Cache-Memory Gated Graph Neural Networks

Guixiang Ma
Intel Labs, guixiang.ma@intel.com

Vy Vo
Intel Labs, vy.vo@intel.com

Theodore Willke
Intel Labs, ted.willke@intel.com

Nesreen K. Ahmed
Intel Labs, nesreen.k.ahmed@intel.com

Abstract

While graph neural networks (GNNs) provide a powerful way to learn structured representations, it remains challenging to learn long-range dependencies in graphs. Recurrent gated GNNs only partly address this problem. In this paper, we propose a memory augmentation to a gated GNN which simply stores the previous hidden states in a cache. We show that the cache-memory gated GNN outperforms other models on synthetic datasets as well as tasks on real-world datasets that require long-range information.

1 Introduction

Graph neural networks (GNNs) provide a powerful framework for modeling complex structural data that exist in the real world [Scarselli et al., 2008]. A graph is a data structure that naturally represents arbitrary relationships among nodes, and is widely used in various domains, such as social, biological, and technological networks. GNNs assign each node a feature vector, and aggregate features from a node's neighbors using a recursive message passing paradigm. After k iterations of aggregation, the resulting feature vector captures information from the node's k-hop neighborhood [Xu et al., 2018]. This paradigm allows the GNN to learn structured representations that may be useful for a variety of tasks such as relational reasoning [Battaglia et al., 2018], navigation [Zweig et al., 2020], and program verification [Li et al., 2015].

Ideally, a deep GNN with L layers would effectively aggregate information from a node's L-hop neighborhood. However, as L increases, the receptive field of a given node increases exponentially. This makes learning long-range dependencies in GNNs an even more challenging problem than in recurrent neural networks, where recursion causes a linear, rather than exponential, increase in the information that needs to be captured by some fixed-length vector [Alon and Yahav, 2020]. This has been dubbed the *over-squashing* bottleneck of GNNs. Previous work has argued that this problem is quite widespread and is present in typical graph learning benchmarks [Alon and Yahav, 2020]. Recently, a long-range graph benchmark has been proposed to evaluate this specific shortcoming of many GNN architectures [Dwivedi et al., 2022]. This representational bottleneck is even more drastic for dynamic graphs, which are tasked with learning representations from topological structures that change over time [Xu et al., 2020, Kazemi et al., 2020].

One way of tackling this issue is to augment GNNs with some form of memory which can store information from more distant nodes [Xiong et al., 2020] or farther back in time [Ma et al., 2020, Rossi et al., 2020, Ma et al., 2022]. Advances in the design of memory mechanisms in other areas of machine learning [Graves et al., 2016], particularly in language modeling [Wu et al., 2022, Rae et al., 2019, Grave et al., 2017, Nematzadeh et al., 2020], suggest that this may be a rather effective way to make progress on long-range graph learning problems. In this work, we propose a cache-memory extension to a gated GNN which extends its ability to model long-range information. We show that

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

the cache-memory gated GNN can capture information in a larger problem radius than other networks, including other previously proposed memory-augmented GNNs.

2 Related work

Our work is related to other efforts to augment neural networks with different types of memory mechanisms. The classic example is some form of recurrent neural network (RNN), such as a long-short term memory (LSTM) network [Hochreiter and Schmidhuber, 1997] as well as more recent modifications such as Gated Recurrent Units (GRUs) [Cho et al., 2014]. These have been adapted to create gated graph neural networks (GGNNs), which incorporates both recurrence and a GRU gating mechanism in the propagation stage to improve long-term information propagation across the graph [Li et al., 2015]. The GGNN updates node information through recurrent propagation for a set number of timesteps N. At each timestep, the GRU update functions incorporate both information from neighboring nodes and information from the previous timestep to update the hidden state of each node v. While GGNNs may partly alleviate the over-squashing issue, previous work has shown that they still fail to propagate information from more distant nodes [Alon and Yahav, 2020].

