Graph Ordering Attention Networks

Michail Chatzianastasis[1*], **Johannes Lutzeyer**[1], **George Dasoulas**[1] and **Michalis Vazirgiannis**[1,2]
[1] DaSciM, LIX, Ecole Polytechnique, Institute Polytechnique de Paris, France
[2]Athens University of Economics and Business (AUEB), Athens, Greece
{michail.chatzianastasis, johannes.lutzeyer}@polytechnique.edu,
george.dasoulas1@gmail.com,
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 like 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 higher-level dependencies 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) have shown remarkable success in solving 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]. Specifically, in such a message passing neural network (MPNN) each node iteratively aggregates the feature vectors or hidden representations of its neighbors to update its own hidden representation. Since there is no specific node ordering, the aggregator has to be a permutation-invariant function such as max or sum pooling [Xu *et al.*, 2019].

Although MPNNs have achieved great results, they have severe limitations in learning on graph structures, as they cannot capture the full range of complex relations between the nodes in a given neighborhood. Their permutation-invariant aggregators treat the 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 dependencies and interactions between them [Murphy *et al.*, 2019; Wagstaff *et al.*, 2021].

In this paper, to better understand these interactions between the nodes in a node classification scenario, we introduce the Partial Information Decomposition (PID) framework [Williams and Beer, 2010] to the graph learning context. We decompose the information that neighbor nodes have about the central node, into three parts: unique information from each node, redundant information, and synergistic information due to combined information from multiple nodes. The concept of synergy expresses the fact that some source variables give more information when observed together than they do when observed independently. Synergy is observed in neuroscience where the target variable corresponds to a stimulus and the source variables are the responses of different neurons [Bizzi and Cheung, 2013]. We show that typical MPNNs cannot capture redundant and synergistic information, leading to insufficient node representations.

To tackle these limitations we propose the *Graph Ordering Attention (GOAT)* layer, a novel and simple architecture that can capture all sources of information, notably including the synergistic information between the nodes. We employ a self-attention to construct a permutation-invariant ranking of the nodes in each neighborhood before we pass the ordered sequence into a recurrent neural network aggregator. Using a permutation-sensitive aggregator like a Long Short-Term Memory (LSTM) model, we obtain larger representational power [Murphy *et al.*, 2019] and are able to capture the redundant and synergistic information since we do not assume

*Contact Author

independence between the nodes as other aggregators do. We further argue that the ordering of the neighbors plays a significant role in the final representation [Vinyals *et al.*, 2016] and demonstrate the effectiveness of GOAT versus other aggregators that are non-trainable or use a permutation-sensitive aggregator with a random ordering [Hamilton *et al.*, 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 (PID). We argue that capturing redundant and synergistic information between the nodes leads to more accurate node representations.
2. We propose the *Graph Ordering Attention (GOAT)* layer, a novel GNN component that can capture the synergistic information between the 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 empirically evaluate GOAT on real-world and synthetic node classification and regression datasets and compare with state-of-the-art GNNs. Our results confirm that GOAT can learn accurate node representations and be state-of-the-art on many benchmarks.

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 $V = \{v_1, v_2, \ldots, v_N\}$ is the node set and E is the edge set. Let $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ denote the adjacency matrix, $\boldsymbol{X} = [x_1, x_2, \ldots, x_N]^T \in \mathbb{R}^{N \times d}$ be the node features and $\boldsymbol{Y} = [y_1, y_2, \ldots, y_N]^T \in \mathbb{N}^N$ the label vector. We denote the neighborhood of a vertex v by $\mathcal{N}(v)$ such that $\mathcal{N}(v) = \{u : (u, v) \in E\}$ and the neighborhood features by the multiset $\boldsymbol{X}_{\mathcal{N}(v)} = \{x_u : u \in \mathcal{N}(v)\}$. We also define the neighborhood of v including v as $\overline{\mathcal{N}}(v) = \mathcal{N}(v) \cup \{v\}$ and the corresponding features as $\boldsymbol{X}_{\overline{\mathcal{N}}(v)}$. 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 accurately from h_u [Gori *et al.*, 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 previous representation,

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

where $h_u^{(l)}$ denotes the feature vector 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. Then, a function $f(\cdot)$ is

- *permutation-equivariant* if for all $\pi \in S_M$ we have $\pi f(\{x_1, x_2, .., x_M\}) = f(\{x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(M)}\})$.
- *permutation-invariant* if for all $\pi \in S_M$ we have $f(\{x_1, x_2, .., x_M\}) = f(\{x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(M)}\})$.

