The Journal of Chemical Physics

Predicting molecular properties with covariant compositional networks

Cite as: J. Chem. Phys. **148**, 241745 (2018); https://doi.org/10.1063/1.5024797
Submitted: 04 February 2018 • Accepted: 06 June 2018 • Published Online: 27 June 2018

Truong Son Hy, Shubhendu Trivedi, Horace Pan, et al.

AIP Publishing

Predicting molecular properties with covariant compositional networks

Truong Son Hy,[1] Shubhendu Trivedi,[2] Horace Pan,[1] Brandon M. Anderson,[1] and Risi Kondor[1,3,a)]

[1]*Department of Computer Science, The University of Chicago, Chicago, Illinois 60637-5418, USA*
[2]*Toyota Technological Institute at Chicago, Chicago, Illinois 60637-2803, USA*
[3]*Department of Statistics, The University of Chicago, Chicago, Illinois 60637-5418, USA*

(Received 4 February 2018; accepted 6 June 2018; published online 27 June 2018)

Density functional theory (DFT) is the most successful and widely used approach for computing the electronic structure of matter. However, for tasks involving large sets of candidate molecules, running DFT separately for every possible compound of interest is forbiddingly expensive. In this paper, we propose a neural network based machine learning algorithm which, assuming a sufficiently large training sample of actual DFT results, can instead *learn* to predict certain properties of molecules purely from their molecular graphs. Our algorithm is based on the recently proposed covariant compositional networks framework and involves tensor reduction operations that are covariant with respect to permutations of the atoms. This new approach avoids some of the representational limitations of other neural networks that are popular in learning from molecular graphs and yields promising results in numerical experiments on the Harvard Clean Energy Project and QM9 molecular datasets. *Published by AIP Publishing.* https://doi.org/10.1063/1.5024797

I. INTRODUCTION

Density functional theory[1] (DFT) is the workhorse of modern quantum chemistry due to its ability to calculate many properties of molecular systems with high accuracy. However, this accuracy comes at a significant computational cost, generally making DFT too costly for applications such as chemical search and drug screening, which may involve hundreds of thousands of compounds. Methods that help overcome this limitation could lead to rapid developments in biology, medicine, and materials engineering.

Recent advances in machine learning, specifically, deep learning,[2] combined with the appearance of large datasets of molecular properties obtained both experimentally and theoretically,[3–5] present an opportunity to *learn* to predict the properties of compounds from their chemical structure rather than computing them explicitly with DFT. A machine learning model could allow for rapid and accurate exploration of huge molecular spaces to find suitable candidates for a desired molecule.

Central to any machine learning technique is the choice of a suitable set of descriptors or *features* used to parametrize and describe the input data. A poor choice of features will limit the expressiveness of the learning architecture and make accurate predictions impossible. On the other hand, providing too many features may make training difficult, especially when training data is limited. Hence, there has been a significant amount of work on designing good features for molecular systems. Predictions of energetics based on molecular geometry have been explored extensively, using a variety of parametrizations.[6,7] This includes bond types and/or angles,[8] radial distance distributions,[9] the Coulomb matrix and related structures,[10–12] the Smooth Overlap of Atomic Positions (SOAP),[13,14] permutation-invariant distances,[15] symmetry functions for atomic positions,[16,17] moment tensor potentials,[18] and scattering networks.[19]

Recently, the problem of learning from the structure of chemical bonds alone, i.e., the molecular graph, has attracted a lot of interest, especially in the light of the appearance of a series of neural network architectures designed specifically for learning from graphs.[20–26] Much of the success of these architectures stems from their ability to pick up on structures in the graph at multiple different scales, while satisfying the crucial requirement that the output be invariant to permutations of the vertices (which, in molecular learning, correspond to atoms). However, as we will show, the specific way that most of these architectures ensure permutation invariance still limits their representational power.

In this paper, we propose a new type of neural network for learning from molecular graphs, based on the the idea of *covariant compositional networks* (CCNs), recently introduced in Ref. 27. Importantly, CCNs are based on explicitly decomposing compound objects (in our case, molecular graphs) into a hierarchy of subparts (subgraphs), offering a versatile framework that is ideally suited to capture the multiscale nature of molecular structures from functional groups through local structures to the overall shape. In a certain sense, the resulting models are the neural networks analog of coarse graining. In addition, CCNs offer a more nuanced take on permutation invariance than other graph learning algorithms; while the overall output of a CCN is still invariant to the permutation of identical atoms, internally, the activations of the network are *covariant* rather than

a)Electronic mail: risi@cs.uchicago.edu

invariant, allowing us to better preserve information about the relationships between atoms. We demonstrate the success of this architecture through experiments on benchmark datasets, including QM9[3] and the Harvard Clean Energy Project.[4]

II. LEARNING FROM MOLECULAR GRAPHS

This paper addresses the problem of predicting chemical properties directly from each compound's molecular graph, which we will denote as $\mathcal{G}$ (Fig. 1, left). As such, it is related to the sizable literature on learning from graphs. In the kernel machines domain, this includes algorithms based on random walks,[28–30] counting subgraphs,[31] spectral ideas,[32] label propagation schemes with hashing,[33,34] and even algebraic ideas.[35]

Recently, a sequence of neural network based approaches has also appeared, starting with Ref. 36. Some of the proposed graph learning architectures[21,22,37] directly seek inspiration from the type of classical convolutional neural networks (CNNs) that are used for image recognition.[38,39] These methods involve moving a filter across vertices to create feature representations of each vertex based on its local neighborhoods. Other notable studies on graph neural networks include Refs. 24 and 40–42. Very recently, Ref. 25 showed that many of these approaches can be seen as message passing schemes and coined the term *message passing neural networks* (MPNNs) to refer to them collectively.

Regardless of the specifics, the two major issues that graph learning methods need to grapple with are invariance to permutations and capturing structures at multiple different scales. Let A denote the adjacency matrix of $\mathcal{G}$, and suppose that we change the numbering of the vertices by applying a permutation σ. The adjacency matrix will then change to A', with

$$A'_{i,j} = A_{\sigma^{-1}(i),\sigma^{-1}(j)}.$$

However, topologically, A and A' still represent the same graph. Permutation invariance means that the representation $\phi(\mathcal{G})$ learned or implied by our graph learning algorithm must be invariant to these transformations. Naturally, in the case of molecules, invariance is restricted to permutations that map each atom to another atom of the same type.

The multiscale property is equally crucial for learning molecular properties. For example, in the case of a protein, an ideal graph learning algorithm would represent $\mathcal{G}$ in a manner that simultaneously captures structures at the level of individual atoms, functional groups, interactions between functional groups, subunits of the protein, and the protein's overall shape.

A. Compositional networks

The idea of representing complex objects in terms of a hierarchy of parts has a long history in machine learning.[43–48] We have recently introduced a general framework for encoding such part-based models in a special type of neural network, called *covariant compositional networks*.[27] In this paper, we show how the CCN formalism can be specialized to learning from graphs, specifically, the graphs of molecules. Our starting point is the following definition.

Definition 1. Let $\mathcal{G}$ be the graph of a molecule made up of n atoms $\{e_1, \ldots, e_n\}$. The ***compositional neural network (comp-net)*** *corresponding to $\mathcal{G}$ is a directed acyclic*

