 9, 2016)

Classification of jets as originating from light-flavor or heavy-flavor quarks is an important task for inferring the nature of particles produced in high-energy collisions. The large and variable dimensionality of the data provided by the tracking detectors makes this task difficult. The current state-of-the-art tools require expert data-reduction to convert the data into a fixed low-dimensional form that can be effectively managed by shallow classifiers. We study the application of deep networks to this task, attempting classification at several levels of data, starting from a raw list of tracks. We find that the highest-level lowest-dimensionality expert information sacrifices information needed for classification, that the performance of current state-of-the-art taggers can be matched or slightly exceeded by deep-network-based taggers using only track and vertex information, that classification using only lowest-level highest-dimensionality tracking information remains a difficult task for deep networks, and that adding lower-level track and vertex information to the classifiers provides a significant boost in performance compared to the state-of-the-art.

PACS numbers:

INTRODUCTION

The search for new particles and interactions at the energy frontier is a rich program with enormous discovery potential. The power to discover this hypothetical new physics relies crucially on the ability to infer the nature of the interaction and the particles produced from the data provided by the detectors which surround the point of collision. One critical element is jet flavor classification, the distinction between hadronic jets produced from light-flavor and heavy-flavor quarks. Such classification plays a central role in identifying heavy-flavor signals and reducing the enormous backgrounds from light-flavor processes [1, 2].

Jets originating from heavy-flavor quarks tend to produce longer-lived particles than those found in jets from light-flavor quarks; these long-lived particles have decays which are displaced from the primary vertex. To identify such vertices, the central tracking chamber measures the trajectories of charged particles which allows for the reconstruction of vertex locations. The large and varying number of particles in a jet leads to a difficult classification problem with large and variable dimensionality without a natural ordering. The first step in typical approaches involves vertex-finding algorithms [3], which transform the task into one of reduced, but still variable, dimensionality. Finally, most state-of-the-art jet flavor classification tools used by experiments [4, 5] rely heavily on expert-designed features which fix and further reduce the dimensionality before applying shallow machine-learning techniques. Such techniques have excellent performance, but are primarily motivated by historical limitations in the ability of shallow learning methods to handle high- and variable-dimensionality datasets.

Recent applications of deep learning to similar problems in high-energy physics [6–9], combined with the lack of a clear analytical theory to provide dimensional reduction without loss of information, suggests that deep learning techniques applied to the lower-level higher-dimensional data could yield improvements in the performance of jet-flavor classification algorithms. General methods for designing and applying recurrent and recursive neural networks to problems with data of variable size or structure have been developed in Refs. [10–14], and applied systematically to a variety of problems ranging from natural language processing [15], to protein structure prediction [16–19] to prediction of molecular properties [20, 21] and to the game of go [22]; previous studies have discussed the extension of such strategies to tasks involving tracks in high energy physics [23, 24].

In this paper, we apply several deep learning techniques to this problem using a structured dataset with features at three levels of processing (tracks, vertices, expert), each of which is a strict function of the previous level(s). The data at the highest level of processing, with smallest dimensionality, is intended to mirror the typical approach used currently by experimental collaborations. The multi-layered structure of the dataset allows us to draw conclusions about the information loss at each stage of processing, and to gauge the ability of machine learning tools to find solutions in the lower- and higher-dimensional levels. These lessons can guide the design of

*The first two authors contributed equally

flavor-tagging algorithms used by experiments.

CLASSIFICATION AND DIMENSIONALITY

The task of the machine learning (ML) algorithm is to identify a function $f(\bar{x}) : \mathbb{R}^N \to \mathbb{R}^1$ whose domain is the observed data at some level of processing (with potentially very large dimensionality N) and which evaluates to a single real value that contains the information necessary to perform the classification. Perfect classification is not expected; instead, the upper bound is performance which matches classification provided by the true likelihood ratio between heavy-flavor (b) and light-flavor quarks (q): $P(\bar{x}|b)/P(\bar{x}|q)$ evaluated in the high-dimensional domain.

Though we lack knowledge of an analytical expression for the likelihood, in principle one could recover such a function from labeled datasets with trivial algorithms, by estimating the likelihood directly in the original high-dimensional space. In practice, this requires an enormous amount of data, making it impractical for problems with anything but the smallest dimensionality in their feature space.

Machine learning plays a critical role in approximating the function $f(\bar{x})$ which reduces the dimensionality of the space to unity by finding the critical information needed to perform the classification task. Such a function may disregard some of the information from the higher-dimensional space if it is not pertinent to the task at hand. However, for very high dimensional spaces (greater than ≈ 50), the task remains very difficult, and until the recent advent of deep learning it appeared to be overwhelming, though it can still require the generation of large samples of training data.

It would be very powerful to compare the performance of a given solution to the theoretical upper limit on performance, provided by the true likelihood. Unfortunately, without knowledge of the true likelihood, it is difficult to assess how well the ML algorithm has captured the necessary information. For this reason, in the studies presented here and in earlier work [6, 7, 9], we built structured datasets with at least two levels of dimensionality: an initial sample with lower-level data at high dimensionality and a reduced sample with expert features at lower dimensionality. Importantly, the expert features are a strict function of the lower-level features, so that they contain a subset of the information. The expertise lies solely in the design of the dimensionality-reducing function, without providing any new information.

This structure allows us to draw revealing conclusions about the information content of the intermediate and expert-level information and the power of classifiers to extract it. Since the higher-level data contains a subset of the information and benefits from expert knowledge, it can provide the basis for a performance benchmark for the tools using lower-level data in place of the unknown true likelihood. Therefore, if the performance of tools using lower-level data fails to match that of tools using the higher-level data (or a combination of both kinds of data), then we may conclude that the tools using the lower-level data have failed to extract the complete information. On the other hand, if the performance of tools using lower-level data exceeds that of tools using the higher-level data, then we may conclude that the higher-level data does not contain all of the information relevant to the classification task, or that it has transformed the problem into a more difficult learning task for the algorithms considered. Regardless of the reason, in this case the transformation to the higher-level lower-dimensional data has failed in its goal.

DATA

Training samples were produced with realistic simulation tools widely used in particle physics. Samples were generated for three classes of jet:

- light-flavor: jets from u, d, s quarks or gluons;
- charm: jets from c quarks;
- heavy-flavor: jets from b quarks.

