GATED RELATIONAL GRAPH ATTENTION NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Relational Graph Neural Networks (GNN), like all GNNs, suffer from a drop in performance when training deeper networks, which may be caused by vanishing gradients, over-parameterization, and over-smoothing. Previous works have investigated methods that improve the training of deeper GNNs, which include normalization techniques and various types of skip connection within a node. However, learning long-range patterns in multi-relational graphs using GNNs remains an under-explored topic. In this work, we propose a novel relation-aware GNN architecture based on the Graph Attention Network that uses gated skip connections to improve long-range modeling between nodes and uses a more scalable vector-based approach for parameterizing relations. We perform extensive experimental analysis on synthetic and real data, focusing explicitly on learning long-range patterns. The proposed method significantly outperforms several commonly used GNN variants when used in deeper configurations and stays competitive to existing architectures in a shallow setup.

1 INTRODUCTION

In this work, we focus on learning long-range patterns in multi-relational graphs using graph neural networks (GNN), which heavily relies on the ability to train deep networks[1]. However, GNNs suffer from decreasing performance when the number of layers is increased. Zhao & Akoglu (2020) point out that this may be due to (1) over-fitting, (2) vanishing gradients, and (3) over-smoothing (the phenomenon where node representations become less distinguishable from each other when more layers are used). Recently, several works have investigated over-fitting (Vashishth et al., 2020), over-smoothing (Li et al., 2018; Chen et al., 2019; Zhao & Akoglu, 2020; Rong et al., 2019; Yang et al., 2020), over-squashing (Alon & Yahav, 2020), and possible vanishing gradient (Hochreiter & Schmidhuber, 1997; Pascanu et al., 2013; He et al., 2016) problems in GNNs (Li et al., 2019a; 2020; Rahimi et al., 2018).

One simple but effective technique to improve the training of deeper GNNs is using skip connections, for example, implemented by the Gated Recurrent Unit (GRU) (Cho et al.) in the Gated GNN (GGNN) (Li et al., 2015). Such connections can improve learning deep GNNs as it avoids vanishing gradients towards lower-layer representations of the same node, and reduces over-smoothing (Hamilton, 2020). However, such vertical skip connections are not sufficient to enable learning long-range patterns. In addition, in relational GNNs, such as the Relational Graph Convolutional Network (RGCN) (Schlichtkrull et al., 2018), training difficulties may arise from the methods used for integrating relation information, which may suffer from over-parameterization and impact backpropagation.

We develop a new GNN architecture for multi-relational graphs that reduces the vanishing gradient and over-parameterization problems that occur with existing methods and improves generalization when learning long-range patterns using deeper networks. We propose several changes to the Graph Attention Network (GAT) (Veličković et al., 2018), including a modified attention mechanism, an alternative GRU-based update function, and a gated relation-aware message function. We perform an extensive experimental study of the proposed method: (1) we show that different existing relation-aware GNNs fail to learn simple patterns in a simple synthetic sequence-based graph classification task, (2) we perform a comparison and ablation study on a synthetic node classification task, (3) we show that our architecture is competitive with existing ones on an entity classification task using real-world data from previous work.

[1] We need at least K GNN layers to capture information that is K hops away.

2 BACKGROUND: GNNS AS MESSAGE PASSING NETWORKS

Many popular GNNs can be formulated within a message passing (MP) framework (Gilmer et al., 2017). Consider a multi-relational graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{E}$ consists of edges e which are specified by a triple (u, v, r) that define a directed edge from u to v, labeled by edge type (relation) r. A GNN maps each node $v \in \mathcal{V}$ onto representation vectors $\mathbf{h}_v^1 \dots \mathbf{h}_v^K$ by repeatedly aggregating the representations of the immediate neighbours of every node and updating node representations in every step of the encoding process, each associated with one of K layers of the GNN. Relational GNNs must also take into account the edge types between nodes in the graph $\mathcal{G}$. In the following sections, we use the message passing framework where a single GNN layer/step is decomposed into a three-step process:

$$\mathbf{h}_v^{(k)} = \phi(\mathbf{h}_v^{(k-1)}, \gamma(\{\mu(\mathbf{h}_u^{(k-1)}, r)\}_{(u,v,r)\in\mathcal{E}(\cdot,v)})) \ , \tag{1}$$

where $\mu(\cdot)$ computes a "message" along a graph edge using the neighbour representation $\mathbf{h}_v^{(k-1)}$ and the edge type r of an edge from u to v [2], $\gamma(\cdot)$ aggregates the incoming messages into a single vector, and $\phi(\cdot)$ computes a new representation for node v. $\mathcal{E}(\cdot, v)$ denotes the set of all edges in $\mathcal{G}$ that end in v. After subsequently applying Eq. 1 K times[3] to each node of the graph, the final node representations $\mathbf{h}_v^{(K)}$ can be used for different tasks, such as graph or node classification. One commonly used message function is a relation-specific linear transformation, which is used by RGCNs and GGNNs: $\mu_{\mathrm{MM}}(\mathbf{h}_u, r) = \mathbf{W}_r \mathbf{h}_u$, where $\mathbf{W}_r$ is a $\mathbb{R}^{D\times D}$ parameter matrix associated with edge type r. Moreover, GGNNs implement the update function $\phi(\cdot)$ using a GRU: $\mathbf{h}_v^{(k)} = \mathrm{GRU}(\overline{\mathbf{h}}_v^{(k-1)}, \mathbf{h}_v^{(k-1)})$, where the GRU's *input* argument is $\overline{\mathbf{h}}_v^{(k-1)} = \gamma(\{\mu(\mathbf{h}_u^{(k-1)}, r)\}_{(u,v,r)\in\mathcal{E}(\cdot,v)})$, the vector for the aggregated neighbourhood of v.

