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

We propose Covariant Compositional Networks (CCNs), a novel neural network architecture for learning on graphs. CCNs use tensor representations for vertex features which can then be manipulated with permutation covariant tensor operations as opposed to the standard symmetric operations used in other graph neural network models. These permutation covariant operations allow us to build more expressive graph representations while still maintaining permutation invariance.

For learning small-scale molecular graphs, we investigate the efficacy of CCNs in estimating Density Functional Theory (DFT), a widely used but expensive approach to compute the electronic structure of matter. We obtain promising results in for this task and outperform other graph learning models on the Harvard Clean Energy Project [4] and QM9 [13] 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, 9 pages.

Anchorage '19, August 05, 2019, Anchorage, AK

1 INTRODUCTION

A central problem in graph learning domains is how to construct expressive vector representation of graphs which can later be used in downstream tasks like graph regression or graph classification. This is challenging because a graph's representation must have a fixed length regardless of the size of the graphs in the dataset and the representation must be invariant to permutations of the graph's vertices. Traditionally, researchers addressed these challenges by using graph kernels. Among the most successful of these kernels is the famed Weisfeiler-Lehman graph kernel [15], which builds a multi-level representation of a graph through a series of message passing and hashing steps. However, kernels methods scale quadratically in the size of the dataset, making them unfeasible beyond a few thousand datapoints. Graph neural networks have emerged as a scaleable alternative to graph kernels. Most graph neural networks can be seen as a differentiable extension of the Weisfeiler-Lehman algorithm that replaces the *fixed* hashing step with a *learnable* non-linear mapping. These networks produce permutation invariant representations of graphs by aggregating local and global vertex information through a series of symmetric operations on vertices (usually by summing vertex features in a 1-hop neighborhood around each vertex). We argue that the permutation invariant vertex aggregation operations performed by most graph neural networks limit their expressive power.

Thus, we propose Covariant Compositional Networks (CCNs) as an alternative. Section 2 describes the general architecture underlying CCNs. In our compositional nets framework, we represent vertex features with higher *covariant tensors* (defined in Section 3). These structured tensors representations then naturally give rise to the tensor operations defined in Section 4 which have the crucial property of

maintaining *permutation covariance*. Section 5 describes the full architecture of a compnet as well as two types of CCNs that can be implemented. Lastly in Section 6, we describe our experiments on two molecular graph datasets: the Harvard Clean Energy Project [4] and QM9 [13], where our model outperforms competing graph neural networks on graph regression tasks.

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. These aggregation functions cannot capture any information about the connectivity of vertices' neighborhoods. Therefore, we introduce **permutation covariance** below and argue that it is a desirable property for our neural activations f.

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 $f_{v'} = R_\pi(f_v)$.

This definition says permuting the vertices of graph G must change the activations of its vertices in a predictable manner that is controlled by some fixed function R_π that depends on the permutation π.

4 MESSAGE PASSING

First order Message Passing

We call standard message passing **zero'th order message passing** where each vertex is represented by a feature vector of length c (or c channels). When we sum together vertex features of this form, we lose identity information on where certain vertex features originated from. Hence, we propose **first order message passing** by instead 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:
$$[P_\pi]_{i,j} \triangleq \begin{cases} 1, & \pi(j) = i \\ 0, & otherwise \end{cases} \quad (1)$$
The transformed activation $f_{v'}$ will be:
$$[f_{v'}]_{a,s} = [f_v]_{\pi^{-1}(a),s}$$
where s is the channel index.

Second order Message Passing

Instead of representing a vertex with a feature matrix (a 2nd order tensor) 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:

Definition 4.2. 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.

Third and higher order Message Passing

Following this pattern, we can define third, fourth, and in general, k'th order nodes in a comp-net. Their activations are k'th order tensors which transform under permutations as $f_v \mapsto f_{v'}$:

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

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 quantity which transforms according to this equation (ignoring the channel index) a **k'th order P-tensor**.

Definition 4.3. 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.

Tensor aggregation rules

The previous sections prescribed 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

$$\{\{a_1^1,..,a_{p_1}^1\},..,\{a_1^q,..,a_{p_q}^q\}\}$$

as the $(k - \sum_j p_j)$ order tensor:

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

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

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

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

Proposition 4.1 shows that all of the above operations as well as linear combinations preserve the permutation covariance property of P-tensors. Therefore, they can be applied within the aggregation function Φ.

Proposition 4.1. *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) *$A_{i_1,i_2,..,i_k}\delta^{a_1^1,..,a_{p_1}^1}\cdots\delta^{a_1^q,..,a_{pq}^q}$ 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 4.2, 4.3 and 4.4 show that tensor promotion, concatenation and taking tensor products with the local adjacency matrix preserve permutation covariance, and hence can be applied within Φ.

