Graph Ordering Attention Networks

Michail Chatzianastasis[1], Johannes Lutzeyer[1], George Dasoulas[1,2], Michalis Vazirgiannis[1]

[1] Ecole Polytechnique, Institut Polytechnique de Paris, France
[2] Harvard University, Cambridge, MA, USA
{michail.chatzianastasis, johannes.lutzeyer}@polytechnique.edu, georgios_dasoulas@hms.harvard.edu, mvazirg@lix.polytechnique.fr

Abstract

Graph Neural Networks (GNNs) have been successfully used in many problems involving graph-structured data, achieving state-of-the-art performance. GNNs typically employ a message-passing scheme, in which every node aggregates information from its neighbors using a permutation-invariant aggregation function. Standard well-examined choices such as the mean or sum aggregation functions have limited capabilities, as they are not able to capture interactions among neighbors. In this work, we formalize these interactions using an information-theoretic framework that notably includes synergistic information. Driven by this definition, we introduce the Graph Ordering Attention (GOAT) layer, a novel GNN component that captures interactions between nodes in a neighborhood. This is achieved by learning local node orderings via an attention mechanism and processing the ordered representations using a recurrent neural network aggregator. This design allows us to make use of a permutation-sensitive aggregator while maintaining the permutation-equivariance of the proposed GOAT layer. The GOAT model demonstrates its increased performance in modeling graph metrics that capture complex information, such as the betweenness centrality and the effective size of a node. In practical use-cases, its superior modeling capability is confirmed through its success in several real-world node classification benchmarks.

1 Introduction

Graph Neural Networks (GNNs) achieve remarkable success in machine learning problems on graphs (Scarselli et al. 2009; Kipf and Welling 2017; Bronstein et al. 2021). In these problems, data arises in the structure of attributed graphs, where in addition to the node and edge sets defining a graph, a set of feature vectors containing data on each node is present. The majority of GNNs learn node representations using a message-passing scheme (Gilmer et al. 2017). In such message passing neural networks (MPNN) each node iteratively aggregates the feature vectors or hidden representations of its neighbors to update its own hidden representation. Since no specific node ordering exists, the aggregator has to be a permutation-invariant function (Xu et al. 2019).

Although MPNNs have achieved great results, they have severe limitations. Their permutation-invariant aggregators treat neighboring nodes as a set and process them individually, omitting potential interactions between the large number of subsets that the neighboring nodes can form. Therefore, current MPNNs cannot observe the entire structure of neighborhoods in a graph (Pei et al. 2020) and cannot capture all *synergistic interactions* between neighbors (Murphy et al. 2019; Wagstaff et al. 2022).

The concept of synergy is important in many scientific fields and is central to our discussion here. It expresses the fact that some source variables are more informative when observed together instead of independently. For example in neuroscience, synergy is observed when the target variable corresponds to a stimulus and the source variables are the responses of different neurons (Bizzi and Cheung 2013). In the Statistics literature (Cox 1984), the modelling interaction of effects, i.e., synergy of independent variables, has also been shown to be advantageous. Synergistic information is often presented in biological cells, where extra information is provided by patterns of coincident spikes from several neurons (Brenner et al. 2000). In gene-gene interactions, synergy is present when the contribution of two mutations to the phenotype of a double mutant is larger than the expected additive effects of the individual mutations (Pérez-Pérez, Candela, and Micol 2009). We believe the consideration of synergistic information to have great potential in the GNN literature.

In this paper, to better understand interactions between nodes, we introduce the Partial Information Decomposition (PID) framework (Williams and Beer 2010) to the graph learning context. We decompose the information that neighborhood nodes have about the central node into three parts: unique information from each node, redundant information, and synergistic information due to the combined information from nodes. We furthermore show that typical MPNNs cannot capture redundant and synergistic information.

To tackle these limitations we propose the *Graph Ordering Attention (GOAT)* layer, a novel architecture that can capture all sources of information. We employ self-attention to construct a permutation-invariant ordering of the nodes in each neighborhood before we pass these ordered sequences to a Recurrent Neural Network (RNN) aggregator. Using a permutation-sensitive aggregator, such as the Long Short-Term Memory (LSTM) model, allows us to obtain larger representational power (Murphy et al. 2019) and to capture the redundant and synergistic information. We further ar-

gue that the ordering of neighbors plays a significant role in the final representation (Vinyals, Bengio, and Kudlur 2016) and demonstrate the effectiveness of GOAT versus other non-trainable and/or permutation-sensitive aggregators with a random ordering (Hamilton, Ying, and Leskovec 2017).

Our main contributions are summarized as follows:

1. We present a novel view of learning on graphs based on information theory and specifically on the Partial Information Decomposition. We further demonstrate that typical GNNs can not effectively capture redundant and synergistic information between nodes.
2. We propose the *Graph Ordering Attention (GOAT)* layer, a novel GNN component that can capture synergistic information between nodes using a recurrent neural network (LSTM) as an aggregator. We highlight that the ordering of the neighbors is crucial for the performance and employ a self-attention mechanism to learn it.
3. We evaluate GOAT in node classification and regression tasks on several real-world and synthetic datasets and outperform an array of state-of-the-art GNNs.

2 Preliminaries and Related Work

We begin by defining our notation and problem context.