2.1 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:

1. **Relational Reasoning**: As we will show in Section 3, the label of the node may depend not only on the individual contribution of each neighbor but also on the fact that multiple nodes appear together and interact [Wagstaff *et al.*, 2021]. With the term "relational reasoning" we describe the property of capturing these interactions when we aggregate the neighborhood messages. We refer to these interactions as "synergistic information."
2. **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. This means that the aggregator should be injective.

The *mean* and *max* are standard pooling functions that are used to aggregate neighborhood information [Kipf and Welling, 2017]. However, they are neither injective [Xu *et al.*, 2019] nor can they 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 still cannot perform relational reasoning and it usually requires a large latent dimension [Wagstaff *et al.*, 2019; Wagstaff *et al.*, 2021].

In the Graph Attention Networks (GAT) [Veličković *et al.*, 2018a], the representation of each node is computed by applying a *weighted summation* of the node representations. 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 *et al.*, 2021] 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 the interaction between two or more neighbor nodes as each attention score is computed based only on the representations of the central node and one neighbor node.

2.2 Permutation Sensitive Aggregators

A few authors propose the use of permutation-sensitive aggregators to tackle the limitations of typical permutation-invariant pooling operators. For example 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 *et al.*, 2017] model,

a random permutation of each neighborhood is considered and then passed to an LSTM. This method allows them to demonstrate 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 *et al.*, 2016]. Whether these high performance orderings are discovered during the training process is left to chance however. In contrast, our method learns a meaningful ordering of neighbors with low complexity by leveraging the attention weights.

3 An Information Theory Perspective for Graphs

In this section, we show how the 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 subsets.

3.1 Partial Information Decomposition for Graphs

The domain of information theory can provide a well-established framework for measuring the neighborhood influence. A few findings on graph representation learning capitalized on information-theoretic tools, either assuming a probability distribution over the attribute 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 omits the information among the neighbors, e.g connected triplets, quadruples and n-tuples. Mutual information is a measure that can give us insight in the omitted informative interactions.

Definition 3.1 (Structure-Informed Mutual Information)**.** 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 $\overline{\mathcal{N}}(u)$. Let, also, $\boldsymbol{A}_S$ denote the adjacency matrix of the subgraph induced by $\overline{\mathcal{N}}(u)$ and the *structure-informed* representation $\boldsymbol{H}^G_{\overline{\mathcal{N}}(u)} = \begin{bmatrix} \boldsymbol{H}_{\overline{\mathcal{N}}(u)} \\ \boldsymbol{A}_S \end{bmatrix} \in \mathbb{R}^{(d+|\overline{\mathcal{N}}(u)|) \times |\overline{\mathcal{N}}(u)|}$ denote the augmented neighbors representation including edge information. Then, if we assume that $\boldsymbol{H}^G_{\overline{\mathcal{N}}(v)}$ and h_v follow distributions $p(\boldsymbol{H}^G_{\overline{\mathcal{N}}(v)})$ and $p(h_v)$, respectively, the mutual information between h_v and $\boldsymbol{H}^G_{\overline{\mathcal{N}}(v)}$ is defined as

$$I(h_v; \boldsymbol{H}^G_{\overline{\mathcal{N}}(v)}) = \iint p\left(h_v, \boldsymbol{H}^G_{\overline{\mathcal{N}}(v)}\right) \log\left(\frac{p\left(h_v, \boldsymbol{H}^G_{\overline{\mathcal{N}}(v)}\right)}{p(h_v)p\left(\boldsymbol{H}^G_{\overline{\mathcal{N}}(v)}\right)}\right) dh_v d\boldsymbol{H}^G_{\overline{\mathcal{N}}(u)}. \quad (1)$$

