Recurrent Distance Filtering for Graph Representation Learning

Yuhui Ding [1] Antonio Orvieto [2] Bobby He [1] Thomas Hofmann [1]

Abstract

Graph neural networks based on iterative one-hop message passing have been shown to struggle in harnessing the information from distant nodes effectively. Conversely, graph transformers allow each node to attend to all other nodes directly, but lack graph inductive bias and have to rely on ad-hoc positional encoding. In this paper, we propose a new architecture to reconcile these challenges. Our approach stems from the recent breakthroughs in long-range modeling provided by deep state-space models: for a given target node, our model aggregates other nodes by their shortest distances to the target and uses a linear RNN to encode the sequence of hop representations. The linear RNN is parameterized in a particular diagonal form for stable long-range signal propagation and is theoretically expressive enough to encode the neighborhood hierarchy. With no need for positional encoding, we empirically show that the performance of our model is comparable to or better than that of state-of-the-art graph transformers on various benchmarks, with a significantly reduced computational cost. Our code is open-source at `https://github.com/skeletondyh/GRED`.

1. Introduction

Graphs are ubiquitous for representing complex interactions between individual entities, such as in social networks (Tang et al., 2009), recommender systems (Ying et al., 2018) and molecules (Gilmer et al., 2017), and have thus attracted a lot of interest from researchers seeking to apply deep learning to graph data. Message passing neural networks (MPNNs) (Gilmer et al., 2017) have been the dominant approach in this field. These models iteratively update the representation of a target node by aggregating the representations of its neighbors. Despite progress in semi-supervised node classification tasks (Kipf & Welling, 2017; Veličković et al., 2018), MPNNs have been shown to have difficulty in effectively harnessing the information of distant nodes (Alon & Yahav, 2021; Dwivedi et al., 2022b). To reach a node that is k hops away from the target node, an MPNN needs at least k layers. As a result, the receptive field for the target node grows exponentially with k, including many duplicates of nodes that are close to the target node. The information from such an exponentially growing receptive field is compressed into a fixed-size representation, making it insensitive to the signals from distant nodes (a.k.a. over-squashing (Topping et al., 2022; Di Giovanni et al., 2023)). This limitation may hinder the application of MPNNs to tasks that require long-range reasoning.

Inspired by the success of attention-based transformer architectures in modeling natural languages (Vaswani et al., 2017; Devlin et al., 2019) and images (Dosovitskiy et al., 2021), several recent works have adapted transformers for graph representation learning to address the aforementioned issue (Ying et al., 2021; Kim et al., 2022; Chen et al., 2022; Ma et al., 2023). Graph transformers allow each node to attend to all other nodes directly through a global atten-

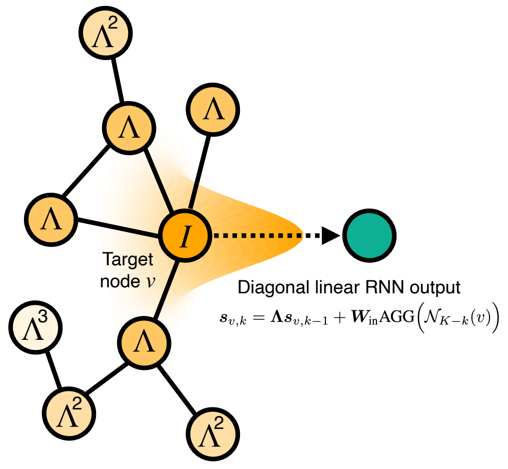

Λ^2
Λ
Λ
Λ
I
Target node v
Diagonal linear RNN output
$s_{v,k} = \Lambda s_{v,k-1} + W_{\text{in}}\text{AGG}\Big(\mathcal{N}_{K-k}(v)\Big)$
Λ^3
Λ
Λ^2
Λ^2

Figure 1. Illustration of the filtering effect on the neighborhood, induced by the linear RNN. The filter weight is determined by the eigenvalues $\mathbf{\Lambda}$ of the transition matrix and the shortest distance to the target node. We expand on this in Section 3.

[1]Department of Computer Science, ETH Zürich [2]ELLIS Institute Tübingen, MPI-IS, Tübingen AI Center. Correspondence to: Yuhui Ding <yuhui.ding@inf.ethz.ch>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

tion mechanism, and therefore make the information flow between distant nodes easier. However, a naive global attention mechanism alone doesn't encode any structural information about the underlying graph. Hence, state-of-the-art graph transformers rely on ad hoc positional encoding (e.g., eigenvectors of the graph Laplacian) as extra features to incorporate the graph inductive bias. There is no consensus yet on the optimal type of positional encoding. Which positional encoding to use and its associated hyper-parameters need to be tuned carefully (Rampášek et al., 2022). Besides, while graph transformers have empirically shown improvement on some graph benchmarks compared with classical MPNNs, the former are much more computationally expensive (Dwivedi et al., 2022b).

Captivated by the above challenges and the need for powerful, theoretically sound and computationally efficient approaches to graph representation learning, we propose a new model, Graph Recurrent Encoding by Distance (GRED). Each layer of our model consists of a permutation-invariant neural network (Zaheer et al., 2017) and a linear recurrent neural network (Orvieto et al., 2023b) that is parameterized in a particular diagonal form following the recent advances in state space models (Gu et al., 2022b; Smith et al., 2023). To generate the representation for a target node, our model categorizes all other nodes into multiple sets according to their shortest distances to the target node. The permutation-invariant neural network generates a representation for each set of nodes that share the same shortest distance to the target node, and then the linear recurrent neural network encodes the sequence of the set representations, starting from the set with the maximum shortest distance and ending at the target node itself. Since the order of the sequence is naturally encoded by the recurrent neural network, our model can encode the neighborhood hierarchy of the target node without the need for positional encoding. The architecture of GRED is illustrated in Figure 2.

The diagonal parameterization of the linear RNN (Orvieto et al., 2023b) has been shown to make long-range signal propagation more stable than a vanilla RNN, and enables our model to effectively harness the information of distant nodes. More specifically, it enables our model to directly learn the eigenvalues of the transition matrix, which control how fast the signals from distant nodes decay as they propagate towards the target node (see Figure 1 for an illustration), and at the same time allows efficient computation with parallel scans. Furthermore, while the use of a linear recurrent neural network is motivated by long-range signal propagation, we theoretically prove its expressive power in terms of injective functions over sequences, which is of independent interest, and based on that we conclude that our model is more expressive than 1-WL (Xu et al., 2019). We evaluate our model on a series of graph benchmarks to support its efficacy. The performance of our model is significantly better than that of MPNNs, and is comparable to or better than that of state-of-the-art graph transformers while requiring no positional encoding and significantly reducing computation time.

