Covariant Compositional Networks for Learning Graphs

Truong Son Hy
University of Chicago
Chicago, IL, USA
hytruongson@uchicago.edu

Shubhendu Trivedi
Toyota Technological Institute at Chicago
Chicago, IL, USA
shubhendu@uchicago.edu

Horace Pan
University of Chicago
Chicago, IL, USA
hopan@uchicago.edu

Brandon M. Anderson
University of Chicago
Chicago, IL, USA
brandona@uchicago.edu

Risi Kondor
University of Chicago
Chicago, IL, USA
risi@uchicago.edu

ABSTRACT

In this paper, we propose Covariant Compositional Networks (CCNs), the state-of-the-art generalized convolution graph neural network for learning graphs. By applying higher-order representations and tensor contraction operations that are permutation-invariant with respect to the set of vertices, CCNs address the representation limitation of all existing neural networks for learning graphs in which permutation invariance is only obtained by summation of feature vectors coming from the neighbors for each vertex via well-known message passing scheme.

For an application of learning small-scale molecular graphs, we investigate the efficiency of CCNs in estimating Density Functional Theory (DFT) that is the most successful and widely used approach to compute the electronic structure of matter but significantly expensive in computation. We obtain a very promising result and outperform other state-of-the-art graph learning models in Harvard Clean Energy Project [1] and QM9 [2] molecular datasets.

KEYWORDS

graph neural networks, graph learning, quantum chemistry, network analysis

ACM Reference Format:
Truong Son Hy, Shubhendu Trivedi, Horace Pan, Brandon M. Anderson, and Risi Kondor. 2019. Covariant Compositional Networks for Learning Graphs. In *Anchorage '19: 15th International Workshop on Mining and Learning with Graphs, August 05, 2019, Anchorage, AK*. ACM, New York, NY, USA, 10 pages.

1 INTRODUCTION

In the field of Machine Learning, standard objects such as vectors, matrices, tensors were carefully studied and successfully applied into various areas including Computer Vision, Natural Language Processing, Speech Recognition, etc. However, none of these standard objects are efficient in capturing the structures of molecules, social networks or the World Wide Web which are not fixed in size. This arises the need of graph representation and extensions of Support Vector Machine and Convolution Neural Network to graphs.

To represent graphs in general and molecules specifically, the proposed models must be *permutation-invariant* and *rotation-invariant*. In addition, to apply kernel methods on graphs, the proposed kernels must be *positive semi-definite*. Many graph kernels and graph similarity functions have been introduced by researchers. Among them, one of the most successful and efficient is the Weisfeiler-Lehman graph kernel which aims to build a multi-level, hierarchical representation of a graph [?]. However, a limitation of kernel methods is quadratic space usage and quadratic time-complexity in the number of examples. The common idea of family of Weisfeiler-Lehman graph kernel is hashing the sub-structures of a graph. Extending this idea, we come to the simplest form of graph neural networks in which the *fixed* hashing function is replaced by a *learnable* one as a non-linearity mapping. In the context of graphs, the sub-structures can be considered as a set of vertex feature vectors. We ultilize the convolution operation by introducing higher-order representations for each vertex, from zeroth-order as a vector to the first-order as a matrix and the second-order as a 3rd order tensor. Also we introduce the notions of tensor contractions and tensor products to keep the orders of tensors manageable without exponentially growing. Our generalized convolution graph neural network is named as Covariant Compositional Networks (CCNs) [? ?]. We propose 2 specific algorithms that are first-order CCN in section 4 and second-order CCN in section 5.

2 COMPOSITIONAL NETWORKS

In this section, we introduce a general architecture called **compositional networks** (**comp-nets**) for representing complex objects as a combination of their parts and show that graph neural networks can be seen as special cases of this framework.

Definition 2.1. Let $\mathcal{G}$ be an object with n elementary parts (atoms) $\mathcal{E} = \{e_1, .., e_n\}$. A **compositional scheme** for $\mathcal{G}$ is a directed acyclic graph (DAG) $\mathcal{M}$ in which each node v is associated with some subset $\mathcal{P}_v$ of $\mathcal{E}$ (these subsets are called **parts** of $\mathcal{G}$) in such a way that:

(1) In the bottom level, there are exactly n leaf nodes in which each leaf node v is associated with an elementary atom e. Then $\mathcal{P}_v$ contains a single atom e.
(2) $\mathcal{M}$ has a unique root node v_r that corresponds to the entire set $\{e_1, .., e_n\}$.
(3) For any two nodes v and v', if v is a descendant of v', then $\mathcal{P}_v$ is a subset of $\mathcal{P}_{v'}$.

One can express message passing neural networks in this compositional framework. Consider a graph $G = (V, E)$ in an $L + 1$ layer network. The set of vertices V is also the set of elementary atoms $\mathcal{E}$. Each layer of the graph neural network (except the last) has one node denoted by v and one feature tensor denoted by f for each vertex of the graph G. The compositional network $\mathcal{N}$ is constructed as follows:

(1) In layer $\ell = 0$, each leaf node v_i^0 represents the single vertex $\mathcal{P}_i^0 = \{i\}$ for $i \in V$. The corresponding feature tensor f_i^0 is initialized by the vertex label l_i.
(2) In layers $\ell = 1, 2, .., L$, node v_i^ℓ is connected to all nodes from the previous level that are neighbors of i in G. The children of v_i^ℓ are $\{v_j^{\ell-1} | j : (i, j) \in E\}$. Thus, $\mathcal{P}_i^\ell = \bigcup_{j:(i,j)\in E} \mathcal{P}_j^{\ell-1}$. The feature tensor f_i^ℓ is computed as an aggregation of feature tensors in the previous layer:
$$f_i^\ell = \Phi(\{f_j^{\ell-1} | j \in \mathcal{P}_i^\ell\})$$
where Φ is some aggregation function.
(3) In layer $\ell = L + 1$, we have a single node v_r that represents the entire graph and collects information from all nodes at level $\ell = L$:
$$\mathcal{P}_r \equiv V$$
$$f_r = \Phi(\{f_i^L | i \in \mathcal{P}_r\})$$

In the following section, we will refer v as the **neuron**, and $\mathcal{P}$ and f as its corresponding **receptive field** and **activation**, respectively.

