# MEMORY-BASED GRAPH NETWORKS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Graph Neural Networks (GNNs) are deep models that operate on data with arbitrary topology represented as graphs. We introduce an efficient memory layer for GNNs that can jointly learn node representations and coarsen the graph. We also introduce two new networks based on this layer: Memory-Based Graph Neural Network (MemGNN) and Graph Memory Network (GMN) that can learn hierarchical graph representations. The experimental results shows that the proposed models achieve state-of-the-art results in eight out of nine 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 represented as graphs such as social networks (Kipf & Welling, 2016), knowledge graphs (Schlichtkrull et al., 2018), molecules (Duvenaud et al., 2015), point clouds (Hassani & Haley, 2019), and robots (Wang et al., 2019). Unlike regular-structured inputs with spatial locality 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. GNNs 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) learn 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. GNNs are shown to learn better representations compared to random walks (Grover & Leskovec, 2016; Perozzi et al., 2014), matrix factorization (Belkin & Niyogi, 2002; Ou et al., 2016), kernel methods (Shervashidze et al., 2011; Kriege et al., 2016), and probabilistic graphical models (Dai et al., 2016).

These models, however, cannot learn hierarchical representation as they do not exploit the graph compositionality. Recent work such as Differentiable Pooling (DiffPool) (Ying et al., 2018), Top-KPool (Gao & Ji, 2019), and Self-Attention Graph Pooling (SAGPool) (Lee et al., 2019) define parametric graph pooling layers that let models learn hierarchical graph representation by stacking interleaved layers of GNN and pooling layers. These layers cluster nodes in the latent space such that the clusters are meaningful with respect to the task. 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 joint graph representation learning and graph coarsening that consists of a multi-head array of memory 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 and unlike GNNs relies on the global information rather than local topology. These properties make them more efficient and improve their performance. We also introduce two networks based on the proposed layer: Memory-based Graph Neural Network (MemGNN) and Graph Memory Network (GMN). MemGNN consists of a GNN that learns the initial node embeddings, and a stack of memory layers that learns hierarchical graph representation up to the global graph embedding. GMN, on the other hand, learns the hierarchical representation purely based on memory layers and hence does not require message passing.

## 2 RELATED WORK

**Memory Augmented Neural Networks (MANNs)** 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 their hidden states, the decoupled memory in MANNs lets them to store and retrieve longer term memories with less parameters. The memory can be implemented as a key-value memory such as neural episodic control (Pritzel et al., 2017) and product-key memory layers (Lample et al., 2019) or as a array-structured memory such as Neural Turing Machine (NTM) (Graves et al., 2014), prototypical networks (Snell et al., 2017), memory networks (Weston et al., 2015), and Sparse Access Memory (SAM) (Rae et al., 2016). Our memory layer consists of a multi-head array of memory keys.

**Graph Neural Networks (GNNs)** use message passing 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 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 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** can be done globally or hierarchically. In former, node embeddings are aggregated into a graph embedding using arithmetic operators such as sum or max (Hamilton et al., 2017a; Kipf & Welling, 2016) or set neural networks such as Set2Set (Vinyals et al., 2015) and SortPool (Morris et al., 2019). In latter, graphs are coarsened in each layer to capture the hierarchical structure. Non-parametric methods such as clique pooling (Luzhnica et al., 2019), kNN pooling (Wang et al., 2018), and Graclus (Dhillon et al., 2007) rely on topological information and are efficient, but are outperformed by parametric models such as edge contraction pooling (Diehl, 2019).

DiffPool (Ying et al., 2018) trains two parallel GNNs to compute node embeddings and cluster assignments using a combination of classification loss, link prediction loss, and entropy loss, whereas Mincut pool (Bianchi et al., 2019) trains a sequence of a GNN and an MLP using classification loss and the minimum cut objective. 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 top scoring nodes. SAGPool (Lee et al., 2019) extends the TopKPool by using graph convolutions to take neighbor node features into account. We use a clustering-friendly distribution to compute the attention scores between nodes and clusters.