Proposition 4.2. *Let node v be a descendant of node v' in a comp-net $\mathcal{N}$ with corresponding receptive fields: $\mathcal{P}_v = (e_{p_1},..,e_{p_m})$ and $\mathcal{P}_{v'} = (e_{q_1},..,e_{q_{m'}})$. We remark that $\mathcal{P}_v \subseteq \mathcal{P}_{v'}$. Define $\chi^{v\to v'} \in \mathbb{R}^{m\times m'}$ as an indicator matrix:*

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

*If F_v is a k'th order P-tensors with respect to permutations $(e_{p_1},..,e_{p_m})$ the following **promoted** tensor is a is a k'th order P-tensor with respect to permutations of $(e_{q_1},..,e_{q_{m'}})$:*

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

In equation 4, node v' promotes P-tensors from its child 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. We remark that promoted tensors are padded with zeros.

Proposition 4.3. *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 4.4 gives us a way to explicitly add topological information to the activation.

Proposition 4.4. *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.*

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 $\mathcal{T}_{i_1,..,i_5} \delta^{i_{a_1}} \delta^{i_{a_2}} \delta^{i_{a_3}}$, 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 $\mathcal{T}_{i_1,..,i_5} \delta^{i_{a_1}} \delta^{i_{a_2},i_{a_3}}$, 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 $\mathcal{T}_{i_1,..,i_5} \delta^{i_{a_1},i_{a_2},i_{a_3}}$. 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.

5 ARCHITECTURE

Recent work on graph neural networks [1, 3, 8, 11] 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 step.

Our architecture represents vertices 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" (CCN 1D) and "second order CCN" (CCN 2D), respectively.

Starting with an input graph $G = (V, E)$, we construct a compositional 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:

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

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 a 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 a 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 between levels of the network. The key to doing this in a way that preserves covariance is to "promote-stack-reduce" the tensors at each level of the network.

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, not all vertices in a given receptive field have same sized tensors. To account for this, we use an index function χ that ensures all tensors are the same size by padding them 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. 4.2. 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. 4.3. 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. 4.4.) Finally, we can reduce the higher order tensor down to the expected size of the vertex representation using the tensor contractions described in Prop. 4.1.

Including all possible tensor contraction will introduce additional channels to the vertex representations. 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 learned through backpropagation. The last step of our aggregation function Φ is to pass this tensor through an element-wise nonlinear function Υ such as a ReLU. (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. 4.1.) We sum up all the reduced vertex representations of a graph to get a single vector which we use as the graph's representation. This final graph representation can then used as an input to a final fully connected layer for regression or classification tasks. 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 as higher order tensors are often too large to store 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 on GPUs.

For example, consider the operations in step 8 in algorithm 2. This requires performing contractions over several indices on the two inputs $\mathcal{F} = \{F^\ell_{w\to v} | w \in \mathcal{P}_v^\ell\}$, in which $\mathcal{F}_{i_1} = F^\ell_{w_{i_1}\to v}$ is of size $|\mathcal{P}_v^\ell| \times |\mathcal{P}_v^\ell| \times c$, and $\mathcal{A} = A\downarrow_{\mathcal{P}_v^\ell}$. The corresponding tensor element can be efficiently computed on-the-fly as follows:

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

where i_6 is the channel index.

In our experiments using CCN 2D, we implement 18 different contractions (see Prop. 4). 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.

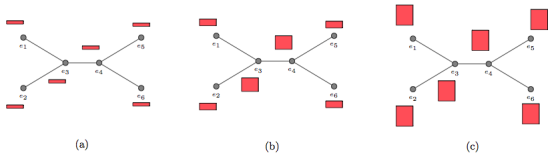

Figure 1: Tensor activations of our CCN-1D architecture on C_2H_4 molecular graph. The tensor activations of each vertex in a CCN 1D model are shown after 0, 1, and 2 rounds of message passing in (a), (b) and (c). The number of rows and columns in the activation tensor are respectively equal to the size of the receptive field.

To better illutrate the CCN 1D and CCN 2D models, we'll run through an example of the message passing and aggregation steps on the molecular graph of C_2H_4. See Algorithm 1 for the pseudocode for CCN 1D and Algorithm 2 for CCN 2D.

Figure 1 shows a visualization of CCN 1D's tensors on C_2H_4's molecular graph. The central 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 initial 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., $\mathcal{P}^1_{e_1} = \{e_1, e_3\}$), 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., $\mathcal{P}^1_{e_3} = \{e_1, e_2, e_3, e_4\}$) and $\mathcal{P}^1_{e_4} = \{e_3, e_4, e_5, e_6\}$), thus $F^1_{e_3}, F^1_{e_4} \in \mathbb{R}^{4\times c}$ (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 $F^\infty_e \in \mathbb{R}^{6\times c}$ (see figure 1(c)).

