MEMORY-BASED GRAPH NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Neural Networks (GNNs) are a class of deep models that operates on data with arbitrary topology and order-invariant structure represented as graphs. We introduce an efficient memory layer for GNNs that can learn to jointly perform graph representation learning and graph pooling. We also introduce two new networks based on our memory layer: Memory-Based Graph Neural Network (MemGNN) and Graph Memory Network (GMN) that can learn hierarchical graph representations by coarsening the graph throughout the layers of memory. The experimental results demonstrate that the proposed models achieve state-of-the-art results in six out of seven graph classification and regression benchmarks. We also show that the learned representations could correspond to chemical features in the molecule data.

1 INTRODUCTION

Graph Neural Networks (GNNs) (Wu et al., 2019; Zhou et al., 2018; Zhang et al., 2018) are a class of deep architectures that operate on data with arbitrary topology and order-invariant structure represented as graphs such as social networks (Kipf & Welling, 2016), knowledge graphs (Schlichtkrull et al., 2018), molecules (Duvenaud et al., 2015), point clouds (Wang et al., 2018b), and robot designs (Wang et al., 2019). Unlike regular-structured inputs with spatial locality properties such as grids (e.g., images and volumetric data) and sequences (e.g., speech and text), GNN inputs are variable-size graphs consisting of permutation-invariant nodes and interactions among them. GNN models such as Gated GNN (GGNN) (Li et al., 2015), Message Passing Neural Network (MPNN) (Gilmer et al., 2017), Graph Convolutional Network (GCN) (Kipf & Welling, 2016), and Graph Attention Network (GAT) (Velikovi et al., 2018) address this by learning node embeddings through an iterative process of transferring, transforming, and aggregating the node embeddings from topological neighbors. Each iteration expands the "receptive field" by one hop and after k iterations the nodes within k hops influence the node embeddings of one another.

GNN models however cannot learn hierarchical representations as they do not exploit the compositionality of the graphs. Recent works such as Differentiable Pooling (DiffPool) (Ying et al., 2018), TopKPool (Gao & Ji, 2019), and Self-Attention Graph Pooling (SAGPool) (Lee et al., 2019) define end-to-end parametric "graph pooling" layers that allow learning hierarchical graph representations by stacking interleaved layers of GNN and pooling layers similar to CNNs. The pooling layers learn node clusters that are meaningful with respect to the task in the embedding space. These clusters might be communities in a social network or potent functional groups within a chemical dataset. Nevertheless, these models are not efficient as they require an iterative process of "message passing" after each pooling layer.

In this paper, we introduce a memory layer for graph representation learning consisting of a multi-head array of keys and a convolution operator to aggregate the soft cluster assignments from different heads. The queries to a memory layer are node embeddings from the previous layer and the outputs are the node embeddings of the coarsened graph. The memory layer does not explicitly require connectivity information (i.e., adjacency matrix) and unlike GNNs relies on the global information rather than local topology. These properties make the memory layers more efficient compared to graph "message passing" and pooling layers while improving the performance. It also enables simultaneous representation learning and pooling. We also introduce two networks based on the proposed memory layers: Memory-based Graph Neural Network (MemGNN) and Graph Memory Network (GMN). MemGNN consists of a GNN encoder that learns the node embeddings, and layers of memory that coarsen the graph by learning hierarchical graph representation up to the graph

embedding. GMN on the other hand learns the hierarchical graph representation purely based on memory layers and hence does not require "message passing".

2 RELATED WORKS

Memory-augmented neural networks are deep models that utilize external memory with differentiable read-write operators allowing them to explicitly access the past experiences and are shown to enhance reinforcement learning (Pritzel et al., 2017), meta learning (Santoro et al., 2016), few-shot learning (Vinyals et al., 2016), and multi-hop reasoning (Weston et al., 2015). Unlike RNNs in which the memory is represented within the hidden states of the network, the explicit representation in the memory-augmented models helps them to store and retrieve longer-term memories with less parameters. The memory is usually implemented either as a differentiable neural dictionary (i.e., key-value memory) (e.g., neural episodic control (Pritzel et al., 2017) and product-key memory layers (Lample et al., 2019)) or a differentiable array (e.g., Neural Turing Machine (NTM) (Graves et al., 2014), prototypical networks (Snell et al., 2017), and memory networks (Weston et al., 2015)). Our memory layer consists of a multi-head array and a convolutional layer to aggregate the heads.

Graph Neural Networks use "message passing" mechanism to learn node embeddings over graphs. GraphSAGE (Hamilton et al., 2017b) learns embedding by sampling and aggregating neighbor nodes whereas GAT (Velikovi et al., 2018) uses attention to aggregate embeddings from all neighbors. GCN models extend the convolution operator to arbitrary topology. Spectral GCNs (Bruna et al., 2014; Defferrard et al., 2016; Kipf & Welling, 2016) use spectral filters over graph Laplacian to define the convolution operator in the Fourier domain. These models are less efficient compared to spatial GCNs (Schlichtkrull et al., 2018; Ma et al., 2019) which directly define the convolution on graph patches centered on nodes. Our memory layer uses a feed-forward network to learn the node embeddings.