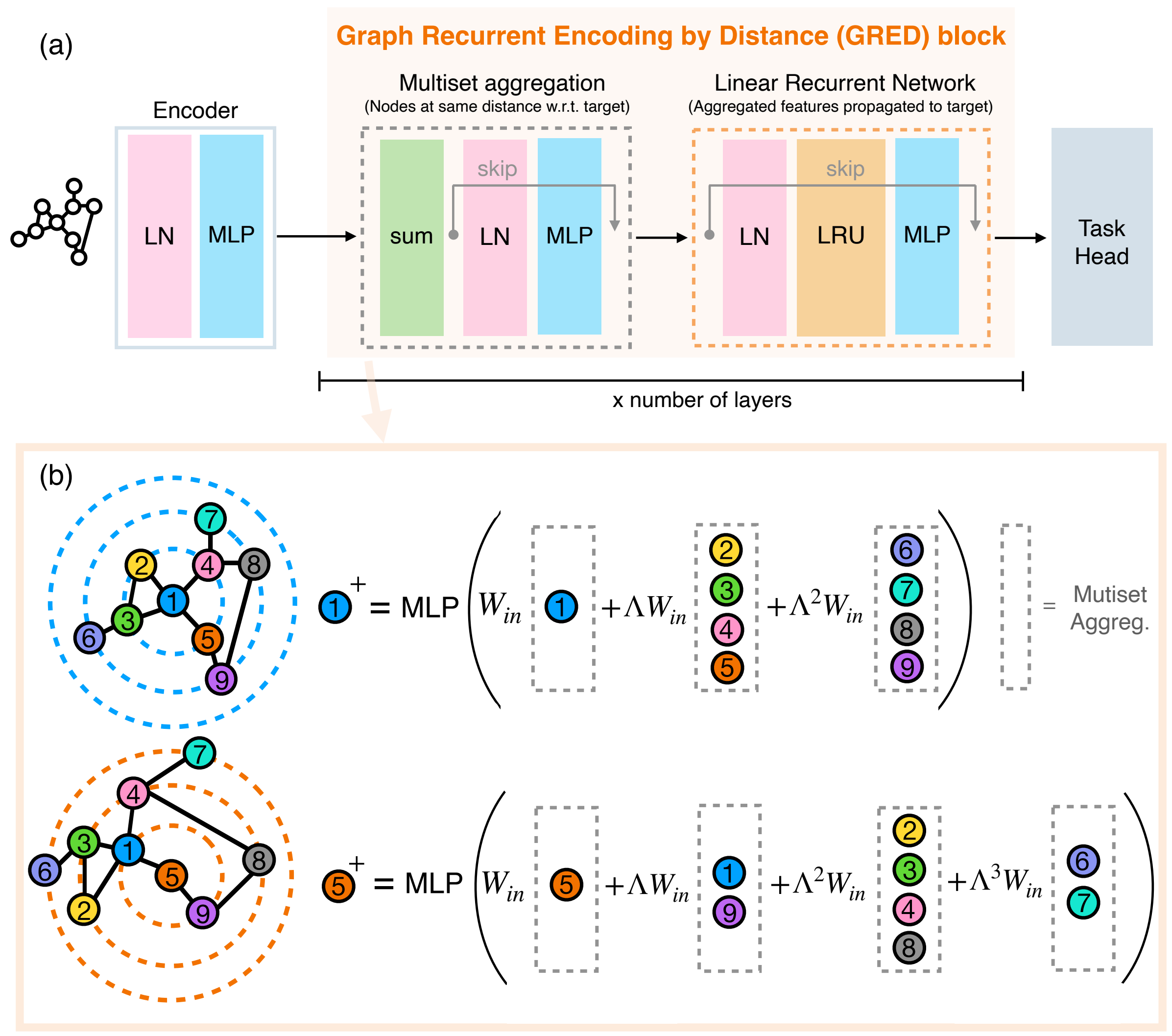
To summarize, the main contributions of our paper are as follows:

- We propose a principled new model for graph representation learning that can effectively and efficiently harness the information of distant nodes. The architecture is composed of permutation-invariant neural networks and linear recurrent neural networks with diagonal parameterization.
- We theoretically prove that a linear recurrent neural network is able to express an injective mapping over sequences, which makes our architecture more expressive than 1-WL.
- Without the need for positional encoding, our model has achieved strong empirical performance on multiple widely used graph benchmarks, which is comparable to or better than that of state-of-the-art graph transformers, with higher training efficiency.

2. Related Work

We review below the literature on expanding MPNN's receptive field, including multi-hop MPNNs and graph transformers, as well as current trends in recurrent models for long-range reasoning on sequential data.

Multi-hop MPNNs. Multi-hop MPNNs leverage the information of multiple hops for each layer. Among existing works, MixHop (Abu-El-Haija et al., 2019) uses powers of the normalized adjacency matrix to access k-hop nodes. k-hop GNN (Nikolentzos et al., 2020) iteratively applies MLPs to combine two consecutive hops and propagates information towards the target node. Feng et al. (2022) theoretically analyze the expressive power of general k-hop MPNNs and enhance it with subgraph information. These works proved that higher-hop information can improve the expressiveness of MPNNs, but they didn't address how to preserve long-range information during propagation as we do. SPN (Abboud et al., 2022) is shown to alleviate oversquashing empirically. It first aggregates neighbors of the same hop but simply uses weighted summation to combine hop representations, which cannot guarantee the expressiveness of the model. On the contrary, we prove that our model, capable of modeling long-range dependency, is also theoretically expressive. PathNN (Michel et al., 2023) encodes each individual path that emanates from a node and aggregates these paths to compute the node representation. DRew (Gutteridge et al., 2023) gradually aggregates more hops at each layer and allows skip connections between different nodes.



$$\textcircled{1}^{+} = \text{MLP}\left(W_{in}\,[\textcircled{1}] + \Lambda W_{in}\,[\textcircled{2},\textcircled{3},\textcircled{4},\textcircled{5}] + \Lambda^2 W_{in}\,[\textcircled{6},\textcircled{7},\textcircled{8},\textcircled{9}]\right)$$

$$\textcircled{5}^{+} = \text{MLP}\left(W_{in}\,[\textcircled{5}] + \Lambda W_{in}\,[\textcircled{1},\textcircled{9}] + \Lambda^2 W_{in}\,[\textcircled{2},\textcircled{3},\textcircled{4},\textcircled{8}] + \Lambda^3 W_{in}\,[\textcircled{6},\textcircled{7}]\right)$$

Figure 2. (a) Sketch of the architecture. MLPs and Layer Normalization operate independently at each node or aggregated multiset. Information of the distant nodes is propagated to the target node through a linear RNN – specifically an LRU (Orvieto et al., 2023b). (b) Depiction of the GRED layer operation for two different target nodes. The gray rectangular boxes indicate the application of multiset aggregation. Finally, the new representation for the target node is computed from the RNN output through an MLP.

Graph transformers. Graph transformers (Ying et al., 2021; Wu et al., 2021; Chen et al., 2022; Rampášek et al., 2022; Zhang et al., 2023; Ma et al., 2023) have recently attracted a lot of interest because the global attention mechanism allows each node to directly attend to all other nodes. To bake in the graph structural information, graph transformers typically use positional encoding (Li et al., 2020; Dwivedi et al., 2022a) as extra features. More specifically, Graphormer (Ying et al., 2021) adds learnable biases to the attention matrix for different shortest distances. However, the sequential order of hops is not encoded into the model, and Graphormer still needs node degrees to augment node features. SAT (Chen et al., 2022) and GraphTrans (Wu et al., 2021) stack message passing layers and self-attention layers together to obtain local information before the global attention. Rampášek et al. (2022) empirically compare different configurations of positional encoding, message passing and global attention. Zhang et al. (2023) suggest the use of resistance distance as relative positional encoding. Ma et al. (2023) use learnable positional encoding initialized with random walk probabilities. He et al. (2023) use MPNNs to encode graph patches generated by a graph clustering algorithm and apply MLP-Mixer (Tolstikhin et al., 2021)/ViT (Dosovitskiy et al., 2021) to patch embeddings, but require node/patch positional encoding and selecting the number of patches.

State space models and linear RNNs. Efficient processing of long sequences is one of the paramount challenges in contemporary deep learning. Attention-based transformers (Vaswani et al., 2017) provide a scalable approach to sequential modeling but suffer from *quadratically increasing inference/memory complexity* as the sequence length grows. While many approaches exist to alleviate this issue, like efficient memory management (Dao et al., 2022; Dao, 2024) and architectural modifications (Wang et al., 2020; Kitaev et al., 2020; Child et al., 2019; Beltagy et al., 2020; Wu et al., 2020), the sequence length in modern large language models is usually kept to $2k/4k$ tokens for this reason (e.g. Llama2 (Touvron et al., 2023)). On top of high inference and memory cost, the attention mechanism often does not provide the correct *inductive bias* for long-range reasoning beyond text (Tay et al., 2021). Due to the issues outlined above, the community has witnessed the rise of innovative *recurrent* alternatives to the attention mechanism, named state space models (SSMs). The first SSM, S4, was introduced by Gu et al. (2022a) based on the theory of polynomial signal approximation (Gu et al., 2020; 2023) and significantly surpassed all modern transformer variants on the challenging long-range benchmark (Tay et al., 2021). Since then, a plethora of variants have been proposed (Hasani et al., 2023; Gupta et al., 2022; Smith et al., 2023; Peng et al., 2023). Deep SSMs have reached outstanding results in various domains, including language (Fu et al., 2023), vision (Nguyen et al., 2022) and audio (Goel et al., 2022). At inference time, all SSMs coincide with a stack of linear RNNs, interleaved with position-wise MLPs and normalization layers. The linearity of the RNNs enables fast parallel processing using FFTs (Gu et al., 2022a) or parallel scans (Smith et al., 2023). The connection between SSMs and linear RNNs is reinforced by Linear Recurrent Unit (LRU) (Orvieto et al., 2023b) that matches the performance of deep SSMs. While SSMs rely on the discretization of a structured continuous-time latent dynamical system, LRU is directly designed for a discrete-time system. The main difference between LRU and a standard linear RNN is that LRU operates in the complex domain and its diagonal transition matrix is trained using polar parameterization for stable signal propagation.

