TOPICAL REVIEW • OPEN ACCESS

Graph neural networks in particle physics

To cite this article: Jonathan Shlomi *et al* 2021 *Mach. Learn.: Sci. Technol.* **2** 021001

View the article online for updates and enhancements.

MACHINE LEARNING
Science and Technology

TOPICAL REVIEW

Graph neural networks in particle physics

Jonathan Shlomi[1] , Peter Battaglia[2] and Jean-Roch Vlimant[3]

[1] Weizmann Institute of Science, Rehovot, Israel
[2] DeepMind, London United Kingdom
[3] California Institute of Technology, PMA, Pasadena, CA, 91125-0002, United States of America

E-mail: jvlimant@caltech.edu

Abstract

Particle physics is a branch of science aiming at discovering the fundamental laws of matter and forces. Graph neural networks are trainable functions which operate on graphs—sets of elements and their pairwise relations—and are a central method within the broader field of geometric deep learning. They are very expressive and have demonstrated superior performance to other classical deep learning approaches in a variety of domains. The data in particle physics are often represented by sets and graphs and as such, graph neural networks offer key advantages. Here we review various applications of graph neural networks in particle physics, including different graph constructions, model architectures and learning objectives, as well as key open problems in particle physics for which graph neural networks are promising.
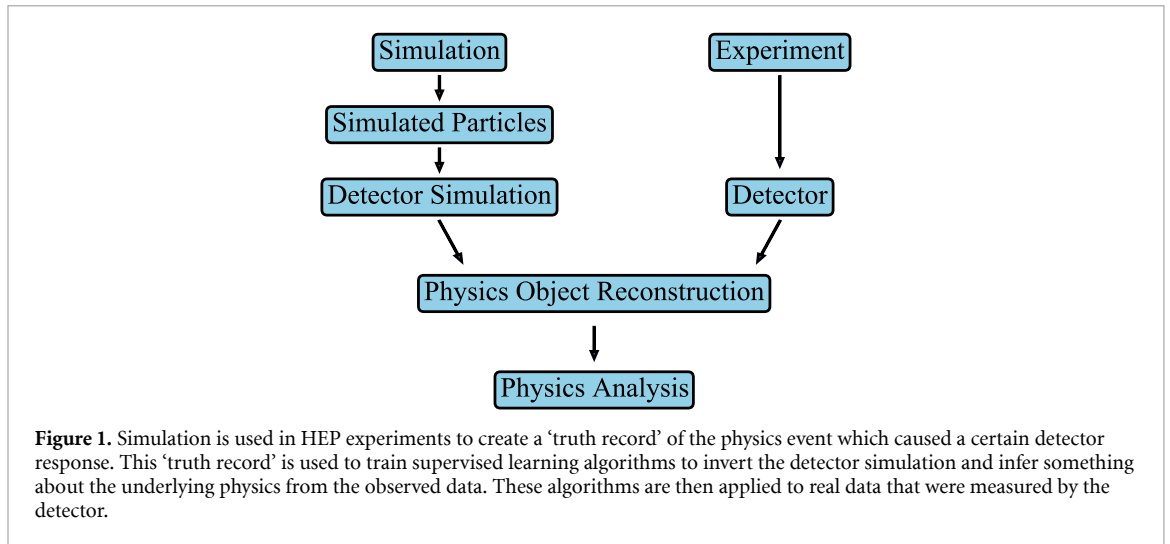
1. Introduction

Particle physics focuses on understanding fundamental laws of nature by observing elementary particles, either in controlled environments (collider physics) or in nature (astro-particle). The standard model of particle physics is a theory of the strong, weak and electromagnetic forces, and elementary particles (quarks and leptons). Physicists are building experiments to measure elementary particles and by using statistical methods can test the validity of various models. The data from the experiments are generally a sparse sampling of a physics process in both time and space.

Machine learning has historically played a significant role in particle physics [1], with classification and regression applications using classical techniques, such as boosted decision trees, support vector machine, simple multi-layer perceptrons, etc. Inspired by the success deep learning has achieved at reaching super-human performance at various tasks, various domains in the physical sciences [2], including particle physics [1, 3–6], have begun exploring deep learning as a unique tool for handling difficult scientific problems that go beyond straightforward classification, to organize and make sense of vast data sources, draw inferences about unobserved causal factors, and even discover physical principles underpinning complex phenomena [7, 8].

High Energy Physics (HEP) experiments often use machine learning for learning complicated inverse functions, trying to infer something about the underlying physics process from the information measured in the detector. This scheme is illustrated in figure 1.

While the most widely used trio of deep learning building blocks—the fully connected network (FC), convolutional neural network (CNN) and recurrent neural network (RNN)—have proven valuable across many scientific domains, the focus of this review is on a class of architectures called graph neural networks (GNNs)—as described below, we regard self-attention as a graph-based architecture—,which can be trained from data to learn functions on graphs. Many problems involve data represented as unordered sets of elements with rich relations and interactions with one another, and can be naturally expressed as graphs. They are however not convenient to represent as vectors, grids, or sequences—the format required by FCs, CNNs, and RNNs, respectively—unless for specific structure of tree [9, 10]. Extensive reviews of GNNs are available in the literature [11–15]. However applications of GNNs in HEP are evolving rapidly, and the purposes of this review are to outline the key principles and uses of GNNs for particle physics, and build

Figure 1. Simulation is used in HEP experiments to create a 'truth record' of the physics event which caused a certain detector response. This 'truth record' is used to train supervised learning algorithms to invert the detector simulation and infer something about the underlying physics from the observed data. These algorithms are then applied to real data that were measured by the detector.

bridges between physics and machine learning by exposing researchers on both sides to important, challenging problems in each others' domains.

1.1. Data representation

Measurements in particle physics are commonly done in large accelerator facilities (CERN KEK, Fermilab, etc.), using detectors with sizes on the order of tens of meters, which capture millions of high-dimensional measurements each second. These detectors are composed of multiple sub-detectors—tracking detector, calorimeters, muon detector, etc.—each using a different technology to measure the trace of particles. The data in particle physics are therefore heterogeneous. Detectors in astrophysics are typically bigger, with size up to kilometers (IceCube, Antares, etc.) constructed around a single measurement technology, the data are therefore homogeneous. In both cases, the measurements are inherently sparse in space, due to the design of the geometry of the sensors. The measurements therefore do not a-priori fit homogeneous, grid-like data structures.
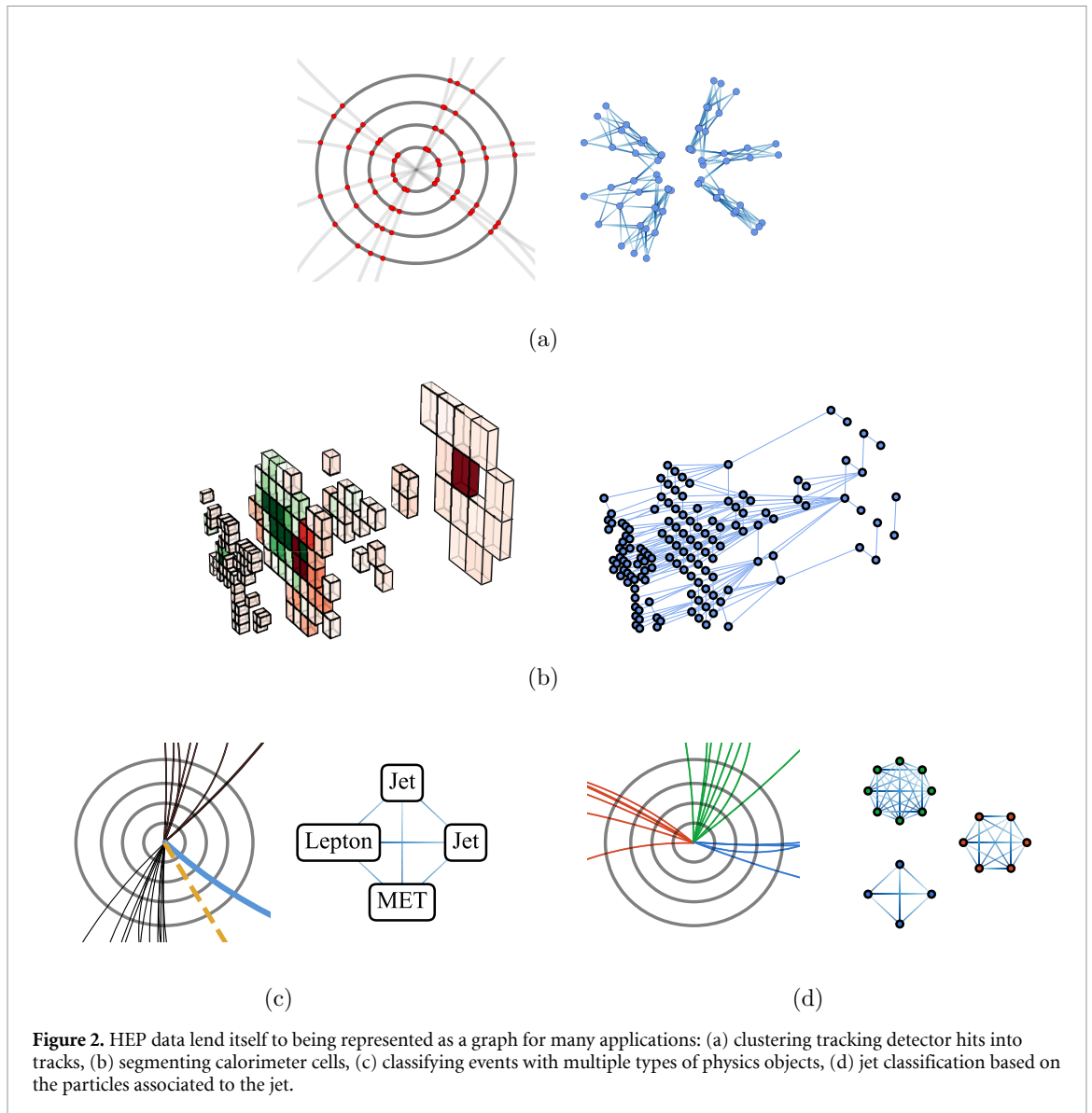
Deep learning is often applied on high level features derived from particle physics data [1]. This can improve over more classical data analysis methods, but does not use the full potential of deep learning, which can be effective when operating on lower level information.

Some data in particle physics can be fractionally interpreted as images and hence computer vision techniques (CNNs) are being applied with improved performances [16–21]. However, image representations face some limitations with irregular geometry of detectors or sparsity of the projections applied. Because of the inherent loss of information, image representations may constrain the amount of information that can be extracted from the data.

Measurement and reconstructed objects can be viewed as sequences, with an order imposed from theoretical or experimental understanding of the data. Methods otherwise applied to natural language processing (e.g. RNNs, LSTMs [22] and GRUs [23]) have thus been explored [24, 25]. While the ordering used can usually be justified experimentally, it is often imposed and therefore constrains how the data are presented to models. This ordering can also be learned [26] in some cases, using prior experimental knowledge of the physics process at stake. This is however not always the case and one may expect that the imposed ordering will reduce the learning performance—ordering that is not required as we will see in the following. For example [27] shows evidence that a permutation invariant network outperforms a sequence based algorithm that uses the exact same input features, for the same classification task.

