ANNUAL REVIEWS

Annual Review of Nuclear and Particle Science

Deep Learning and Its Application to LHC Physics

Dan Guest,[1] Kyle Cranmer,[2] and Daniel Whiteson[1]

[1]Department of Physics and Astronomy, University of California, Irvine, California 92697, USA

[2]Physics Department, New York University, New York, NY 10003, USA

Annu. Rev. Nucl. Part. Sci. 2018. 68:161–81

The *Annual Review of Nuclear and Particle Science* is online at nucl.annualreviews.org

https://doi.org/10.1146/annurev-nucl-101917-021019

Keywords

deep learning, LHC, machine learning, particle physics

Abstract

Machine learning has played an important role in the analysis of high-energy physics data for decades. The emergence of deep learning in 2012 allowed for machine learning tools which could adeptly handle higher-dimensional and more complex problems than previously feasible. This review is aimed at the reader who is familiar with high-energy physics but not machine learning. The connections between machine learning and high-energy physics data analysis are explored, followed by an introduction to the core concepts of neural networks, examples of the key results demonstrating the power of deep learning for analysis of LHC data, and discussion of future prospects and concerns.

Contents

1. INTRODUCTION

The physics program of the Large Hadron Collider (LHC) has the potential to address many of the most fundamental questions in modern physics: the nature of mass, the dimensionality of space, the unification of the fundamental forces, the particle nature of dark matter, and the fine-tuning of the Standard Model. The importance of these questions and the scale of the experimental program needed to probe them demand that we do our utmost to extract the relevant information from the collected data.

The data collected by high-energy physics (HEP) experiments are complex and high dimensional. Traditional data analysis techniques in HEP use a sequence of Boolean decisions followed by statistical analysis on the selected data. Typically, both the individual decisions and the subsequent statistical analysis are based on the distribution of a single observed quantity motivated by physics considerations, which is not easily extended to higher dimensions.

For several decades, particle physicists have sought to improve the power of their analyses by employing algorithms that utilize multiple variables simultaneously. Within HEP this approach is often referred to as multivariate analysis (MVA); however, outside of physics these techniques would be considered examples of machine learning. Physicists have used a wide variety of machine learning techniques, including artificial neural networks, kernel density estimation, support vector machines, genetic algorithms, random forests, and boosted decision trees. For several years, the status quo of machine learning in HEP was to use boosted decision trees implemented in the software package TMVA (1). These tools provided an important boost for many data analysis tasks, but their capabilities were understood to be limited; they often failed to match the performance of physicist-engineered solutions, especially when the dimensionality of the data grew large.

The emergence of deep learning began around 2012, when a convergence of techniques enabled training of very large neural networks that greatly outperformed the previous state of the art (2–5). These new tools could adeptly handle higher-dimensional and more complex problems than

previously feasible. In the intervening years there has been an explosion in deep learning research, moving beyond image classification into natural languages, self-driving cars, and many areas of science.

This review is aimed at the reader who is familiar with data analysis for HEP but less familiar with machine learning. The remainder of this section explains the importance of machine learning to HEP data analysis and describes the basics of neural networks in order to clearly define the concept of deep learning. Section 2 reviews many of the key applications of deep learning to open challenging problems in LHC data analysis, including several breakthroughs in tasks which were previously thought to be intractable. Section 3 discusses the direction of current work and potential concerns regarding the application of deep learning. The final section discusses several possible future directions and prospects.

1.1. Why Is Machine Learning Relevant for Physics?

The data collected by the LHC experiments are vast both in the number of collisions and in the complexity of each collision. The colliding beams at the LHC are grouped into bunches of protons, which cross with a frequency of ∼40 MHz. Each collision has the potential to produce a large number of particles, and the LHC detectors have $\mathcal{O}(10^8)$ sensors used to record these particles. These high data rates are necessary because collisions that produce interesting products are very rare.

As a result of the quantum-mechanical nature of the collisions and the interaction of their products with LHC detectors, the observations resulting from a particular interaction are fundamentally probabilistic. Therefore, the approach to data analysis and the conclusions drawn from the data must be framed in statistical terms, which include not only low-level tasks, such as the reconstruction and identification of particles and the measurement of their energies and momenta, but also high-level tasks, such as searches for new particles and measurements. In classical statistics, tasks such as classification, hypothesis testing, regression, and goodness-of-fit testing are based on a statistical model $p(\boldsymbol{x}|\theta)$ describing the probability of observing $\boldsymbol{x}$ given the parameters θ of a theory.

The high dimensionality and large volume of LHC data pose a problem because the statistical model $p(\boldsymbol{x}|\theta)$ over the high-dimensional space of their experimental data is not known explicitly in terms of an equation that can be evaluated. Instead, one typically has access to large samples of simulated data generated by stochastic simulation programs that model the physics of particle interactions on various scales. If the data were fairly low dimensional ($d < 5$), the problem of estimating the unknown statistical model from the simulated samples would not be difficult. Histograms or kernel-based density estimates provide reasonable estimates in low-dimensional spaces. These fairly naïve strategies, however, suffer from the curse of dimensionality. In a single dimension, N samples may be required to describe the source probability density function. In d dimensions, the number of samples required increases to the power of the data's dimensionality: $\mathcal{O}(N^d)$. The consequence is that any dimensionality greater than five or ten requires impractical or impossible computational resources, regardless of the speed of the sample generator.

Traditionally, HEP physicists have approached this problem by reducing the dimensionality of the data through a series of steps that operate both on individual collision events and on collections of events. For an individual event, reconstruction algorithms process the raw sensor data into low-level objects such as calorimeter clusters and tracks. From these low-level components, the algorithms attempt to estimate the energy, momentum, and identity of individual particles. Next, from these reconstructed objects, event-level summaries are constructed. Event selection algorithms then select subsets of the collision data for further analysis on the basis of the

information associated with individual events. Traditionally, the reconstruction and event selection operations are based on specific, engineered features in the data. For instance, the identification of an electron and a photon is based on specific features that summarize the shape of the shower in the electromagnetic calorimeter and discriminate from the energy deposits left by charged and neutral hadrons. The cumulative product of these steps reduces the dimensionality of the problem to a number small enough to allow the missing statistical model $p(\boldsymbol{x}|\theta)$ to be estimated using samples generated by simulation tools.

While the traditional approaches to reconstruction and event selection have worked fairly well, there is no guarantee that they are optimal. Given the complex nature of the data and the subtle signatures of potential new physics, it is reasonable to suspect that there may be a significant performance gap between traditional approaches and the optimal one. A central role of machine learning in LHC physics is to improve this data reduction, reducing the relevant information contained in the low-level, high-dimensional data into a higher-level and lower-dimensional space.

1.2. The Role of Simulators

Physicists often refer to the set of simulation tools such as PYTHIA (6), HERWIG (7), MADGRAPH (8), SHERPA (9), and GEANT (10) as Monte Carlo tools, since these simulators are probabilistic and rely heavily on Monte Carlo sampling techniques. The simulators capture the relevant physics on a hierarchy of scales starting with the microscopic interactions within a proton–proton collision and ending with the interaction of particles in the enormous LHC detectors.[1]

We can think of a simulated data set $\{\boldsymbol{x}_i\}_{i=1}^N$ as being N independent and identically distributed samples from some underlying distribution $p(\boldsymbol{x}|\theta)$, where θ corresponds to the settings of the simulator. Moreover, we know the settings of the simulator used, which means that the generated data automatically come with ground-truth labels. For instance, we can generate samples of interactions involving a Higgs boson for any desired mass value.

In this context, the goal of simulation is to approximate the probability $p(\boldsymbol{x}|\theta)$ by sampling from an enormous space of unobserved, or latent, processes: $p(\boldsymbol{x}|\theta) = \int p(\boldsymbol{x}, \boldsymbol{z}|\theta)\mathrm{d}\boldsymbol{z}$. A fixed value of $\boldsymbol{z}$ specifies everything about the simulated events, from the momentum of the initial particles created in the hard scattering to the detailed interactions in the detector. Physicists often refer to $\boldsymbol{z}$ as Monte Carlo truth. Most reconstruction algorithms can be regarded as estimates of some components of $\boldsymbol{z}$ (particle type, momentum, energy, etc.) given the observed data $\boldsymbol{x}$. Here simulation fulfills a second experimental need: In addition to an estimate of $p(\boldsymbol{x}|\theta)$, the simulation provides a data set $\{\boldsymbol{x}_i, \boldsymbol{z}_i\}_{i=1}^N$ that allows physicists to study reconstruction algorithms directly.