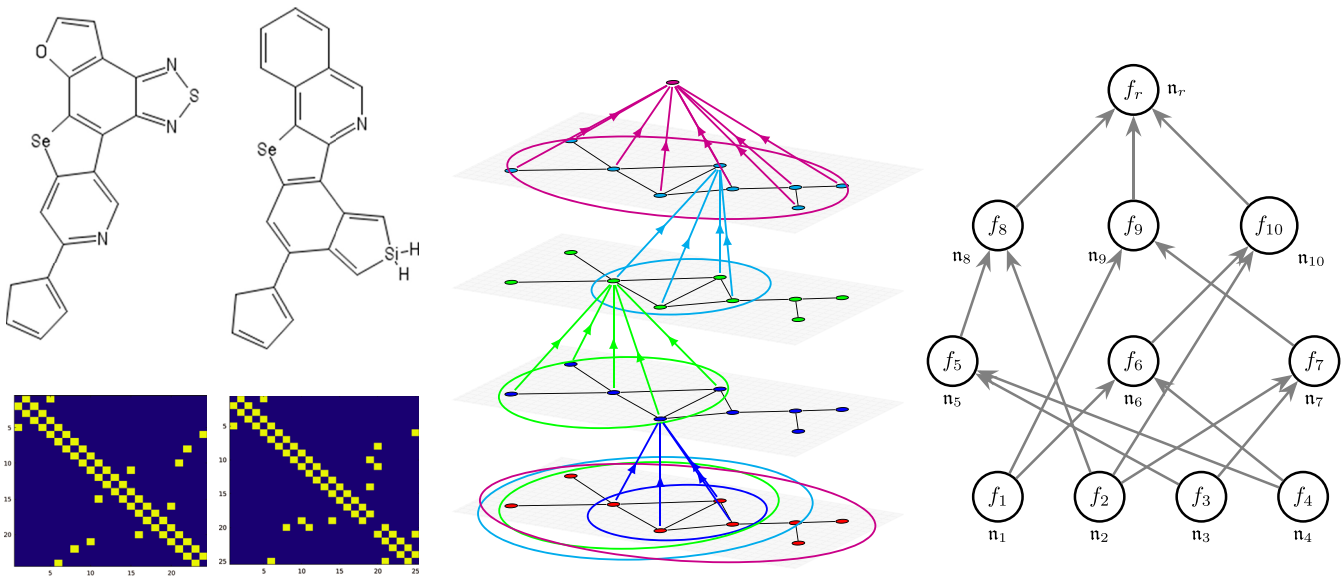

FIG. 1. **Left**: Molecular graphs for $C_{18}H_9N_3OSSe$ and $C_{22}H_{15}NSeSi$ from the Harvard Clean Energy Project (HCEP)[4] dataset with corresponding adjacency matrices. **Center and right**: The comp-net of a graph $\mathcal{G}$ is constructed by decomposing $\mathcal{G}$ into a hierarchy of subgraphs $\{\mathcal{P}_i\}$ and forming a neural network $\mathcal{N}$ in which each "neuron" $\mathfrak{n}_i$ corresponds to one of the $\mathcal{P}_i$ subgraphs and receives inputs from other neurons that correspond to smaller subgraphs contained in $\mathcal{P}_i$. The center pane shows how this can equivalently be thought of as an algorithm in which each vertex of $\mathcal{G}$ receives and aggregates messages from its neighbors. To keep the figure simple, we only marked aggregation at a single vertex in each round (layer).

graph (DAG) $\mathcal{N}$ in which each node (neuron) $\mathfrak{n}_i$ is associated with a subgraph $\mathcal{P}_i$ of $\mathcal{G}$ and carries an activation f_i. Moreover,

1. *If $\mathfrak{n}_i$ is a leaf node, then $\mathcal{P}_i$ is just a single vertex of $\mathcal{G}$, i.e., an atom $e_{\xi(i)}$, and the activation f_i is some initial label l_i. In the simplest case, f_i is a "one-hot" vector that identifies what type of atom resides at the given vertex.*
2. *$\mathcal{N}$ has a unique root node $\mathfrak{n}_r$ for which $\mathcal{P}_r = \mathcal{G}$, and the corresponding f_r represents the entire molecule.*
3. *For any two nodes $\mathfrak{n}_i$ and $\mathfrak{n}_j$, if $\mathfrak{n}_i$ is a descendant of $\mathfrak{n}_j$, then $\mathcal{P}_i$ is a subgraph of $\mathcal{P}_j$, and*

$$f_i = \Phi_i(f_{c_1}, f_{c_2}, \ldots, f_{c_k}),$$

where $f_{c_1}, \ldots, f_{c_k}$ denote the activations of the children of $\mathfrak{n}_i$. Here, Φ_i is called the aggregation function of node i.

Note that we now have two separate graphs: the molecular graph $\mathcal{G}$, and a corresponding comp-net $\mathcal{N}$ constructed according to Definition 1. One of the fundamental ideas of this paper is that $\mathcal{N}$ can be interpreted as a neural network, in which each node $\mathfrak{n}_i$ is a "neuron" that receives inputs $(f_{c_1}, f_{c_2}, \ldots, f_{c_k})$ and outputs the activation $f_i = \Phi_i(f_{c_1}, f_{c_2}, \ldots, f_{c_k})$ (Fig. 1, right). For now, we treat the activations as vectors but will soon generalize them to be tensors.

There is some freedom in how the system $\{\mathcal{P}_i\}$ of subgraphs is defined, but the default choice is $\left\{\mathcal{P}_i^\ell\right\}$, where $\mathcal{P}_i^\ell$ is the subgraph of vertices within a radius of ℓ of vertex i. In this case, $\mathcal{P}_i^0 = \{i\}$, $\mathcal{P}^\ell$ consist of the immediate neighbors of i, plus i itself, and so on. The aggregation function is discussed in detail in Sec. IV.

Conceptually, comp-nets are closely related to convolutional neural networks (CNNs), which are the mainstay of neural networks in computer vision. In particular,

1. Each neuron in a CNN only aggregates information from a small set of neurons from the previous layer, similar to how each node of a comp-net only aggregates information from its children.
2. The so-called *effective receptive fields* of the neurons in a CNN, i.e., the image patches for which each neuron is responsible for, form a hierarchy of nested sets similar to the hierarchy of $\{\mathcal{P}_i\}$ subgraphs.

In this sense, CNNs are a specific kind of compositional network, where the atoms are pixels. Because of this analogy, in the following, we will sometimes refer to $\mathcal{P}_i$ as the receptive field of neuron i, dropping the "effective" qualifier for brevity.

As mentioned above, an alternative popular framework for learning from graphs is message passing neural networks (MPNNs).[25] An MPNN operates in rounds: in each round $\ell = 1, \ldots, L$, every vertex collects the labels of its immediate neighbors, applies a nonlinear function Ψ, and updates its own label accordingly. From the neural networks point of view, the rounds correspond to layers and the labels correspond to the f_i^ℓ activations (Algorithm 1). More broadly, the classic Weisfeiler–Lehman test of isomorphism follows the same logic[49–51] and so does the related Weisfeiler–Lehman kernel, arguably the most successful kernel-based approach to graph learning.[33]

Algorithm 1. High level schematic of message passing type algorithms[25] in the notations of the present paper. Here l_i are the starting labels, and $\mathrm{ne}_1^i, \ldots, \mathrm{ne}_k^i$ denote the neighbors of vertex i.

```
for each vertex i
    f_i^0 ← l_i
for ℓ = 1 to L
    for each vertex i
        f_i^ℓ ← Ψ(f_{ne_1^i}^{ℓ-1}, ..., f_{ne_k^i}^{ℓ-1})
φ(G) ≡ f_r ← Ψ_r(f_1^L, ..., f_n^L)
```

In the above mentioned base case, when $\left\{\mathcal{P}_i^\ell\right\}$ is the collection of all subgraphs of $\mathcal{G}$ of radii $\ell = 0, 1, 2, \ldots$, a comp-net can also be thought of as a message passing algorithm: the messages received by vertex i in round ℓ are the activations $\{f_{u_j}^{\ell-1} | u_j \in B(i, 1)\}$, where $B(i, 1)$ is the ball of radius one centered at i (note that this contains not just the neighbors of i but also i itself). Conversely, MPNNs can be seen as comp-nets, where $\mathcal{P}_i^\ell$ is the subgraph defined by the receptive field of vertex i in round ℓ. A common feature of MPNNs, however, is that the Ψ aggregation functions that they employ are invariant to permuting their arguments. Most often, Ψ just *sums* all incoming messages and then applies a nonlinearity. This certainly guarantees that the final output of the network, $\phi(\mathcal{G}) = f_r$, will be permutation invariant. However, in Sec. III, we argue that it comes at the price of a significant loss of representational power.