At many levels the data are, by definition, sets (unordered collection) of items. If one considers relation between items (geometrical, or physical) a set transforms into a graph with the addition of an adjacency matrix. There is a-priori less limitation in applying deep learning on this intrinsic representation of the data, than at the other levels mentioned above. A variety of HEP data and their formulation as graphs is illustrated in figure 2.

We concentrate in this review on the applications of GNNs to HEP. We argue why graphs are a very useful data representation, and review key architectures. Common traits in graph construction and model architecture will be linked to the specific requirements of the HEP problems under consideration. By providing a normalized description of the models through the formalism introduced in [13] we hope to make the adoption and further development of GNNs for HEP simpler.

Figure 2. HEP data lend itself to being represented as a graph for many applications: (a) clustering tracking detector hits into tracks, (b) segmenting calorimeter cells, (c) classifying events with multiple types of physics objects, (d) jet classification based on the particles associated to the jet.

This review paper is organized as follows. An overview of the field of geometrical deep learning is given in section 2. Existing applications to particle physics are reviewed in 3. General guidelines for formulating HEP tasks for GNNs are given in section 4. In particular we go in the details of the different approaches in building the graph connectivity in section 4.2, the various model architecture adopted in section 4.3. This paper concludes with a discussion on the various approaches and the remaining open questions in section 5.

2. Geometric deep learning

2.1. Overview

Deep learning has been central to the past decade's advances in machine learning and artificial intelligence [28, 29], and can be understood as the confluence of several key factors. First, large neural networks can express very complex functions. Second, valuable information in big data can be encoded into the parameters of large neural networks via gradient-based training procedures. Third, parallel computer hardware can perform such training in hours or days, which is efficient enough for many important use cases. Fourth, well-designed software frameworks, such as TensorFlow [30] and PyTorch [31], lower the technical bar to developing and distributing deep learning applications, making powerful machine learning tools broadly accessible to practitioners.

Fully connected, convolutional, and recurrent *layers* have been the primary building blocks in modern deep learning, each of which carries different *inductive biases*, which incentivize or constrain the learning algorithm to prioritize one solution over another. For example, convolutional layers share their underlying kernel function across spatial dimensions of the input signal, while recurrent layers share across the temporal

dimension of the input. These building blocks are most suitable for approximating functions on vectors, grids, and sequences, but when a problem involves data with richer structure, these modules are not always convenient or effective to apply. For example, consider learning functions over sets of particles—while it is possible to order them, for example sorting by the transverse momentum p_T of the particle, the imposed ordering in not unique, and it fails to reflect that particles are fundamentally unordered. The aforementioned deep learning modules do not have appropriate inductive biases to exploit this richer graphical structure.

Graph-structured data are ubiquitous across science, engineering, and many other problem domains. A graph is defined, minimally, as a set of nodes as well as a set of edges adjacent to pairs of nodes. Richer varieties and special cases include: trees, where there is exactly one sequence of edges connecting any two nodes; directed graphs, where the two nodes associated with an edge are ordered; attributed graphs, which include node-level, edge-level, or graph-level attributes; multigraphs, where more than one edge may exist between a pair of nodes; hypergraphs, where more than two nodes are associated with an edge; etc. Crucially, graphs are a natural and powerful way of representing many complex systems [11–15], e.g. trees for representing evolution of species, or the hierarchical structure of sentences; lattices and meshes for representing regular and irregular discretizations of space, respectively; dynamic networks for representing traffic on roads and social relationships over time.

GNNs [11–13, 32] are a class of deep learning architectures which implement strong relational inductive biases for learning functions that operate on graphs. They implement a form of parameterized message-passing whereby information is propagated across the graph, allowing sophisticated edge-, node-, and graph-level outputs to be computed. Within a GNN there are one or more standard neural network building blocks, typically fully connected layers, which implement the message computations and propagation functions. The first GNNs [32, 33] were developed and applied for network analysis, especially on internet data, and were trained not with the back-propagation algorithm, but with fixed point iteration via the Almeida–Pineda algorithm [34, 35]. Li *et al*'s [36]'s gated graph sequence neural networks helped integrate more recent deep learning innovations into GNNs, adding RNN modules for improving multiple rounds of message-passing and optimizing their parameters by the back-propagation learning rule [28, 29].

In recent years, the field of GNNs has grown very rapidly, with applications to science and engineering. For example, graph convolution has been used for molecular fingerprinting [37]. Message-passing neural networks [12], which provided a general formulation of GNNs which captured a number of previous methods, were introduced for quantum chemistry. Interaction networks [38] and graph networks [13] have been developed for learning to simulate increasingly complex physical systems [38–41].

GNNs are situated within the broader family of what Bronstein *et al* [11] term *geometric deep learning*, which, aside from GNNs, captures related deep learning methods which apply to data structures beyond vectors, tensors, sequences, etc. Their survey explores graph signal processing and how it can be connected to deep learning, with substantial discussion on how the general principles of CNNs applied to Euclidean signals can be transferred to graph-structured signals. Key examples of spectral graph convolution approaches are [42–44], which applied neural networks to the eigenvalues and eigenvectors of the graph Laplacian.

Much work on GNs has focused on learning physical simulation [38–40, 45], similar to Lagrangian methods for particle-based simulation in engineering and graphics. The system is represented as a set of particle vertices, whose interactions are represented by edges and computed via learned functions. Recent work by [41] highlights how far this sub-field has advanced: they trained models to predict systems of thousands of particles, which represent fluids, solids, sand, and 'goop', and show generalization to orders of magnitude more particles and longer trajectories than experienced during training. Because GNs are highly parallelizable on modern deep learning hardware (GPUs, TPUs, FPGAs), their approach scaled well, and its speed was on par with heavily engineered state-of-the-art fluid simulation engines, despite that they did not optimize for speed in their work.

Recently GNs have been extended by adding inductive biases derived from physics, adjusting their architectures to be consistent with Hamiltonian [46] and Lagrangian mechanics [47], which can improve performance and generalization on various physical prediction problems. Other recent work [7] has shown symbolic physical laws can be extracted from the learned functions within a GN.

2.2. The graph network formalism

Here we focus on the *graph network* (GN) formalism [13], which generalizes various GNNs, as well as other methods (e.g. Transformer-style self-attention [48]). GNs are graph-to-graph functions, whose output graphs have the same node and edge structure as the input. Adopting [13]'s formalism, a graph can be represented by, $G = (\mathbf{u}, V, E)$, with N_v vertices and N_e edges. The $\mathbf{u}$ represents graph-level attributes. The set of *nodes* (or vertices) are $V = \{\mathbf{v}_i\}_{i=1:N_v}$, where $\mathbf{v}_i$ represents the ith node's attributes. The set of edges are $E = \{(\mathbf{e}_k, r_k, s_k)\}_{k=1:N_e}$, where $\mathbf{e}_k$ represents the kth edge's attributes, and r_k and s_k are the indices of the two (*r*eceiver and *s*ender, respectively) nodes connected by the k-th edge.
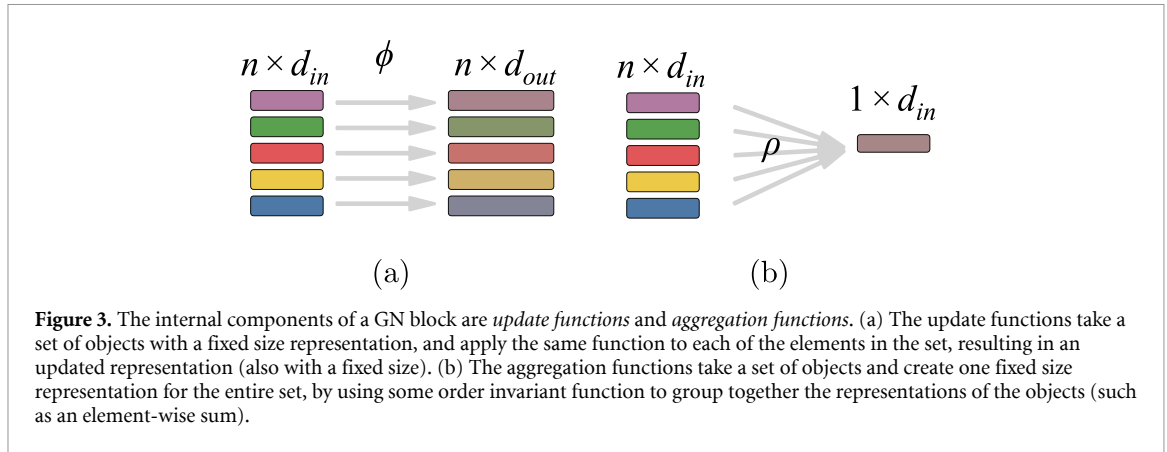
Figure 3. The internal components of a GN block are *update functions* and *aggregation functions*. (a) The update functions take a set of objects with a fixed size representation, and apply the same function to each of the elements in the set, resulting in an updated representation (also with a fixed size). (b) The aggregation functions take a set of objects and create one fixed size representation for the entire set, by using some order invariant function to group together the representations of the objects (such as an element-wise sum).

A GN's stages of processing are as follows.

$$
\begin{aligned}
\mathbf{e}'_k &= \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right) & \bar{\mathbf{e}}'_i &= \rho^{e \to v}\left(E'_i\right) & &\triangleright \text{Edge block} \\
\mathbf{v}'_i &= \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right) & \bar{\mathbf{e}}' &= \rho^{e \to u}\left(E'\right) & &\triangleright \text{Vertex block} \\
\mathbf{u}' &= \phi^u\left(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}\right) & \bar{\mathbf{v}}' &= \rho^{v \to u}\left(V'\right) & &\triangleright \text{Global block}
\end{aligned}
\tag{1}
$$

A GN block contains 6 internal functions: 3 *update functions* (ϕ^e, ϕ^v, and ϕ^u) and 3 *aggregation functions* ($\rho^{e \to v}$, $\rho^{e \to u}$, and $\rho^{v \to u}$). The GN formalism is not a specific model architecture, it does not determine what exactly those functions are. The update functions are functions of fixed size input and fixed size output, and the aggregation functions take in a variable-sized set of inputs (such as a set of edges connected to a particular node) and output a fixed size representation of the input set. This is illustrated in figure 3.

The *edge block* computes one output for each edge, $\mathbf{e}'_k$, and aggregates them by their corresponding receiving node, $\bar{\mathbf{e}}'_i$, where E'_i is the set of edges incident on the ith node. The *vertex block* computes one output for each node, $\mathbf{v}'_i$. The edge- and node-level outputs are all aggregated in order to compute the *global block*. The output of the GN is the set of all edge-, node-, and graph-level outputs, $G' = (\mathbf{u}', V', E')$. See figure 4(a).