Beyond modifying the recurrent unit itself, some work has added other types of memory modules to RNNs, such as a stack memory (i.e. a pushdown automaton) [Joulin and Mikolov, 2015, Yogatama et al., 2018] or a continuous neural cache [Grave et al., 2017]. The neural cache is a simple structure that stores the previous hidden states from the RNN, as well as a copy of the input. No transformation is applied to the information stored in memory or retrieved from memory. Our proposal is related to the continuous neural cache in that we store an untransformed history of the most recent hidden states in a memory – however, we do not store an additional copy of the input.

Beyond this, there has been work to augment other ANNs with differentiable memory mechanisms [Graves et al., 2014, 2016]. While these have not been adapted to work with GNNs, other work has proposed a variety of memory mechanisms for different GNN tasks (see Ma et al. [2022] for review). This includes anything from a static knowledge base [Moon et al., 2019] to some form of key-value memory [Khasahmadi et al., 2020, Ma et al., 2020]. These memory-augmented GNNs may store global graph information [Xiong et al., 2020] or retain some local structural information, such as position in an image [Khademi, 2020]. Our proposal is most related to work that focuses on augmenting alongside some type of recurrent propagation.

3 Framework

Cache-memory Gated GNNs We propose a new memory-augmented GNN framework based on GGNNs, called "Cache-memory Gated GNNs (CGNNs)", for capturing longer-range dependencies in the learned representation of graphs. Specifically, we propose to store the past hidden activations of gated GNNs in an external, non-differentiable cache memory and access them through a dot product with the current hidden activation (Eq. 7, Figure 1). This allows the history of aggregation computations to be incorporated when updating the node representations. While we define the formulations based on the propagation rule given in Li et al. [2015], our memory proposal is generally applicable to other forms of recurrent GNNs.

Given a graph $G = (V, E)$, where V is the set of nodes, $E \subseteq V \times V$ is the set of edges. Let $|V|$ denote the number of nodes in the graph, $\mathbf{A}$ be the adjacency matrix and $\mathbf{h}_v^t$ denote the node representation for node v at timestep t. Then the propagation rule of the gated GNN [Li et al., 2015] is given by:

$$\mathbf{a}_v^t = \mathbf{A}_v^T[\mathbf{h}_1^{t-1} \cdots \mathbf{h}_{|V|}^{t-1}]^T + \mathbf{b} \tag{1}$$

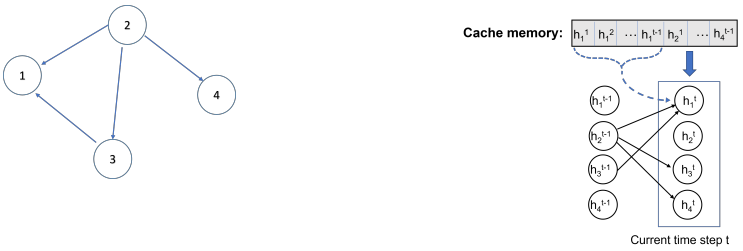
$$\mathbf{z}_v^t = \sigma(\mathbf{W}^z\mathbf{a}_v^t + \mathbf{U}^z\mathbf{h}_v^{t-1}) \tag{2}$$

$$\mathbf{r}_v^t = \sigma(\mathbf{W}^r\mathbf{a}_v^t + \mathbf{U}^r\mathbf{h}_v^{t-1}) \tag{3}$$

$$\tilde{\mathbf{h}_v^t} = tanh(\mathbf{W}\mathbf{a}_v^t + \mathbf{U}(\mathbf{r}_v^t \odot \mathbf{h}_v^{t-1})) \tag{4}$$

$$\mathbf{h}_v^t = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \tilde{\mathbf{h}_v^t} \tag{5}$$

Equations 3 and 4 define the update gate $\mathbf{z}$ and the reset gate $\mathbf{r}$, and depend on both the adjacency matrix and the previous timestep. These both alter the final hidden state $\mathbf{h_v^t}$. For the CGNN, we