3 GATED RELATIONAL GRAPH ATTENTION NETWORKS

We develop our method starting from the GAT (Veličković et al., 2018) architecture, which uses (multi-head) attention to selectively assimilate the features of the nodes in the neighbourhood of a node into a new representation for that node. Compared to the simple addition used in GCNs (Kipf & Welling, 2016; Schlichtkrull et al., 2018) and GGNNs (Li et al., 2015), the attention mechanism can learn to "filter" the neighbourhood of a node, omitting nodes irrelevant for the task. We believe this is crucial for learning long-range dependencies, since a simple addition squashes the entire K-hop *receptive field* of a K-layer GNN into a single vector uniformly, making it very challenging to distinguish features from the exponentially growing number of connected distant nodes. We propose several changes to the GAT architecture, that improve its ability to model long-range patterns and that are described in the following. We call the resulting architecture the *Gated Relational Graph Attention neTwork* (GR-GAT).

3.1 INTEGRATING EDGE TYPE INFORMATION

Using the simple matrix multiplication-based $\mu_{\mathrm{MM}}(\cdot)$ function for adding edge type information to a node state can be problematic because it may over-parameterize the edge type update, leading to easier over-fitting and requiring matrix decomposition to reduce the over-parameterization (Schlichtkrull et al., 2018). Alternatively, VR-GCN (Ye et al., 2019) and CompGCN (Vashishth et al., 2020) make use of vector-based edge type parameterization, and explore composition functions inspired by graph embedding learning methods (Bordes et al., 2013; Yang et al., 2014; Nickel et al., 2016).

We also parameterize our edge types using vectors but propose the following gated message function (see also Figure 1 for an illustration):

$$\mathbf{z} = \mathsf{CELU}(\mathbf{b}_{\mathrm{A}} + \mathbf{W}_{\mathrm{A}}[\mathbf{h}_u; \mathbf{a}_r])) \ , \qquad \mathbf{m} = \sigma(\mathbf{b}_{\mathrm{M}} + \mathbf{W}_{\mathrm{M}}\mathbf{z}) \ , \tag{2}$$

$$\mathbf{u} = \mathbf{b}_{\mathrm{B}} + \mathbf{W}_{\mathrm{B}}\mathbf{z} \ , \qquad \mu_{\mathrm{GCM}}(\mathbf{h}_u, r) = \mathbf{m} \odot \mathbf{h}_u + (1 - \mathbf{m}) \odot \mathbf{u} \ , \tag{3}$$

[2] More generally, $\mu(\mathbf{h}_u^{(k-1)}, \mathbf{h}_v^{(k-1)}, \mathbf{h}_e)$ can also be defined, which supports multiple features $\mathbf{h}$ for edge e.
[3] Optionally with different parameters for $\mu(\cdot)$, $\phi(\cdot)$ and/or γ in every step.

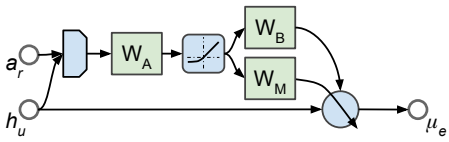

Figure 1: The proposed relation-aware message function.

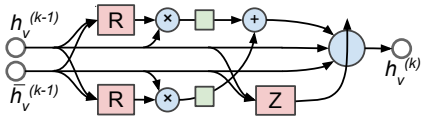

Figure 2: The proposed update function, the Symmetrically Gated Recurrent Unit.

where $\mathbf{a}_r$ is a vector $\in \mathbb{R}^D$ associated with the edge type from u to v, and $\mathbf{W}_\mathrm{A}$, $\mathbf{W}_\mathrm{B}$, and $\mathbf{W}_\mathrm{M}$ are trainable weight matrices (which have their corresponding biases $\mathbf{b}_\mathrm{A}$, $\mathbf{b}_\mathrm{B}$, and $\mathbf{b}_\mathrm{M}$) that are shared between different edge types. We use the CELU (Barron, 2017) activation function. Similarly to highway networks (Srivastava et al., 2015), $\mu_\mathrm{GCM}(\cdot)$ implements a gate that interpolates (based on $\mathbf{m}$) between the original node state $\mathbf{h}_u$ and the relation-aware update $\mathbf{u}$ using a two-layer feedforward network. Many different other choices are possible for incorporating relation information into the message from a neighbour. We chose the above formulation because it is more expressive, and enables us to effortlessly support additional edge features that are not based on the relation embedding. We also expect it to improve backpropagation and reduce over-parameterization. See also Appendix A.2.