III. COVARIANT COMPOSITIONAL NETWORKS

One of the messages of the present paper is that *invariant* aggregation functions, of the type popularized by message passing neural networks, are *not* the most general way to build compositional models for compound objects, such as graphs. To understand why this is the case, once again, an analogy with image recognition is helpful. Classical CNNs face two types of basic image transformations: translations and rotations. With respect to translations (barring pooling, edge effects, and other complications), CNNs behave in a quasi-invariant way, in the sense that if the input image is translated by an integer amount (t_x, t_y), the activations in each layer $\ell = 1, 2, \ldots, L$ translate the same way: the activation of neuron $\mathfrak{n}_{i,j}^\ell$ is simply transferred to neuron $\mathfrak{n}_{i+t_1,j+t_2}^\ell$, i.e., $f'^\ell_{i+t_1,j+t_2} = f_{i,j}^\ell$. This is the simplest manifestation of a well studied property of CNNs called *equivariance*.[52,53]

With respect to rotations, however, the situation is more complicated: if we rotate the input image by, e.g., 90°, not only the part of the image that fell in the receptive field of neuron $\mathfrak{n}_{i,j}^\ell$ will move to the receptive field of a different neuron $\mathfrak{n}_{-j,i}^\ell$ but also the orientation of the receptive field will change. For example, features which were previously picked up by horizontal filters will now be picked up by vertical filters. Therefore, in general, $f'^\ell_{-j,i} \neq f_{i,j}^\ell$ (Fig. 2).

It can be shown that one cannot construct a CNN for images that behaves in a quasi-invariant way with respect to

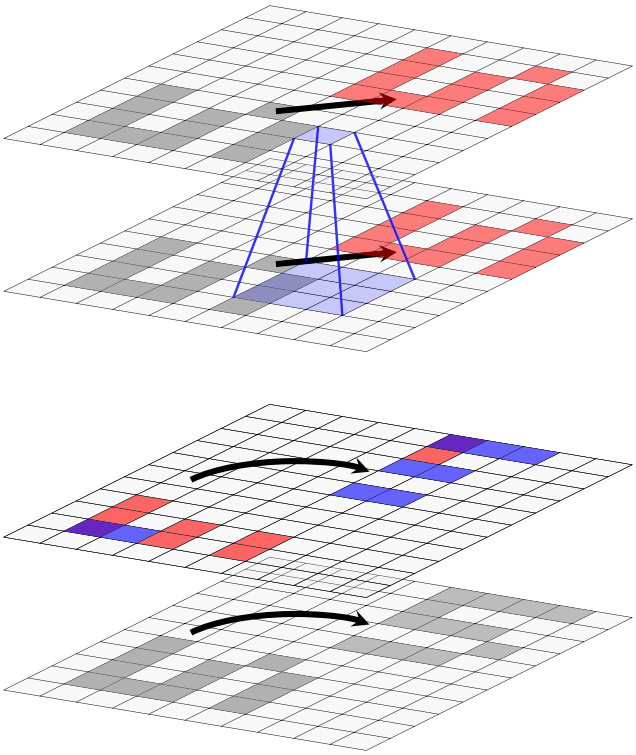

FIG. 2. In a convolutional neural network, if the input image is translated by some amount (t_1, t_2), what used to fall in the receptive field of neuron, $\mathfrak{n}^{\ell}_{i,j}$ is moved to the receptive field of $\mathfrak{n}^{\ell}_{i+t_1,j+t_2}$. Therefore, the activations transform in the very simple way $f'^{\ell}_{i+t_1,j+t_2} = f^{\ell}_{i,j}$. By contrast, rotations not only move the receptive fields around but also permute the neurons in the receptive field internally, so the activations transform in a more complicated manner. The right hand figure shows that if the CNN has a horizontal filter (blue) and a vertical one (red), then after a rotation by 90°, their activations are exchanged. In steerable CNNs, if (i,j) is moved to (i',j') by the transformation, then $f'^{\ell}_{i',j'} = R(f^{\ell}_{i,j})$, where R is some fixed linear function of the rotation angle.

both translations and rotations, unless every filter is directionless. It is, however, possible to construct a CNN in which the activations transform in a *predictable and reversible* way, $f'^{\ell}_{-j,i} = R(f^{\ell}_{i,j})$, for some fixed function R. This phenomenon is called *steerability* and has a significant literature in both classical signal processing[54–58] and the neural networks field.[59]

The situation in compositional networks is similar. The comp-net and message passing architectures that we have examined so far, by the virtue of the aggregation function being symmetric in its arguments, are all *quasi-invariant* (with respect to permutations) in the sense that if $\mathcal{N}$ and $\mathcal{N}'$ are two comp-nets for the same graph differing only in a reordering σ of the vertices of the underlying graph $\mathcal{G}$, and $\mathfrak{n}_i$ is a neuron in $\mathcal{N}$ while $\mathfrak{n}'_j$ is the corresponding neuron in $\mathcal{N}'$, then $f_i = f'_j$ for any permutation $\sigma \in \mathbb{S}_n$.

Quasi-invariance amounts to asserting that the activation f_i at any given node must only depend on $\mathcal{P}_i = \{e_{j_1}, \ldots, e_{j_k}\}$ as a *set* and not on the internal ordering of the atoms $e_{j_1}, \ldots, e_{j_k}$ making up the receptive field. At first sight, this seems desirable since it is exactly what we expect from the overall representation $\phi(G)$. On closer examination, however, we realize that this property is potentially problematic since it means that $\mathfrak{n}_i$ loses all information about which vertex in its receptive field has contributed what to the aggregate information f_i. In the CNN analogy, we can say that we have lost information about the *orientation* of the receptive field. In particular, if higher up in the network f_i is combined with some other feature vector f_j from a node with an overlapping receptive field, the aggregation process has no way of taking into account which parts of the information in f_i and f_j come from shared vertices and which parts do not (Fig. 3).

The solution is to regard the $\mathcal{P}_i$ receptive fields as *ordered sets* and explicitly establish how f_i *co-varies* with the internal ordering of the receptive fields. To emphasize that henceforth the $\mathcal{P}_i$ sets are ordered, we will use parentheses rather than braces to denote them.

*Definition 2. Assume that $\mathcal{N}$ is the comp-net of a graph $\mathcal{G}$ and $\mathcal{N}'$ is the comp-net of the same graph but after its vertices have been permuted by some permutation σ. Given any $\mathfrak{n}_i \in \mathcal{N}$ with receptive field $\mathcal{P}_i = (e_{p_1}, \ldots, e_{p_m})$, let $\mathfrak{n}'_j$ be the corresponding node in $\mathcal{N}'$ with receptive field $\mathcal{P}'_j = (e_{q_1}, \ldots, e_{q_m})$. Assume that $\pi \in \mathbb{S}_m$ is the permutation that aligns the orderings of the two receptive fields, i.e., for which $e_{q_{\pi(a)}} = e_{p_a}$. We say that the comp-nets are **covariant to permutations** if for any π, there is a corresponding function R_π such that $f'_j = R_\pi(f_i)$.*

To make the form of covariance prescribed by this definition more specific, we make the assumption that the $\{f \mapsto R_\pi(f)\}_{\pi \in \mathbb{S}_m}$ maps are *linear*. This allows us to treat them as matrices, $\{R_\pi\}_{\pi \in \mathbb{S}_n}$. Furthermore, linearity also implies that $\{R_\pi\}_{\pi \in \mathbb{S}_m}$ form a *representation* of $\mathbb{S}_m$ in the group theoretic sense of the word[60] (this notion of representation should not be confused with the neural networks sense of representations of objects, as "f_i is a representation of $\mathcal{P}_i$").

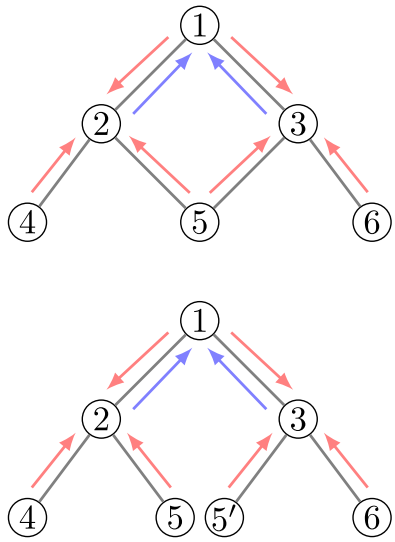

FIG. 3. These two graphs are not isomorphic, but after a single round of message passing (red arrows), the labels (activations) at vertices 2 and 3 will be identical in both graphs. Therefore, in the second round of message passing, vertex 1 will get the same messages in both graphs (blue arrows) and will have no way of distinguishing whether 5 and 5′ are the same vertex or not.