In practice the ϕ^e, ϕ^v, and ϕ^u are often implemented as a simple trainable neural network, e.g. a FC. The $\rho^{e \to v}$, $\rho^{e \to u}$, and $\rho^{v \to u}$ functions are typically implemented as permutation invariant reduction operators, such as element-wise sums, means, or maximums. The ρ functions must be permutation invariant if the GN block is to maintain permutation equivariance.
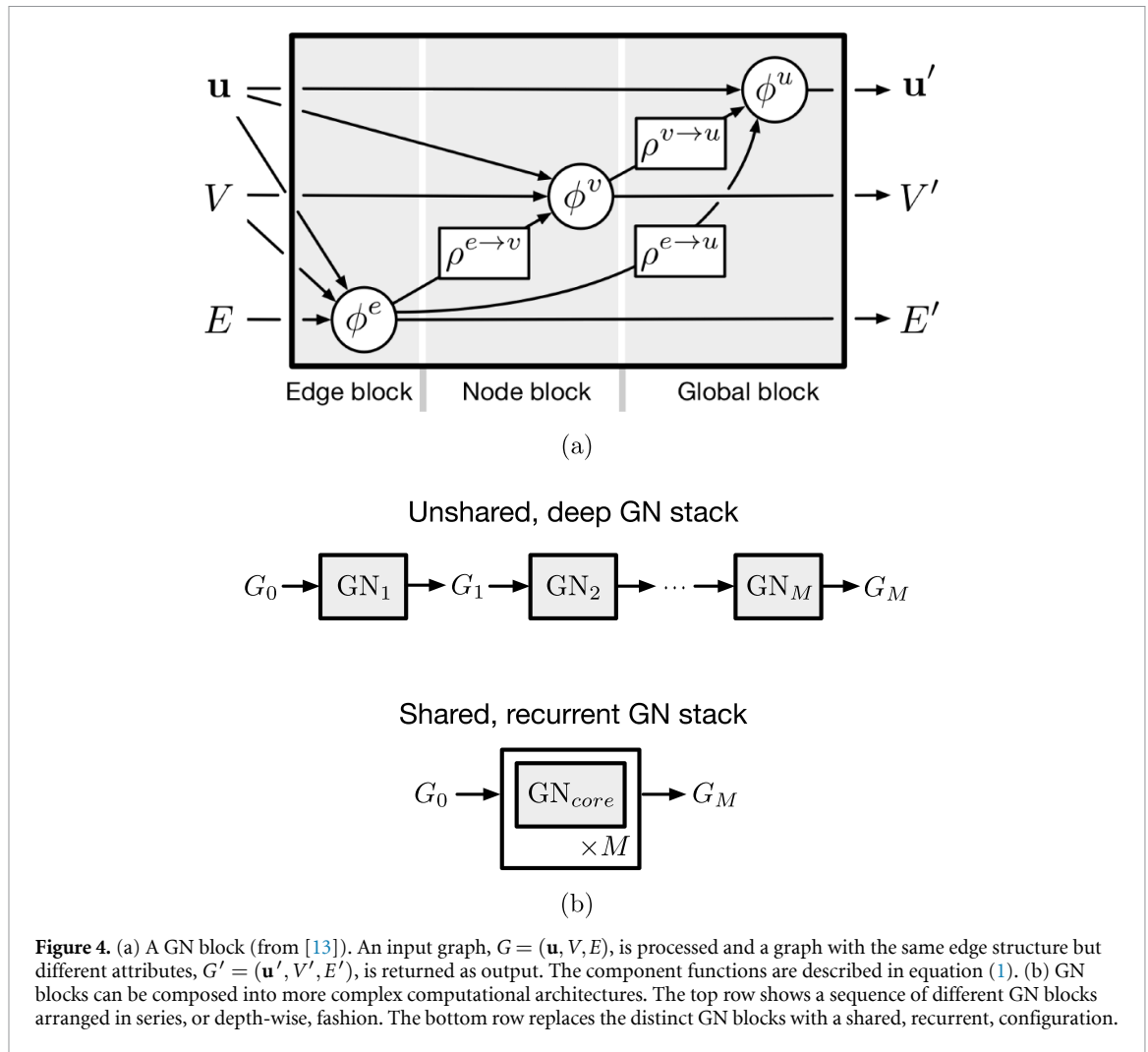
Some key benefits of GNs are that they are generic: if a problem can be expressed as requiring a graph to be mapped to another graph or some summary output, GNs are often suitable. They also tend to generalize well to graphs not experienced during training, because the learning is focused on the edge- and node-level—in fact if the global block is omitted, the GN is not even aware of the full graph in any of its computations, as the edge and node blocks take only their respective localities as input. Yet when multiple GN blocks are arranged in deep or recurrent configurations, as in figure 4(b), information can be processed and propagated across the graph's structure, to allow more complex, long-range computations to be performed.

The GN formalism is a general framework which can capture a variety of other GNN architectures. Such architectures can be expressed by removing or rearranging internal components of the general GN block in figure 4, and implementing the various ϕ and ρ functions using specific functional forms. For example, one very popular GNN architecture is the graph convolutional network (GCN) [49]. Using the GN formalism [12, 13], a GCN can be expressed as,

$$
\mathbf{e}'_k = \phi^e\left(\mathbf{e}_k, \mathbf{v}_{s_k}\right) = \mathbf{e}_k \mathbf{v}_{s_k}, \qquad \text{where } \mathbf{e}_k = \frac{1}{\sqrt{\text{degree}(r_k)\text{degree}(s_k)}}
$$

$$
\bar{\mathbf{e}}'_i = \rho^{e \to v}\left(E'_i\right) = \sum_{\{k \,|\, r_k = i\}} \mathbf{e}'_k
$$

$$
\mathbf{v}'_i = \phi^v\left(\bar{\mathbf{e}}'_i\right) = \sigma\left(\bar{\mathbf{e}}'_i W\right)
$$

Figure 5 shows the correspondence between the GCN and the GN depicted in figure 4.

In section 4 we will discuss the considerations taken into account when deciding how to choose the actual implementation of the GNs internal functions. The choice of the specific architecture is motivated by the relationships that exist between the elements in the input data and the task one is trying to solve with the model.

Figure 4. (a) A GN block (from [13]). An input graph, $G = (\mathbf{u}, V, E)$, is processed and a graph with the same edge structure but different attributes, $G' = (\mathbf{u}', V', E')$, is returned as output. The component functions are described in equation (1). (b) GN blocks can be composed into more complex computational architectures. The top row shows a sequence of different GN blocks arranged in series, or depth-wise, fashion. The bottom row replaces the distinct GN blocks with a shared, recurrent, configuration.

3. Survey of applications to particle physics

Beyond discriminating signals from background in physics analysis, machine learning can be applied in many of the steps of the event: triggering, reconstruction and simulation. GNNs are used in three different ways to make predictions: at the level of the graph, or node, or edge, depending on the task at hand. We described briefly below the challenges and the methods applied, coming back in further details in 4. All the presented methods were developed on simulated events, and no performance on real data is reported so far.

In each line of work described below, a decision was first made about how the data could be expressed as a graph: What are the entities and relations which would be represented as nodes and edges, respectively? What is the required output, i.e. edge-, node-, or graph-level predictions? From there, choices about the specific GNN architecture were made to reflect the desired computation: Is a global output network required to produce graph-level outputs? Should pairwise interactions among nodes be computed, or more GCN-like summation and non-linear transformation? How many message-passing steps should be used, in order to propagate information among distant nodes in the graph?

3.1. Graph classification

3.1.1. Jet classification

Jets or *showers* are sprays of stable particles that are stemming from multiple successive interaction and decays of particles, originating from a single initial object. The identification of this original object is of paramount importance in particle physics. Because of the rather large lifetime of the b-hadrons [50] and hence a significantly displaced decay vertex, identification of b-jet (*b-tagging*) using classical methods has been rather successful. With the advent of deep learning methods, lower level information has been used to improve the performance of b-tagging, and opened the possibility of identifying jets coming from other particle (c-hadron, top-quark, tau, etc.). The jets coming from pure hadronic interaction driven by quantum chromo-dynamics (QCD) (so called *QCD jets*), are covering an extremely large phase space and constitute an
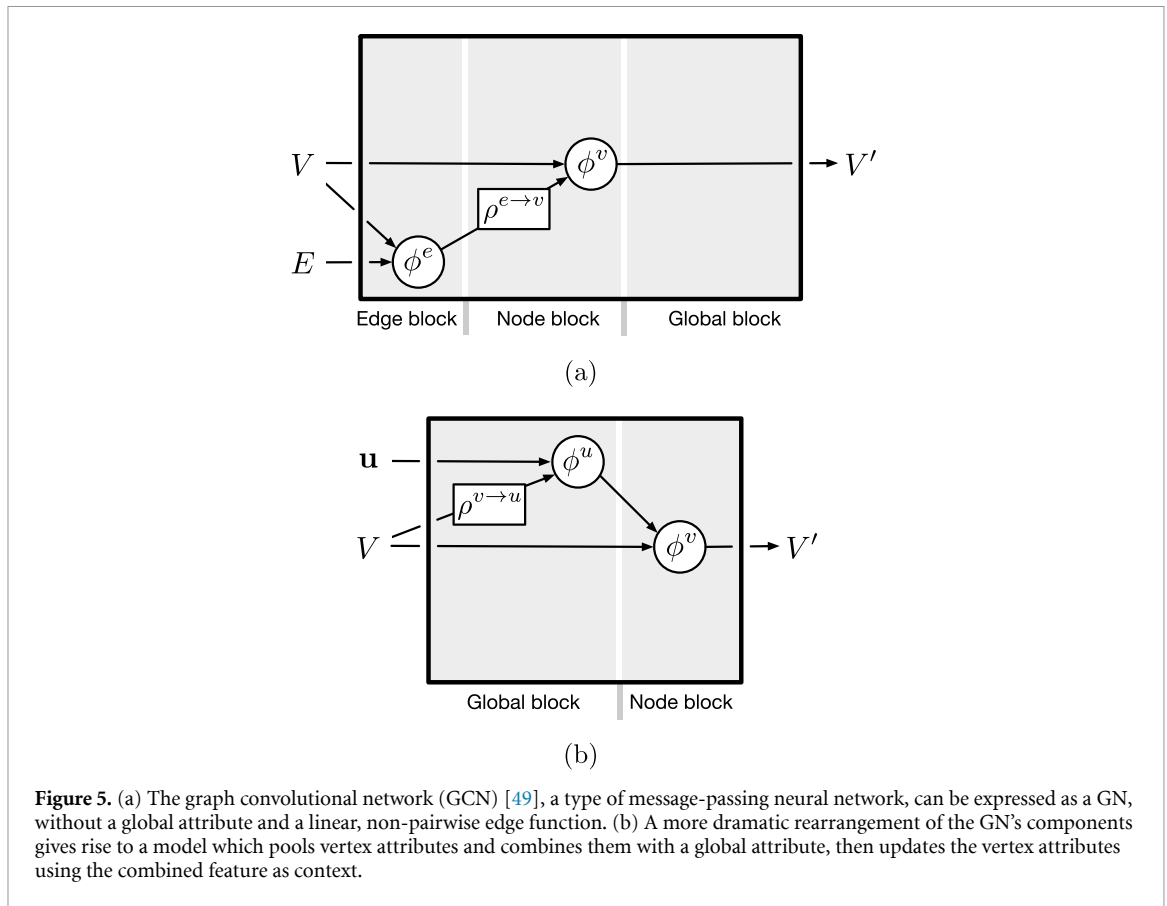
Figure 5. (a) The graph convolutional network (GCN) [49], a type of message-passing neural network, can be expressed as a GN, without a global attribute and a linear, non-pairwise edge function. (b) A more dramatic rearrangement of the GN's components gives rise to a model which pools vertex attributes and combines them with a global attribute, then updates the vertex attributes using the combined feature as context.

irreducible background to other classes of jets. In particular, within the framework of the particle flow reconstruction [51], the event is interpreted through a set of particle candidates. As such, in references [52–58] the collection of particle candidates is represented on a graph and various methods are applied.

