Memory-Based Graph Networks (K. Hassani et al., ICLR'20)

Zhiming Xu

zx2rw@virginia.edu

School of Engineering and Applied Science
University of Virginia

November 12, 2021

Table of Contents

Neural Networks w/ Memory

Memory-based Graph Networks

- Memory Addressing
- GMN
- MemGNN

Experimental Results

- Graph Classification
- Graph Regression

Conclusions & Future Directions

Neural Networks w/ Memory

Analogous to human's memory, memory used in neural networks is an encoding of knowledge that can facilitate the capability of NN-based learning models.

A typical example is Long Short-Term Memory (LSTM), a variation of recurrent neural networks (RNNs). It passes the memory along the temporal dimension, and controls the memory flow with some gating mechanism.

Memory Augmented Neural Networks

Memory augmented neural networks (MANNs) utilize an auxiliary memory with differentiable read-write operators, which allows the *storage, access, and use* of explicit past experiences.

This kind of memory is different from the hidden states in LSTMs in that it can store and retrieve longer term memories with fewer parameters.

Compare Long/Short-term Memory

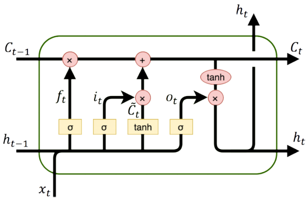


Figure 1: An LSTM cell, C_t is the short-term memory passed along the termporal dimension [1].

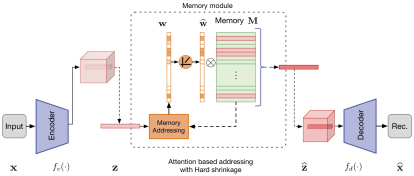


Figure 2: An autoencoder with a memory module **M** in between encoding and decoding module [2]

Approaches to Modeling Memory

The memory in NNs can be implemented in many forms, such as

- *Key-value memory* Accessing memory is like looking up a word in a dictionary, i.e., matching a key with its value.
- *Array-structured memory* Memory is encoded in a vector or matrix, accessing them is like applying a linear layer.

Table of Contents

Neural Networks w/ Memory

Memory-based Graph Networks

- Memory Addressing
- GMN
- MemGNN

Experimental Results

- Graph Classification
- Graph Regression

Conclusions & Future Directions

Memory-based Graph Networks

In *Memory-based Graph Networks* [3], the authors propose to augment graph neural networks (GNNs) with an array-structured memory module to learn graph representations from gradually coarsening nodes.

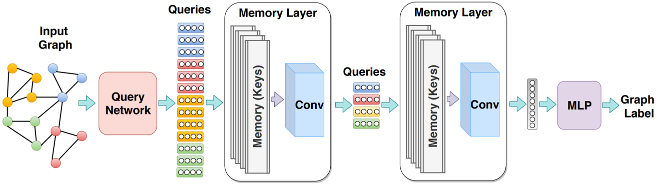


Figure 3: The proposed model in this paper. Query network initialize the node representations for subsequent memory layers. Each memory layer produces coarser node representations until a vector, i.e., graph representation is obtained.

Array-structured Memory Layer

Memory Layer

$\mathcal{M}^{(l)} : \mathbb{R}^{n_l \times d_l} \mapsto \mathbb{R}^{n_{l+1} \times d_{l+1}}$

- Takes in n_l query vectors in the l-th layer of size d_l.
- Outputs n_{l+1} query vectors of size d_{l+1} to the next layer with $n_{l+1} < n_l$.
- Each query vector is the node representations of the input graph. Each output is the next query to the next memory layer.
- $\mathcal{M}$ jointly performs feature transformation and graph pooling.

Memory Addressing

The memory addressing scheme can be formulated as computing the similarity between memory keys to a given query. The weight for a query q w.r.t the i-th key k_j is defined as $w_j = \mathrm{softmax}\left(\mathrm{d}\left(q, k_j\right)\right)$, and the retrieved memory is thus $r = \sum_j w_j k_j$.