(a) Example input graph (b) One unrolled timestep in CGNN

Figure 1: An example of an unrolled timestep in CGNN. For the current time t, each node representation is updated by incorporating information from its neighboring nodes and from its own history of hidden states stored in the cache memory.

redefine Equation 5 to represent another intermediate, $\mathbf{h'_v}^{\mathbf{t}}$. We then add a final equation which sums over the Hadamard product of the cached hidden states with $\mathbf{h'_v}^{\mathbf{t}}$,

$$\mathbf{h'}_v^t = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \tilde{\mathbf{h}_v^t} \tag{6}$$

$$\mathbf{h}_v^t = \lambda \mathbf{h'}_v^t + (1 - \lambda) \sum_{i=1}^{t-1} (\mathbf{h'}_v^t \odot \mathbf{h}_v^i) \tag{7}$$

Together Equations 1-4, 6, and 7 represent our proposed CGNN. Note that the history of hidden states $\mathbf{h}_1^1 \cdots \mathbf{h}_{|V|}^{t-1}$ is stored without any transformation to the hidden states. The hyperparameter λ which determines how much of the next hidden state relies on memories retrieved from the cache. The size of the memory cache (i.e. how many hidden states are stored) can also be set, but in the majority of experiments shown here we set this value to its maximum, which is $N - 1$, where N is the total number of timesteps considered.

4 Experiments

We consider five different GNN models in the performance comparison, including GCN [Kipf and Welling, 2016], MemGCN (the memory-augmented GCN from Xiong et al. [2020]), GAT [Veličković et al., 2017], GGNN [Li et al., 2015], and our proposed CGNN. We trained all models in an end-to-end manner using Adam optimizer with the default Pytorch configuration. We repeated each experiment three times and report the average and standard deviation. We tuned parameters such as the embedding size and λ on the validation set. For the NeighborsMatch dataset, we follow the settings in Alon and Yahav [2020] and set the number of layers in the GNNs to $r + 1$, where r is the problem radius. For the other two datasets, we treat the number of graph layers as a parameter and vary it from 1 to 5 on the validation set to obtain the optimal number. We also tune the embedding size of each GNN model on the validation set by doing grid search from {16, 32, 64, 128}. Both the MemGCN and CGNN have a hyperparameter λ which controls the weight they put on the component related to external memory. We tuned λ for both models on each validation set by doing grid search from {0.1, 0.2, ...,0.9}. We use $N = 6$ as the number of timesteps considered in GGNN and CGNN.

NeighborsMatch. Alon and Yahav [2020] introduced a synthetic benchmark problem that requires long-range information. The goal is to predict which of the other graph nodes has the same number of neighbors n as the target node. We use the same settings as Alon and Yahav [2020] to generate 32,000 graphs per problem radius ranging from 2 to 8, and use 60% for training and 20% each for validation and test. We set the number of layers in the GNNs to $r + 1$, where r is the problem radius. Models are trained for 2000 epochs, $1e - 3$ learning rate and batch size 1024.

Reachability analysis. The DeepDataFlow datasets [Cummins et al., 2021] contains a set of graphs constructed from real-world LLVM-IR files for compiler analysis. We use 10,000 graphs from this dataset for the reachability analysis, which predicts a binary label indicating if a node is reachable from the root node of the graph. Due to a large class imbalance, we evaluate performance by

computing the precision of the predicted labels on the reachable class only [Ma et al., 2021]. Models are trained for 500 epochs at a learning rate of $1e-3$ and a batch size of 64.

Image analysis. Dwivedi et al. [2022] preprocessed the Pascal VOC 2011 image dataset [Everingham et al., 2010] to create the long-range graph learning benchmark PascalVOC-SP. Each image is represented as a graph, and each node corresponds to a region of the image with some semantic segmentation label. We use the same data splits and experimental setup as Dwivedi et al. [2022] and train for 1000 epochs at a learning rate of $1e-4$ with a batch size of 32.