The representation theory of symmetric groups is a rich subject that goes beyond the scope of the present paper.[61] However, there is one particular representation of $\mathbb{S}_m$ that is likely familiar even to non-algebraists, the so-called *defining representation*, given by the $P_\pi \in \mathbb{R}^{n\times n}$ permutation matrices

$$[P_\pi]_{i,j} = \begin{cases} 1 & \text{if } \pi(j) = i, \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to verify that $P_{\pi_2\pi_1} = P_{\pi_2}P_{\pi_1}$ for any $\pi_1, \pi_2 \in \mathbb{S}_m$, so $\{P_\pi\}_{\pi\in\mathbb{S}_m}$ is indeed a representation of $\mathbb{S}_m$. If the transformation rules of the f_i activations in a given comp-net are dictated by this representation, then each f_i must necessarily be a $|\mathcal{P}_i|$ dimensional vector, and intuitively each component of f_i carries information related to one specific atom in the receptive field or the interaction of that specific atom with all the others collectively. We call this case first order permutation covariance.

Definition 3. We say that $\mathfrak{n}_i$ *is a* ***first order covariant node*** *in a comp-net if under the permutation of its receptive field* $\mathcal{P}_i$ *by any* $\pi \in \mathbb{S}_{|\mathcal{P}_i|}$ *its activation transforms as* $f_i \mapsto P_\pi f_i$.

If $\left(R_g\right)_{g\in\mathfrak{G}}$ is a representation of a group $\mathfrak{G}$, the matrices $\left(R_g \otimes R_g\right)_{g\in\mathfrak{G}}$ also form a representation. Thus, one step up in the hierarchy from P_π-covariant comp-nets is $P_\pi \otimes P_\pi$-covariant comp-nets, where the f_i feature vectors are now $|\mathcal{P}_i|^2$ dimensional vectors that transform under permutations of the internal ordering by π as $f_i \mapsto (P_\pi \otimes P_\pi)f_i$. If we reshape f_i into a matrix $F_i \in \mathbb{R}^{|\mathcal{P}_i|\times|\mathcal{P}_i|}$, then the action

$$F_i \mapsto P_\pi F_i P_\pi^\top$$

is equivalent to $P_\pi \otimes P_\pi$ acting on f_i. In the following, we will prefer this more intuitive matrix view since it makes it clear that feature vectors that transform this way express *relationships* between the different constituents of the receptive field. Note, in particular, that if we define $A{\downarrow_{\mathcal{P}_i}}$ as the restriction of the adjacency matrix to $\mathcal{P}_i$ (i.e., if $\mathcal{P}_i = \left(e_{p_1}, \ldots, e_{p_m}\right)$ then $[A{\downarrow_{\mathcal{P}_i}}]_{a,b} = A_{p_a,p_b}$), then $A{\downarrow_{\mathcal{P}_i}}$ transforms exactly as F_i does in the equation above.

Definition 4. We say that $\mathfrak{n}_i$ *is a* ***second order covariant node*** *in a comp-net if under the permutation of its receptive field* $\mathcal{P}_i$ *by any* $\pi \in \mathbb{S}_{|\mathcal{P}_i|}$ *its activation transforms as* $F_i \mapsto P_\pi F_i P_\pi^\top$.

Taking the pattern further, lets us define third, fourth, and general k'th order nodes, in which the activations are k'th order tensors, transforming under permutations as $F_i \mapsto F_i'$, where

$$[F_i']_{j_1,\ldots,j_k} = [P_\pi]_{j_1 j_1'} [P_\pi]_{j_2 j_2'} \ldots [P_\pi]_{j_k j_k'} [F_i]_{j_1',\ldots,j_k'}. \quad (1)$$

Here and in the following, for brevity, we use the Einstein summation convention, whereby any dummy index that appears twice on the right hand side of an equation is automatically summed over.

In general, we will call any quantity which transforms under permutations according to (1) a *k'th order P-tensor*. Saying that a given quantity is a P–tensor then not only means that it is representable by an $m \times m \times \cdots \times m$ array but also that this array transforms in a specific way under permutations.

Since scalars, vectors, and matrices can be considered 0th, 1st, and 2nd order tensors, the following definition covers both quasi-invariance and first and second order covariance as special cases. To unify notation and terminology, in the following, we will always talk about *feature tensors* rather than *feature vectors* and denote the activations as F_i rather than f_i.

Definition 5. We say that $\mathfrak{n}_i$ *is a* ***k'th order covariant node*** *in a comp-net if the corresponding activation* F_i *is a k'th order P–tensor, i.e., it transforms under permutations of* $\mathcal{P}_i$ *according to (1), or the activation is a sequence of d separate P–tensors* $F_i^{(1)}, \ldots, F_i^{(d)}$ *corresponding to d distinct channels.*

A **covariant compositional network (CCN)** is a comp-net in which each node's activation is covariant to permutations in the above sense. Hence we can talk about first, second, third, etc., order CCNs (CCN 1D, 2D, . . .) (Fig. 4).

The real significance of covariance, both here and in classical CNNs, is that while it allows for a richer internal representation of the data than fully invariant architectures, the final ouput of the network can still easily be made invariant. In covariant comp-nets, this is achieved by collapsing the input tensors of the root node $\mathfrak{n}_r$ at the top of the network into invariant scalars, for example, by computing their sums

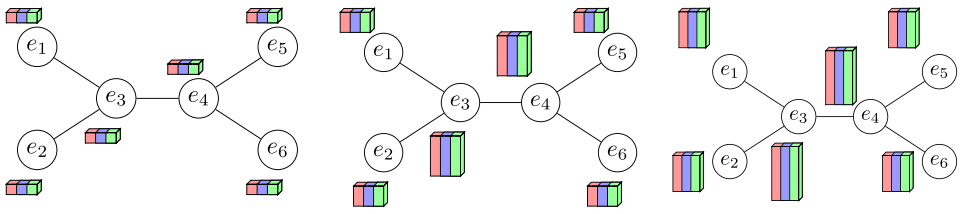

FIG. 4. Feature tensors in a first order CCN for ethylene (C_2H_4) assuming three channels (red, green, and blue). Vertices e_1, e_2, e_5, e_6 are hydrogen atoms, while vertices e_3, e_4 are carbons. Edge (e_3, e_4) is a double bond and the other edges are single bonds. (a) At the input layer, the receptive field for each atom is just the atom itself, so the feature matrix has size 1×3. (b) At level $\ell = 1$, the size of the receptive field of each atom grows depending on the local topology. The receptive fields for each hydrogen atom grow to include its neighboring carbon atom, resulting in a feature tensor of size 2×3; the receptive fields of the carbon atoms grow to include four atoms each and therefore have size 4×3. (c) At layer $\ell = 2$, the receptive fields of the carbons will include every atom in the molecule, while the receptive fields of the hydrogens will only be of size 4.

and traces (reducing them to zeroth order tensors) and outputting permutation invariant combinations of these scalars, such as their sum.

IV. COVARIANT AGGREGATION FUNCTIONS

It remains to explain how to define the Φ aggregation functions so as to guarantee covariance. Specifically, we show how to construct Φ such that if the $F_{c_1}, \ldots, F_{c_k}$ inputs of a given node $\mathfrak{n}_a$ at level ℓ are covariant k'th order P–tensors, then $F_a = \Phi(F_{c_1}, \ldots, F_{c_k})$ will also be a k'th order P–tensor. The aggregation function that we define consists of five operations executed in sequence: promotion, stacking, reduction, mixing, and an elementwise nonlinear transform. (See Fig. 5.) Practically relevant CCNs tend to have multiple channels, so each F_{c_i} is actually a sequence of d separate P–tensors $F_{c_i}^{(1)}, \ldots, F_{c_i}^{(d)}$. However, except for the mixing step, each channel behaves independently, so for simplicity, for now we drop the channel index.