Following [Williams and Beer, 2010], (1) can be decomposed into three components as follows:

$$I(h_v; \boldsymbol{H}^G_{\overline{\mathcal{N}}(v)}) = \sum_{u \in \overline{\mathcal{N}}(v)} U_u + R + S, \quad (2)$$

where

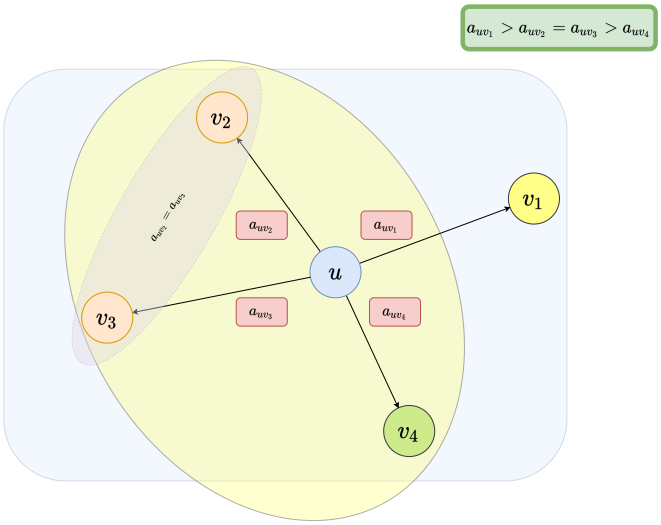

Figure 1: In the neighborhood of node u, the attention scores a_{uv_j}'s contribute to the unique information terms U_{v_j}'s. The relationships among the attention scores of the neighbors contribute to the synergy information S. The yellow circle includes dependencies among u_1, u_2 and u_3, and the light blue curved square refers to all neighbors. u_2, u_3 share the same representation, contributing to the redundant information.

- the *unique information* $U_u \, \forall u \in \overline{\mathcal{N}}(v)$ 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 (e.g. in identical feature vectors) and
- the *synergistic information* S expresses the rest of the information that can be captured only if we take the interactions among the neighbors into account.

To exemplify this concept we choose to discuss it in the context of the much-used Cora dataset, for which node feature vectors contain binary indication of the presence or absence of certain key words in the abstracts of scientific publications [Sen *et al.*, 2008]. For this dataset unique information takes the form of key words, which are present in only one abstract in a given neighborhood, redundant information refers to keywords, which are repeatedly present, and synergistic information refers to insight that can be gained by observing a certain combination of keywords or edges in a given neighborhood.

3.2 Information Captured by Aggregators

Now we observe which information is captured by the standard GNN formulation [Bronstein *et al.*, 2021], where the representation of node u is updated as

$$h_u = \text{Aggregate}(\{\phi(h_u, h_v) : v \in \overline{\mathcal{N}}(u)\}), \quad (3)$$

where the aggregator collects pairwise messages between the central node and each neighbor. Given that ϕ function processes representations pairwise and in conjunction with the

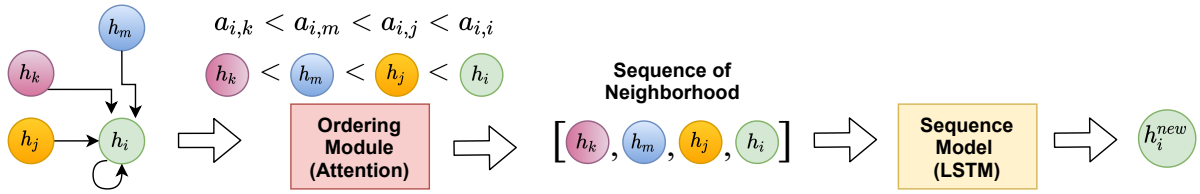

Figure 2: An illustration of the aggregation and update of the representation of node v_i using a GOAT layer. A self-attention mechanism is used in order to obtain a ranking between the nodes of the neighborhood and then the ordered neighborhood is given as input into a sequence model (LSTM) to produce the updated representation of node v_i.

central node, any interactions among neighbors are ignored by the aggregator (see Figure 1). Thus, the amount of information that is captured equals $\sum_{v \in \overline{\mathcal{N}}(u)} I(h_u; h_v)$.