Graph pooling methods can be categorized into two classes: global and hierarchical pooling. In former all node embeddings are pooled into a single graph embedding using arithmetic operators (e.g., summation) (Hamilton et al., 2017a; Kipf & Welling, 2016) or neural networks operating over sets (e.g., Set2Set (Vinyals et al., 2015) and SortPool (Morris et al., 2019)). Hierarchical methods coarsen the graph in each layer to capture the hierarchical structure information. Non-parametric pooling methods (e.g., Clique pooling (Luzhnica et al., 2019), kNN pooling (Wang et al., 2018a), and Graclus (Dhillon et al., 2007)) relying on topological information are efficient, but cannot compete with the performance of parametric models (e.g., DiffPool (Ying et al., 2018) and edge contraction pooling (Diehl, 2019)).

DiffPool (Ying et al., 2018) trains two parallel GNNs to compute node embeddings and cluster assignments respectively, using a graph classification loss along with link prediction and entropy losses to regularize the model, whereas Mincut pooling (Bianchi et al., 2019) trains a sequence of a GNN and an MLP using a task-specific loss and an unsupervised loss that encourages the model to minimize the minCUT objective while clustering the connected nodes together. TopKPool (Cangea et al., 2018; Gao & Ji, 2019) computes a node score by learning a projection vector and then drops all the nodes except the nodes with top scores. SAGPool (Lee et al., 2019) extends the TopKPool by using graph convolutions to take neighbor node features into account. We impose a clustering-friendly distribution as our distance metric to define the pooling operator.

3 METHOD

3.1 MEMORY LAYER

We define a memory layer $M^{(l)} : \mathbb{R}^{n_l \quad d_l} \; 7 \; ! \; \mathbb{R}^{n_{l+1} \quad d_{l+1}}$ in layer l as a parametric function that learns to map n_l node embeddings of size d_l to n_{l+1} node embeddings of size d_{l+1} such that $n_{l+1} < n_l$. The memory layer learns to coarsen an input set of nodes while projecting their embeddings into a new space. A memory layer shown in Figure 1 consists of a multi-head array of keys and a convolutional layer. A common practice in memory networks for computing the output is to use softmax scores over inner product of queries and keys (i.e., $w_j = softmax(q_j^\top k_j)$ and $m = \sum_j w_j v_j$ where w_j is the normalized score and m is the output vector) (Lample et al., 2019). In

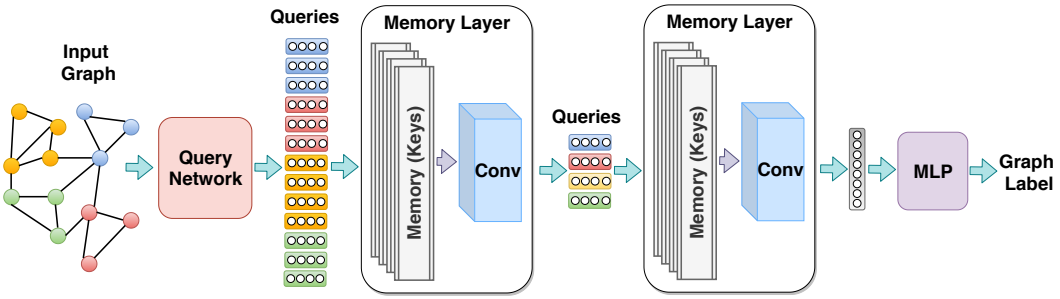


Figure 1: Proposed Architecture for hierarchical graph representation learning using proposed memory layers. Query network learns to project the initial node attributes to a query embedding space and each memory layer coarsens the queries while mapping them to a newer query space.

this work we instead treat the input node embeddings as queries $\mathbf{Q}^l \in \mathbb{R}^{n_l \times d_l}$ and the keys $\mathbf{K}^l \in \mathbb{R}^{n_{l+1} \times d_l}$ as the cluster centroids of the queries. Hence we impose a clustering-friendly distribution as our distance metric. Following (Xie et al., 2016; Maaten & Hinton, 2008), we use the Student's t-distribution as a kernel to measure the normalized similarity between query q_i and key k_j:

$$C_{i,j} = \frac{\left(1 + \|q_i - k_j\|^2/\tau\right)^{-\frac{\tau+1}{2}}}{\sum_{j'} \left(1 + \|q_i - k_{j'}\|^2/\tau\right)^{-\frac{\tau+1}{2}}} \tag{1}$$

where C_{ij} is the normalized score between query q_i and key k_j (i.e., probability of assigning node i to cluster j) and τ is the degree of freedom of the Student's t-distribution (i.e., temperature). One can interpret $\mathbf{C} \in \mathbb{R}^{n_{l+1} \times n_l}$ as either the soft cluster assignments or the attention scores.

To increase the model capacity we model the memory as a multi-head array. Applying a query against the memory therefore will generate a multi-head cluster assignment tensor $[\mathbf{C}^0, ..., \mathbf{C}^{h-1}]$ where h denotes the number of heads. To aggregate the heads into a single assignment matrix we apply a convolution over the heads as follows:

$$\mathbf{C}^{(l)} = \phi\left(\|_{h=0}^{h-1} \mathbf{C}^h\right) \in \mathbb{R}^{n_l \times n_{l+1}} \tag{2}$$

where ϕ is a convolutional layer parametrized by ϕ, $\|$ is the concatenation operator, and $\mathbf{C}^{(l)}$ is the aggregated cluster assignment matrix.