1.3. Core Concepts in Machine Learning

Fortunately, many of the tasks encountered in HEP can be naturally reformulated as machine learning problems. Typically, the problems are formulated in terms of a search for some function $f : \boldsymbol{X} \to \boldsymbol{Y}$, from the space of the observed data $\boldsymbol{X}$ to a low-dimensional space of a desired target label $\boldsymbol{Y}$, which optimizes some metric of our choosing. This metric is called a loss function and is written as $L[\boldsymbol{y}, f(\boldsymbol{x})]$.

Ideally, a learning algorithm would find the function that optimizes L over all possible values of $(\boldsymbol{x}, \boldsymbol{y})$, but this is intractable owing to the curse of dimensionality and an infinite number of

[1] In the language of statistics and machine learning, the full simulation chain would be considered a generative model for the data, as they can be used to generate synthetic data of the same complexity and format as the actual collision data.

functions to choose from. Instead, in supervised learning, one has labeled training data $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$ sampled from $p(\boldsymbol{x}, \boldsymbol{y})$. [2] Furthermore, the function space is restricted to a model—a highly flexible family of functions $f_\phi(\boldsymbol{x})$ parameterized by ϕ. In this case, the algorithms minimize the loss function directly with respect to the model parameters ϕ. Neural networks, support vector machines, and decision trees are examples of types of models commonly used in machine learning. These models often have a large number of parameters, and in the case of neural networks, finding the optimal f_ϕ can be a difficult problem.

An essential goal in machine learning is generalization—the ability of the model to perform well on data that were not used in training. Failure in this task is called overtraining. A vast array of techniques exist to avoid overtraining (or overfitting), all of which can be considered forms of regularization. Regularization techniques such as dropout were key to advances in image recognition with deep learning (2, 3). Theoretical analysis of the generalization of deep learning is difficult because it involves a complicated interaction between the specifics of the model f_ϕ, the optimization algorithms, the loss function, regularization techniques, and the specifics of the finite training samples and the true underlying distribution $p(\boldsymbol{x}, \boldsymbol{y})$. Empirically, deep learning models generalize much better than existing theoretical analysis might suggest. While a more powerful theoretical analysis of generalization for deep learning would be valuable, in practical terms it is not necessary as long as statistically independent data, not used in training, are available to validate the performance.

1.4. Neural Network Basics

In the language of neural networks, the space of functions searched is defined by the structure of the networks, which defines a series of transformations. These transformations map the input $\boldsymbol{x}$ onto internal or "hidden" states $\boldsymbol{h}_i$, until the final transformation maps these hidden states onto the function output $\boldsymbol{y}$.

Mathematically, these transformation are expressed as

$$\boldsymbol{h}_{i+1} = g_i(W_i \boldsymbol{h}_i + \boldsymbol{b}_i), \qquad 1.$$

where g_i is some function, called the activation function, and a particular $\boldsymbol{h}_i$ is the ith transformation of the information in $\boldsymbol{x}$, called the embedding. In a simple case, the first embedding is simply the input vector $\boldsymbol{h}_0 \equiv \boldsymbol{x}$, and the final embedding is the output of the network. The elements of the matrix W are referred to as weights and those of vector $\boldsymbol{b}$ as biases. The general structure of these transformations, such as the dimensionality of each W and the choice of activation function, is referred to as the network architecture, which, taken together with the training parameters, constitutes the hyperparameters of the network.

The weights and biases of the network are initialized randomly. The function that optimizes the loss function is found through an iterative process called training. Conceptually, this process uses the labeled training examples $(\boldsymbol{x}, \boldsymbol{y})$ and calculates the gradient of the loss function with respect to the model parameters, $\nabla_\phi L[f_\phi(\boldsymbol{x}), \boldsymbol{y}]$. In practice, the calculations are done through a technique called backpropagation, which is an efficient means of computing this gradient. In principle, backpropagation places only one restriction on L and $f(\boldsymbol{x})$: They must be differentiable for a gradient to be defined.

[2] Other machine learning paradigms, such as unsupervised, semisupervised, and weakly supervised learning, relax or remove the need for labels in the training data.

1.5. Deep Learning

Initially, the term deep neural networks referred to neural networks with many hidden layers, and it was used to differentiate such networks from shallow neural networks, which had only one hidden layer. For many years, it was argued that using a shallow network was not a restriction, because of the theoretical analysis that demonstrated that any function can be approximated by a shallow network (14). However, an effective shallow network may require an enormous number of nodes in the hidden layer, and in practice, shallow neural networks often failed to discover useful functions from high-dimensional data sets.

The traditional strategy for discovering the optimal function for a given application involves a gradient search through f_ϕ. In practice, this becomes much more difficult to accomplish as the neural network becomes deeper. As the difference between the function value $f_\phi(\boldsymbol{x})$ and the desired output $\boldsymbol{y}$ is propagated back through the various embeddings, the gradient $\nabla_\phi L[f_\phi(\boldsymbol{x}), \boldsymbol{y}]$ rapidly approaches zero, making it difficult to improve the performance by adjusting the model parameters. This vanishing gradient problem (15, 16) has been overcome in recent years using a variety of strategies, including computational boosts from graphical processing units (GPUs), larger training samples, new regularization techniques such as dropout (17), and pretraining of initial embeddings with unsupervised learning methods such as autoencoders (18, 19). Autoencoders attempt to learn a useful layered representation of the data without having to backpropagate through a deep network; standard gradient descent is used only at the end to fine-tune the network.

More generally, deep learning can refer to a broad class of machine learning methods emphasizing hierarchical representations of the data and modular, differentiable components. Not only do these deep networks have more expressive capacity, but also the layers can be interpreted as building up a hierarchical representation of the data. In natural images, for example, the first layers learn low-level features like edges and corners, the middle layers learn midlevel features like eyes, and the final layers learn high-level features like faces. The processes that produce particle physics data naturally lead to compositionality and hierarchical structured data. For instance, a typical event at the LHC is composed of jets, jets are composed of hadrons, hadrons lead to tracks and calorimeter clusters, tracks are composed of hits, and calorimeter clusters are composed of calorimeter cells. The analogy also extends to higher levels with groups of particles forming resonances in a cascade decay. For these reasons, one might anticipate deep learning to be particularly effective at the LHC.

Modern deep learning is characterized by the composition of modular, differentiable components (20). Among the first of these modular components was the convolutional filter, which is arguably the most important innovation in deep learning applied to image processing (21). Convolutional architectures are natural when the input data have some notion of locality, the individual components of $\boldsymbol{x}$ are of the same type (e.g., neighboring pixels in an image), and the interesting features are equally likely to appear in any local patch. The kernel $\boldsymbol{k}$ of the convolution can be interpreted as a bank of filters that operate on a local patch of the input $\boldsymbol{x}_i$ as

$$\boldsymbol{h} = g(W\boldsymbol{x} + \boldsymbol{b}) \quad \rightarrow \quad h_{i,j} = g(\boldsymbol{k}_j \cdot \boldsymbol{x}_i + b_j), \qquad 2.$$

where i indexes the local patches and j indexes the filters. Because the same kernel is applied as it is swept over the input, it has the effect of sharing weights in a dense network. Weight sharing imposes translational symmetry on the network, and it drastically reduces the number of parameters in the network and the amount of data needed to train them. Convolutional layers are usually followed by a pooling layer, which summarizes the result of applying the filters in a local patch (e.g., by taking the maximum or average). These convolutional and pooling layers can be composed to build a hierarchical representation of the data going from low- to mid- to high-level

features. Other modular components include normalization layers (22) and residual layers (23). By training the different layers of these networks jointly, deep convolutional neural networks learn hierarchical features that tend to outperform engineered features for image processing tasks.

Working with variable-length input (e.g., words in a sentence) requires the network architecture to be adaptive in some way. Variable-length input can be cropped or padded with zeros to fit into a fixed-size vector $\boldsymbol{x}$, but these blunt solutions either discard potentially useful information or force a network to accommodate placeholder values. A far more natural solution is to rely on networks that can adjust to the input size dynamically. A particularly illustrative case is a simple recursive unit, which maps a pair of inputs, $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$, onto an output, $\boldsymbol{h}$, as follows:

$$\boldsymbol{h} = g_i(W_1\boldsymbol{h}_1 + W_2\boldsymbol{h}_2 + \boldsymbol{b}). \qquad 3.$$

Assuming that one or both of the input vectors are of the same dimension as $\boldsymbol{h}$, the output can be fed into the input recursively and condense an arbitrary length sequence of inputs into a fixed-dimensional representation: $\{\boldsymbol{h}_i\} \rightarrow \boldsymbol{h}$. More generally, a neural network can be visualized as a directed acyclic graph (24, p. 118) in which edges represent the various internal $\boldsymbol{h}$ vectors. **Figure 1** illustrates several such graphs.