Figure 2 shows a visualization of CCN 2D's tensors on C_2H_4 molecular graph after a few steps of message passing. In the bottom layer $\ell = 0$, $|\mathcal{P}^0_e| = 1$ and $F^0_e \in \mathbb{R}^{1\times 1\times c}$ for every atom e (see figure 2(a)). In the first layer $\ell = 1$, $|\mathcal{P}^1_e| = 2$ and $F^1_e \in \mathbb{R}^{2\times 2\times c}$ for hydrogen atom $e \in \{e_1, e_2, e_5, e_6\}$, and for carbon atoms $|\mathcal{P}^1_{e_3}| = |\mathcal{P}^1_{e_4}| = 4$ and $F^1_{e_3}, F^1_{e_4} \in \mathbb{R}^{4\times 4\times c}$ (see figure 2(b)). In all other layers $\ell = \infty$, $\mathcal{P}^\infty_e \equiv V$ and $F^\infty_e \in \mathbb{R}^{6\times 6\times c}$ $(\forall e)$ (see figure 2(c)).

Figure 3 shows the difference between zeroth, first and second order message passing (see from left to right) with layer $\ell \geq 2$. Note that after 2 rounds of message passing on the graph of C_2H_4, every vertice's receptive field will consist of all 6 vertices in the graph. In a zeroth order model (figure 3(a)), the vertex representation will still be vector of c channels. In a first order model (figure 3(b)), the vertex

Algorithm 1: First-order CCN

Input: G, l_v, L
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.
$F_v^0 \leftarrow \Upsilon(W_0 l_v + b_0 \mathbb{1})\ (\forall v \in V)$
Reshape F_v^0 to $1 \times c\ (\forall v \in V)$
for $\ell = 1, .., L$ **do**
  for $v \in V$ **do**
    $F^\ell_{w\to v} \leftarrow \chi \times F_w^{\ell-1}$ where $\chi = \chi^\ell_{w\to v}\ (\forall w \in \mathcal{P}_v^\ell)$
    Concatenate the promoted tensors in $\{F^\ell_{w\to v} | w \in \mathcal{P}_v^\ell\}$ and apply 2 tensor contractions that results in $\overline{F}_v^\ell \in \mathbb{R}^{|\mathcal{P}_v^\ell|\times(2c)}$.
    $F_v^\ell \leftarrow \Upsilon(\overline{F}_v^\ell \times W_\ell + b_\ell \mathbb{1})$
  end
end
$F^\ell \leftarrow \sum_{v\in V} \Theta(F_v^\ell)\ (\forall \ell)$
Graph feature $F \leftarrow \bigoplus_{\ell=0}^{L} F^\ell \in \mathbb{R}^{(L+1)c}$
Use F for downstream tasks.

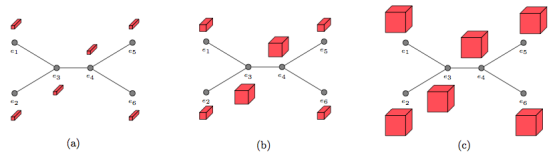

Figure 2: Tensor activations for our CCN-2D architecture applied to a C_2H_4 molecular graph. The tensor activations of each vertex in a CCN 2D model are shown after 0, 1, and 2 rounds of message passing in (a), (b) and (c). Here the rows and columns correspond to the size of the receptive field, whereas the depth of the tensor is determined by the number of channels.

representation is a matrix of size $6 \times c$—each channel is represented by a vector of size 6. In a second order model (figure 3(c)), the vertex representation will be a 3rd order tensor of size $6\times6\times c$—each channel is represented by matrix of size 6×6.

6 EXPERIMENTS

We now compare our CCN framework (Section 5) 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 [4].

Algorithm 2: Second-order CCN

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

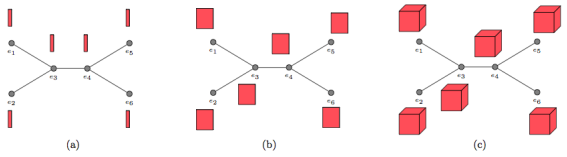

Figure 3: Geometry of the tensor activations in zeroth (CCN 0D), first (CCN 1D), and second (CCN 2D) order message passing algorithms. Vertices have a vector (zeroth order), matrix (first order), and second order tensor representations corresponding as shown in (a), (b), and (c).