The values ($\mathbf{V}^l \in \mathbb{R}^{n_{l+1} \times d_l}$) are defined as the product of the scores and the original queries and represent the coarsened embeddings in the same space as the input queries.

$$\mathbf{V}^{(l)} = \mathbf{C}^{(l)\top}\mathbf{Q}^{(l)} \in \mathbb{R}^{n_{l+1} \times d_l} \tag{3}$$

The values are then passed through a non-linear unit consisting of a weight matrix $\mathbf{W}$ and a LeakyReLU activation function to project the coarsened embeddings from $\mathbb{R}^{n_{l+1} \times d_l}$ into a new space $\mathbb{R}^{n_{l+1} \times d_{l+1}}$.

$$\mathbf{Q}^{(l+1)} = \text{LeakyReLU}\left(\mathbf{W}\mathbf{V}^{(l)}\right) \in \mathbb{R}^{n_{l+1} \times d_{l+1}} \tag{4}$$

Thanks to these parametrized transformations, a memory layer learns both hierarchical graph pooling and node embeddings simultaneously in an end-to-end fashion. The computed outputs are the input queries to the subsequent memory layer $M^{(l+1)}$. Hence for a graph classification task, one can simply stack layers of memory up to the level where the input graph is coarsened into a single node representing the global graph embedding and then feed it to a fully-connected layer to predict the graph class as follows:

$$Y = \text{softmax}\left(\text{MLP}\left(M^{(l)}\left(M^{(l-1)}\left(....M^{(0)}(\mathbf{Q}_0)\right)\right)\right)\right) \tag{5}$$

where $\mathbf{Q}_0$ is the initial query embedding generated by the query network [1].

[1] We use initial node embeddings and initial query embeddings interchangeably throughout the paper.

3.2 GMN ARCHITECTURE

A GMN is a stack of memory layers on top of a query network that generates the initial query embeddings without any "message passing" processes involved. Similar to set neural networks (Vinyals et al., 2015) and transformers (Vaswani et al., 2017) graph nodes in a GMN are treated as a permutation-invariant set of embeddings. The query network projects the initial node attributes into an embedding space representing the initial query space.

Assume a training set $D = [g_1, g_2, ..., g_N]$ of N graphs where each graph is represented as $g = (\mathbf{A}, \mathbf{X}, Y_n, Y_g)$ and $\mathbf{A} \in \{0,1\}^{n \times n}$ denotes the adjacency matrix, $\mathbf{X} \in \mathbb{R}^{n \times d_{in}}$ is the initial node attribute, $Y_n \in \mathbb{R}^n$ is the node label, and Y_g is the graph label. Depending on the task $\mathbf{X}$, Y_n or Y_g can be empty. Considering that the GMN model treats a graph as a set of order-invariant nodes that does not use "message passing", and considering that the memory layers do not rely on connectivity information, the topological information of each node should be encoded into its initial node embedding by the query network. Inspired by transformers (Vaswani et al., 2017) we encode the topological information along with the initial node attribute into the initial query embeddings using the query network as follows:

$$\mathbf{Q}^{(0)} = \text{LeakyReLU}\left(\mathbf{W}_1 \left[\text{LeakyReLU}(\mathbf{A}\mathbf{W}_0) \,\|\, \mathbf{X}\right]\right) \tag{6}$$

Here adjacency matrix $\mathbf{A}$ is fed to a non-linear unit parameterized by $\mathbf{W}_0$, concatenated by the initial node attribute, and then jointly fed to a non-linear unit parameterized by $\mathbf{W}_1$. Hence, the query network is implemented as a two-layer feed-forward neural network.

3.3 MEMGNN ARCHITECTURE

Unlike the GMN architecture, the query network in MemGNN relies on the iterative process of "message passing" and aggregation to compute the initial query embeddings. The initial query $\mathbf{Q}_0$ generated by the query network of a MemGNN is defined as follows.

$$\mathbf{Q}^{(0)} = G_\theta\left(\mathbf{A}, \mathbf{X}\right) \tag{7}$$

where query network G_θ is an arbitrary parameterized "message passing" GNN (Gilmer et al., 2017; Li et al., 2015; Kipf & Welling, 2016; Velikovi et al., 2018).

In our implementation of MemGNN we use a modified variant of GAT (Velikovi et al., 2018). Specifically, we introduce an extension to the original GAT model called edge-based GAT (**e-GAT**) and use it as the query network. Unlike GAT, e-GAT learns attention weights not only from the neighbor nodes but also from the input edge attributes as well. This is especially important for data containing edge information (e.g., various bonds among atoms represented as edges in molecule datasets). In an e-GAT layer, attention score between two neighbor nodes is computed as follows.

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{W}\left[\mathbf{W}_n h_i^{(l)} \,\|\, \mathbf{W}_n h_j^{(l)} \,\|\, \mathbf{W}_e h_{i,j}^{(l)}\right]\right)\right)}{\sum_{k \in \mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(\mathbf{W}\left[\mathbf{W}_n h_i^{(l)} \,\|\, \mathbf{W}_n h_k^{(l)} \,\|\, \mathbf{W}_e h_{i,k}^{(l)}\right]\right)\right)} \tag{8}$$

where $h_i^{(l)}$ and $h_{i,j}^{(l)}$ denote the embedding of node i and the embedding of the edge connecting node i to its one-hop neighbor node j in layer l, respectively. $\mathbf{W}_n$ and $\mathbf{W}_e$ are learnable node and edge weight matrices and $\mathbf{W}$ is the parameter of a single-layer feed-forward network that computes the attention. The attention score is then used to update the node embeddings as follows:

$$h_i^{(l+1)} = \text{LeakyReLU}\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} h_j^{(l)}\right) \tag{9}$$