Performance Analysis. Figure 2 shows the training accuracy of the GNN models over different problem radii for the NeighborsMatch task. We observe that as the problem radius increases beyond 3, all GNNs started to decline in the training performance, especially the GNN and MemGCN models. This is consistent with the findings in Alon and Yahav [2020] regarding the bottleneck of GNNs in fitting long-range signals due to over-squashing. Our proposed CGNN model outperforms all the baseline GNNs in both training and test accuracy (Figure 2) at each problem radius, especially for larger problem radii $r > 4$. For instance, our model improves the testing accuracy by up to 12% (31% GGNN vs. 43% CGNN) on $r = 6$. Our results demonstrate that the cache memory augmentation in CGNN improves long-range representation learning in a graph for the NeighborsMatch problem.

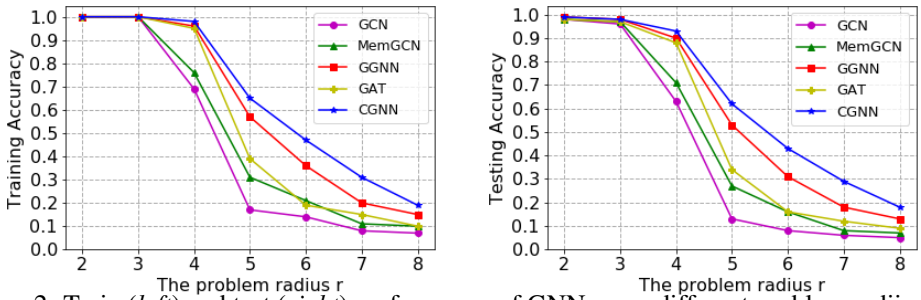


Figure 2: Train (*left*) and test (*right*) performance of GNNs over different problem radii on NeighborsMatch task.

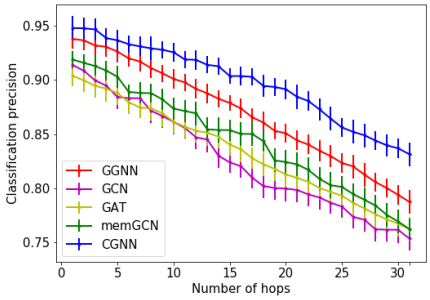


Figure 3: Classification precision for predicting reachable nodes at various numbers of hops.

Table 1: Node classification precision for reachability task, averaged across hops.

Model	Precision
GCN	0.826 ± 0.011
MemGCN	0.845 ± 0.010
GAT	0.833 ± 0.011
GGNN	0.865 ± 0.009
CGNN	**0.892 ± 0.009**

Table 2: Node classification F1 scores on the PascalVOC-SP dataset.

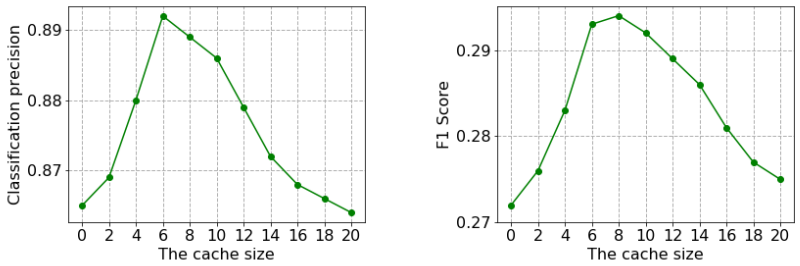
Model	F1 score
GCN	0.125 ± 0.011
MemGCN	0.196 ± 0.018
GAT	0.183 ± 0.034
GGNN	0.272 ± 0.030
CGNN	**0.294 ± 0.023**

Table 1 and Figure 3 show the performance of the models on the DeepDataFlow dataset. We find that the CGNN outperforms all of the other GNNs on the reachability analysis. Notably, the performance improvement grows with an increasing number of hops from the root node (Figure 3), illustrating that the CGNN is particularly helpful for learning long-range interactions in code graphs.