Collisions and immediate decays were generated with MADGRAPH5 [25] v2.2.3, showering and hadronization simulated with PYTHIA [26] v6.428, and response of the detectors simulated with DELPHES [27] v3.2.0. Studies with additional pp interactions (*pileup*) are reserved for future work; here we assume that pileup effects will not alter the relative performance of the different methods, and is not likely to have a large impact at luminosities recorded to date, given effective techniques to isolate pileup tracks and vertices from the vertices of interest to this study.

The detector simulation was augmented with a simple tracking model that smears truth particles to yield tracks similar to those expected at ATLAS [28]. Tracks follow helical paths in a perfectly homogeneous 2 T magnetic field. No attempt was made to account for material interactions or remove strange hadrons. As a result the tracking model lacks the sophistication of models developed by LHC collaborations while retaining enough realism to run vertex reconstruction and compare the relative performance of various machine learning approaches.

Jets are reconstructed from calorimeter energy deposits with the anti-k_{T} clustering algorithm [29] as implemented in FastJet [30], with a distance parameter of $R = 0.4$. Tracks are assigned to jets by requiring that they be within a cone of $\Delta R \equiv (\Delta\eta^2 + \Delta\phi^2)^{1/2} < 0.4$ of the jet axis. Jets are labeled by matching to partons within a cone of $\Delta R < 0.5$. If a b or c quark is found

within this cone the jet is labeled heavy or charm flavor respectively, with b taking precedence if both are found. Otherwise the jet is labeled light-flavor.

To reconstruct secondary vertices, we use the adaptive vertex reconstruction algorithm implemented in RAVE v6.24 [3, 31]. The algorithm begins by fitting a primary vertex to the event and removing all compatible tracks. For each jet, secondary vertices are then reconstructed iteratively: a vertex is fit to a point that minimizes χ^2 with respect to all tracks in the jet, less compatible tracks are down-weighted, and the vertex fit is repeated until the fit stabilizes.

Since a b-hadron decay typically cascades through a c-hadron, jets may include multiple secondary vertices. To account for this, tracks with large weights in the secondary vertex fit are removed and the fit is repeated with the remaining tracks. The process repeats until all tracks are assigned to a secondary vertex.

As described earlier, we organize the information in three levels of decreasing dimensionality and increasing pre-processing using expert knowledge where each level is a strict function of the lower-level information. The classification is done per-jet rather than per-event, and at every level the transverse momentum and pseudorapidity of the jet is included.

The lowest-level information considered is the list of reconstructed tracks. Each helical track has five parameters in addition to a 5×5 symmetric covariance matrix with 15 independent entries. The number of tracks varies from 1 to 33 in these samples, with a mean of 4.

The intermediate-level information comes from the output of the vertexing algorithm. The features are the vertex mass, number of tracks associated to the vertex, the fraction of the total energy in jet tracks which is associated to those tracks, vertex displacement, vertex displacement significance, and angular separation in $\Delta\eta$ and $\Delta\phi$ with respect to the jet axis for each vertex. In cases where both low and intermediate level features are used the track to vertex association weight is also included. The number of vertices varies from 1 to 13 in these samples, with a mean of 1.5.

The highest-level information is designed to model the typical features used in current experimental applications; see Fig. 1 for distributions of these features for each jet class. There are fourteen such features:

- The d_0 and z_0 significance of the 2nd and 3rd tracks attached to a vertex, ordered by d_0 significance.
- The number of tracks with d_0 significance greater than 1.8σ.
- The JetProb [32] light jet probability, calculated as the product over all tracks in the jet of the probability for a given track to have come from a light-quark jet.
- The width of the jet in η and ϕ, calculated for η as

$$\left(\frac{\sum_i p_{\mathrm{T}i}\Delta\eta_i^2}{\sum_i p_{\mathrm{T}}}\right)^{1/2}$$

  and analogously for ϕ.
- The combined vertex significance,

$$\frac{\sum_i d_i/\sigma_i^2}{\sqrt{\sum_i 1/\sigma_i^2}}$$

  where d is the vertex displacement and σ is the uncertainty in vertex position along the displacement axis.
- The number of secondary vertices.
- The number of secondary-vertex tracks.
- The angular distance ΔR between the jet and vertex.
- The decay chain mass, calculated as the sum of the invariant masses of all reconstructed vertices, where particles are assigned the pion mass.
- The fraction of the total track energy in the jet associated to secondary vertices [1]

The dataset consists of 10 million labeled simulated jets. The corresponding target labels are light-flavor, charm, and heavy-flavor. The data contains 44, 11, 45 percent of each class respectively. This data is available from the UCI Machine Learning in Physics Web portal at `http://mlphysics.ics.uci.edu/`.

METHODS

In the experiments, we typically use 8 million samples for training, one million for validation, and one million for testing. Since there are three labels but we are interested in the study of signal vs background and classification, the labels are converted to binary by mapping bottom quark to one, and both charm and light quark to zero. We study the light-quark and charm-quark rejection separately.

Machine Learning Approaches

To each simulated collision is attached a set of tracks and a set of vertices. This poses challenges for a machine learning approach in that the size of these sets is

[1] The vertex energy fraction is not a strict fraction; it can be greater than unity if tracks are assigned to multiple vertices.