Problem Formulation and Basic Notation. Let a graph be denoted by $G = (V, E)$, where $V = \{v_1, \ldots, v_N\}$ is the node set and E is the edge set. Let $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ denote the adjacency matrix, $\boldsymbol{X} = [x_1, \ldots, x_N]^T \in \mathbb{R}^{N \times d_I}$ be the node features and $\boldsymbol{Y} = [y_1, \ldots, y_N]^T \in \mathbb{N}^N$ the label vector. We denote the neighborhood of a vertex u by $\mathcal{N}(u)$ such that $\mathcal{N}(u) = \{v : (v, u) \in E\}$ and the neighborhood features by the multiset $\boldsymbol{X}_{\mathcal{N}(u)} = \{x_v : v \in \mathcal{N}(u)\}$. We also define the neighborhood of u including u as $\overline{\mathcal{N}}(u) = \mathcal{N}(u) \cup \{u\}$ and the corresponding features as $\boldsymbol{X}_{\overline{\mathcal{N}}(u)}$. The goal of semi-supervised node classification and regression is to predict the labels of a test set given a training set of nodes.

Graph Neural Networks. GNNs exploit the graph structure $\boldsymbol{A}$ and the node features $\boldsymbol{X}$ in order to learn a hidden representation h_u of each node u such that the label y_u can be predicted from h_u (Gori, Monfardini, and Scarselli 2005; Scarselli et al. 2009). Most approaches use a neighborhood message-passing scheme, in which every node updates its representation by aggregating the representations of its neighbors and combining them with its own representation,

$$m_u^{(l)} = \text{Aggregate}^{(l)}\left(\left\{h_v^{(l-1)} : v \in \mathcal{N}(u)\right\}\right), \quad (1)$$
$$h_u^{(l)} = \text{Combine}^{(l)}\left(h_u^{(l-1)}, m_u^{(l)}\right),$$

where $h_u^{(l)}$ denotes the hidden representation of node u at the l^{th} layer of the GNN architecture. Note that we often omit the superscript (l) to simplify the notation.

Typically GNNs employ a permutation-invariant "Aggregate" function to yield a permutation-equivariant GNN layer (Bronstein et al. 2021). Permutation invariance and equivariance will be defined formally now.

Definition 2.1. *Let S_M denote the group of all permutations of a set containing M elements. A function $f(\cdot)$ is permutation-equivariant if for all $\pi \in S_M$ we have $\pi f(\{x_1, x_2, .., x_M\}) = f(\{x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(M)}\})$. A function $f(\cdot)$ is permutation-invariant if for all $\pi \in S_M$ we have $f(\{x_1, x_2, .., x_M\}) = f(\{x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(M)}\})$.*

Common Aggregators and Their Limitations

We now describe some of the most well-known aggregators and discuss their limitations. Our analysis is based on two important properties that an aggregator should have:

Relational Reasoning: The label of a node may depend not only on the unique information of each neighbor but also on the joint appearance and interaction of multiple nodes (Wagstaff et al. 2022). With the term "relational reasoning" we describe the property of capturing these interactions, i.e., synergistic information, when aggregating neighborhood messages.

Injectivity: As shown in Xu et al. (2019), a powerful GNN should map two different neighborhoods, i.e., multisets of feature vectors, to different representations. Hence, the aggregator should be injective.

The *mean* and *max* functions are commonly used to aggregate neighborhood information. However, they are neither injective nor able to perform relational reasoning as they process each node independently. The *summation operator followed by a multilayer perceptron* was recently proposed (Xu et al. 2019). This aggregator is injective but cannot perform relational reasoning and usually requires a large latent dimension (Wagstaff et al. 2019, 2022). In the Graph Attention Networks (GAT) (Veličković et al. 2018a), the representation of each node is computed by applying a *weighted summation* of the representations of its neighbors. However, the attention function is not injective since it fails to capture the cardinality of the neighborhood. Recently, an improved version of the GAT was published (Brody, Alon, and Yahav 2022) and also, a new type of attention was proposed (Zhang and Xie 2020), that preserves the cardinality of the neighborhood and therefore is injective. Nevertheless, none of these models can capture interactions between neighbor nodes as each attention score is computed based only on the representations of the central node and one neighbor node. In Section 3 we provide further details on why typical aggregators fail, from an information theoretic perspective.

Permutation-Sensitive Aggregators

Several authors have proposed the use of permutation-sensitive aggregators to tackle the limitations of permutation-invariant aggregators. In particular, Niepert, Ahmed, and Kutzkov (2016) propose to order nodes in a neighborhood according to some labeling, e.g., the betweeness centrality or PageRank score, to assemble receptive fields, possibly extending beyond the 1-hop neighborhood of a central node, which are then fed to a Convolutional Neural Network (CNN) architecture. While this approach demonstrates good performance, it relies on the fixed chosen ordering criterion to be of relevance in the context of a given dataset and chosen learning task. Gao, Wang, and Ji (2018) propose to only work with the k largest hidden state values for each hidden state dimension in each neighborhood.

While not explicitly ordering the neighboring nodes, this operation summarises any given neighborhood in a fixed size feature matrix and enables the use of permutation-sensitive aggregators, CNNs in their case. Of course, the choice of k involves a loss of information in almost all cases, i.e., when k is smaller than the maximal degree in the graph. In the Janossy Pooling (Murphy et al. 2019) approach, a permutation-invariant aggregator is obtained by applying a permutation-sensitive function to all $n!$ permutations. Since the computational cost of this approach is very high, they also propose an approximation, sampling only a limited number of permutations. Similarly, in the GraphSage (Hamilton, Ying, and Leskovec 2017) model, a random permutation of each neighborhood is considered and then passed to an LSTM. However, it has been observed that even in the graph domain, where typically no natural ordering of nodes is known, there exist some orderings that lead to better model performance (Vinyals, Bengio, and Kudlur 2016). Whether these high performance orderings are discovered during the training process is left to chance in the GraphSage and Janossy Pooling models. In contrast, our method learns a meaningful ordering of neighbors with low complexity by leveraging the attention scores.