## 3 METHOD

### 3.1 MEMORY LAYER

We define a memory layer $\mathcal{M}^{(l)} : \mathbb{R}^{n_l \times d_l} \longmapsto \mathbb{R}^{n_{l+1} \times d_{l+1}}$ in layer $l$ as a parametric function that takes in $n_l$ query vectors of size $d_l$ and generates $n_{l+1}$ query vectors of size $d_{l+1}$ such that $n_{l+1} < n_l$. The input and output queries represent the node features of the input graph and the coarsened graph, respectively. The memory layer learns to jointly coarsen the input nodes (i.e., pooling) and transform their features (i.e., representation learning). As shown in Figure 1, it consists of arrays of memory keys (i.e., multi-head memory) and a convolutional layer. Assuming $|h|$ memory heads, a shared input query is compared against all the keys in each head resulting in $|h|$ attention matrices which are then aggregated into a single attention matrix using the convolution layer.

In a *content addressable memory* (Graves et al., 2014; Sukhbaatar et al., 2015; Weston et al., 2015), the task of attending to memory (i.e., addressing scheme) is formulated as computing the similarly between memory keys to a given query $q$. Specifically, the attention weight of key $k_j$ for query $q$ is defined as $w_j = softmax(d(q, k_j))$ where $d$ is a similarity measure, typically Euclidean distance or cosine similarity (Rae et al., 2016). The soft read operation on memory is defined as a weighted average over the memory keys: $r = \sum_j w_j k_j$.

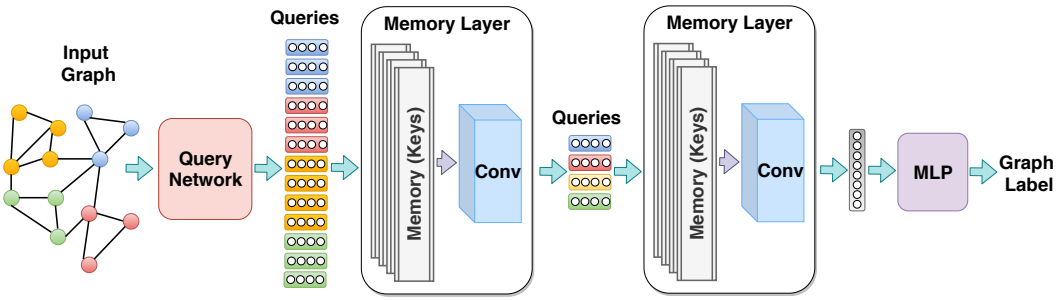



Figure 1: The proposed Architecture for hierarchical graph representation learning using the introduced memory layer. The query network projects the initial node attributes to a query embedding space and each memory layer jointly coarsens the input queries and transforms them into a new query space.

In this work, we treat the input queries $\mathbf{Q}^{(l)} \in \mathbb{R}^{n_l \times d_l}$ as the node embeddings of an input graph and treat the keys $\mathbf{K}^{(l)} \in \mathbb{R}^{n_{l+1} \times d_l}$ as the cluster centroids of the queries. To satisfy this assumption, we impose a clustering-friendly distribution as the distance metric between keys and a query. 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 $k_j$ as follows:

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

where $C_{ij}$ is the normalized score between query $q_i$ and key $k_j$ (i.e., probability of assigning node $i$ to cluster $j$ or attention score between query $q_i$ and memory key $k_j$) and $\tau$ is the degree of freedom of the Student's t-distribution (i.e., temperature).

To increase the model capacity, we model the memory keys as a multi-head array. Applying a shared input query against the memory keys produces a tensor of cluster assignments $[\mathbf{C}_0^{(l)} ... \mathbf{C}_{|h|}^{(l)}] \in \mathbb{R}^{|h| \times n_{l+1} \times n_l}$ where $|h|$ denotes the number of heads. To aggregate the heads into a single assignment matrix, we treat the heads and the matrix rows and columns as depth, height, and width in standard convolution analogy and apply a convolution operator over them. Because there is no spatial structure, we use $[1 \times 1]$ convolution to aggregate the information across heads and therefore the convolution behaves as a weighted pooling that reduces the heads to a single matrix. The aggregated assignment matrix is computed as follows:

$$\mathbf{C}^{(l)} = \text{softmax}\left(\Gamma_\phi \left(\mathop{\Big\|}_{k=0}^{|h|} \mathbf{C}_k^{(l)}\right)\right) \in \mathbb{R}^{n_l \times n_{l+1}} \tag{2}$$

where $\Gamma_\phi$ is a $[1 \times 1]$ convolutional operator parametrized by $\phi$, $||$ is the concatenation operator, and $\mathbf{C}^{(l)}$ is the aggregated soft assignment matrix.