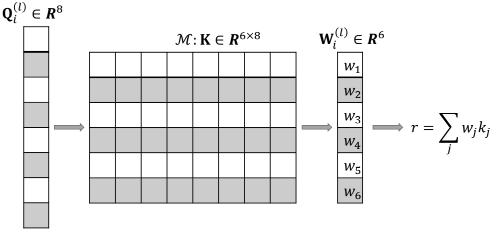

$\mathbf{Q}_i^{(l)} \in \boldsymbol{R}^8$
$\mathcal{M}: \mathbf{K} \in \boldsymbol{R}^{6\times 8}$
$\mathbf{W}_i^{(l)} \in \boldsymbol{R}^6$
w_1
w_2
w_3
w_4
w_5
w_6
$r = \sum_j w_j k_j$

Figure 4: A memory access that maps feature from $\mathbb{R}^8$ to $\mathbb{R}^6$.

Memory Addressing (cont.)

In this paper, input queries $\mathbf{Q}^{(l)} \in \mathbb{R}^{n_l \times d_l}$ are representations of the input graph, memory keys $\mathbf{K}^{(l)} \in \mathbb{R}^{n_{l+1} \times d_l}$ are treated as the cluster centroids of the queries.

To make the similarity metric $\mathrm{d}(\cdot, \cdot)$ clustering-friendly, Student's t-distribution is used, i.e.

$$w_j = C_j = \frac{\left(1 + \left\|\mathbf{q} - \mathbf{k}_j\right\|^2 / \tau\right)^{-\frac{r+1}{2}}}{\sum_i \left(1 + \left\|\mathbf{q} - \mathbf{k}_i\right\|^2 / \tau\right)^{-\frac{r+1}{2}}}. \qquad (1)$$

C_j is normalized weight for q w.r.t. j, i.e., the probability of assigning the vector representation $\mathbf{q}$ from $\mathbf{Q}$ to the cluster $\mathbf{k}_j$. τ is then degree of freedom of the Student's t-distribution.

Memory Addressing (cont.)

More than one group of keys $\mathbf{K}$ can be used to form a multi-head memory array $\mathbf{K} \in \mathbb{R}^{|h| \times n_{l+1} \times d_l}$ where $|h|$ denotes the number of heads. The weights for all node representations thus become a three-dimensional tensor $\left[\mathbf{C}_0^{(l)}, \mathbf{C}_1^{(l)}, \cdots, \mathbf{C}_{|h|-1}^{(l)}\right]$.

To aggregate this tensor, the three dimensions are treated as height, width, depth channels, and filtered with a 1×1 convolution Γ_ϕ.

$$\mathbf{C}^{(l)} = \operatorname{softmax}\left(\Gamma_\phi\left(\left[\mathbf{C}_0^{(l)} : \mathbf{C}_1^{(l)} : \cdots : \mathbf{C}_{|h|-1}^{(l)}\right]\right)\right) \in \mathbf{R}^{n_l \times n_{l+1}}. \quad (2)$$

Memory Addressing (cont.)

Coarser node representations are obtained through the weights multiplied with the corresponding key, followed by an MLP to generate the queries for next layer.

$$\begin{aligned} \mathbf{V}^{(l)} &= \mathbf{C}^{(l)\top}\mathbf{Q}^{(l)} \in \mathbb{R}^{n_{l+1}\times d_l}, \\ \mathbf{Q}^{(l+1)} &= \sigma\left(\mathbf{V}^{(l)}\mathbf{W}\right) \in \mathbb{R}^{n_{l+1}\times d_{l+1}}. \end{aligned} \tag{3}$$

Stacked Memory

The procedure above is denoted by $\mathcal{M}^{(l)}$, and it can be stacked multiple times for coarser and coarser node representations until a single vector as the graph representation.

$$\mathcal{Y} = \mathrm{softmax}\left(\mathrm{MLP}\left(\mathcal{M}^{(l)}\left(\mathcal{M}^{(l-1)}\left(\ldots\left(\mathcal{M}^{(0)}\left(\mathbf{Q}_0\right)\right)\right)\right)\right)\right). \tag{4}$$

$\mathbf{Q}_0$ is the initial node representations, which can be obtained through GNNs and other graph embedding methods. Two architectures that initialize $\mathbf{Q}_0 = f_q(g)$ differently, called GMN and MemGNN, are introduced by the authors.

GMN

GMN applies an initialization function $f_q(g)$ without message-passing. Nodes in GMN are treated as permutation-invariant set of representations.

