
CS660 - Midway Report

Applying Mamba to GNNs

Jyotirmaya Shivottam

23226001

School of Computer Science

National Institute of Science Education and Research

Bhubaneswar, Odisha

`jyotirmaya.shivottam@niser.ac.in`

Abstract

In this report, we present the work done so far towards the semester project for the CS660 - Machine Learning course. As mentioned in our proposal, we attempt to combine the efficiency and effectiveness of a novel Structured State Space Model, called “Mamba” [1] on long-range sequence modeling, with message-passing graph neural networks. We do so by devising a new state update method for the embeddings of the graph nodes. We test this approach on the Planetoid [2] dataset and conduct several experiments. Finally, we discuss the insights from the experiments and list future work that can further develop this approach.

1 Introduction

1.1 A Refresher on Graph Neural Networks

We begin by reviewing the basics of GNNs, particularly Message Passing Neural Networks (MPNNs) [3]. First, let us introduce the mathematical notation we use throughout this work. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph with vertex (node) set, \mathcal{V} and edge set, \mathcal{E} . Let $N = |\mathcal{V}|$. For simplicity, we assume that the graph is undirected and unweighted, but the methods we discuss in this report are easily extensible to directed and weighted graphs. We use $\mathcal{N}[i]$ to represent the closed set of neighbors of node i . Moreover, let $X \in \mathbb{R}^{N \times F}$ be the node feature matrix, where F is the number of features per node, i.e., $X = [x_1, x_2, \dots, x_N]^T$, where $x_i \in \mathbb{R}^F$. It should be noted that, in set notation, X and intermediate feature maps are multisets, as two nodes can have identical features. We use $A \in \mathbb{R}^{N \times N}$ to denote the graph’s adjacency matrix.

Message Passing Neural Networks (MPNNs) MPNNs are a class of GNNs that aggregate & update node features based on the features of their neighbors. The following set of iterative steps defines these models:

$$m_i^{(l)} = \text{MESSAGE} \left(\{h_j^{(l-1)} : j \in \mathcal{N}[i]\} \right) \quad (1)$$

$$a_i^{(l)} = \text{AGGREGATE} \left(\{m_j^{(l-1)} : j \in \mathcal{N}[i]\} \right) \quad (2)$$

$$h_i^{(l)} = \text{UPDATE} \left(h_i^{(l-1)}, a_i^{(l)} \right) \quad (3)$$

Here, l denotes a particular GNN layer, $h_i^{(l)}$ is the hidden state of node i at layer l , $m_i^{(l)}$ is the “message” generated for node i using the hidden states at layer l , and $a_i^{(l)}$ denotes the aggregated message for

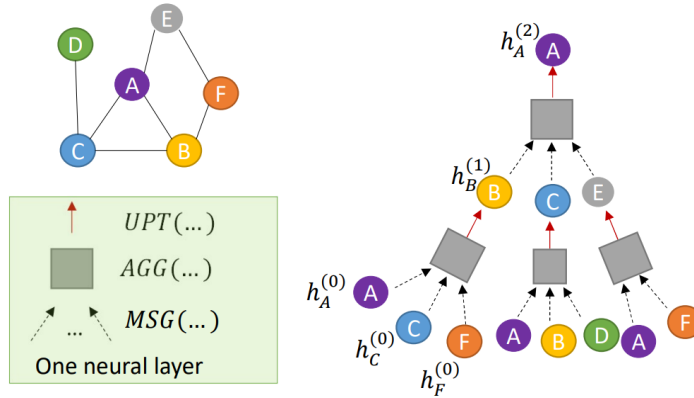


Figure 1: Application of a Message-Passing GNN layer, shown for one node (Taken from [4])

node i at layer l . The functions MESSAGE, AGGREGATE, and UPDATE are typically parameterized by neural network layers, such as multi-layer perceptrons (MLPs) and are shared across all nodes in the graph. Figure 1 illustrates these. The final output is obtained by applying a READOUT function to the final hidden states, i.e., $O = \text{READOUT}(\{h_i^{(L)} : i \in [N]\})$, where L is the number of layers in the GNN, at either node or graph level, depending on the task at hand, which can be node classification, graph classification, etc.