3 An Information Theory Perspective

In this section, we show how neighborhood dependencies can be encoded in the Partial Information Decomposition framework. This decomposition will motivate us to build a more expressive GNN layer, that is able to capture various interactions among neighborhood nodes.

Partial Information Decomposition

The domain of information theory provides a well-established framework for measuring neighborhood influence. A few graph representation learning results capitalize on information-theoretic tools, either assuming a probability distribution over the feature vectors (Veličković et al. 2018b; Peng et al. 2020) or over the structural characteristics (Luo et al. 2021; Dasoulas et al. 2020).

The majority of GNNs (including the attention-based models) use an aggregation that does not capture interactions among neighbors. Mutual information is a measure that can give us insight into the omitted informative interactions.

Definition 3.1. *For a given node $u \in V$, let $\boldsymbol{H}_{\overline{\mathcal{N}}(u)} = [h_{v_1}, \ldots, h_{v_{|\overline{\mathcal{N}}(u)|}}] \in \mathbb{R}^{|\overline{\mathcal{N}}(u)| \times d}$ denote the hidden representations of the nodes in $\overline{\mathcal{N}}(u)$. Then, if we assume that $\boldsymbol{H}_{\overline{\mathcal{N}}(u)}$ and h_u follow distributions $p(\boldsymbol{H}_{\overline{\mathcal{N}}(u)})$ and $p(h_u)$, respectively, the mutual information between h_u and $\boldsymbol{H}_{\overline{\mathcal{N}}(u)}$ is defined as*

$$I(h_u; \boldsymbol{H}_{\overline{\mathcal{N}}(u)}) = \iint p\left(h_u, \boldsymbol{H}_{\overline{\mathcal{N}}(u)}\right) \log\left(\frac{p\left(h_u, \boldsymbol{H}_{\overline{\mathcal{N}}(u)}\right)}{p(h_u)p\left(\boldsymbol{H}_{\overline{\mathcal{N}}(u)}\right)}\right) dh_u d\boldsymbol{H}_{\overline{\mathcal{N}}(u)}. \quad (2)$$

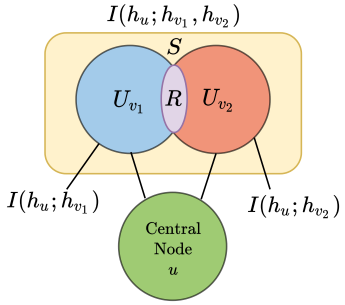

Figure 1: An illustration of the Partial Information Decomposition for the case of one central node u and two neighbors v_1, v_2. Each of the mutual information terms $I(h_u; h_{v_1})$ and $I(h_u; h_{v_2})$ consists of the unique information provided by v_1 (U_{v_1}, blue patch) and v_2 (U_{v_2}, red patch), respectively, as well as the shared information of v_1 and v_2 (R, purple patch). The joint mutual information $I(h_u; h_{v_1}, h_{v_2})$ (yellow box encompassing the inner two circles) consists of four elements: the unique information in the neighbors v_1 and v_2, their redundant information and additionally the synergistic information, $I(h_u; h_{v_1}, h_{v_2}) = U_{v_1} + U_{v_2} + R + S$.

Following Williams and Beer (2010), (2) can be decomposed into three components as follows:

$$I(h_u; \boldsymbol{H}_{\overline{\mathcal{N}}(u)}) = \sum_{v \in \overline{\mathcal{N}}(u)} U_v + R + S, \quad (3)$$

- The *unique information* U_v for all $v \in \overline{\mathcal{N}}(u)$ corresponds to the information a neighbor carries independently and no other neighbor has,
- The *redundant information* R is the information that can be found overlapping in two or more neighbors and
- The *synergistic information* S expresses the information that can be captured only if we take the interactions among neighbors into account.

In Figure 1, we provide an illustration of the PID framework. To exemplify this concept we discuss it in the context of the much-used Cora dataset, for which node feature vectors contain a binary indication of the presence or absence of certain keywords in the abstracts of scientific publications (Sen et al. 2008). For this dataset unique information takes the form of keywords, which are present in only one abstract in a given neighborhood, redundant information refers to keywords, which are repeatedly present without their total number of appearances being of consequence to our learning task, and synergistic information refers to insight that can be gained by observing a certain combination of keywords.

Information Captured by Aggregators

To better understand the information captured by standard GNNs, we first analyze the contribution of each neighboring node to the aggregated representation of a central node.

We assume the structure of an MPNN model, in which each node updates its hidden representation by aggregating information of its neighbors. Further, we denote the message

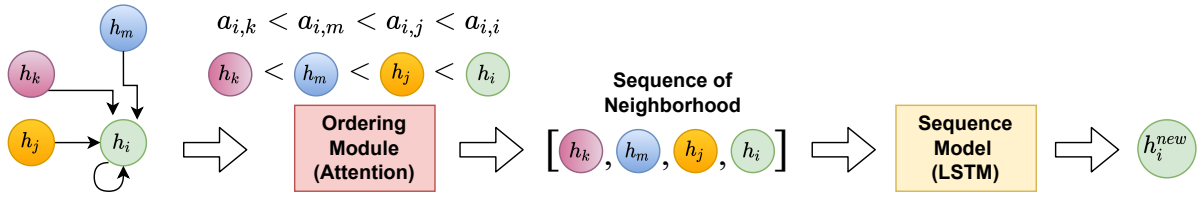

Figure 2: An illustration of a GOAT layer applied to a single node v_i. A self-attention mechanism is used to rank neighborhood nodes. Then, the ordered neighborhood representations are passed to an LSTM producing the updated representation of node v_i.

that a given central node u receives from a neighboring node v by $c_{uv} \in \mathbb{R}^d$. Then, c_{uv} can be interpreted as the *contribution* of node v to the hidden state of u and the aggregated messages in (1) can be expressed as $m_u = \sum_{v \in \mathcal{N}(u)} c_{uv}$.