3 COVARIANCE

Standard message passing neural networks used summation or averaging operation as the aggregation function Φ of neighboring vertices' feature tensors. That would lead to loss of topological information. Therefore, we propose **permutation covariance** requirement for our neural activations f defined as follows.

Definition 3.1. For a graph G with the comp-net $\mathcal{N}$, and an isomorphic graph G' with comp-net $\mathcal{N}'$, let v be any neuron of $\mathcal{N}$ and v' be the corresponding neuron of $\mathcal{N}'$. Assume that $\mathcal{P}_v = (e_{p_1}, .., e_{p_m})$ while $\mathcal{P}_{v'} = (e_{q_1}, .., e_{q_m})$, and let $\pi \in \mathbb{S}_m$ be 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 $\mathcal{N}$ is **covariant to permutations** if for any π, there is a corresponding function R_π such that $f_{v'} = R_\pi(f_v)$.

The definition can be understood as permuting the vertices of graph G will change the activations of its vertices in a manner that is controlled by some fixed function R_π that depends on the permuation π.

4 FIRST ORDER MESSAGE PASSING

We will call the standard message passing as the **zero'th order message passing** in which each vertex is represented by a feature vector of length c (or c channels) which was not expressive enough to capture the structure of its local neighborhood. Hence, we propose **first order message passing** by representing each vertex v by a matrix: $f_v^\ell \in \mathbb{R}^{|\mathcal{P}_v^\ell| \times c}$, each row of this feature matrix corresponds to a vertex in the neighborhood of v.

Definition 4.1. We say that v is a **first order covariant node** in a comp-net if under the permutation of its receptive field $\mathcal{P}_v$ by any $\pi \in \mathbb{S}_{|\mathcal{P}_v|}$, its activation transforms as $f_v \mapsto P_\pi f_v$, where P_π is the permutation matrix:
$$[P_\pi]_{i,j} \triangleq \begin{cases} 1, & \pi(j) = i \\ 0, & otherwise \end{cases} \tag{1}$$
The transformed activation $f_{v'}$ will be:
$$[f_{v'}]_{a,s} = [f_v]_{\pi^{-1}(a),s}$$
where s is the channel index.

5 SECOND ORDER MESSAGE PASSING

Instead of representing a vertex with a feature matrix as done in first order message passing, we can represent it by a

3rd order tensor $f_v^\ell \in \mathbb{R}^{|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times c}$ and require these feature tensors to transform covariantly in a similar manner:

Definition 5.1. We say that v is a **second order covariant node** in a comp-net if under the permutation of its receptive field $\mathcal{P}_v$ by an $\pi \in \mathbb{S}_{|\mathcal{P}_v|}$, its activation transforms as $f_v \mapsto P_\pi f_v P_\pi^T$. The transformed activation $f_{v'}$ will be:

$$[f_{v'}]_{a,b,s} = [f_v]_{\pi^{-1}(a), \pi^{-1}(b), s}$$

where s is the channel index.

6 THIRD AND HIGHER ORDER MESSAGE PASSING

The similar pattern is applied further for third, forth, and general, k'th order nodes in the comp-net, in which the activations are k'th order tensors, transforming under permutations as $f_v \mapsto f_{v'}$:

$$[f_{v'}]_{i_1, i_2, .., i_k, s} = [f_v]_{\pi^{-1}(i_1), \pi^{-1}(i_2), .., \pi^{-1}(i_k), s} \qquad (2)$$

All but the channel index s (the last index) is permuted when we go from f_v to $f_{v'}$ after some permutation π of the receptive field $\mathcal{P}_v$. In general, we will call any quantiy which transforms according to this equation (ignoring the channel index) as a **k'th oder P-tensor**. Since scalars, vectors and matrices can be considered as zeroth, first and second order tensors, respectively, the following definition covers the previous definitions as special cases.

Definition 6.1. We say that v is a **k'th order covariant node** in a comp-net if the corresponding activation f_v is a k'th order P-tensor, i.e., it transforms under permutations of $\mathcal{P}_v$ according to 2.

7 TENSOR AGGREGATION RULES

The previous sections precribed how activations must transform in comp-nets of different orders. Tensor arithmetic provides a compact framework for deriving the general form of the permutation covariant operations. For convenience, we denote tensors as capital letters. Since the activation f is a tensor in general, we will denote it by capital F in the following sections. Recall the four basic operations that can be applied to tensors:

(1) The **tensor product** of $A \in \mathcal{T}^k$ with $B \in \mathcal{T}^p$ yields a tensor $C = A \otimes B \in \mathcal{T}^{p+k}$ where:

$$C_{i_1, i_2, .., i_{k+p}} = A_{i_1, i_2, .., i_k} B_{i_{k+1}, i_{k+2}, .., i_{k+p}}$$

(2) The **contraction** of $A \in \mathcal{T}^k$ along the pair of dimensions $\{a, b\}$ (assuming $a < b$) yields a $k-2$ order tensor:

$$C_{i_1, i_2, .., i_k} = \sum_j A_{i_1, .., i_{a-1}, j, i_{a+1}, .., i_{b-1}, j, i_{b+1}, .., k}$$

where we assume that i_a and i_b have been removed from the indices of C. Using Einstein notation, this can be written much more compactly as

$$C_{i_1, i_2, .., i_k} = A_{i_1, i_2, .., i_k} \delta^{i_a, i_b}$$

where δ^{i_a, i_b} is the diagonal tensor with $\delta^{i,j} = 1$ if $i = j$ and 0 otherwise. We also generalize contractions to (combinations of) larger sets of indices $\{\{a_1^1, .., a_{p_1}^1\}, \{a_1^2, .., a_{p_2}^2\}, .., \{a_1^q$ as the $(k - \sum_j p_j)$ order tensor:

$$C_{...} = A_{i_1, i_2, .., i_k} \delta^{a_1^1, .., a_{p_1}^1} \delta^{a_1^2, .., a_{p_2}^2} \dots \delta^{a_1^q, .., a_{p_q}^q}$$

(3) The **projection** of a tensor is defined as a special case of contraction:

$$A \downarrow_{a_1, .., a_p} = A_{i_1, i_2, .., i_k} \delta^{i_{a_1}} \delta^{i_{a_2}} \dots \delta^{i_{a_k}}$$

where projection of A among indices a_1, .., a_p is denoted as $A \downarrow_{a_1, .., a_p}$.