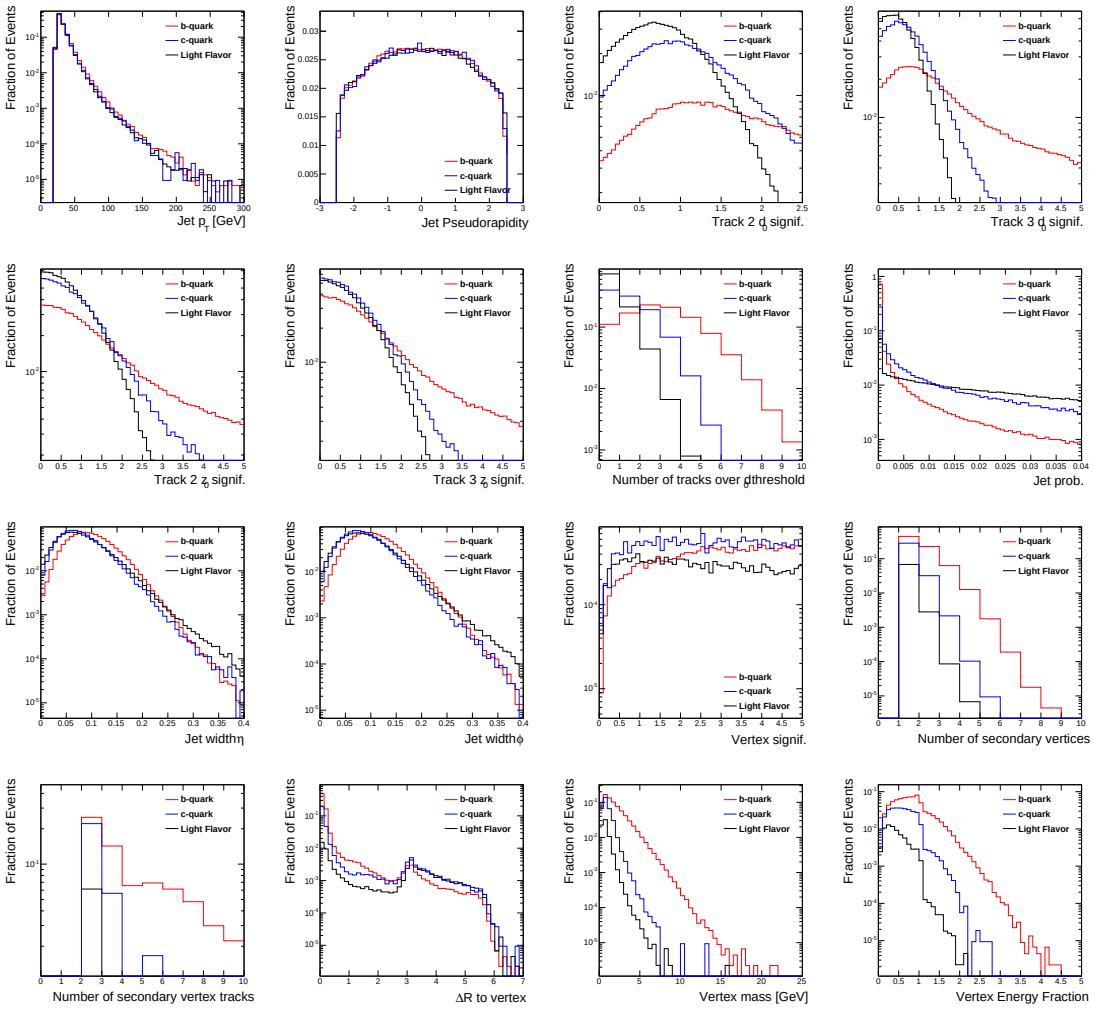

FIG. 1: Distributions in simulated samples of high-level jet flavor variables widely used to discriminate between jets from light-flavor and heavy-flavor quarks.

variable as seen in Fig. 2 and the sets are unordered, although as usual an arbitrary order is often used to list their elements. To address and explore these challenges we use three different deep learning approaches: feedforward neural networks, recurrent neural networks with LSTM (Long Short Term Memory) units, and outer recursive neural networks.

Feedforward Neural Networks

The track feature set and the vertex feature set have variable size for a given collision. However, the structure of feedforward networks requires a fixed-size input to make predictions. Thus the use of feedforward neural networks requires first an arbitrary ordering and then a capping of the size of the input set, with zero padding for sets that are smaller than the capped size. To resolve the arbitrary ordering the tracks were sorted by decreasing absolute d_0 significance. This ordering also ensures that tracks from a secondary vertex, which typically have large d_0, are unlikely to be removed by the capping. Random ordering before adding the padding was also tested but the performance was lower than using the absolute d_0 significance ordering.

To create a fixed size input, the number of tracks was limited to 15, from a maximum of 33. Using 15 as the cutoff value ensures that 99.97% of the samples preserve all their original tracks; see Fig. 2. Tracks are associated to vertices by concatenating the track parameters with those from the associated vertex. Before training, the samples are preprocessed by shifting and scaling such that each feature has a mean of zero and a standard deviation of one. Jets with fewer than 15 tracks are zero-padded after preprocessing. After the cut on the number of tracks, the maximum number of vertices is 12 with an

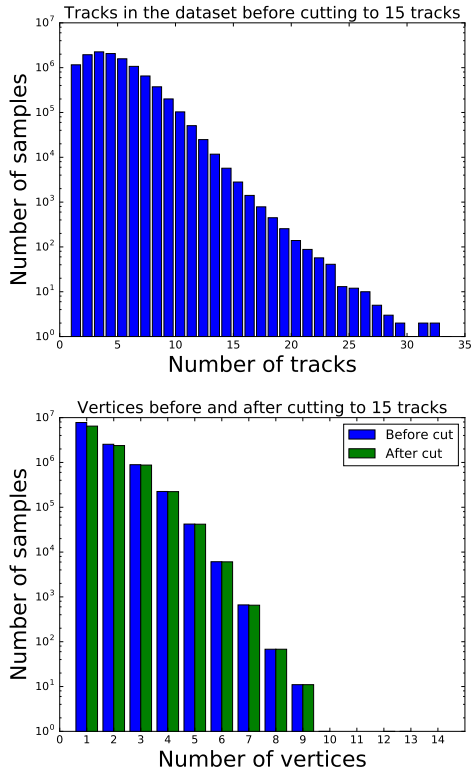

FIG. 2: Top: Distribution of the number of tracks associated to a jet in simulated samples. Bottom: Distribution of the number of vertices associated to a jet in simulated samples, before and after removing tracks which exceed the maximum allowed value of 15.

average of 1.5; see Fig. 2.

The feedforward neural networks were trained on 8 million training samples with one million more for validation using stochastic gradient descent with mini-batches of 100 samples. They were trained for 100 epochs and the best model was chosen based on the validation error. Momentum for the weights updated was used and linearly increased from zero to a final value over a specified number of epochs. Learning rate decayed linearly from 0.01 to a final value starting and finishing at a specified number of epochs. Dropout (in which nodes are removed during training) with values of p from 0.0 to 0.5 were used at several combinations of layers to add regularization [33, 34]. These networks had 9 fully connected hidden layers with rectified linear units [35, 36].

Shared weights for each track object were used at the first layer to preserve information about the structure of the data; see Fig 3. When adding the vertex and high level variables to the tracks, these were also included within the set of variables with shared weights. The weights for all but the last layer were initialized from a uniform distribution between $[-\sqrt{6/C}, \sqrt{6/C}]$ where C is the total number of incoming and outgoing connections [37]. The weights for the last layer were initialized from a uniform distribution between -0.05 and 0.05. A manual optimization was performed over all the hyperparameters to find the best model.