In practice, since recursive networks can grow very deep, simple recursive units encounter problems with vanishing or exploding gradients. These longer sequences can be handled using a technique known as gating, in which activation functions and transformations are applied selectively, or inputs can be ignored entirely. These alleviate the exploding and vanishing gradient

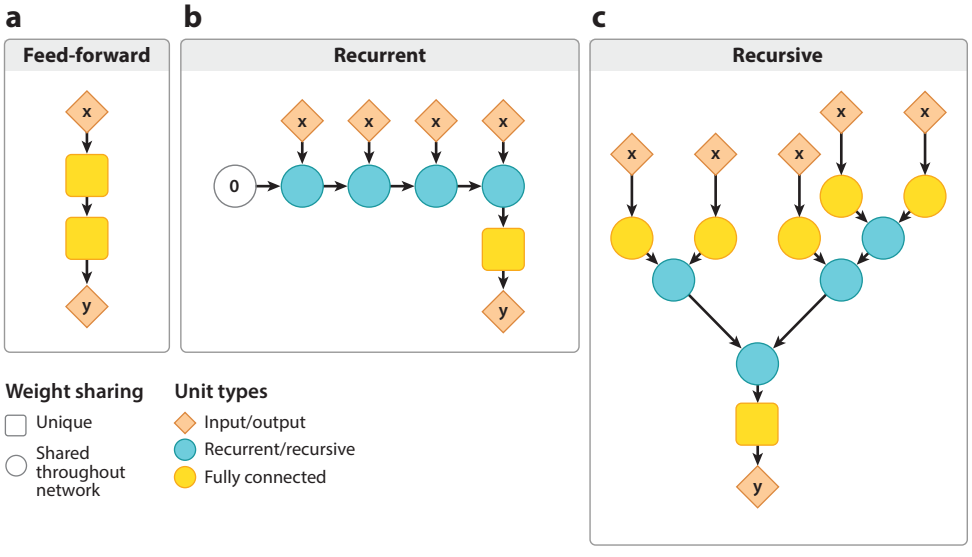

Figure 1

Schematic showing (*a*) feed-forward, (*b*) recurrent, and (*c*) recursive neural network architectures. Inputs and outputs are represented by diamonds, and processing units are represented by circles and squares. Arrows between processing units represent embeddings $\boldsymbol{h}$. Standard feed-forward networks map a fixed length $\boldsymbol{x}$ into $\boldsymbol{y}$, whereas recurrent and recursive networks can process a sequence of inputs $\{\boldsymbol{x}_i\}$. Units represented as circles are shared throughout the network: Once the network is trained, the units can be used to build a network of arbitrary size. Recurrent networks can be viewed as a subset of recursive networks, in which each node combines one input $\boldsymbol{x}_i$ and the output from the previous recurrent node $\boldsymbol{h}_{i-1}$ to produce $\boldsymbol{h}_i$, and where $\boldsymbol{h}_0 = 0$. Recursive units map each pair of inputs to an output in the same space: $(\boldsymbol{h}_i, \boldsymbol{h}_j) \rightarrow \boldsymbol{h}_k$. Note that these components can also be chained: Any output node can also serve as an input node to another component.

problem at the expense of a more complicated recurrent unit; examples are long-short-term-memory units (LSTMs) (25) and gated recurrent units (26).

Both convolutional and recurrent layers are examples of network architectures that use shared weights. In the convolutional case, each element of $\boldsymbol{k}$ acts in multiple dot products, whereas in the recurrent case, the transformation in Equation 3 is applied multiple times for each pattern. Weight sharing can be viewed as a type of regularization: By reusing the same transformation in multiple places throughout the network, the network designer can encode domain-specific structure.

2. SURVEY OF APPLICATIONS

Machine learning has found numerous natural applications in particle physics, in which many tasks require classification in high-dimensional variable spaces. At the lowest level, machine learning tools can perform hit reconstruction (27) or track finding (28) in individual detector systems. These tools can also perform object identification by using information from various detector systems, such as electron (29), photon (30), or τ lepton (31) identification. Finally, machine learning tools have been widely used to classify entire events as background-like or signal-like, both in the final statistical analysis (32) and at the initial trigger decision (33). These machine learning tools have found high-profile applications in single t quark searches (34), early Higgs boson searches (35), and the Higgs boson discovery (30).

2.1. Event Selection and High-Level Physics Tasks

The earliest successes of deep learning in HEP came in improvements in event selection for signal events with complex topologies. In the past few years, several studies have demonstrated that the traditional shallow networks based on physics-inspired engineered (i.e., high-level) features are outperformed by deep networks based on higher-dimensional features that receive less preprocessing (i.e., low-level features). Prior to the advent of deep learning, such preprocessing was necessary, as shallow network performance on low-level features fell short. The deep learning results discussed below demonstrate that deep networks using low-level features surpass the shallow networks using high-level features. This finding confirms the suspicion that feature engineering, applying physics knowledge to construct high-level features, is often suboptimal.

An early study (36) compared the performance of shallow and deep networks in distinguishing a cascading decay of new exotic Higgs bosons from the dominant background. This study used a structured data set in which a large set of basic low-level features (object four-momenta) was reduced to a smaller set of physics-inspired high-level engineered features. Because the high-level features were a strict function of the low-level features, they contained a subset of the information, so that the expertise encoded by the high-level features was solely in the design of these dimensionality-reducing functions rather than the introduction of new information. This gave rise to revealing comparisons about the relative information content of the low- and high-level features and the power of classifiers to extract it. In their study, Baldi et al. (36) found that deep networks using the low-level data significantly outperformed shallow networks that relied on physics-inspired features such as reconstructed invariant masses (**Figure 2**). The high-level engineered features captured real insights, but clearly sacrificed some useful information.

Such conclusions are not universal, however, but rather are dependent on the specifics of the classification task. Using the same approach, Baldi et al. (36) analyzed a supersymmetric particle search that has received significant feature engineering in the literature, and found that shallow networks using low-level data very nearly matched the performance of both shallow networks using engineered features and deep networks on either set of features. The authors concluded that

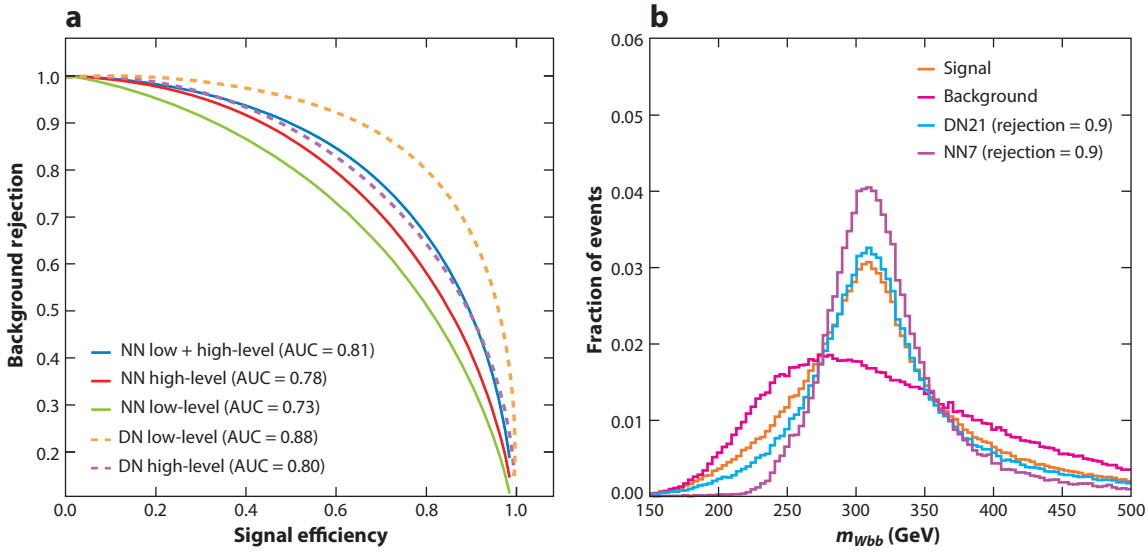

Figure 2

(*a*) A comparison between the performance of deep networks (DNs) in signal–background classification and that of shallow networks (NNs) with a variety of low- and high-level features demonstrates that DNs with only low-level features outperform all other approaches. (*b*) A comparison between the distributions of invariant mass of events selected by a DN (DN21) using only object momentum and those of an NN (NN7) that has been trained using this feature, at equivalent background rejection. Also shown in panel *b* are the distributions in pure signal and background samples. The NN relies heavily on this invariant mass quantity to discriminate. The DN has also discovered the value of this feature but is able to recover signal further from peak. Abbreviation: AUC, area under the curve. Panel *a* adapted from Reference 36.

this application requires only simple linear functions of the lower-level data, and may not require a deep network or deserve such attention to feature engineering. Similar conclusions were reached in Reference 37.

