Transformers Generalize DeepSets and Can be Extended to Graphs & Hypergraphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present a generalization of Transformers to any-order permutation invariant data (sets, graphs, and hypergraphs). We begin by observing that Transformers generalize DeepSets, or first-order (set-input) permutation invariant MLPs. Then, based on recently characterized higher-order invariant MLPs, we extend the concept of self-attention to higher orders and propose higher-order Transformers for order-k data ($k = 2$ for graphs and $k > 2$ for hypergraphs). Unfortunately, higher-order Transformers turn out to have prohibitive complexity $\mathcal{O}(n^{2k})$ to the number of input nodes n. To address this problem, we present sparse higher-order Transformers that have quadratic complexity to the number of input hyperedges, and further adopt the kernel attention approach to reduce the complexity to linear. In particular, we show that the sparse second-order Transformers with kernel attention are theoretically more expressive than message passing operations while having an asymptotically identical complexity. Our models achieve significant performance improvement over invariant MLPs in graph classification and set-to-graph prediction tasks, and are competitive to state-of-the-art graph neural networks.

1 Introduction

Graph is a universal data modality used to model social networks [24], chemical compounds [9], biological structures [8], and interactions in particle physics [15, 25]. Recent graph neural networks (GNNs) adopt a message-passing scheme [40, 39, 32], where the node features are propagated and aggregated recurrently according to the neighborhood structure given in the input graph. Despite the simplicity, the local and recurrent nature makes them unable to discover dependency between any two nodes with a distance longer than the message-passing steps [10]. The locality of message-passing is also known to be related to the over-smoothing problem that prevents scaling of GNNs [18, 4, 23].

An alternative approach is using a more general set of operations that involve global interactions while respecting the permutation symmetry of graphs. Such operations either produce the same output regardless of the node ordering of input graph (permutation invariance), or commute with node reordering (permutation equivariance). Message-passing is an equivariant operation, but is restricted to local neighborhoods defined by the input graph. Recently, Maron et. al. (2019) [20] characterized the full space of invariant and equivariant linear layers. It turned out that these layers span not only the local neighborhood interactions of message-passing GNNs, but also global interactions such as the one between disconnected nodes, and even edge-to-node, or node-to-edge interactions (Figure 1). Notably, their formulation naturally extends to layers with different input & output orders (e.g., edge in, node out), and higher-order layers for hypergraphs. In theory, an invariant MLP composed of these layers should be more powerful than message-passing GNNs as they can model long-range dependency between nodes (Figure 1). However, they are currently not widely adopted due to relatively low performance and high asymptotic memory complexity.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

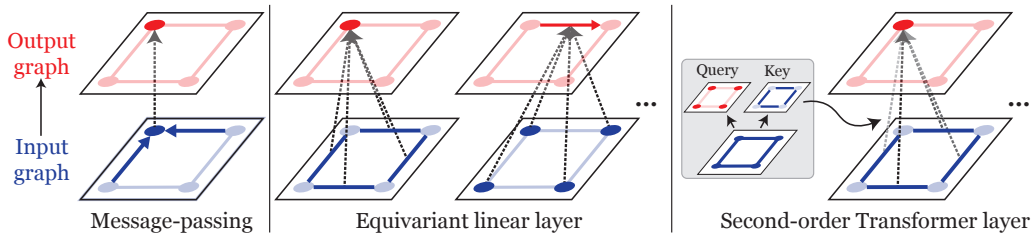


Figure 1: Illustrated operations of a message-passing GNN, an equivariant linear layer, and a second-order Transformer layer. A single output node is highlighted.

Contributions In this work, we present higher-order Transformers that address the low performance and high complexity of invariant MLPs. First, based on an observation that the renowned Transformer encoder generalizes first-order equivariant linear layers or DeepSets [37], we formulate higher-order Transformer layers that generalize equivariant linear layers by extending self-attention to higher orders (Figure 1). Second, while Transformer layers with input and output orders k, l has $\mathcal{O}(n^{k+l})$ asymptotic complexity, we show that by leveraging the sparsity of input hypergraphs, we can obtain $\mathcal{O}(m^2)$ complexity given input with m hyperedges. Further adopting kernel attention approaches, we propose a variant that reduces the complexity to $\mathcal{O}(m)$ and theoretically prove that they are more expressive than message passing networks. Finally, we test higher-order Transformers on a range of tasks, and demonstrate that they achieve significant improvements over invariant MLPs and are highly competitive in performance and scalability to the state-of-the-art GNNs.

2 Preliminary

In this section, we first describe how (hyper)graphs can be treated as higher-order tensors. We then describe linear layers that operate on higher-order tensors while respecting node permutation symmetry. In particular, we analyze operations within linear layers for second-order tensors (graphs) and show they involve global interactions on top of local aggregation.

Let us define some notations. We denote a set as $\{a, ..., b\}$, a tuple as $(a, ..., b)$, and denote $[n] = \{1, ..., n\}$. We denote the space of order-k tensors as $\mathbb{R}^{n^k \times d}$ where d is feature dimension. For an order-k tensor $\mathbf{A} \in \mathbb{R}^{n^k \times d}$, we use multi-index $\mathbf{i} = (i_1, ..., i_k) \in [n]^k$ to denote $\mathbf{A}_\mathbf{i} = \mathbf{A}_{i_1,...,i_k} \in \mathbb{R}^d$. Let S_n be the set of all permutations of $[n]$. $\pi \in S_n$ acts on $\mathbf{i}$ by $\pi(\mathbf{i}) = (\pi(i_1), ..., \pi(i_k))$, and acts on an order-$k$ tensor $\mathbf{A}$ by $(\pi \cdot \mathbf{A})_\mathbf{i} = \mathbf{A}_{\pi^{-1}(\mathbf{i})}$.

(Hyper)graphs as tensors Generally, a (hyper)graph data G can be represented as a tuple $(V, \mathbf{A})$, where V is a set of n nodes and $\mathbf{A} \in \mathbb{R}^{n^k \times d}$ encodes features attached to hyperedges. The type of the hypergraph is indicated by the order k of the tensor $\mathbf{A}$. First-order tensor is a set of features (e.g., point cloud, bag-of-words) where $\mathbf{A}_i$ is the feature of node i. Second-order tensor encodes edge features (e.g., adjacency) where $\mathbf{A}_{i_1,i_2}$ is the feature of edge (i_1, i_2). Generally, an order-k tensor encodes hyperedge features (e.g., mesh) where $\mathbf{A}_{i_1,...,i_k}$ is the feature of hyperedge $(i_1, ..., i_k)$.

Permutation invariance and equivariance Our problem of interest is building a functional relation $f(\mathbf{A}) \approx T$ between tensor $\mathbf{A}$ and target T. If T is a single output vector, we often require that f is *permutation invariant*, that it satisfies $f(\pi \cdot \mathbf{A}) = f(\mathbf{A})$; if T is a tensor $T = \mathbf{T}$, we often require that f is *permutation equivariant*, satisfying $f(\pi \cdot \mathbf{A}) = \pi \cdot f(\mathbf{A})$, for all $\pi \in S_n$ and $\mathbf{A} \in \mathbb{R}^{n^k \times d}$. In a typical design setup where a neural network f is built using linear layers and non-linear activations, the construction of f reduces to finding invariant and equivariant *linear* layers.