Since graph data is naturally unordered [5], i.e., the elements of \mathcal{V} are unordered, the neural network layers representing the functions mentioned above that operate on node features must be permutation-invariant [5]. In fact, the primary reason to learn on graphs is the inductive bias or symmetry that graph data is inherently order-independent. As such, MPNNs usually enforce that AGGREGATE remains permutation-invariant by using functions such as sum, mean, or max for fixed (unlearned) aggregation or by combining these with learned weights.

1.2 Related Works

In practice, different MPNN variants, such as Graph Convolution (GCN) [6], Gated Graph Convolution (GGsNN) [7], GraphSAGE [8] and Graph Attention (GAT) [9, 10] or Graph Transformers [11], differ mainly in the way they define the functions – MESSAGE, AGGREGATE, and UPDATE, and hence, in the way they collect and update the node features. For instance, GCN updates node features via a first-order approximation of spectral graph convolutions in a way quite similar to fixed image convolution kernels; GGsNN uses the Gated Recurrent Unit update for node features, followed by a neighbourhood sum aggregation, while GraphSAGE trains a set of aggregator functions operating on different hops on sampled node neighbourhoods. In contrast, Graph Attention (GAT) uses a self-attention mechanism on node neighborhoods with sum aggregation.

As discussed by Xu et al. [12], all of the aforementioned models are at most as expressive as the 1-Weisfeiler-Lehman (1-WL) graph isomorphism test. This becomes particularly problematic for graph data with long-range dependencies, e.g., in many real-world applications, such as social network analysis, recommendation systems, bioinformatics, or when learning on heterophilic graphs. Moreover, approaches like GAT or Graph Transformers, which can technically model long-range dependencies as long as multi-hop neighborhoods are considered, are hampered by the $\mathcal{O}(N^2)$ complexity scaling of the attention mechanism, which makes them infeasible for large graphs. In particular, Graph Transformers treat graphs as a sequence of nodes, much like sequences of tokens in natural language processing, and process nodes within pre-defined context windows. To mitigate the quadratic complexity of attention, they usually perform ‘attention sparsification’, where nodes are subsampled prior to computing attention scores.

In the next sections, we discuss Mamba, a new time-varying state-space model proposed recently by Gu et al. [1], that comes equipped with a ‘selection’ mechanism, that scales linearly with sequence length and how we can adapt it for MPNNs.

2 Selective State Space-based Sequence Modeling via Mamba

Recently, Gu et al. [1] proposed a novel sequence modeling framework, based on structured state space models (SSM), with a selection mechanism, called *Mamba*, that is capable of learning long-range dependencies in sequential data in an input-dependent manner. SSMs relate a continuous input sequence, $x(t) \in \mathbb{R}$, to a continuous output sequence, $y(t) \in \mathbb{R}$, through an implicit latent state, $h(t) \in \mathbb{R}^d$, using the following first-order ordinary differential equation:

$$h'(t) = Ah(t) + Bx(t) \quad (4)$$

$$y(t) = Ch(t) + Dx(t) \quad (5)$$

Here, $A \in \mathbb{R}^{d \times d}$, $B \in \mathbb{R}^{d \times 1}$, $C \in \mathbb{R}^{1 \times d}$, and $D \in \mathbb{R}$ are the state transition matrix, the input matrix, the output matrix, and the feedforward matrix, respectively. We omit D going forward, as it is usually modeled as a skip connection. As we work with discrete data, these equations, and hence, the associated parameters, are discretized using Zero-Order Hold (ZOH) to obtain the following update (recurrence) equations:

$$\bar{A} \stackrel{\text{ZOH}}{:=} \exp(\Delta A) \quad (6)$$

$$\bar{B} \stackrel{\text{ZOH}}{:=} (\Delta A)^{-1}(\exp(\Delta A) - \mathbb{I}) \cdot \Delta B \quad (7)$$