Proposition 7.1 shows that all of the above operations as well as linear combinations preserve permutation covariance property of P-tensors. Therefore, they can be combined within the aggregation function Φ.

Proposition 7.1. *Assume that A and B are k'th and p'th order P-tensors, respectively. Then:*

(1) *$A \otimes B$ is a $(k + p)$'th order P-tensors.*
(2) *$A_{i_1, i_2, .., i_k} \delta^{a_1^1, .., a_{p_1}^1} \dots \delta^{a_1^q, .., a_{p_q}^q}$ is a $(k - \sum_j p_j)$'th order P-tensor.*
(3) *If A_1, .., A_u are k'th order P-tensors and α_1, .., α_u are scalars, then $\sum_j \alpha_j A_j$ is a k'th order P-tensor.*

Propositions 7.2, 7.3 and 7.4 show that tensor promotion, concatenation and production preverse permutation covariance, and hence can be applied within Φ.

Proposition 7.2. *Assume that node v is a descendant of node v' in a comp-net $\mathcal{N}$. The corresponding receptive fields are $\mathcal{P}_v = (e_{p_1}, .., e_{p_m})$ and $\mathcal{P}_{v'} = (e_{q_1}, .., e_{q_{m'}})$. Remark that $\mathcal{P}_v \subseteq \mathcal{P}_{v'}$. Define $\chi^{v \to v'} \in \mathbb{R}^{m \times m'}$ as an indicator matrix:*

$$\chi_{i,j}^{v \to v'} = \begin{cases} 1, & q_j = p_i \\ 0, & otherwise \end{cases} \qquad (3)$$

*Assume that F_v is a k'th order P-tensors with respect to permutations $(e_{p_1}, .., e_{p_m})$. We have the **promoted** tensor:*

$$[F_{v \to v'}]_{i_1, .., i_k} = \chi_{i_1, j_1}^{v \to v'} \cdots \chi_{i_k, j_k}^{v \to v'} [F_v]_{j_1, .., j_k} \qquad (4)$$

is a k'th oder P-tensor with respect to permutations of $(e_{q_1}, .., e_{q_{m'}})$.

In equation 4, node v' promotes P-tensors from its children nodes v with respect to its own receptive field $\mathcal{P}_{v'}$ by the appropriate $\chi^{v \to v'}$ matrix such that all promoted tensors $F_{v \to v'}$ have the same size. Remark that promoted tensors are padded with zeros.

Proposition 7.3. *Let nodes $v_1, .., v_n$ be the children of v in a message passing type comp-net (the corresponding vertices of these nodes are in $\mathcal{P}_v$) with corresponding k'th order tensor activations $F_{v_1}, .., F_{v_n}$. Let*

$$[F_{v_t \to v}]_{i_1,..,i_k} = [\chi^{v_t \to v}]_{i_1,j_1} \cdots [\chi^{v_t \to v}]_{i_k,j_k} [F_{v_t}]_{j_1,..,j_k}$$

be the promoted tensors ($t \in \{1, .., n\}$). We ***concatenate*** *or* ***stack*** *them into a $(k+1)$'th order tensor:*

$$[\overline{F}_v]_{t,i_1,..,i_k} = [F_{v_t \to v}]_{i_1,..,i_k}$$

Then the concatenated tensor $\overline{F}_v$ is a $(k+1)$'th order P-tensor of v.

The restriction of the adjacency matrix to $\mathcal{P}_v$ is a second order P-tensor. Proposition 7.4 gives us a way to explicitly add topolocical information to the activation.

Proposition 7.4. *If F_v is a k'th order P-tensor at node v, and $A \downarrow_{\mathcal{P}_v}$ is the restriction of the adjacency matrix to $\mathcal{P}_v$, then $F_v \otimes A \downarrow_{\mathcal{P}_v}$ is a $(k+2)$'th order P-tensor.*

8 SECOND ORDER TENSOR AGGREGATION WITH THE ADJACENCY MATRIX

The first nontrivial tensor contraction case occurs when $F_{v_1 \to v}, .., F_{v_n \to v}$ are second order tensors, and we multiply with $A \downarrow_{\mathcal{P}_v}$, since in that case $\mathcal{T}$ is 5th order (6th order if we consider the channel index), and can be contracted down to second order in the following ways:

1. The **1+1+1** case contracts $\mathcal{T}$ in the form $\mathcal{T}_{i_1,i_2,i_3,i_4,i_5} \delta^{i_{a_1}} \delta^{i_{a_2}} \delta^{i_{a_3}}$, i.e., it projects $\mathcal{T}$ down along 3 of its 5 dimensions. This can be done in $\binom{5}{3} = 10$ ways.
2. The **1+2** case contracts $\mathcal{T}$ in the form $\mathcal{T}_{i_1,i_2,i_3,i_4,i_5} \delta^{i_{a_1}} \delta^{i_{a_2},i_{a_3}}$, i.e., it projects $\mathcal{T}$ along one dimension, and contracts it along two others. This can be done in $3\binom{5}{3} = 30$ ways.
3. The **3** case is a single 3-fold contraction $\mathcal{T}_{i_1,i_2,i_3,i_4,i_5} \delta^{i_{a_1},i_{a_2},i_{a_3}}$. This can be done in $\binom{5}{3} = 10$ ways.

Totally, we have 50 different contractions that result in 50 times more channels. In practice, we only implement 18 contractions for efficiency.