Invariant and equivariant linear layers We summarize invariant linear layers $L_{k \to 0} : \mathbb{R}^{n^k \times d} \to \mathbb{R}^{d'}$ and equivariant linear layers $L_{k \to l} : \mathbb{R}^{n^k \times d} \to \mathbb{R}^{n^l \times d'}$ identified in Maron et. al. (2019) [20]. Note that invariant layer is a special case of $L_{k \to l}$ with $l = 0$. In summary, given an input $\mathbf{A} \in \mathbb{R}^{n^k \times d}$, the order-$l$ output of an equivariant layer $L_{k \to l}$ can be written with $\mathbf{i} \in [n]^k, \mathbf{j} \in [n]^l$ as:

$$L_{k \to l}(\mathbf{A})_\mathbf{j} = \sum_\mu \sum_\mathbf{i} \mathbf{B}^\mu_{\mathbf{i},\mathbf{j}} \mathbf{A}_\mathbf{i} w_\mu + \sum_\lambda \mathbf{C}^\lambda_\mathbf{j} b_\lambda, \tag{1}$$

where $w_\mu \in \mathbb{R}^{d \times d'}$, $b_\lambda \in \mathbb{R}^{d'}$ are weight and bias parameters, $\mathbf{B}^\mu \in \mathbb{R}^{n^{k+l}}$ and $\mathbf{C}^\lambda \in \mathbb{R}^{n^l}$ are *basis tensors* (will be defined shortly), and μ and λ are *equivalence classes* of order-$(k + l)$ and order-l

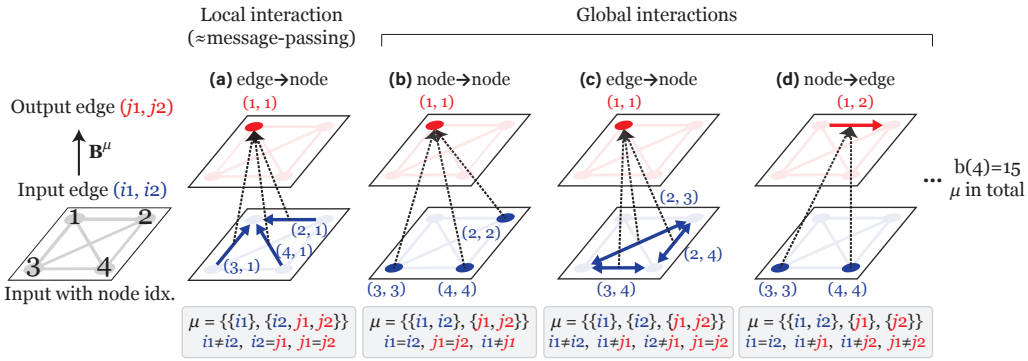


Figure 2: Example operations in a second-order layer $L_{2\to 2}$. We illustrate how input edges $(i_1, i_2) = \mathbf{i}$ are aggregated to an output edge $(j_1, j_2) = \mathbf{j}$ in various equivalence classes $(\mathbf{i}, \mathbf{j}) \in \mu$ via basis tensor $\mathbf{B}^{\mu}_{\mathbf{i},\mathbf{j}}$. Note that a loop ($(i_1, i_1)$ or (j_1, j_1)) represents a node.

multi-indices, respectively. The equivalence classes are defined upon equivalence relation $\sim$ that, for multi-indices $\mathbf{i}, \mathbf{j} \in [n]^k$, $\mathbf{i} \sim \mathbf{j}$ iff $(i_1, ..., i_k) = (\pi(j_1), ..., \pi(j_k))$ for some node permutation $\pi \in S_n$. Then, a multi-index $\mathbf{i}$ and all members $\mathbf{j}$ in its equivalence class have the same (permutation-invariant) equality pattern: $\mathbf{i}_a = \mathbf{i}_b \Leftrightarrow \mathbf{j}_a = \mathbf{j}_b$ for all $a, b \in [k]$. Consequently, each equivalence class μ (or λ) is a distinct set of all order-$(k + l)$ (or order-l) multi-indices having a specific equality pattern.

Notably, we can represent each equivalence class of order-k multi-indices as a unique partition of $[k]$ regardless of n, where the partition specifies the equality pattern. *e.g.,* with $k = 2$, we have two partitions and respective equivalent classes: $\mu_1 = \{\{i_1, i_2\}\}$ the set of all (i_1, i_2) with $i_1 = i_2$, and $\mu_2 = \{\{i_1\}, \{i_2\}\}$ the set of all (i_1, i_2) with $i_1 \neq i_2$. Thus, with $\mathrm{b}(k)$ the k-th Bell number or number of partitions of $[k]$, we have $\mathrm{b}(k+l)$ equivalence classes μ for the weight and $\mathrm{b}(l)$ equivalence classes λ for the bias. We guide the readers interested in derivation to Maron et. al. (2019) [20].

With the equivalence classes, basis tensors are then defined as follows:

$$\mathbf{B}^{\mu}_{\mathbf{i},\mathbf{j}} = \begin{cases} 1 & (\mathbf{i}, \mathbf{j}) \in \mu \\ 0 & \text{otherwise} \end{cases} \; ; \quad \mathbf{C}^{\lambda}_{\mathbf{j}} = \begin{cases} 1 & \mathbf{j} \in \lambda \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

In Eq. (1), each equivalence class μ determines which (hyper)edges participate and how they interact in summation $\sum_{\mathbf{i}} \mathbf{B}^{\mu}_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}}$. As an example, let us consider $L_{2\to 2}$ that maps input edges $\mathbf{i} = (i_1, i_2)$ to output edges $\mathbf{j} = (j_1, j_2)$. An equivalence class $\mu = \{\{i_1, i_2\}, \{j_1, j_2\}\}$ represents all $(\mathbf{i}, \mathbf{j}) \in \mu$ that $i_1 = i_2$, $j_1 = j_2$, and $i_1 \neq j_1$. Then, due to masking by $\mathbf{B}^{\mu}$, only input elements $\mathbf{A}_{i_1,i_2}$ with $i_1 = i_2$ participate in the summation and gives output $L_{2\to 2}(\mathbf{A})_{j_1,j_2}$ for $j_1 = j_2$, $i_1 \neq j_1$. Intuitively, this is analogous to computing a node feature by sum-pooling all the other nodes (Fig. 2(b)). Different μ accounts for other interactions as shown in Figure 2. Notably, it contains a richer set of operations beyond local interactions modeled by message-passing (Fig. 2(a)), such as global interaction across all nodes (Fig. 2(b)), and edge-to-node (Fig. 2(c)), node-to-edge interactions (Fig. 2(d)).