3.4 TRAINING

We train the model using a loss function that consists of two terms: an unsupervised clustering loss and a task-specific loss. The unsupervised loss is inspired by deep clustering methods (Razavi et al., 2019; Xie et al., 2016; Aljalbout et al., 2018). It encourages the model to learn clustering-friendly embeddings in the latent space by urging it to learn from high confidence assignments

with the help of an auxiliary target distribution. To apply this, we define the unsupervised loss as the KullbackLeibler (KL) divergence loss between the soft assignments $C^{(l)}$ (i.e., computed in Equation 2) and the auxiliary distribution $P^{(l)}$ as follows:

$$L_h^{(l)} = KL(P^{(l)} || C^{(l)}) = \sum_i \sum_j P_{ij}^{(l)} log \frac{P_{ij}^{(l)}}{C_{ij}^{(l)}} \tag{10}$$

For the target distributions $P^{(l)}$, we use the distribution proposed in Xie et al. (2016) which normalizes the loss contributions and improves the cluster purity while emphasizing on the samples with higher confidence. This distribution is defined as follows.

$$P_{ij}^{(l)} = \frac{\left(C_{ij}^{(l)}\right)^2 / \sum_i C_{ij}^{(l)}}{\sum_{j'} \left(C_{ij'}^{(l)}\right)^2 / \sum_i C_{ij'}^{(l)}} \tag{11}$$

We then define the loss as follows:

$$L = \frac{1}{N} \sum_{n=1}^{N} \left(w L_o + (1 \quad w) \sum_{l=1}^{L} L_h^{(l)} \right) \tag{12}$$

where L_o is the task-dependent objective (e.g., cross-entropy loss for graph classification), L is the number of memory layers, $L_h^{(l)}$ is the KL divergence between soft cluster assignments and the target distribution in layer l, and w is a scalar weight. To stabilize the training the gradients of the task-specific loss are back-propagated batch-wise while the gradients of the unsupervised loss are applied epoch-wise by periodically switching w between 0 and 1. We optimize the model parameters w.r.t the defined objective using mini-batch stochastic gradient descent.

4 EXPERIMENTS

4.1 DATASETS

We use seven graph benchmarks including five graph classification and two graph regression benchmarks to evaluate the proposed method. These datasets are commonly used in both graph kernel (Borgwardt & Kriegel, 2005; Yanardag & Vishwanathan; Shervashidze et al., 2009; Ying et al., 2018; Shervashidze et al., 2011; Kriege et al., 2016) and GNN (Cangea et al., 2018; Ying et al., 2018; Lee et al., 2019; Gao & Ji, 2019) literature. They are as follows (i.e., first two benchmarks are regression tasks and the rest are classification tasks):

ESOL (Delaney, 2004) contains water solubility data for compounds.
Lipophilicity (Gaulton et al., 2016) contains experimental results of octanol/water distribution of compounds.
Bace (Subramanian et al., 2016) provides quantitative binding results for a set of inhibitors of human β-secretase 1 (BACE-1).
DD (Dobson & Doig, 2003) is used to distinguish enzyme structures from non-enzymes.
Enzymes (Schomburg et al., 2004) is for predicting functional classes of enzymes.
Proteins (Dobson & Doig, 2003) is used to predict the protein function from structure.
COLLAB (Yanardag & Vishwanathan) is for predicting the field of a researcher given her ego-collaboration graph.

For more information regarding the characteristics of the datasets refer to Appendix A.4 and for implementation details see A.1.

4.2 RESULTS

To evaluate the performance of our models on DD, Enzymes, Proteins, and COLLAB datasets, we follow the experimental protocol in (Ying et al., 2018) and perform 10-fold cross-validation to evaluate the model performance and report the mean accuracy over all folds. We also report the performance of four kernel-based methods including Graphlet (Shervashidze et al., 2009), shortest

Table 1: Mean validation accuracy over 10-folds.

Method	**Dataset**			
	Enzymes	Proteins	DD	COLLAB
Graphlet (Shervashidze et al., 2009)	41.03	72.91	64.66	64.66
Shortest Path (Borgwardt & Kriegel, 2005)	42.32	76.43	78.86	59.10
Weisfeiler-Lehman (WL) (Shervashidze et al., 2011)	53.43	73.76	74.02	78.61
WL Optimal Assignment (Kriege et al., 2016)	60.13	75.26	79.04	80.74
PatchySan (Niepert et al., 2016)		75.00	76.27	72.60
GraphSage + MeanPool (Hamilton et al., 2017a)	54.25	70.48	75.42	68.25
ECC (Simonovsky & Komodakis, 2017)	53.50	72.65	74.10	67.79
Set2Set (Vinyals et al., 2015)	60.15	74.29	78.12	71.75
SortPool (Morris et al., 2019)	57.12	75.54	79.37	73.76
DiffPool (Ying et al., 2018)	60.53	76.25	80.64	75.48
CliquePool (Luzhnica et al., 2019)	60.71	72.59	77.33	74.50
Sparse HGC (Cangea et al., 2018)	64.17	75.46	78.59	75.46
Mincut Pool (Bianchi et al., 2019)		76.5	80.3	**83.4**
TopKPool (Gao & Ji, 2019)		77.68	82.43	77.56
SAGPool (Lee et al., 2019)		71.86	76.45	
GMN (ours)	**78.66**	**82.07**	82.24	77.26
MemGNN (ours)	75.50	81.35	**82.92**	77.0

path (Borgwardt & Kriegel, 2005), Weisfeiler-Lehman (WL) (Shervashidze et al., 2011), and WL Optimal Assignment (Kriege et al., 2016), and eleven deep models. The results are shown in Table 1.