3.2 Attention-based Neighbourhood Aggregation

To implement the aggregation function $\gamma(\cdot)$, we adapt the scaled multi-head multiplicative attention mechanism (Vaswani et al., 2017). The per-head attention distributions are computed as described for transformers, with two small but important differences: (1) we use the edge type embedding directly in computing the attention, and (2) we do *not* use linear transformations to obtain value vectors.

The query and key vectors for each attention head h (of in total H heads), $\mathbf{q}_v^{(h)} \in \mathbb{R}^D$ for node v and $\mathbf{k}_e^{(h)} \in \mathbb{R}^D$ for an edge $e = (u, v, r)$ ending in v, respectively, are computed using head-specific linear transformations parameterized by $\mathbf{Q}^{(h)} \in \mathbb{R}^{D/H \times D}$ and $\mathbf{K}^{(h)} \in \mathbb{R}^{D/H \times 2D}$:

$$\mathbf{q}_v^{(h)} = \mathbf{Q}^{(h)}\mathbf{h}_v \ , \qquad \mathbf{k}_e^{(h)} = \mathbf{K}^{(h)}[\mu(\mathbf{h}_u, r); \mathbf{a}_r] \ , \tag{4}$$

where $[\cdot;\cdot]$ denotes the (vertical) concatenation of two (column) vectors and $\mathbf{a}_r$ is the same edge type-specific parameter vector that is used in Eqs. 2-3. Note that Eq. 4 is slightly different from a standard attention mechanism, which would compute key vectors as $\mathbf{K}^{(h)}\mu(\mathbf{h}_u, r)$. We expect that including the edge type vector $\mathbf{a}_r$ directly in the keys will make it easier to learn to place attention.

The attention score $w_e^{(h)}$ for an edge e and head h is computed by a scaled dot product and then normalized to obtain $\alpha^{(h)}$ as usual: $w_e^{(h)} = \frac{\mathbf{q}_v^{(h)} \cdot \mathbf{k}_e^{(h)}}{\sqrt{D/H}}$ and $\alpha_e^{(h)} = \frac{\exp(w_e^{(h)})}{\sum_{e^* \in \mathcal{E}(\cdot,v)} \exp(w_{e^*}^{(h)})}$.

Unlike done for multi-head attention in transformers and previous work (Veličković et al., 2018; Busbridge et al., 2019), we do not use linear transformations to obtain the value vectors. Instead, we split the vector $\mu(\mathbf{h}_u, r) = \boldsymbol{\mu}_e = [\boldsymbol{\mu}_e^{(1)}, \ldots, \boldsymbol{\mu}_e^{(H)}]$ in H equally sized chunks and take the h-th part $\boldsymbol{\mu}_e^{(h)} \in \mathbb{R}^{D/H}$ as value vector for the h-th head. The attention scores $\alpha_e^{(h)}$ and value vectors $\boldsymbol{\mu}_e^{(h)}$ are then used to compute the neighbourhood aggregation vector $\overline{\mathbf{h}}_v^{(h)}$ for head h as a weighted sum. The full neighbourhood aggregation vector $\overline{\mathbf{h}}_v$ is then given as a concatenation over all heads h:

$$\overline{\mathbf{h}}_v^{(h)} = \sum_{e \in \mathcal{E}(\cdot,v)} \alpha_e^{(h)} \boldsymbol{\mu}_e^{(h)} \ , \qquad \overline{\mathbf{h}}_v = [\overline{\mathbf{h}}_v^{(1)}; \ldots; \overline{\mathbf{h}}_v^{(H)}] \ . \tag{5}$$

3.3 Update function

The simplest option for the update function $\phi(\cdot)$ is to use an identity function $\phi(x) = x$ or an element-wise activation function, such as a sigmoid: $\phi(x) = \sigma(x)$. However, inspired by the GRU-based update in GGNNs (Li et al., 2015), we explore two other options: (1) a simple gated two-layer MLP update that uses Eqs. 2-3, substituting $\mathbf{h}_u$ with $\mathbf{h}_v$ and $\mathbf{a}_r$ with $\overline{\mathbf{h}}_v$ and (2) a GRU-based function (see Section 3.3.1 below), specifically adapted for use in GNNs.

3.3.1 SYMMETRICALLY GATED RECURRENT UNIT

The GGNN has been used for different NLP tasks (Li et al., 2015; Bogin et al., 2019; Beck et al., 2018; Marcheggiani & Titov, 2017). While the GRU used in GGNNs enables easy backpropagation over a large number of layers *"vertically"* (from top-level state $\mathbf{h}_v^{(K)}$ of a node to its initial state $\mathbf{h}_v^{(0)}$), it may suffer from vanishing gradients "*horizontally*" (w.r.t. distant nodes) with increasing depth of the network. In fact, considering the GRU equations, it becomes clear that every backpropagation path to a distant node can suffer from vanishing or exploding gradients, similarly to a simple (non-gated) RNN, because of repeated matrix multiplications (both in $\mu(\cdot)$ and in $\phi(\cdot) = \mathrm{GRU}(\cdot)$), and $\tanh(\cdot)$ nonlinearities. See Appendix A.3 for a more detailed discussion.