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t \quad (8)$$

$$y_t = Ch_t \quad (9)$$

Here, $x_t, y_t \in \mathbb{R}^F$, the subscript, t , is the time step along the sequence, and Δ denotes a learned (adaptive) discretization step size. In contrast with prior SSM architectures, in Mamba, the Δ parameter is made input-dependent, which indirectly makes A, B & C input-dependent, using the following learned projections:

$$B \leftarrow s_B(x) = \text{Linear}_N(x) \quad (10)$$

$$C \leftarrow s_C(x) = \text{Linear}_N(x) \quad (11)$$

$$s_\Delta(x) \leftarrow \text{Broadcast}_D(\text{Linear}(x)) \quad (12)$$

$$\Delta \leftarrow \text{softplus}(\text{Parameter} + s_\Delta(x)) \quad (13)$$

$$\bar{A}, \bar{B} \leftarrow \text{ZOH}(\Delta, A, B) \quad (14)$$

These specific choices have been made due to a connection to the RNN gating mechanism for particular s_Δ and τ_Δ (cf. Theorem 1 in [1]). Here, Δ acts as a **gating**, or what the Mamba authors term as a **selection** mechanism, that decides which parts of the input and the latent space are relevant to the current output, and hence, imparts shift-variant context-awareness to the Mamba block, allowing it to focus on or filter out different aspects or features of the input sequence, thereby, learning time-varying long-range dynamics. In addition, the state transition matrix, A , is “structured” as a diagonal matrix to stabilize the state update over long sequences and simplify the computation, because a diagonal A makes the discretization (exponentiation) trivial to calculate. Several fixed initializations exist for A . They are reasoned via the HiPPO theory [13], which allows these structured SSMs to memorize the input history in a controlled manner. The latent state, $h(t)$, is responsible for storing “compressed context” (h is d -dimensional, where $d \ll L$, the sequence length), as opposed to a transformer’s full uncompressed context, which leads to Mamba’s high efficiency. In general, Mamba’s computational complexity only scales as $\mathcal{O}(L)$ as opposed to $\mathcal{O}(L^2)$ for transformer-based approaches. Finally, a clever hardware-aware implementation of an associative scan algorithm for the recurrence enables efficient training on modern hardware, making Mamba a practical choice for sequence modeling tasks. Figure 2 depicts a typical Mamba block, where the input is linearly projected (“state expansion”) before passing through a short 1D depth-wise convolution and the main SSM update, with interleaved non-linearities. The final output of the block is a deflating linear projection. The structure of this block is inspired by the vanilla transformer and its simplifications used in prior SSM architectures (See the discussion in [1]).

3 Our Approach – Expressive Graph Mamba (EGM)

In this section, we describe how we integrate the Mamba block into a GNN to obtain what we call the “Expressive Graph Mamba” (EGM) layer, based on the discussion in previous sections, particularly

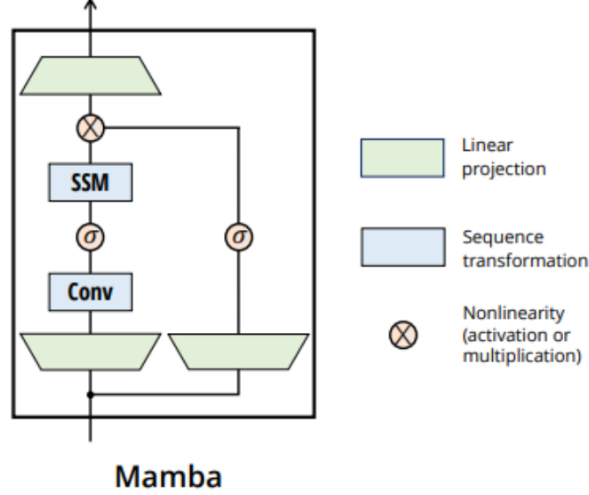


Figure 2: A Mamba Block [1] with state expansion (Linear(\cdot))

on Mamba in Section 2. Now, let us delineate our Expressive Graph Mamba layer. Like other MPNN variants, we redefine one of the MPNN iterative steps – UPDATE. We start by considering a sequence of graph representations at different time steps, i.e., X^0, X^1, \dots, X^L , where $X^0 = X$ is the initial node feature matrix, and L is the number of time steps (a hyperparameter). If edge attributes are present, they can be easily added to either X or the adjacency matrix, A . We use a shared Mamba block to update the node features at each time step, thereby leveraging the selection mechanism to allow the model to self-select the relevant aspects of the node features. Node features at time step 0, i.e., $X^0 = [x_1^0, \dots, x_N^0]^T$ form the input to our model. Then, for each node, $x_i \in \mathbb{R}^F$ (where F is the number of features per node), we proceed as follows:

UPDATE via Mamba block:

$$h_i^{(t+1)} = \bar{A}h_i^t + \bar{B}x_i^t \quad (15)$$

$$y_i^{(t+1)} = Ch_i^t \quad (16)$$

Here, $h \in \mathbb{R}^d$, where d denotes the model dimension, and $y \in \mathbb{R}^F$ is the output of the Mamba block. The parameters, \bar{A}, \bar{B}, C , are shared across all nodes in the graph and are learned during training (via the discretization, Δ , for \bar{A} and \bar{B}). Here, we can choose to evolve the hidden states for a fixed number of time steps, say T , before passing the output to the following AGGREGATE step. This can be understood as a form of warmup for the Mamba block, similar to warmup for certain RNN or attention-based models.

The aggregation of node features in our case can be learned or fixed, e.g., using the usual sum, max, or mean aggregation. For learned aggregation, we combine the Mamba update with a shared attention mechanism, in the same vein as GAT or GATv2 [9, 10], reinforcing the model’s ability to capture long-range dependencies in graph-structured data.

AGGREGATE via shared attention

$$\alpha_{ij} = \frac{\exp \left[\sigma \left(\bar{a}^{(t+1)T} \left[W^{(t+1)} y_i^{(t+1)} \parallel W^{(t+1)} y_j^{(t+1)} \right] \right) \right]}{\sum_{k \in \mathcal{N}[i]} \exp \left[\sigma \left(\bar{a}^{(t+1)T} \left[W^{(t+1)} y_i^{(t+1)} \parallel W^{(t+1)} y_k^{(t+1)} \right] \right) \right]} \quad (17)$$

$$x_i^{(t+1)} := y_i^{(t+1)} = \rho \left(\sum_{j \in \mathcal{N}[i]} \alpha_{ij} W^{(t+1)} y_j^{(t+1)} \right) \quad (18)$$

Here, $W^{(t+1)} \in \mathbb{R}^{F \times F}$ is a shared weight matrix, which transforms the output of the Mamba block; $\bar{a}^{(t+1)} \in \mathbb{R}^{2F}$ is a weight vector characterizing a single-layer feedforward NN, while σ and ρ are

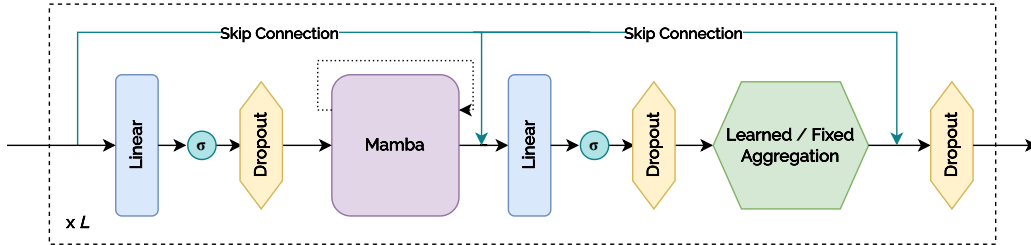


Figure 3: An EGM Block. Here, σ denotes SiLU non-linearity, while the aggregation can be fixed (sum, max, mean) or learned. L designates the number of EGM blocks or layers.

non-linearities. Furthermore, $(\cdot)^T$ denotes transposition, while \parallel represents concatenation. Since the Mamba block already calculates linear projections of the input features via state-expansion, W can be optional, as \vec{a} alone should be sufficient for adding learnability to the computation of the attention coefficients. For this report, however, we utilize the entire GAT block.