Table 1 demonstrates that: (i) our proposed models significantly improve the performance on DD, Enzymes, and Proteins datasets by absolute margins of 14.49%, 4.75%, and 0.49% accuracy, respectively, (ii) both proposed models achieve better performance on these three datasets compared to the baselines, (iii) MemGNN outperforms GMN on COLLAB whereas MGN achieves better result on the Enzymes, Proteins, and DD datasets.

On the COLLAB dataset our models are outperformed by Mincut pool (Bianchi et al., 2019), a variant of DiffPpool (i.e., diffpool-det) (Ying et al., 2018) and WL Optimal Assignment (Kriege et al., 2016). The first two models are GNNs augmented with explicit clustering domain-knowledge [2] whereas the third method is based on graph kernels. We speculate that because of the high edge-to-node ratio of COLLAB (i.e., 33.23) the augmentations help in extracting near-optimal cliques.

For the remaining datastes we follow the evaluation protocol in (Wu et al., 2018) and report the Root-Mean-Square Error (RMSE) for the regression tasks (i.e., ESOL and Lipophilicity) and the Area Under the Curve Receiver Operating Characteristics (AUC-ROC) measure for the BACE classification task. Considering that these datasets contain initial edge attributes (refer to Appendix A.4 for further details), we train the MemGNN model and compare the results to the baseline models reported in (Wu et al., 2018) including graph-based methods such as GCN, MPNN, Directed Acyclic Graph (DAG) based models, Weave as well as other conventional methods such as Kernel Ridge Regression (KRR) and Influence Relevance Voting (IRV). Tables 2 and 3 demonstrate that our MemGNN model achieves state-of-the-art results by absolute margin of 0.04 and 0.1 RMSE on ESOL and Lipophilicity benchmarks and margin of 4.0% AUC-ROC on BACE benchmark. For further details on these datasets and the baselines see (Wu et al., 2018).

4.3 Ablation Study

4.3.1 Effect of introducing edge feature

To investigate the effect of the proposed e-GAT model, we train our MemGNN model using both GAT and e-GAT models as the query network. Considering that ESOL, Lipophilicity, and BACE datasets contain different edge types, we set them as our target benchmarks. Since nodes have richer

[2] In diffpool-det assignment matrices are generated using a deterministic graph clustering algorithm and Mincut pool explicitly introduces graph mincut objective to the loss function.

Table 2: RMSE on ESOL and Lipophilicity.

Method	**Dataset**			
	ESOL		Lipophilicity	
	validation	test	validation	test
Multitask	1.17	1.12	0.85	0.86
Random Forest	1.16	1.07	0.84	0.88
XGBoost	1.1	0.99	0.78	0.80
GCN	1.05	0.97	0.68	0.66
MPNN	0.55	0.58	0.76	0.72
KRR	1.65	1.53	0.89	0.89
DAG	0.74	0.82	0.86	0.84
Weave	0.57	0.61	0.73	0.72
MemGNN (ours)	**0.53**	**0.54**	**0.55**	**0.56**

Table 3: AUC-ROC for BACE dataset.

Method	validation	test
Logistic Regression	71.9	78.1
KernelSVM	73.9	86.2
XGBoost	75.6	85.0
Random Forest	72.8	86.7
IRV	71.5	83.8
Multitask	69.6	82.4
Bypass	74.5	82.9
GCN	62.7	78.3
Weave	63.8	80.6
MemGNN (ours)	**84.4**	**90.7**

features compared to edges, we set the node and edge feature dimensions to 16 and 4, respectively. The comparative evaluation of the performance of the two models on the ESOL dataset is shown in Appendix A.2 demonstrating that e-GAT achieves better results on the validation set in each epoch compared to the standard GAT model. We observed the same effect on Lipophilicity and BACE datasets too.

4.3.2 TOPOLOGICAL EMBEDDING

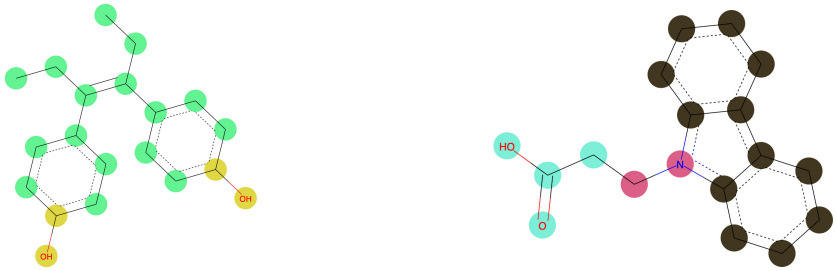
To investigate the effect of the topological embeddings on the GMN model, we evaluated three initial topological features including adjacency matrix, normalized adjacency matrix, and Random Walk with Restart (RWR). RWR is the steady diffusion state of Markovian random walk that captures the pair-wise relevance score between nodes within a graph and is used in methods such as personalized PageRank (Tong et al., 2006). It is computed once before the training using $\vec{t_i} = (1 \quad p)(I \quad c\hat{A})^{\ 1}\vec{e_i}$ where $\vec{t_i}$ denotes the RWR vector of score for node i, p is the restart probability, $\hat{A}$ is the normalized adjacency matrix, and $\vec{e_i}$ is the one-hot vector representing node i. For further details about RWR relevance scores please see section A.3. The results suggested that using the adjacency matrix as the initial feature achieves the best performance. For instance, 10-fold cross validation accuracy of a GMN model trained on ENZYMES with adjacency matrix, normalized adjacency matrix, and RWR is 78.66%, 77.16%, and 77.33%, respectively.