In addition to optimizing event selection for a fixed signal versus background problem, Cranmer and colleagues (38) showed that it is possible to approximate the likelihood ratio $p(\boldsymbol{x}|\theta)/p_0(\boldsymbol{x})$ for a continuous family of signal models parameterized by θ. Since binary classification amounts to approximating a likelihood ratio, this generalization is called a parameterized classifier. This is a common-use case at the LHC because most signals are predicted by theories with several free parameters. For instance, Baldi et al. (39) used this technique to create a classifier for $X \to t\bar{t}$ versus Standard Model background parameterized by the mass of the resonance m_X. The approximate likelihood can also be used in a likelihood fit to estimate the parameters θ (e.g., masses, coupling constants), providing a novel form of likelihood-free inference. The CARL software package provides a convenient interface for this technique (38).

2.2. Jet Classification

Machine learning has been applied to a wide range of jet classification problems to identify jets from heavy (c, b, t) or light (u, d, s) quarks; gluons; and W, Z, and H bosons. Traditionally these classification problems have been grouped into flavor tagging, which discriminates between b, c, and light quarks; jet substructure tagging, which discriminates between jets from W, Z, t, and H; and quark–gluon tagging.

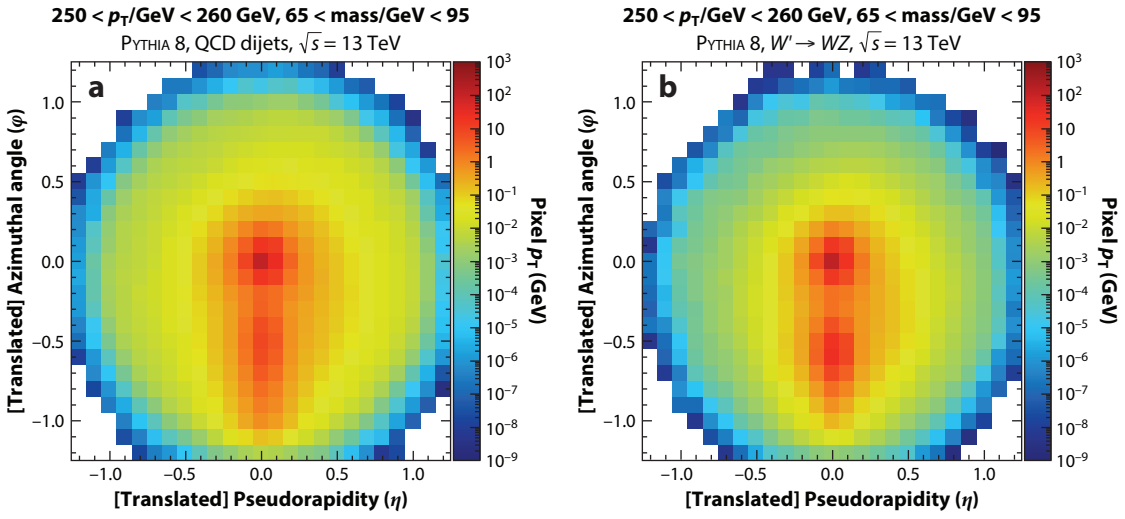

Figure 3

Example jet image inputs from the jet substructure classification problem described in Reference 46. (*a*) The background jets are characterized by a large central core of deposited energy from a single hard hadronic parton, while (*b*) the signal jets tend to have a subtle secondary deposition due to the two-prong hadronic decay of a high-p_T vector boson. Use of image-analysis techniques such as convolutional neural networks allows for powerful analysis of this high-dimensional input data.

In flavor tagging, the discriminating information is spatial: Heavy quarks decay weakly in a matter of picoseconds, which is sufficient time for a highly boosted quark to travel roughly a centimeter from the interaction point. Due to this measurable separation, flavor tagging relies heavily on tracks reconstructed by high-granularity sensors near the interaction point, and on vertices fit to these tracks. The use of machine learning in flavor tagging dates to LEP (40), where libraries such as JETNET (41) were used to identify *b* and *c* quarks, and has continued through LHC Runs 1 and 2 (42–44).

In contrast to jet flavor tagging, jet substructure and quark–gluon tagging rely on information created at one spatial location during the decay of the original particle. The spread in decay product momenta translates to spatial separation as the particles travel away from the interaction point, but the underlying physics is localized at the decay point. Thus, while the power of flavor tagging is limited primarily by the tracking detector resolution, jet substructure and quark–gluon discrimination are subject to quantum-mechanical limitations. Theoretical and experimental physicists have expended considerable effort in quantifying these limitations, and in engineering discriminating variables based on jet substructure (45).

Recently, the realization that lower-level, higher-dimensional data could contain additional power led to a rapid proliferation of studies that challenged established substructure approaches (45; see **https://indico.physics.lbl.gov/indico/event/546/overview**). In 2014, Cogan et al. (47) recognized that the projective tower structure of calorimeters present in nearly all modern HEP detectors was similar to the pixels of an image (**Figure 3**). This representation of the data allowed physicists to leverage the advances in image classification such as convolutional neural networks. The image-based networks discriminated as well as or better than shallow networks using jet substructure–based inputs (46, 48). The discrimination persisted in the presence of pileup and jet grooming (49), across generators (50), and could be generalized to three-dimensional detectors using multiple stacked channels analogous to colors (51). Outside collider physics, a similar approach was used to tackle object-identification tasks in neutrino experiments (52, 53).

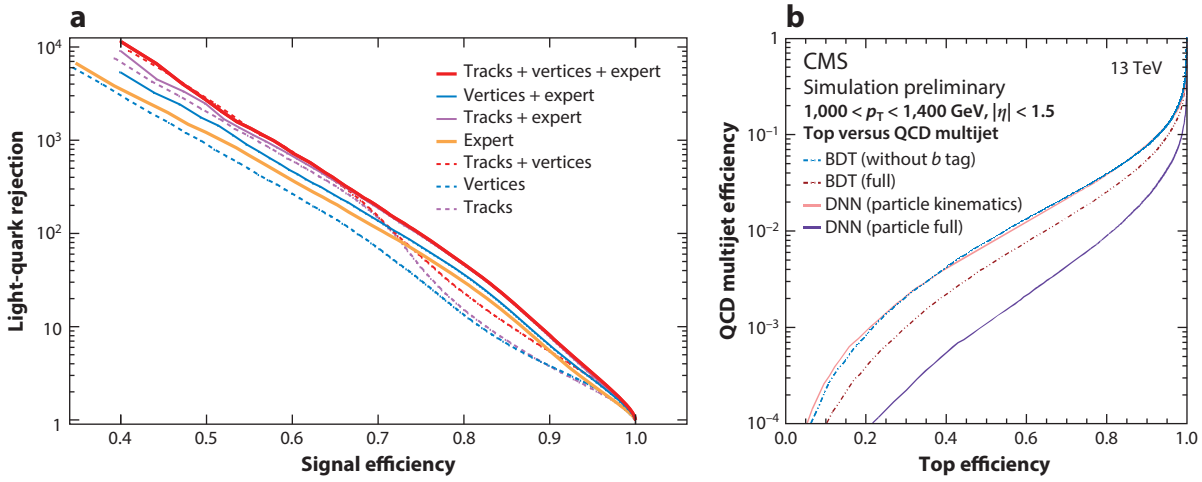

Figure 4

(*a*) A comparison (54) of the jet flavor–tagging performance of deep networks with varying levels of feature engineering. The lowest-level data (labeled Tracks) contain all of the information and perform well, but networks that are also given vertices or higher-level features (labeled Vertices and Expert) still show some improvement. (*b*) A comparison (59) of the performance of two boosted decision tree (BDT) taggers and two particle-based deep neural network (DNN) taggers in simulated events of four top jets as signal and QCD multijets as background.

While the image-based approach has been successful, the actual detector geometry is not perfectly regular; thus, some preprocessing is required to represent the jet as an image. In addition, jet images are typically very sparse. The sparsity can be alleviated by enlarging pixels, but the harsher discretization sacrifices resolution in η and ϕ. Given that the jets themselves are composed of a varying number of reconstructed constituents, each with well-defined coordinates and parameters, jet-tagging algorithms that can work with a variable number of inputs are desirable.