A memory read generates a value matrix $\mathbf{V}^{(l)} \in \mathbb{R}^{n_{l+1} \times d_l}$ that represents the coarsened node embeddings in the same space as the input queries and is defined as the product of the soft assignment scores and the original queries as follows:

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

The value matrix is fed to a single layer neural network consisting of a weight matrix $\mathbf{W} \in \mathbb{R}^{d_l \times d_{l+1}}$ and a LeakyReLU activation function to project the coarsened embeddings from $\mathbb{R}^{n_{l+1} \times d_l}$ into $\mathbb{R}^{n_{l+1} \times d_{l+1}}$ representing the output queries $\mathbf{Q}^{(l+1)}$:

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

Thanks to these parametrized transformations, a memory layer can jointly learn the node embeddings and coarsens the graph end-to-end. The computed queries $\mathbf{Q}^{(l+1)}$ are the input queries to the

subsequent memory layer $\mathcal{M}^{(l+1)}$. For graph classification, 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:

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

where $\mathbf{Q}_0 = f_q(g)$ is the initial query embedding[1] generated by the query network $f_q$ over graph $g$. We introduce two architectures based on the memory layer: GMN and MemGNN. These two architectures are different in the way that the query network is implemented. More specifically, GMN uses a feed-forward network for initializing the query: $f_q(g) = \text{FFN}_\theta(g)$, whereas MemGNN implements the query network as a message passing GNN: $f_q(g) = \text{GNN}_\theta(g)$.

### 3.2 GMN Architecture

A GMN is a stack of memory layers on top of a query network $f_q(g)$ that generates the initial query embeddings without any message passing. 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 that represents the initial query space.

Assume a training set $\mathcal{D} = [g_1, g_2, ..., g_N]$ of $N$ graphs where each graph is represented as $g = (\mathbf{A}, \mathbf{X}, Y)$ 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, and $Y \in \mathbb{R}^n$ is the graph label. Considering that the GMN model treats a graph as a set of permutation-invariant nodes and does not use message passing, and also considering that the memory layers do not rely on connectivity information, the topological information of each node should be somehow encoded into its initial embedding. Inspired by transformers (Vaswani et al., 2017), we encode this information along with the initial attribute into the initial query embeddings using a query network $f_q$ implemented as a two-layer feed-forward neural network:

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

where $\mathbf{W}_0 \in \mathbb{R}^{n\times d_{in}}$ and $\mathbf{W}_1 \in \mathbb{R}^{2d_{in}\times d_0}$ are the parameters of the query networks, and $||$ is the concatenation operator.

### 3.3 MemGNN Architecture

Unlike the GMN architecture, the query network in MemGNN relies on the iterative process of passing messages and aggregating them to compute the initial query $\mathbf{Q}_0$:

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

where query network $G_\theta$ 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. 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.

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

where $h_i^{(l)}$and $h_{i\to j}^{(l)}$ 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 trainable node and edge weights and $\mathbf{W}$ is the parameter of a single-layer feed-forward network that computes the attention score.

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

### 3.4 TRAINING

We jointly train the model using two loss functions: a supervised classification loss and an unsupervised clustering loss. The supervised loss denoted as $\mathcal{L}_{ent}$ is defined as the cross-entropy loss between the predicted and true graph class labels. The unsupervised clustering 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. The unsupervised loss is defined as the Kullback-Leibler (KL) divergence loss between the soft assignments $\mathbf{C}^{(l)}$ and the auxiliary distribution $\mathbf{P}^{(l)}$ as follows:

$$\mathcal{L}_{KL}^{(l)} = \text{KL}\left(\mathbf{P}^{(l)} || \mathbf{C}^{(l)}\right) = \sum_i \sum_j P_{ij}^{(l)} log \frac{P_{ij}^{(l)}}{C_{ij}^{(l)}} \tag{9}$$

For the target distributions $\mathbf{P}^{(l)}$, 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:

$$P_{ij}^{(l)} = \frac{\left(C_{ij}^{(l)}\right)^2 / \sum_i C_{ij}^{(l)}}{\sum_{j'} \left(C_{ij'}^{(l)}\right)^2 / \sum_i C_{ij'}^{(l)}} \tag{10}$$

We define the total loss as follows where $L$ is the number of memory layers and $\lambda$ is a scalar weight.