The topological information of each node in the whole graph is preserved by embeddings from random walk with restart (RWR), i.e., diffusion matrix $\mathbf{S} \in \mathbb{R}^{n \times n}$ on the adjacency matrix $\mathbf{A}$. The authors propose to sort the embeddings row-wise to ensure permutation invariance.

$$\mathbf{S} = \mathrm{sorted}_{\mathrm{row}(\mathbf{D})}\left(\mathbf{D}\right), \mathbf{D} = \mathrm{GraphDiff}\left(\mathbf{A}\right). \qquad (5)$$

GMN (cont.)

The topological embeddings $\mathbf{S}$ are transformed by a parameter matrix, and then concatenated to node features $\mathbf{X}$, similar to the positional embeddings in Transformer.

$$\mathbf{Q}^{(0)} = \sigma\left(\left[\sigma\left(\mathbf{S}\mathbf{W}_0 \| \mathbf{X}\right)\mathbf{W}_1\right]\right). \tag{6}$$

It can be shown GMN is invariant to node permutations because of the sorted diffusion matrix.

MemGNN

For MemGNN, message-passing GNNs are used to initialize the node representations.

$$\mathbf{Q}^{(0)} = \mathrm{GNN}_\theta\left(\mathbf{A}, \mathbf{X}\right). \tag{7}$$

The authors propose an extension to graph attention network (GAT) as the query network $f_q(g)$ that also considers the weights of edges.

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

Loss Function

The model is trained with two loss functions, a supervised loss for the supervised task, and an unsupervised loss for the clusters in memory.

- The supervised loss comes from specific tasks, such as cross entropy for graph classification, and mean square error for graph regression.
- The unsupervised loss aims to ensure the embeddings correspond to clusters well in the latent space of memory. It is defined as the KL divergence between the clustering assignments $\mathbf{C}^{(l)}$ and the auxiliary distribution $\mathbf{P}^{(l)}$.

Loss Function (cont.)

The auxiliary distribution focuses on reducing clustering noises and strengthening samples with high confidence.

$$P_{ij}^{(l)} = \frac{\left(C_{ij}^{(l)}\right)^2 / \sum_i C_{ij}^{(l)}}{\sum_{j'} \left(C_{ij'}^{(l)}\right)^2 / \sum_i C_{ij'}^{(l)}}. \tag{9}$$

The KL divergence from cluster assignments from $\mathbf{C}^{(l)}$ to $\mathbf{P}^{(l)}$ is

$$\mathcal{L}_{\mathrm{KL}\left(\mathbf{P}^{(l)} \| \mathbf{Q}^{(l)}\right)} = \mathrm{KL}\left(\mathbf{P}^{(l)} \| \mathbf{Q}^{(l)}\right) = \sum_i \sum_j P_{ij}^{(l)} \log \frac{P_{ij}^{(l)}}{C_{ij}^{(l)}}. \tag{10}$$

Total Loss and Training Approach

The total loss is defined as follows.

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left(\lambda \mathcal{L}_{\textsf{sup}} + (1-\lambda) \sum_{l=1}^{L} \mathcal{L}_{\mathrm{KL}\left(\mathbf{P}^{(l)} \| \mathbf{Q}^{(l)}\right)} \right). \tag{11}$$

The optimization is performed every batch w.r.t $\mathcal{L}_{\textsf{sup}}$, and every epoch w.r.t $\mathcal{L}_{\mathrm{KL}\left(\mathbf{P}^{(l)} \| \mathbf{Q}^{(l)}\right)}$ to stabilize training and avoid trivial solutions.

Table of Contents

Neural Networks w/ Memory

Memory-based Graph Networks

- Memory Addressing
- GMN
- MemGNN

Experimental Results

- Graph Classification
- Graph Regression

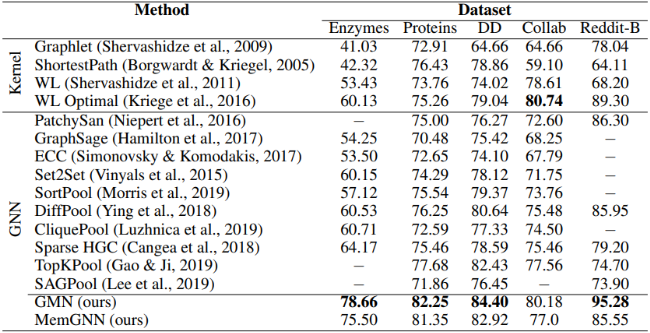
Conclusions & Future Directions

Graph Classification

Datasets: Enzymes, Proteins, DD (biology and chemistry); Collab (collaboration); Reddit-B (social media)

Performances:

Table 1: Mean validation accuracy over 10-folds.