For the Graph Isomorphism Network (GIN) and Graph Convolutional Network (GCN) we observe that $c_{uv} = f(\boldsymbol{S}_{uv}, h_v) = \boldsymbol{S}_{uv} h_v$, where $\boldsymbol{S} \in \mathbb{R}^{N \times N}$ is a graph shift operator, such as $\boldsymbol{A}$ or $(\boldsymbol{D}+\boldsymbol{I})^{-1/2}(\boldsymbol{A}+\boldsymbol{I})(\boldsymbol{D}+\boldsymbol{I})^{-1/2}$, and $\boldsymbol{D}$ denotes the graph's degree matrix. The contribution of each neighbor v is only determined by its hidden state h_v and the value $\boldsymbol{S}_{uv}$ of the graph shift operator. For the GAT we observe that $c_{uv} = f(h_u, h_v) = a_{uv} h_v$, where a_{uv} is the attention score that is computed from h_u and h_v. The contribution of each neighbor is also affected by the hidden state of the central node, but is not affected by the other neighbors.

We argue that processing each neighbor individually limits its current aggregators, as any interactions among neighbors are ignored by design. Therefore, they can not capture synergistic information between nodes, i.e., the amount of information that is captured equals $\sum_{v \in \mathcal{N}(u)} I(h_u; h_v)$. Consider the example of a neighborhood with two neighbors v_1, v_2. The information captured by a standard GNN is expressed in terms of the PID as follows, $I(h_u; h_{v_1}) + I(h_u; h_{v_2}) = U_{v_1} + U_{v_2} + 2R$, which is different from the joint mutual information $I(h_u; h_{v_1}, h_{v_2}) = U_{v_1} + U_{v_2} + R + S$. Thus, the captured information from a standard GNN is less than the information present in the neighborhood due to the absence of synergistic information.

To address this problem, we introduce a dependence of the contribution c_{uv} of the neighbor node v on all neighbors of u. Therefore, c_{uv} is now a function not only of h_u and h_v, but also of h_j for $j \in \mathcal{N}(u)$, i.e., $c_{uv} = f(\boldsymbol{S}, \boldsymbol{H}_{\overline{\mathcal{N}}(u)})$. To achieve this, we learn a meaningful ordering of the neighbor nodes using an attention mechanism and then use an RNN to aggregate the representations of the neighbors.

4 Graph Ordering Attention Layer

We now present the architecture of our *Graph Ordering Attention (GOAT)* layer and highlight its theoretical advantages over other message-passing models. A deep GNN can be constructed by stacking several GOAT layers or combining GOAT layers with other GNN layers. A GOAT layer (illustrated in Figure 2) consists of two parts:

1) The **Ordering Part** (red box in Figure 2) transforms the unordered multiset of neighbor hidden state vectors, each of dimension d, $\{h_1, \ldots, h_Q\}$, with $Q = \left|\overline{\mathcal{N}}(u)\right|$, into an ordered sequence, using an attention mechanism,

$$[h_{\pi(1)}, \ldots, h_{\pi(Q)}] = \text{OrderingPart}(\{h_1, \ldots, h_Q\}),$$

where the ordering is given by the permutation function $\pi(\cdot)$.

Specifically, similar to the GAT model, for each node $v_i \in V$, we first apply a shared linear transformation parameterized by a *weight matrix* $\mathbf{W_1} \in \mathbb{R}^{d \times d}$ and then perform a shared self-attention mechanism parameterized by $\vec{w}_2 \in \mathbb{R}^{2d}$ to compute the *attention scores*

$$a_{ij} = \text{LeakyReLU}\left(\vec{w}_2^T [\mathbf{W_1} h_i \| \mathbf{W_1} h_j]\right), \tag{4}$$

for all j such that $v_j \in \overline{\mathcal{N}}(v_i)$. Then, we sort the coefficients in decreasing order of magnitude

$$a_{i\pi(1)}, \ldots, a_{i\pi(Q)} = sort\left(a_{i1}, \ldots, a_{iQ}\right), \tag{5}$$

obtaining a specific permutation π of the nodes in the neighborhood. When all attention scores are different from each other, we observe that the sorting function in (5) is deterministic and permutation invariant. In cases where two or more nodes have equal attention scores, we resort to an additional sorting criterion, described in Appendix A, to ensure that our sorting function is deterministic and permutation invariant.

Once we obtain the permutation π, we construct the sorted sequence of neighbourhood hidden states

$$h_{sorted(i)} = \left[\frac{e^{a_{i\pi(1)}}}{\sum_{j=1}^{Q} e^{a_{i\pi(j)}}} \mathbf{W_1} h_{\pi(1)}, \ldots, \frac{e^{a_{i\pi(Q)}}}{\sum_{j=1}^{Q} e^{a_{i\pi(j)}}} \mathbf{W_1} h_{\pi(Q)} \right]. \tag{6}$$

Note that we use the attention scores to both order the hidden states and, after normalization via the softmax function, as coefficients for the hidden states. Only due to the occurrence of the attention coefficients in (6) are we able to obtain gradients in the backpropagation algorithm for $\mathbf{W_1}$ and $\vec{w}_2$ (the sorting function in (5) is not differentiable). Note further that any self-attention mechanism, such as the GATv2 by Brody, Alon, and Yahav (2022), can be used instead of GAT, to obtain the attention scores in a GOAT layer.