A. Promotion

Each child tensor F_{c_i} captures information about a different receptive field $\mathcal{P}_{c_i}$, so before combining them we must "promote" each F_{c_i} to a $|\mathcal{P}_a| \times \cdots \times |\mathcal{P}_a|$ tensor $\widetilde{F}_{c_i}$, whose dimensions are indexed by the vertices of $\mathcal{P}_a$ rather than the vertices in each $\mathcal{P}_{c_i}$. Assuming that $\mathcal{P}_{c_i} = (e_{q_1}, \ldots, e_{q_{|\mathcal{P}_{c_i}|}})$ and $\mathcal{P}_a = (e_{p_1}, \ldots, e_{p_{|\mathcal{P}_a|}})$, this is done by defining a $|\mathcal{P}_a| \times |\mathcal{P}_{c_i}|$ indicator matrix

$$\chi_{i,j}^{c_i \to a} = \begin{cases} 1 & \text{if } q_j = p_i, \\ 0 & \text{otherwise} \end{cases}$$

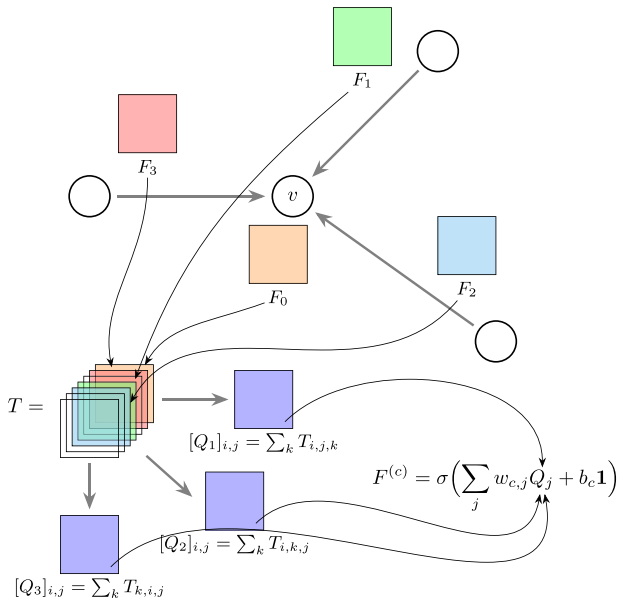

FIG. 5. Schematic of the aggregation process at a given vertex v of $\mathcal{G}$ in a second order CCN not involving tensor products with $A{\downarrow_{\mathcal{P}_b}}$ and assuming a single input channel. Feature tensors F_0, …, F_3 are collected from the neighbors of v as well as from v itself, promoted, and stacked to form a third order tensor T. In this example, we compute just three reductions Q_1, Q_2, Q_3. These are then combined with the $w_{c,j}$ weights and passed through the σ nonlinearity to form the output tensors $\left(F^{(1)}, F^{(2)}, F^{(3)}\right)$. For simplicity, in this figure, the "ℓ" and "a" indices are suppressed.

and setting

$$[\widetilde{F}_{c_i}]_{j_1,\ldots,j_k} = [\chi^{c_i \to a}]_{j_1 j_1'} [\chi^{c_i \to a}]_{j_2 j_2'} \cdots [\chi^{c_i \to a}]_{j_k j_k'} [F_{c_i}]_{j_1',\ldots,j_k'},$$

where, once again, Einstein summation is in use, so summation over $j_1', \ldots, j_k'$ is implied. Effectively, the promotion step aligns all the child tensors by permuting the indices of F_{c_i} to conform to the ordering of the atoms in $\mathcal{P}_a$ and padding with zeros where necessary.

B. Stacking

Now that the promoted tensors $\widetilde{F}_{c_1}, \ldots, \widetilde{F}_{c_s}$ all have the same shape, they can be stacked to form a $|\mathcal{P}_a| \times \cdots \times |\mathcal{P}_a|$ dimensional $k+1$'th order tensor T, with

$$T_{j_0,\ldots,j_k} = \begin{cases} [\widetilde{F}_{c_i}]_{j_0,\ldots,j_k} & \text{if } \mathcal{P}_{c_i} \text{ is the subgraph} \\ & \text{centered at } e_{p_{j_0}}, \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to see that T itself transforms as a P-tensor of order $(k+1)$.

We may also add additional information to T by taking its tensor product with another P-tensor. In particular, to explicitly add information about the local topology, we may tensor multiply T by $[A{\downarrow_{\mathcal{P}_a}}]_{i,j} = A_{e_{p_i}, e_{p_j}}$, the restriction of the adjacency matrix to $\mathcal{P}_a$. This will give an order $(k+3)$ tensor $S = T \otimes A{\downarrow_{\mathcal{P}_a}}$. Otherwise, we just set $S = T$. Note that in most other graph neural networks, the topology of the underlying graph is only accounted for implicitly, by the pattern in which different activations are combined. Being able to add the local adjacency matrix explicitly greatly extends the representational power of CCNs.

C. Reduction

Stacking and the optional tensor product increase the order of our tensors from k to $k+1$ or $k+3$. We reduce this to k again by a combination of generalized tensor contractions, more accurately called *tensor reductions*. For example, one way to drop the rank of S from $k+3$ to k is to sum out three of its indices,

$$Q_{i_1,\ldots,j_k} = \sum_{i_{\alpha_1}, i_{\alpha_2}, i_{\alpha_3}} S_{i_1,\ldots,j_k}.$$

Note that while this notation does not make it explicit, $i_{\alpha_1}, i_{\alpha_2}$ and i_{α_3} must be removed from amongst the indices of Q. Another way to drop the rank is to *contract* over three indices,

$$Q_{i_1,\ldots,j_k} = \sum_{i_{\alpha_1}, i_{\alpha_2}, i_{\alpha_3}} S_{i_1,\ldots,j_k} \delta_{i_{\alpha_1}, i_{\alpha_2}, i_{\alpha_3}},$$

where $\delta_{i_{\alpha_1}, i_{\alpha_2}, i_{\alpha_3}}$ is the generalized Kronecker symbol. A third way is to contract over two indices and sum over one index and so on. The crucial fact is that the result of each of these tensor operations is still a covariant P–tensor.

In general, the number of different ways that an order $k+q$ tensor S can be reduced to order k depends on both q and k. For example, when $k = 2$ and $q = 3$, there are 50 different possible tensor reductions (excluding diagonals). By contrast, when $q = 1$ (i.e., we are not multiplying by the adjacency

matrix), we only have $k+1$ possibilities, corresponding to summing S over each of its dimensions. No matter which case we are in, however, and how many contractions $Q_1, \ldots, Qj_{d'}$ our network actually computes (in our experiments using second order nodes, we compute 18 different ones), what is important is that the resulting order k tensors satify the P–tensor property.

D. Mixing with learnable weights

The reduction step can potentially produce quite a large number of order k tensors, $Q_1, \ldots, Q_r$. We reduce this number by linearly *mixing* $Q_1, \ldots, Q_r$, i.e., taking $d' < r$ linear combinations of the form

$$\widetilde{Q}^{(i)} = \sum_{j=1}^{r} w_{i,j}^{(\ell)} Q_j. \tag{2}$$

This is again a covariant operation, and the mixing weights are the actual learnable parameters of our neural network.

It is natural to interpret $\widetilde{Q}^{(1)}, \ldots, \widetilde{Q}^{(d')}$ as d' separate channels in neural network terminology. Recall that we also allow the input tensors to have multiple channels, but up to now the corresponding index has been suppressed. The mixing step is the point where these channels are all allowed to interact, so (2) becomes

$$\widetilde{Q}^{(c)} = \sum_{c',j} w_{c,c',j}^{(\ell)} Q_j^{(c')}, \tag{3}$$

and the learnable weights of the network at each level form a third order tensor $W_\ell = (w_{c,c',j}^{(\ell)})_{c,c',j}$. The channel index c is not to be confused with $c_1, \ldots, c_{j_k}$ denoting the children of node a. Equation (2) is the main mechanism whereby CCNs are able to learn increasigly complex topological features as we move up the network.

E. Nonlinearity

Finally, to get the actual activation of our neuron $\mathfrak{n}_a$, we add an additive bias $b_{\ell,c}$ and an elementwise nonlinearity σ (specifically, the ReLU operator $\sigma(x) = \max\{0, x\}$), as is standard in neural networks. Thus, ultimately, the output of the aggregation function is the collection of P–covariant tensors

$$F_a^{(c)} = \sigma\left[\sum_{c'=1}^{d} \sum_{j=1}^{r} w_{c,c',j}^{(\ell)} Q_j^{(c')} + b_{\ell,c}\, \mathbb{1} \right] \tag{4}$$

with $c \in \{1, 2, \ldots, d'\}$. As usual, in neural networks, the W_ℓ weight tensors are learned by backpropagation and some form of stochastic gradient descent.