To discuss this aggregation scheme we consider the example of a neighborhood with only two independently distributed neighbors v_1, v_2. The captured information 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 total structure-informed 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 GNN is less than the information present in the neighborhood due to the absence of synergistic information. Therefore, instead of using a pairwise function ϕ, we advocate the use of a function $\psi : \mathbb{R}^{|\mathcal{N}(u)| \times d} \to \mathbb{R}^{|\mathcal{N}(u)| \times d}$ that simultaneously processes all the information among neighbors as follows

$$h_u = \text{Aggregate}(\psi(\{h_v : v \in \overline{\mathcal{N}}(u)\})).$$

From an information-theoretic viewpoint, our formulation has the following advantages: Firstly, the redundant information can be efficiently treated, as common parts of R can be detected and ignored if needed. Secondly, the synergistic information can now be captured, due to the ability of ψ function to observe the behavior of the neighborhood $\overline{\mathcal{N}}(u)$ altogether.

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.

4.1 GOAT Architecture

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 $\{h_1, h_2, \ldots, h_P\}$, where $P = \left|\overline{\mathcal{N}}(v)\right|$, into an ordered sequence of feature vectors,

$$[h_{\pi(1)}, h_{\pi(2)}, \ldots, h_{\pi(P)}] = \text{OrderingPart}(\{h_1, h_2, \ldots, h_P\}),$$

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

Similar to the GAT [Veličković *et al.*, 2018a] 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 a weight $\vec{w}_2 \in \mathbb{R}^{2d'}$, that takes as input the concatenated features of the two nodes, and computes the *attention coefficients*

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

for all j s.t. $v_j \in \overline{\mathcal{N}}(v_i)$. Then, we sort the coefficients

$$e_{i\pi(1)}, e_{i\pi(2)}, \ldots, e_{i\pi(P)} = sort\left(e_{i1}, e_{i2}, \ldots, e_{iP}\right),$$

obtaining a specific permutation π of the nodes in the neighborhood. Then we construct the sorted sequence of node features based on the permutation π: $h_{sorted(i)} = [h_{\pi(1)}, h_{\pi(2)}, \ldots, h_{\pi(P)}]$. Notice that we use the attention coefficients only to construct an ordering of the neighborhood. Therefore, we do not apply the softmax function to them and we do not multiply the attention scores with the hidden states.

2) The **Sequence modeling part** (yellow box in Figure 2) takes as input the ordered sequence of the nodes, processes it using a shared sequence model and generates the new hidden state. In our experiments we used an LSTM [Hochreiter and Schmidhuber, 1997] as the sequence model

$$h_i^{new} = \text{LSTM}(h_{sorted(i)}). \tag{5}$$

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 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

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

Then we sort the K sets of attention coefficients obtaining multiple orderings in the neighborhood,

$$h_{sorted(i)}^k = [h_{\pi^k(1)}, h_{\pi^k(2)}, \ldots, h_{\pi^k(P)}].$$

To generate the final representation of the nodes we concatenate the features from the K independent LSTM models, i.e.,