3. Architecture

In this section, we present the GRED layer, which is the building unit of our architecture. We start with some preliminary notations and then describe how our layer computes node representations. Finally, we analyze its computational complexity.

Preliminaries. Let $G = (V, E)$ denote an undirected and unweighted graph, where V denotes the set of nodes and E denotes the set of edges. For any two nodes $v, u \in V$, we use $d(v, u)$ to represent the shortest distance between v and u, and we let $d(v, v) = 0$. For each target node v, we categorize all other nodes into different hops according to their shortest distances to v:

$$\mathcal{N}_k(v) = \{u \mid d(v,u) = k\} \quad \text{for} \quad k = 0, 1, \dots, K \tag{1}$$

where K is the diameter of G or a hyper-parameter specified for the task in hand. $\{\mathcal{N}_k(v)\}_{k=1}^{K}$ can be obtained for every node $v \in V$ by running the Floyd–Warshall algorithm (Floyd, 1962; Warshall, 1962) in parallel during data preprocessing and they are saved as masks.

GRED layer. The input to the ℓ-th layer is a set of node representations $\{\{\boldsymbol{h}_v^{(\ell-1)} \in \mathbb{R}^d \mid v \in V\}\}$. To compute the output representation $\boldsymbol{h}_v^{(\ell)}$ of this layer for a generic target node v, the layer first generates a representation for each set of nodes that share the same shortest distance to v (grey dashed boxes in Figure 2):

$$\boldsymbol{x}_{v,k}^{(\ell)} = \text{AGG}\left(\{\{\boldsymbol{h}_u^{(\ell-1)} \mid u \in \mathcal{N}_k(v)\}\}\right) \tag{2}$$

where $\{\{\cdot\}\}$ denotes a multiset, and AGG is an *injective* multiset function which we parameterize with two wide multilayer perceptrons (MLPs)[1], as usual in the literature (Zaheer et al., 2017; Xu et al., 2019):

$$\boldsymbol{x}_{v,k}^{(\ell)} = \text{MLP}_2\left(\sum\nolimits_{u \in \mathcal{N}_k(v)} \text{MLP}_1\left(\boldsymbol{h}_u^{(\ell-1)}\right)\right) \in \mathbb{R}^d. \tag{3}$$

These set representations $(\boldsymbol{x}_{v,0}^{(\ell)}, \boldsymbol{x}_{v,1}^{(\ell)}, \dots, \boldsymbol{x}_{v,K}^{(\ell)})$ naturally form a sequence according to the shortest distances. Then we encode this sequence using a linear RNN:

$$\boldsymbol{s}_{v,k}^{(\ell)} = \boldsymbol{A}\boldsymbol{s}_{v,k-1}^{(\ell)} + \boldsymbol{B}\boldsymbol{x}_{v,K-k}^{(\ell)} \quad \text{for} \quad k = 0, \dots, K \tag{4}$$

where $\boldsymbol{s}_{v,k}^{(\ell)} \in \mathbb{R}^{d_s}$ represents the hidden state of the RNN and $\boldsymbol{s}_{v,-1}^{(\ell)} = \boldsymbol{0}$. $\boldsymbol{A} \in \mathbb{R}^{d_s \times d_s}$ denotes the state transition matrix and $\boldsymbol{B} \in \mathbb{R}^{d_s \times d}$ is a matrix to transform the input of the RNN. Here in Equation (4) the RNN encoding starts from $\boldsymbol{x}_{v,K}^{(\ell)}$, proceeds from right to left, and ends at $\boldsymbol{x}_{v,0}^{(\ell)}$, which corresponds to the signals from distant nodes propagating towards the target node. The neighborhood hierarchy of the target node v would then be encoded into the final hidden state $\boldsymbol{s}_{v,K}^{(\ell)}$ of the RNN. Note that as in Figure 2(b), different nodes have different sequences to describe their respective neighborhoods, and the RNN computations for all nodes can be batched. Although for a particular target node, some edges between hop k ($k \geq 1$) and hop $k+1$ are omitted by converting its neighborhood into a sequence, those edges would be taken into account for other target nodes. Therefore, considering all node representations as a whole, our model preserves the full graph structural information. We theoretically prove the expressiveness of the linear RNN and our model in Section 4.

[1] In practice, with just one hidden layer.

In our model, we parameterize the linear RNN in a particular diagonal form. Recall that, over the space of $d_s \times d_s$ non-diagonal real matrices, the set of non-diagonalizable (in the complex domain) matrices has measure zero (Bhatia, 2013). Hence, with probability one over random initializations, $\boldsymbol{A}$ is diagonalizable, i.e., $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}$, where $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_{d_s}) \in \mathbb{C}^{d_s \times d_s}$ gathers the eigenvalues of $\boldsymbol{A}$, and columns of $\boldsymbol{V}$ are the corresponding eigenvectors. Equation (4) is then equivalent to the following diagonal recurrence in the complex domain, up to a linear transformation of the hidden state $\boldsymbol{s}$ which can be merged with the output projection $\boldsymbol{W}_{\text{out}}$ (Equation (7)):

$$\boldsymbol{s}_{v,k}^{(\ell)} = \boldsymbol{\Lambda}\boldsymbol{s}_{v,k-1}^{(\ell)} + \boldsymbol{W}_{\text{in}}\boldsymbol{x}_{v,K-k}^{(\ell)} \tag{5}$$

where $\boldsymbol{W}_{\text{in}} = \boldsymbol{V}^{-1}\boldsymbol{B} \in \mathbb{C}^{d_s \times d}$. Unrolling the recurrence, we have:

$$\boldsymbol{s}_{v,K}^{(\ell)} = \sum_{k=0}^{K} \boldsymbol{\Lambda}^k \boldsymbol{W}_{\text{in}}\boldsymbol{x}_{v,k}^{(\ell)}. \tag{6}$$

Equation (6) can be thought of as a *filter over the hops from the target node* (Figure 1), and the filter weights are determined by the magnitudes of the eigenvalues $\boldsymbol{\Lambda}$ and the shortest distances to the target node. Following the modern literature on deep SSMs (Gupta et al., 2022; Gu et al., 2022b), we directly initialize (without loss of generality) the system in the diagonal form and have $\boldsymbol{\Lambda}$ and $\boldsymbol{W}_{\text{in}}$ as trainable parameters[2]. To guarantee stability (the eigenvalues should be bounded by the unit disk), we adopt the recently introduced LRU initialization (Orvieto et al., 2023b) that parameterizes the eigenvalues with log-transformed polar coordinates. Through directly learning eigenvalues $\boldsymbol{\Lambda}$, our model learns to control the influence of signals from distant nodes on the target node, and thus addresses over-squashing caused by iterative 1-hop mixing. Another advantage of the diagonal linear recurrence is that it can leverage parallel scans (Blelloch, 1990; Smith et al., 2023) to avoid computing $\boldsymbol{s}$ sequentially on modern hardware.