The authors of [52] use a fully connected graph, and message passing architecture to learn the adjacency matrix, comparing several directed and undirected graph constructions. The classification of jets originating from the hadronic decay of a *W* boson and QCD jets is shown to improve with the proposed method. Work on physics-based inductive biases is left for future work to improve the learning of the adjacency matrix. It should be noted that learning the adjacency matrix is related to learning attention in [57].

In [54] the authors use the *edgeconv* method from [59] to derive a point cloud architecture for jet tagging. The connectivity of the graph is defined dynamically by computing node neighborhoods over the distance in either the input space, or an intermediate latent space when graph layers are stacked. The architecture respects the particle permutation invariance by mean of averaging of contributions from the connected neighbors. The performance of this model for the quark/gluon discrimination (separating jets originating from a quark or a gluon) and top tagging (discriminating hadronic top decay and *QCD* jet) tasks is reported to be better than other previously studied architectures. The learned edge function is constrained to taking as input a node feature and the feature difference between this node and the connected node. In [58] the same model architecture is applied to the specific case of semi-visible jet originating from the cascade decay of hypothetical dark hadrons. The method outperforms neural networks that operate on images, as well as models including physical inductive biases [60]. The authors demonstrate an order of magnitude improvement on the sensitivity of dark matter search when using this method.

The authors of [55, 56] take inspiration from [38] and adapt the interaction network architecture to the purpose of graph categorization. Using a fully connected graph over the particles of a jet and primary vertices of the event, a graph category is extracted after one step of message passing. The performance of this model on a multi-class categorization (light quarks, gluon, W and Z bosons hadronic decays, and hadronic top jets) is better than other non-graph-based architectures against which it was compared. On the specific use case of tagging jets which stem from Higgs bosons decaying onto a pair of b quarks, the algorithm outperforms state of the art methods, even when the proper mass decorrelation method [61] is applied. The authors report some potential computation performance issues with running the model for predictions. The measurement however, is done with a model obtained from a format conversion between major frameworks, and the performance could be improved with a native implementation instead.

With [53] the authors applied the *Deep Sets* method from [62] to jet tagging. They propose a simplified model architecture with provable physics properties, such as infrared and colinear safety. The features of each particle are encoded into a latent space and the graph category is extracted from the summed representation in that latent space. The model has no connectivity, and thus no attention or message passing, and pools information globally across all the elements before the categorization is output, and yet the performance of this simple model on the quark/gluon classification is surprisingly on par with other more complicated models. The authors provide ways of interpreting what the model has learned, and are able to extract closed-form observables from their trained model.

In [57] the *graph attention network* from [63] is adapted for graph categorization. The node and edge features are created and updated by means of multiple fully connected neural networks, operating on the graph, and an additional attention factor, equivalent to a weighted, directed adjacency matrix is computed per directed edge, and used in the update rule. A k-nearest neighborhood connectivity pattern is constructed using the distance over the edge features, initialized to the difference between node features, and later in a latent space when using stacked graph layers. Stability of the models is improved with the use of a multi-head mechanism, and skip connections at multiple level are added to facilitate the information flow. Their model outperforms the model from [54] on the quark/gluon classification task, indicating the importance of the attention mechanism—to which we come back to in sections 4.3 and 5.

3.2. Event classification

Here we use the term *event* for the capture by an experiment of the full history of a physics process. In astroparticle, for example it is the collection of signals that covers the interaction of an high energy particle interacting with the atmosphere. The jet tagging task presented in the previous section is part of a full event identification in collider physics. Event classification is the task of predicting or inferring the physics process at the origin of the recorded data.

The authors of [64] applied a graph convolution method for the classification of the signal in the IceCube detector, to determine if a muon originated from a cosmic neutrino, or from a cosmic ray showering in the earth atmosphere. The adjacency matrix of a fully connected graph of the detector sensors is constrained to a Gaussian kernel on the physical distance, with a learnable locality parameter. Node features are updated by application of the adjacency matrix and non-linear activation. The graph property is extracted from the sum over the latent features of the nodes of the graph. This GNN model yields a signal-to-background ratio about three times as big as the baseline analysis of such signal.

In [65], the *message passing neural network* architecture from [12] is used over a fully connected graph composed of the final state particles, and the missing transverse energy. Messages are computed from the node features and a distance in the azimuth-rapidity plane first, then in the node latent space for later iterations. Such messages are passed across the graph in two iterations, and each node receives a categorization. The node-averaged value is used to predict the event category. The model is compared to densely connected models, and is showing superior performance when comparing the $S/\sqrt{B}$ analysis significance. From the same authors, in [66, 67], a similar architecture is applied to event classification for other signal topologies, demonstrating the versatility of the method.

3.3. Node Classification and Regression

3.3.1. Pileup mitigation

In a view to increase the overall probability of producing rare processes and exotic events, the particle density of bunches composing the colliding beams can be increased. This results in multiple possible interactions per beam crossing. The downside of this increased probability is that, when occurring, an interesting interaction will be accompanied with other spurious, less interesting interactions (*pileup*), considered as noise for the analysis. Mitigation of pileup is of prime importance for analysis at colliders. While it is rather easy to suppress charged particles by virtue of the primary vertex they are originating from, neutral particles are harder to suppress. In a particle flow reconstruction [51], the state of the art is to compute a pileup weight per particle [68], and use it for mitigation.

In [69] the authors utilize the *gated graph network architecture* [36] to predict a per particle probability of belonging to the pileup part of the event. The graph is composed of one node per charged and neutral particle in the event, and the connectivity is imposed to $\Delta R \equiv \sqrt{\delta\phi^2 + \delta\eta^2} < 0.3$ in the azimuth-pseudorapidity plane. An averaged R-dependent message is computed and gated with each previous node representation by mean of a gated recurrent unit (GRU) to form the new node representation. The per-particle pileup probability is extracted with a dense model, after three stacked graph layers, and a skip connection into the last graph layer. The model outperforms other standard methods for pileup subtraction and improves resolution of several physical observables.

The authors of [57] take inspiration from the *graph attention network* from [63] to predict a per-particle pileup probability. An architecture very similar to the one used for the jet classification (described previously) is used to create a global graph latent representation, which in turn is used to compute an output that is mapped back to each node, thanks to a given order of the latter. This method is shown to improve the resolution on the jet and di-jet mass observables, while being stable over a large range of pileup density.

3.3.2. Calorimeter reconstruction

A *calorimeter* is a detector which goal is to contain and measure the total energy of a system. In particle physics, a calorimeter is commonly composed on the one hand of inactive material inducing showering of particles and energy loss (*absorber*), and on the other hand a sensitive material that aims at measuring the collective released energy in the absorber. Reconstruction of the energy of the incoming particle in such a sampling calorimeter involves calibration and clustering of the signal of various cells.

With [70] a GN based approach is proposed to cluster and assign the signal in a high granularity calorimeter to two incoming particles. A latent edge representation is constructed in the latent space of the nodes, using a potential function of the distance also in the latent space. Two methods are proposed for the graph connectivity, one—*GravNet*—using nearest neighbors in a latent space, the other — *GarNet* — using a fixed number of additional nodes (dubbed *aggregator*) in the graph. Node features are updated using concatenated message from multiple aggregation methods, and provides in output the fraction of energy of the cell belonging to each particle. The proposed methods are slightly improving over more classical approaches, and could be beneficial in more complex detector geometry than the one studied.

3.3.3. Particle flow reconstruction

Typically, detectors in particle physics are composed of multiple sub-detectors with various sensing technologies. Each sub-detector is targeting the measurement of specific characteristic of the particle. The assembly of all measurements allows for the characterisation of the particle properties. The *particle flow*—or *energy flow*—reconstruction is an algorithm that aims at assigning to a candidate particle all the measurements in each sub-detector [51]. Since all particles produced during a collision can potentially be reconstructed, *particle flow reconstruction* allows for fine grained interpretation and analysis of collision events.

The author of [71] proposes the *object condensation* loss formulation, using a GNN method to extract the particles' information from the graph of individual measurements. In this context, the model is set to predict the properties of a smaller number of particles than there are measurements, in essence doing a graph reduction. A stacked-*GravNet*-based model performs node-wise regression of a kinematic corrective factor together with a *condensation weight*. The latter indicates whether a node of the graph has to be considered as representative of a particle in the event, and have its regressed quantities be assigned to that particle. The performance of this algorithm is compared with a baseline particle-flow algorithm on rather sparse large hadron collider (LHC) environments. The proposed method is shown to be more efficient and produces less fake particles than the standard approach.

3.3.4. Efficiency parameterization

The analysis of particle physics data—in particular collider experiment data—requires applying selection criteria on the large volume of data, in a view to enhance the proportion of interesting signals. It is crucial to determine with as little uncertainty as possible the fraction of signal passing these selections, if one wants to measure the rate of production of that signal during the experiment. Much care is taken to determine these selection efficiencies, as they play significant roles in measuring the cross section of known processes, or while setting limits on production of unknown signals. The efficiencies can be measured from data or simulation, per event or any component of it. It is often the case that the efficiency of a specific selection on a component of the full events also depends on the other components of the event. Taking into account the correlation between all components of an event is a hard task that machine learning can help with.

The authors of [72] use GNNs to learn the per-jet tagging efficiency, from a fully connected graph representation of the jets in the event. The model is a message passing GNN. The edge update and node updates are both implemented as simple FN. The final node representation is used to predict the per-jet efficiency for each jet in an event. The GN allows taking into account the dependency of the per-jet efficiency on the other jets in the event. The comparison is made with the classical method of explicitly parameterizing the per-jet efficiency with a two dimensional histogram, whose axis are the jet transverse momentum and pseudo-rapidity. The authors show how the graph representation and GNN parametrisation allows improving determination of the per-jet efficiency, compared to the more traditional method.

3.4. Edge classification

3.4.1. Charged particle tracking

Charged particles have the property of ionizing the material they traverse. This property is utilized in a tracking device (*tracker*) to perform precise measurement of the passage of charged particles. Contrary to calorimeters, trackers should not alter too much the energy of the incoming particle, as such it usually produces a sparse spatial sampling of the trajectory. The reconstruction of the trajectory of original particles amounts to finding what set of isolated measurement (*hits*) belong to the same particle. Most tracking devices are embedded in a magnetic field that will curve the trajectories and hence provide a handle at measuring the particle momentum component transverse to the magnetic field, since this quantity and the curvature are inversely proportional.