We finish the section by writing out the first-order equivariant layer $L_{1\to 1}$. As $\mathrm{b}(2) = 2$, the layer has two equivalence classes $\mu_1 = \{\{i_1, j_1\}\}$ and $\mu_2 = \{\{i_1\}, \{j_1\}\}$. Then, we have $\mathbf{B}^{\mu_1} = I_n$ and $\mathbf{B}^{\mu_2} = 1_n 1_n^\top - I_n$, with $1_n \in \mathbb{R}^n$ vector of ones. Then, given a set of features $\mathbf{A} \in \mathbb{R}^{n\times d}$,

$$L_{1\to 1}(\mathbf{A}) = I_n \mathbf{A} w_1' + (1_n 1_n^\top - I_n)\mathbf{A} w_2' + 1_n b^\top \tag{3}$$

$$= I_n \mathbf{A} w_1 + 1_n 1_n^\top \mathbf{A} w_2 + 1_n b^\top, \tag{4}$$

where $w_1, w_2, w_1', w_2' \in \mathbb{R}^{d\times d'}$, $b \in \mathbb{R}^{d'}$. $L_{1\to 1}$ is analogous to a combination of elementwise feedforward (μ_1) and sum-pooling of set elements (μ_2), and is also known as a DeepSet layer [37].

3 Higher-Order Transformers

In Section 2, we introduced higher-order linear equivariant layers $L_{k\to l}$, and showed that they contain various global and node/edge interactions that are not covered by message-passing. In this section, we establish a connection between the first-order equivariant layer $L_{1\to 1}$ and self-attention of Transformer encoder layers [29]. Then, we extend the relationship to higher orders by tensorizing queries and keys, and formulate higher-order Transformer layers. We finish the section by proposing a principled weight-sharing for queries and keys, which reduces a fair amount of attention computation.

3.1 Transformers generalize DeepSets

As shown in Section 2, first-order linear layer, or DeepSet, has a simple structure composed of feedforward and sum-pooling (Eq. (4)). Although it is theoretically proven to be a universal approximator of permutation-invariant functions [37], static sum-pooling could be limited in capturing interactions of set elements, motivating the use of sophisticated pooling. In particular, the self-attention mechanism of Transformer encoder [29] was shown to achieve a large performance gain in set modeling via context-aware weighted pooling [17, 36]. To see this, let us write out the Transformer encoder layers.

A Transformer encoder layer is a function $\mathrm{Enc} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ consisting of two layers: a self-attention layer $\mathrm{Attn} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ and an elementwise feedforward layer $\mathrm{MLP} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$. For a set of n input vectors $X \in \mathbb{R}^{n\times d}$, a Transformer layer computes the following:

$$\mathrm{Attn}(X)_i = X_i + \sum_{h=1}^{H}\sum_{j=1}^{n} \alpha_{ij}^h X_j w_h^V w_h^O, \tag{5}$$

$$\mathrm{Enc}(X)_i = \mathrm{Attn}(X)_i + \mathrm{MLP}(\mathrm{Attn}(X))_i, \tag{6}$$

where $\alpha^h = \sigma(Xw_h^Q(Xw_h^K)^\top)$ is an attention coefficient with an activation σ, H is the number of heads, d_H is head size, d_F is hidden dimension, and $w_h^O \in \mathbb{R}^{d_H\times d}$, $w_h^V, w_h^K, w_h^Q \in \mathbb{R}^{d\times d_H}$.[1]

Now, we show that Transformer layers are generalized first-order linear equivariant functions. By setting $\alpha_{ij}^h = 1$ and assuming that $\mathrm{MLP}(Y)$ approximates a linear layer $Yw^F + b^{F\top}$ following the universal approximation theorem [12], the Transformer layer reduces to the following [2] :

$$\mathrm{Enc}(X)_i = X_i(I_n + w^F) + \sum_{j=1}^{n} X_j w^{VO}(I_n + w^F) + b^{F\top}, \tag{7}$$

where $w^{VO} = \sum_{h=1}^{H} w_h^V w_h^O$. This is equivalent to a DeepSet layer in Eq. (4) with $w^1 = I_n + w^F$, $w^2 = w^{VO}(I_n + w^F)$, and $b = b^F$. In other words, we can convert DeepSets to Transformers by changing the static pooling to attention and replacing elementwise linear mapping with an MLP. We generalize this approach to higher-order input and output, which leads to the formulation of higher-order Transformers for graphs and hypergraphs.

3.2 Higher-order Transformer layers

In Section 3.1, we showed that Transformer layers are generalized first-order equivariant linear layers $L_{1\to1}$. Notably, the generalization procedure was equivalent to changing static pooling to attention and adding feedforward MLP. In this section, we generalize the approach to $L_{k\to l}$ with arbitrary orders k and l and formulate higher-order Transformer layer $\mathrm{Enc}_{k\to l}$.

In general, we define higher-order Transformer layer as a function $\mathrm{Enc}_{k\to l} : \mathbb{R}^{n^k\times d} \to \mathbb{R}^{n^l\times d}$ with two layers: a higher-order self-attention layer $\mathrm{Attn}_{k\to l} : \mathbb{R}^{n^k\times d} \to \mathbb{R}^{n^l\times d}$ and a feedforward layer $\mathrm{MLP}_{l\to l} : \mathbb{R}^{n^l\times d} \to \mathbb{R}^{n^l\times d}$. For an input tensor $\mathbf{A} \in \mathbb{R}^{n^k\times d}$, a Transformer layer computes:

$$\mathrm{MLP}_{l\to l}(\mathrm{Attn}_{k\to l}(\mathbf{A})) = L_{l\to l}^2\left(\mathrm{ReLU}(L_{l\to l}^1(\mathrm{Attn}_{k\to l}(\mathbf{A})))\right), \tag{8}$$

$$\mathrm{Enc}_{k\to l}(\mathbf{A}) = \mathrm{Attn}_{k\to l}(\mathbf{A}) + \mathrm{MLP}_{l\to l}(\mathrm{Attn}_{k\to l}(\mathbf{A})), \tag{9}$$

where $L_{l\to l}^1 : \mathbb{R}^{n^l\times d\to n^l\times d_F}$ and $L_{l\to l}^2 : \mathbb{R}^{n^l\times d_F\to n^l\times d}$ are equivariant linear layers with hidden dimension d_F. Remaining question is how to define and compute higher-order self-attention $\mathrm{Attn}_{k\to l}(\mathbf{A})$.

To design $\mathrm{Attn}_{k\to l}$, we remove the bias from Eq. (1) and introduce attention coefficients. It is achieved by changing each $\mathbf{B}^\mu \in \mathbb{R}^{n^{k+l}}$ to an attention coefficient tensor $\boldsymbol{\alpha}^{h,\mu} \in \mathbb{R}^{n^{k+l}}$ with multiple heads:

$$\mathrm{Attn}_{k\to l}(\mathbf{A})_{\mathbf{j}} = \sum_{h=1}^{H}\sum_{\mu}\sum_{\mathbf{i}} \boldsymbol{\alpha}_{\mathbf{i},\mathbf{j}}^{h,\mu}\mathbf{A}_{\mathbf{i}} w_{h,\mu}^V w_{h,\mu}^O, \tag{10}$$

where $w_{h,\mu}^O \in \mathbb{R}^{d_H\times d}$, $w_{h,\mu}^V \in \mathbb{R}^{d\times d_H}$ are learnable parameters, H denotes the number of heads, and d_H denotes head size. Then similar to first-order case (Section 3.1), we can show the following:

[1] Note that we omitted normalization after $\mathrm{Attn}(\cdot)$ and $\mathrm{MLP}(\cdot)$ for simplicity as in [36, 11].