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

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

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	0.0038 ±0.0012
GraphSAGE (mean)	61.45 ±5.79	0.0401 ±0.0158	0.0221 ±0.0069	0.0374 ±0.0085	0.0243 ±0.0056
GraphSAGE (lstm)	65.05 ±8.71	0.0094 ±0.0073	0.0153 ±0.0105	0.0022 ±0.0017	0.0008 ±0.0002
GIN	56.40 ±5.26	0.0083 ±0.0052	0.0042 ±0.0015	0.0024 ±0.0016	0.0007 ±0.0003
GAT	53.34 ±2.43	0.0409 ±0.0158	0.0220 ±0.0068	0.0382 ±0.0079	0.0248 ±0.0056
GOAT	**71.70** ±5.17	**0.0032** ±0.0014	**0.0005** ±0.0001	**0.0010** ±0.0005	**0.00007** ±0.00003

Permutation-Equivariance and Injectivity of GOAT Recall from Section 2.1 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. The GOAT layer performs a *permutation-equivariant transformation* of the hidden states.

The proof of Proposition 4.1 can be found in Appendix A.

Proposition 4.2. The GOAT layer is able to approximate any measurable injective function arbitrarily well in probability.

The proof of Proposition 4.2 relies on the universal approximation result by [Hammer, 2000] and is in Appendix B.

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 (see Sections 5.1 and 5.2) as well as on six node-classification benchmarks (see Section 5.3). Our implementation can be found in the Supplementary Material.

Baselines. We compare GOAT against the following state-of-the-art GNNs for node classification: (1) GCN [Kipf and Welling, 2017] the classical graph convolution neural network, (2) GraphSAGE(mean) [Hamilton *et al.*, 2017] that aggregates by taking the elementwise mean value, (3) GraphSAGE(lstm) [Hamilton *et al.*, 2017] that aggregates by feeding the neighborhood hidden states with a random order into an LSTM, (4) GIN [Xu *et al.*, 2019] the injective summation aggregator, (5) GAT [Veličković *et al.*, 2018a] that aggregates with a learnable weighted summation operation. We also compare with a (6) standard MLP that only uses node features and does not incorporate the graph structure.

Setup For a fair comparison we use the same training process for all the 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. More information about the datasets, the training procedure, and the hyperparameters of the models are in Appendix C.

5.1 Top-2-Pooling

We designed this synthetic task to highlight redundancy and synergistic information in a node classification task.

Setup. 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. Then, we label each node with a function $\phi(\cdot, \cdot)$ of the two 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% of nodes for training, 20% of nodes for validation and 20% for testing.

We report the average classification accuracy and the standard deviation across 10 different random graphs in Table 1. *Our model outperforms the other GNNs with a large margin.* In the context of this simulation study, we explain this performance gap with the following hypothesis. To find the largest element of a set you 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 is important for the task.

5.2 Prediction of Graph Structural Properties

The synthetic experiments in this section establish the ability of our GOAT model to predict structural properties of nodes in the absence of node features. 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 are affected by the interactions between the neighbor nodes so synergistic information is crucial for these tasks.

Table 2: Results in terms of classification accuracy on node classification benchmarks. We highlight the best performing model and underline the second best.

Method	Cora	Citeseer	Disease	LastFM Asia	Computers	Photo
MLP	43.8	52.9	79.10 ±0.97	62.23	80.04	89.68
GCN	81.4	67.5	88.98 ±2.21	75.87	84.81	92.16
GraphSAGE (mean)	77.2	65.3	88.79 ±1.95	75.80	87.53	<u>93.01</u>
GraphSAGE (lstm)	74.1	59.9	<u>90.50</u> ±2.15	<u>80.72</u>	89.53	92.61
GIN	75.5	62.1	90.20 ±2.23	78.43	<u>89.56</u>	92.88
GAT	**83**	<u>69.3</u>	89.13 ±2.22	77.57	79.28	91.77
GOAT	<u>82.8</u>	**69.5**	**90.53** ±2.26	**81.02**	**89.67**	**93.21**

The betweenness centrality $b(u)$ is a measure of centrality of a node u based on shortest paths involving u. It has many applications in network science, as it is a useful metric for analyzing communication dynamics [Goh *et al.*, 2003]. It can be computed using the following equation

$$b(u) = \sum_{s,t \in V} \frac{\sigma(s,t|u)}{\sigma(s,t)},$$

where $\sigma(s,t)$ is the number of distinct shortest paths between vertices s and t, and $\sigma(s,t|u)$ is the number of these shortest paths passing through u.