The authors of [73] propose a GNN approach to charged particle tracking using edge classification. Each node of the graph represents one sparse measurement, or hit, with edge constructed between pairs of hits with geometrically plausible relations. Using multiple updates of the node representation and edge weight over the graph (using the edge weight as attention), the model learns what are the edges truly connecting hits belonging to the same track. This approach transforms the clustering problem into an edge classification that defines the sub-graphs of hits belonging to the same trajectory. The performance of this method has high accuracy when applied in a simplified case, and is promising for more realistic scenarios. In [74], a GNN model involving message passing is presented and provides improved performance.

3.4.2. Secondary vertex reconstruction

The particles within a jet often originate from various intermediate particles that are worth identifying for the purpose of identifying the origin of the jet (see the paragraph on jet identification above). The decay of the intermediate particles are identified as secondary vertices within the jet, using clustering algorithms on the particles, such as the adaptive vertex reconstruction [75]. Based on the association to secondary vertex, the particles within a jet can henceforth be partitioned.

In [76], the authors develop a general formalism for *set-to-graph* neural networks and provide mathematical proof that their formulation is a universal approximation of function mapping a graph structure onto an input set—all invariance taken into account. In particular, they apply a *set-to-2-edge* — predicting single edge characteristics from the input set—approximation to the problem of particle association within a jet. The model is a composition of an embedding model, a fixed broadcasting mapping and a graph-to-graph model. All components are actually rather simple and the expressivity of the full model stems from the specific equivariant formulation. Their model outperforms the standard methods on jet partitioning by about 10% over multiple metrics.

## 4. Formulating HEP tasks with GNN

The articles described in section 3 make use of multiple graph connectivity schemes, model architecture and loss functions. Experience shows that using our knowledge about the underlying physics in order to encode the relationship between the nodes—whatever they may represent—in both the input graph and the model architecture is key in developing algorithms. Unfortunately it is not always clear which methods and model architectures will outperform the others. This section aims to clarify the choices made and provide a checklist of considerations for the particle physicist looking to develop a new application using a GNN.

4.1. Task definition

The first step is to decide what function one wants to learn with the GNN. In some applications this is trivial—for example jet, event or particle classification. In those cases a GNN is used to learn some representation of the node or the entire graph/set and a standard classifier is trained on that representation.

For tasks such as segmentation or clustering, there is a choice between formulating the task as edge classification or something like the object condensation method which uses node representations to formulate a partition of the input set. The object condensation method has an important advantage, in that it computes relationships between objects (the attractive or repulsive potential) only while training the algorithm, in the computation of the loss function. An edge classifier will learn an edge representation and use that to classify edges. The number of edges can be large, increasing the computation and memory requirements of the algorithm. The determination of the set partition in the object condensation method is a simple function of the node representation, which greatly reduces those requirements.

None of the work presented in section 3 is using a mapping of the input onto the edges of the graph. Because an edge can only link two nodes—while a node can be connected to as many edges as desirable—construction of such graph would require a specific structure of the input. One such use case could be in situations where observations arise from two concurrent measurements, such as hit position in
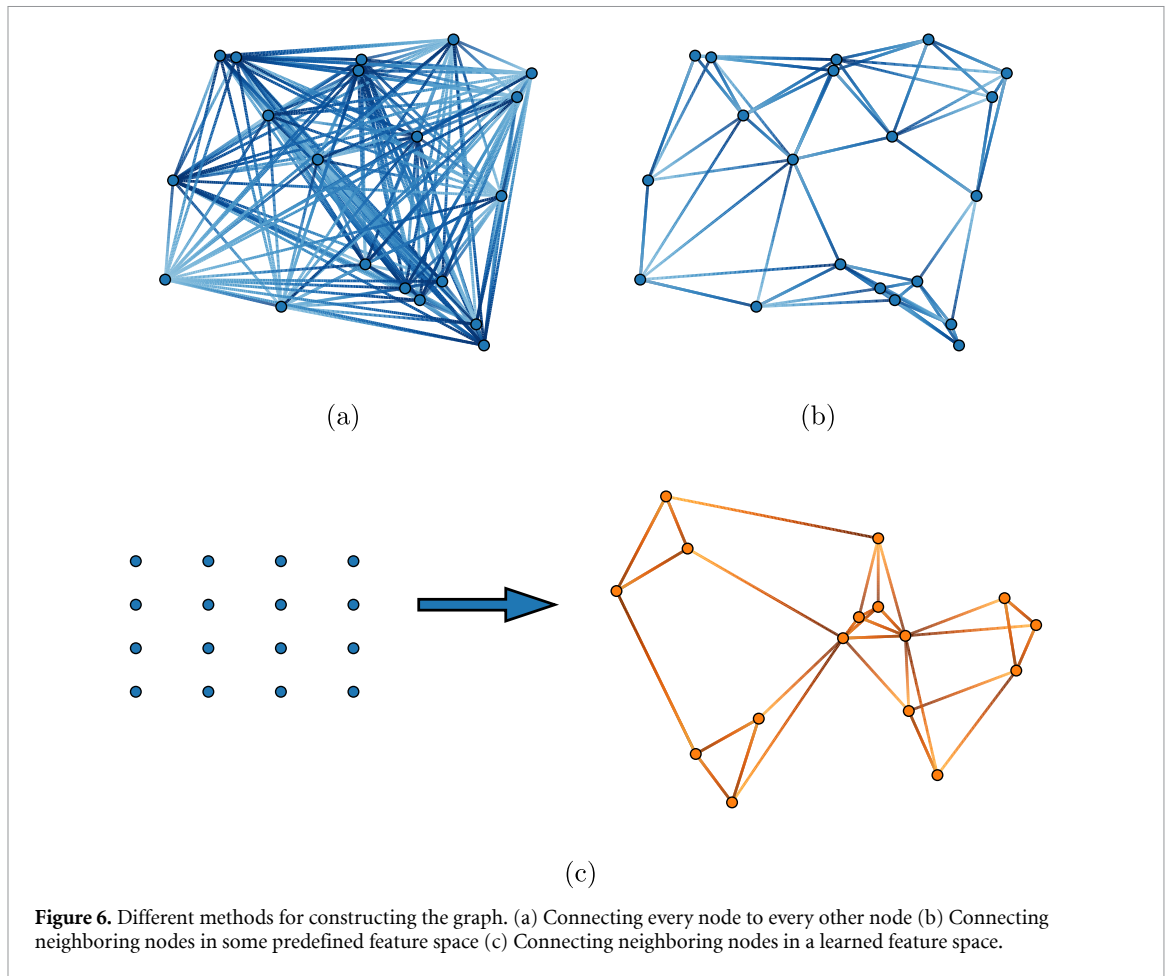
Figure 6. Different methods for constructing the graph. (a) Connecting every node to every other node (b) Connecting neighboring nodes in some predefined feature space (c) Connecting neighboring nodes in a learned feature space.

stereo strip detectors. The detector is composed of two rectangular modules with a thin strip of sensors along one dimension, and the modules are tilted with respect to each other by a couple of degrees so as to have the strip sensors overlapping and hence creating a grid. With strip measurement positioned on the nodes, the important information would be located on the edges, as a combination of two such hits. Other examples in network communication might also be relevant.

4.2. Graph construction

In most particle physics applications, the nature of the relationships between different elements in the set are not clear cut (as it would be for a molecule or a social network). Therefore a decision needs to be made about how to construct a graph from the set of inputs. Different graph construction methods are illustrated in figure 6. Depending on the task, one might even want to avoid creating any pairwise relationships between nodes. If the objects have no pairwise conditional dependence—a DeepSet [53] architecture with only node and global properties might be more suitable. Edges in the graph serve 3 roles:

(a) The edges are communication channels among the nodes.
(b) Input edge features can indicate a relationship between objects, and can encode physics motivated variables about that relationship (such as ΔR between objects).
(c) Latent edges store relational information computed during message-passing, allowing the network to encode such variables it sees relevant for the task.

In cases where the input sets are small ($N_v \sim \mathcal{O}(10)$) the typical and easiest choice is to form a fully connected graph, allowing the network to learn which object relationships are important. In larger sets, as the number of edges between all nodes increases as $N_e \propto (N_v)^2$, the computational load of using a neural network to create an edge representation or compute attention weights becomes prohibitive. One possible work-around is to choose a fixed edge feature that is easy to pre-compute—such as distance between detector modules.

If an edge-level computation is required, it is necessary to only form some edges. Edges can be formed based on a relevant metric such as the ΔR between particles in a detector, or the physical distance between

detector modules. Given a distance measure between nodes, some criterion for connecting them needs to be formulated, such as connecting k-nearest neighbors in the feature space.

The node features used to connect edges can also be based on a learned representation. This is sometimes referred to as *dynamic* graph construction, and used by the EdgeConv [54] and GravNet [70] architectures, for example. We will discuss this in more detail in section 4.3, showing the connection between the idea of dynamic graph construction and attention mechanisms.

When the graph is constructed dynamically, such as using the node representation to connect edges between k-nearest neighbors, the gradient of the neural network parameters is only affected by those nodes that have actually been connected. Since the indexing of node-neighborhood is non differentiable, its parameters cannot be learn with gradient descent, but can be optimized on hyper-parameter search.

In initial stages of the training, the edge formation is essentially random, allowing the network to explore which node representations should be closer together in the latest space. During later stages of the training, one may wish to encourage further exploration by the network. One possible way to do this is to inject random edges—for example besides connecting nodes to k-nearest neighbors in latest space, connecting an additional small number of random connections to nodes further away in the latent space.

A recent paper [77] introduces a reinforcement learning agent which traverses an input graph to reach nodes which should be connected by new edges. Its policy is optimized for some downstream task performance, so that the nodes it chooses to connect with new edges improve the task performance.