[2] In practice, Transformer employs softmax in attention and deviates from Deepsets due to normalization.

Theorem 1. $\text{Enc}_{k\to l}$ *(Eq.* (9)*) is a generalization of* $L_{k\to l}$ *(Eq.* (1)*).*

Proof. Let $\boldsymbol{\alpha}^{h,\mu}_{\mathbf{i},\mathbf{j}} = 1$ for all h, μ, and $(\mathbf{i},\mathbf{j}) \in \mu$. This renders $\boldsymbol{\alpha}^{h,\mu} = \mathbf{B}^{\mu}$ from the definition of $\mathbf{B}^{\mu}$. Additionally, let $\text{MLP}_{l\to l}(\text{Attn}_{k\to l}(\mathbf{A}))_{\mathbf{j}} = \sum_{\lambda} \mathbf{C}^{\lambda}_{\mathbf{j}} b_{\lambda}$ for all $\mathbf{j} \in [n]^l$. That is, $\text{MLP}_{l\to l}$ ignores input and reduces to an invariant bias in Eq. (1). Then, Eq. (9) reduces to the following:

$$\text{Enc}_{k\to l}(\mathbf{A})_{\mathbf{j}} = \sum_{\mu} \sum_{\mathbf{i}} \mathbf{B}^{\mu}_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}} \sum_{h=1}^{H} w^{V}_{h,\mu} w^{O}_{h,\mu} + \sum_{\lambda} \mathbf{C}^{\lambda}_{\mathbf{j}} b_{\lambda}, \tag{11}$$

which is equivalent to Eq. (1) with $w_{\mu} = \sum_{h=1}^{H} w^{V}_{h,\mu} w^{O}_{h,\mu}$. □

Now, we describe how to compute each attention tensor $\boldsymbol{\alpha}^{\mu} \in \mathbb{R}^{n^{k+l}}$ from input $\mathbf{A} \in \mathbb{R}^{n^k \times d}$ (Eq. (10), we drop head index h for brevity). We obtain each attention tensor from higher-order query and key:

$$\boldsymbol{\alpha}^{\mu}_{\mathbf{i},\mathbf{j}} = \begin{cases} \sigma(\mathbf{Q}^{\mu}_{\mathbf{j}}, \mathbf{K}^{\mu}_{\mathbf{i}})/Z_{\mathbf{j}} & (\mathbf{i},\mathbf{j}) \in \mu \\ 0 & \text{otherwise} \end{cases} \text{ where } \mathbf{Q}^{\mu} = L^{\mu}_{k\to l}(\mathbf{A}), \mathbf{K}^{\mu} = L^{\mu}_{k\to k}(\mathbf{A}). \tag{12}$$

where $Z_{\mathbf{j}} = \sum_{\mathbf{i}} \sigma(\mathbf{Q}^{\mu}_{\mathbf{j}}, \mathbf{K}^{\mu}_{\mathbf{i}})$ is a normalization constant. Note that query and key tensors are computed from the input $\mathbf{A}$ using the equivariant linear layers in Eq. (1). This leads to permutation equivariance (or invariance) of Transformer encoder layer $\text{Enc}_{k\to l}$ in Eq. (9).

Reducing orders of query and key Although Eq. (10) and Eq. (12) provide a simple and generic definition of higher-order self-attention, we observe that there exist a lot of unnecessary computations. Specifically, there exist elements of query $\mathbf{Q}^{\mu}$ and key $\mathbf{K}^{\mu}$ that are unused in computation of attention coefficient $\boldsymbol{\alpha}^{\mu}_{\mathbf{i},\mathbf{j}}$, as it depends only on indices satisfying $(\mathbf{i},\mathbf{j}) \in \mu$. In fact, it turns out that the effective orders of query and key are much smaller than l and k, as we show below:

Proposition 1. *From Eq.* (12)*, let* $u(\cdot)$ *denote the number of unique entries in a multi-index. With* $u_q = u(\mathbf{j})$*,* $u_k = u(\mathbf{i})$ *for some* $(\mathbf{i},\mathbf{j}) \in \mu$*, we can always find suitable linear layers* $L^{\mu}_{k\to u_q}$*,* $L^{\mu}_{k\to u_k}$ *and index space mappings* $f^{\mu}_{q} : [n]^l \to [n]^{u_q}$*,* $f^{\mu}_{k} : [n]^k \to [n]^{u_k}$ *that satisfy the following.*

$$\boldsymbol{\alpha}^{\mu}_{\mathbf{i},\mathbf{j}} = \sigma(\tilde{\mathbf{Q}}^{\mu}_{\mathbf{j}'}, \tilde{\mathbf{K}}^{\mu}_{\mathbf{i}'})/\tilde{Z}_{\mathbf{j}'} \; \forall (\mathbf{i},\mathbf{j}) \in \mu, \tag{13}$$

where $\tilde{Z}_{\mathbf{j}} = \sum_{\mathbf{i}} \sigma(\tilde{\mathbf{Q}}^{\mu}_{\mathbf{j}'}, \tilde{\mathbf{K}}^{\mu}_{\mathbf{i}'})$*,* $\tilde{\mathbf{Q}}^{\mu} = L^{\mu}_{k\to u_q}(\mathbf{A})$*,* $\tilde{\mathbf{K}}^{\mu} = L^{\mu}_{k\to u_k}(\mathbf{A})$*,* $\mathbf{j}' = f^{\mu}_{q}(\mathbf{j})$*,* $\mathbf{i}' = f^{\mu}_{k}(\mathbf{i})$*.*

Proof. See the supplementary file. □

Based on Proposition 1, we can compute query and key in Eq. (12) in a much compact way using linear layers with output orders u_q and u_k instead of l and k. In addition to reducing memory footprint and computations, this also provides a principled approach for parameter-sharing within $\text{Enc}_{k\to l}$. Specifically, in Eq. (13), we propose to share a query tensor $\tilde{\mathbf{Q}} = L_{k\to u_q}(\mathbf{A})$ over μ with same u_q, and share a key tensor $\tilde{\mathbf{K}} = L_{k\to u_k}(\mathbf{A})$ over μ with same u_k, so that $\tilde{\mathbf{Q}}$ and $\tilde{\mathbf{K}}$ are reused over computation of several $\boldsymbol{\alpha}^{\mu}$. In our experiments, we observe that this weight-sharing is critical in achieving a feasible memory usage during backpropagation.