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \left( \lambda \mathcal{L}_{ent} + (1-\lambda) \sum_{l=1}^{L} \mathcal{L}_{KL}^{(l)} \right) \tag{11}$$

We initialize the model parameters, the keys, and the queries randomly and optimize them jointly with respect to $\mathcal{L}$ using mini-batch stochastic gradient descent. To stabilize the training, the gradients of $\mathcal{L}_{ent}$ are back-propagated batch-wise while the gradients of $\mathcal{L}_{KL}^{(l)}$ are applied epoch-wise by periodically switching $\lambda$ between 0 and 1. Moreover, updating the centroids (i.e., memory keys) with the same frequency as the network parameters can destabilize the training. To address this, we optimize all model parameters and the queries in each batch with respect to $\mathcal{L}_{ent}$ and in each epoch with respect to $\mathcal{L}_{KL}^{(l)}$. Memory keys, on the other hand, are only updated at the end of each epoch by the gradients of $\mathcal{L}_{KL}^{(l)}$. This technique has also been applied in (Hassani & Haley, 2019; Caron et al., 2018) to avoid trivial solutions in deep clustering problem.

## 4 EXPERIMENTS

### 4.1 DATASETS

We use nine graph benchmarks including seven classification and two regression datasets to evaluate the proposed method. These datasets are commonly used in both graph kernel (Borgwardt & Kriegel, 2005; Yanardag & Vishwanathan, 2015; 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. The summary of these datasets is 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 $\beta$-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, 2015) is for predicting the field of a researcher given her

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 |
| ShortestPath (Borgwardt & Kriegel, 2005) | 42.32 | 76.43 | 78.86 | 59.10 |
| WL (Shervashidze et al., 2011) | 53.43 | 73.76 | 74.02 | 78.61 |
| WL Optimal (Kriege et al., 2016) | 60.13 | 75.26 | 79.04 | **80.74** |
| PatchySan (Niepert et al., 2016) | – | 75.00 | 76.27 | 72.60 |
| GraphSage (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 |
| 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 |

ego-collaboration graph.
**REDDIT-Binary** (Yanardag & Vishwanathan, 2015) is for predicting the type of community given a graph of online discussion threads.
**Tox21** (Challenge, 2014) is for predicting toxicity on 12 different targets.

For more information about the datasets and implementation details refer to Appendix A.2 and A.1, repectively.

### 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 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 path (Borgwardt & Kriegel, 2005), Weisfeiler-Lehman (WL) (Shervashidze et al., 2011), and WL Optimal Assignment (Kriege et al., 2016), and ten deep models. The results shown in Table 1 suggest that: (i) our 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 GMN achieves better result on the Enzymes, Proteins, and DD datasets. On COLLAB, our models are outperformed by a variant of DiffPpool (i.e., diffpool-det) (Ying et al., 2018) and WL Optimal Assignment (Kriege et al., 2016). The former is a GNN augmented with deterministic clustering algorithm[2], whereas the latter is a graph kernel method. We speculate that because of the high edge-to-node ratio of COLLAB, these augmentations help in extracting near-optimal cliques.

For the ESOL and Lipophilicity datasets, we follow the evaluation protocol in (Wu et al., 2018) and report the Root-Mean-Square Error (RMSE) for these regression benchmarks. Considering that these datasets contain initial edge attributes (refer to Appendix A.2 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 5 show 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, respectively. For further details on these datasets and the baselines see (Wu et al., 2018).

We also achieve state-of-the-art results on the Bace, Reddit-Binary, and Tox21 datasets. For more details see Appendix A.3.

[2]In diffpool-det assignment matrices are generated using a deterministic graph clustering algorithm.

Table 2: RMSE on ESOL and Lipophilicity.