(2) **QM9**, a dataset of ~134k organic molecules with up to nine heavy atoms (C, O, N and F) [13] out of the GDB-17 universe of molecules [14]. Each molecule contains data including 13 target chemical properties, along with the spatial position of every constituent atom.

DFT [5, 9] 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.

We are interested in the ability of our algorithm to learn using only the adjacency matrices and vertex labels of graphs, and using additional physical features of the graphs. As such, we perform three experiments. We start with two experiments based only upon atomic identity (vertex labels) and molecular graph topology (adjacency matrices):

(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.

To test our algorithm's ability to learn on DFT data based upon physical features, we perform the following experiment:

(3) **QM9(b)**: On the QM9 dataset, 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 vector.

    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 the following way: 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. 4). 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 [2].

For the HCEP experiment, we compared CCNs to lasso, ridge regression, random forests, gradient boosted trees, optimal assignment Weisfeiler–Lehman graph kernel [10] (WL), neural graph fingerprints [1], and the "patchy-SAN" convolutional type algorithm (referred to as PSCN) [12]. 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 [2].

We initialized the synthesized graph-based features of each vertex with computed histogram alignment features, inspired by [10], 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.

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 [7] 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 [1], PSCN [12] and Gated Graph Neural Networks [11] are implemented based on the GraphFlow framework [6].

Tables 1, 2, and 3 show the results of HCEP, QM9(a) and QM9(b) experiments, respectively.

Discussion

On the subsampled HCEP dataset, CCN outperforms all other methods by a 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,

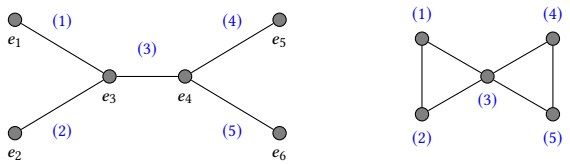

Figure 4: Molecular graph of C_2H_4 (left) and its corresponding line graph (right).

	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 1: HCEP regression results. MAE and RMSE results of each model on predicting the Power Conversion Efficiency (PCE) for graphs on the test set of HCEP. Lower values are better.

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 2: Results of training various learning algorithms on the QM9(a) dataset. Mean Absolute Error (MAE) results of training QM9(a) on WLGK, NGF, PSCN, and CCN 2D to predict each of the 13 learning targets discussed in the main text. All MAEs are presented with respected to the standardized learning targets. Lower results are better.

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).

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

Table 3: Regression results of CCN-1D architecture applied to QM9(b). A comparison between CCN prediction error and DFT error known as "chemical accuracy."

7 CONCLUSION

We extended Message Passing Neural Networks using higher-order tensors and tensor aggregation operations that preserve permutation covariance. Our resulting models outperform other graph learning models on prediction tasks over the Harvard Clean Energy Project and QM9 datasets, highlighting the potential of our CCN architecture for learning expressive graph models.

REFERENCES

[1] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*. 2224–2232.

[2] 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. 2017. Prediction Errors of Molecular Machine Learning Models Lower than Hybrid DFT Error. *J. Chem. Theory Comput.* 13 (09 2017), 5255–5264.

[3] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212* (2017).

[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. 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.

[5] P. Hohenberg and W. Kohn. 1964. Inhomogeneous Electron Gas. *Phys. Rev.* 136 (1964), 864–871.

[6] T. S. Hy, S. Trivedi, H. Pan, B. M. Anderson, and R. Kondor. 2018. Predicting molecular properties with covariant compositional networks. *Journal of Chemical Physics* 148 (2018). Issue 24.

[7] D. P. Kingma and J. Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proc. ICLR*. San Diego.

[8] T. N. Kipf and M. Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of International Conference on Learning Representations*.

[9] W. Kohn and L. J. Sham. 1965. Self-Consistent Equations Including Exchange and Correlation Effects. *Phys. Rev.* 140 (1965), 1133–1138.

[10] N. M. Kriege, P. Giscard, and R. Wilson. 2016. On Valid Optimal Assignment Kernels and Applications to Graph Classification. *Advances in Neural Information Processing Systems 29* (2016).

[11] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[12] M. Niepert, M. Ahmed, and K. Kutzkov. 2016. Learning convolutional neural networks for graphs. In *Proceedings of the International Conference on Machine Learning*.

[13] 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).

[14] L. Ruddigkeit, R. van Deursen, L. C. Blum, and Jean-Louis Reymond. 2012. Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17. *J. Chem. Inf. Model.* 52 (2012), 2864–2875. Issue 11.

[15] N. Shervashidze, P. Schweitzer, E. J. van Leeuwan, K. Mehlhorn, and K. M. Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research* 12 (2011), 2539–2561.