The effective size $e(u)$ [Everett and Borgatti, 2020] of node u is based on the concept of redundancy and for the case of unweighted and undirected graphs, can be computed as

$$e(u) = n - \frac{2q}{n},$$

where q is the number of ties in the subgraph induced by the node set $\mathcal{N}(u)$ (excluding ties involving u) and $n = |\mathcal{N}(u)|$ is the number of neighbors (excluding the central node).

We set the input features as the identity matrix, i.e., $\mathbf{X} = I_N$ and we 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 average mean and standard deviation across 10 graphs of each type in Table 1. Our proposed GOAT model outperforms all models in both tasks and in both graph parameter settings. GOAT is able to capture the synergistic information between the nodes, which is crucial for predicting the betweenness centrality and effective size. The other aggregators lose the structural information of nodes in neighborhoods. We also observe that GraphSAGE with LSTM aggregator that uses a random node ordering is not on par with GOAT, indicating that the learned ordering in GOAT is of use.

5.3 Node Classification Benchmarks

We utilize six well-known node classification benchmarks to validate our proposed model in real-world scenarios. Specifically, we use 2 citation network benchmark datasets: Cora, Citeseer [Sen *et al.*, 2008], 1 disease spreading model: Disease [Chami *et al.*, 2019], 2 co-purchase graphs: Amazon Computers, Amazon Photo [Shchur *et al.*, 2019] and 1 social network: LastFM Asia [Rozemberczki and Sarkar, 2020].

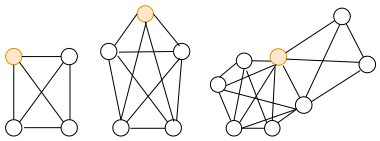

Figure 3: Neighborhoods of yellow nodes (ids: 467, 5313 and 6931) in the Amazon Photo dataset, which are misclassified by GAT and GIN and correctly classified by GOAT.

We report the classification accuracy results in Table 2. Our model outperforms the others in five out of six datasets. This demonstrates the ability of GOAT to capture the interactions of the nodes, that are crucial for the learning task.

5.4 Case Study

In Figure 3 we visualize three neighborhoods of nodes in the Amazon Photo dataset, which were misclassified by the GAT and GIN models, but correctly classified by our GOAT model. In all three neighborhoods, we observed the same structural pattern: they are dense, i.e., the node to be classified has a small effective size. Dense neighborhoods contain a large amount of synergistic information. This observed pattern supports our hypothesis, that the GOAT model outperforms current GNN models due to its ability to capture synergistic information and to effectively discard redundant information.

6 Conclusion

We have introduced 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 (GOAT) layer, which makes use of a permutation-sensitive aggregator capable of capturing synergistic and redundant information, while maintaining permutation-equivariance of the GOAT layer. The GOAT layer is implemented by first learning an ordering of nodes using a self-attention and by then applying a recurrent neural network 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. Potential future work includes the experimentation with thresholds on the attention to improve the computational efficiency of GOAT and to investigate alternative attention mechanisms.

References

[Anderson and May, 1992] Roy M Anderson and Robert M May. *Infectious diseases of humans: dynamics and control*. Oxford university press, 1992.

[Bizzi and Cheung, 2013] Emilio Bizzi and Vincent C. K. Cheung. The neural origin of muscle synergies. *Front. Comput. Neurosci*, 7:51, 2013.

[Brody *et al.*, 2021] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv:2105.14491*, 2021.

[Bronstein *et al.*, 2021] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv:2104.13478*, 2021.

[Chami *et al.*, 2019] Ines Chami, Rex Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. *NeurIPS*, 2019.

[Dasoulas *et al.*, 2020] George Dasoulas, Giannis Nikolentzos, Kevin Scaman, Aladin Virmaux, and Michalis Vazirgiannis. Ego-based entropy measures for structural representations. *ICASSP*, 2020.