Following the ideas behind LSTMs and GRUs, we propose to change the GRU equations such that both inputs of the update function are treated as states. We call the proposed update function, which is described by the following equations, a *Symmetrically Gated Recurrent Unit* (SGRU):

$$\mathbf{r}_h^{(k)} = \sigma(W_{r_h}\overline{\mathbf{h}}_v^{(k-1)} + U_{r_h}\mathbf{h}_v^{(k-1)} + b_{r_h}) \ , \quad \mathbf{r}_x^{(k)} = \sigma(W_{r_x}\overline{\mathbf{h}}_v^{(k-1)} + U_{r_x}\mathbf{h}_v^{(k-1)} + b_{r_x}) \tag{6}$$

$$\mathbf{z}_x^{(k)} = W_{z_x}\overline{\mathbf{h}}_v^{(k-1)} + U_{z_x}\mathbf{h}_v^{(k-1)} + b_{z_x} \ , \quad \mathbf{z}_h^{(k)} = W_{z_h}\overline{\mathbf{h}}_v^{(k-1)} + U_{z_h}\mathbf{h}_v^{(k-1)} + b_{z_h} \tag{7}$$

$$\mathbf{z}_u^{(k)} = W_{z_u}\overline{\mathbf{h}}_v^{(k-1)} + U_{z_u}\mathbf{h}_v^{(k-1)} + b_{z_u} \ , \quad \hat{z}_{x,i}^{(k)}, \hat{z}_{h,i}^{(k)}, \hat{z}_{u,i}^{(k)} = \mathrm{softmax}([z_{x,i}^{(k)}, z_{h,i}^{(k)}, z_{u,i}^{(k)}]) \tag{8}$$

$$\hat{\mathbf{h}}_v^{(k)} = \tanh(W(\overline{\mathbf{h}}_v^{(k-1)} \odot \mathbf{r}_x^{(k)}) + U(\mathbf{h}_v^{(k-1)} \odot \mathbf{r}_h^{(k)})) \tag{9}$$

$$\mathbf{h}_v^{(k)} = \hat{\mathbf{z}}_x^{(k)} \odot \overline{\mathbf{h}}_v^{(k-1)} + \hat{\mathbf{z}}_h^{(k)} \odot \mathbf{h}_v^{(k-1)} + \hat{\mathbf{z}}_u^{(k)} \odot \hat{\mathbf{h}}_v^{(k)} \ . \tag{10}$$

The SGRU differs from the GRU used in the GGNN in (i) introducing an additional reset gate $\mathbf{r}_x^{(k)}$ that is applied to the aggregated neighbour states $\overline{\mathbf{h}}_v^{(k-1)}$ and (ii) computing the output state $\mathbf{h}_v^{(k)}$ as a three-way mixture between the previous node state $\mathbf{h}_v^{(k-1)}$, the aggregated neighbour states $\overline{\mathbf{h}}_v^{(k-1)}$, and the candidate state $\hat{\mathbf{h}}_v^{(k)}$ instead of a two-way mixture between $\mathbf{h}_v^{(k-1)}$ and $\hat{\mathbf{h}}_v^{(k)}$. The interpolation coefficients $\hat{\mathbf{z}}$ are produced by an elementwise softmax between the three $\mathbf{z}$ vectors. See also Figure 2 for an illustration.

4 EXPERIMENTAL ANALYSIS

We wish to answer the following questions in our experiments:

- Q.1 How well can the baselines learn simple rules for a sequence classification task? (4.1)
- Q.2 How does the proposed model compare to the baselines on a node classification task on graphs where deep networks are required for the task to be solved well? (4.2)
- Q.3 Are all the proposed changes necessary to achieve the best performance? (4.2)
- Q.4 How does the proposed method perform in a shallow setup on entity classification using real-world data? (4.3)
- Q.5 Does the performance degrade on entity classification when deeper networks are used? (4.3)

We use the following baselines: (1) RGCN (Schlichtkrull et al., 2018), (2) GGNN (Li et al., 2015) and (3) RGAT (Busbridge et al., 2019). We implement our RGCN and GGNN baselines based on the code provided by the DGL framework. For GGNN, we share the parameters between layers. We found several relational variants of the GAT in the literature (Busbridge et al., 2019; Brockschmidt, 2019; Sinha et al., 2019) and chose to adapt (Busbridge et al., 2019) for our attention-based baseline (RGAT). See Appendix A.1 for an elaborate description of the RGAT we implement and other details regarding memory-efficiency.

We consider the following ablations of our full method (GR-GAT) in our experiments: (1) "GR-GAT (*X*), mean agg" replaces the attention based aggregation function with a simple mean (equivalent to uniform attention), (2) "GR-GAT (*X*), edge matrix" replaces μ_{GCM} with μ_{MM}, and (3) "GR-GAT (*X*), value transform" uses a linear layer to transform the value while using attention, as is standard in for example transformers. The *X* in GR-GAT(*X*) is the update function and can be one of the following: *Ident* (identity function), *GRU* (Gated Recurrent Unit), *Gate* or *SGRU* (see Section 3.3). If *X* is omitted, the *SGRU* has been used. We use weight sharing between all layers of GR-GAT.