9 ARCHITECTURE

In this section, we will describe how our compositional architecture is a generalization of previous works with an extension to higher-order representations.

Recent works on graph neural networks [? ? ? ?] can all be seen as instances of *zeroth order message passing* where each vertex representation is a vector (1st order tensor) of c channels in which each channel is represented by a scalar (zeroth order P-tensor). This results in the loss of certain structural information during the message aggregation, and the network loses the ability to learn topological information of the graph's multiscale structure.

Our architecture represents generalized vertex representations with higher-order tensors which can retain this structural information. There is significant freedom in the choice of this tensor structure, and we now explore two examples, corresponding to the tensor structures, which we call "first order CCN" and "second order CCN", respectively.

We start with an input graph $G = (V, E)$ and construct a network with $L + 1$ levels, indexed from 0 (input level) to L (top level). Initially, each vertex v is associated with an input feature vector $l_v \in \mathbb{R}^c$ where c denotes the number of channels. The receptive field of vertex v at level ℓ is denoted by $\mathcal{P}_v^\ell$ and is defined recursively as follows:

$$\mathcal{P}_v^\ell \triangleq \begin{cases} \{v\}, & \ell = 0 \\ \bigcup_{(u,v) \in E} \mathcal{P}_u^\ell, & \ell = 1, \ldots, L \end{cases} \tag{5}$$

The vertex representation of vertex v at level ℓ is denoted by a feature tensor F_v^ℓ. In zeroth order message passing, $F_v^\ell \in \mathbb{R}^c$ is a vector of c channels. Let N be the number of vertices in $\mathcal{P}_v^\ell$. In the first order CCN, each vertex is represented by a matrix (second order tensor) $F_v^\ell \in \mathbb{R}^{N \times c}$ in which each row corresponds to a vertex in the receptive field $\mathcal{P}_v^\ell$, and each channel is represented by a vector (first order P-tensor) of size N. In the second order CCN, F_ℓ^v is promoted into a third order tensor of size $N \times N \times c$ in which each channel has a second order representation (second order P-tensor). In general, we can imagine a series of feature tensors of increasing order for higher order message passing. Note that the components corresponding to the channel index does not transform as a tensor, whereas the remaining indices do transform as a P-tensor. The tensor F_v^ℓ transforms in a *covariant* way with respect to the permutation of the vertices in the receptive field $\mathcal{P}_v^\ell$.

Now that we have established the structure of the high order representations of the vertices at each site, we turn to the task of constructing the aggregation function Φ from one level to another. The key to doing this in a way that preserves covariance is to "promote-stack-reduce" the tensors as one traverses the network at each level.

We start with the promotion step. Recall that we want to accumulate information at higher levels based upon the receptive field of a given vertex. However, it is clear that not all vertices in the receptive field have the same size tensors. To account for this, we use an index function χ that ensures all tensors are the same size by padding with zeros when necessary. At level ℓ, given two vertices v and w such that $\mathcal{P}_w^{\ell-1} \subseteq \mathcal{P}_v^\ell$, the permutation matrix $\chi_\ell^{w \to v}$ of size

$|\mathcal{P}_v^\ell| \times |\mathcal{P}_w^{\ell-1}|$ is defined as in Prop. 7.2. In CCN 1D & 2D, the resizing is done by (broadcast) matrix multiplication $\chi \cdot F_w^{\ell-1}$ and $\chi \times F_w^{\ell-1} \times \chi^T$ where $\chi = \chi_\ell^{w\to v}$, respectively. Denote the resized tensor as $F_{w\to v}^\ell$. (See step 7 in algorithm 2.) This promotion is done for all tensors of every vertex in the receptive field, and stacked/concatenated into a tensor one order higher. (See Prop. 7.3. Notice that the stacked index has the same size as the receptive field.) From here, as in CCN 2D, we can compute the *tensor product* of this higher order tensor with the restricted adjacency matrix (subject to the receptive field) and obtain an even higher order tensor. (See Prop. 7.4.) Finally, we can reduce the higher order tensor down to the expected size of the vertex representation using the tensor contractions discussed in Prop. 7.1.

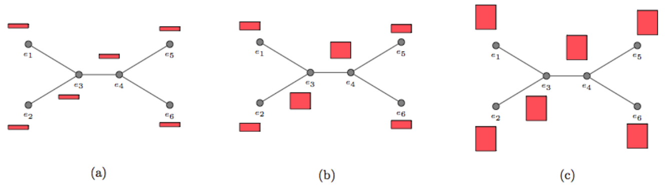
We include all possible tensor contractions, which introduces additional channels. To avoid an exponential explosion in the number of channels with deep networks, we use a learnable set of weights that reduces the number of channels to a fixed number c. These weights are learnable through backpropagation. To complete our construction of Φ, this tensor is passed through an element-wise nonlinear function Υ such as a ReLU to form the feature tensor for a given vertex at the next level. (See steps 4 and 9 in algorithm 2.)

Finally, at the output of the network, we again reduce the vertex representations F_v^ℓ into a vector of channels $\Theta(F_v^\ell) = F_v^\ell \downarrow_{i_1,..,i_p}$ where $i_1, .., i_p$ are the non-channel indices. (See Prop. 7.1.) We sum up all the reduced vertex representations of a graph to get a single vector which we use as the graph representation. This final graph representation can then be used for regression or classification with a fully connected layer. In addition, we can construct a richer graph representation by concatenating the shrunk representation at each level. (See steps 12, 13 and 14 in algorithm 2.)

The development of higher order CCNs require efficient tensor algorithms to successively train the network. A fundamental roadblock we face in implementing CCNs is that fifth or sixth order tensors are often too large to be held in memory. To address this challenge, we do not construct the tensor product explicitly. Instead we introduce a *virtual indexing system* for a *virtual tensor* that computes the elements of tensor only when needed given the indices. This allows us to implement the tensor contraction operations efficiently with GPUs on virtual tensors.