The output representation $\boldsymbol{h}_v^{(\ell)}$ is generated by a non-linear transformation of the last hidden state $\boldsymbol{s}_{v,K}^{(\ell)}$:

$$\boldsymbol{h}_v^{(\ell)} = \mathrm{MLP}_3\left(\Re\left[\boldsymbol{W}_{\text{out}}\boldsymbol{s}_{v,K}^{(\ell)}\right]\right) \tag{7}$$

where $\boldsymbol{W}_{\text{out}} \in \mathbb{C}^{d \times d_s}$ is a trainable weight matrix and $\Re[\cdot]$ denotes the real part of a complex-valued vector. While sufficiently wide MLPs with one hidden layer can parameterize any non-linear mapping, following again the literature on state-space models we choose to place here a gated linear unit (GLU) (Dauphin et al., 2017): $\mathrm{GLU}(\boldsymbol{x}) = (\boldsymbol{W}_1\boldsymbol{x}) \odot \sigma(\boldsymbol{W}_2\boldsymbol{x})$, with σ the sigmoid function and $\odot$ the element-wise product.

The final architecture is composed of stacking several of such layers described above. In practice, we merge MLP_1 in Equation (3) with the non-linear transformation in Equation (7) of the previous layer (or of the feature encoder) to make the entire architecture more compact. We add skip connections to both the MLP and the LRU and apply layer normalization to the input of each residual branch. The overall architecture is illustrated in Figure 2(a).

Computational complexity. For each distance k, the complexity of aggregating the representations of nodes from $\mathcal{N}_k(v)$ for every $v \in V$ is that of one round of message passing, which is $O(|E|)$. So the total complexity of Equation (3) for all nodes and distances is $O(K|E|)$. In practice, since $\{\mathcal{N}_k(v)\}_{k=1}^{K}$ are pre-computed, Equation (3) for different k's can be performed in parallel to speed up the computation. The sequential computation of Equation (5) has total complexity $O(K|V|)$. However, the linearity of the recurrence and the diagonal state transition matrix enable fast parallel scans to further improve the efficiency. In the above analysis, K is upper bounded by the graph diameter, which is usually much smaller than the number of nodes in real-world datasets. Even in the worst case where the diameter is large, we can keep the complexity of each layer tractable with a smaller constant K and still access the global information by ensuring the product of model depth and K is no smaller than the diameter. As a result of the compact and parallelizable architectural design, our model is highly efficient during training, as evidenced by our experimental results.

4. Expressiveness Analysis

In this section, we theoretically analyze the expressive capabilities of the linear RNN (Equation (5)) and the overall model. Wide enough linear RNNs have been shown to be able to approximate convolutional filters (Li et al., 2022), and model non-linear dynamic systems when interleaved with MLPs (Orvieto et al., 2023a). In the context of this paper, we are interested in whether the linear RNN can accurately encode the sequence of hop representations (generated by Equation (3)) that describes the neighborhood hierarchy of the target node. To answer this question, in the following, we prove that if the hidden state is large enough, a linear RNN can express an injective mapping over sequences:

Theorem 4.1 (Injectivity of linear RNNs)**.** *Let $\{\boldsymbol{x}_v = (\boldsymbol{x}_{v,0}, \boldsymbol{x}_{v,1}, \boldsymbol{x}_{v,2}, \ldots, \boldsymbol{x}_{v,K_v}) \mid v \in V\}$ be a set of sequences (of different lengths $K_v \leq K$) of vectors with a (possibly uncountable) set of features $\mathcal{X} \subset \mathbb{R}^d$. Consider a diagonal linear complex-valued RNN with d_s-dimensional*

[2] As done in all state-space models (Gu et al., 2022a; Smith et al., 2023), we do not optimize over the complex numbers but instead parameterize, for instance, real and imaginary components of $\boldsymbol{W}_{\text{in}}$ as real parameters. The imaginary unit i is then used to aggregate the two components in the forward pass.

Table 1. Test classification accuracy (in percent) of our model in comparison with baselines. Performance of baselines is reported by the benchmark (Dwivedi et al., 2023) or their original papers. "-" indicates the baseline didn't report its performance on that dataset. We follow the parameter budget $\approx$ 500K.

Model	MNIST	CIFAR10	PATTERN	CLUSTER
GCN (Kipf & Welling, 2017)	90.705±0.218	55.710±0.381	**85.614±0.032**	69.026±1.372
GAT (Veličković et al., 2018)	95.535±0.205	64.223±0.455	78.271±0.186	70.587±0.447
GIN (Xu et al., 2019)	96.485±0.252	55.255±1.527	85.590±0.011	64.716±1.553
GatedGCN (Bresson & Laurent, 2017)	**97.340±0.143**	**67.312±0.311**	85.568±0.088	**73.840±0.326**
EGT (Hussain et al., 2022)	98.173±0.087	68.702±0.409	86.821±0.020	79.232±0.348
SAN (Kreuzer et al., 2021)	-	-	86.581±0.037	76.691±0.65
SAT (Chen et al., 2022)	-	-	86.848±0.037	77.856±0.104
GPS (Rampášek et al., 2022)	98.051±0.126	72.298±0.356	86.685±0.059	78.016±0.180
Graph MLP-Mixer (He et al., 2023)	**98.320±0.040**	73.960±0.330	-	-
GRIT (Ma et al., 2023)	98.108±0.111	**76.468±0.881**	**87.196±0.076**	**80.026±0.277**
GRED (Ours)	**98.383±0.012**	**76.853±0.185**	**86.759±0.020**	**78.495±0.103**

hidden state, parameters $\mathbf{\Lambda} \in diag(\mathbb{C}^{d_s}), \boldsymbol{W}_{in} \in \mathbb{C}^{d_s \times d}$ and recurrence rule $\boldsymbol{s}_{v,k} = \mathbf{\Lambda}\boldsymbol{s}_{v,k-1} + \boldsymbol{W}_{in}\boldsymbol{x}_{v,K_v-k}$, initialized at $\boldsymbol{s}_{v,-1} = \mathbf{0} \in \mathbb{R}^{d_s}$ for each $v \in V$. If $d_s \geq (K+1)d$, then there exist $\mathbf{\Lambda}, \boldsymbol{W}_{in}$ such that the map $R : (\boldsymbol{x}_{v,0}, \boldsymbol{x}_{v,1}, \boldsymbol{x}_{v,2}, \ldots, \boldsymbol{x}_{v,K}) \mapsto \boldsymbol{s}_{v,K}$ (with zero right-padding if $K_v < K$) is bijective. Moreover, if the set of RNN inputs has countable cardinality $|\mathcal{X}| = N \leq \infty$, then selecting $d_s \geq d$ is sufficient for the existence of an injective linear RNN mapping R.

The proof can be found in Appendix A. Here we assume zero-padding for $K_v < K$ (for mini-batch training). If some nodes coincidentally have zero-valued features, we can select a special token which is not in the dictionary of node features as the padding token. In practice, such an operation is not necessary because node representations are first fed into an MLP before the linear RNN, which can learn to shift them away from zero.

Based on Theorem 4.1, and the well-known conclusion that the parameterization given by Equation (3) can express an injective multiset function (Xu et al., 2019), we have the following corollary:

Corollary 4.2. *A wide enough GRED layer is capable of expressing an injective mapping of the list $(\boldsymbol{h}_v, \{\!\{\boldsymbol{h}_u \mid u \in \mathcal{N}_1(v)\}\!\}, \{\!\{\boldsymbol{h}_u \mid u \in \mathcal{N}_2(v)\}\!\}, \ldots, \{\!\{\boldsymbol{h}_u \mid u \in \mathcal{N}_{K_v}(v)\}\!\})$ for each $v \in V$.*

This corollary in turn implies the following result:

Corollary 4.3 (Expressiveness of GRED)**.** *When $K > 1$, one wide enough GRED layer is more expressive than any 1-hop message passing layer.*

Proof. We note that 1-WL assumes an injective mapping of 1-hop neighborhood, i.e., $(\boldsymbol{h}_v, \{\!\{\boldsymbol{h}_u \mid u \in \mathcal{N}_1(v)\}\!\})$, which is a special case of GRED ($K = 1$). When $K > 1$, the output of one GRED layer at node v, given the injectivity of the linear RNN and AGG, provides a more detailed characterization of its neighborhood than 1-hop message passing. This means that if v's 1-hop neighborhood changes, the output of the GRED layer will also be different. Therefore, GRED is able to distinguish any two non-isomorphic graphs that are distinguishable by 1-WL. Moreover, GRED can distinguish two non-isomorphic graphs which 1-WL cannot (see Figure 7 in the appendix for an example). $\square$

We note that Feng et al. (2022) have already proven that multi-hop MPNNs are more expressive than 1-WL, but are upper bounded by 3-WL, which also applies to our model. Different from them, we achieve such expressiveness with a compact and parameter-efficient architecture (i.e., the number of parameters does not increase with K), which is of independent interest and bridges the gap between theory and practice.