4 Asymptotically Efficient Higher-Order Transformers

In Section 3, we formulated higher-order Transformer layers $\text{Enc}_{k\to l}$ and showed that they generalize linear equivariant layers $L_{k\to l}$. However, this capability comes with a cost; the high asymptotic complexity of the Transformer encoder limits its practical merits. Specifically, we show the following:

Property 1. *Given input size* n*, the asymptotic complexity of a linear layer* $L_{k\to l}$ *(Eq.* (1)*) is* $\mathcal{O}(n^{k+l})$*, and complexity of an encoder layer* $\text{Enc}_{k\to l}$ *(Eq.* (9)*) is* $\mathcal{O}(n^{k+l} + n^{2k} + n^{2l})$*.*

Proof. See the supplementary file. □

Thus, in this section, we further analyze the encoder layer and propose a number of optimization and relaxation to reduce the asymptotic complexity with a minimal impact on capability. Notably, combining all our strategies reduces the complexity to $\mathcal{O}(m)$ given a hypergraph with m hyperedges. Even with such efficiency, we show that the reduced version of higher-order Transformer is theoretically more expressive than all message-passing neural networks.

4.1 Linear layers with reduced complexity

A major computation bottleneck within Transformer encoder layer $\text{Enc}_{k\to l}$ is the higher-order linear layer, since it is used for key, query, and feedforward layer. By exploiting only a subset of basis, we show that we can implement lightweight version of linear layer with reduced asymptotic complexity.

Proposition 2. *Given a linear layer $L_{k\to l}$ in Eq.* (1)*, we can always find a nonempty subset $\mathcal{M}$ of equivalence classes such that computation of the following for all $\mathbf{j}$ has $\mathcal{O}(n^k + n^l)$ complexity.*

$$\bar{L}_{k\to l}(\mathbf{A})_{\mathbf{j}} = \sum_{\mu\in\mathcal{M}} \sum_{\mathbf{i}} \mathbf{B}^{\mu}_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}} w_{\mu} + \sum_{\lambda} \mathbf{C}^{\lambda}_{\mathbf{j}} b_{\lambda}, \tag{14}$$

Proof. See the supplementary file. □

We term the reduced linear layer $\bar{L}_{k\to l}$ in Eq. (14) as a *lightweight* linear layer. In practice, we choose $\mathcal{M}$ among equivalence classes that do not involve summation over input. For instance, in case of $L_{1\to 1}$ in Eq. (4), we use only the basis for elementwise mapping (I_n) and drop the one for sum-pooling ($1_n 1_n^\top$). Intuitively, employing such subset of basis in the linear layers of $\text{Enc}_{k\to l}$ is still sufficient to model interactions between input elements, as they are handled by attention (Eq. (12)). Yet, it significantly reduces computational cost as discussed in Proposition 2. Importantly, we can show that this implementation does not hurt Theorem 1, because attention coefficients can still reduce to one and MLP can reduce to invariant bias. We provide exact proof in the supplementary file.

By adopting $\bar{L}_{k\to l}$, the complexity of $L^{\mu}_{k\to u_k}$, $L^{\mu}_{l\to u_q}$, and $\text{MLP}_{l\to l}$ in Eq. (9) reduces to $\mathcal{O}(n^k)$, $\mathcal{O}(n^l)$, and $\mathcal{O}(n^l)$, respectively, and we have $\mathcal{O}(n^{k+l})$ in total for $\text{Enc}_{k\to l}$. As a result, we have higher-order Transformer layers $\text{Enc}_{k\to l}$ that generalize $L_{k\to l}$ while retaining the complexity $\mathcal{O}(n^{k+l})$. From now on we assume that all linear layers for key, query, and MLP are lightweight ($\bar{L}$).

4.2 Sparse Transformer layers

Even with the lightweight linear layers, $\mathcal{O}(n^{k+l})$ complexity of $\text{Enc}_{k\to l}$ is still impractical. For example, in graph domain, complexity larger than $\mathcal{O}(n^2)$ is considered prohibitive while $\text{Enc}_{2\to 2}$ is $\mathcal{O}(n^4)$. Fortunately, it turned out that leveraging the sparsity inherent in real-world graphs can significantly reduce the complexity. As we will show, it reduces the complexity of $\text{Enc}_{k\to l}$ to $\mathcal{O}(m^2)$ for processing a hypergraph with m hyperedges.

Let E a set of hyperedges defined for an input hypergraph G. Each hyperedge is generally represented by a multi-index $\mathbf{i} \in [n]^k$, so we denote $E = \{\mathbf{i}_1, ..., \mathbf{i}_m\}$ with m hyperedges. Leveraging sparsity of E is straightforward when the order of the hypergraph is fixed (e.g., $L_{k\to k}$); we can perform computations with only respect to the existing hyperedges $\mathbf{i}, \mathbf{j} \in E$. However, in our framework, order can change arbitrarily by the equivariant layers (*e.g.*, $L_{k\to l}$), making it difficult to directly transfer E from order-k input to different-order output. To address this problem, we take a simple yet generic heuristic; given the input hyperedges E, we construct proxy hyperedges E' in arbitrary order such that for any $\mathbf{j} \in E'$, there exists $\mathbf{i} \in E$ containing every unique element of $\mathbf{j}$.

Then, we integrate the input hyperedge set E into computation of linear layer in Eq. (14) as follows:

$$L_{k\to l}(\mathbf{A}, E)_{\mathbf{j}} = \begin{cases} \sum_{\mu\in\mathcal{M}} \sum_{\mathbf{i}\in E} \mathbf{B}^{\mu}_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}} w_{\mu} + \sum_{\lambda} \mathbf{C}^{\lambda}_{\mathbf{j}} b_{\lambda} & \text{if } \mathbf{j} \in E' \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

Likewise, integrating E into attention computation in Eq. (10) and Eq. (12) we have:

$$\text{Attn}_{k\to l}(\mathbf{A}, E)_{\mathbf{j}} = \begin{cases} \sum_{h=1}^{H} \sum_{\mu} \sum_{\mathbf{i}\in E} \boldsymbol{\alpha}^{h,\mu}_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}} w^{V}_{h,\mu} w^{O}_{h,\mu} & \text{if } \mathbf{j} \in E' \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

$$\text{where } \boldsymbol{\alpha}^{h,\mu}_{\mathbf{i},\mathbf{j}} = \begin{cases} \sigma(\mathbf{Q}^{h,\mu}_{\mathbf{j}}, \mathbf{K}^{h,\mu}_{\mathbf{i}})/Z_{\mathbf{j}} & \text{if } (\mathbf{i},\mathbf{j}) \in \mu, \mathbf{i} \in E, \mathbf{j} \in E' \\ 0 & \text{otherwise} \end{cases}, \tag{17}$$

where $\mathbf{Q}^{h,\mu} = L^{h,\mu}_{k\to l}(\mathbf{A}, E)$, $\mathbf{K}^{h,\mu} = L^{h,\mu}_{k\to k}(\mathbf{A}, E)$.