For example, consider the operations in step 8 in algorithm 2. This requires performing contractions over several indices on the two inputs $\mathcal{F} = \{F_{w\to v}^\ell | w \in \mathcal{P}_v^\ell\}$, in which $\mathcal{F}_{i_1} = F_{w_{i_1}\to v}^\ell$ is of size $|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times c$, and $\mathcal{A} = A \downarrow_{\mathcal{P}_v^\ell}$. One

Figure 1: CCN 1D on C_2H_4 molecular graph



naive strategy would be to stack all tensors in $\mathcal{F}$ into a new object and then directly compute the tensor product with $\mathcal{A}$ to form a sixth order tensor, given by a tuple of indices $(i_1, i_2, i_3, i_4, i_5, i_6)$. Instead, the corresponding tensor element is computed on-the-fly through simple multiplication:

$$\mathcal{T}_{i_1,i_2,i_3,i_4,i_5,i_6} = (\mathcal{F}_{i_1})_{i_2,i_3,i_6} \cdot \mathcal{A}_{i_4,i_5} \tag{6}$$

where i_6 is the channel index.

In our experiments of CCN 2D, we implement 18 different contractions (see Prop. 8) such that each contraction results in a $|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times c$ tensor. The result of step 8 is $\overline{F}_\ell^v$ with 18 times more channels.

Algorithm CCN 1D is described in pseudocode 1. Figure 1 shows a visualization of CCN 1D's tensors on C_2H_4 molecular graph. Vertices e_3 and e_4 are carbon (C) atoms, and vertices e_1, e_2, e_5 and e_6 are hydrogen (H) atoms. Edge (e_3, e_4) is a double bond (C, C) between two carbon atoms. All other edges are single bonds (C, H) between a carbon atom and a hydrogen atom. In the bottom layer $\ell = 0$, the receptive field of every atom e only contains itself, thus its representation F_e^0 is a tensor of size $1 \times c$ where c is the number of channels (see figure 1(a)). In the first layer $\ell = 1$, the receptive field of a hydrogen atom contains itself and the neighboring carbon atom (i.e., $\mathcal{P}_{e_1}^1 = \{e_1, e_3\}$), thus tensors for hydrogen atoms are of size $2 \times c$. Meanwhile, the receptive field of a carbon atom contains itself, the another carbon and two other neighboring hydrogens (i.e., $\mathcal{P}_{e_3}^1 = \{e_1, e_2, e_3, e_4\}$) and $\mathcal{P}_{e_4}^1 = \{e_3, e_4, e_5, e_6\}$), thus $F_{e_3}^1, F_{e_4}^1 \in \mathbb{R}^{4\times c}$ (see figure 1(b)). In all later layers denoted $\ell = \infty$, the receptive field of every atom contains the whole graph (in this case, 6 vertices in total), thus $F_e^\infty \in \mathbb{R}^{6\times c}$ (see figure 1(c)).

Algorithm CCN 2D is described in pseudocode 2. Figure 2 shows a visualization of CCN 2D's tensors on C_2H_4 molecular graph. In the bottom layer $\ell = 0$, $|\mathcal{P}_e^0| = 1$ and $F_e^0 \in \mathbb{R}^{1\times 1\times c}$ for every atom e (see figure 2(a)). In the first layer $\ell = 1$, $|\mathcal{P}_e^1| = 2$ and $F_e^1 \in \mathbb{R}^{2\times 2\times c}$ for hydrogen atom $e \in \{e_1, e_2, e_5, e_6\}$, and for carbon atoms $|\mathcal{P}_{e_3}^1| = |\mathcal{P}_{e_4}^1| = 4$ and

Algorithm 1: First-order CCN

```
1  **Input:** $G, l_v, L$
2  **Parameters:** Matrices $W_0 \in \mathbb{R}^{c \times c}$, $W_1, .., W_L \in \mathbb{R}^{(2c) \times c}$
   and biases $b_0, .., b_L$. For CCN 1D, we only implement 2
   tensor contractions.
3  $F_v^0 \leftarrow \Upsilon(W_0 l_v + b_0 \mathbb{1})$ $(\forall v \in V)$
4  Reshape $F_v^0$ to $1 \times c$ $(\forall v \in V)$
5  **for** $\ell = 1, .., L$ **do**
6      **for** $v \in V$ **do**
7          $F_{w \to v}^\ell \leftarrow \chi \times F_w^{\ell-1}$ where $\chi = \chi_{w \to v}^\ell$ $(\forall w \in \mathcal{P}_v^\ell)$
8          Concatenate the promoted tensors in
             $\{F_{w \to v}^\ell | w \in \mathcal{P}_v^\ell\}$ and apply 2 tensor
             contractions that results in $\overline{F}_v^\ell \in \mathbb{R}^{|\mathcal{P}_v^\ell| \times (2c)}$.
9          $F_v^\ell \leftarrow \Upsilon(\overline{F}_v^\ell \times W_\ell + b_\ell \mathbb{1})$
10     **end**
11 **end**
12 $F^\ell \leftarrow \sum_{v \in V} \Theta(F_v^\ell)$ $(\forall \ell)$
13 Graph feature $F \leftarrow \bigoplus_{\ell=0}^{L} F^\ell \in \mathbb{R}^{(L+1)c}$
14 Use $F$ for downstream tasks.
```