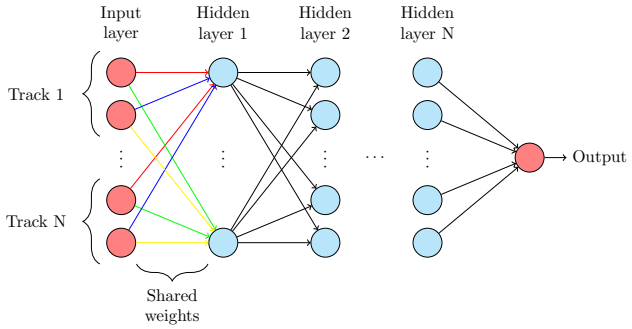

FIG. 3: Feedforward neural network architecture. In the first layer, connections of the same color represent the same value of the shared weight. The others layers are fully connected without shared weights.

LSTM Networks

A natural approach to handling variable-sized input is to use recursive neural networks. Broadly speaking, there are two classes of approaches for designing such architectures, the inner approach and the outer approach [38]. In the inner approach, neural networks are used inside the data graphs to crawl the corresponding edges and compute the final output. This process requires the data graphs to be directed and acyclic. Since here the data consists of a set of vertices and tracks, we first convert the data into a sequence by ordering the vertices and tracks as described previously and then use recursive neural networks for sequences, in combination with Long Short Term Memory units [39, 40] to better capture long range dependencies. In the underlying acyclic graph, the variables associated with each node are a function of the variables associated with the parent nodes. Each such function can be parameterized by a neural network. Because the directed acyclic graph has a regular structure, the same network can be applied at different locations of the graph, ultimately producing the LSTM grid network in Figure 4.

We follow the standard implementation of LSTMs with three gates (input, forget, output) and initialize the connections to random orthonormal matrices. The input data consists of a sequence of concatenated track, vertex, and expert features (or different sub-combinations thereof) which are sorted by their absolute d_0 significance, as was the case with the fully connected models. The main difference is that we do not need zero-padding as the LSTM networks can handle sequences of

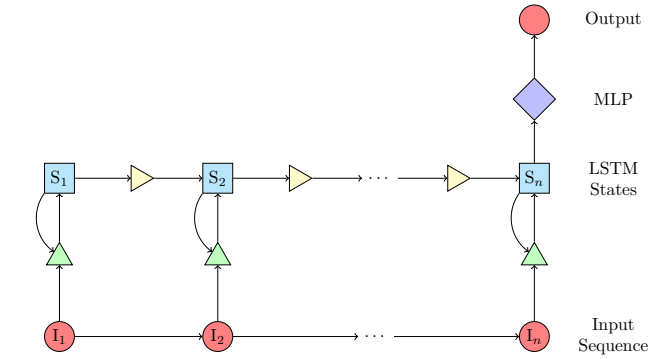

FIG. 4: Architecture of the Long Short Term Memory networks as described in the text.

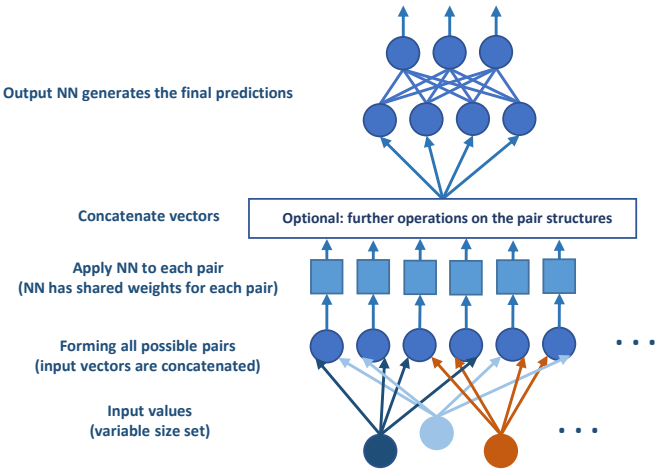

FIG. 5: Architecture of the outer recursive networks as described in the text.

arbitrary length, though we retain the same maximum of 15 tracks for comparability. The final model consists of one LSTM layer comprising between 50 and 500 neurons, and a feedforward neural network with one to four hidden layers that receives its input from the LSTM network and produces the final predictions (where each layer has between 50 and 500 units). We add dropout layers in between the LSTM and each hidden fully connected layer. For hyperparameter-optimization we performed a random search over these parameters as well as the individual dropout rates that are part of the model. We trained the LSTM networks for 100 epochs using SGD with a momentum of 0.9 and decay the step-size parameter from initially $2 \cdot 10^{-3}$ down to 10^{-4} over the course of training.

Outer Recursive Networks

Alternatively, to handle inputs of variable size, we can use an outer recursive approach, where neural networks are built in a direction perpendicular to the original data graph, with horizontal weight sharing. The outer approach can be used to build more symmetric deep architectures; see Fig. 5. For instance, in our case the input consists of up to 15 tracks, from which we can sample all possible pairs of tracks and use a shared neural network that processes these in the first layer of the outer approach. In this case, there are at most $\binom{15}{2} = 105$ unordered pairs, or 210 ordered pairs, which is manageable especially considering that there is a single network shared by all pairs. Using ordered pairs would yield the most symmetric overall network. At the next level of the architecture, one can for instance use a network for each track t_i that combines the outputs of all the networks from the first layer associated with pairs containing t_i, and so forth. In the second level of the outer architecture, for simplicity here we use a fully connected feedforward network that computes the final output using the outputs of all the pair networks. More specifically, for each data sample we compute the list of stacked track features for all 210 pairs and process each pair with a shared nonlinear hidden layer (with 5 to 20 neurons). The resulting outputs for all pairs are then concatenated and fed into a multilayer perceptron as was the case for the LSTM models, with one to four hidden layers containing between 100 and 600 hidden units. We again use dropout layers in between the hidden layers and optimize the dropout rates and network depth and size using random search.

Hardware and Software Implementations

All computations were performed using machines with 16 Intel Xeon cores, NVIDIA Titan graphics processors, and 64 GB memory. All neural networks were trained using the GPU-accelerated Theano software library [41] and, for the feed forward neural networks, also the Keras software library [42].