| **Method** | **Dataset** | | | |
|---|---|---|---|---|
| | ESOL | | Lipophilicity | |
| | validation | test | validation | test |
| Multitask | $1.17 \pm 0.13$ | $1.12 \pm 0.19$ | $0.852 \pm 0.048$ | $0.859 \pm 0.013$ |
| Random Forest | $1.16 \pm 0.15$ | $1.07 \pm 0.19$ | $0.835 \pm 0.036$ | $0.876 \pm 0.040$ |
| XGBoost | $1.05 \pm 0.10$ | $0.99 \pm 0.14$ | $0.783 \pm 0.021$ | $0.799 \pm 0.054$ |
| GCN | $1.05 \pm 0.15$ | $0.97 \pm 0.01$ | $0.678 \pm 0.040$ | $0.655 \pm 0.036$ |
| MPNN | $0.55 \pm 0.02$ | $0.58 \pm 0.03$ | $0.757 \pm 0.030$ | $0.715 \pm 0.035$ |
| KRR | $1.65 \pm 0.19$ | $1.53 \pm 0.06$ | $0.889 \pm 0.009$ | $0.899 \pm 0.043$ |
| DAG | $0.74 \pm 0.04$ | $0.82 \pm 0.08$ | $0.857 \pm 0.050$ | $0.835 \pm 0.039$ |
| Weave | $0.57 \pm 0.04$ | $0.61 \pm 0.07$ | $0.734 \pm 0.011$ | $0.715 \pm 0.035$ |
| MemGNN (ours) | $\mathbf{0.53 \pm 0.03}$ | $\mathbf{0.54 \pm 0.01}$ | $\mathbf{0.555 \pm 0.039}$ | $\mathbf{0.556 \pm 0.023}$ |

### 4.3 Ablation Study

#### 4.3.1 Effect of edge features

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 the ESOL, Lipophilicity, and BACE datasets contain edge attributes, we use them as the benchmarks. Since nodes have richer features compared to edges, we set the node and edge feature dimensions to 16 and 4, respectively. The comparative performance evaluation of the two models on the ESOL dataset is shown in Appendix A.4 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 Effect of Topological Embedding

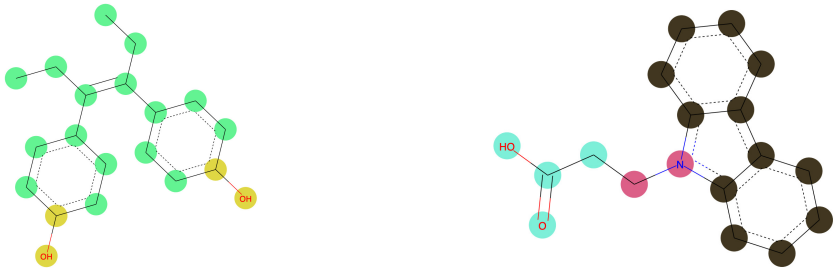
To investigate the effect of the topological embeddings on the GMN model, we evaluated three initial topological features including adjacency matrix, normalized adjacency matrix, and Random Walk with Restart (RWR). For further details on RWR, see section A.5. The results suggested that using the adjacency 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 with 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 sampling.

#### 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 ENZYMES 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 observed that keeping the number of parameters fixed, increasing the number of memory heads improves the 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.6

#### 4.3.5 What Do the Keys Represent?

Intuitively, the memory keys represent the cluster centroids and enhance the model performance by capturing meaningful structures and coarsening the graph. To investigate this intuition, we used the learned keys to interpret the knowledge learned by the 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, and 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. It is noteworthy that we tried initializing the memory keys using K-Means algorithm over the initial node embeddings to warm-start them but we did not observe any significant improvement over the randomly selected keys.

## 5 Conclusion

We proposed an efficient memory layer and two deep models for hierarchical graph representation learning. We evaluated the proposed models on nine graph classification and regression tasks and achieved state-of-the-art results on eight of them. We also experimentally showed that the learned representations can capture the well-known chemical features of the molecules. Our study indicated that node attributes concatenated with corresponding topological embeddings in combination with one or more memory layers achieves notable results without using message passing. We also showed that for the topological embeddings, the binary adjacency matrix is sufficient and thus no further preprocessing step is required for extracting them. Finally, we showed that although connectivity information is not directly imposed on the model, the memory layer can process node embeddings and properly cluster and aggregate the learned embeddings.

**Limitations**: In section 4.2, we discussed that on the COLLAB dataset, kernel methods or deep models augmented with deterministic clustering algorithm achieve better performance compared to our models. 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. Moreover, the results of the DD dataset reveals that our MemGNN model outperforms the GMN model which implies that we need message passing to perform better on this dataset. We speculate that this is because the DD dataset relies more on local information. The most important features to train an SVM on this dataset are surface features which have local behavior. This suggest that for data with strong local interactions, message passing is required to improve the performance.

**Future Directions**: We are planning to introduce a model based on the MemGNN and GMN architectures that can perform node classification by attending to the node embeddings and centroids of the clusters from different layers of hierarchy that the node belongs to. We are also planning to investigate the representation learning capabilities of the proposed models in self-supervised setting.