Algorithm 2: Second-order CCN

```
1  **Input:** $G, l_v, L$
2  **Parameters:** Matrices $W_0 \in \mathbb{R}^{c \times c}$, $W_1, .., W_L \in \mathbb{R}^{(18c) \times c}$
   and biases $b_0, .., b_L$.
3  $F_v^0 \leftarrow \Upsilon(W_0 l_v + b_0 \mathbb{1})$ $(\forall v \in V)$
4  Reshape $F_v^0$ to $1 \times 1 \times c$ $(\forall v \in V)$
5  **for** $\ell = 1, .., L$ **do**
6      **for** $v \in V$ **do**
7          $F_{w \to v}^\ell \leftarrow \chi \times F_w^{\ell-1} \times \chi^T$ where $\chi = \chi_{w \to v}^\ell$
             $(\forall w \in \mathcal{P}_v^\ell)$
8          Apply virtual tensor contraction algorithm
             (Sec.8) with inputs $\{F_{w \to v}^\ell | w \in \mathcal{P}_v^\ell\}$ and the
             restricted adjacency matrix $A \downarrow_{\mathcal{P}_v^\ell}$ to compute
             $\overline{F}_v^\ell \in \mathbb{R}^{|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times (18c)}$.
9          $F_v^\ell \leftarrow \Upsilon(\overline{F}_v^\ell \times W_\ell + b_\ell \mathbb{1})$
10     **end**
11 **end**
12 $F^\ell \leftarrow \sum_{v \in V} \Theta(F_v^\ell)$ $(\forall \ell)$
13 Graph feature $F \leftarrow \bigoplus_{\ell=0}^{L} F^\ell \in \mathbb{R}^{(L+1)c}$
14 Use $F$ for downstream tasks.
```

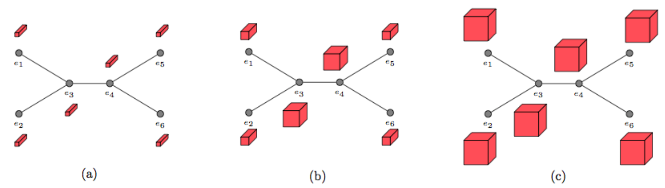


Figure 2: CCN 2D on C_2H_4 molecular graph

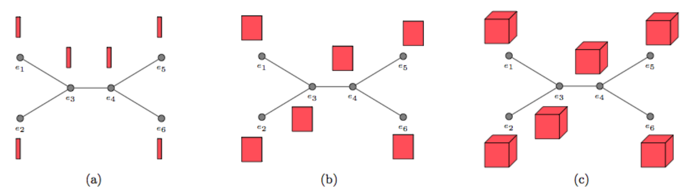


Figure 3: Zeroth, first and second order message passing

$F_{e_3}^1, F_{e_4}^1 \in \mathbb{R}^{4 \times 4 \times c}$ (see figure 2(b)). In all other layers $\ell = \infty$, $\mathcal{P}_e^\infty \equiv V$ and $F_e^\infty \in \mathbb{R}^{6 \times 6 \times c}$ $(\forall e)$ (see figure 2(c)).

Figure 3 shows the difference among zeroth, first and second order message passing (see from left to right) with layer $\ell \geq 2$. In the zeroth order, the vertex representation is always a vector of c channels (see figure 3(a)). In the first and second order (see figures 3(b)(c)), the vertex representation is a matrix of size $6 \times c$ or a 3rd order tensor of size $6 \times 6 \times c$ in which each channels is represented by a vector of length 6 or a matrix of size 6×6, respectively. With higher orders, CCNs can capture more topological information.

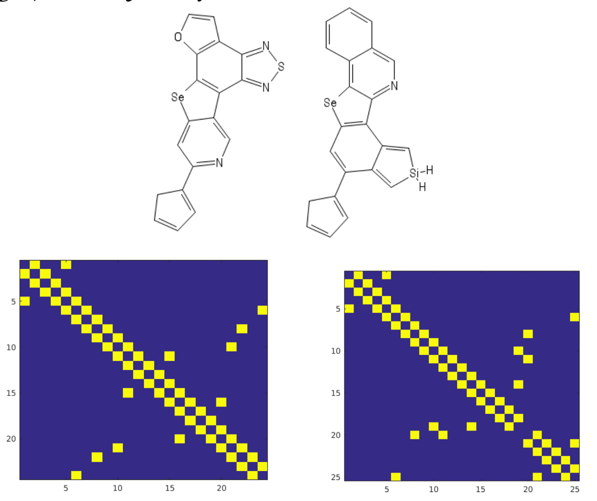
10 EXPERIMENTS

We now compare our CCN framework (Section 9) to several standard graph learning algorithms. We focus on two datasets that contain the result of a large number of Density Functional Theory (DFT) calculations:

(1) **The Harvard Clean Energy Project (HCEP)**, consisting of 2.3 million organic compounds that are candidates for use in solar cells [1]. Figure 4 shows an example of two molecules with similar graph structures in the HCEP dataset.
(2) **QM9**, a dataset of ~134k organic molecules with up to nine heavy atoms (C, O, N and F) [2] out of the GDB-17 universe of molecules [?]. Each molecule contains data including 13 target chemical properties, along with the spatial position of every constituent atom.

DFT [? ?] is the workhorse of the molecular chemistry community, given its favorable tradeoff between accuracy and computational power. Still, it is too costly for tasks such as drug discovery or materials engineering, which may require searching through millions of candidate molecules. An accurate prediction of molecular properties would significantly aid in such tasks.

Figure 4: Molecules $C_{18}H_9N_3OSSe$ (left) and $C_{22}H_{15}NSeSi$ (right) with adjacency matrices



We are interested in the ability of our algorithm to learn on both pure graphs, and also on physical data. As such, we perform three experiments. We start with two experiments based only upon atomic identity and molecular graph topology:

(1) **HCEP**: We use a random sample of 50,000 molecules of the HCEP dataset; our learning target is Power Conversion Efficiency (PCE), and we present the mean average error (MAE). The input vertex feature l_v is a one-hot vector of atomic identity concatenated with purely synthesized graph-based features.

(2) **QM9(a)**: We predict the 13 target properties of every molecule. For this text we consider only heavy atoms and exclude hydrogen. Vertex feature initialization is performed in the same manner as the HCEP experiment. For training the neural networks, we normalized all 13 learning targets to have mean 0 and standard deviation 1. We report the MAE with respect to the normalized learning targets.

We also tested our algorithm's ability to learn on DFT data based upon physical features. We perform the following experiment:

(3) **QM9(b)**: The QM9 dataset with each molecule including hydrogen atoms. We use both physical atomic information (vertex features) and bond information (edge features) including: atom type, atomic number, acceptor, donor, aromatic, hybridization, number of hydrogens, Euclidean distance and Coulomb distance between pair of atoms. All the information is encoded in a vectorized format.