With the computations, we can show the following:

Property 2. *When given E with m elements, the equivariant linear layer in Eq.* (15) *has $\mathcal{O}(m)$ complexity, and self-attention computation in Eq.* (16) *has $\mathcal{O}(m^2)$ complexity. Consequently, the computation done by a* $\text{Enc}_{k\to l}$ *composed of the layers has $\mathcal{O}(m^2)$ complexity.*

Proof. See the supplementary file. □

4.3 Kernel attention trick

Section 4.2 shows that, by constraining linear layers within $\text{Enc}_{k\to l}$ to have sparse input and output, we can reduce $\mathcal{O}(n^{k+l})$ complexity to quadratic $\mathcal{O}(m^2)$ to input size. Yet, even $\mathcal{O}(m^2)$ can be demanding with large or dense input. As the quadratic term comes from self-attention computation, we follow the prior work in efficient attention [14, 5, 2] and view attention coefficients as pairwise dot-product scores. As we will show, this allows us to further reduce the complexity of $\text{Enc}_{k\to l}$ to $\mathcal{O}(n^k + n^l)$ and even $\mathcal{O}(m)$ for sparse version, at the cost of relaxing some modeling assumption.

We begin by approximating the attention coefficient in Eq. (12) using the pairwise dot-product kernel [14, 5, 2]:

$$\boldsymbol{\alpha}^{\mu}_{\mathbf{i},\mathbf{j}} = \begin{cases} \phi(\mathbf{Q}^{\mu}_{\mathbf{j}})^{\top}\phi(\mathbf{K}^{\mu}_{\mathbf{i}}) & \text{if } (\mathbf{i},\mathbf{j}) \in \mu \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

where $\phi : \mathbb{R}^{d_H} \to \mathbb{R}^{d_H}_{+}$ is kernel feature map. We discuss the choice of the kernel and its normalization in the supplementary file.

Substituting Eq. (18) in Eq. (10), we have:

$$\begin{aligned} \text{Attn}_{k\to l}(\mathbf{A})_{\mathbf{j}} &= \sum_{\mu}\sum_{\mathbf{i}} \phi(\mathbf{Q}^{\mu}_{\mathbf{j}})^{\top}\phi(\mathbf{K}^{\mu}_{\mathbf{i}})\mathbf{A}_{\mathbf{i}} w^{V}_{\mu} w^{O}_{\mu} \\ &= \sum_{\mu} \phi(\mathbf{Q}^{\mu}_{\mathbf{j}})^{\top} \sum_{\mathbf{i}} \phi(\mathbf{K}^{\mu}_{\mathbf{i}})\mathbf{A}_{\mathbf{i}} w^{V}_{\mu} w^{O}_{\mu}, \end{aligned} \tag{19}$$

Note that the inner-summation in Eq. (19) is over the key index $\mathbf{i}$, which is implicitly *coupled* with the query index $\mathbf{j}$ by $(\mathbf{i},\mathbf{j}) \in \mu$. This coupling causes the major computational bottleneck, since the inner-summation should be computed for every query $\mathbf{Q}^{\mu}_{\mathbf{j}}$. We propose an approximation by decoupling the key and query indices and taking inner-summation over $\mathcal{I} = \bigcup_{\mathbf{j}} \{\mathbf{i}\}, \forall(\mathbf{i},\mathbf{j}) \in \mu$ and share it across different queries $\mathbf{j}$:

$$\text{Attn}_{k\to l}(\mathbf{A})_{\mathbf{j}} \approx \sum_{\mu} \phi(\mathbf{Q}^{\mu}_{\mathbf{j}})^{\top} \sum_{\mathbf{i}\in\mathcal{I}} \phi(\mathbf{K}^{\mu}_{\mathbf{i}})\mathbf{A}_{\mathbf{i}} w^{V}_{\mu} w^{O}_{\mu} \tag{20}$$

Now we show the following:

Property 3. *The encoder $\text{Enc}_{k\to l}$ with approximation in Eq.* (20) *has a complexity of $\mathcal{O}(n^k + n^l)$. Exploiting sparsity further reduces the complexity to $\mathcal{O}(m)$, linear to the number of hyperedges m.*

Proof. See the supplementary file. □

4.4 Comparison to message-passing

We showed that exploiting sparsity and adopting kernel attention reduces the computational complexity of the Transformer encoder to linear to input edges. Considering graphs ($k = l = 2$), this complexity is equivalent to or better than $\mathcal{O}(n + m)$ complexity of the message passing operation[3]. Still, we show that our (approximate) encoder is theoretically more expressive than message passing.

Specifically, we show the following:

Theorem 2. *A composition of two sparse Transformer layers $\text{Enc}_{2\to 2}$ with kernel attention can approximate any message passing algorithms (Gilmer et. al. (2017) [9]) to arbitrary precision, while the opposite is not true.*

Proof. See the supplementary file. □

This leads to the following corollary:

Corollary 1. *Second-order sparse Transformers with kernel attention are more expressive than any message-passing neural networks within the framework of Gilmer et. al. (2017) [9].*

In particular, we note that local information propagation in message-passing GNNs is carried out in $\text{Enc}_{2\to 2}$ by a single $\mu = \{\{i_1\}, \{i_2, i_3, i_4\}\}$. Other types of μ would carry out different operations, which provides intuition on the powerfulness of Transformers.

[3] In practice, we place node features on the diagonals of the adjacency matrix, leading to $\mathcal{O}(n + m)$.

5 Experiments

In this section, we demonstrate the capability of higher-order Transformers on a variety of tasks including synthetic data, real-world graph classification, and set-to-graph prediction. Specifically, we use a synthetic node classification dataset from Gu et. al. (2020) [10], 6 graph classification datasets from Yanadarg et. al. (2015) [34], and 2 set-to-graph prediction datasets from Serviansky et. al. (2020) [25]. Details including the datasets and hyperparameters can be found in the supplementary file. We implemented second-order invariant MLP in Maron et. al. (2019) [20] as one of baselines, which we abbreviate as MLP_2. We build our model by gradually adding lightweight linear layers (Sec. 4.1), sparse linear layers (Sec. 4.2), and kernel attention (Sec. 4.3) and denote them by (D), (S), and (ϕ), respectively.

Synthetic chains To test the ability of higher-order Transformers in modeling long-range interactions in graphs, we use a synthetic dataset where the task is node classification in chain graphs. Binary class information is provided in a terminal node, so a model is required to handle long-range dependency across the chain. We use training and test sets with chains of length 20 and 200 respectively. As baselines, we use 3 message-passing networks, a sparse invariant MLP, and ablated versions of sparse second-order Transformers where μ for global pooling are removed (w/o global).

The test performances are in Table 1. We first note that second-order Transformers successfully capture long-range dependency up to 200 nodes apart, while message-passing networks fail. Importantly, when the subset of basis that accounts for global pooling is eliminated, the performance of Transformers drops similar to message-passing networks, showing their importance in modeling long-range interaction. Yet, simply having global basis is not enough, as seen in the failure of MLP_2. This indicates fine-grained interaction modeling via attention is essential even in this simple task.

Table 1: Chain node classification results.