2) The **Sequence Modeling Part** (yellow box in Figure 2) takes the ordered sequences of nodes produced by the Ordering Part as input and processes them using an RNN, that is shared across all neighborhoods, to generate the new hidden states. In the PID context, the Bidirectional LSTM (Hochreiter and Schmidhuber 1997) appears to be the best suited RNN available. Its forget gate allows us to discard redundant information; the input gate is sufficiently expressive to isolate unique information; while its memory states allow for the identification of synergistic information.

$$h_i^{new} = \text{LSTM}\left(h_{sorted(i)}\right) \in \mathbb{R}^{d_O}. \tag{7}$$

Since we utilize a Bidirectional LSTM the contribution of each node, discussed in Section 3, depends on all other hidden states in the neighborhood. Specifically, each direction in the Bidirectional LSTM ensures that both the nodes preceding and succeeding a particular node j are taken into account when calculating the contribution c_{ij} of node j.

Note that the choice of the LSTM is made without loss of generality and any RNN could be chosen in the Sequence Modeling Part. Indeed, in Section 5 we will observe results for a variety of RNNs chosen as part of our GOAT layer. To work with a faster, more scalable implementation we pad all neighborhood sequences with zero vectors to be equal in length to the sequence of hidden states arising in the largest neighborhood in the graph. This allows us to train the LSTM on larger batches of neighborhoods in parallel. The alternative implementation, where neighborhood sequences of different lengths are fed to the LSTM individually is equally valid, while slower, implementation.

Multi-Head Attention Ordering. We can also employ multi-head attention to provide additional representational power to our model. We see several advantages in the consideration of multiple heads in our architecture. If only one sensible ordering of the nodes in a neighborhood exists, then multiple heads can help us estimate this ordering more robustly. If on the other hand, there exist several sensible orderings of the nodes in a neighborhood, then a multi-head architecture allows us to take all of these into account in our model. Let K be the number of the attention heads. Equation (4) for the k-th attention head is transformed as

$$a_{ij}^k = a^k(\mathbf{W_1^k}h_i, \mathbf{W_1^k}h_j).$$

Then we sort the K sets of attention scores obtaining multiple orderings of the neighborhood, $h^k_{sorted(i)}$ for $k \in \{1, \ldots, K\}$. To generate the final representation of the nodes we concatenate the features from the K independent Bidirectional LSTM models, i.e.,

$$h_i^{new} = \Big\Vert_{k=1}^{K} \text{LSTM}^k\left(h^k_{sorted(i)}\right).$$

Complexity. The time complexity of a single-head GOAT layer is $\mathcal{O}(|V|d_O d + |E|d_O + |V|d_{\max}\log(d_{\max}) + |V|d_{\max}4d(d_O + d + 3))$, where $d_{\max}$ denotes the maximal degree in the graph. For $d_{\max} \ll d$, the only additional complexity introduced by our model manifests in the multiplicative $d_{\max}$ term in the last summand of the complexity expression. Limiting the maximal degree by applying a sampling strategy, limits the additional complexity our model introduces. Hence, the time complexity of our GOAT model can be comparable to standard MPNNs models. Note that the space complexity of a GOAT layer only exceeds the space complexity of a GAT layer by the space complexity of the LSTM model. A more detailed discussion of the complexity of our model can be found in Appendix B.

Permutation-Equivariance and Injectivity of GOAT. Recall from Section 2 that the permutation-equivariance and injectivity are desirable properties for a GNN layer to have. We will now prove that our GOAT layer satisfies both of these criteria.

Proposition 4.1 (Permutation-Equivariance of GOAT)**.** *Our GOAT layer performs a permutation-equivariant transformation, with respect to node label permutations, of the hidden states corresponding to the nodes in a graph.*

Hence, our GOAT layer benefits from the expressivity of a permutation-sensitive aggregator, while acting on the node's hidden representations in a permutation-equivariant way. Our result on the injectivity of a GOAT layer relies on the concept of function approximation in probability. We reproduce the definition of this concept from Hammer (2000).

Definition 4.1. *Let $\mathcal{X}$ denote the space of finite lists with elements in $\mathbb{R}^q$ for $q \in \mathbb{N}$ and P be a probability measure on $\mathcal{X}$. For measurable functions $f_1, f_2 : \mathcal{X} \to \mathbb{R}^t$ we say that f_1 approximates f_2 with accuracy $\epsilon > 0$ and confidence $\delta > 0$ in probability if $P(x \in \mathcal{X} \mid |f_1(x) - f_2(x)| > \epsilon) < \delta$.*

Theorem 1 (Injectivity of GOAT)**.** *Assume that for all nodes $u \in V$ the multisets of hidden states corresponding to its neighbors are finite and have elements in $\mathbb{R}^q$ for $q \in \mathbb{N}$. Then, there exists a GOAT layer approximating a measurable function arbitrarily well in probability for which any two distinct multisets are mapped to distinct node representations.*

Therefore, we have shown that our GOAT layer is sufficiently expressive to approximate any measurable injective function in probability. The proofs of Proposition 4.1 and Theorem 1 are in Appendices C and D, respectively.

5 Experimental Evaluation

We perform an extensive evaluation of our GOAT model and compare against a wide variety of state-of-the-art GNNs, on three synthetic datasets as well as on nine node-classification benchmarks. Our code is publicly available on GitHub[1].