[Everett and Borgatti, 2020] Martin Everett and Stephen Borgatti. Unpacking burt's constraint measure. *Soc. Netw.*, 62:50–57, 07 2020.

[Gilmer *et al.*, 2017] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272, 2017.

[Goh *et al.*, 2003] K-I Goh, Eulsik Oh, Byungnam Kahng, and Doochul Kim. Betweenness centrality correlation in social networks. *Phys. Rev. E*, 67(1):017101, 2003.

[Gori *et al.*, 2005] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IJCNN*, pages 729–734, 2005.

[Hamilton *et al.*, 2017] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

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

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, 11 1997.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[Luo *et al.*, 2021] Gongxu Luo, Jianxin Li, Hao Peng, Carl Yang, Lichao Sun, Philip S. Yu, and Lifang He. Graph entropy guided node embedding dimension selection for graph neural networks. *arXiv:2105.03178*, 2021.

[Murphy *et al.*, 2019] Ryan L. Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *ICLR*, 2019.

[Pei *et al.*, 2020] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *ICLR*, 2020.

[Peng *et al.*, 2020] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. Graph representation learning via graphical mutual information maximization. In *WWW*, 2020.

[Rozemberczki and Sarkar, 2020] Benedek Rozemberczki and Rik Sarkar. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *CIKM*, page 1325–1334. ACM, 2020.

[Scarselli *et al.*, 2009] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Trans. Neural Netw.*, 20(1):61–80, 2009.

[Sen *et al.*, 2008] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Mag.*, 29(3):93, 2008.

[Shchur *et al.*, 2019] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. In *R2L Workshop at NeurIPS*, 2019.

[Veličković *et al.*, 2018a] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[Veličković *et al.*, 2018b] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *ICLR*, 2018.

[Vinyals *et al.*, 2016] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *ICLR*, 2016.

[Wagstaff *et al.*, 2019] Edward Wagstaff, Fabian B. Fuchs, Martin Engelcke, Ingmar Posner, and Michael Osborne. On the limitations of representing functions on sets. In *ICML*, 2019.

[Wagstaff *et al.*, 2021] Edward Wagstaff, Fabian B. Fuchs, Martin Engelcke, Michael A. Osborne, and Ingmar Posner. Universal approximation of functions on sets. *arXiv:2107.01959*, 2021.

[Williams and Beer, 2010] Paul L. Williams and Randall D. Beer. Nonnegative decomposition of multivariate information. *arXiv:1004.2515*, 2010.

[Xu *et al.*, 2019] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[Zhang and Xie, 2020] Shuo Zhang and Lei Xie. Improving attention mechanism in graph neural networks via cardinality preservation. In *IJCAI*, 2020.

Appendix

A Proof of Proposition 4.1

For this proof we will make use to of the following fact, typically GNNs construct permutation-equivariant functions on graphs by applying a permutation-invariant local function over the neighborhood of each node [Bronstein *et al.*, 2021]. To establish the permutation-equivariance of the GOAT layer it therefore suffices to show that the node-wise operation performed by our GOAT layer is permutation-invariant. To do so we make use of the following proposition which concerns the permutation-invariance of composed functions.

Proposition A.1. For any function $f : X \to Y$ and for any permutation-invariant function $g : Z \to X$, their composition $f \circ g$ is permutation-invariant.

Now since the GOAT layer is formed by the composition of the Sequence Modelling Part and the Ordering Part, by Proposition A.1 it suffices to show that the Ordering Part is permutation-invariant to establish the permutation-invariance of their composition. Recall, that in the Ordering Part of the GOAT layer we implement an attention mechanism on the hidden states of the central node and each neighboring node. Then, nodes are reordered according to the magnitude of the attention coefficients. Crucially, these computations are independent of the node labelling, making the Ordering Part of the GOAT layer permutation-invariant. Consequently, we apply a local permutation-invariant function rendering the action of the GOAT layer on the graph permutation-equivariant.