Method	Micro-F_1 (%)	Macro-F_1 (%)
GCN	37	27.01
GIN-0	63.85	42.55
GAT	38.85	30.30
MLP_2 (S)	37	27
Ours (S) w/o global	38.89	30.80
Ours (S, ϕ) w/o global	38.89	30.80
Ours (S)	**100**	**100**
Ours (S, ϕ)	**100**	**100**

Graph classification To evaluate the ability of Transformers in real-world graph modeling, we used standard graph classification benchmark. We used a total of 6 datasets (3 bioinformatics, 3 social). Following the experimental and evaluation protocols in Xu et. al. (2019) [33], Maron et. al. (2019) [19], we report averaged prediction accuracy and standard deviation from 10-fold cross-validation.

Table 2: Graph classification results.

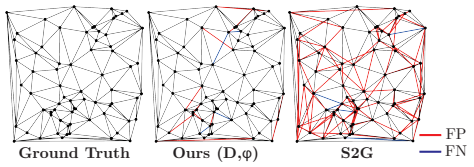
dataset	MUTAG	PTC	PROTEINS	COLLAB	IMDB-B	IMDB-M
GK ([27])	81.39±1.7	55.65±0.5	71.39±0.3	NA	NA	NA
RW ([31])	76±2.7	59.5±2.4	73.68±0.7	NA	NA	NA
PK ([21])	84.11±1.9	57.97±2.5	74.68±0.5	NA	NA	NA
WL ([26])	84.11±1.9	57.97±2.5	74.68±0.5	NA	NA	NA
FGSD ([30])	92.12	62.80	73.42	80.02	73.62	52.41
AWE-DD ([13])	NA	NA	NA	73.93±1.9	74.45±5.8	51.54±3.6
AWE-FB ([13])	87.87±9.7	NA	NA	70.99±1.4	73.13±3.2	51.58±4.6
GCN ([16])	87.20±5.11	NA	75.65±3.24	**81.72±1.64**	73.30±5.29	51.20±5.13
GIN-0 ([33])	89.40±5.60	64.60±7.00	76.20±2.80	80.20±1.90	75.10±5.10	52.30±2.80
GAT ([7])	89.40±6.10	66.70±5.10	74.70±2.20	75.80±1.60	70.50±2.30	47.80±3.10
DGCNN ([38])	85.83±1.7	58.59±2.5	75.54±0.9	73.76±0.5	70.03±0.9	47.83±0.9
PSCN (k=10) ([22])	88.95±4.4	62.29±5.7	75±2.5	72.6±2.2	71±2.3	45.23±2.8
DCNN ([1])	NA	NA	61.29±1.6	52.11±0.7	49.06±1.4	33.49±1.4
ECC ([28])	76.11	NA	NA	NA	NA	NA
DGK ([34])	87.44±2.7	60.08±2.6	75.68±0.5	73.09±0.3	66.96±0.6	44.55±0.5
DiffPool ([35])	NA	NA	78.1	75.5	NA	NA
CCN ([16])	91.64±7.2	**70.62±7.0**	NA	NA	NA	NA
MP ([19])	90.55±8.7	66.17±6.54	77.2±4.73	80.16±1.11	72.6±4.9	50±3.15
MLP_2	88.89±0	58.82±4.56	OOM	OOM	76.3±1.00	51.27±0.63
MLP_2 (S)	88.89±0	65.29±2.56	71.35±0.67	OOM	76.6±1.28	49.93±1.62
Ours (D)	87.22±2.55	OOM	OOM	OOM	OOM	OOM
Ours (S)	**93.33±2.22**	67.06±6.76	OOM	OOM	76.6±2.50	**52.53±1.33**
Ours (S, ϕ)	91.11±2.72	67.65±3.72	**80.81±1.40**	78.78±1.87	**79.8±2.23**	52.33±1.20
Rank	$\mathbf{1^{st}}$	$\mathbf{2^{nd}}$	$\mathbf{1^{st}}$	$\mathbf{5^{th}}$	$\mathbf{1^{st}}$	$\mathbf{1^{st}}$

The results are in Table 2. As stated in Section 3, naïvely implemented second-order Transformers have at least quadratic complexity and, despite high performance, yield out-of-memory error in most settings. On the other hand, sparse second-order Transformer with kernel attention successfully runs in all 6 datasets and achieves better or competitive performance to a wide range of state-of-the-arts. Furthermore, Transformer with kernel attention outperforms invariant MLPs in all datasets, and often achieves better scores than softmax attention presumably because it allows scale-up in model size.

Set-to-graph prediction An important advantage of our framework distinguished from most existing GNNs is that, by design, it can be applied to any learning scenario with different input and output orders (mixed-order). To demonstrate this, we test higher-order Transformers in set-to-graph prediction tasks where the goal is to predict edge structure of a graph given a set of node features. We used two datasets following the prior work [25]. The first dataset Jets originates from particle physics experiments, where collision of high-energy particles gives a set of observed particles. The task is to partition the feature set of observed particles according to their origin. By viewing each subset of particles as a fully-connected graph, the problem is cast as a set-to-graph prediction. For the second dataset Delaunay, the task is to predict Delaunay triangulation [6] given a set of points in 2D space. Two datasets are used for this task, one containing 50 points and the other containing varying number of points $\in \{20, ..., 80\}$. For the baselines, we take the scores reported in Serviansky et. al. (2020) [25], which includes GNNs and invariant MLPs (S2G, S2G+) with $L_{1\to1}$ and $L_{1\to2}$. Our model is made by substituting the linear layers with $\text{Enc}_{1\to1}$ and $\text{Enc}_{1\to2}$ (mixed-order) respectively. Further details including the datasets, metrics, and baselines can be found in the supplementary file.

Table 3: Set-to-graph results. Lower-right panel shows delaunay (20-80) example from ours and S2G.