To include the edge features into our model along with the vertex features, we used the concept of a *line graph* 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. 9). 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. (See step 12 of the algorithm 2).

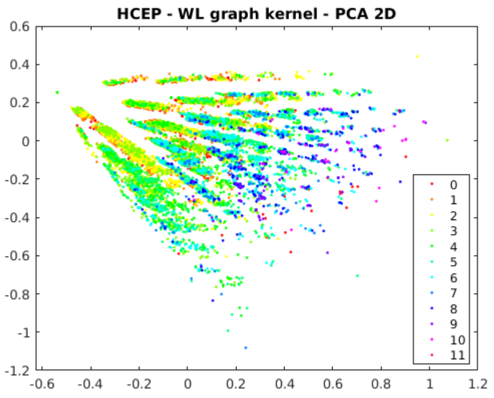
In QM9(b), we report the mean average error for each learning target in its corresponding physical unit and compare it against the Density Functional Theory (DFT) error given by [?].

In the case of HCEP, we compared CCNs to lasso, ridge regression, random forests, gradient boosted trees, optimal assignment Weisfeiler–Lehman graph kernel [?] (WL), neural graph fingerprints [?], and the "patchy-SAN" convolutional type algorithm (referred to as PSCN) [?]. 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 uses atom labels of each vertex as base features. For ridge regression and lasso, we cross validated over λ. For random forests and gradient boosted trees, we used 400 trees, and cross validated over max depth, minimum samples for a leaf, minimum samples to split a node, and learning rate (for GBT). For neural graph fingerprints, we used 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.

For QM9(a), we compared against the Weisfeiler–Lehman graph kernel, neural graph fingerprints, and PSCN. The settings for NGF and PSCN are as described for HCEP. For QM9(b), we compared against DFT error provided in [?].

We initialized the synthesized graph-based features of each vertex with computed histogram alignment features, inspired by [?], of depth up to 10. Each vertex receives a base label $l_v = \text{concat}_{d=1}^{10} H_v^d$ where $H_v^d \in \mathbb{R}^c$ (with c being the total number of distinct discrete node labels) is the vector of relative frequencies of each label for the set of vertices at distance equal to d from vertex v. Our CCNs architecture contains up to five levels.

Figure 5: 2D PCA projections of Weisfeiler-Lehman features in HCEP



In each experiment we separated 80% of the dataset for training, 10% for validation, and evaluated on the remaining 10% test set. We used Adam optimization [?] with the initial learning rate set to 0.001 after experimenting on a held out validation set. The learning rate decayed linearly after each step towards a minimum of 10^{-6}.

Our method, Covariant Compositional Networks, and other graph neural networks such as Neural Graph Fingerprints [?], PSCN [?] and Gated Graph Neural Networks [?] are implemented based on the GraphFlow framework (see chapter ??).

Tables 1, 2, and 3 show the results of HCEP, QM9(a) and QM9(b) experiments, respectively. Figures 5 and 6 show the 2D PCA projections of *learned* molecular representations in HCEP dataset with Weisfeiler-Lehman, Covariant Compositional Networks 1D & 2D, respectively. On the another hand, figures 7 and 8 show the 2D projections with t-SNE [?]. The colors represent the PCE values ranging from 0 to 11. Figure 10 shows the distributions between ground-truth and prediction of CCN 1D & 2D in HCEP.

Discussion

On the subsampled HCEP dataset, CCN outperforms all other methods by a very large margin. In the QM9(a) experiment, CCN obtains better results than three other graph learning algorithms for all 13 learning targets. In the QM9(b) experiment, our method gets smaller errors comparing to the DFT calculation in 11 out of 12 learning targets (we do not have the DFT error for R2).

Figure 6: 2D PCA projections of CCNs graph representations in HCEP

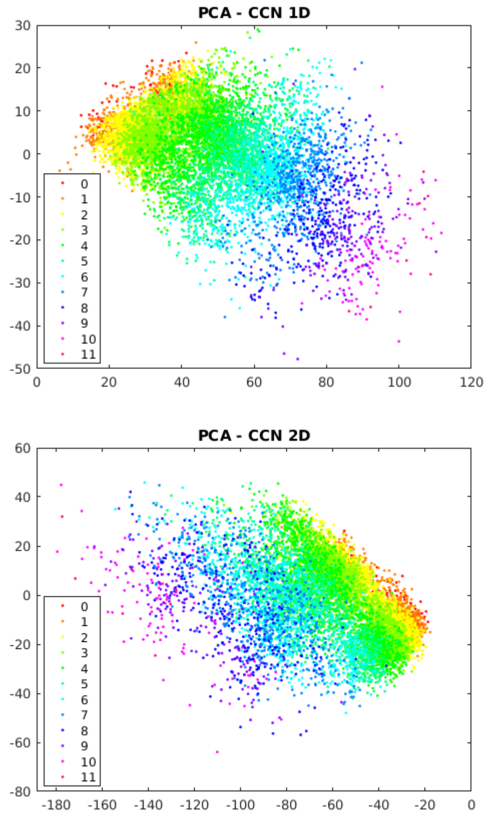
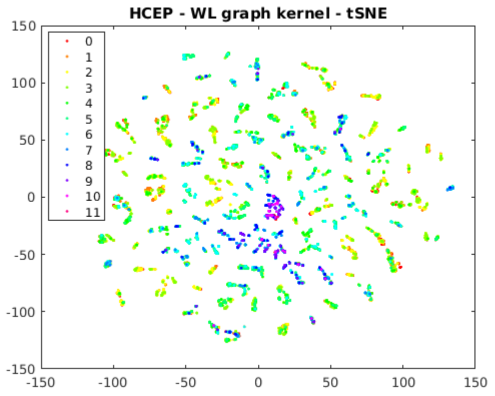