Several flavor-tagging applications have made use of deep networks trained on variable-length arrays of track parameters. Guest et al. (54) investigated the need for feature engineering by defining low-, mid-, and high-level features, where the mid- and high-level features were inspired by typical flavor-tagging variables and derived from a strict subset of the low-level feature information. The authors found similar discrimination using fixed-size, zero-padded networks and recurrent architectures, and determined that the best performance came from using all three levels of features (**Figure 4**). Both ATLAS and CMS have since commissioned flavor-tagging neural networks that rely on individual tracks or, in the CMS case, particle-flow candidates (55). The ATLAS recurrent network–based approach reduces backgrounds by roughly a factor of two when combined with traditional high-level variables (44, 56). CMS's DeepFlavor (57, 58) neural network first embeds each flow candidate with a transformation that is shared across candidates, then combines the candidates' high-level variables in a single zero-padded dense network.

Networks trained on variable-length arrays of jet constituents proved equally useful in boosted top tagging. In one series of studies, a zero-padded dense network showed promise (60), but backgrounds were halved by replacing the dense network with a recurrent network (61). The CMS Collaboration experimented with two variants of the DeepJet (62) algorithm. The first was similar to DeepFlavor, whereas the second replaced the dense network with a recurrent neural network. In comparison to a baseline that combined high-level variables in a boosted decision tree, QCD multijet misidentification was reduced by a factor of approximately four at 60% top-tagging efficiency (**Figure 4**) (59).

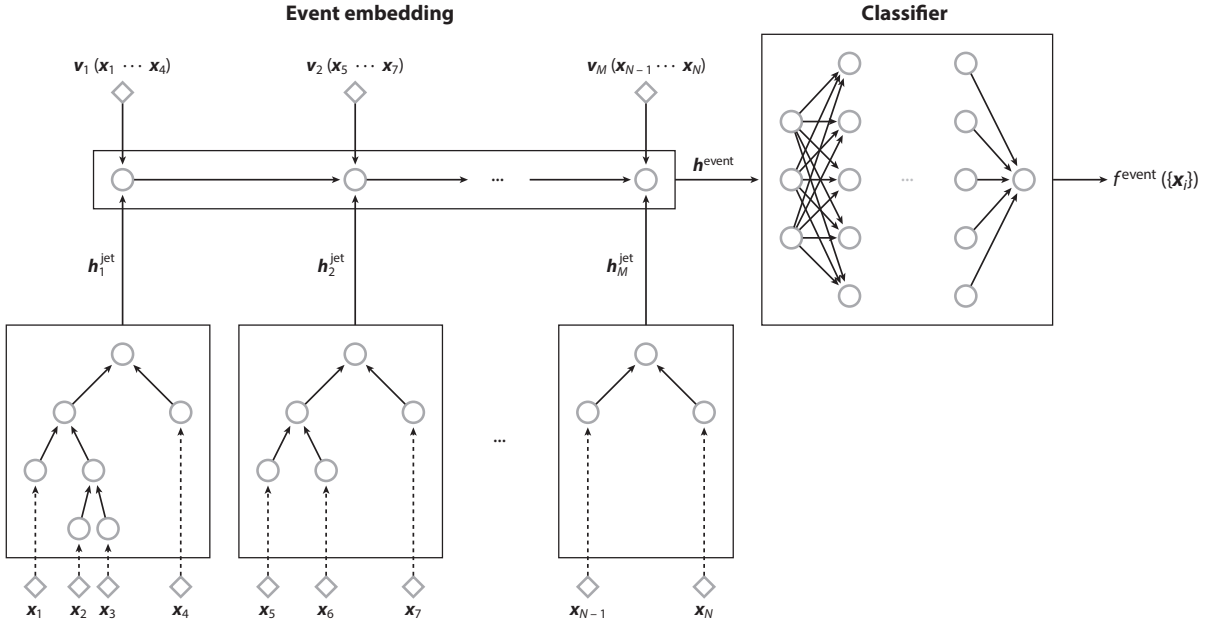

Figure 5

The hierarchical composition of a deep learning model following the outline of the traditional high-energy physics data pipeline. The lowest-level detector inputs are represented by $\boldsymbol{x}_i$ (*diamonds*), which are then fed into recurrent networks (*lower boxes*) to form jet embeddings $\boldsymbol{h}_1^{\text{jet}}$. These are augmented by jet-level features $\boldsymbol{v}_i$. The jet embeddings are processed by a network to form a final event-level embedding $\boldsymbol{h}^{\text{event}}$, which is then fed into a classifier, leading to the output $f^{\text{event}}(\{\boldsymbol{x}_i\})$. The entire network can be learned jointly, or individual components can be pretrained. Adapted from Reference 63.

Recurrent networks act on sequences, requiring an ordering of the particles. While several natural orderings exist, the most natural is arguably the k_{T} jet clustering history, which defines a tree that can be used as the scaffolding for a recursive neural network. Louppe et al. (63) applied a recursive neural network over the jet clustering history, providing a hybrid QCD-aware neural network strategy. In the same W-versus-QCD jet classification problem as studied in Reference 50, the recursive network showed no improvement over image-based networks when trained on jet images, but improved substantially when the image preprocessing was removed. Notably, the recursive and recurrent networks had fewer parameters and thus required far fewer data to train. The same recursive neural network has been applied to quark–gluon tagging (64).

The clustering of objects need not end at the jet level (**Figure 5**). The outputs from the jet-level recurrent network can be fed into a recurrent network to produce a high-level event embedding.

More recently, Henrion et al. (65) investigated representing jets as a graph instead of as a tree. Graph convolutional networks provide a generalization of convolutional neural networks that can be applied to irregularly sampled data (66). In this picture, the particles represent the nodes of the graph and the edges can encode how close the particles are in a learned adjacency matrix. Henrion et al. showed that such a network outperformed a recursive network for the same W-versus-QCD jet-tagging problem studied in References 50 and 63.

2.3. Tracking

Track reconstruction algorithms are among the most intensive of all low-level reconstruction tasks in terms of central processing unit (CPU) and data requirements. The initial stage of track

reconstruction involves finding hits, or points where some charge is deposited on a sensing element. In the case of the pixel sensors that form the innermost layer of the detector, neighboring hits are clustered into pixel clusters, which then form track seeds. These seeds form a starting point for a Kalman filter, which extends the seeds into full tracks that extend to the calorimeters. The entire procedure can be viewed as a sequence of clustering algorithms, in which the zero-suppressed readout from $\mathcal{O}(10^8)$ channels provides $\mathcal{O}(10^4)$ hits, which are then clustered into $\mathcal{O}(10^3)$ tracks per event.

Machine learning has proven useful in several aspects of track reconstruction. In cases where multiple tracks pass through the same pixel cluster, ATLAS relies on neural networks to return a measurement for each track rather than assigning each to the cluster center (67, 68). LHCb makes use of several shallow networks in track reconstruction as well. Due to the long distances between sensor elements in the LHCb tracker, falsely connected hits forming so-called ghost tracks are a significant source of background. A simple three-layer network reduces this background by a factor of two in comparison to a χ^2-based discriminant (69). Several other networks are evaluated to filter out fake tracks before running a full track fit or to recover tracks with missing hits.

Thanks to these algorithms and careful tuning, track reconstruction is nearly 100% efficient and spuriously reconstructed tracks are rare, meaning that the clustering aspect of tracking is largely solved. Reducing the CPU overhead remains a significant problem, however, especially within high-level trigger farms. Within ATLAS and CMS, these are clusters of $\mathcal{O}(10^4)$ processors that must reconstruct $\mathcal{O}(10^5)$ events per second (70). To keep CPU costs for tracking manageable, the experiments reconstruct tracks only in limited regions of the detector. These regions are selected on the basis of their proximity to muons or to calorimeter energy deposits that are consistent with relatively rare physical signatures such as leptons or high-p_{T} jets. While effective, this selective tracking severely hampers searches and physical measurements that rely on low-p_{T} track-based signatures.

These issues will be compounded considerably at the High-Luminosity LHC (HL-LHC). The majority of the tracking CPU budget is currently allocated to the track-building phase, which depends on an expensive Kalman filter (71, 72). With the higher track densities expected at the HL-LHC, the number of false seeds is expected to increase combinatorially (73), as is the probability of the Kalman filter building a branching track. More discriminating seeding algorithms or trainable deep Kalman filters (74, 75) could help reduce the number of track-fitting iterations. Unfortunately, most tracking software is deeply interwoven with the experiments' reconstruction frameworks and, as a result, is poorly suited for the quick exploratory studies that will be needed to develop the next generation of algorithms.