	L=5	L=10	L=15
RGCN	91.8 ± 2.4	53.5 ± 4.9	6.3 ± 8.9
GGNN	69.7 ± 3.3	2.5 ± 1.8	3.3 ± 2.4
RGAT	91.0 ± 1.2	75.1 ± 3.2	13.4 ± 7.4
GR-GAT	$\mathbf{98.6 \pm 1.0}$	$\mathbf{96.7 \pm 0.7}$	$\mathbf{93.2 \pm 2.7}$

Table 1: Conditional recall results. The average accuracy on the test set is reported. We use the SGRU update function in this experiment. L is the length of the generated sequences.

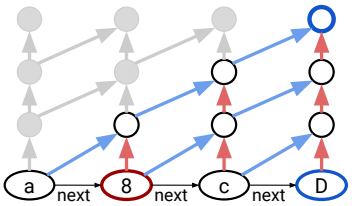

Figure 3: The input graph (ovals) for an example of the Conditional Recall task and the computational graph for 3 GNN layers applied on it.

4.1 Conditional Recall

In this experiment, we aim to answer Q.1 and we show that the different popular relational GNNs are not able to solve a simple sequence classification task that involves remembering one of the symbols in a sequence according to a set of simple rules.

We define a sequence classification task where given a sequence of characters $[x_1, \ldots, x_N]$, the model is asked to predict the correct class based on the representation of the last node (corresponding to input x_N). The input sequences consist of strings of letters and digits of a given length (which was varied between different experiments). The class of the sequence is determined by the following rules: (i) if there is a digit in the sequence, the first digit corresponds to the class label; (ii) otherwise, if there is an upper case character, the first upper case character is the class label; (iii) otherwise, the class is given by the first character in the sequence. Some examples are: "**a**bcdefg" $\rightarrow$ "a", "abc**D**efg" $\rightarrow$ "D", "abcd**3**Fg" $\rightarrow$ "3", "abCd**3**fg" $\rightarrow$ "3". We generated 20 examples per output class for a total of 1220 examples and split the data in 80/10/10 train/validation/test splits. For more details about our experimental setup for this task we refer to Appendix A.4.

We transform the input sequences into a graph by (i) creating a node for every character of the sequence and (ii) adding edges labeled using the `next` type between every adjacent element in the sequence. Given these edges, the GNN has to use at least N layers/steps in order to propagate information from the first node to the last node. We increase the number of layers for tasks with longer sequences: the number of layers is $N + 1$. The readout for prediction is done by taking the representation of the last node in the sequence.

Note that a simple GRU or LSTM would solve the task well, and different graph representation and readout methods could perform better. However, here we deliberately choose the above formulation to measure how well the different GNNs can learn over a larger numbers of hops in a very simple graph where the decision node is several hops away.

4.1.1 Results and Discussion:

We show the results for the different GNNs and varying sequence lengths (L) in the task in Table 1 (Note that all examples have the same, specified length in one experiment). While all baselines reach acceptable accuracy for length 5, their accuracy severely suffers when the length is increased. Surprisingly, the GGNN severely underperforms compared to the skip-less RGCN and RGAT in our experiments. We suspect that, because of the gated skip connections to lower-layer node representations, it is easier for the GGNN to memorize combinations of nearby nodes rather than learning meaningful patterns. In contrast, our method solves the task well for all lengths. Please note that the construction of the problem ensures that each node receives only one message – from the previous token, minimizing possible over-smoothing and over-squashing and also minimizing the advantages of using weighted sums (such as average or attention). Also, there are only two edge types in this task (`next` and `self`), which avoids over-parameterization due to the number of relations.

It is important to keep in mind that the computational graph for a GNN applied to this graph is different from an RNN applied to the same sequence. The "update distance" (the number of RNN or GNN updates performed) between the last state h_T of an RNN and a token x_{T-k} (from k timesteps before) is k. In contrast, as we illustrate in Figure 3, the update distance between the top-level features

of any node and the initial features of any node is always equal to the number of layers in the GNN. In the figure, the final vector of the blue node is used for prediction and the red node specifies the desired output. Red arrows are "vertical" self-messages and blue arrows "horizontal" messages to neighbours. In the case of an RGCN, the neighbours and the node itself are treated equally, while the GGNN provides a skip connection to previous node features, enabling easier backpropagation "vertically" within the node. However, this does not improve backpropagation to distant nodes in the graph (see also Appendix A.3).

We also inspected the gradients w.r.t. the input vectors and experimented with different initializations for the edge type matrices of the baselines. We found that the initialization indeed strongly affects the gradient magnitudes and affects training behaviour. However, even when the weights are initialized correctly and the initial gradients are not small, the baselines are still not able to solve the task well for larger task lengths. We believe that our proposed changes encourage more effective weight sharing across nodes and layers and may benefit from other effects that make it easier for the network to generalize. In their work on ResNets, He et al. (2016) similarly found that the improvement can not be explained by gradient magnitudes alone.

4.2 TREE MAX

In this experiment, we aim to answer Q.2 and Q.3 using a synthetic node classification task on trees. The input graphs are trees with nodes labeled with random integers between 1 and 100. The expected output labels for the node classification task are defined as the largest value of all the descendants of a node and the node itself. The graphs contain edges from a parent to its children, and from children to their parent, as well as self-edges. We use numbered child edges and numbered child-of edges, for example, :CHILD-1-OF for the edge going from the first child of a node to its parent.