RESULTS

The best feedforward neural networks have 9 fully connected hidden layers with 400 rectified linear units and a single sigmoid unit at the end. On the first layer the networks have shared weights. The first five tracks have one set of shared weights per track, tracks 6 to 10 have a second set of shared weights per track and the last five tracks have a third set of shared weights per track. They have a momentum term of 0 which starts to linearly increase at the first epoch and reaches its final value of 0.5

at epoch 100. Initially, the learning rate is set at 0.01 and, starting at epoch 80, it is linearly decreased to a final value of 0.001 at epoch 100. Dropout was used in the first two layers with a value of p=0.3. The same architecture was used across all the combinations of features except in the case of using only high level features, in which case the first layer is fully connected without any shared weights.

We found that the main characteristic of the best LSTM models is a relatively small size of the hidden state representation of the LSTM module (about 70 units), while the size of the MLP, which is sitting on top of it, is of secondary importance for overall performance of the model. The best models using the outer recursive approach contain between two and three hidden layers on top of the shared-weight layer (which operates on all paired tracks) and those contain 17 or more neurons.

Final results are shown in Table I. The metric used is the Area Under the Curve (AUC), calculated in signal efficiency versus background efficiency, where a larger AUC indicates better performance. In Fig. 6, the signal efficiency is shown versus background rejection, the inverse of background efficiency. Figures 7 and 8 show the efficiency versus jet p_{T} and pseudorapidity for fixed values of background rejection. Figures 9 and 10 show the rejection versus jet p_{T} and pseudorapidity for fixed values of signal efficiency.

The results can be analyzed to draw conclusions regarding the power of the learning algorithms to extract information at different levels of preprocessing, and to compare the three learning approaches.

The state-of-the-art performance is represented by the networks which use only the expert-level features. Networks using only tracking or vertexing features do not match this performance, though networks using both tracking and vertexing do slightly exceed it. In addition, networks which combine expert-level information with track and/or vertex information outperform the expert-only benchmark, in some cases by a significant margin.

For any given set of features, the feedforward deep networks most often give the best performance, though in some cases by a small margin over the LSTM approach. This may be somewhat unexpected since LSTMs were created to handle variable sized input data as is the case here. We must note, however, that unlike truly sequential data like speech or text there is no natural order in the data that we are working on. The tracks have been ordered by absolute d_0 significance, which tends to cluster tracks belonging to the same vertex, but a sequential model with this ordering may not be superior to processing tracks in parallel, as in the connected DNN with tied weights.

While one cannot probe the strategy of the ML algorithm, it is possible to compare distributions of events categorized as signal-like by the different algorithms in order to understand how the classification is being accomplished. To compare distributions between different algorithms, we study simulated events with equivalent background rejection, see Fig. 11 for a comparison of the selected regions in the expert features for classifiers with and without the lower-level information.

TABLE I: Performance results for networks using track-level, vertex-level or expert-level information. In each case the jet p_T and pseudorapidity are also used. Shown for each method is the Area Under the Curve (AUC), the integral of the background efficiency versus signal efficiency, which have a statistical uncertainty of 0.001 or less. Signal efficiency and background rejections are shown in Figs. 6-10.

Inputs			Technique	AUC
Tracks	Vertices	Expert		
✓			Feedforward	0.916
✓			LSTM	0.917
✓			Outer	0.915
	✓		Feedforward	0.912
	✓		LSTM	0.911
	✓		Outer	0.911
✓	✓		Feedforward	0.929
✓	✓		LSTM	0.929
✓	✓		Outer	0.928
		✓	Feedforward	0.924
		✓	LSTM	0.925
		✓	Outer	0.924
✓		✓	Feedforward	0.937
✓		✓	LSTM	0.937
✓		✓	Outer	0.936
	✓	✓	Feedforward	0.931
	✓	✓	LSTM	0.930
	✓	✓	Outer	0.929
✓	✓	✓	Feedforward	0.939
✓	✓	✓	LSTM	0.939
✓	✓	✓	Outer	0.937

DISCUSSION

Our experiments support four conclusions.

The existing expert strategies for dimensional reduction sacrifice or distort useful information. Networks which include lower-level information outperform networks using exclusively higher-level information. For example, if the vertex-level information contained all of the classification power of the track-level information but with lower dimensionality, one would expect the vertex-only network to match the performance of the tracks-and-vertex network, as the lower-dimensional problem should be simpler to learn. Instead, networks using tracks and vertices outperform those which use only vertices. Similarly, networks using tracks and expert features outperform those with only expert features. We