In anticipation of the coming HL-LHC data onslaught, however, improved tracking is essential. Several projects aim to accomplish this goal indirectly by increasing the visibility and accessibility of tracking software. Most ambitiously, the ACTS project (see **http://acts.web.cern.ch/ACTS**, **https://github.com/trackml**) seeks to implement a modular and experiment-independent software stack for tracking-related studies, including detector geometry, an event data model, and seed-finding and track-fitting tools. Simplified tracking models (76) have been used as a basis for studies showing the viability of LSTMs for track building (77) or for tracking data challenges (78).

2.4. Fast Simulation

The ability to model high-dimensional distributions not only enables improved statistical analysis but also provides a new path toward fast simulation. Fast simulation is valuable because the full simulators, which faithfully describe the low-level interactions of particles with matter, are very

computationally intensive and consume a significant fraction of the computing budgets of current experimental collaborations.

Until now, the dominant approach to fast simulation has been to develop fast parametric distributions largely by hand (79). A more recent deep learning approach is to train a network to learn the simulation from an initial pool of traditionally simulated events. This approach creates a generative model G that approximates the distribution of samples produced by the simulator by mapping an input space of random numbers to the space of the data.

One promising approach is based on generative adversarial networks (GANs). The training of a generative model G is accomplished through competition with an adversarial network A. While G generates simulated samples, A tries to determine whether a given sample is from the generative model or from the full simulator. The two networks are pitted against each other: A attempts to identify differences between the traditional samples and those generated by G, while G attempts to fool A into accepting its events, and in doing so learns to mimic the original sample generation. The stability of such a training arrangement, however, can be difficult to achieve, and expert knowledge is required to construct an effective network.

Paganini et al. (86) applied the GAN approach to simulation of the electromagnetic showers in a multilayer calorimeter, one of the most computationally demanding steps for the low-level simulator. They reported large computational speedups and achieved reasonable modeling of the energy deposition, though it does not yet match the accuracy of the full simulation. In a related approach, de Oliveira et al. (87) found similar success in simulating jet images. Future simulation tools built on GANs may provide important speed boosts for the slower elements of the simulation chain, or they may be sophisticated enough to provide end-to-end simulations.

The resulting network evaluation is much less computationally demanding than the low-level simulation, and can be viewed as a nonparametric fast simulation. The promise of this approach to mitigate the computational burden for simulation has been invoked in the strategic planning for HL-LHC software efforts (88, 89). We refer the reader to References 80–85 for alternative approaches to fast simulation.

2.5. Impact

Taken together, the new tools made possible by deep learning promise to make a significant impact on HEP. The specific examples described above—mass reconstruction, jet substructure and jet flavor classification—are important benchmarks and long-standing challenges. The significant improvements offered by deep learning in these areas support the claim that many areas of LHC data analysis suffer from long-standing suboptimal feature engineering and deserve re-examination.

3. CONCERNS

3.1. What Is the Optimization Objective?

A challenge of incorporating machine learning techniques into HEP data analysis is that tools are often optimized for performance on a particular task that is several steps removed from the ultimate physical goal of searching for a new particle or testing a new physical theory. Moreover, some tools are used in multiple applications, which may have conflicting demands. For instance, a jet flavor–tagging algorithm based on deep learning might be used for searches for supersymmetry as well as precision measurements of the Higgs sector, which may have different needs with respect to balancing signal efficiency and background rejection.

These considerations are further complicated by the fact that the sensitivity to high-level physics questions must account for systematic uncertainties, which involve a nonlinear trade-off between

the typical machine learning performance metrics and the systematic uncertainty estimates. For example, a new classifier may have a better false-positive rate than a baseline algorithm, yet simultaneously be more susceptible to systematic mismodeling between the simulation and the real data. Whether or not this new classifier will improve the sensitivity for the ultimate high-level physics goal depends on details such as the signal-to-background ratio, the total number of data, and the size of the systematic uncertainty, which are not typically included in the classifier training.

Traditionally, HEP physicists have taken these considerations into account through heuristics and intuition. But as deep learning penetrates into the analysis pipeline, it is important to revisit these trade-offs and attempt to make them explicit to design new loss functions and learning algorithms that directly optimize for our ultimate physics goals.

For example, in order to be robust to systematic uncertainties, one can use a classifier parameterized in terms of the nuisance parameters (39, 83), allowing for major speedups compared with earlier strategies (90). An alternative approach is to train a network to be insensitive to the systematic uncertainty, which is achieved either by boosting (91) or by using an adversarial training procedure that encourages the output of the network to be independent of the nuisance parameters (92, 93). The technique can also be used to enforce independence from another variable, such as the jet mass (94). In general, there is a trade-off between performance in the classification (or regression) task and robustness to systematics, which can be adjusted via a hyperparameter λ in the loss function. Unfortunately, optimization of λ requires retraining the network, and the objective function may not be differentiable with respect to λ.

Optimization of differentiable components is efficiently handled with various forms of stochastic gradient descent, although these algorithms often come with their own hyperparameters. The optimization with respect to hyperparameters that arise in the network architecture, loss function, and learning algorithms are often performed through a black-box optimization algorithm that does not require gradients. Examples include Bayesian optimization (95, 96) and genetic algorithms (90), as well as variational optimization (97, 98).

3.2. Interpretability and Reliance on Simulation

Machine learning provides an effective and powerful solution to many data analysis challenges in HEP, in some cases lessening the need for engineered features driven by physical insight. However, in some sense this approach is not satisfactory, as progress on the computational side is not always matched by gains in physical understanding and is heavily reliant on simulation programs.

The nonparametric nature of the neural network approach makes it very difficult to interpret the solution. Unlike a simple analytic function written in mathematical language familiar to physicists, a neural network cannot be easily inspected to discover the structure of its learned solution. Due to the high-dimensional nature of the input data $\boldsymbol{x}$, reverse-engineering the classification strategy to identify signal-like or background-like regions of the original space is also very challenging (99). This is not surprising, and generically we should anticipate a trade-off between performance and interpretability.

Aside from understanding the nature of the decision, the use of machine learning also complicates the scientific communication and theoretical interpretation of LHC results. While traditional cut-based analyses can be conveyed in tables and prose, a learned neural network cannot. Not only does this make it difficult for a reader to glean the essential physics, it is also an impediment to reinterpreting the result in the context of a different theoretical model. This issue further motivates analysis preservation and reinterpretation systems, such as RECAST (100, 101), that can re-execute the original data analysis pipeline.

Systematic uncertainties due to mismodeling in the simulation are also a major concern. The networks are routinely trained using large samples of simulated collisions, and the nature of their solution is relevant to experimental data only if those simulated samples are faithful descriptions of the collected data. Although the simulation programs have been extensively tuned and validated over years of use, skepticism remains about their ability to accurately describe the correlations in a high-dimensional space, leading to reasonable concerns about whether a network's learned solution relies on a well-modeled physical effect or an overlooked weak point. This concern applies to shallow networks as well, but is even more severe when working with higher-dimensional lower-level data.

One means to assuage these concerns is a classic piece of experimental scientific strategy: validation using adjacent control regions. The primary concern is whether the correlations among the input features are well modeled, which can be verified through a comparison of the network function evaluation in real and simulated data. Adjacent control regions can be employed to keep the data that are most sensitive to the hypothetical signal blind. This technique is often applied in the case of single-dimensional data analysis, and is essentially a generalization of the sideband approach. If the input feature correlations that the network function relies on are poorly modeled, the distributions of the real and simulated data will disagree. This provides some validation of the network function, but no insight into its structure.

An alternative approach relaxes the reliance on simulated samples by using real data in the training step and avoids the need for training labels by using weakly supervised learning (102, 103). In one approach, one needs to know only the proportion of labels in different subsets of data (102), for instance, samples of events with known proportions of quark and gluon jets. One strategy, known as classification without labels, requires only that the different samples of events have different label proportions even if they are unknown (103).

In other cases, the weak points of the modeling are well known to physicists, who would prefer a learned solution that avoids detailed reliance on these features. For example, most simulated samples of collisions that result in jets use either PYTHIA (6) or HERWIG (7) to model the parton shower, but these two heuristic approaches can make significantly different predictions about the jet images (50). One way to address this issue is to explicitly parameterize the network in the space of the unknown nuisance parameter (39, 83) so that the dependence can be studied or constrained with data. Another is to attempt to explicitly reduce the dependence of the network on aspects that are sensitive to the underlying uncertainties (93), making the resulting network less sensitive to these uncertainties. This can be regarded as a constrained optimization problem; for example, one might seek an optimal combination of jet substructure–tagging variables that does not distort the smooth background (94).