## 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.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems 14*, pp. 585–591. MIT Press, 2002.

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.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 132–149, 2018.

Tox21 Data Challenge. Tox21 data challenge 2014, 2014.

Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pp. 2702–2711, 2016.

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.

Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, pp. 855–864. ACM, 2016.

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.

Kaveh Hassani and Mike Haley. Unsupervised multi-task feature learning on point clouds. In *The IEEE International Conference on Computer Vision (ICCV)*, pp. 8160–8171, 2019.

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.

Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, pp. 1105–1114. ACM, 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.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th International Conference on Knowledge Discovery and Data Mining*, pp. 701–710. ACM, 2014.

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.

Jack Rae, Jonathan J Hunt, Ivo Danihelka, Timothy Harley, Andrew W Senior, Gregory Wayne, Alex Graves, and Timothy Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes. In *Advances in Neural Information Processing Systems*, pp. 3621–3629, 2016.

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 $\beta$-secretase 1 (bace-1) inhibitors using ligand based approaches. *Journal of chemical information and modeling*, 56(10):1936–1949, 2016.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pp. 2440–2448, 2015.

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, 2018.

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

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 International Conference on Knowledge Discovery and Data Mining*, pp. 1365–1374. ACM, 2015.

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 it using Adam (Kingma & Ba, 2014) optimizer. We trained the model for a maximum number of 2000 epochs and decayed the learning rate every 500 epochs by 0.5. The model uses batch-normalization (Ioffe & Szegedy, 2015), skip-connections, LeakyRelu activation functions, and dropout (Srivastava et al., 2014) for regularization. We decided the hidden dimension and number of model parameters using random hyper-parameter search strategy. The best performing hyper-parameters for the datasets are shown in Table 3.

## A.2 Dataset Statistics

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

Table 3: Hyper-parameters of the best models.

| Dataset | #Keys | #Heads | #Layers | Hidden Dimension | Batch |
|---|---|---|---|---|---|
| Enzymes | [10,1] | 5 | 2 | 100 | 20 |
| Proteins | [10,1] | 5 | 2 | 80 | 20 |
| DD | [16, 8, 1] | 5 | 3 | 120 | 64 |
| COLLAB | [32, 8, 1] | 5 | 3 | 100 | 64 |
| REDDIT-B | [32,1] | 1 | 2 | 16 | 32 |
| ESOL | [64,1] | 5 | 2 | 16 | 32 |
| Lipophilicity | [32,1] | 5 | 2 | 16 | 32 |
| BACE | [32,1] | 5 | 2 | 8 | 32 |

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 |
| REDDIT-B | classification | 2000 | 2 | 429.63 | 497.75 | 0 | 0 |
| Bace | classification | 1513 | 2 | 34.09 | 36.86 | 32 | 7 |
| Tox21 | classification | 8014 | 2 | 17.87 | 18.50 | 32 | 7 |
| ESOL | regression | 1144 | - | 13.29 | 13.68 | 32 | 7 |
| Lipophilicity | regression | 4200 | - | 27.04 | 29.50 | 32 | 7 |

### A.3 Additional graph classification benchmarks

In addition to results discussed in section 4.2, we report the evaluation results on three other graph classification benchmarks including BACE (Subramanian et al., 2016), Tox21 (Challenge, 2014), and Reddit-Binary (Yanardag & Vishwanathan, 2015) datasets. The BACE dataset provides quantitative binding results for a set of inhibitors of human $\beta$-secretase 1 (BACE-1), whereas the is Tox21 dataset is for predicting toxicity on 12 different targets. We follow the evaluation protocol in (Wu et al., 2018) and report the Area Under the Curve Receiver Operating Characteristics (AUC-ROC) measure for this task. Moreover, considering that the BACE and Tox21 datasets contain initial edge attributes, we train the MemGNN model and compare its performance to the baseline models reported in (Wu et al., 2018). The results shown in Table 5 suggest that our model achieves state-of-the-art results by absolute margin of $4.0\%$ AUC-ROC on BACE benchmark. The results also suggest that our model is competitive with the state-of-the-art GCN model on the Tox21 dataset.