V. EXPERIMENTS

We tested our CCN framework on three different types of datasets that involve learning the properties of molecules from their structure: (a) four relatively small datasets of small molecules that are standard benchmarks in the kernel literature (MUTAG, PTC, NCI1 and NCI109), (b) the Harvard Clean Energy Project,[4] and (c) QM9.[3] The first two types of datasets are pure graph classification/regression tasks. QM9 also has physical (spatial) features, which go somewhat beyond the scope of the present paper. Therefore, on QM9, we conducted separate experiments with and without these physical features [QM9(b) vs. QM9(a)].

In each case, we used second order CCNs (CCN2D) and included the tensor product with the restricted adjacency matrix, as described in Sec. IV B. However, for computational reasons, we only used 18 of the 50 possible contractions. The base features of each vertex were initialized with computed histogram alignment kernel features[63] of depth up to 10: each vertex receives a base label $l_i = \mathrm{concat}_{j=1}^{10} H_j(i)$ where $H_j(i) \in \mathbb{R}^d$ (with d being the total number of distinct discrete node labels) is the vector of relative frequencies of each label for the set of vertices at a distance equal to j from vertex i. The network was chosen to be three levels deep, with the number of output channels at each level fixed to 10.

To run our experiments, we used our own custom-built neural network library called GraphFlow.[64] Writing our own deep learning software became necessary because at the time we started work on the experiments, none of the standard frameworks such as TensorFlow,[65] PyTorch,[66] or MXNet[67] had efficient support for the type of tensor operations required by CCN. GraphFlow is a fast, general purpose deep learning library that offers automatic differentiation, dynamic computation graphs, multithreading, and Graphics Processing Unit (GPU) support for tensor contractions. In addition to CCN, it also implements other graph neural networks, including neural graph fingerprints,[21] PSCN,[37] and gated graph neural networks.[40] We also provide a reference implementation of CCN1D and CCN2D in PyTorch.[68]

In each experiment, we used 80% of the dataset for training, 10% for validation, and 10% for testing. For the kernel datasets, we performed the experiments on 10 separate training/validation/test stratified splits and averaged the resulting classification accuracies. Our training technique used mini-batch stochastic gradient descent with the Adam optimization method[69] and a batch size of 64. The initial learning rate was set to 0.001 and decayed linearly after each step toward a minimum of 10^{-6}.

A. Graph kernel datasets

Our first set of experiments involved three standard "graph kernels" datasets: (1) MUTAG, which is a dataset of 188 mutagenic aromatic and heteroaromatic compounds,[70] (2) PTC, which consists of 344 chemical compounds that have been tested for positive or negative toxicity in lab rats,[71] and (3) NCI1 and NCI109, which have 4110 and 4127 compounds, respectively, each screened for activity against small cell lung cancer and ovarian cancer lines.[72] In each of these datasets, each molecule has a discrete label (i.e., toxic/non-toxic and aromatic/heteroaromatic) and the goal is to predict this label. We compare CCN 2D against the graph kernel results reported in Ref. 62 (C-SVM algorithm with the Weisfeiler–Lehman kernel,[33] Weisfeiler–Lehman edge kernel,[33] shortest paths graph kernel,[29] graphlet kernel,[31] and the multiscale Laplacian graph kernel[62]), neural graph fingerprints[21] (with up to 3 levels and a hidden size of 10), and the "patchy-SAN" convolutional algorithm (PSCN).[37] The results are presented in Table I.

TABLE I. Classification of results on the kernel datasets (accuracy ± standard deviation).

	MUTAG	PTC	NCI1	NCI109
Wesifeiler–Lehman kernel[33]	84.50 ± 2.16	59.97 ± 1.60	84.76 ± 0.32	85.12 ± 0.29
Wesifeiler–Lehman edge kernel[33]	82.94 ± 2.33	60.18 ± 2.19	84.65 ± 0.25	85.32 ± 0.34
Shortest path kernel[29]	85.50 ± 2.50	59.53 ± 1.71	73.61 ± 0.36	73.23 ± 0.26
Graphlets kernel[31]	82.44 ± 1.29	55.88 ± 0.31	62.40 ± 0.27	62.35 ± 0.28
Random walk kernel[28]	80.33 ± 1.35	59.85 ± 0.95	Timed out	Timed out
Multiscale Laplacian graph kernel[62]	87.94 ± 1.61	63.26 ± 1.48	81.75 ± 0.24	81.31 ± 0.22
PSCN(k = 10)[37]	88.95 ± 4.37	62.29 ± 5.68	76.34 ± 1.68	N/A
Neural graph fingerprints[21]	89.00 ± 7.00	57.85 ± 3.36	62.21 ± 4.72	56.11 ± 4.31
Second order CCN (our method)	91.64 ± 7.24	70.62 ± 7.04	76.27 ± 4.13	75.54 ± 3.36

B. Harvard clean energy project

The Harvard Clean Energy Project (HCEP)[4] dataset consists of 2.3×10^6 organic compounds that are candidates for use in solar cells. The inputs are molecular graphs (derived from their SMILES strings), and the regression target is the power conversion efficiency (PCE). The experiments were ran on a random sample of 50 000 molecules.

On this dataset, we compared CCN to the following algorithms: Lasso, ridge regression, random forests, Gradient Boosted Trees (GBT), the optimal assignment Wesifeiler–Lehman graph kernel,[33] neural graph fingerprints,[21] and PSCN.[37] For the first four of these baseline methods, we created simple feature vectors from each molecule: the number of bonds of each type (i.e., number of H—H bonds, number of C—O bonds, etc.) and the number of atoms of each type. Molecular graph fingerprints use atom labels of each vertex as base features. For ridge regression and Lasso, we cross-validated over λ. For random forests and GBT, we used 400 trees and cross validated over maximum depth, minimum samples for a leaf, minimum samples to split a node, and learning rate (for GBT). For neural graph fingerprints, we used up to 3 layers and a hidden layer size of 10. In PSCN, we used a patch size of 10 with two convolutional layers and a dense layer on top as described in their paper (Tables II–IV).

C. QM9 dataset

QM9 has recently emerged as a molecular dataset of significant interest. QM9 contains ∼134 K organic molecules with up to nine atoms (C, H, O, N, and F) out of the GDB-17 universe of molecules. Each molecule is described by a SMILES string, which is converted to the molecular graph. The molecular properties are then calculated using DFT at the level of either B3LYP or 6-31G(2df, p), returning the spatial configurations of each atom along with thirteen molecular properties:

(a) U_0: atomization energy at 0 K (eV),
(b) U: atomization at room temperature (eV),
(c) H: enthalpy of atomization at room temperature (eV),
(d) G: free energy of atomization (eV),
(e) ω_1: highest fundamental vibrational frequency (cm^{-1}),
(f) ZPVE: zero point vibrational energy (eV),
(g) HOMO: highest occupied molecular orbital, energy of the highest occupied electronic state (eV),
(h) LUMO: lowest unoccupied molecular orbital, energy of the lowest unoccupied electronic state (eV),
(i) GAP: difference between HOMO and LUMO (eV),
(j) R^2: electronic spatial extent ($bohr^2$),
(k) μ: norm of the dipole moment (D),
(l) α: norm of the static polarizability ($bohr^3$),
(m) C_v: heat capacity at room temperature (cal/mol/K).

We performed two experiments on the QM9 dataset, with the goal of providing a benchmark of CCN as a graph learning

TABLE II. HCEP regression results. Error of predicting power conversion efficiency in units of percent. (Best results indicated in bold.)

	Test MAE	Test RMSE
Lasso	0.867	1.437
Ridge regression	0.854	1.376
Random forest	1.004	1.799
Gradient boosted trees	0.704	1.005
Weisfeiler–Lehman kernel[33]	0.805	1.096
Neural graph fingerprints[21]	0.851	1.177
PSCN (k = 10)[37]	0.718	0.973
Second order CCN (our method)	**0.340**	**0.449**

TABLE III. QM9(a) regression results (mean absolute error). Here we have only used the graph as the learning input without any physical features. (Best results indicated in bold.)