4.3.3 DOWN-SAMPLING NEIGHBORS BASED ON RANDOM WALKS

We investigated two methods to down-sample the neighbors in dense datasets such as COLLAB (i.e., average of 66 neighbors per node) to enhance the memory and computation. The first method randomly selects 10% of the edges whereas the second method ranks the neighbors based on their RWR scores with respect to the center node and then keeps the top 10% of the edges. We trained the MemGNN model on COLLAB using both sampling methods which resulted in 73.9% and 73.1% 10-fold cross validation accuracy for random and RWR-based sampling methods respectively, suggesting that random sampling performs slightly better than a random walk ranking and sampling method.

4.3.4 EFFECT OF NUMBER OF KEYS AND HEADS

We stipulate that although keys represent the clusters, the number of keys is not necessarily proportional to the number of the nodes in the input graphs. In fact, datasets with smaller graphs might have more meaningful clusters to capture. For example, molecules are comprised of numerous functional groups and yet the average number of nodes in the ESOL dataset is 13.3. Moreover, our experiments show that for the ENZYMES dataset with average number of 32.69 nodes, the best performance is achieved with 10 keys whereas for the ESOL dataset 64 keys results in the best performance. In ESOL 8, 64, and 160 keys result in RMSE of 0.56, 0.52, and 0.54, respectively. We also observe that keeping the number of parameters fixed, increasing the number of memory heads improves performance. For instance, when the model is trained on ESOL with 160 keys and 1 head it achieves RMSE of 0.54 whereas when trained with 32 keys of 5 heads the same model achieves RMSE of 0.53.



(a) (b)

Figure 2: Visualization of the learned clusters of two molecules instances from (a) ESOL and (b) Lipophilicity datasets. The visualizations show that the learned clusters correspond to known chemical groups. Note that a node without label represents a carbon atom. For more visualizations and discussion see section A.5

4.3.5 What Do the Keys Represent?

Intuitively, the keys in our memory layers represent the cluster centroids and enhance the model performance by capturing meaningful structures and coarsening the graph. To investigate this intuition we use the learned keys to interpret the knowledge learned by models through visualizations. Figure 2 visualizes the learned clusters over atoms (i.e., atoms with same color are within the same cluster) indicating that the clusters mainly consist of meaningful chemical substructures such as a carbon chain and a Hydroxyl group (OH) (i.e., Figure 2a), as well as a Carboxyl group (COOH) and a benzene ring (i.e., Figure 2b). From a chemical perspective, Hydroxyl and Carboxyl groups as well as carbon chains have a significant impact on the solubility of the molecule in water or lipid. This confirms that the network has learned chemical features that are essential for determining the molecule solubility.

5 Conclusion

We proposed an efficient memory layer and two deep models for hierarchical graph representation learning. We evaluated the proposed models on seven graph classification and regression tasks and achieved state-of-the-art results on six of them. We also experimentally showed that the learned representations can capture the chemical features of the molecules that are well-known and effective for each specific task. Our study indicates that node attributes concatenated with corresponding proper topological embeddings in combination with one or more memory layers achieves notable results without using any message passing mechanism.
We also showed that for the topological embedding, that arises from the graph structure, the binary adjacency matrix is sufficient and thus no further pre-processing step is required for extracting these positional embeddings such as random walk with restart relevance scores. Although connectivity of the nodes is not directly imposed to the model, keys in the embedding space can process the set of nodes and properly cluster and aggregate the learned embeddings.

6 Limitations and Future Directions

In section 4.2 we discussed that on the COLLAB dataset approaches injected with domain-knowledge such as min-cut pooling can achieve better performance. Analyzing samples in this dataset suggests that in graphs with dense communities, such as cliques, our model lacks the ability to properly detect these dense sub-graphs.
We plan to introduce an approach based on our current architecture that can be employed for node classification tasks as well by considering the learned features from clusters from different layers of hierarchies.

REFERENCES

Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, Maximilian Strobel, and Daniel Cremers. Clustering with deep learning: Taxonomy and new methods. *arXiv preprint arXiv:1801.07648*, 2018.

Filippo M. Bianchi, Daniele Grattarola, and Cesare Alippi. Mincut pooling in graph neural networks. *arXiv preprint arXiv:1907.00481*, 2019.

Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining*, pp. 8–pp. IEEE, 2005.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representation (ICLR)*, 2014.

Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. In *Advances in neural information processing systems, Relational Representation Learning Workshop*, 2018.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pp. 3844–3852, 2016.

John S Delaney. Esol: estimating aqueous solubility directly from molecular structure. *Journal of chemical information and computer sciences*, 44(3):1000–1005, 2004.

Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11): 1944–1957, 2007.

Frederik Diehl. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019.

Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *International Conference on Machine Learning*, pp. 2083–2092, 2019.

Anna Gaulton, Anne Hersey, Michał Nowotka, A Patrícia Bento, Jon Chambers, David Mendez, Prudence Mutowo, Francis Atkinson, Louisa J Bellis, Elena Cibrián-Uhalte, et al. The chembl database in 2017. *Nucleic acids research*, 45(D1):D945–D954, 2016.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272. JMLR. org, 2017.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017a.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017b.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, Proceedings of Machine Learning Research, pp. 448–456, Jul 2015.

Diederik P Kingma and Jimmy Lei Ba. Adam: Amethod for stochastic optimization. In *International Conference on Learning Representation (ICLR)*, 2014.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016.