4.3. Model architecture

Designing the model architecture should reflect a logical combination of the inputs towards the learning task. In the language of the GN formalism (section 2.2), we need to select a concrete implementation of the GN block update and aggregation functions ϕ and ρ, and decide how to configure their sequence inside the GN block. Additionally we need to decide which kinds of GN blocks we want to combine and how to stack them together. As explained in section 2.2, different architectures such as Graph Convolution Networks, Graph Attention Networks, are specific choices for constructing a GNN—but they are all equivalent in the sense that their output is a graph with learned node/edge/graph representations which are then used to perform the actual task.
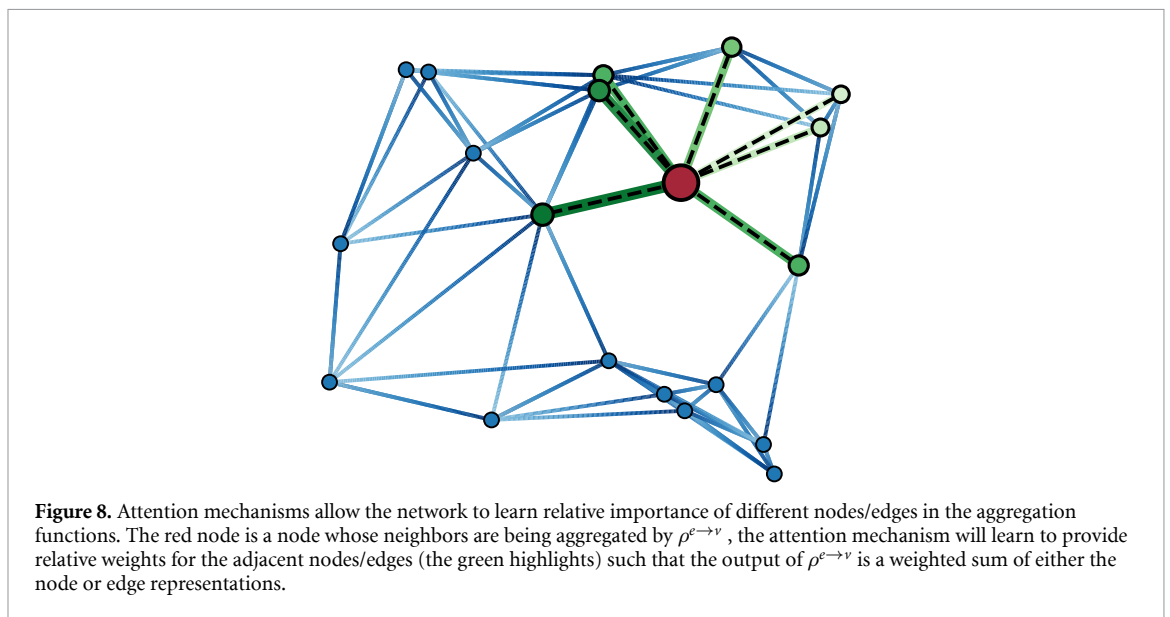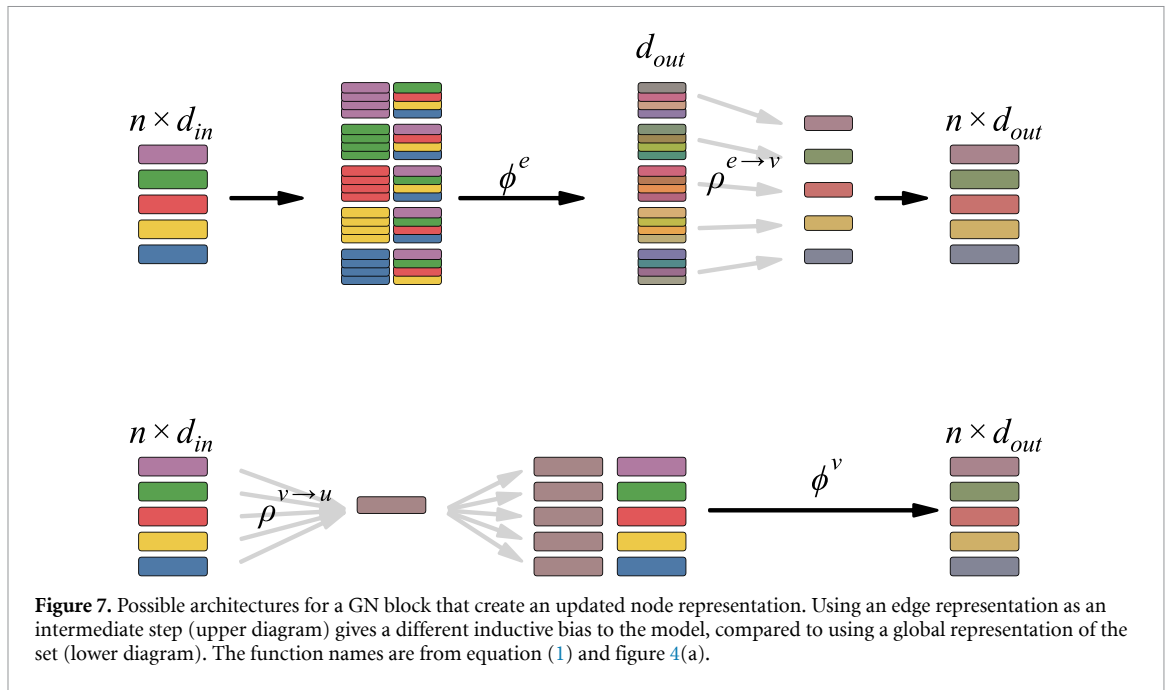
4.3.1. GN block functions

The key question here is what logical steps one would take to form the GN block output in a way that serves the task, and which parts of this logical process should be modeled with neural networks? The most general GN block (as shown in figure 4(a)) could have all of its update functions implemented as neural networks, which allows the most flexibility in the learning processes. This flexibility might not be required for the task, and it might carry computational costs that we wish to keep to a minimum. Therefore its probably better to start with a simple architecture, and only add complexity gradually, until the algorithms performance is satisfactory.

Figure 7 shows two examples of possible configurations, either creating an edge representation before aggregating edges and forming a node update, or using global aggregation before a node update. Both configurations result in an updated node representation, but one of them is based on a sum of pair-wise representations, and the other on a global sum of node representations—the information content is the same, but the inductive bias is different. For example, the authors of [72] assumed that the jet-tagging efficiency is heavily affected by the ΔR between neighboring jets—therefore an edge update step created a representation of pair-wise interaction between jets, which was then summed for each jet to create the updated node representation. In contrast the authors of [53] used a DeepSet architecture, where each node representation is created independently from its neighbors, the node representations are then summed to create the graph representation, with each node representation weighted by the particles energy.

4.3.2. Attention mechanisms

Another important component that can be used in defining the $\rho^{e \to v}$ and $\rho^{v,e \to u}$ aggregation functions is using *attention mechanisms*, as illustrated in figure 8. The term *attention* is rooted in the perceptual psychology and neuroscience literatures, where it refers to the phenomenon and mechanisms by which a subset of incoming sensory information is selected for more extensive processing, while other information is deprioritized or filtered out.

The key consideration for defining and adding an attention mechanism is whether different parts of the input data are more important than others. For example, in classifying jets, some particles that originate from a secondary decay are an important footprint of a particular class of jets—therefore those particles may be more important for the classification task. There are a few different implementations of attention mechanisms. They all share the basic concept of using a neural network or a pre-defined function to

Figure 7. Possible architectures for a GN block that create an updated node representation. Using an edge representation as an intermediate step (upper diagram) gives a different inductive bias to the model, compared to using a global representation of the set (lower diagram). The function names are from equation (1) and figure 4(a).

Figure 8. Attention mechanisms allow the network to learn relative importance of different nodes/edges in the aggregation functions. The red node is a node whose neighbors are being aggregated by $\rho^{e \to v}$, the attention mechanism will learn to provide relative weights for the adjacent nodes/edges (the green highlights) such that the output of $\rho^{e \to v}$ is a weighted sum of either the node or edge representations.

compute weights which represent the relative importance of different elements in a set. In the GN block ρ functions, these weights are used to create weighted sums of the representations of the different elements.

Here we want to draw attention to the connection between attention mechanisms and dynamic graph construction. Figure 9 shows the structure of two architectures discussed in section 3, the EdgeConv, GravNet layers. These are both GN block implementations, they take as input a set of nodes (without explicit edges) and output an updated node representation. Both begin with a node embedding stage, which creates a node representation without exchanging information between the nodes. This node embedding (or only part of its feature vector, in the case of GravNet) is interpreted as a position of the node in a latent euclidean space, and edges are formed between k-nearest neighbors. This can be thought of as a fully connected graph with an attention mechanism that assigns a weight of 1 to nodes within the set of k-nearest neighbors, and 0 otherwise. The advantage of this procedure over using a neural network to compute attention weights is the much lower computational cost of both computing the edge attention weight and the subsequent edge-related operations.

It is worth noting that the GarNet layer [70] can be described as a form of *multi-headed* self-attention mechanism [48]. The GarNet layer interprets the node embedding as s different 'distances' (with s being the dimension of the embedding). These distances are attention weights over each node of the graph, and they

Figure 9. The GN block structure of the EdgeConv, GravNet layers as described in the GN formalism. The node embedding stage is a GN block which operates on the nodes independently (without any information exchange between them), followed by a GN block which creates an edge representation for every pair of vertices, aggregates edges for each node and then updates the vertex representation. The edge update function ϕ^e does not use a neural network, but uses a pre-defined function of the node representation—leading to a reduction in computational cost.

Figure 10. Each iteration of message passing between nodes increases a nodes receptive field. For example the node in red communicates with its three connected neighbors (red outline) in the first message passing step. The orange and yellow dotted lines represent the nodes that communicate after two and three iterations, respectively. The node left out of the yellow line have not exchanged information with the red node, after three iterations only.

are used to compute s different weighted sums—these are the s different *heads* of the attention mechanism. The weighted sums are propagated back to the nodes again via attention weights of each node to each of the s attention heads. The reason GarNet is computationally affordable without a hard cutoff—such as k-nearest neighbors—is that ϕ^v, the node embedding function, is the only one computed with a neural network. The attention weights are all computed with pre-defined functions given the node embedding (specifically, the function is $\exp(-|w|)$ where w is the attention weight).

4.3.3. Stacking GN blocks
A stack of GN blocks (as described in figure 4(b)) serves two purposes. First, in the same way that stacked layers in any neural network architecture (such as a CNN) can be thought of as gradually constructing a high level representation of the data, GN blocks arranged sequentially serve the same purpose for constructing the node/edge and graph representations. Therefore, additional GN blocks increase the depth of the model and its expressive power.

Second, after one iteration of message passing in a single GN block, the node has only exchanged information with its immediate connected neighbors. This is illustrated in figure 10. Multiple iterations with a GN block (either the same block applied multiple times, or different blocks applied in a sequence) increase each nodes receptive field, as the representation of its neighboring nodes was previously updated with information from their neighbors. Often skip or residual connections, which combine the input with the output, are used to prevent corruption of the updated representations, and preservation of the gradient signal, over many message passing steps, as is common in CNNs and RNNs.

5. Summary and discussion

The papers reviewed in section 3 can be seen as the first wave of application of graph neural network architectures to diverse tasks in HEP. The methods show superior performance over other model architecture, thanks to the inductive bias, reduction of number of parameters, more elaborated loss function, and above all a much more natural data representation. Graphs are constructed from observable in various ways, often with sparse connectivity to lessen computational requirements.

While multiple architectures are presented with different names, and slightly different formalisms, they all share the core concept of exchanging information across the graph. We deciphered the variety of models in section 4 by providing some considerations on how the models were build. We provide in the following some new directions to be considered as future direction for the next generation of graph neural network applications in HEP.

5.1. Transformer, reformer, etc

Following the discussion of the GravNet and EdgeConv layers in section 4.3 and their relation to attention mechanisms, another class of models which are closely related to GNNs, and which perform a type of soft structural prediction, are Transformer architectures, based on the self-attention mechanism [48]. In GNN language, a Transformer computes normalized edge weights in a complete graph (i.e. a graph with edges connecting all pairs of nodes), and passes messages along the edges in proportion to these weights, analogous to a hybrid of graph attention networks [63] and GCNs [49].