As an example of this task, consider the following input tree: *(1 (2 (3) (4)) (5 (6) (7 (8) (9) (10))))*. The nodes in the tree should be labeled with the value of their highest-valued descendant. Thus, the tree with its output labels will become: *(10 (4 (3) (4)) (10 (6) (10 (8) (9) (10))))*. In this case, to correctly predict the output label of the root node (1), the GNN must handle three hops. Please see Figure 5 in Appendix A.5 for an illustration of the graph constructed for this example.

We generate 800 examples, each containing one tree with a depth between 5 and 15. The largest generated trees contained more than 200 nodes. We use 17-layer GNNs in our experiments, and experiment with fewer layers for our baselines. A 50/25/25% training/validation/test split is used. More details about the data and the experimental setup can be found in Appendix A.6.

4.2.1 RESULTS AND DISCUSSION:

As shown in Table 2, the best node-level and graph-level[4] accuracies were obtained using the proposed GR-GAT model. The choice of update function (SGRU, Gate or Identity) does not have a large effect on the results. However, we can see that the proposed SGRU performs substantially better than the normal GRU. The ablation study further indicates that the gated edge function (Section 3.1) and adapted attention mechanism (Section 3.2) are essential for achieving the best performance since the "edge matrix" and "value transform" variants reach significantly worse performance. For the RGAT baseline, we obtained the best results with 12 layers, which is insufficient to capture the longest necessary dependencies.

Note that for correct node classification, most nodes don't require long-range information propagation to other nodes and can be solved within five hops. In fact, only 5.633 ± 0.131 % of all nodes across all examples in the test portion of our dataset require handling five hops or more and only 0.421 ± 0.035 % require handling ten hops or more. However, it is important to keep in mind that the number of updates separating the final representation of any node to the initial representation of any node is equal to the number of layers (see also Figure 3). Thus, we expected the GGNN to perform better than RGCN because of the easier backpropagation to lower node representations.

[4]The graph-level accuracy is 100% for an example only if all nodes in the graph have been classified correctly, and is 0% otherwise

	Node-level	Graph-level
RGCN	95.7 ± 0.4	31.7 ± 1.5
GGNN	97.0 ± 0.2	40.6 ± 3.5
RGAT	89.9 ± 1.3	13.4 ± 3.8
GR-GAT (SGRU)	$\mathbf{99.7 \pm 0.1}$	$\mathbf{89.0 \pm 3.1}$
GR-GAT (Gate)	99.7 ± 0.1	86.9 ± 1.9
GR-GAT (Ident)	99.6 ± 0.2	85.3 ± 6.8
GR-GAT (SGRU), mean agg	99.0 ± 0.1	68.5 ± 3.2
GR-GAT (GRU)	95.0 ± 6.3	$48.1 \pm 25.$
GR-GAT (SGRU), edge matrix	96.8 ± 1.9	45.1 ± 18.5
GR-GAT (Ident), value transform	94.3 ± 1.5	23.2 ± 11.5

Table 2: Node and graph-level accuracies over the test set of the Tree Max task. Top part shows performance of commonly used baseline methods. Middle parts shows performance of our proposed method using different update functions. Bottom part show different ablations (see text).

	AM
RGCN	89.3
SynGCN	86.2 ± 1.9
WGCN	90.2 ± 0.9
CompGCN	$\mathbf{90.6 \pm 0.2}$
GR-GAT	89.7 ± 0.6

Table 3: Accuracy on the test set of the AM dataset.

	L=2 (2 hops)	L=10 (2 hops)	L=10
RGCN	95.8 (*)	88.0 ± 1.3	89.8 ± 6.9
GGNN	94.4 ± 2.2	92.6 ± 1.3	95.4 ± 2.6
GR-GAT	96.1 ± 1.4	92.6 ± 1.3	93.2 ± 3.9
(Ident)	$\mathbf{96.3 \pm 1.3}$	92.6 ± 2.6	$\mathbf{96.3 \pm 1.3}$

Table 4: Accuracy on test set of AIFB for different data settings (see text). (*) - number taken from Schlichtkrull et al. (2018). L is the number of GNN layers used. A (randomly sampled) subgraph of many hops is used for the L=10 setting in the last column.

4.3 Entity classification using real-world data

To answer Q.4 and Q.5, we run experiments on real-world data for the node classification tasks, where a certain property about an entity must be predicted. It is not clear whether this dataset could benefit from a GNN that is able to model longer patterns so we do not expect to see improvements using the original data. With these entity classification experiments, we mainly want to verify that our proposed model does not severly underperform compared to previously proposed relation-aware GNNs in a shallow setup and study the performance of our model when more layers are used.

We use the same experimental setup as in Schlichtkrull et al. (2018), who use previously proposed datasets for entity classification in knowledge graphs. We use their preprocessed datasets and the same train/test splits. However, compared to Schlichtkrull et al. (2018), who used whole-batch gradient descent using CPU and fit the entire datasets in memory, we implement subsampled mini-batch training, which can affect the results. For more details on the experimental setup, the graph subsampling procedures and hyperparameter choice, see Appendix A.7.

We perform experiments using two layers (similarly to Schlichtkrull et al. (2018)) and report the results in Table 3 for the AM dataset and in the first column of Table 4 for the AIFB dataset. We also perform further experiments using the AIFB dataset to measure how much GR-GAT degrades when more layers are used.