Nils M Kriege, Pierre-Louis Giscard, and Richard Wilson. On valid optimal assignment kernels and applications to graph classification. In *Advances in Neural Information Processing Systems*, pp. 1623–1631, 2016.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. *arXiv preprint arXiv:1907.05242*, 2019.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3734–3743, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations*, 2015.

Enxhell Luzhnica, Ben Day, and Pietro Lio. Clique pooling for graph classification. In *International Conference on Learning Representations Workshop, Representation Learning on Graphs and Manifolds*, 2019.

Jianxin Ma, Peng Cui, Kun Kuang, Xin Wang, and Wenwu Zhu. Disentangled graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 4212–4221, 2019.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9:2579–2605, 2008.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4602–4609, 2019.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pp. 2014–2023, 2016.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 2827–2836, 2017.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *arXiv preprint arXiv:1906.00446*, 2019.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *Proceedings of the 33th International Conference on Machine Learning*, pp. 1842–1850, 2016.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pp. 593–607. Springer, 2018.

Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pp. 488–495, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3693–3702, 2017.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pp. 4077–4087, 2017.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Govindan Subramanian, Bharath Ramsundar, Vijay Pande, and Rajiah Aldrin Denny. Computational modeling of β-secretase 1 (bace-1) inhibitors using ligand based approaches. *Journal of chemical information and modeling*, 56(10):1936–1949, 2016.

Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. Fast random walk with restart and its applications. In *Sixth International Conference on Data Mining (ICDM'06)*, pp. 613–622. IEEE, 2006.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Petar Velikovi, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Li, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *International Conference on Learning Representations*, 2015.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pp. 3630–3638, 2016.

Chu Wang, Babak Samari, and Kaleem Siddiqi. Local spectral graph convolution for point set feature learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 52–66, 2018a.

Tingwu Wang, Yuhao Zhou, Sanja Fidler, and Jimmy Ba. Neural graph evolution: Automatic robot design. In *International Conference on Learning Representations*, 2019.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018b.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *International Conference on Learning Representation (ICLR)*, 2015.

Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pp. 478–487, 2016.

Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, pp. 4800–4810, 2018.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202*, 2018.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

A Appendix

A.1 Implementation Details

We implemented the model with Pytorch (Paszke et al., 2017) and optimized the network using Adam (Kingma & Ba, 2014) optimizer. We trained the model for a maximum of 2000 epochs with scheduler that decreases learning rate every 500 epochs. The model uses batch-normalization (Ioffe & Szegedy, 2015), skip-connection, LeakyRelu activation function, and dropout (Srivastava et al., 2014) for regularization. Since we benchmarked our model on different datasets, we decided the hidden dimension and number of learnable parameters for each using random hyper-parameter search strategy. We plan to make the code along with the best performing hyper-parameters publicly available soon.

A.2 e-GAT comparison

In section 4.3.1 we introduced e-GAT. Figure 3a illustrates RMSE for both GAT and e-GAT based MemGNN on the validation set of the ESOL dataset. Since this is a regression task we plotted the R^2 score for these methods on the validation set. It is clear that e-GAT performs better compared to normal GAT on both metrics.

A.3 Random Walk with Restart

Consider a weighted or unweighted graph G. A random agent starts traversing the graph from node i and iteratively walks towards its neighbors with a probability proportional to the edge weight that connects them. The agent also may restart the traverse from the starting node i with probability p. Eventually, the agent will stop at node j with a probability called relevance score of node j with respect to node i (Tong et al., 2006). The relevance score of node i with every other node of the graph is summarized in $\vec{t_i}$ in equation 13 where $\vec{t_i}$ is the RWR corresponding to node i, p is the restart probability, $\mathbf{A}$ is the normalized adjacency matrix and $\vec{e_i}$ is the one-hot vector representing node i.

$$\vec{t_i} = p\mathbf{A}\vec{t_i} + (1 \quad p)\vec{e_i} \tag{13}$$

Solving this linear system results in $\vec{r_i}$ defined in equation 14

$$\vec{t_i} = (1 \quad p)(I \quad c\mathbf{A})^{\ 1}\vec{e_i} \tag{14}$$

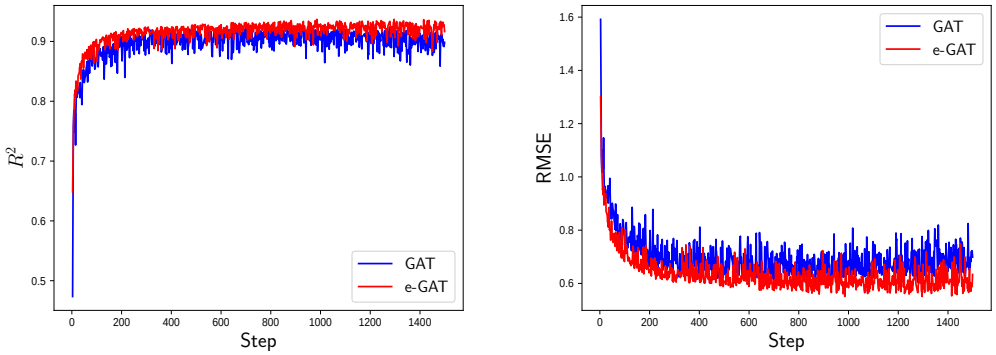
Now we can nominate the nodes with higher relevance score w.r.t. node i for receiving messages in an MeMGNN or use them as topological embeddings in a GMN. Note that the restart probability in equation 14 defines how far the agent can walk from the source node and therefore the $\vec{t_i}$ can represent whether local or global structures around the node i. We used $p = 0.5$ in our studies.
Calculating the inverse of the adjacency matrix of a big graph is costly. Although we could exactly compute it for all of our datasets but there are existing methods to make the estimation more efficient (Tong et al., 2006).