Moreover, we observe from Table 2 that CGNN achieved the highest F1 score in the node classification task on PascalVOC-SP. This indicates CGNN's superior capability in learning long-range dependencies between regions on images compared to the baseline GNNs.

Computational and Memory Complexity Analysis. While the cache memory mechanism in CGNN brings the performance improvements on the graph learning tasks as demonstrated above, we now analyse the cost introduced by the cache to the gated GNN. According to Eq. 7, the time complexity introduced for each layer by using the cache memory in updating node representations is $O(N \times |V| \times D)$, where D is a constant that represents the dimension of the node hidden features, N is the number of timesteps, and $|V|$ is the number of nodes. The memory complexity is also $O(N \times |V| \times D)$ because we set the size of the memory cache to its maximum, $N - 1$.

In practical applications, the cache size can be varied to determine how many hidden states to store in the memory. In order to provide some insights on how the cache size in CGNN may affect its performance, we conducted experiments on the DeepDataFlow dataset and the PascalVOC-SP with different cache sizes from the range of $\{0, 2, 4, 6, \cdots, 20\}$, where we use $N = 20$ for the number of timesteps. When the cache size k is less than $N - 1$, we only store the most recent k hidden states in the cache. As we can see from Figure 4, as the cache size increases, the performance of CGNN climbs first until reaching the peak and then starts to decline. This indicates that while storing and using the past hidden states in memory could help improve the learning capabilities of the gated GNNs, incorporating too much prior history information could hinder the performance improvement by the memory. Therefore, selecting the proper cache size or having a mechanism to control what is stored in memory is important when applying such memory models in practice.



(a) DeepDataFlow (b) PascalVOC-SP

Figure 4: Performance of CGNN with different cache sizes on DeepdataFlow and PascalVOC-SP

5 Conclusion and Future Directions

We found that the cache-memory gated GNN (CGNN) outperforms all the models we tested, including the GGNN, GCN, and memory-augmented MemGCN, on tasks that require long-range information to be propagated across the graph. This includes the synthetic NeighborsMatch task, as well tasks on real-world datasets such as images. However, the tasks that we show here only address memory for information across the graph, rather than across time. Future work should investigate the use of the CGNN to model temporal data such as video. The current proposal stores all recent timesteps, which may be suboptimal when modeling long temporal sequences. Some mechanism to control what is stored in memory may be required. We also did not investigate how the CGNN may address over-squashing in dynamic graphs. Finally, recent work has suggested that long-range relationships can be captured by Graph Transformer architectures [Dwivedi et al., 2022, Rampášek et al., 2022]. Future investigations should compare memory-augmented models like the CGNN to Transformers, and evaluate the strengths and weaknesses of each.

References

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Tal Ben-Nun, Alice Shoshana Jakobovits, and Torsten Hoefler. Neural code comprehension: A learnable representation of code semantics. *Advances in Neural Information Processing Systems*, 31, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Chris Cummins, Zacharias V Fisches, Tal Ben-Nun, Torsten Hoefler, Michael FP O'Boyle, and Hugh Leather. Programl: A graph-based program representation for data flow analysis and compiler optimizations. In *International Conference on Machine Learning*, pages 2244–2253. PMLR, 2021.

Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *arXiv:2206.08164*, Jun 2022. URL `http://arxiv.org/abs/2206.08164`.

Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2): 303–338, Jun 2010. ISSN 1573-1405. doi: 10.1007/s11263-009-0275-4.

Edouard Grave, Armand Joulin, and Nicolas Usunier. Improving neural langauge models with a continuous cache. page 9, 2017.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv:1410.5401 [cs]*, Oct 2014. URL `http://arxiv.org/abs/1410.5401`.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626): 471–476, Oct 2016. ISSN 1476-4687. doi: 10.1038/nature20101.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *arXiv:1503.01007 [cs]*, Jun 2015. URL `http://arxiv.org/abs/1503.01007`.

Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. (arXiv:1905.11485), Apr 2020. doi: 10.48550/arXiv.1905.11485. URL `http://arxiv.org/abs/1905.11485`.

Mahmoud Khademi. Multimodal Neural Graph Memory Networks for Visual Question Answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7177–7188, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.643.

Amir Hosein Khasahmadi, Kaveh Hassani, Parsa Moradi, Leo Lee, and Quaid Morris. Memory-based graph networks. page 16, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Chen Ma, Liheng Ma, Yingxue Zhang, Jianing Sun, Xue Liu, and Mark Coates. Memory Augmented Graph Neural Networks for Sequential Recommendation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):5045–5052, April 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i04.5945.

Guixiang Ma, Yao Xiao, Mihai Capotă, Theodore L. Willke, Shahin Nazarian, Paul Bogdan, and Nesreen K. Ahmed. Learning code representations using multifractal-based graph networks. In *2021 IEEE International Conference on Big Data (Big Data)*, page 1858–1866, Dec 2021. doi: 10.1109/BigData52589.2021.9671685.

Guixiang Ma, Vy Vo, Theodore Willke, and Nesreen K. Ahmed. Memory-augmented graph neural networks: A neuroscience perspective. (arXiv:2209.10818), Sep 2022. doi: 10.48550/arXiv.2209.10818. URL `http://arxiv.org/abs/2209.10818`.

Seungwhan Moon, Pararth Shah, Anuj Kumar, and Rajen Subba. Memory Graph Networks for Explainable Memory-grounded Question Answering. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 728–736, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/K19-1068.

Aida Nematzadeh, Sebastian Ruder, and Dani Yogatama. On Memory in Human and Artificial Language Processing Systems. In *"Bridging AI and Cognitive Science" at International Conference on Learning Representations*, 2020.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive Transformers for Long-Range Sequence Modelling. In *International Conference on Learning Representations*, September 2019.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. (arXiv:2205.12454), Jul 2022. URL `http://arxiv.org/abs/2205.12454`.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *arXiv:2006.10637 [cs, stat]*, October 2020.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing Transformers. *arXiv:2203.08913 [cs]*, March 2022.

Tao Xiong, Liang Zhu, Ruofan Wu, and Yuan Qi. Memory Augmented Design of Graph Neural Networks. September 2020.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv:2002.07962*, Feb 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. Memory architectures in recurrent neural network language models. 2018.

Aaron Zweig, Nesreen Ahmed, Theodore L Willke, and Guixiang Ma. Neural algorithms for graph navigation. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020.

A Appendix

A.1 Detailed Descriptions of Datasets

NeighborsMatch Alon and Yahav [2020] introduce this synthetic benchmark problem that requires long-range information. For some example graph in the dataset, each node has an alphabetical label (e.g. A, B, C...) and a number of neighbors n. Given some target node, the task is to predict which of the other graph nodes has the same number of neighbors n and return the corresponding alphabetical label (e.g. node C). Every example in the dataset has a different mapping between the label and number of neighbors, so solving this problem requires message propagation and matching for every graph in the dataset. To control the receptive field size required to solve the problem, each target node is the root of a binary tree of some depth r, a.k.a. the problem radius.

Reachability analysis The DeepDataFlow datasets [Cummins et al., 2021] contains a set of graphs constructed from a corpus of real-world LLVM-IR files from various sources (e.g. C, C++, OpenCL, etc.) for compiler analysis. We use 10,000 graphs from this dataset for the control reachability analysis, which predicts if a node (i.e., instruction) is reachable from the root node (i.e., starting code instruction) of the graph. This set of graphs have an average of 1370 nodes and 2390 edges, and each node comes with inst2vec [Ben-Nun et al., 2018] features and a binary label indicating if it is reachable from the root.