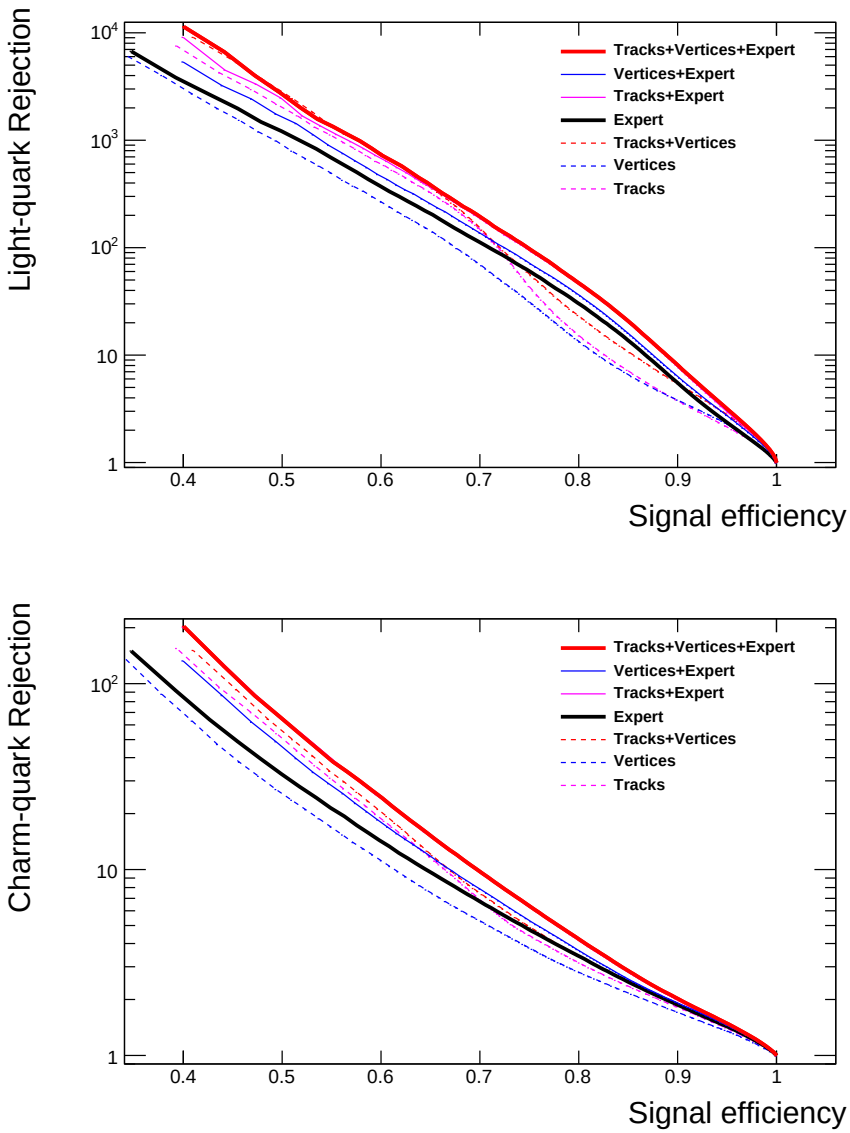

FIG. 6: Signal efficiency versus background rejection (inverse of efficiency) for deep networks trained on track-level, vertex-level or expert-level features. The top pane shows the performance for b-quarks versus light-flavor quarks, the bottom pane for b-quarks versus c-quarks.

note that these conclusions apply to the expert strategies considered here, and in the case of the simulated environment we have studied; however, we feel that both are representative of the current state-of-the-art.

The task remains a challenge for deep networks. Networks which use only the lower-level information do not match the performance of networks which use the higher-level information. Since the higher-level features are strict functions of the lower-level features, the lower-level features are a superset of the information contained in the high-level features. The performance of the networks which use the high-level features then provides a baseline against which to measure the ability of the network to extract the relevant information in the more difficult higher-dimensional space of lower-level features. Networks using only track information do not match the performance of those which use only the high-level features (but note that track-only networks outperform vertex-only networks, giving a clue as to the area of difficulty).

Networks using track and vertex information outperform those with expert features. Networks trained with track and vertex information but without the benefit of expert-level guidance and dimensional re-

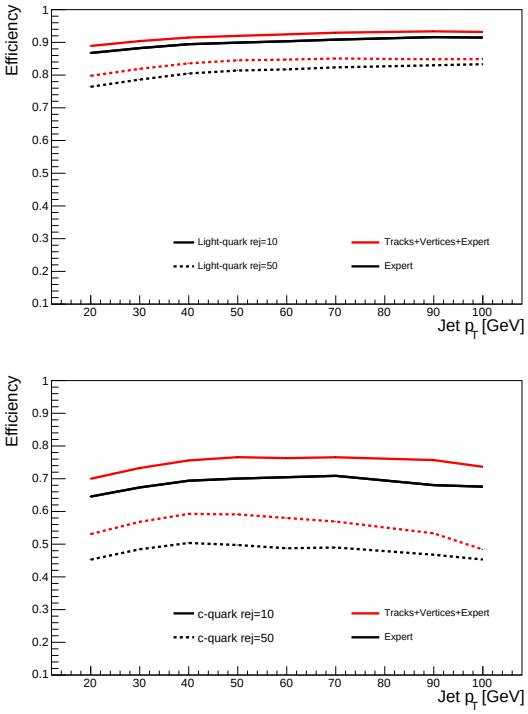

FIG. 7: Signal efficiency versus minimum jet p_{T} relative to light quarks (top) or charm quarks (bottom). In each case, efficiency is shown for fixed values of background rejection for networks trained with only expert features or networks trained with all features (tracks, vertices and expert features).

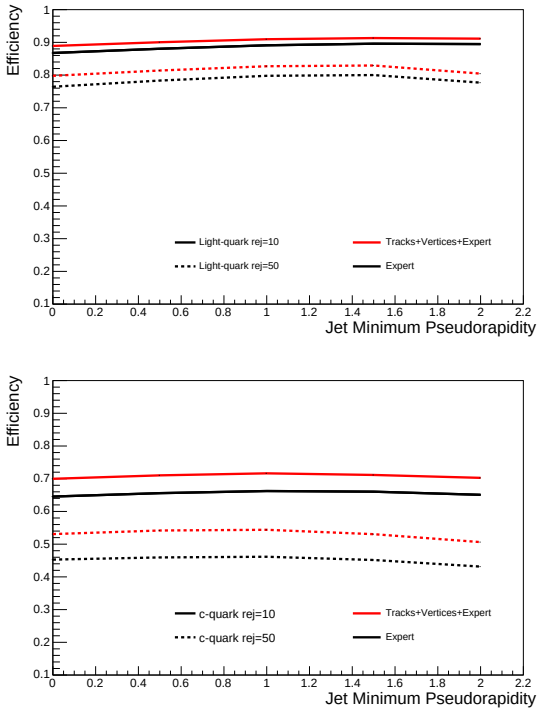

FIG. 8: Signal efficiency versus minimum jet pseudo-rapidity relative to light quarks (top) or charm quarks (bottom). In each case, efficiency is shown for fixed values of background rejection for networks trained with only expert features or networks trained with all features (tracks, vertices and expert features).

duction manage to achieve better performance than those which use only expert-level features. This is remarkable, as the dimensionality of the tracks+vertices features is very large and expert-only networks represent the current state-of-the-art. Note, however, that for high signal efficiency ($> 75\%$) the expert-only networks outperform the networks using tracks+vertices.

Networks which combine expert features with low-level information have the best performance. Combining the lowest-level information for completeness with the low-dimensional hints from expert features significantly outperforms the state-of-the-art networks which use only expert features. While in principle all of the information exists in the lowest-level features and it should be possible to train a network which matches or exceeds this performance without expert knowledge, this is neither necessary nor desirable. Expert knowledge exists and is well-established, and there is no reason to discard it.

In addition, this expert guidance encourages the network to identify discrimination strategies based on well-understood properties of the jet flavor problem and decreases the likelihood of relying on learning strategies based on spurious or poorly-modeled corners of the space. We note that the use of high-dimensional lower-level data will require careful validation of the simulation models; reasonable strategies exist, such as a combination of the validation of individual features in one-dimensional projections with validation of the network output in control samples, which probes the use of information in multi-feature correlations.

These improvements in the performance of the tagger can give important boosts to physics studies which rely on the identification of jet flavor.