A.4 Datatasets Statistics

Table 4 indicates the statistics of the datasets we used for graph classification and regression tasks.

Table 4: Properties of the benchmark dataset.

Name	Task	Graphs	Classes	Avg. Nodes	Avg. Edges	Node Attr.	Edge Attr.
Enzymes	classification	600	6	32.63	62.14	18	0
Proteins	classification	1113	2	39.06	72.82	29	0
DD	classification	1178	2	284.32	715.66	0	0
COLLAB	classification	5000	3	74.49	2475.78	0	0
Bace	classification	1513	2	34.09	36.86	32	7
ESOL	regression	1144	-	13.29	13.68	32	7
Lipophilicity	regression	4200	-	27.04	29.50	32	7

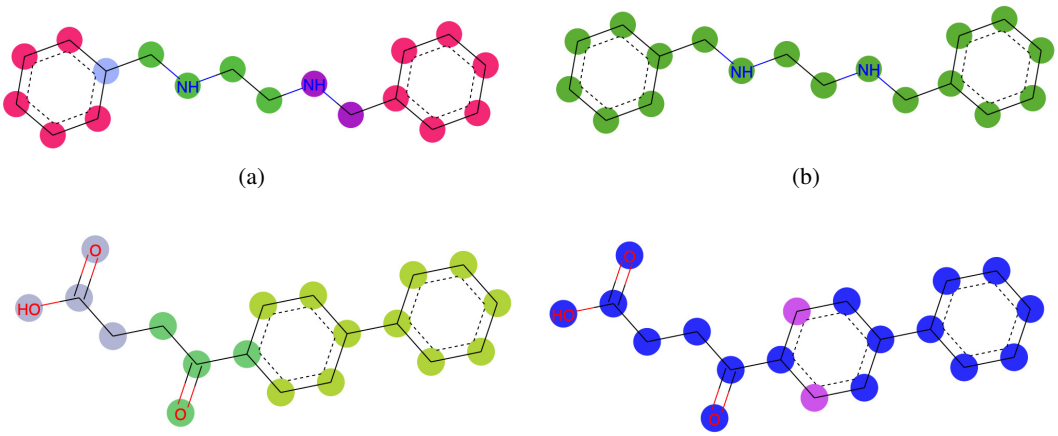


(a) (b)

Figure 3: (a) Validation R^2 score on ESOL with GAT and e-GAT. (b) Validation MSE on ESOL with GAT and e-GAT.

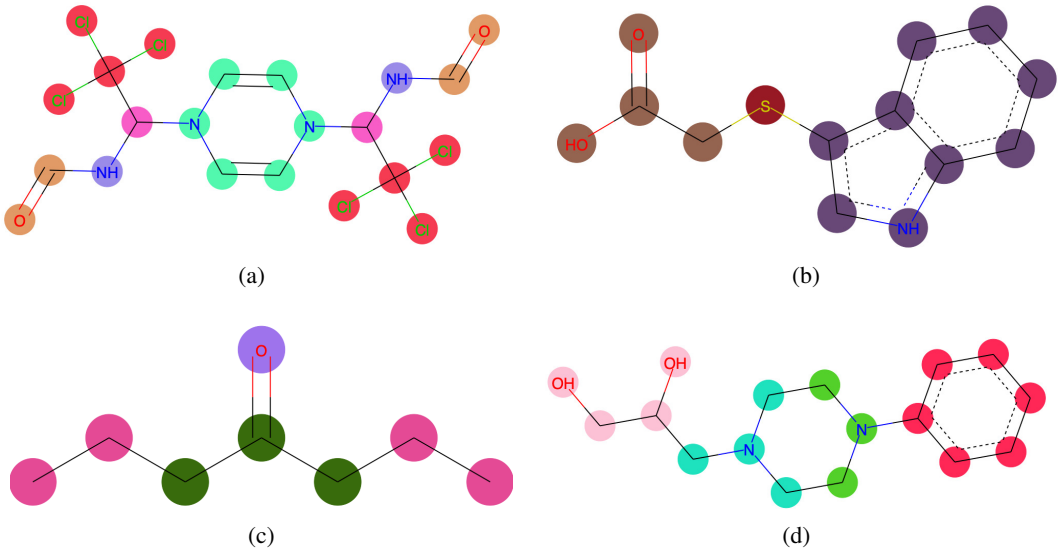
A.5 Network Learned Clusters

Figure 4 shows how unsupervised loss helps the model to push the nodes into different clusters resulting in distinct clusters. Figures 4a and 4c demonstrate clusters with unsupervised loss and Figures 4b and 4d show computed clusters without unsupervised loss. Clearly the unsupervised loss helps the model to avoid collapsing into a few number of meaningless clusters.
Figure 5 represents meaningful chemical groups extracted by MemGNN. Figures 5b and 5d are from LIPO and figure 5a and 5c are from ESOL dataset respectively.



(a) (b) (c) (d)

Figure 4: Unsupervised loss helps the model with learning distinct clusters.



(a) (b) (c) (d)

Figure 5: Clusters learned by a MeMGNN for ESOL and LIPO dataset. Chemical groups like OH (hydroxyl group), CCl3, COOH (carboxyl group), CO (ketone group) as well as benzene rings have been recognized during the learning procedure. These chemical groups are highly active and have a great impact on the solubility of molecules.