Baselines. We compare GOAT against the following state-of-the-art GNNs for node classification: 1) GCN the classical graph convolution neural network, 2) GraphSAGE(mean) that aggregates by taking the elementwise mean value, 3) GraphSAGE(lstm) that aggregates by feeding the neighborhood hidden states in a random order to an LSTM, 4) GIN the injective summation aggregator, 5) GAT that aggregates with a learnable weighted summation operation, 6) PNA (Corso et al. 2020) that combines multiple aggregators with degree-scalers. We also compare with 7) a standard MLP that only uses node features and does not incorporate the graph structure. To better understand the impact of the choice of RNN in the GOAT architecture, we

[1]Code: https://github.com/MichailChatzianastasis/GOAT

Method	Top-2 pooling (Accuracy)	Betweenness Centrality (MSE)		Effective Size (MSE)	
		N=100, p=0.09	N=1000, p=0.01	N=100, p=0.09	N=1000, p=0.01
GCN	57.35 ±4.13	0.0063 ±0.0036	0.0020 ±0.0008	0.0135 ±0.0067	0.00380 ±0.00120
GraphSAGE (mean)	61.45 ±5.79	0.0401 ±0.0158	0.0221 ±0.0069	0.0374 ±0.0085	0.02430 ±0.00560
GraphSAGE (lstm)	65.05 ±8.71	0.0094 ±0.0073	0.0153 ±0.0105	0.0022 ±0.0017	0.00080 ±0.00020
GIN	56.40 ±5.26	0.0083 ±0.0052	0.0042 ±0.0015	0.0024 ±0.0016	0.00070 ±0.00030
GAT	53.34 ±2.43	0.0409 ±0.0158	0.0220 ±0.0068	0.0382 ±0.0079	0.02480 ±0.00560
PNA	61.50 ±10.9	0.0115 ±0.0089	0.0020 ±0.0008	0.0121 ±0.0119	0.00137 ±0.00035
GOAT(lstm)	**69.21** ±5.10	**0.0038** ±0.0019	**0.0006** ±0.0002	**0.0016** ±0.0008	**0.00020** ±0.00008

Table 1: Classification accuracy (± standard deviation) on the "Top-2 pooling" synthetic dataset and MSE (± standard deviation) results on the synthetic datasets "Betweenness Centrality" and "Effective Size" for two different types of random graphs.

provide results from three different GOAT architectures, in which the standard RNN, GRU (Cho et al. 2014) and LSTM are used.

Setup. For a fair comparison we use the same training process for all models adopted by Veličković et al. (2018a). We use the Adam optimizer (Kingma and Ba 2015) with an initial learning rate of 0.005 and early stopping for all models and datasets. We perform a hyperparameter search for all models on a validation set. The hyperparameters include the size of hidden dimensions, dropout, and number of attention heads for GAT and GOAT. We fix the number of layers to 2. In our experiments, we combine our GOAT layer with a GAT or GCN layer to form a 2-layer architecture. More information about the datasets, training procedure, and hyperparameters of the models is in Appendix E.

Top-2-Pooling

In this task, we sample Erdős–Rényi random graphs with 1000 nodes and a probability of edge creation of 0.01. We draw 1-dimensional node features from a Gaussian Mixture model with three equally weighted components with means $1, 1$ and 2 and standard deviations $1, 4$ and 1. We label each node with a function $\phi(\cdot, \cdot)$ of the two 1- or 2-hop neighbors that have the two different largest features, i.e., to each node $u \in V$ we assign a label $y_u = \phi(x_a, x_b)$, where x_a and x_b are the largest, distinct node features of all nodes in the 2-hop neighborhood of u with nodes features at a distance of 2 being down-weighted by a factor of 0.8. We set ϕ to be $\phi(x_a, x_b) = \sqrt{exp(x_a) + exp(x_b)}$. Finally, to transform this task to node classification we bin the y values into two equally large classes. We use 60/20/20 percent of nodes for training, validation and testing.

We report the average classification accuracy and the standard deviation across 10 different random graphs in Table 1. *Our GOAT model outperforms the other GNNs by a large margin.* Specifically, our model leads to an 18.36% increase in performance over GAT and 6.65% increase over GraphSage(lstm). We explain this performance gap with the following hypothesis. To find the largest element of a set one must consider 2-tuple relationships therefore synergistic information is crucial for this task. An LSTM can easily perform the necessary comparisons with a 2-dimensional hidden space. As nodes are processed they can either be discarded via the forget gate, if they are smaller than the current hidden state, or the hidden state is updated to contain the new feature node. In contrast, typical GNNs need exponentially large hidden dimensions in order to capture the necessary information as they cannot efficiently discard redundant information. We observe that GraphSage(lstm) is the second-best performing model due to its LSTM aggregator. However, it does not learn a meaningful ordering of the nodes that simplifies this task.

Prediction of Graph Structural Properties

The experiments in this section establish the ability of our GOAT model to predict structural properties of nodes. The first task is to predict the *betweenness centrality* of each node and the second task is to predict the *effective size* of each node. Both of these metrics, defined in Appendix E, are affected by the interactions between the neighbor nodes, so synergistic information is crucial for these tasks. We set the input features to the identity matrix, i.e., $\mathbf{X} = \mathbf{I}$ and use two parameter settings to sample Erdős–Rényi random graphs, namely $(N, p) \in \{(100, 0.09), (1000, 0.1)\}$, where N is the number of nodes and p is the probability of edge creation. We use 60% of nodes for training, 20% for validation and 20% for testing. We train the models by minimizing the Mean Squared Error (MSE).

We report the mean and standard deviation accuracy and MSE across 10 graphs of each type in Table 1. Our model outperforms all models in both tasks and in both graph parameter settings. GOAT can capture the synergistic information between the nodes, which is crucial for predicting the betweenness centrality and effective size. The other aggregators miss the structural information of nodes in neighborhoods. We observe that GraphSAGE(lstm) that uses a random node ordering is not on par with GOAT, indicating that the learned ordering in GOAT is valuable here also.