Our goal here is to use the attention mechanism to aggregate the node features, while the SSM dynamics handles the latent space updates. One can relate this to the MPNN steps listed in Section 1.1 – message generation, aggregation, and update, (Eqs. 1, 2, and 3). The SSM dynamics combines the message generation and update steps, while the shared attention mechanism combines the update and aggregation steps. After evolving the graph for L time steps, we obtain the final graph representation, $X^L = [x_1^L, \dots, x_N^L]^T$, which can be used for various downstream tasks. Similar to MPNNs, we use a learned READOUT to obtain the final output, $O = \text{READOUT}(X^L)$, via a single Linear layer. Figure 3 shows a single EGM layer. It can be stacked analogously to other MPNN blocks like GCN to allow information flow between multi-hop neighbourhoods. The following section describes the experiments performed using the EGM layer and discusses the corresponding results.

4 Experiments & Results

Table 1: Dataset Information

| Name | #nodes | #edges | #features | #classes |
|----------|--------|--------|-----------|----------|
| Cora | 2,708 | 10,556 | 1,433 | 7 |
| CiteSeer | 3,327 | 9,104 | 3,703 | 6 |
| PubMed | 19,717 | 88,648 | 500 | 3 |

Due to time constraints for this report¹, we test the effectiveness of our approach on only the Planetoid (Cora, Citeseer, and Pubmed) datasets [2], that comprise citation networks in various domains, with the nodes in the graphs representing academic documents and the edges characterizing citation links. We used the Planetoid dataset class from PyTorch Geometric² to load the data, as it comes with pre-made binary masks for the training, validation, and tests splits. In short, the Planetoid datasets are relatively fast to train on, enabling us to iterate on the design of our network architecture quickly. Table 1 summarizes the details of these datasets.

The previously mentioned MPNN models – GCN, GAT, GATv2, GraphSAGE, and GIN, serve as the baselines for comparison in this report. Specifically, we train all these models for the node classification task. As depicted in Fig. 3, we use interleaved skip connections, SiLU activation, and dropout layers to facilitate stable training for our EGM model. To further stabilize training, we use Layer Normalization [14] between the EGM layers. Here, the Mamba block is taken from the mamba-ssm³ Python library by the original Mamba authors. The rest of the model has been implemented from scratch in PyTorch⁴ and PyTorch Geometric. Table 2 outlines the remaining

¹All of the experiments were performed in a 10-day period in late February 2024.

²<https://pytorch-geometric.readthedocs.io/en/latest/>

³<https://github.com/state-spaces/mamba>

⁴<https://pytorch.org/>



Figure 4: Performance comparison of our model (EGM) with baselines on the Planetoid datasets. Here, mlp is a simple multi-layer perception model with sum aggregation.

training-related information. We use CrossEntropy as the loss function, as the task is multi-class node classification. Note that we use the RAdam optimizer instead of Adam or AdamW, as RAdam constrains the variance of the adaptive learning rate during the early stages of the training [15], which leads to more stable training during the initial epochs in our case.

Table 2: Training Information. (*) indicates the best-performing component.

| | |
|---------------|---|
| Loss Function | CrossEntropy |
| Scheduler | ExponentialLR (*), LinearLR, OneCycleLR |
| Optimizer | RAdam (*), Adam, AdamW |
| Weight Decay | 0.0005 |
| Learning Rate | [0.0001, 0.01] |
| EGM Layers | {1, 2, 3, 4, 6, 8} |
| Epochs | [150, 500] |

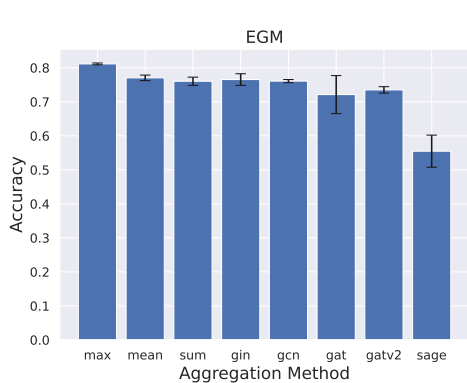
We experiment with different numbers of EGM layers, but for every comparison, we maintain a similar structure for all the models with an unconstrained parameter budget. Note that, due to time limitations, no hyperparameter search was performed for any of the models. Each experiment was run with 3 - 5 different seeds (depending on dataset-dependent VRAM requirement), and the mean and standard deviation of the test accuracy were computed. In total, roughly 400 experiments were conducted and tracked on WandB⁵, resulting in a total of 1,439 runs. In the next section, we compare against the aforementioned baselines and discuss the results of this work.