ACKNOWLEDGEMENTS

We thank David Kirkby, Gordon Watts, Shimon Whiteson, David Casper, and Kyle Cranmer for useful comments and helpful discussion. We thank Yuzo Kanomata for computing support. We also wish to acknowledge a hardware grant from NVIDIA and NSF grant IIS-1321053 to PB.

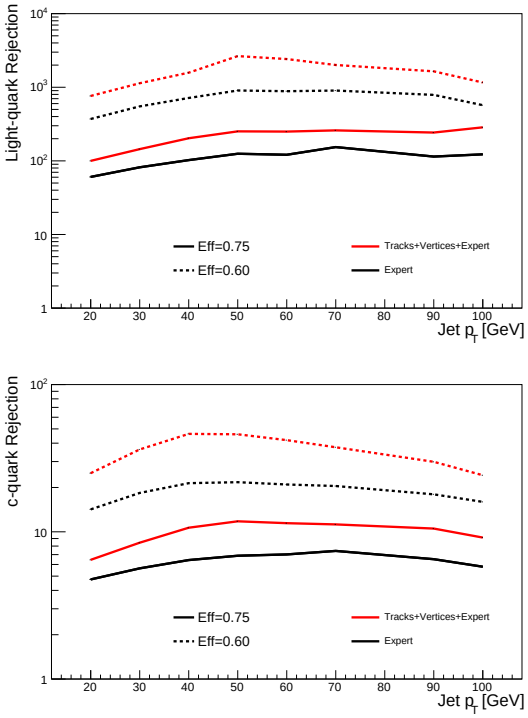

FIG. 9: Rejection of light quarks (top) or charm quarks (bottom) versus minimum jet p_{T}. In each case, rejection is shown for fixed values of signal efficiency for networks trained with only expert features or networks trained with all features (tracks, vertices and expert features).

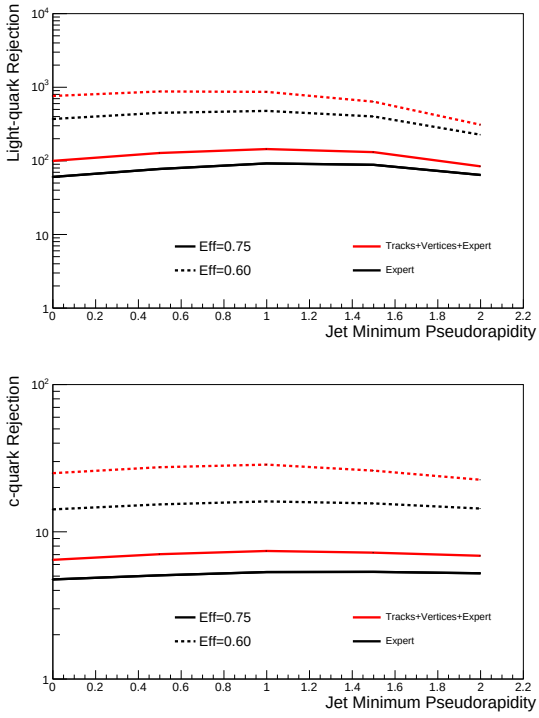

FIG. 10: Rejection of light quarks (top) or charm quarks (bottom) versus minimum jet pseudo-rapidity. In each case, rejection is shown for fixed values of signal efficiency for networks trained with only expert features or networks trained with all features (tracks, vertices and expert features).

[1] Georges Aad et al. A search for top squarks with R-parity-violating decays to all-hadronic final states with the ATLAS detector in $\sqrt{s} = 8$ TeV proton-proton collisions. 2016.

[2] Georges Aad et al. Search for single production of a vector-like quark via a heavy gluon in the $4b$ final state with the ATLAS detector in pp collisions at $\sqrt{s} = 8$ TeV. *Phys. Lett.*, B758:249–268, 2016.

[3] W. Waltenberger, W. Mitaroff, F. Moser, B. Pflugfelder, and H. V. Riedel. The RAVE/VERTIGO vertex reconstruction toolkit and framework. *J. Phys. Conf. Ser.*, 119:032037, 2008.

[4] Georges Aad et al. Performance of b-Jet Identification in the ATLAS Experiment. *JINST*, 11(04):P04008, 2016.

[5] Serguei Chatrchyan et al. Identification of b-quark jets with the CMS experiment. *JINST*, 8:P04013, 2013.

[6] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5, July 2014.

[7] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Enhanced higgs boson to $\tau^+\tau^-$ search with deep learning. *Phys. Rev. Lett.*, 114:111801, Mar 2015.

[8] P. Sadowski, J. Collado, D. Whiteson, and P. Baldi. *Journal of Machine Learning Research, Workshop and Conference Proceedings*, 42:81–97, 2015.

[9] Pierre Baldi, Kevin Bauer, Clara Eng, Peter Sadowski, and Daniel Whiteson. Jet Substructure Classification in High-Energy Physics with Deep Neural Networks. 2016.

[10] Pierre Baldi and Yves Chauvin. Hybrid modeling, hmm/nn architectures, and protein applications. *Neural Comput.*, 8(7):1541–1565, October 1996.

[11] C. Goller and A Kuchler. Learning task-dependent distributed representations by backpropagation through structure. *IEEE International Conference on Neural Networks*, page 347352, 1996.

[12] P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *Trans. Neur. Netw.*, 9(5):768–786, September 1998.

[13] F.A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstms. *"Neural Computation"*, 12:2451–2471, 2000.

[14] Pierre Baldi and Gianluca Pollastri. The principled design of large-scale recursive neural network architectures–dag-rnns and the protein structure prediction problem. *J. Mach. Learn. Res.*, 4:575–602, December 2003.

[15] R. Socher, A. Perelygin, J.Y. Wu, J. Chuang, C.D. Manning, A.Y. Ng, and C Potts. *Proceedings of the conference on empirical methods in natural language processing*, 1631:1642, 2013.

[16] P. Baldi, S. Brunak, P. Frasconi, G. Pollastri, and G. Soda. Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15:937946, 1999.