Node Classification Benchmarks

We utilize nine well-known node classification benchmarks to validate our proposed model in real-world scenarios originating from a variety of different applications. Specifically, we use 3 citation network benchmark datasets: Cora, CiteSeer (Sen et al. 2008), ogbn-arxiv (Hu et al. 2020), 1 disease spreading model: Disease (Chami et al. 2019), 1 social network: LastFM Asia (Rozemberczki and Sarkar 2020), 2 co-purchase graphs: Amazon Computers, Amazon Photo (Shchur et al. 2019) and 2 co-authorship graphs: Coauthor CS,Physics (Shchur et al. 2019). For the GOAT

Method	Cora	CiteSeer	Disease	LastFM Asia	Computers	Photo	CS	Physics
MLP	43.8	52.9	79.10 ±0.97	72.27 ±1.00	79.53 ±0.66	87.89 ±1.04	93.76 ±0.26	95.85 ±0.20
GCN	81.4	67.5	88.98 ±2.21	83.58 ±0.93	90.72 ±0.50	93.99 ±0.42	92.96 ±0.32	96.27 ±0.22
GraphSAGE (mean)	77.2	65.3	88.79 ±1.95	83.07 ±1.19	91.47 ±0.37	94.32 ±0.46	94.11 ±0.30	96.31 ±0.22
GraphSAGE (lstm)	74.1	59.9	90.50 ±2.15	**86.85** ±1.07	91.26 ±0.51	94.32 ±0.64	93.46 ±0.29	96.40 ±0.16
GIN	75.5	62.1	90.20 ±2.23	82.94 ±1.25	84.68 ±2.33	90.07 ±1.19	92.38 ±0.38	96.38 ±0.16
GAT	83.0	69.3	89.13 ±2.22	77.57 ±1.82	85.41 ±2.95	90.30 ±1.76	92.78 ±0.27	96.17 ±0.18
PNA	76.4	58.9	86.84 ±1.89	83.24 ±1.10	90.80 ±0.51	94.35 ±0.68	91.83 ±0.33	96.25 ±0.21
GOAT(lstm)	**84.9**	69.5	**92.11** ±1.88	83.29 ±0.91	91.34 ±0.50	**94.38** ±0.66	**94.21** ±0.42	**96.69** ±0.31
GOAT(gru)	83.5	**70.0**	91.97 ±1.90	83.35 ±0.91	**91.54** ±0.48	94.22 ±0.58	93.62 ±0.22	96.32 ±0.24
GOAT(rnn)	84.2	67.9	91.67 ±1.69	83.21 ±0.98	89.10 ±0.51	92.45 ±0.60	93.48 ±0.19	96.44 ±0.20

Table 2: Node classification accuracy using different train/validation/test splits. We highlight the best-performing model and underline the second-best. Since there exists a single standardized split for Cora and CiteSeer no standard deviations are given.

Method	ogbn-arxiv
GCN	33.3 ±1.2
GraphSAGE	54.6 ±0.3
GAT	54.1 ±0.5
GOAT(lstm)	**55.1** ±0.4

Table 3: Node classification accuracy on the ogbn-arxiv dataset. We used the same setup and the reported results from Kim and Oh (2022).

Method	Cora	Disease
GOAT	83.36 ±0.42	90.64 ±0.40
GOAT-fixed-0	81.56 ±0.19	90.48 ±0.29
GOAT-fixed-100	**83.80** ±1.14	**90.90** ±0.26
GOAT-fixed-200	82.13 ±0.27	90.57 ±0.59
GOAT-fixed-500	82.27 ±0.34	90.50 ±0.49

Table 4: Accuracy of the GOAT model using fixed orderings extracted from different epochs of the baseline model's training.

results on the ogbn-arxiv dataset we randomly sample 100 neighbors per node to represent the neighborhoods for faster computation. We report the classification accuracy results in Tables 2 and 3. Our model outperforms the others in eight of nine datasets.

Ablation Studies on the Learned Ordering

In our GOAT architecture, we make the implicit assumption that ordering neighborhoods by the magnitude of the trainable attention scores is an ordering that results in a well-performing model. We now perform several ablation studies where we compare the GOAT model to models with fixed neighborhood node orderings (GOAT-fixed).

We train our GOAT model on the Cora and Disease datasets, using 8 attention heads and the LSTM aggregator. We store the ordering of the nodes in each neighborhood for each attention head for each epoch. Then, we train various (GOAT-fixed) models that use different fixed orderings extracted from the initial model. Specifically, we train 4 different models with orderings extracted from the $0, 100, 200, 500$ epochs, respectively. We run the experiment 3 times and report the results in Table 4. We observe that GOAT-fixed-0, which uses random ordering, since the ordering is extracted before training, achieves the worst performance. This highlights the importance of a meaningful ordering of the nodes, and the ability of our model to learn one. We also observe that the fixed ordering extracted from epoch 100 outperforms the GOAT model. We believe this phenomenon to be associated with the training dynamics of our model. Having a fixed ordering may lead to more stability in the learning of high-performing model parameters not associated with the ordering. Learning an ordering in the first run of our model and then training with a fixed ordering may therefore be most advisable.

We perform additional ablation studies in Appendix F. Firstly, we investigate the potential use of the GATv2 model instead of the GAT model in a GOAT layer and find that the two model variants perform comparably. Secondly, we observe the GOAT model to significantly outperform the Janossy Pooling approach on the Cora, CiteSeer and Disease datasets. We further examine the impact of the number of attention heads in our model. We observe one attention head to yield optimal performance on Cora, four attention heads are optimal for CiteSeer, while eight attention heads resulted in the best performing model in the Disease dataset.