In GN notation, described in [13] and used explicitly in graph attention networks [63], the Transformer uses a ϕ^e which produces both a vector message and a scalar unnormalized weight, and the $\rho^{e \to v}$ function normalizes the weights before computing a weighted sum of the message vectors. This allows a set of input items to be treated as nodes in a graph, without observed input edges, and the edge structure to be inferred and used within the architecture for message-passing. Different variants of attention mechanisms are a way to give different weights in the pooling operations $\rho^{e \to v}$, $\rho^{v,e \to u}$, as illustrated if figure 8. The implementation of attention should reflect the nature of the interaction between the objects in the set, as they relate to the task.

The Reformer [78] architecture overcomes the quadratic computational and memory costs that challenge traditional Transformer-based methods, by projecting nodes into a learned high-dimensional embedding space where nearest neighbors are efficiently computed to inform a sparse graph over which to pass messages. The recent Linformer [79] method is similar, but with a low rank approximation to the soft adjacency matrix.

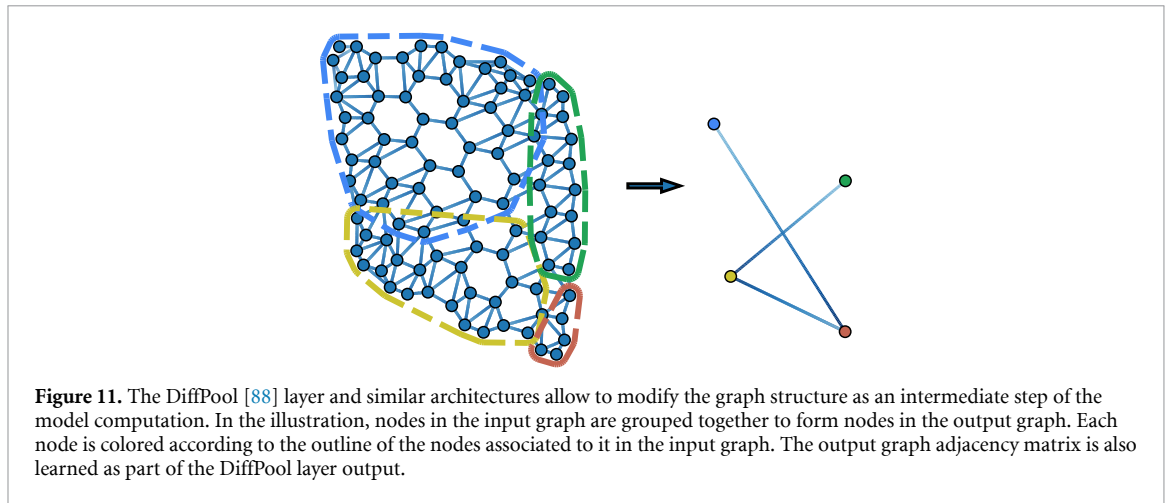### 5.2. Graph generative models

Importantly, the GN does not predict structural changes directly. However, many recent papers use GNs (or other GNNs) to decide how to modify a graph's structure. For example, [80] and [81] are autogressive graph generators, which use a GN or Transformer to predict whether a new vertex should be added to a graph (by the graph-level output), and which existing vertices to connect it to with edges (by the vertex-level outputs). The GraphRNN [82], and Graphite [83] are generative models over edges that use an RNN for sequential prediction, and GraphGAN [84] is an analogous method based on generative adversarial networks. [85]'s Neural Relational Inference treats the existence of edges as latent random variables, and trains a posterior edge inference front-end via variational autoencoding. In [86] and [87], a GN is used to guide the policy of a reinforcement learning agent and build graphs that represent physical scenes. The DiffPool [88] architecture (illustrated in figure 11 is an attention-based soft edge prediction mechanism, but over hierarchies of graphs, where lower-level ones are pooled to higher-level ones.

Generative models of graphs have not been explored much in particle physics, though some unpublished work is on-going. The need for computational resource for simulation in particle physics is almost as large as the requirements for event reconstruction. There is a breadth of efforts on using machine learning as surrogate simulators in particle physics. For the reasons exposed in section 1 that data in particle physics can often be represented as graphs, it is natural to investigate the use of generative models using graphs as a possible solution. Models under development are for example predicting energy deposition in the cells of a calorimeter or the particle candidates obtained from a particle flow reconstruction algorithm. In all cases, the generated quantities are naturally represented as a set or graph, with fixed or variable size.

5.3. Computation performance

An important consideration for building and efficiently training GNNs on hardware is whether to use dense or sparse implementations of the graph's edges. The number of edges in a graph usually defines the memory and speed bottleneck, because there are typically more edges than nodes and the ϕ^e function is applied the most times. A dense adjacency matrix supports fast, parallel matrix multiplication to compute $E\prime$, which, for example, is exploited in speed-efficient GCN- and Transformer-style models. The downside is that the

Figure 11. The DiffPool [88] layer and similar architectures allow to modify the graph structure as an intermediate step of the model computation. In the illustration, nodes in the input graph are grouped together to form nodes in the output graph. Each node is colored according to the outline of the nodes associated to it in the input graph. The output graph adjacency matrix is also learned as part of the DiffPool layer output.

adjacency matrix's memory footprint is quadratic in the number of nodes. Alternatively, using sparse adjacency matrices allows the memory to scale linearly in the number of edges, which allows much larger graphs to be processed. But the sparse indexing operations required to implement sparse matrix multiplication can incur greater time costs than their dense counterparts—this is an active area of development for both software and hardware acceleration. However, sparse operations are a key bottleneck in current deep learning hardware, and should next generation hardware substantially improve their speed, this would potentially improve the relative advantage of sparse edge implementations of GNNs. In a computing environment in HEP, one cannot expect to have access to dedicated accelerators (GPU, TPU, FPGA, etc.)—although work is going in the direction of building the infrastructure—and one needs to keep into consideration the time for running the model in production.

5.4. Final remarks

Neural networks that operate on sets are increasing in popularity in HEP tasks, both in event reconstruction, and in physics analysis. These neural networks are performing well in proof-of-concept studies, either surpassing or matching existing state of the art techniques. They have not yet been tested in the field with real detector data. It is important to understand that all of the models use the same basic building blocks to perform their tasks, and the most important consideration in designing the architecture for these neural networks is to correctly model the nature of interaction between the objects in the input set. It is probably the best practice to start with a simple graph model and architecture then build up on additional complexity geared towards incorporating scientific understanding of the physical process at stake.

ORCID iDs

Jonathan Shlomi ⬤ https://orcid.org/0000-0002-2628-3470
Peter Battaglia ⬤ https://orcid.org/0000-0003-3622-7111
Jean-Roch Vlimant ⬤ https://orcid.org/0000-0002-9705-101X

References

[1] Radovic A, Williams M, Rousseau D, Kagan M, Bonacorsi D, Himmel A, Aurisano A, Terao K and Wongjirad T 2018 Machine learning at the energy and intensity frontiers of particle physics *Nature* **560** 41–8

[2] Carleo G, Cirac I, Cranmer K, Daudet L, Schuld M, Tishby N, Vogt-Maranto L and Lenka Z 2019 Machine learning and the physical sciences *Rev. Mod. Phys.* **91** 045002

[3] Guest D, Cranmer K and Whiteson D 2018 Deep Learning and its application to LHC physics *Ann. Rev. Nucl. Part. Sci.* **68** 161–81

[4] Bourilkov D 2020 Machine and deep learning applications in particle physics *Int. J. Mod. Phys.* A **34** 1930019

[5] Larkoski A J, Moult I and Nachman B 2020 Jet substructure at the Large Hadron Collider: a review of recent advances in theory and machine learning *Phys. Rept.* **841** 1–63

[6] Community H E P 2020 *A Living Review of Machine Learning for Particle Physics* (https://iml-wg.github.io/HEPML-LivingReview/)

[7] Cranmer M D, Rui X, Battaglia P, and Shirley H 2019 Learning symbolic physics with graph networks arXiv:1909.05862

[8] Cranmer M, Sanchez-Gonzalez A, Battaglia P, Rui X, Cranmer K, Spergel D, and Shirley H 2020 Discovering symbolic models from deep learning with inductive biases arXiv:2006.11287

[9] Mou L, Li G, Zhang L, Wang T and Jin Z 2014 Convolutional Neural Networks Over Tree Structures for Programming Language Processing arXiv:1409.5718

[10] Shen Y, Tan S, Sordoni A and Courville A 2018 *Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks* Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks arXiv:1810.09536

[11] Bronstein M M, Bruna J, LeCun Y, Szlam A and Vandergheynst P 2017 Geometric deep learning: going beyond euclidean data *IEEE Signal Process. Mag.* **34** 18–42

[12] Gilmer J, Schoenholz P F, Riley O V and Dahl G E 2017 *Neural Message Passing for Quantum Chemistry* Neural Message Passing for Quantum Chemistry arXiv:1704.01212

[13] Battaglia P *et al* 2018 *Relational Inductive Biases, Deep Learning and Graph Networks* Relational Inductive Biases, Deep Learning and Graph Networks arXiv:1806.01261

[14] Zhou J, Cui G, Zhang Z, Yang C, Liu Z, Wang L, Changcheng Li, and Sun M 2018 Graph neural networks: A review of methods and applications arXiv:1812.08434

[15] Zonghan W, Pan S, Chen F, Long G, Zhang C and Yu P S 2019 A Comprehensive Survey on Graph Neural Networks arXiv:1901.00596

[16] The ATLAS Collaboration 2017 Quark versus Gluon Jet Tagging Using Jet Images with the ATLAS Detector *Technical Report* No. ATL-PHYS-PUB-2017-017 CERN

[17] Kasieczka G, Plehn T, Russell M and Schell T 2017 Deep-learning Top Taggers or The End of QCD? *JHEP* **05** 006

[18] Macaluso S and Shih D 2018 Pulling out all the tops with computer vision and deep learning *JHEP* **10** 121

[19] Andrews M, Paulini M, Gleyzer S and Poczos B 2020 End-to-end physics event classification with CMS open data: applying image-based deep learning to detector data for the direct classification of collision events at the LHC *Comput. Softw. Big Sci.* **4** 6

[20] Lin J, Freytsis M, Moult I and Nachman B 2018 Boosting $H \rightarrow b\bar{b}$ with Machine Learning *JHEP* **10** 101

[21] ATLAS 2019 Convolutional Neural Networks with Event Images for Pileup Mitigation with the ATLAS Detector

[22] Hochreiter S and Schmidhuber Jurgen 1997 Long short-term memory *Neural Comput.* **9** 1735–80

[23] Cho K, Bart Van Merriënboer, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, and Bengio Y 2014 Learning phrase representations using RNN encoder-decoder for statistical machine translation arXiv:1406.1078