5 Results & Discussion

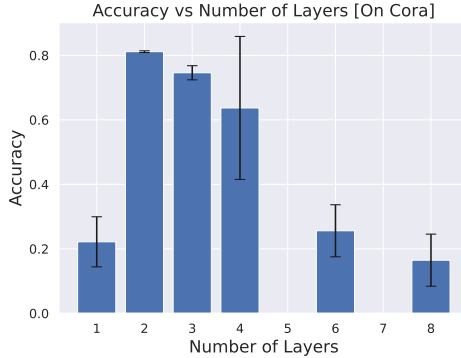
In this section, we present the results of the experiments conducted using our approach on the Planetoid dataset.

The performance plots depicted in Fig. 4 demonstrate that EGM achieves comparable results to other MPNN baselines. Its performance falls within the standard deviation range of other models across all datasets we could test. Interestingly, we observe that employing fixed aggregation functions, such as mean, sum, and particularly max, yields superior results compared to learned aggregation schemes that integrate message-passing. This observation is illustrated in Figure 5(a). Additionally, this suggests that the previously proposed GAT-based aggregation method may not be suitable for our model. A probable cause here could be the potential over-smoothing of node features, as the learned aggregators in our setup incorporate their own aggregation schemes, except for simpler aggregators like GIN or GCN. Furthermore, as depicted in Fig. 5(b), the accuracy of our model notably decreases as the number of layers increases. While over-smoothing may contribute to this phenomenon too, we also observed large activations in the Mamba block beyond a certain number of layers (> 3), suggesting that the Mamba block struggles to learn stably. This is at odds with Graph Transformer

⁵<https://wandb.ai/js-exps/graphmamba>



(a) Testing different fixed aggregation schemes for EGM. Surprisingly, fixed aggregation performs better than learned.



(b) Model accuracy drops drastically with increasing variance, as the number of EGM layers increases. No tests were done for layer count = 5 and 7.

Figure 5: Different experiments performed on the EGM architecture

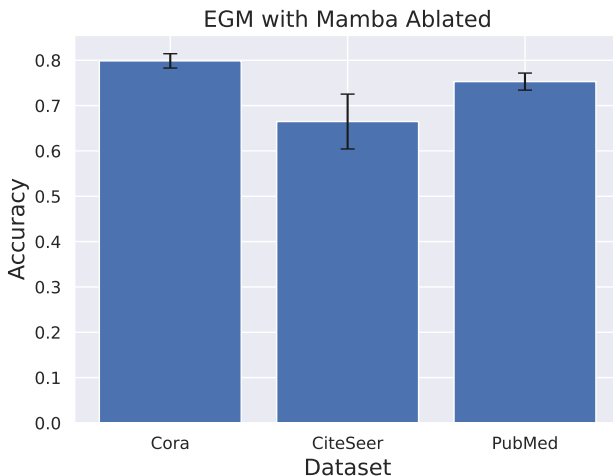


Figure 6: Accuracy scores after ablating the Mamba block from the EGM block. Unexpectedly, the performance without Mamba is comparable to the complete EGM architecture (cf. Figure 4).

networks, which can effectively learn on graphs (albeit when nodes are treated sequentially) with as many as six layers [11].

Importantly, we made an intriguing observation during ablative experiments on the Mamba block. As depicted in Fig. 6, removing the Mamba block from our architecture, i.e., excluding the block from Fig. 3, does not significantly alter the overall performance. This suggests that the Mamba block may not be training effectively on these datasets. An alternative perspective on removing the Mamba block from EGM is to consider the GIN update equation [12]:

$$h_v^{(l)} = \text{MLP}^{(l)} \left((1 + \epsilon^{(l)}) \cdot h_v^{(l-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(l-1)} \right). \quad (19)$$