	Method	F1	RI	ARI		Method	Acc	Prec	Rec	F1
Jets (B)	Ours (D)	0.667	0.746	0.520	Delaunay (50)	Ours (D)	**0.994**	**0.981**	0.967	**0.974**
	Ours (D, ϕ)	**0.670**	**0.751**	**0.526**		Ours (D, ϕ)	0.991	0.967	0.952	0.959
	S2G	0.646	0.736	0.491		S2G	0.984	0.927	0.926	0.926
	S2G+	0.655	0.747	0.508		S2G+	0.983	0.927	0.925	0.926
	GNN	0.586	0.661	0.381		GNN0	0.826	0.384	0.966	0.549
	SIAM	0.606	0.675	0.411		GNN5	0.809	0.363	**0.985**	0.530
	SIAM-3	0.597	0.673	0.396		GNN10	0.759	0.311	0.978	0.471
	MLP	0.533	0.643	0.315		SIAM	0.939	0.766	0.653	0.704
	AVR	0.565	0.612	0.318		SIAM-3	0.911	0.608	0.538	0.570
Jets (C)	Ours (D)	0.755	0.732	0.469	Delaunay (20-80)	Ours (D)	**0.993**	**0.982**	0.960	**0.971**
	Ours (D, ϕ)	**0.757**	**0.735**	**0.473**		Ours (D, ϕ)	0.989	0.948	0.956	0.952
	S2G	0.747	0.727	0.457		S2G	0.947	0.736	0.934	0.799
	S2G+	0.751	0.733	0.467		S2G+	0.947	0.735	0.934	0.798
	GNN	0.720	0.689	0.390		GNN0	0.810	0.387	0.946	0.536
	SIAM	0.729	0.695	0.406		GNN5	0.777	0.352	**0.975**	0.506
	SIAM-3	0.719	0.710	0.421		GNN10	0.746	0.322	0.970	0.474
	MLP	0.686	0.658	0.319		SIAM	0.919	0.667	0.764	0.687
	AVR	0.695	0.650	0.326		SIAM-3	0.895	0.578	0.622	0.587
Jets (L)	Ours (D)	**0.974**	**0.972**	**0.935**						
	Ours (D, ϕ)	**0.974**	**0.972**	**0.935**						
	S2G	0.972	0.970	0.931						
	S2G+	0.971	0.969	0.929						
	GNN	0.972	0.970	0.929						
	SIAM	0.973	0.970	0.925						
	SIAM-3	0.895	0.876	0.729						
	MLP	0.960	0.957	0.894						
	AVR	0.970	0.965	0.922						



The results are outlined in Table 3. Mixed-order Transformers, both softmax and kernel attention, have favorable scores over all baselines. Especially, they outperform all baselines by a large margin in Delaunay: note that GNNs fall into trivial solution with high recall but very low precision. We particularly note that the Transformers' performance in Delaunay (20-80) is comparable to Delaunay (50), with 0.3-0.7% drop in F1 score. Compared with S2G that exhibits $\sim 12\%$ drop, this indicates attention mechanism within $\text{Enc}_{1\to2}$ is helpful in modeling varying number of nodes.

6 Discussion

In this paper, we proposed a generalization of Transformers to higher-orders, and applied a number of design strategies that reduce their complexity to a feasible level. Higher-order Transformers are attractive, both in theory and application. In theoretical aspect, it inherits the theoretical completeness and expressive power of invariant MLPs. In application aspect, it is potentially more powerful than message-passing neural networks due to global interaction modeling, and can be extended to a variety of useful mixed-order tasks involving sets, graphs, and hypergraphs (e.g., mesh).

At the same time, our work has some limitations that need to be addressed in future work: 1) Although complexity to input size can be lowered to linear, the number of basis grows rapidly with increasing order due to $\mathcal{O}((0.792k/\ln(k+1))^k)$ asymptotic formula of k-th Bell number [3], still making the model infeasible in higher orders. Improvement approaches such as finding a compact subset of basis that retains universality [25] and exploiting unorderedness [20] are promising in this direction. 2) Due to the limited support and performance of sparse tensor operations in current state, efficient implementation of sparse higher-order Transformers poses a challenge. Hopefully, this issue will be resolved as better sparse tensor support is added to current deep learning libraries.

References

[1] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In *NeurIPS*, 2016.

[2] I. Bello. Lambdanetworks: Modeling long-range interactions without attention. *arXiv*, 2021.

[3] D. Berend and T. Tassa. Improved bounds on bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics.*, 2010.

[4] C. Cai and Y. Wang. A note on over-smoothing for graph neural networks. *arXiv*, 2020.

[5] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlós, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. Colwell, and A. Weller. Rethinking attention with performers. *arXiv*, 2020.

[6] B. N. Delaunay. Sur la sphère vide. *Bulletin de l'Académie des Sciences de l'URSS, Classe des Sciences Mathématiques et Naturelles. 6: 793–800.*, 1934.

[7] X. Fan, M. Gong, Y. Xie, F. Jiang, and H. Li. Structured self-attention architecture for graph-level representation learning. *Pattern Recognit.*, 2020.

[8] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur. Protein interface prediction using graph convolutional networks. In *NIPS*, 2017.

[9] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[10] F. Gu, H. Chang, W. Zhu, S. Sojoudi, and L. E. Ghaoui. Implicit graph neural networks. In *NeurIPS*, 2020.

[11] B. Hanin and M. Sellke. Approximating continuous functions by relu nets of minimal width. *arXiv*, 2017.

[12] K. Hornik, M. B. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989.

[13] S. Ivanov and E. Burnaev. Anonymous walk embeddings. In *ICML*, 2018.

[14] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*, 2020.

[15] T. N. Kipf, E. Fetaya, K. Wang, M. Welling, and R. S. Zemel. Neural relational inference for interacting systems. In *ICML*, 2018.

[16] R. Kondor, H. T. Son, H. Pan, B. M. Anderson, and S. Trivedi. Covariant compositional networks for learning graphs. In *ICLR*, 2018.

[17] J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *ICML*, 2019.

[18] Q. Li, Z. Han, and X. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

[19] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. In *NeurIPS*, 2019.

[20] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman. Invariant and equivariant graph networks. In *ICLR*, 2019.

[21] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting. Propagation kernels: efficient graph kernels from propagated information. *Mach. Learn.*, 2016.

[22] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, 2016.

[23] K. Oono and T. Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *ICLR*, 2020.

[24] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang. Deepinf: Social influence prediction with deep learning. In *KDD*, 2018.

[25] H. Serviansky, N. Segol, J. Shlomi, K. Cranmer, E. Gross, H. Maron, and Y. Lipman. Set2graph: Learning graphs from sets. In *NeurIPS*, 2020.

[26] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 2011.

[27] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. M. Borgwardt. Efficient graphlet kernels for large graph comparison. In *AISTATS*, 2009.

[28] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *CVPR*, 2017.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[30] S. Verma and Z. Zhang. Hunt for the unique, stable, sparse and fast feature learning on graphs. In *NeurIPS*, 2017.

[31] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. *J. Mach. Learn. Res.*, 2010.

[32] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 2021.

[33] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[34] P. Yanardag and S. V. N. Vishwanathan. Deep graph kernels. In *KDD*, 2015.

[35] Z. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018.

[36] C. Yun, S. Bhojanapalli, A. S. Rawat, S. J. Reddi, and S. Kumar. Are transformers universal approximators of sequence-to-sequence functions? In *ICLR*, 2020.

[37] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola. Deep sets. In *NeurIPS*, 2017.

[38] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *AAAI*, 2018.

[39] Z. Zhang, P. Cui, and W. Zhu. Deep learning on graphs: A survey. *arXiv*, 2018.

[40] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *arXiv*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See the abstract and introduction.
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [No] Due to the lack of space, we will discuss it in the supplementary file.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] See supplementary.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We will provide a code or link to the code in the supplementary file.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See supplementary file.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We reported 10-fold cross validation results in Table 2.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See supplementary file.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See citations.
   (b) Did you mention the license of the assets? [N/A] Not applicable.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] Not applicable.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All datasets used are open-sourced.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] No datasets contain identifiable content.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] Not applicable.
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] Not applicable.
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] Not applicable.