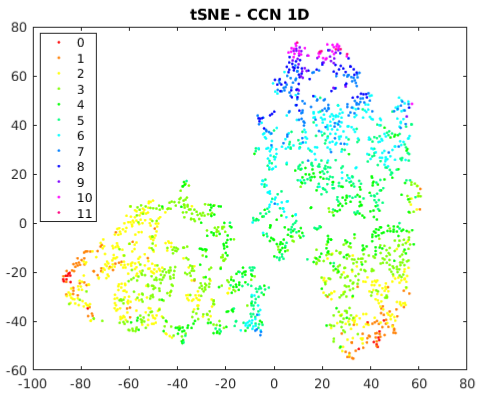
Figure 7: 2D t-SNE projections of Weisfeiler-Lehman features in HCEP

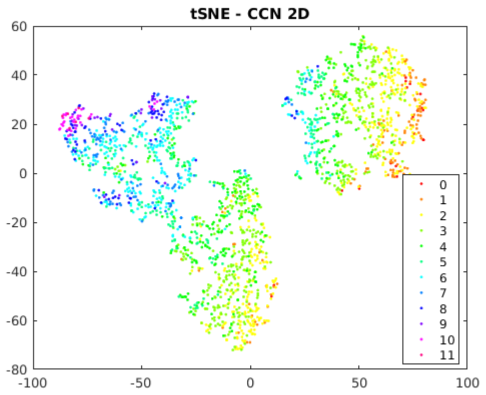


11 CONCLUSION

We extended Message Passing Neural Networks and generalized convolution operation for Covariant Compositional

Figure 8: 2D t-SNE projections of CCNs graph representations in HCEP





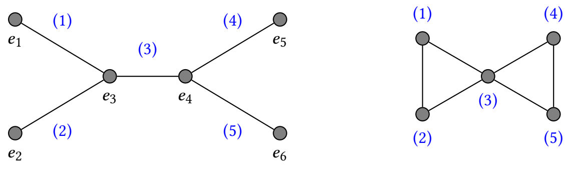


Figure 9: Molecular graph of C_2H_4 (left) and its corresponding line graph (right).

Networks by higher-order representations in order to approximate Density Functional Theory. We obtained very promising results and outperformed other state-of-ther-art graph neural networks such as Neural Graph Fingerprint and Learning Convolutional Neural Networks on Harvard Clean Energy Project and QM9 datasets.

REFERENCES

[1] 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. 2011. The Harvard Clean Energy Project: Large-Scale Computational Screening and Design of Organic Photovoltaics on the World Community Grid. *J. Phys. Chem. Lett.* 2 (2011), 2241–2251.

Table 1: HCEP regression results

	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
WL graph kernel	0.805	1.096
Neural graph fingerprints	0.851	1.177
PSCN	0.718	0.973
CCN 1D	**0.216**	**0.291**
CCN 2D	**0.340**	**0.449**

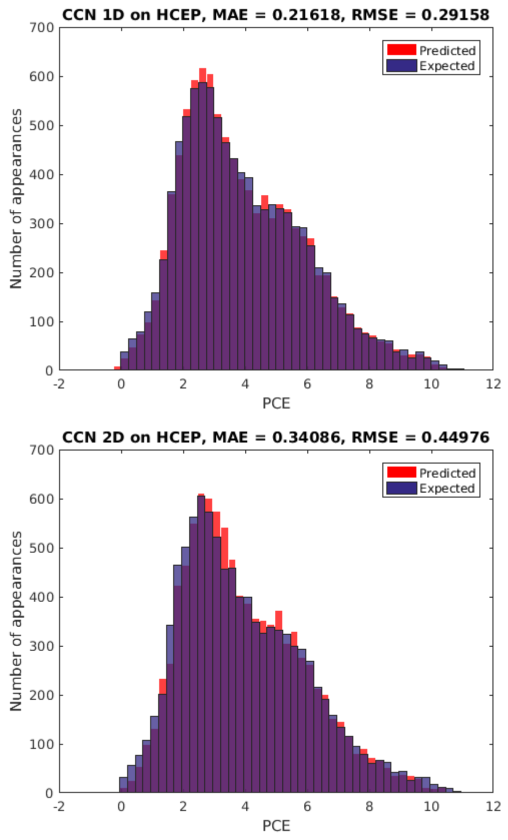
Table 2: QM9(a) regression results (MAE)

Target	WLGK	NGF	PSCN	CCN 2D
alpha	0.46	0.43	0.20	**0.16**
Cv	0.59	0.47	0.27	**0.23**
G	0.51	0.46	0.33	**0.29**
gap	0.72	0.67	0.60	**0.54**
H	0.52	0.47	0.34	**0.30**
HOMO	0.64	0.58	0.51	**0.39**
LUMO	0.70	0.65	0.59	**0.53**
mu	0.69	0.63	0.54	**0.48**
omega1	0.72	0.63	0.57	**0.45**
R2	0.55	0.49	0.22	**0.19**
U	0.52	0.47	0.34	**0.29**
U0	0.52	0.47	0.34	**0.29**
ZPVE	0.57	0.51	0.43	**0.39**

Table 3: QM9(b) regression results (MAE)

Target	CCNs	DFT error	Physical unit
alpha	**0.19**	0.4	Bohr^3
Cv	**0.06**	0.34	cal/mol/K
G	**0.05**	0.1	eV
gap	**0.11**	1.2	eV
H	**0.05**	0.1	eV
HOMO	**0.08**	2.0	eV
LUMO	**0.07**	2.6	eV
mu	0.43	**0.1**	Debye
omega1	**2.54**	28	cm^{-1}
R2	5.03	-	Bohr^2
U	**0.06**	0.1	eV
U0	**0.05**	0.1	eV
ZPVE	**0.0043**	0.0097	eV

Figure 10: Distributions of ground-truth and prediction of CCN 1D & 2D in HCEP



[2] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld. 2014. Quantum chemistry structures and properties of 134 kilo molecules. *Sci. Data* 1, 140022 (2014).

[3] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Magazine* 29, 3 (2008).