3.3. Software

The growing complexity of neural networks, in terms of both the architecture and the raw computational power required for training, might seem overwhelming to the pragmatic HEP physicist. Worse, the summary presented in this article is merely a brief review of state-of-the-art deep learning, and given the rapid pace at which the field evolves, techniques may change in the near future. Fortunately, a number of software packages (11–13) already provide efficient automatic gradient computation and interfaces to hardware such as GPUs. These are well supported outside HEP. Thanks to these tools, the role of the physicist is reduced to choosing an appropriate problem, data representation, architecture, and training strategy.

The software landscape continues to evolve rapidly compared with typical timescales for software in collider physics. This contrast presents a challenge for any experiment using deep

learning: By choosing one of the dozen currently available frameworks, the experiment risks being marooned with an unsupported and bloated dependency when the deep learning industry moves on. Fortunately, while projections into the future of a specific deep learning package, or even a particular architecture, would be premature, the underlying representation of a network as a stack of differentiable tensor operations has proven quite robust.

As deep learning matures, the language and specifications become more precise. Several ongoing projects with significant commercial backing seek to formalize these specifications further (104; for a list of useful tools and references bridging the gap between collider physics and machine learning, see the following repository: **https://github.com/iml-wg/HEP-ML-Resources**). Such formal specifications allow a factorization between the training phase—which can depend on specific hardware and a myriad of software packages—and the application or inference phase. Training a neural network requires millions of iterations, whereas inference requires only a single pass per classified pattern. As a result, computational demands at the inference stage are mild. The factorization enables inference-only implementations (105) or autogenerated inference functions to be the primary vehicle for incorporation into the trigger and reconstruction software of the LHC experiments.

In contrast, as machine learning is incorporated into high-level data analysis, there is a benefit to having closer integration of modern machine learning frameworks and statistical analysis software. Various tools are beginning to blend deep learning, probabilistic modeling, and statistical inference (38, 106–108).

4. PROSPECTS

In the past few years, advances in machine learning have enabled the development of tools that have the power to transform the nature of data analysis in HEP.

Expected increases in computational power and further advances in training strategies can be reasonably expected to extend the power of these tools. But important questions remain regarding how to best apply this power. Should physicists aim for end-to-end learning, providing the network with the lowest-level and highest-dimensionality data that it can effectively process, and ask it to solve the entire problem at once? Or is it more sensible to maintain outlines of the existing structures, but replace engineered solutions based on domain knowledge with learned solutions? A middle road, inspired by the hierarchical nature of the LHC data and the success of highly structured networks, suggests building end-to-end tools whose internal structure reflects the outlines of existing analysis pipelines (**Figure 5**). Planning for such future efforts is already under way (88, 89).

Beyond the application of deep learning to the problems described in this review, related research in the fields of statistical and machine learning offers promising solutions to the challenges of data analysis in particle physics. For example, the problem of modeling smooth background distributions from observed data has long been treated using ad hoc parametric functions; techniques from the study of Gaussian processes have recently been shown to provide a powerful and promising alternative (109). Another area with significant untapped potential with relevance to collider physics is that of anomaly detection; recent techniques (110) have improved the data compression and anomaly detecting speed such that applications to use these techniques to search for anomalous signatures in the LHC data set may be practical. Recently, Bayesian optimization has been used for more efficient tuning of the simulation programs (111), and adversarial training of GANs has been extended to the tuning of nondifferentiable simulators (98). Finally, several groups have used machine learning to grapple with the interpretation of results in high-dimensional parameter spaces for theories such as supersymmetry (112, 113; also see **https://indico.cern.ch/event/632141/**).

One of the most profound developments in machine learning is the ability to model high-dimensional distributions from large samples of data. The ability to estimate high-dimensional probability densities or density ratios enables probabilistic inference in situations that were previously intractable. The key machine learning developments in this area (80–85, 114) allow the tuning of the classification tool for the particular problem at hand, opening the door to deeper levels of optimization and potentially more powerful analyses.

Further work in this area includes efforts to modify the simulation tools for improved sampling of the high-dimensional space (115, 116). An exciting direction of this research is to automatically discover, in the simulation of a background process, the sequence of events that lead to rare events being misclassified as signal.

Deep learning has already influenced data analysis at the LHC and sparked a new wave of collaboration between the machine learning and particle physics communities, which is progressing at a rapid pace. While it is difficult to predict the ultimate impact these developments will have, we anticipate that new applications will be found, motivating new strategies for analysis of the LHC data and yielding deeper insights into fundamental questions in particle physics.

DISCLOSURE STATEMENT

The authors are not aware of any affiliations, memberships, funding, or financial holdings that might be perceived as affecting the objectivity of this review.

ACKNOWLEDGMENTS

The authors thank Ben Nachman for helpful comments. D.W. and D.G. are supported by the Office of Science at the US Department of Energy. K.C. is supported by the National Science Foundation (ACI-1450310 and PHY-1505463) and by the Moore-Sloan Data Science Environment at at NYU.

LITERATURE CITED

1. Hocker A, et al. *Proc. Sci.* ACAT:040 (2007)
2. Krizhevsky A, Sutskever I, Hinton GE. In *Proceedings of the 25th International Conference on Neural Information Processing Systems* (*NIPS 12*), p. 1097. New York: Curran (2012)
3. Russakovsky O, et al. *Int. J. Comput. Vis.* 115:211 (2015)
4. LeCun Y, Bengio Y, Hinton G. *Nature* 521:436 (2015)
5. Schmidhuber J. *Neural Netw.* 61:85 (2015)
6. Sjöstrand T, Mrenna S, Skands PZ. *J. High Energy Phys.* 0605:026 (2006)
7. Bahr M, et al. *Eur. Phys. J. C* 58:639 (2008)
8. Alwall J, et al. *J. High Energy Phys.* 07:079 (2014)
9. Gleisberg T, et al. *J. High Energy Phys.* 02:007 (2009)
10. Agostinelli S, et al. *Nucl. Instrum. Methods A* 506:250 (2003)
11. Keras Document. Team. Keras. *Software package*. **https://keras.io/** (2018)
12. LISA Lab. Theano. *Software package*. **http://deeplearning.net/software/theano/** (2018)
13. TensorFlow Dev. Team. TensorFlow. *Software package*. **https://www.tensorflow.org/** (2018)
14. Hornik K, Stinchcombe M, White H. *Neural Netw.* 2:359 (1989)
15. Hochreiter S. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 6:107–116 (1998)
16. Bengio Y, Simard P, Frasconi P. *Neural Netw.* 5:157 (1994)
17. Hinton G, et al. arXiv:1207.0580 [cs.NE] (2012)
18. Hinton GE, Osindero S, Teh YW. *Neural Comput.* 18:1527 (2006)