	WLGK	NGF	PSCN	CCN 2D
α ($bohr^3$)	3.75	3.51	1.63	**1.30**
C_v [cal/(mol K)]	2.39	1.91	1.09	**0.93**
G (eV)	4.84	4.36	3.13	**2.75**
GAP (eV)	0.92	0.86	0.77	**0.69**
H (eV)	5.45	4.92	3.56	**3.14**
HOMO (eV)	0.38	0.34	0.30	**0.23**
LUMO (eV)	0.89	0.82	0.75	**0.67**
μ (D)	1.03	0.94	0.81	**0.72**
ω_1 (cm^{-1})	192.16	168.14	152.13	**120.10**
R_2 ($bohr^2$)	154.25	137.43	61.70	**53.28**
U (eV)	5.41	4.89	3.54	**3.02**
U_0 (eV)	5.36	4.85	3.50	**2.99**
ZPVE (eV)	0.51	0.45	0.38	**0.35**

TABLE IV. The mean absolute error of CCN compared to the DFT error when using the complete set of physical features used in Ref. 25 in addition to the graph of each molecule. (Results below DFT error indicated in bold.)

	CCN	DFT error
α (bohr^3)	**0.22**	0.4
C_v [cal/(mol K)]	**0.07**	0.34
G (eV)	**0.06**	0.1
GAP (eV)	**0.12**	1.2
H (eV)	**0.06**	0.1
HOMO (eV)	**0.09**	2.0
LUMO (eV)	**0.09**	2.6
μ (D)	0.48	0.1
ω_1 (cm^{-1})	**2.81**	28
R_2 (bohr^2)	4.00	...
U (eV)	**0.06**	0.1
U_0 (eV)	**0.05**	0.1
ZPVE (eV)	**0.0039**	0.0097

framework and demonstrating that our framework can predict molecular properties to the same level as DFT. In both cases, we trained our system on each of the thirteen target properties of QM9 independently and report the MAE for each.

1. QM9(a)

We use the QM9 dataset to benchmark the CCN architecture against the Weisfeiler–Lehman graph kernel, neural graph fingerprints, and PSCN. For this test, we consider only heavy atoms and exclude hydrogen. The CCN architecture is as described above, and settings for NGF and PSCN are as described for HCEP.

2. QM9(b)

To compare the DFT error, we performed a test of the QM9 dataset with each molecule including hydrogen atoms. We used both physical atomic information (vertex features) and bond information (edge features) including the atom type, atomic number, acceptor, donor, aromatic, hybridization, number of hydrogens, Euclidean distance, and Coulomb distance between pairs of atoms. All the information is encoded in a vectorized format. Our physical features were taken directly from the dataset used in Ref. 25 without any special feature engineering.

To include the edge features into our model along with the vertex features, we used the concept of line graphs from graph theory. We constructed the line graph for each molecular graph in such a way that an edge of the molecular graph corresponds to a vertex in its line graph, and if two edges in the molecular graph share a common vertex, then there is an edge between the two corresponding vertices in the line graph. (See Fig. 6.) The edge features become vertex features in the line graph. The inputs of our model contain both the molecular graph and its line graph. The feature vectors F_ℓ between the two graphs are merged at each level ℓ.

We present results for both the CCN 1D and CCN 2D architectures. For CCN 1D, our network is seven layers with 64 input channels; at the first and second layer, the number of

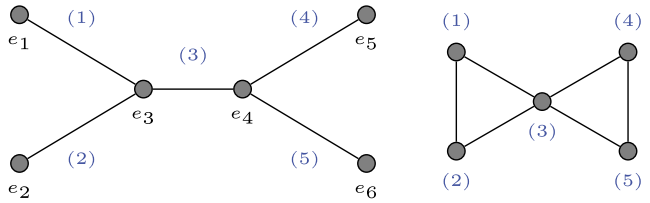

FIG. 6. Molecular graph of C_2H_4 (left) and its corresponding line graph (right). The vertices of the line graph correspond to edges of the molecular graph; two vertices of the line graph are connected by an edge if their corresponding edges on the molecular graph share a vertex.

channels is halved, and beyond that each layer has 16 channels. For the CCN 2D architecture, we used three layers, with 32 channels at the input and 16 at the remaining layers. We report the mean average error for each learning target in its corresponding physical unit and compare it against the DFT error given in Ref. 25.

D. Discussion

Overall, our CCN outperforms the other algorithms on a significant fraction of the experiments we implemented. On the subsampled HCEP dataset, CCN outperforms all other methods by a very large margin. For the graph kernel datasets, the SVM with the Weisfeiler–Lehman kernels achieve the highest accuracy on NCI1 and NCI109, while CCN wins on MUTAG and PTC. Perhaps this poor performance is to be expected since the datasets are small and neural networks usually require tens of thousands of training examples to be effective. Indeed, neural graph fingerprints and PSCN also perform poorly compared to the Weisfeiler–Lehman kernels. In the QM9(a) experiment, CCN obtains better results than the three other graph learning algorithms on all 13 learning targets.

In the QM9(b) experiment, the error of CCN is smaller than that of DFT itself on 11 of 12 learning targets (Ref. 25 does not have DFT errors for R2). However, other recent studies[14,25] have obtained even stronger results. Looking at our results, we find that values depending strongly on position information, such as the dipole moment and average electronic spatial extent, are predicted poorly when we include physical features. By contrast, properties that are not expected to strongly depend on spatial extent are predicted significantly better. This suggests that our spatial input features were not fully exploited and that feature engineering position information could significantly enhance the power of our CCN.

Our custom deep learning library[64] enabled all the above results to be obtained reasonably efficiently. The prediction time for CCN 1D and CCN 2D on QM9(b) comes out to 6.0 ms/molecule and 7.2 ms/molecule, respectively, making it possible to search through a million candidate molecules in less than 2 h.

VI. CONCLUSIONS

In this paper, we presented a general framework called covariant compositional networks (CCNs) for learning the properties of molecules from their graphs. Central to this framework are two key ideas: (1) a compositional structure

that generalizes message passing neural networks (MPNNs) and (2) the concept of covariant aggregation functions based on tensor algebra.

We argue that CCNs can extract multiscale structures from molecular graphs and keep track of the local topology in a manner the MPNNs are not able to. We also introduced the GraphFlow software library that provides an efficient implementation of CCNs. Using GraphFlow, we were able to show that CCNs often outperform existing state-of-the-art algorithms in learning molecular properties.

ACKNOWLEDGMENTS

The authors would like to thank the Institute for Pure and Applied Mathematics and the participants of its "Understanding Many Particle Systems with Machine Learning" program for inspiring this research. Our work was supported by DARPA Young Faculty Award No. D16AP00112 and used computing resources provided by the University of Chicago Research Computing Center.