	Method	Dataset				
		Enzymes	Proteins	DD	Collab	Reddit-B
Kernel	Graphlet (Shervashidze et al., 2009)	41.03	72.91	64.66	64.66	78.04
	ShortestPath (Borgwardt & Kriegel, 2005)	42.32	76.43	78.86	59.10	64.11
	WL (Shervashidze et al., 2011)	53.43	73.76	74.02	78.61	68.20
	WL Optimal (Kriege et al., 2016)	60.13	75.26	79.04	**80.74**	89.30
GNN	PatchySan (Niepert et al., 2016)	–	75.00	76.27	72.60	86.30
	GraphSage (Hamilton et al., 2017)	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	85.95
	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	79.20
	TopKPool (Gao & Ji, 2019)	–	77.68	82.43	77.56	74.70
	SAGPool (Lee et al., 2019)	–	71.86	76.45	–	73.90
	GMN (ours)	**78.66**	**82.25**	**84.40**	80.18	**95.28**
	MemGNN (ours)	75.50	81.35	82.92	77.0	85.55

Graph Regression

Datasets: ESOL (water solubility), Lipophilicity (tendency to dissolve into fats, oils, etc.)

Performances:

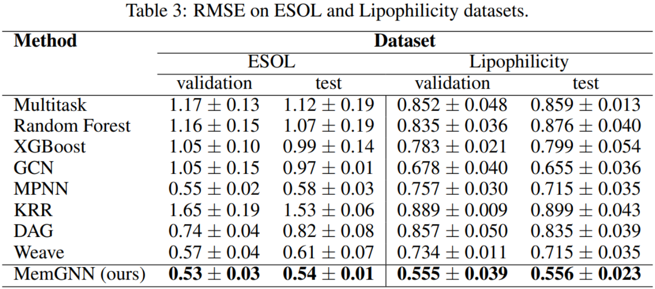


Table 3: RMSE on ESOL and Lipophilicity datasets.

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

Learned Clusters

The authors also show some example of the learned clustering assignments in chemical compounds. They can correspond to some meaning functional groups, especially trained with the unsupervised clustering loss.

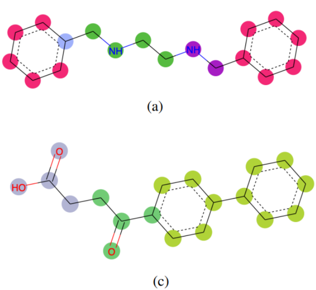


Figure 4: Figures (b) and (d) show computed clusters without using unsupervised clustering loss, whereas Figures (a) and (c) show the clusters learned using the unsupervised clustering loss. The visualizations suggest that the unsupervised loss helps the model in learning distinct and meaningful clusters.

Table of Contents

Neural Networks w/ Memory

Memory-based Graph Networks

- Memory Addressing
- GMN
- MemGNN

Experimental Results

- Graph Classification
- Graph Regression

Conclusions & Future Directions

Conclusions

- The authors propose an efficient memory layer to learn graph representations hierarchically.
- For graph-level representations learning, topological embeddings and memory layers can achieve performances on par with message-passing GNNs.
- The memory clusters learned with unsupervised clustering loss can correspond to meaningful substructures in graphs.

Future Directions

- The current framework cannot address node classification tasks, since it learns a global representation by hierarchically coarsening the nodes.
- Other topological initialization methods besides graph diffusion can be studied.
- The memory module with unsupervised clustering loss might be useful in self-supervised learning and unsupervised learning tasks.

Thank You for Your Attention

Q & A

References

[1] *Machine learning for engineers*, [Online]. Available: https://apmonitor.com/pds/index.php/Main/LongShortTermMemory.

[2] D. Gong, L. Liu, V. Le, *et al.*, "Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection," in *CVPR 2019*.

[3] A. H. K. Ahmadi, K. Hassani, P. Moradi, L. Lee, and Q. Morris, "Memory-based graph networks," in *ICLR 2020*.