5. Experiments

In this section, we evaluate our model on widely used graph benchmarks (Dwivedi et al., 2023; 2022b). In all experiments, we train our model using the Adam optimizer with weight decay (Loshchilov & Hutter, 2019) and use the cosine annealing schedule with linear warm-up for the first 5% epochs. We compare our model against popular MPNNs including GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), GIN (Xu et al., 2019), GatedGCN (Bresson & Laurent, 2017), and multi-hop MPNN variants (Feng et al., 2022; Michel et al., 2023; Gutteridge et al., 2023), as well as several state-of-the-art graph transformers including Graphormer (Ying et al., 2021), SAT (Chen et al., 2022), GPS (Rampášek et al., 2022), Graph MLP-Mixer (He et al.,

Table 2. Test MAE on ZINC 12K with parameter budget ≈ 500K.

Model	Test MAE ↓
GCN (Kipf & Welling, 2017)	0.278±0.003
GAT (Veličković et al., 2018)	0.384±0.007
GIN (Xu et al., 2019)	0.387±0.015
GatedGCN (Bresson & Laurent, 2017)	0.282±0.015
PNA (Corso et al., 2020)	0.188±0.004
KP-GIN (Feng et al., 2022)	0.093±0.007
PathNN (Michel et al., 2023)	**0.090±0.004**
SAN (Kreuzer et al., 2021)	0.139±0.006
Graphormer (Ying et al., 2021)	0.122±0.006
K-subgraph SAT (Chen et al., 2022)	0.094±0.008
GPS (Rampášek et al., 2022)	0.070±0.004
Graph MLP-Mixer (He et al., 2023)	0.073±0.001
GRIT (Ma et al., 2023)	**0.059±0.002**
GRED (Ours)	**0.077±0.002**

Table 3. Test performance on Peptides-func/struct.

Model	Peptides-func Test AP ↑	Peptides-struct Test MAE ↓
GCN*	0.6860±0.0050	**0.2460±0.0007**
GINE*	0.6621±0.0067	0.2473±0.0017
GatedGCN*	0.6765±0.0047	0.2477±0.0009
PathNN	0.6816±0.0026	0.2540±0.0046
DRew	0.6996±0.0076	0.2781±0.0028
DRew+LapPE	**0.7150±0.0044**	0.2536±0.0015
SAN+LapPE	0.6384±0.0121	0.2683±0.0043
GPS	0.6535±0.0041	0.2500±0.0005
Graph-MLPMixer	0.6970±0.0080	0.2475±0.0015
GRIT	**0.6988±0.0082**	**0.2460±0.0012**
GRED (Ours)	**0.7085±0.0027**	**0.2503±0.0019**
GRED+LapPE	**0.7133±0.0011**	**0.2455±0.0013**

2023) and GRIT (Ma et al., 2023). We also measure the training time and memory consumption of GRED to demonstrate its high efficiency. We use three distinct colors to indicate the performance of our model, the best MPNN, and the best graph transformer. We detail the hyper-parameters used for our model in the appendix (Table 5). In Appendix B, we validate GRED's robustness to over-squashing and compare GRED with SPN (Abboud et al., 2022).

Benchmarking GNNs. We first evaluate our model on the node classification datasets: PATTERN and CLUSTER, and graph classification datasets: MNIST and CIFAR10 from (Dwivedi et al., 2023). To get the representation for the entire graph, we simply do average pooling over all node representations. Our model doesn't use any positional encoding. We train our model four times with different random seeds and report the average accuracy with standard deviation. The results are shown in Table 1. From the table we see that graph transformers generally perform better than MPNNs. Among the four datasets, PATTERN models communities in social networks and all nodes are reachable within 3 hops, which we conjecture is why the performance gap between graph transformers and MPNNs is only marginal. For a more difficult task, like CIFAR10, that requires information from a relatively larger neighborhood, graph transformers work more effectively. GRED performs well on all four datasets and consistently outperforms MPNNs. Notably, on MNIST and CIFAR10, GRED achieves the best accuracy, outperforming state-of-the-art models Graph MLP-Mixer and GRIT, which validates that our model can effectively aggregate information beyond the local neighborhood.

ZINC 12K. Next, we report the test MAE of our model on ZINC 12K (Dwivedi et al., 2023). The average MAE and standard deviation of four runs with different random seeds are shown in Table 2 along with baseline performance from their original papers. From Table 2 we can observe that the performance of our model is significantly better than that of existing MPNNs. In particular, GRED outperforms other multi-hop MPNN variants (Feng et al., 2022; Michel et al., 2023), which shows our architecture is more effective in aggregating multi-hop information. Comparing GRED with graph transformers, we find that it outperforms several graph transformer variants (SAN, Graphormer, and K-subgraph SAT) and approaches the state-of-the-art model. This is impressive given that our model doesn't require any positional encoding. These results evidence that our model can encode graph structural information through the natural inductive bias of recurrence.

Long Range Graph Benchmark. To further test the long-range modeling capability of GRED, we evaluate it on the Peptides-func and Peptides-struct datasets from (Dwivedi et al., 2022b). We follow the 500K parameter budget and train our model four times with different random seeds. The results are displayed in Table 3. The performance of GCN, GINE and GatedGCN (marked with $*$) comes from a recent report (Tönshoff et al., 2023) that extensively tuned their hyper-parameters *with* positional encoding. Performance of other baselines is reported by respective papers. We can observe that, even without positional encoding, GRED significantly outperforms all baselines except DRew+LapPE on Peptides-func, and its performance on Peptides-struct also matches that of the best graph transformer. Note that on Peptides-struct, DRew+LapPE performs worse than GRED. These results demonstrate the strong long-range modeling capability of our architecture itself. As a supplement, we test GRED+LapPE by concatenating Laplacian positional encoding with node features, and we find it slightly im-

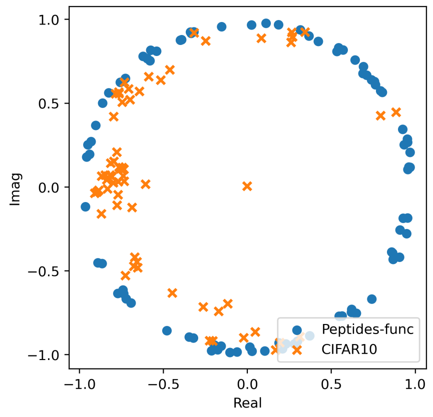


Figure 3. Learned (complex) eigenvalues of the first GRED layer on CIFAR10 and Peptides-func.

proves the performance. We leave the combination of more advanced positional encoding with GRED to future work.

To illustrate how GRED can learn to preserve long-range information, we examine the eigenvalues learned by the linear RNN (i.e., $\mathbf{\Lambda}$ in Equation (5)) after training, as shown in Figure 3. We observe from the figure that the eigenvalues are pushed close to 1 for the long-range task Peptides-func, which prevent the signals of distant nodes from decaying too fast. Compared with Peptides-func, CIFAR10 requires the model to utilize more information from the local neighborhood, so the magnitudes of the eigenvalues become smaller.

Table 4. Average training time per epoch and GPU memory consumption for GRIT and GRED.

Model	ZINC 12K	CIFAR10	Peptides-func
GRIT	23.9s 1.9GB	244.4s 4.6GB	225.6s 22.5GB
GRED	3.7s 1.5GB	27.8s 1.4GB	158.9s 18.5GB
Speedup	**6.5**×	**8.8**×	**1.4**×

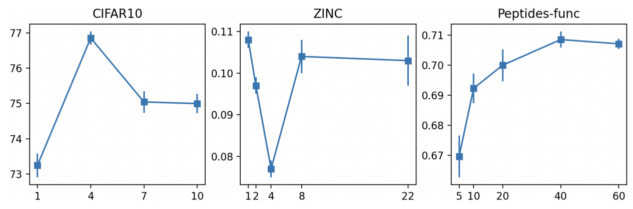


Figure 4. Effect of K on performance.