[1] P. Hohenberg and W. Kohn, Phys. Rev. **136**, B864 (1964).
[2] Y. LeCun, Y. Bengio, and G. Hinton, Nature **521**, 436–444 (2015).
[3] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, Sci. Data **1**, 140022 (2014).
[4] J. Hachmann, R. Olivares-Amaya, S. Atahan-Evrenk, C. Amador-Bedolla, R. S. Sanchez-Carrera, A. Gold-Parker, L. Vogt, A. M. Brockway, and A. Aspuru-Guzik, J. Phys. Chem. Lett. **2**, 2241–2251 (2011).
[5] S. Kirklin, J. E. Saal, B. Meredig, A. Thompson, J. W. Doak, M. Aykol, S. Rühl, and C. Wolverton, npj Comput. Mater. **1**, 15010 (2015).
[6] K. Hansen, F. Biegler, R. Ramakrishnan, W. Pronobis, O. A. von Lilienfeld, K.-R. Müller, and A. Tkatchenko, J. Phys. Chem. Lett. **6**, 2326 (2015).
[7] B. Huang and O. A. von Lilienfeld, J. Chem. Phys. **145**, 161102 (2016).
[8] F. A. Faber, L. Hutchison, B. Huang, J. Gilmer, S. S. Schoenholz, G. E. Dahl, O. Vinyals, S. Kearnes, P. F. Riley, and O. A. von Lilienfeld, J. Chem. Theory Comput. **13**, 5255–5264 (2017).
[9] O. A. von Lilienfeld, R. Ramakrishnan, M. Rupp, and A. Knoll, Int. J. Quantum Chem. **115**, 1084 (2015).
[10] M. Rupp, A. Tkatchenko, K. R. Müller, and O. A. von Lilienfeld, Phys. Rev. Lett. **108**, 058301 (2012).
[11] K. Hansen, G. Montavon, F. Biegler, S. Fazli, M. Rupp, M. Scheffler, O. A. von Lilienfeld, A. Tkatchenko, and K.-R. Müller, J. Chem. Theory Comput. **9**, 3404 (2013).
[12] G. Montavon, M. Rupp, V. Gobre, A. Vazquez-Mayagoitia, K. Hansen, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, New J. Phys. **15**, 095003 (2013).
[13] A. P. Bartók, R. Kondor, and G. Csányi, Phys. Rev. B **87**, 184115 (2013).
[14] A. P. Bartók, S. De, C. Poelking, N. Bernstein, J. R. Kermode, G. Csányi, and M. Ceriotti, Sci. Adv. **3**, e1701816 (2017).
[15] G. Ferré, J.-B. Maillet, and G. Stoltz, J. Chem. Phys. **143**, 104114 (2015).
[16] J. Behler and M. Parrinello, Phys. Rev. Lett. **98**, 146401 (2007).
[17] J. Behler, J. Chem. Phys. **134**, 074106 (2011).
[18] A. V. Shapeev, Multiscale Model. Simul. **14**, 1153 (2016).
[19] M. Hirn, S. Mallat, and N. Poilvert, Multiscale Model. Simul. **15**, 827 (2017).
[20] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, in *Proceedings of International Conference on Learning Representations (ICLR)* (PMLR, 2014), Vol. 3.
[21] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, in *Advances in Neural Information Processing Systems* (NIPS, 2015), Vol. 28, p. 2224.
[22] S. Kearns, K. McCloskey, M. Brendl, V. Pande, and P. Riley, J. Comput.-Aided Mol. Des. **30**, 595 (2016).
[23] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, IEEE Signal Process. Mag. **34**, 18 (2017).
[24] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, Nat. Commun. **8**, 13890 (2017).
[25] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, in *Proceedings of International Conference on Machine Learning (ICML)* (PMLR, 2017), Vol. 70.
[26] K. Schütt, P.-J. Kindermans, H. E. Sauceda Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller, in Proceedings of NIPS, 2017.
[27] R. Kondor, T. S. Hy, H. Pan, S. Trivedi, and B. M. Anderson, in Proceedings of International Conference on Machine Learning (ICLR), 2018.
[28] T. Gärtner, in NIPS 2002 Workshop on Unreal Data, 2002.
[29] K. M. Borgwardt and H. P. Kriegel, in *Proceedings of IEEE International Conference on Data Mining* (IEEE, 2005), Vol. 5, p. 74.
[30] A. Feragen, N. Kasenburg, J. Peterson, M. de Bruijne, and K. M. Borgwardt, in *Advances in Neural Information Processing Systems* (NIPS, 2013), Vol. 26.
[31] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt, in *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)* (PMLR, 2009), Vol. 12, p. 488.
[32] S. V. N. Vishwanathan, N. N. Schraudolf, R. Kondor, and K. M. Bogwardt, J. Mach. Learn. Res. **11**, 1201 (2010).
[33] N. Shervashidze, P. Schweitzer, E. J. van Leeuwan, K. Mehlhorn, and K. M. Borgwardt, J. Mach. Learn. Res. **12**, 2539 (2011).
[34] M. Neumann, R. Garnett, C. Baukhage, and K. Kersting, Mach. Learn. **102**, 209 (2016).
[35] R. Kondor and K. M. Borgwardt, in *Proceedings of International Conference on Machine Learning (ICML)* (PMLR, 2008), Vol. 25, p. 496.
[36] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, IEEE Trans. Neural Networks **20**, 61 (2009).
[37] M. Niepert, M. Ahmed, and K. Kutzkov, in *Proceedings of International Conference on Machine Learning (ICML)* (PMLR, 2016), Vol. 33, p. 2014.
[38] Y. LeCun, Y. Bengio, and P. Haffner, Proc. IEEE **86**, 2278 (1998).
[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton, in *Advances in Neural Information Processing Systems* (NIPS, 2012), Vol. 25, p. 1097.
[40] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, in *Proceedings of International Conference on Learning Representations* (PMLR, 2016), Vol. 4.
[41] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, and K. Kavukcuoglu, in *Advances in Neural Information Processing Systems* (NIPS, 2016), Vol. 29, p. 4502.
[42] T. N. Kipf and M. Welling, in *Proceedings of International Conference on Learning Representations* (PMLR, 2017), Vol. 5.
[43] M. Fischler and R. Elschlager, IEEE Trans. Comput. **C-22**, 67 (1973).
[44] Y. Ohta, T. Kanade, and T. Sakai, in *Proceedings of IJCPR* (Institute of Electrical and Electronics Engineers, 1978), Vol. 4, p. 752.
[45] Z. W. Tu, X. R. Chen, A. L. Yuille, and S. C. Zhu, Int. J. Comput. Vision **63**, 113 (2005).
[46] P. F. Felzenszwalb and D. P. Huttenlocher, Int. J. Comput. Vision **61**, 55 (2005).
[47] S. Zhu and D. Mumford, Found. Trends Comput. Graphics Vision **2**, 259 (2006).
[48] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, IEEE Trans. Pattern Anal. Mach. Intell. **32**, 1627 (2010).
[49] B. Weisfeiler and A. A. Lehman, Nauchno-Tekh. Inf. **2**(9), 12 (1968).
[50] R. C. Read and D. G. Corneil, J. Graph Theory **1**, 339 (1977).
[51] J. Y. Cai, M. Furer, and N. Immerman, Combinatorica **12**, 389 (1992).
[52] T. Cohen and M. Welling, in *Proceedings of International Conference on Machine Learning (ICML)* (PMLR, 2016), Vol. 33, p. 2990.
[53] D. E. Worrall, S. Garbin, D. Turmukhambetov, and G. J. Brostow, in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (IEEE, 2017).
[54] W. T. Freeman and E. H. Adelson, IEEE Trans. Pattern Anal. Mach. Intell. **13**, 891 (1991).
[55] E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger, IEEE Trans. Inf. Theory **38**, 587 (1992).
[56] P. Perona, IEEE Trans. Pattern Anal. Mach. Intell. **17**, 488 (1995).
[57] P. C. Teo and Y. Hel-Or, Pattern Recognit. Lett. **19**, 7 (1998).
[58] R. Manduchi, P. Perona, and D. Shy, IEEE Trans. Signal Process. **46**, 1168 (1998).
[59] T. Cohen and M. Welling, in *Proceedings of International Conference on Learning Representations (ICLR)* (PMLR, 2017), Vol. 5.
[60] J.-P. Serre, *Linear Representations of Finite Groups*, Graduate Texts in Mathamatics (Springer-Verlag, 1977), Vol. 42.
[61] B. E. Sagan, *The Symmetric Group*, Graduate Texts in Mathamatics (Springer, 2001).
[62] R. Kondor and H. Pan, in *Advances in Neural Information Processing Systems* (NIPS, 2016), Vol. 29, p. 2982.
[63] N. M. Kriege, P. Giscard, and R. Wilson, *Advances in Neural Information Processing Systems* (NIPS, 2016), Vol. 20, p. 1623.

[64]T. S. Hy, "GraphFlow: A C++ deep learning framework," https://github.com/HyTruongSon/GraphFlow, 2017.
[65]M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.
[66]A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, in *Advances in Neural Information Processing Systems* (NIPS, 2017), Vol. 30.
[67]T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, in Conference and Workshop on Neural Information Processing Systems (NIPS), 2016.
[68]S. Trivedi, T. S. Hy, and H. Pan, "CCN in PyTorch," https://github.com/horacepan/CCN, 2017.
[69]D. P. Kingma and J. Ba, in Proceedings of International Conference on Learning Representations (ICLR), San Diego, 2015.
[70]A. K. Debnat, R. L. L. de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, J. Med. Chem. **34**, 786 (1991).
[71]H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, Bioinformatics **19**, 1183 (2003).
[72]N. Wale, I. A. Watson, and G. Karypis, Knowl. Inf. Syst. **14**, 347 (2008).