Reddit-Binary dataset is for predicting the type of community (i.e., question-answer-based or discussion-based communities) given a graph of online discussion threads. In this dataset, nodes represent the users and edges denote interaction between users. To evaluate the performance of our models on this dataset, 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. The results reported in Table 6 show that the introduced GMN model achieves state-of-the-art accuracy by absolute margin of 0.44%.

### A.4 Effect of e-GAT

In section 4.3.1, we introduced e-GAT. Figure 3a illustrates the RMSE on the validation set of the ESOL dataset for a MemGNN model using GAT and e-GAT implementation as its query network, respectively. Since ESOL is a regression benchmark, we also plotted the $R^2$ score in Figure 3b. As shown in these figures, e-GAT performs better compared to GAT on both metrics.

### A.5 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

Table 5: AUC-ROC for BACE dataset.

| **Method** | validation | test |
|---|---|---|
| Logistic Regression | $0.719 \pm 0.003$ | $0.781 \pm 0.010$ |
| KernelSVM | $0.739 \pm 0.000$ | $0.862 \pm 0.000$ |
| XGBoost | $0.756 \pm 0.000$ | $0.850 \pm 0.008$ |
| Random Forest | $0.728 \pm 0.004$ | $0.867 \pm 0.008$ |
| IRV | $0.715 \pm 0.001$ | $0.838 \pm 0.000$ |
| Multitask | $0.696 \pm 0.037$ | $0.824 \pm 0.0006$ |
| Bypass | $0.745 \pm 0.017$ | $0.829 \pm 0.006$ |
| GCN | $0.627 \pm 0.015$ | $0.783 \pm 0.014$ |
| Weave | $0.638 \pm 0.014$ | $0.806 \pm 0.002$ |
| MemGNN (ours) | $\mathbf{0.859 \pm 0.000}$ | $\mathbf{0.907 \pm 0.000}$ |

Table 6: Mean validation accuracy over 10-folds.

| **Method** | **REDDIT-BINARY** |
|---|---|
| Graphlet (Shervashidze et al., 2009) | 78.04 |
| WL (Shervashidze et al., 2011) | 68.2 |
| DiffPool (Ying et al., 2018) | 85.95 |
| TopKPool (Gao & Ji, 2019) | 74.70 |
| SAGPool (Lee et al., 2019) | 73.90 |
| GMN (ours) | **86.39** |
| MemGNN (ours) | 85.55 |

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 12 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 $i$.

$$\vec{t_i} = p\tilde{\mathbf{A}}\vec{t_i} + (1-p)\vec{e_i} \tag{12}$$

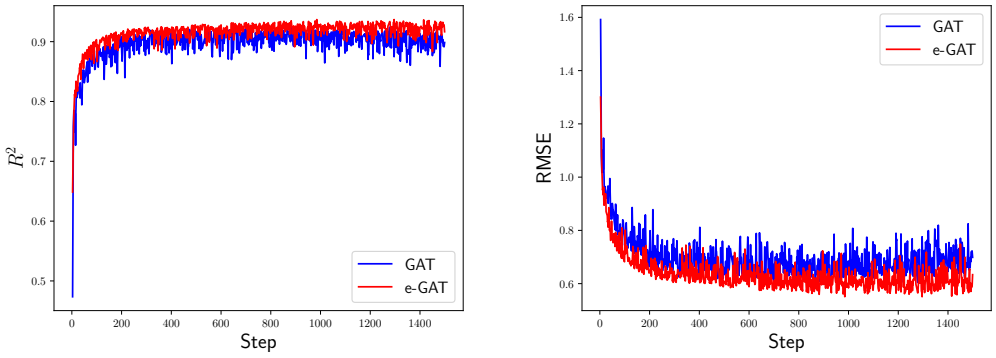
Solving this linear system results in $\vec{r_i}$ defined in equation 13

$$\vec{t_i} = (1-p)(I - c\tilde{\mathbf{A}})^{-1}\vec{e_i} \tag{13}$$

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 13 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).