B Proof of Proposition 4.2

Our GOAT layer is a functional composition of the Ordering Part and the Sequence Modeling Part described in Section 4.1. Since the composition of two injective functions is injective itself, it suffices to show that each of the two compoents is injective.

We begin by considering the Ordering Part, which maps a multiset of hidden states to an ordered multiset of hidden states leaving the elements of these multisets unchanged. If therefore, for two multisets the same output is generated in the Ordering Part, then their elements are equal. Two multisets with all equal elements are equal themselves. Therefore, the Ordering Part of our GOAT layer is an injective function.

For the Sequence Modeling Part we make use of Theorem 3 from [Hammer, 2000, p. 6], which establishes that recurrent nerual networks can approximate any measurable function arbitrarily well in probability. Therefore, the LSTM that we employ in the Sequence Modeling Part can approximate any measurable injective function arbitrarily well in probability, thus providing us with the desired result.

C Experimental details

Datasets Details In our experiments we utilize six well-known node classification benchmarks. We describe them below:

- 2 citation network benchmark datasets: Cora, Citeseer [Sen *et al.*, 2008], where nodes represent scientific papers, edges are citations between them, and node labels are academic topics. We follow the experimental setup of [Kipf and Welling, 2017] and use 140 nodes for training, 300 for validation and 1000 for testing. We optimize hyperparameters on Cora and use the same hyperparameters for Citeseer.
- 1 disease spreading model: Disease [Chami *et al.*, 2019]. It simulates the SIR disease spreading model [Anderson and May, 1992], where the label of a node indicates if it is infected or not. We follow the experimental setup of [Chami *et al.*, 2019] and use 30/10/60% for training, validation and test sets and report the average results in 10 different splits.
- 2 co-purchase graphs: Amazon Computers, Amazon Photo [Shchur *et al.*, 2019]. Nodes represent products and edges represent that two products are frequently bought together. The node label indicates the product category. We use 60/20/20% for training, validation and test sets. We optimize hyperparameters on Computers and use the same hyperparameters for Photo.
- 1 social network: LastFM Asia [Rozemberczki and Sarkar, 2020]. Nodes are LastFM users from Asian countries and edges are mutual follower relationships between them. The label of each node is the country of the user. We use 60/20/20% for training, validation and test sets.

We report further summary statistics of these datasets in Table 3.

Synthetic Experiments: Prediction of Graph Structural Properties (node regression) For the GCN and GraphSage model we transform the input features with a linear layer and then use 2 convolutional layers followed by 1 linear layer. To optimize the hyper-parameters we perform a grid-search on the following values: linear $= \{4, 8, 16, 32, 64\}$ for the first linear layer, conv1 $= \{4, 8, 16, 32, 64\}$ for the first convolutional layer, conv2 $= \{4, 8, 16, 32\}$ for the second convolutional layer. For the GAT and GOAT model we optimize the following hyper-parameters: nheads $= \{1, 4, 8\}$ for the number of heads, conv1 $= \{4, 8, 16, 32, 64\}$ for the first convolutional layer, conv2 $= \{4, 8, 16, 32, 64\}$ for the second convolutional layer. We search for the best model on $(N = 100, p = 0.09)$ and we use the same models for the other configuration of each task $(N = 1000, p = 0.1)$.

Node classification Benchmarks. For node classification benchmarks we follow the same model configurations as with node regression above and we just remove the last linear layers from all the models.

	Cora	Citeseer	Disease	LastFM Asia	Computers	Photo
# Nodes	2708	3327	1044	7624	13752	7650
# Edges	5429	4732	1043	27806	245778	119043
# Features/Node	1433	3703	1000	128	767	745
# Classes	7	6	2	18	10	8
# Training Nodes	140	120	312	4574	9625	5354
# Validation Nodes	300	500	105	1525	1376	765
# Test Nodes	1000	1000	627	1525	2751	1531

Table 3: Summary of the datasets used in our experiments.