Training efficiency. To demonstrate the high efficiency of our model, we record the average training time per epoch and GPU memory consumption on ZINC, CIFAR10 and Peptides-func. We compare our measurements with those of the state-of-the-art graph transformer GRIT. Both models are trained using the same batch size and on a single RTX A5000 GPU with 24GB memory. As shown in Table 4, our model improves the training efficiency by a huge margin, which stems from our compact and parallelizable architecture design.

Effect of K on performance. Recall that the length of recurrence K can be regarded as a hyper-parameter in GRED. In Figure 4, we show how different K values affect the performance of GRED on CIFAR10, ZINC and Peptides-func, keeping the depth and hidden dimension of the architecture unchanged (without positional encoding). On CIFAR10 and ZINC, while directly setting K as the diameter already outperforms classical MPNNs, we find that the optimal K value that yields the best performance lies strictly between 1 and the diameter. This may be because information that is too far away is less important for these two tasks (interestingly, the best K value for CIFAR10 is similar to the width of a convolutional kernel on a normal image). On Peptides-func, the performance is more monotonic with K. When $K = 40$, GRED outperforms the best graph transformer GRIT. We observe no further performance gain on Peptides-func when we increase K to 60.

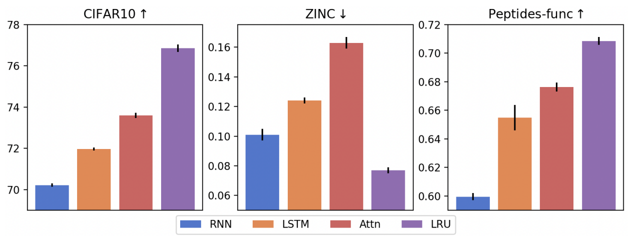


Figure 5. Performance of GRED using RNNs of different flavors.

Comparing RNNs of different flavors. Finally, we highlight the necessity of the LRU component (Equation (5)) of GRED by replacing it with a vanilla RNN, a standard LSTM cell or 8-head self-attention. The performance of different variants is shown in Figure 5. We use the same number of layers and K for all models and tune the learning rate, weight decay and dropout rate in the same grid. None of the variants use positional encoding. We can observe that GRED$_{\text{LSTM}}$ performs better than GRED$_{\text{RNN}}$ on CIFAR10 and Peptides-func. Since LSTM can alleviate the training instability of the vanilla RNN, the improvement of GRED$_{\text{LSTM}}$ over GRED$_{\text{RNN}}$ is particularly large on the long-range dataset Peptides-func. GRED$_{\text{Attn}}$ allows direct interaction with each hop and thus also yields good performance on Peptides-func. However, self-attention cannot provide good inductive bias because it cannot model the order of the hop sequence, which can explain why the performance of $\text{GRED}_{\text{Attn}}$ is the worst on ZINC. GRED_{LRU} consistently outperforms the other variants, attributed to its advanced parameterization for stable signal propagation and great expressive power.

6. Conclusion

In this paper, we introduce the Graph Recurrent Encoding by Distance (GRED) model for graph representation learning. By integrating permutation-invariant neural networks with linear recurrent neural networks, GRED effectively harnesses information from distant nodes without the need for positional encoding or computationally expensive attention mechanisms. Theoretical and empirical evaluations confirm GRED's superior performance compared with existing MPNNs and highly competitive results compared with state-of-the-art graph transformers at a higher training efficiency. This positions GRED as a powerful, efficient, and promising model for graph representation learning.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback, which helped us improve the paper. Antonio Orvieto acknowledges the financial support of the Hector Foundation. Yuhui Ding would like to personally thank Jiaxin Zhang for her support during the stressful time before the deadline.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Abboud, R., Dimitrov, R., and Ceylan, I. I. Shortest path networks for graph property prediction. In *Learning on Graphs Conference*, 2022.

Abu-El-Haija, S., Perozzi, B., Kapoor, A., Alipourfard, N., Lerman, K., Harutyunyan, H., Ver Steeg, G., and Galstyan, A. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *ICML*, 2019.

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. In *ICLR*, 2021.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Bhatia, R. *Matrix analysis*. Springer Science & Business Media, 2013.

Blelloch, G. E. Prefix sums and their applications, 1990.

Bresson, X. and Laurent, T. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Chen, D., O'Bray, L., and Borgwardt, K. Structure-aware transformer for graph representation learning. In *ICML*, 2022.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. In *NeurIPS*, 2020.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. In *ICLR*, 2024.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *NeurIPS*, 2022.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. Language modeling with gated convolutional networks. In *ICML*, 2017.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

Di Giovanni, F., Giusti, L., Barbero, F., Luise, G., Lio, P., and Bronstein, M. M. On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *ICML*, 2023.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Graph neural networks with learnable structural and positional representations. In *ICLR*, 2022a.

Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. Long range graph benchmark. In *NeurIPS*, 2022b.

Dwivedi, V. P., Joshi, C. K., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *JMLR*, 2023.

Feng, J., Chen, Y., Li, F., Sarkar, A., and Zhang, M. How powerful are k-hop message passing graph neural networks. In *NeurIPS*, 2022.

Floyd, R. W. Algorithm 97: shortest path. *Communications of the ACM*, 1962.

Fu, D. Y., Dao, T., Saab, K. K., Thomas, A. W., Rudra, A., and Re, C. Hungry hungry hippos: Towards language modeling with state space models. In *ICLR*, 2023.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *ICML*, 2017.

Goel, K., Gu, A., Donahue, C., and Ré, C. It's raw! audio generation with state-space models. In *ICML*, 2022.

Gu, A., Dao, T., Ermon, S., Rudra, A., and Ré, C. Hippo: Recurrent memory with optimal polynomial projections. In *NeurIPS*, 2020.

Gu, A., Goel, K., and Re, C. Efficiently modeling long sequences with structured state spaces. In *ICLR*, 2022a.

Gu, A., Gupta, A., Goel, K., and Ré, C. On the parameterization and initialization of diagonal state space models. In *NeurIPS*, 2022b.

Gu, A., Johnson, I., Timalsina, A., Rudra, A., and Ré, C. How to train your hippo: State space models with generalized orthogonal basis projections. In *ICLR*, 2023.

Gupta, A., Gu, A., and Berant, J. Diagonal state spaces are as effective as structured state spaces. In *NeurIPS*, 2022.

Gutteridge, B., Dong, X., Bronstein, M. M., and Di Giovanni, F. Drew: Dynamically rewired message passing with delay. In *ICML*, 2023.

Hasani, R., Lechner, M., Wang, T.-H., Chahine, M., Amini, A., and Rus, D. Liquid structural state-space models. In *ICLR*, 2023.

He, X., Hooi, B., Laurent, T., Perold, A., LeCun, Y., and Bresson, X. A generalization of vit/mlp-mixer to graphs. In *ICML*, 2023.

Hussain, M. S., Zaki, M. J., and Subramanian, D. Global self-attention as a replacement for graph convolution. In *SIGKDD*, 2022.

Kim, J., Nguyen, D., Min, S., Cho, S., Lee, M., Lee, H., and Hong, S. Pure transformers are powerful graph learners. In *NeurIPS*, 2022.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. In *ICLR*, 2020.

Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. Rethinking graph transformers with spectral attention. In *NeurIPS*, 2021.

Li, P., Wang, Y., Wang, H., and Leskovec, J. Distance encoding: Design provably more powerful neural networks for graph representation learning. In *NeurIPS*, 2020.

Li, Z., Han, J., E, W., and Li, Q. Approximation and optimization theory for linear continuous-time recurrent neural networks. *JMLR*, 2022.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *ICLR*, 2019.

Ma, L., Lin, C., Lim, D., Romero-Soriano, A., Dokania, P. K., Coates, M., Torr, P., and Lim, S.-N. Graph inductive biases in transformers without message passing. In *ICML*, 2023.