Here, l represents a GNN layer, and $(1 + \epsilon^{(l)})$, with a learnable ϵ , acts as a pointwise transformation of the node embeddings. We can replace this with another Linear layer, making Eq. 19 analogous to learning a transformation for a node embedding via an MLP. Similarly, when the Mamba block in Fig. 3 is ablated, the EGM block performs a related set of operations:

$$h_v^{(l)} = h_v^{(l)} \oplus \text{MLP}^{(l)} \left(\text{MLP}^{(l)}(h_v^{(l-1)}) \oplus \sum_{u \in \mathcal{N}(v)} h_u^{(l-1)} \right). \quad (20)$$

Here, we have replaced the $+$ symbol with \oplus to emphasize the pointwise sum. This discussion indicates that the ablated EGM learns similarly to the GIN, explaining the good performance of the simplified Mamba-less model.

As to why Mamba struggles to learn effectively, we propose that treating each node as having its own sequence of states in a static graph (i.e., with a per-node sequence length of 1) contradicts Mamba’s expectation of long sequences and an implementation detail that assumes the sequence length to be the fastest-changing dimension⁶, thereby creating a **bottleneck**. It is unlikely that this bottleneck can be alleviated for static graphs unless we transition to operating on a sequence of nodes, similar to Graph Transformers, which, however, violates the graph inductive bias. In the next section, we conclude this work and discuss potential directions for future research.

6 Future Work

Our future work will address the identified limitations and further investigate the potential of incorporating Mamba into GNNs. Specifically, we plan to conduct experiments with deeper models on much larger graphs, e.g., from the LRGB dataset, as well as on dynamic graphs, e.g., from the Temporal Graph Benchmark (TGB) [16], to evaluate the scalability and adaptability of the EGM architecture. Dynamic graphs, in particular, lend themselves to our approach more naturally than static graphs, as learning on such graphs entails processing a “sequence of graph-states”. We are currently working on implementing these experiments, so that more conclusive comparisons can be made with more baselines. Another minor extension to our approach is to simplify the construction of the Mamba layer by using only the primary structured SSM equations for state updates and by decoupling the remaining layers, for example, linear projections and non-linearities, so that we can experiment with more modular architectures. The goal here would be to work around or remove the sequence length bottleneck due to the current way the Mamba block processes data. This would also allow us to examine the importance of the selection mechanism on graph-based learning tasks. Further, we are exploring the potential of more sophisticated normalization schemes, such as Spectral Normalization [17], that should, in principle, stabilize the training by rescaling the impact of large activations on weight updates.

References

- [1] Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *arXiv e-prints*, art. arXiv:2312.00752, December 2023. doi: 10.48550/arXiv.2312.00752.
- [2] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting Semi-Supervised Learning with Graph Embeddings. *arXiv e-prints*, art. arXiv:1603.08861, March 2016. doi: 10.48550/arXiv.1603.08861.
- [3] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.
- [4] Pan Li and Jure Leskovec. *The Expressive Power of Graph Neural Networks*, pages 63–98. Springer Nature Singapore, Singapore, 2022. ISBN 978-981-16-6054-2. doi: 10.1007/978-981-16-6054-2_5. URL https://doi.org/10.1007/978-981-16-6054-2_5.
- [5] Petar Veličković. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology*, 79:102538, April 2023. ISSN 0959-440X. doi: 10.1016/j.sbi.2023.102538. URL <http://dx.doi.org/10.1016/j.sbi.2023.102538>.
- [6] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL <https://openreview.net/forum?id=SJU4ayYgl>.
- [7] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL <http://arxiv.org/abs/1511.05493>.

⁶https://github.com/state-spaces/mamba/blob/main/mamba_ssm/ops/selective_scan_interface.py#L192

- [8] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS' 17*, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.
- [9] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=rJXMpikCZ>.
- [10] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=F72ximsx7C1>.
- [11] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. In *Advances in Neural Information Processing Systems*, pages 11960–11970, 2019.
- [12] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=ryGs6iA5Km>.
- [13] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *CoRR*, abs/2008.07669, 2020. URL <https://arxiv.org/abs/2008.07669>.
- [14] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.
- [15] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020. URL <https://openreview.net/forum?id=rkgz2aEKDr>.
- [16] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. *Advances in Neural Information Processing Systems*, 2023.
- [17] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=B1QRgziT->.