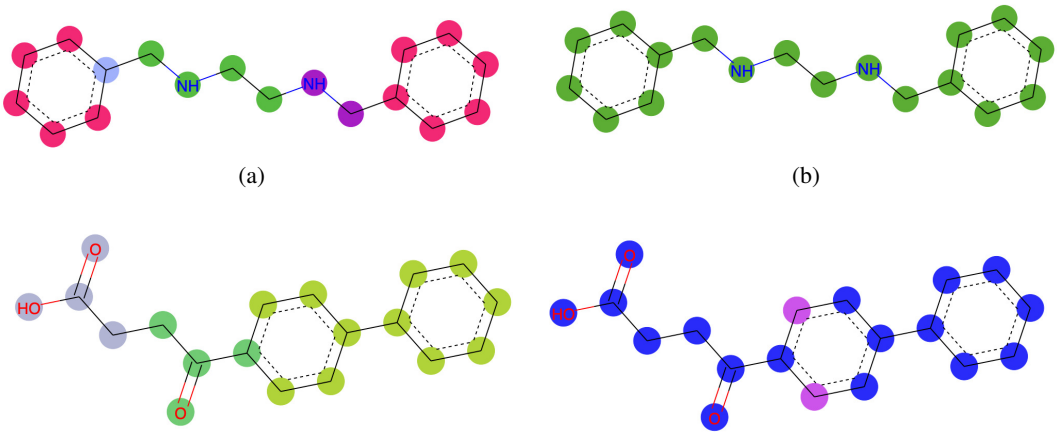
### A.6 Learned Clusters

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)

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



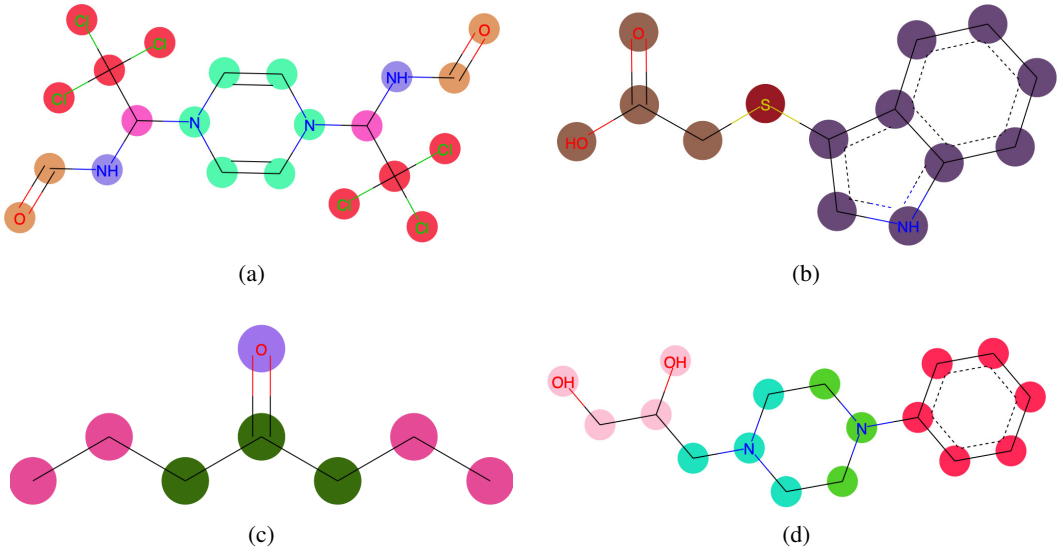

(a) (b)

(c) (d)

Figure 4: Figures (b) and (d) show clusters without using unsupervised clustering, whereas Figures (a) and (c) show clusters generated using the unsupervised clustering loss implying that the unsupervised loss helps the model with learning distinct and meaningful clusters.

Table 7: AUC-ROC for Tox21 dataset.

| **Method** | validation | test |
|---|---|---|
| Logistic Regression | $0.772 \pm 0.011$ | $0.794 \pm 0.015$ |
| KernelSVM | $0.818 \pm 0.010$ | $0.822 \pm 0.006$ |
| XGBoost | $0.775 \pm 0.018$ | $0.794 \pm 0.014$ |
| Random Forest | $0.763 \pm 0.002$ | $0.769 \pm 0.015$ |
| IRV | $0.807 \pm 0.006$ | $0.799 \pm 0.006$ |
| Multitask | $0.795 \pm 0.017$ | $0.803 \pm 0.012$ |
| Bypass | $0.800 \pm 0.008$ | $0.810 \pm 0.013$ |
| GCN | $0.825 \pm 0.013$ | $\mathbf{0.829 \pm 0.006}$ |
| Weave | $0.828 \pm 0.008$ | $0.820 \pm 0.010$ |
| MemGNN (ours) | $\mathbf{0.862 \pm 0.009}$ | $\mathbf{0.828 \pm 0.004}$ |




(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.