Michel, G., Nikolentzos, G., Lutzeyer, J. F., and Vazirgiannis, M. Path neural networks: Expressive and accurate graph neural networks. In *ICML*, 2023.

Nguyen, E., Goel, K., Gu, A., Downs, G. W., Shah, P., Dao, T., Baccus, S. A., and Ré, C. S4nd: Modeling images and videos as multidimensional signals using state spaces. In *NeurIPS*, 2022.

Nikolentzos, G., Dasoulas, G., and Vazirgiannis, M. k-hop graph neural networks. *Neural Networks*, 2020.

Orvieto, A., De, S., Gulcehre, C., Pascanu, R., and Smith, S. L. On the universality of linear recurrences followed by nonlinear projections. *arXiv preprint arXiv:2307.11888*, 2023a.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., and De, S. Resurrecting recurrent neural networks for long sequences. In *ICML*, 2023b.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Cao, H., Cheng, X., Chung, M., Grella, M., GV, K. K., et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. In *NeurIPS*, 2022.

Smith, J. T., Warrington, A., and Linderman, S. W. Simplified state space layers for sequence modeling. In *ICLR*, 2023.

Tang, J., Sun, J., Wang, C., and Yang, Z. Social influence analysis in large-scale networks. In *SIGKDD*, 2009.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long range arena: A benchmark for efficient transformers. In *ICLR*, 2021.

Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., et al. Mlp-mixer: An all-mlp architecture for vision. In *NeurIPS*, 2021.

Tönshoff, J., Ritzert, M., Rosenbluth, E., and Grohe, M. Where did the gap go? reassessing the long-range graph benchmark. *arXiv preprint arXiv:2309.00367*, 2023.

Topping, J., Di Giovanni, F., Chamberlain, B. P., Dong, X., and Bronstein, M. M. Understanding over-squashing and bottlenecks on graphs via curvature. In *ICLR*, 2022.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *NeurIPS*, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. In *ICLR*, 2018.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Warshall, S. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 1962.

Wu, Z., Liu, Z., Lin, J., Lin, Y., and Han, S. Lite transformer with long-short range attention. In *ICLR*, 2020.

Wu, Z., Jain, P., Wright, M., Mirhoseini, A., Gonzalez, J. E., and Stoica, I. Representing long-range context for graph neural networks with global attention. In *NeurIPS*, 2021.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *ICLR*, 2019.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform badly for graph representation? In *NeurIPS*, 2021.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*, 2018.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. In *NeurIPS*, 2017.

Zhang, B., Luo, S., Wang, L., and He, D. Rethinking the expressive power of gnns via graph biconnectivity. In *ICLR*, 2023.

A. Proof of Theorem 4.1

Proof. For now, let us assume for ease of exposition that all sequences are of length K. Also, let us, for simplicity, omit the dependency on $v \in V$ and talk about generic sequences.

The proof simply relies on the idea of writing the linear recurrence in matrix form (Gu et al., 2022b; Orvieto et al., 2023a). Note that for a generic input $\boldsymbol{x} = (\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_K) \in \mathbb{R}^{d\times(K+1)}$, the recurrence output can be rewritten in terms of powers of $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \lambda_2, \dots, \lambda_{d_s})$ as follows:

$$\boldsymbol{s}_K = \sum_{k=0}^{K} \boldsymbol{\Lambda}^k \boldsymbol{W}_{\text{in}} \boldsymbol{x}_k. \tag{8}$$

We now present sufficient conditions for the map $R : (\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_K) \mapsto \boldsymbol{s}_K$ to be injective or bijective. The proof for bijectivity does not require the set of node features to be in a countable set, and it is simpler.

Bijective mapping. First, let us design a proper matrix $\boldsymbol{W}_{\text{in}} \in \mathbb{R}^{d_s \times d}$. We choose $d_s = (K+1)d$ and set $\boldsymbol{W}_{\text{in}} = \boldsymbol{I}_{d\times d} \otimes \boldsymbol{1}_{(K+1)\times 1}$. As a result, the RNN will independently process each dimension of the input with a sub-RNN of size $(K+1)$. The resulting $\boldsymbol{s}_K \in \mathbb{R}^{(K+1)d}$ will gather each sub-RNN output by concatenation. We can then restrict our attention to the first dimension of the input sequence:

$$(\boldsymbol{s}_K)_{1:(K+1)} = \sum_{k=0}^{K} \operatorname{diag}(\lambda_1, \lambda_2, \dots, \lambda_{K+1})^k \boldsymbol{1}_{(K+1)\times 1} x_{k,1}. \tag{9}$$

This sum can be written conveniently by multiplication using a Vandermonde matrix:

$$(\boldsymbol{s}_K)_{1:(K+1)} = \begin{pmatrix} \lambda_1^K & \lambda_1^{K-1} & \cdots & \lambda_1 & 1 \\ \lambda_2^K & \lambda_2^{K-1} & \cdots & \lambda_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \lambda_{K+1}^K & \lambda_{K+1}^{K-1} & \cdots & \lambda_{K+1} & 1 \end{pmatrix} \boldsymbol{x}^{\leftarrow}_{0:K,1}. \tag{10}$$

where $\boldsymbol{x}^{\leftarrow}_{0:K,1}$ is the input sequence (first dimension) in reverse order. The proof is concluded by noting that Vandermonde matrices of size $(K+1)\times(K+1)$ are full-rank since they have non-zero determinant $\prod_{1\le i<j\le(K+1)}(\lambda_i - \lambda_j) \neq 0$, under the assumption that all λ_i are distinct. Note that one does not need complex eigenvalues to achieve this, both $\boldsymbol{\Lambda}$ and $\boldsymbol{W}_{\text{in}}$ can be real. However, as discussed by Orvieto et al. (2023a), complex eigenvalues improve conditioning of the Vandermonde matrix.

Injective mapping. The condition for injectivity is that if $\boldsymbol{x} \neq \hat{\boldsymbol{x}}$, then $R(\boldsymbol{x}) \neq R(\hat{\boldsymbol{x}})$. In formulas,

$$\boldsymbol{s}_K - \hat{\boldsymbol{s}}_K = \sum_{k=0}^{K} \boldsymbol{\Lambda}^k \boldsymbol{W}_{\text{in}} (\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k) \neq \boldsymbol{0} \tag{11}$$

Let us assume the state dimension coincides with the input dimension, and let us set $\boldsymbol{W}_{\text{in}} = \boldsymbol{I}_{d\times d}$. Then, we have the condition:

$$\boldsymbol{s}_K - \hat{\boldsymbol{s}}_K = \sum_{k=0}^{K} \boldsymbol{\Lambda}^k (\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k) \neq \boldsymbol{0}. \tag{12}$$

Since $\boldsymbol{\Lambda} = \operatorname{diag}(\lambda_1, \lambda_2, \dots, \lambda_d)$ is diagonal, we can study each component of $\boldsymbol{s}_K - \hat{\boldsymbol{s}}_K$ separately. We therefore require

$$s_{K,i} - \hat{s}_{K,i} = \sum_{k=0}^{K} \lambda_i^k (x_{k,i} - \hat{x}_{k,i}) \neq 0 \qquad \forall i \in \{1, 2, \dots, d\}. \tag{13}$$

We can then restrict our attention to linear one-dimensional RNNs (i.e. filters) with one-dimensional input $\boldsymbol{x} \in \mathbb{R}^{1\times(K+1)}$. We would like to choose $\lambda \in \mathbb{C}$ such that

$$\sum_{k=0}^{K} \lambda^k (x_k - \hat{x}_k) \neq 0 \tag{14}$$

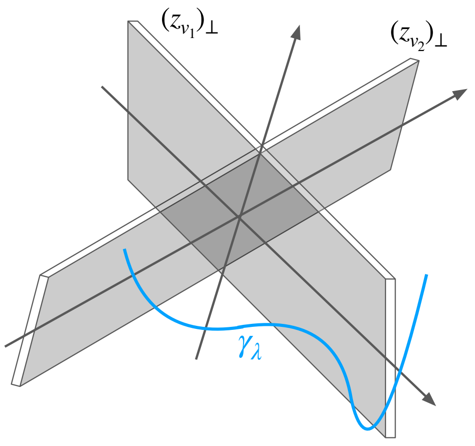


Figure 6. Proof illustration for Theorem 4.1. The set $\mathcal{Z}_\perp$ is depicted as union of hyperplanes, living in $\mathbb{R}^{K+1}$ and here sketched in three dimensions. The curve $\gamma_\lambda : \lambda \mapsto (1, \lambda, \lambda^2, \cdots, \lambda^K)$ is shown as a blue line. The proof shows that, for $\lambda \in \mathbb{R}$, the support of γ_λ is not entirely contained in $\mathcal{Z}_\perp$.