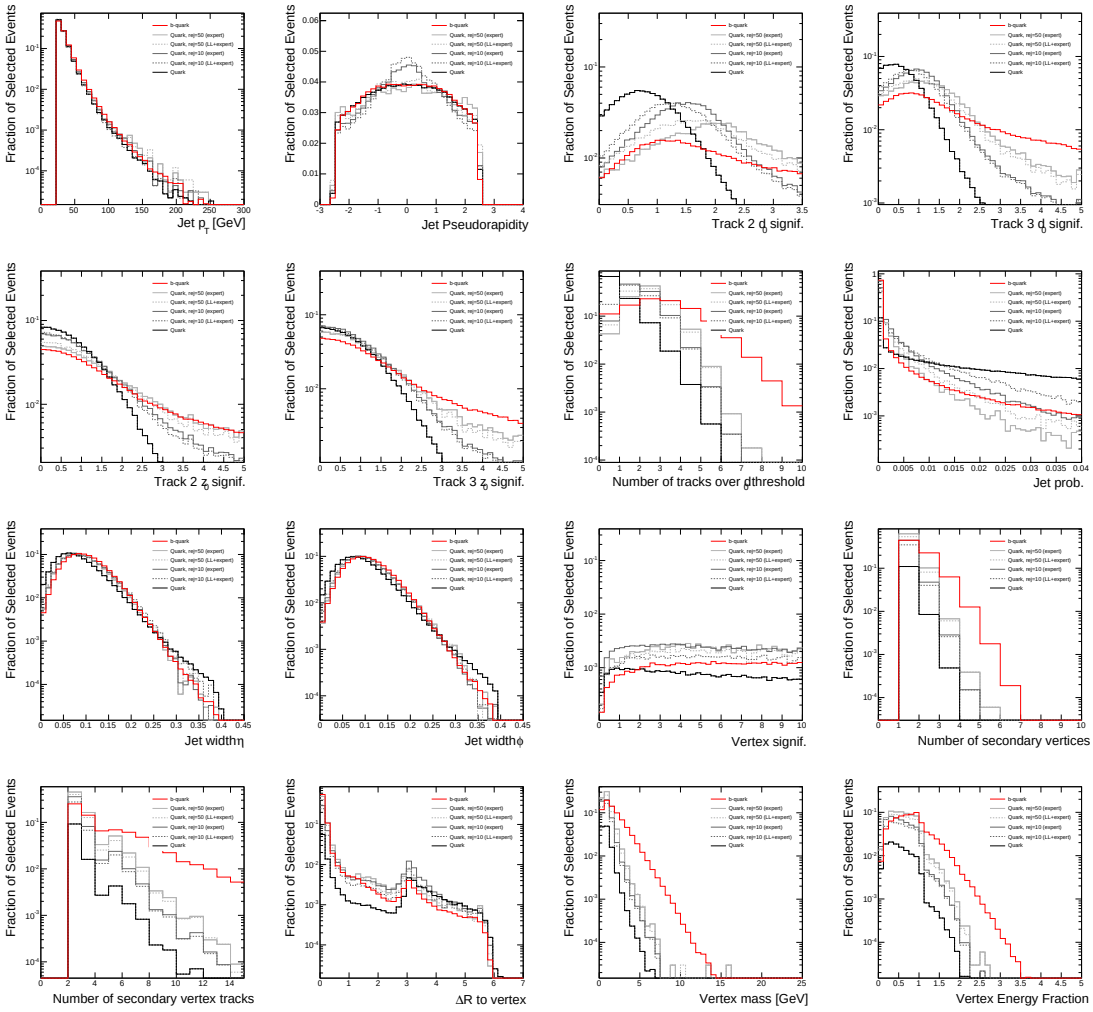

FIG. 11: Distributions of expert-level features for heavy-flavor and light-flavor classes. Also shown are distributions of light-flavor and charm jets surviving network threshold selections chosen to given rejection of 10 and 50, for networks using only expert information and networks using expert information in addition to lower-level information.

[17] A.N. Tegge, Z. Wang, J. Eickholt, and J. Cheng. Nncon: improved protein contact map prediction using 2d-recursive neural networks. *Nucleic acids research*, 37:515518, 2009.

[18] P. Di Lena, K. Nagata, and P. Baldi. Deep architectures for protein contact map prediction. *Bioinformatics*, 28:24492457, 2012.

[19] C.N. Magnan and P. Baldi. Sspro/accpro 5: Almost perfect prediction of protein secondary structure and relative solvent accessibility using profiles, machine learning, and structural similarity. *Bioinformatics*, 30:25922597, 2014.

[20] A. Lusci, G. Pollastri, and P. Baldi. Deep architectures and deep learning in chemoinformatics: the prediction of aqueous solubility for drug-like molecules. *Journal of chemical information and modeling*, 53:15631575, 2013.

[21] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2215–2223. Curran Associates, Inc., 2015.

[22] Lin Wu and Pierre Baldi. Learning to play go using recursive neural networks. *Neural Networks*, 21(9):1392 – 1400, 2008.

[23] L. de Oliveira. `https://indico.cern.ch/event/395374/`, 2015. DataScience@LHC.

[24] P. Baldi. `https://indico.cern.ch/event/395374/`, 2015. DataScience@LHC.

[25] Johan Alwall et al. MadGraph 5 : Going Beyond. *JHEP*, 1106:128, 2011.

[26] T. Sjostrand et al. PYTHIA 6.4 physics and manual. *JHEP*, 05:026, 2006.

[27] S. Ovyn, X. Rouby, and V. Lemaitre. DELPHES, a framework for fast simulation of a generic collider experiment. 2009.

[28] G. Aad et al. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003, 2008.

[29] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez.

The Anti-k(t) jet clustering algorithm. *JHEP*, 04:063, 2008.

[30] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. FastJet User Manual. *Eur. Phys. J.*, C72:1896, 2012.

[31] Wolfgang Waltenberger. RAVE: A detector-independent toolkit to reconstruct vertices. *IEEE Trans. Nucl. Sci.*, 58:434–444, 2011.

[32] Impact parameter-based b-tagging algorithms in the 7 TeV collision data with the ATLAS detector: the Track-Counting and JetProb algorithms. Technical Report ATLAS-CONF-2010-041, CERN, Geneva, Jul 2010.

[33] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*, July 2012.

[34] Pierre Baldi and Peter Sadowski. The dropout learning algorithm. *Artificial Intelligence*, 210:78–122, May 2014.

[35] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume JMLR W&CP 15, Fort Lauderdale, FL, USA, 2011.

[36] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, September 2009.

[37] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010.

[38] Pierre Baldi. The inner and outer approaches to the design of recursive neural architectures. *submitted*, 2016.

[39] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[40] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*, 2015.

[41] Theano Development Team. Theano: A python framework for fast computation of mathematical expressions. *arXiv e-prints arXiv:1605.02688*, 2016.

[42] Franois Chollet. *Keras*. GitHub, 2015.