6 Conclusion

We have presented a novel view of learning on graphs by introducing the Partial Information Decomposition to the graph context. This has allowed us to identify that current aggregation functions used in GNNs often fail to capture synergistic and redundant information present in neighborhoods. To address this issue we propose the Graph Ordering Attention layer, which makes use of a permutation-sensitive aggregator capable of capturing synergistic and redundant information, while maintaining the permutation-equivariance property. The GOAT layer is implemented by first learning an ordering of nodes using a self-attention mechanism and by then applying an RNN to the ordered representations. This theoretically grounded architecture yields improved accuracy in the node classification and regression tasks on both synthetic and real-world networks.

Acknowledgments

The work of Dr. Johannes Lutzeyer and Prof. Michalis Vazirgiannis is supported by the ANR chair AML-HELAS (ANR-CHIA-0020-01).

References

Bizzi, E.; and Cheung, V. C. 2013. The neural origin of muscle synergies. *Frontiers in computational neuroscience*, 7: 51.

Brenner, N.; Strong, S.; Koberle, R.; Bialek, W.; and Steveninck, R. 2000. Synergy in a Neural Code. *Neural computation*, 12: 1531–52.

Brody, S.; Alon, U.; and Yahav, E. 2022. How Attentive are Graph Attention Networks? In *International Conference on Learning Representations (ICLR)*.

Bronstein, M. M.; Bruna, J.; Cohen, T.; and Veličković, P. 2021. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv:2104.13478*.

Chami, I.; Ying, R.; Ré, C.; and Leskovec, J. 2019. Hyperbolic Graph Convolutional Neural Networks. *Advances in neural information processing systems (NeurIPS)*.

Cho, K.; van Merriënboer, B.; Bahdanau, D.; and Bengio, Y. 2014. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 103–111.

Corso, G.; Cavalleri, L.; Beaini, D.; Liò, P.; and Veličković, P. 2020. Principal Neighbourhood Aggregation for Graph Nets. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Cox, D. R. 1984. Interaction. *International Statistical Review/Revue Internationale de Statistique*, 1–24.

Dasoulas, G.; Nikolentzos, G.; Scaman, K.; Virmaux, A.; and Vazirgiannis, M. 2020. Ego-based Entropy Measures for Structural Representations. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Gao, H.; Wang, Z.; and Ji, S. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 1416–1424.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning (ICML)*, 1263–1272. PMLR.

Gori, M.; Monfardini, G.; and Scarselli, F. 2005. A new model for learning in graph domains. In *IEEE international joint conference on neural networks (IJCNN)*, 729–734.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems (NIPS)*, 30.

Hammer, B. 2000. On the approximation capability of recurrent neural networks. *Neurocomputing*, 31(1-4): 107–123.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural computation*, 9(8): 1735–1780.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33: 22118–22133.

Kim, D.; and Oh, A. 2022. How to find your friendly neighborhood: Graph attention design with self-supervision. *International Conference on Learning Representations (ICLR)*.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

Luo, G.; Li, J.; Peng, H.; Yang, C.; Sun, L.; Yu, P. S.; and He, L. 2021. Graph Entropy Guided Node Embedding Dimension Selection for Graph Neural Networks. *arXiv:2105.03178*.

Murphy, R. L.; Srinivasan, B.; Rao, V.; and Ribeiro, B. 2019. Janossy Pooling: Learning Deep Permutation-Invariant Functions for Variable-Size Inputs. In *International Conference on Learning Representations (ICLR)*.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning (ICML)*, 2014–2023. PMLR.

Pei, H.; Wei, B.; Chang, K. C.-C.; Lei, Y.; and Yang, B. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

Peng, Z.; Huang, W.; Luo, M.; Zheng, Q.; Rong, Y.; Xu, T.; and Huang, J. 2020. Graph representation learning via graphical mutual information maximization. In *Proceedings of The Web Conference (WWW)*, 259–270.

Pérez-Pérez, J. M.; Candela, H.; and Micol, J. L. 2009. Understanding synergy in genetic interactions. *Trends in Genetics*, 25(8): 368–376.

Rozemberczki, B.; and Sarkar, R. 2020. Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models. In *Proceedings of the 29th ACM international conference on information & knowledge management (CIKM)*, 1325–1334.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1): 61–80.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective Classification in Network Data. *AI Magazine*, 29(3): 93.

Shchur, O.; Mumme, M.; Bojchevski, A.; and Günnemann, S. 2019. Pitfalls of Graph Neural Network Evaluation. In *R2L Workshop at NeurIPS*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018a. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*.

Veličković, P.; Fedus, W.; Hamilton, W. L.; Liò, P.; Bengio, Y.; and Hjelm, R. D. 2018b. Deep Graph Infomax. In *International Conference on Learning Representations (ICLR)*.

Vinyals, O.; Bengio, S.; and Kudlur, M. 2016. Order Matters: Sequence to sequence for sets. In *International Conference on Learning Representations (ICLR)*.

Wagstaff, E.; Fuchs, F.; Engelcke, M.; Posner, I.; and Osborne, M. A. 2019. On the limitations of representing functions on sets. In *International Conference on Machine Learning (ICML)*, 6487–6494. PMLR.

Wagstaff, E.; Fuchs, F. B.; Engelcke, M.; Osborne, M. A.; and Posner, I. 2022. Universal approximation of functions on sets. *Journal of Machine Learning Research*, 23(151): 1–56.

Williams, P. L.; and Beer, R. D. 2010. Nonnegative Decomposition of Multivariate Information. *arXiv:1004.2515*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations (ICLR)*.

Zhang, S.; and Xie, L. 2020. Improving attention mechanism in graph neural networks via cardinality preservation. In *IJCAI: Proceedings of the Conference*, volume 2020, 1395. NIH Public Access.