[24] The ATLAS Collaboration 2017 Identification of Jets Containing b-Hadrons With Recurrent Neural Networks at the Atlas Experiment *Technical Report* No. ATL-PHYS-PUB-2017-003 CERN

[25] Collaboration C M S 2020 Identification of heavy, energetic, hadronically decaying particles using machine-learning techniques *JINST* **15** P06005

[26] Louppe G, Cho K, Becot C and Cranmer K 2019 QCD-aware recursive neural networks for jet physics *JHEP* **01** 057

[27] Deep sets based neural networks for impact parameter flavour tagging in ATLAS *Technical Report* ATL-PHYS-PUB-2020-014, CERN, Geneva, May 2020

[28] Schmidhuber Jurgen 2015 Deep learning in neural networks: An overview *Neural Netw.* **61** 85–117

[29] LeCun Y, Bengio Y and Hinton G 2015 Deep learning *Nature* **521** 436–44

[30] Abadi M *et al* 2015 TensorFlow: large-scale machine learning on heterogeneous systems Software available from (https://tensorflow.org)

[31] Paszke A *et al* 2017 Automatoic differentiation of PyTorch *31st Conf. on Neural Information Processing Systems (NIPS 2017)*,

[32] Scarselli F, Gori M, Tsoi A C, Hagenbuchner M and Monfardini G 2008 The graph neural network model *IEEE Trans. Neural Netw.* **20** 61–80

[33] Gori M, Monfardini G and Scarselli F 2005 A new model for learning in graph domains. In Proc.. *2005 IEEE Int. Conf. on Neural Networks, 2005* IEEE vol 2 pp 729–734

[34] ALMEIDA L B 1987 A learning rule for asynchronous perceptrons with feedback in a combinatorial environment *Proc. of the IEEE 1st International Conference on Neural Networks, 1987* pp 609–618

[35] Pineda F J 1987 Generalization of back-propagation to recurrent neural networks *Phys. Rev. Lett.* **59** 2229

[36] Yujia Li, Tarlow D, Brockschmidt M and Zemel R 2015 *Gated Graph Sequence Neural Networks* Gated Graph Sequence Neural Networks arXiv:1511.05493

[37] Kearnes S, McCloskey K, Berndl M, Pande V and Riley P 2016 Molecular graph convolutions: moving beyond fingerprints *J. Computer-Aided Mol. Design* **30** 595–608

[38] Battaglia P W, Pascanu R, Lai M, Rezende D and Kavukcuoglu K 2016 Interaction networks for learning about objects, relations and physics arXiv:1612.00222

[39] Sanchez-Gonzalez A, Heess N, Springenberg J T, Merel J, Riedmiller M, Hadsell R, and Battaglia P 2018 Graph networks as learnable physics engines for inference and control arXiv:1806.01242

[40] Yunzhu Li, Jiajun W, Tedrake R, Tenenbaum J B, and Torralba A 2018 Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids arXiv:1810.01566

[41] Sanchez-Gonzalez A, Godwin J, Pfaff T, Ying R, Leskovec J, and Battaglia P W 2020 Learning to simulate complex physics with graph networks arXiv:2002.09405

[42] Bruna J, Zaremba W, Szlam A, and Yann L 2013 Spectral networks and locally connected networks on graphs arXiv:1312.6203

[43] Defferrard Mel, Bresson X, and Vandergheynst P 2016 Convolutional neural networks on graphs with fast localized spectral filtering *Adv. Neural Inf. Process. Syst.* 3844–52

[44] Henaff M, Bruna J, and Yann L 2015 Deep convolutional networks on graph-structured data arXiv:1506.05163

[45] Ummenhofer B, Prantl L, Thuerey N and Koltun V 2020 Lagrangian fluid simulation with continuous convolutions *Int. Conf. on Learning Representations*

[46] Sanchez-Gonzalez A, Bapst V, Cranmer K, and Battaglia P 2019 Hamiltonian graph networks with ODE integrators arXiv:1909.12790

[47] Cranmer M, Greydanus S, Hoyer S, Battaglia P, Spergel D, and Shirley H 2020 Lagrangian neural networks arXiv:2003.04630

[48] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser Łukasz and Polosukhin I 2017 Attention is all you need *Advances in Neural Information Processing Systems* pp 5998–6008

[49] Kipf T N and Welling M 2016 Semi-supervised classification with graph convolutional networks arXiv:1609.02907

[50] Tanabashi M *et al* 2018 Review of Particle Physics *Phys. Rev.* D **98** 030001

[51] Sirunyan A M *et al* 2017 Particle-flow reconstruction and global event description with the CMS detector *JINST* **12** P10003

[52] Bruna J, Cho K, Cranmer K, Louppe G Henrion I and Brehmer J *et al* 2017 Neural message passing for jet physics *Deep Learning for Physical Sciences Workshop at the 31st Conf. on Neural Information Processing Systems (NIPS)*

[53] Komiske P T, Metodiev E M and Thaler J 2019 Energy flow networks: deep sets for particle jets *J. High Energy Phys.* **01** 121

[54] Huilin Q and Gouskos L 2020 ParticleNet: Jet tagging via particle clouds **101** 056019

[55] Moreno E A *et al* 2020 Jedi-net: a jet identification algorithm based on interaction networks *Eur. Phys. J.* C **80** 58

[56] Moreno E A, Nguyen T Q, Vlimant J-R, Cerri O, Newman H B, Periwal A, Spiropulu M, Duarte J M and Pierini M 2020 Interaction networks for the identification of boosted $h \to b\bar{b}$ decays **102** 012010

[57] Mikuni V and Canelli F 2020 ABCnet: An Attention-Based Method for Particle Tagging *Eur. Phys. J. Plus* **135** 463

[58] Bernreuther E, Finke T, Kahlhoefer F, Krämer M and Alexander M 2020 Casting a graph net to catch dark showers arXiv:2006.08639

[59] Wang Y, Sun Y, Liu Z, Sarma S E, Bronstein M M and Solomon J M 2018 *Dynamic Graph cnn for Learning on Point Clouds* arXiv:1801.07829

[60] Butter A, Kasieczka G, Plehn T and Russell M 2018 Deep-learned Top Tagging with a Lorentz Layer *SciPost Phys.* **5** 028

[61] CERN 2018 Performance of mass-decorrelated jet substructure observables for hadronic two-body decay tagging in ATLAS *Technical Report* No. ATL-PHYS-PUB-2018-014

[62] Zaheer M, Kottur S, Ravanbakhsh S, Poczos B, Salakhutdinov R R and Smola A J 2017 Deep Sets *Advances in Neural Information Processing Systems 30* eds I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan and R Garnett (Red Hook, NY: Curran Associates, Inc) 3391–3401 (http://papers.nips.cc/paper/6931-deep-sets.pdf)

[63] Veličković P, Cucurull G, Casanova A, Romero A Bengio Y and Pietro L 2018 Graph attention networks *Int. Conf. on Learning Representations*

[64] Choma N, Monti F, Gerhardt L, Palczewski T, Ronaghi Z, Prabhat W B, Bronstein M M, Klein S R and Bruna J 2018 *Graph Neural Networks for Icecube Signal Classification*

[65] Abdughani M, Ren J, Lei W and Yang J M 2019 Probing stop pair production at the LHC with graph neural networks *JHEP* **08** 055

[66] Ren J, Lei W and Yang J M 2020 Unveiling CP property of top-Higgs coupling with graph neural networks at the LHC *Phys. Lett.* B **802** 135198

[67] Abdughani M, Wang D, Lei W, Yang J M and Zhao J 2020 Probing triple Higgs coupling with machine learning at the LHC arXiv:2005.11086

[68] Bertolini D, Harris P, Low M and Tran N 2014 Pileup Per Particle Identification *JHEP* **10** 059

[69] Martinez J A, Cerri O, Pierini M, Spiropulu M and Vlimant J-R 2018 Pileup Mitigation at the Large Hadron Collider With Graph Neural Networks arXiv:1810.07988

[70] Qasim S R, Kieseler J, Iiyama Y and Pierini M 2019 Learning representations of irregular particle-detector geometry with distance-weighted graph networks *Eur. Phys. J.* C **79** 7

[71] Kieseler J 2020 Object condensation: one-stage grid-free multi-object reconstruction in physics detectors, graph and image data arXiv:2002.03605

[72] Badiali C, Di Bello F A, Frattari G, Gross E, Ippolito V, Kado M, and Shlomi J 2020 Efficiency parameterization with neural networks arXiv:2004.02665

[73] Farrell S *et al* 2018 Novel Deep Learning Methods for Track Reconstruction arXiv:1810.06111

[74] Xiangyang J *et al* 2020 Graph neural networks for particle reconstruction in high energy physics detectors *33rd Conf. on Neural Information Processing Systems* vol 3

[75] Waltenberger W 2011 Rave—a detector-independent toolkit to reconstruct vertices *IEEE Trans. Nucl. Sci.* **58** 434–44

[76] Serviansky H, Segol N, Shlomi J, Cranmer K, Gross E, Maron H and Lipman Y 2020 Set2graph: learning graphs from sets

[77] Johnson D D, Larochelle H, and Tarlow D 2020 Learning graph structure with a finite-state automaton layer arXiv:2007.04929

[78] Kitaev N, Kaiser Łukasz, and Levskaya A 2020 Reformer: the efficient transformer arXiv:2001.04451

[79] Wang S, Belinda Li, Khabsa M, Fang H, and Hao M 2020 Linformer: self-attention with linear complexity arXiv:2006.04768

[80] Yujia Li, Vinyals O, Dyer C, Pascanu R, and Battaglia P 2018 Learning deep generative models of graphs arXiv:1803.03324

[81] Nash C, Ganin Y, Eslami S M, and Battaglia P W 2020 Polygen: an autoregressive generative model of 3d meshes arXiv:2002.10880

[82] You J, Ying R, Ren X, Hamilton W L, and Leskovec J 2018 Graphrnn: Generating realistic graphs with deep auto-regressive models arXiv:1802.08773

[83] Grover A, Zweig A, and Ermon S 2018 Graphite: Iterative generative modeling of graphs arXiv:1803.10459

[84] Wang H, Wang J, Wang J, Zhao M, Zhang W, Zhang F, Xie X and Guo M 2018 GraphGAN: Graph representation learning with generative adversarial nets *32nd AAAI Conf. on Artificial Intelligence*

[85] Kipf T, Fetaya E, Wang K-C, Welling M, and Zemel R 2018 Neural relational inference for interacting systems arXiv:1802.04687

[86] Hamrick J B, Allen K R, Bapst V, Zhu T, McKee K R, Tenenbaum J B, and Battaglia P W 2018 Relational inductive bias for physical construction in humans and machines arXiv:1806.01203

[87] Bapst V, Sanchez-Gonzalez A, Doersch C, Stachenfeld K L, Kohli P, Battaglia P W, and Hamrick J B 2019 Structured agents for physical construction arXiv:1904.03177

[88] Ying Z, You J, Morris C, Ren X, Hamilton W and Leskovec J 2018 Hierarchical graph representation learning with differentiable pooling *Adv. Neural Inf. Process. Syst.* 4800–4810