19. Bengio Y, Lamblin P, Popovici D, Larochelle H. In *Proceedings of the 19th Annual Conference on Advances in Neural Information Processing Systems* (*NIPS 06*), ed. B Schölkopf, J Platt, T Hoffman, p. 153. Cambridge, MA: MIT Press (2007)
20. Goodfellow I, Bengio Y, Courville A. *Deep Learning*. Cambridge, MA: MIT Press. **http://www.deeplearningbook.org** (2016)
21. Lecun Y, Bottou L, Bengio Y, Haffner P. *Proc. IEEE* 86:2278 (1998)
22. Ioffe S, Szegedy C. In *Proceedings of the 32nd International Conference on Machine Learning*, p. 448. proceedings.mlr.press: PMLR (2015)
23. He K, Zhang X, Ren S, Sun J. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, p. 1026. Piscataway, NJ: IEEE (2016)
24. Thulasiraman K, Swamy MNS. *Graphs: Theory and Algorithms*. New York: Wiley (1992)
25. Hochreiter S, Schmidhuber J. *Neural Comput.* 9:1735 (1997)
26. Cho K, et al. arXiv:1406.1078 [cs] (2014)
27. Aad G, et al. *J. Instrum.* 9:P09009 (2014)
28. Peterson C. *Nucl. Instrum. Methods A* 279:537 (1989)
29. Abramowicz H, Caldwell A, Sinkus R. *Nucl. Instrum. Methods A* 365:508 (1995)
30. CMS Collab. *Eur. Phys. J. C* 74:3076 (2014)
31. Abazov VM, et al. *Phys. Rev. D* 71:072004 (2005); Erratum. *Phys. Rev. D* 77:039901 (2008)
32. DELPHI Collab. *Phys. Lett. B* 295:383 (1992)
33. Kohne JK, et al. *Nucl. Instrum. Methods A* 389:128 (1997)
34. D0 Collab. *Phys. Lett. B* 517:282 (2001)
35. CDF Collab. *Phys. Rev. Lett.* 104:141801 (2010)
36. Baldi P, Sadowski P, Whiteson D. *Nat. Commun.* 5:4308 (2014)
37. Santos R, et al. *J. Instrum.* 12:P04014 (2017)
38. Louppe G, Cranmer K, Pavez J. *J. Open Source Softw.* 1:11 (2016)
39. Baldi P, et al. *Eur. Phys. J. C* 76:235 (2016)
40. Behnke T, Charlton DG. *Phys. Scr.* 52:133 (1995)
41. Peterson C, Rögnvaldsson T, Lönnblad L. *Comput. Phys. Commun.* 81:185 (1994)
42. CMS Collab. Tech. rep. CMS-PAS-BTV-15-001, CERN, Geneva (2016)
43. ATLAS Collab. *J. Instrum.* 11:P04008 (2016)
44. ATLAS Collab. Tech. rep. ATL-PHYS-PUB-2017-013, CERN, Geneva (2017)
45. Larkoski AJ, Moult I, Nachman B. arXiv:1709.04464 [hep-ph] (2017)
46. de Oliveira L, et al. *J. High Energy Phys.* 07:069 (2016)
47. Cogan J, Kagan M, Strauss E, Schwartzman A. *J. High Energy Phys.* 02:118 (2015)
48. Almeida LG, et al. *J. High Energy Phys.* 07:086 (2015)
49. Baldi P, et al. *Phys. Rev. D* 93:094034 (2016)
50. Barnard J, Dawe EN, Dolan MJ, Rajcic N. *Phys. Rev. D* 95:014018 (2017)
51. Komiske PT, Metodiev EM, Schwartz MD. *J. High Energy Phys.* 01:110 (2017)
52. Acciarri R, et al. *J. Instrum.* 12:P03011 (2017)
53. Aurisano A, et al. *J. Instrum.* 11:P09001 (2016)
54. Guest D, et al. *Phys. Rev. D* 94:112002 (2016)
55. CMS Collab. *J. Instrum.* 12:P10003 (2017)
56. ATLAS Collab. Tech. rep. ATL-PHYS-PUB-2017-003, CERN, Geneva (2017)
57. CMS Collab. Tech. rep. CMS-DP-2017-012, CERN, Geneva (2017)
58. CMS Collab. Tech. rep. CMS-DP-2017-005, CERN, Geneva (2017)
59. CMS Collab. Tech. rep. CMS-DP-2017-049, CERN, Geneva (2017)
60. Pearkes J, Fedorko W, Lister A, Gay C. arXiv:1704.02124 [hep-ex] (2017)
61. Egan S, et al. arXiv:1711.09059 [hep-ex] (2017)
62. CMS Collab. Presented at the Workshop on Deep Learning for Physical Sciences of the 31st Annual Conference on Advances in Neural Information Processing Systems (NIPS 17), Long Beach, CA (2017)
63. Louppe G, Cho K, Becot C, Cranmer K. arXiv:1702.00748 [hep-ph] (2017)
64. Cheng T. arXiv:1711.02633 [hep-ph] (2017)

65. Henrion I, et al. Presented at the Workshop on Deep Learning for Physical Sciences of the 31st Annual Conference on Advances in Neural Information Processing Systems (NIPS 17), Long Beach, CA (2017)
66. Bronstein MM, et al. arXiv:1611.08097 [cs.CV] (2016)
67. ATLAS Collab. *Eur. Phys. J. C* 77:673 (2017)
68. Salzburger A. Tech. rep. ATL-SOFT-PROC-2015-056.7, CERN, Geneva (2015)
69. Stahl M. *J. Phys. Conf. Ser.* 898:042042 (2017)
70. Machado Miguens J. Tech. rep. ATL-DAQ-PROC-2016-025, CERN, Geneva (2016)
71. Adam W, Fruhwirth R, Strandlie A, Todorov T. *J. Phys. G* 31:9 (2005)
72. ATLAS Collab. Tech. rep. ATLAS-CONF-2012-047, CERN, Geneva (2012)
73. Salzburger A. *J. Phys. Conf. Ser.* 664:072042 (2015)
74. Krishnan RG, Shalit U, Sontag D. arXiv:1511.05121 [stat.ML] (2015)
75. Krishnan RG, Shalit U, Sontag D. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, p. 2101. Menlo Park, CA: AAAI (2017)
76. Farrell S, et al. *Eur. Phys. J. Web Conf.* 150:00003 (2017)
77. Farrell S, et al. Presented at the Workshop on Deep Learning for Physical Sciences of the 31st Annual Conference on Advances in Neural Information Processing Systems (NIPS 17), Long Beach, CA. **https://dl4physicalsciences.github.io/files/nips_dlps_2017_28.pdf** (2017)
78. Amrouche S, et al. *Eur. Phys. J. Web Conf.* 150:00015 (2017)
79. ATLAS Collab. Tech. rep. ATL-PHYS-PUB-2010-013, ATL-COM-PHYS-2010-838, CERN, Geneva (2010)
80. Larochelle H, Murray I. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, p. 29. Cambridge, MA: MIT Press (2011)
81. Jimenez Rezende D, Mohamed S. arXiv:1505.05770 [stat.ML] (2015)
82. Dinh L, Krueger D, Bengio Y. arXiv:1410.8516 [cs.LG] (2014)
83. Cranmer K, Pavez J, Louppe G. arXiv:1506.02169 [stat.AP] (2015)
84. Kingma DP, Salimans T, Welling M. arXiv:1606.04934 (2016)
85. Papamakarios G, Murray I, Pavlakou T. In *Proceedings of the 31st Annual Conference on Advances in Neural Information Processing Systems* (*NIPS 17*), p. 2335. New York: Curran
86. Paganini M, de Oliveira L, Nachman B. arXiv:1705.02355 [hep-ex] (2017)
87. de Oliveira L, Paganini M, Nachman B. arXiv:1701.05927 [stat.ML] (2017)
88. Elmer P, Neubauer M, Sokoloff MD. arXiv:1712.06592 [physics.comp-ph] (2017)
89. Alves AA Jr., et al. arXiv:1712.06982 [physics.comp-ph] (2017)
90. Aaltonen T, et al. *Phys. Rev. Lett.* 102:152001 (2009)
91. Stevens J, Williams M. *J. Instrum.* 8:P12013 (2013)
92. Ajakan H, et al. arXiv:1412.4446 [stat.ML] (2014)
93. Louppe G, Kagan M, Cranmer K. arXiv:1611.01046 [stat.ME] (2016)
94. Shimmin C, et al. arXiv:1703.03507 [hep-ex] (2017)
95. Snoek J, Larochelle H, Adams RP. arXiv:1206.2944 [stat.ML] (2012)
96. Head T, et al. Scikit-Optimize. *Software package*. **https://scikit-optimize.github.io/** (2017)
97. Staines J, Barber D. arXiv:1212.4507 [stat] (2012)
98. Louppe G, Cranmer K. arXiv:1707.07113 [stat] (2017)
99. Chang S, Cohen T, Ostdiek B. arXiv:1709.10106 [hep-ph] (2017)
100. Cranmer K, Yavin I. *J. High Energy Phys.* 04:038 (2011)
101. Cranmer K, Heinrich L. *J. Phys. Conf. Ser.* 898:102019 (2017)
102. Dery LM, Nachman B, Rubbo F, Schwartzman A. *J. High Energy Phys.* 05:145 (2017)
103. Metodiev EM, Nachman B, Thaler J. *J. High Energy Phys.* 10:174 (2017)
104. ONNX Dev. Team. Open Neural Network Exchange (ONNX). *Software package*. **https://github.com/onnx/onnx** (2017)
105. Guest DH, et al. lwtnn (version 2.6). *Software package*. **https://github.com/lwtnn/lwtnn/releases** (2017)
106. Tran D, et al. Edward. *Software package*. **http://edwardlib.org** (2017)
107. Tran D, et al. arXiv:1610.09787 [stat] (2016)
108. Pyro Dev. Team. Pyro. *Software package*. **http://pyro.ai** (2017)

109. Frate M, et al. arXiv:1709.05681 [physics.data-an] (2017)
110. Luo C, Shrivastava A. arXiv:1706.06664 [cs.DB] (2017)
111. Ilten P, Williams M, Yang Y. *J. Instrum.* 12:P04028 (2017)
112. Bertone G, et al. arXiv:1611.02704 [hep-ph] (2016)
113. Caron S, et al. *Eur. Phys. J. C* 77:257 (2017)
114. Cranmer K, Louppe G. *J. Brief Ideas*. **http://doi.org10.5281/zenodo.198541** (2016)
115. Le TA, Baydin AG, Wood F. arXiv:1610.09900 [cs.AI] (2016)
116. Casado ML, et al. Presented at the Workshop on Deep Learning for Physical Sciences of the 31st Annual Conference on Advances in Neural Information Processing Systems (NIPS 17), Long Beach, CA (2017)