Under the assumption $|\mathcal{X}| = N \leq \infty$, $\boldsymbol{x} - \bar{\boldsymbol{x}}$ is a generic signal in a countable set ($N(N-1)/2 = \Omega(N^2)$ possible choices). Let us rename $\boldsymbol{z} := \boldsymbol{x} - \bar{\boldsymbol{x}} \in \mathcal{Z} \subset \mathbb{R}^{1\times(K+1)}$, $|\mathcal{Z}| = \Omega(N^2)$. We need

$$\langle \bar{\boldsymbol{\lambda}}, \boldsymbol{z} \rangle \neq 0, \qquad \forall \boldsymbol{z} \in \mathcal{Z}, \qquad \text{where} \quad \bar{\boldsymbol{\lambda}} = (1, \lambda, \lambda^2, \cdots, \lambda^K) \tag{15}$$

Such λ can always be found *in the real numbers*, and the reason is purely geometric. We need

$$\bar{\boldsymbol{\lambda}} \notin \mathcal{Z}_\perp := \bigcup_{\boldsymbol{z}\in\mathcal{Z}} \boldsymbol{z}_\perp.$$

Note that $\dim(\boldsymbol{z}_\perp) = K$, so $\dim(\mathcal{Z}_\perp) = K$ due to the countability assumption — in other words the Lebesgue measure vanishes: $\mu(\mathcal{Z}_\perp; \mathbb{R}^{K+1}) = 0$. If $\bar{\boldsymbol{\lambda}}$ were an arbitrary vector, we would be done since we can pick it at random and with probability one $\bar{\boldsymbol{\lambda}} \notin \mathcal{Z}_\perp$. But $\bar{\boldsymbol{\lambda}}$ is structured (lives on a 1-dimensional manifold), so we need one additional step.

Note that $\bar{\boldsymbol{\lambda}}$ is parametrized by λ, and in particular $\mathbb{R} \ni \lambda \mapsto \bar{\boldsymbol{\lambda}} \in \mathbb{R}^{K+1}$ is a curve in $\mathbb{R}^{K+1}$, we denote this as γ_λ. Now, crucially, note that the support of γ_λ is a smooth curved manifold for $K > 1$. In addition, crucially, $\mathbf{0} \notin \gamma_\lambda$. We are done: it is impossible for the γ_λ curve to live in a K dimensional space composed of a union of hyperplanes; it indeed has to span the whole $\mathbb{R}^{K+1}$, without touching the zero vector (see Figure 6). The reason why it spans the whole $\mathbb{R}^{K+1}$ comes from the Vandermonde determinant! Let $\{\lambda_1, \lambda_2, \cdots, \lambda_{K+1}\}$ be a set of $K+1$ distinct λ values. The Vandermonde matrix

$$\begin{pmatrix} \lambda_1^K & \lambda_1^{K-1} & \cdots & \lambda_1 & 1 \\ \lambda_2^K & \lambda_2^{K-1} & \cdots & \lambda_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \lambda_{K+1}^K & \lambda_{K+1}^{K-1} & \cdots & \lambda_{K+1} & 1 \end{pmatrix}$$

has determinant $\prod_{1\leq i<j\leq(K+1)}(\lambda_i - \lambda_j) \neq 0$ — it's full rank, meaning that the vectors $\bar{\boldsymbol{\lambda}}_1, \bar{\boldsymbol{\lambda}}_2, \ldots, \bar{\boldsymbol{\lambda}}_{K+1}$ span the whole $\mathbb{R}^{K+1}$. Note that $\lambda \mapsto \bar{\boldsymbol{\lambda}}$ is a continuous function, so even though certain $\bar{\boldsymbol{\lambda}}_i$ might live on $\mathcal{Z}_\perp$ there exists a value in between them which is not contained in $\mathcal{Z}_\perp$. $\square$

B. Additional Results

To validate the robustness of GRED to over-squashing, we consider the Tree-NeighborsMatch task proposed by Alon & Yahav (2021). Following the same experimental setup as Alon & Yahav (2021), we report the training accuracy of GRED in Table 6 to show how well GRED can harness long-range information to fit the data. As a comparison, we quote

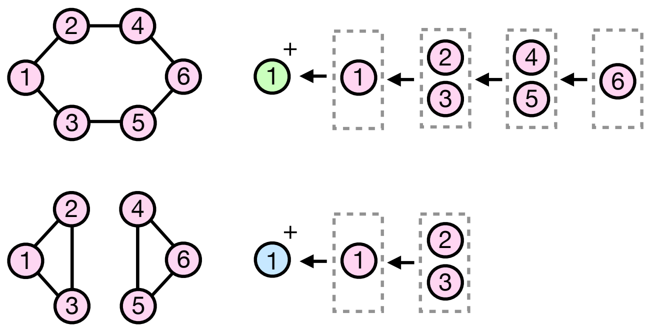


Figure 7. GRED provides distinct updates for the two graphs above. Such graphs, however, are indistinguishable by the 1-WL isomorphism test, assuming (worst-case) nodes have identical features.

Table 5. Hyper-parameters for GRED. For PATTERN and CLUSTER, K is the diameter of the graph. For GRED+LapPE in Table 3, the Laplacian PE uses the 10 smallest eigenvectors and a hidden dimension of 16.

Hyper-parameter	ZINC 12K	MNIST	CIFAR10	PATTERN	CLUSTER	Peptides-func	Peptides-struct
Layers	11	4	8	10	16	8	4
K	4	2	4	-	-	40	4
Dropout	0.2	0.15	0.15	0.2	0.2	0.2	0.2
d	72	128	96	64	64	88	128
d_s	72	96	64	64	64	88	96
Learning rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Weight decay	0.1	0.1	0.1	0.1	0.2	0.2	0.2
Epochs	2000	600	600	100	100	200	200
Batch size	32	16	16	32	32	32	32

the performance of GIN which uses the same multiset aggregation as GRED. For GIN, a network with $r+1$ layers is trained for each tree depth in the original paper (Alon & Yahav, 2021), while for GRED the number of layers is only around half of the tree depth, with an appropriate $K > 1$ to avoid under-reaching. Over-squashing starts to affect GIN at $r = 4$, preventing the model from effectively using distant information to perfectly fit the data. On the contrary, GRED is not affected by over-squashing across different tree depths.

We further evaluate GRED on NCI1 and PROTEINS from TUDataset. We follow the experimental setup of SPN (Abboud et al., 2022), and report the average test accuracy and standard deviation across 10 train/val/test splits, as shown in Table 7. We use the same K for GRED as for SPN and cite the performance reported by the SPN paper (Abboud et al., 2022). Our model generalizes well to TUDataset and shows good performance. Furthermore, GRED outperforms SPN (Abboud et al., 2022) with the same number of hops, which verifies that GRED is a better architecture for aggregating large neighborhoods.

Table 6. Accuracy across tree depths.

Model	$r=2$	3	4	5	6	7	8
GIN	1.0	1.0	0.77	0.29	0.20	-	-
GRED	1.0	1.0	1.0	1.0	1.0	1.0	0.95

Table 7. Performance (accuracy) of GRED on TUDataset.

Model	NCI1	PROTEINS
DGCNN	76.4±1.7	72.9±3.5
DiffPool	76.9±1.9	73.7±3.5
ECC	76.2±1.4	72.3±3.4
GIN	80.0±1.4	73.3±4.0
GraphSAGE	76.0±1.8	73.0±4.5
SPN ($K=10$)	78.2±1.2	74.5±3.2
GRED ($K=10$)	**82.6±1.4**	**75.0±2.9**