PascalVOC-SP Dwivedi et al. [2022] preprocessed the Pascal VOC 2011 image dataset [Everingham et al., 2010] to create this long-range graph learning benchmark. Each image is represented as a graph, and each node corresponds to a region of the image with some semantic segmentation label. There are a total of 21 class labels, and the dataset has an average of 479 nodes and 2710 edges. We use the same data splits and experimental setup as Dwivedi et al. [2022]: 8498 training graphs, 1428 validation, and 1429 test.

A.2 Details of the Compared Methods

We compare the performance of the proposed CGNN model with four other GNN models: GCN Kipf and Welling [2016], MemGCN Xiong et al. [2020], GAT Veličković et al. [2017], and GGNN Li et al. [2015]. For the NeighborsMatch dataset, we follow the settings in Alon and Yahav [2020] and set the number of layers in the GNNs to $r + 1$, where r is the problem radius. For the other two datasets, we treat the number of graph layers as a parameter and vary it from 1 to 5 on the validation set to obtain the optimal number. We also tune the embedding size of each GNN model on the validation set by doing grid search from {16, 32, 64, 128}. Both the MemGCN and CGNN have a hyperparameter λ which controls the weight they put on the component related to external memory. We tuned λ for both models on each validation set by doing grid search from {0.1, 0.2, ...,0.9}. We use $N = 6$ as the number of timesteps considered in GGNN and CGNN.

A.3 Additional Results

In addition to the training and testing performance plots shown in Figure 2 of Section 4.2, here we also provide the detailed numbers for these performance with average accuracy as well as standard deviation in Table 3 and Table 4.

Table 3: Training accuracy of the GNN models over different problem radii on the NeighborsMatch dataset (mean ± std).

problem radius	2	3	4	5	6	7	8
GCN	1.0 ± 0.0	1.0 ± 0.0	0.69 ± 0.02	0.17 ± 0.03	0.14 ± 0.03	0.08 ± 0.03	0.07 ± 0.03
MemGCN	1.0 ± 0.0	1.0 ± 0.0	0.76 ± 0.02	0.31 ± 0.03	0.21 ± 0.03	0.11 ± 0.03	0.10 ± 0.03
GAT	1.0 ± 0.0	1.0 ± 0.0	0.95 ± 0.01	0.39 ± 0.02	0.20 ± 0.02	0.15 ± 0.02	0.10 ± 0.03
GGNN	1.0 ± 0.0	1.0 ± 0.0	0.96 ± 0.01	0.57 ± 0.02	0.36 ± 0.02	0.20 ± 0.03	0.15 ± 0.03
CGNN	1.0 ± 0.0	1.0 ± 0.0	0.98 ± 0.01	0.65 ± 0.02	0.47 ± 0.02	0.31 ± 0.02	0.19 ± 0.03

Table 4: Testing accuracy of the GNN models over different problem radii on the NeighborsMatch dataset (mean ± std).

problem radius	2	3	4	5	6	7	8
GCN	0.98 ± 0.01	0.96 ± 0.01	0.63 ± 0.02	0.13 ± 0.03	0.08 ± 0.03	0.06 ± 0.03	0.05 ± 0.03
MemGCN	0.98 ± 0.01	0.97 ± 0.01	0.71 ± 0.02	0.27 ± 0.03	0.16 ± 0.03	0.08 ± 0.03	0.07 ± 0.03
GAT	0.98 ± 0.01	0.97 ± 0.01	0.88 ± 0.02	0.34 ± 0.02	0.16 ± 0.02	0.12 ± 0.02	0.08 ± 0.03
GGNN	0.99 ± 0.01	0.98 ± 0.01	0.90 ± 0.01	0.53 ± 0.02	0.31 ± 0.02	0.18 ± 0.03	0.13 ± 0.03
CGNN	0.99 ± 0.01	0.98 ± 0.01	0.93 ± 0.01	0.62 ± 0.02	0.43 ± 0.02	0.29 ± 0.02	0.18 ± 0.03