4.3.1 Results and Discussion

The results in Tables 3 and the first column of Table 4 show that our method is competitive with the previously proposed relation-aware GNNs in a two-layer setup. In the second column of Table 4, we can see that GR-GAT degrades less strongly than RGCN when ten layers are used on the same two-hop subgraph as in the first column. The GGNN, which has vertical skip-connections and also shares parameters between layers, is similarly more resilient to the increase in layer count.

When we use a sampled subgraph built by random walks from the labeled node (see also Appendix A.7), which is the setting reported in the last column of Table 4, we see some small improvement from all methods compared to a 10-layer network applied on the two-hop subgraphs. To verify whether this increase might have been due to the regularizing effects of dropping nodes and edges, we ran some additional experiments with a 10-layer GR-GAT (Ident) on a random walk-sampled subgraph where the reset probability was raised to 50%, which strongly discourages long walks. The number of nodes and edges was reduced to 500 and 1000, respectively (see also Appendix A.7). We found that this setup results in 96.3 ± 1.3 % test accuracy with the same hyperparameters. Thus, we conjecture that the improvements are due to the regularizing effect of randomly removing nodes and edges. To conclude, none of the tested methods show improvements in performance on the AIFB dataset when wider subgraphs and more layers are used. We believe this is due to the characteristics of the dataset.

5 Related Work

Li et al. (2019a) investigate the GCN equivalents of ResNet (He et al., 2016) and DenseNet (Huang et al., 2017), evaluated on point cloud semantic segmentation, as well as dilated aggregation. Residual-GCNs and Dense-GCNs (Li et al., 2019a), Highway GCN (Rahimi et al., 2018), Column Networks (Pham et al., 2017) and Li et al. (2020) are similar to the GGNN (Li et al., 2015) in using gated, residual or concatenated skip connections to previous nodes states. Xu et al. (2018) propose to add skip connections from all layers straight to the output layer. The neighbourhood dilation proposed by Li et al. (2019a) for point cloud semantic segmentation increases the receptive field of the network without requiring the information to pass through other nodes. However, it is not clear how to apply this technique to multi-relational graphs. Additionally, different normalization techniques (Zhou et al., 2020; Zhao & Akoglu, 2020; Li et al., 2020) have very recently been proposed that seem to improve the training of deep GNNs. In contrast to previous work, we focus on improving the communication between neighbours in GNNs for multi-relational graphs.

A family of methods closely related to GNNs are TreeLSTMs (Tai et al., 2015) and GraphLSTMs (Peng et al., 2017; Liang et al., 2016; Song et al., 2018; Bresson & Laurent, 2017). Some versions of these methods have similar beneficial properties regarding backpropagation to distant nodes but are not suited for use in a general graph setup. Song et al. (2018) adapt Graph LSTMs into a message passing network, however, their implementation of the LSTM-based update suffers from the same issues as the GGNN. Compared to our proposed method, GraphLSTMs use independent forget gates instead of attention. Song et al. (2018) uses a global forget gate, which reduces the ability of the model to focus more on certain nodes. The mixing gates in our SGRU bear some similarity to those in the Sentence State LSTM (Zhang et al., 2018). However, the original formulation of the Sentence State LSTM is not applicable for graphs in general. Fi-GNN (Li et al., 2019b), similarly to the base model in our work, uses a GGNN with attention-based aggregation. However, it lacks our modifications and investigation to improve long-range pattern learning.

6 Conclusion

In this work, we proposed a novel GNN architecture that can handle long-term dependencies in multi-relational graphs better than commonly used GNN architectures. In experiments on synthetic tasks, we show that popular GNNs are bad at learning long-range patterns, and are outperformed by our proposed method. In experiments on entity classification using real data, we show that our proposed method is competitive with previous work. Our ablation study shows that the proposed changes w.r.t. the GAT and the GGNN are essential. However, it appears that a node update function is not necessary and GR-GAT (Ident) works well. While it is indeed possible for the attention mechanism alone to retain previous node representations, we believe that the SGRU could be useful for node-wise updates and as an alternative, more fine-grained, mechanism to retain previous node representations. In future work, we plan to further extend the experimental analysis, identify real-world datasets that benefit more from long-range modeling abilities and investigate how to further improve training, such as investigating normalization and regularization techniques.

REFERENCES

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

Jonathan T Barron. Continuously differentiable exponential linear units. *arXiv*, pp. arXiv–1704, 2017.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 273–283, 2018.

Ben Bogin, Jonathan Berant, and Matt Gardner. Representing schema structure with graph neural networks for text-to-sql parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4560–4565, 2019.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pp. 2787–2795, 2013.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Marc Brockschmidt. Gnn-film: Graph neural networks with feature-wise linear modulation. 2019.

Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y Hammerla. Relational graph attention networks. *arXiv preprint arXiv:1904.05811*, 2019.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *arXiv preprint arXiv:1909.03211*, 2019.

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.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272. JMLR. org, 2017.

William L. Hamilton. Graph representation learning (draft, september 2020). *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. URL `http://arxiv.org/abs/1412.6980`.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9267–9276, 2019a.

Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

Zekun Li, Zeyu Cui, Shu Wu, Xiaoyu Zhang, and Liang Wang. Fi-gnn: Modeling feature interactions via graph neural networks for ctr prediction. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 539–548, 2019b.

Xiaodan Liang, Xiaohui Shen, Jiashi Feng, Liang Lin, and Shuicheng Yan. Semantic object parsing with graph LSTM. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (eds.), *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I*, volume 9905 of *Lecture Notes in Computer Science*, pp. 125–143. Springer, 2016. doi: 10.1007/978-3-319-46448-0_8. URL https://doi.org/10.1007/978-3-319-46448-0_8.

Diego Marcheggiani and Ivan Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1506–1515, 2017.

Maximilian Nickel, Lorenzo Rosasco, Tomaso A Poggio, et al. Holographic embeddings of knowledge graphs. In *AAAI*, volume 2, pp. 3–2, 2016.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pp. 1310–1318, 2013.

Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics*, 5:101–115, 2017.

Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. Column networks for collective classification. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 2485–2491, 2017.

Afshin Rahimi, Trevor Cohn, and Timothy Baldwin. Semi-supervised user geolocation via graph convolutional networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2009–2019, 2018.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2019.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pp. 593–607. Springer, 2018.

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L Hamilton. Clutrr: A diagnostic benchmark for inductive reasoning from text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4496–4505, 2019.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. N-ary relation extraction using graph-state lstm. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2226–2235, 2018.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1556–1566, 2015.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BylA_C4tPr.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Petar 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.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5449–5458. PMLR, 2018. URL http://proceedings.mlr.press/v80/xu18c.html.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

Chaoqi Yang, Ruijie Wang, Shuochao Yao, Shengzhong Liu, and Tarek Abdelzaher. Revisiting" over-smoothing" in deep gcns. *arXiv preprint arXiv:2003.13663*, 2020.

Rui Ye, Xin Li, Yujie Fang, Hongyu Zang, and Mingzhong Wang. A vectorized relational graph convolutional network for multi-relational network alignment. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 4135–4141. ijcai.org, 2019. doi: 10.24963/ijcai.2019/574. URL https://doi.org/10.24963/ijcai.2019/574.

Yue Zhang, Qi Liu, and Linfeng Song. Sentence-state lstm for text representation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 317–327, 2018.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkecl1rtwB.

Kuangqi Zhou, Yanfei Dong, Wee Sun Lee, Bryan Hooi, Huan Xu, and Jiashi Feng. Effective training strategies for deep graph neural networks. *arXiv preprint arXiv:2006.07107*, 2020.

A APPENDIX

A.1 BASELINES

A.1.1 A NOTE ON MEMORY-EFFICIENCY OF THE BASELINES

When experimenting with the baselines (RGCN, GGNN and RGAT), we quickly ran out of memory for larger hidden dimensions of the GNN and larger numbers of layers. This is caused by the relation matrices, which have $O(n^2)$ parameters for every relation. In the implementations we used, the relations were indexed from a tensor and have to be retained in memory for every edge. While more memory-efficient implementations are possible, the existing ones sacrifice execution time, which can increase significantly with a large number of different relations.

Gradient accumulation significantly slows down training. As a simple fix to maintain reasonable training speed for our experiments, we replace $\mu_{\text{MM}}(\cdot)$ in our RGCN and GGNN with $\mu_{\text{MM-red}}(\mathbf{h}_u) = W_B W_r^* W_A \mathbf{h}_u$, where W_r^* is an edge type specific square matrix of lower dimensionality than W_r and W_A and W_B are matrices projecting into and out of W_r^*'s dimensionality that are shared for all edge types. Note that we did this only for the Tree Max experiments.

A.1.2 RGAT BASELINE

Like in (Busbridge et al., 2019), we use relation-specific transformation matrices $\mathbf{W}^{(r)}$ and relation and head-specific query and key matrices $\mathbf{Q}^{(r,h)}$ and $\mathbf{K}^{(r,h)}$.

First, we define relation-dependent representations for a node, which is computed based on its current state $\mathbf{z}^{(k-1)}$:

$$\mathbf{g}_i^{(r)} = \mathbf{W}^{(r)} \mathbf{z}_i^{(k-1)} \quad . \tag{11}$$

Subsequently, for every head h, we define the relation-specific query, key and value projections as:

$$\mathbf{q}_i^{(h,r)} = \mathbf{Q}^{(h,r)} \mathbf{g}_i^{(r)} \tag{12}$$

$$\mathbf{k}_i^{(h,r)} = \mathbf{K}^{(h,r)} \mathbf{g}_i^{(r)} \tag{13}$$

$$\mathbf{v}_i^{(h,r)} = \mathbf{V}^{(h,r)} \mathbf{z}_i^{(k-1)} \tag{14}$$

We compute attention between messages $\boldsymbol{\mu}_x$, where x indexes over all edges in the input graph. First, we compute the attention score for a message as in (Vaswani et al., 2017):

$$s_{\boldsymbol{\mu}_x}^{(h)} = \mathbf{q}_i^{(h,r)} \cdot \mathbf{k}_j^{(h,r)} \quad , \tag{15}$$

where $\boldsymbol{\mu}_x$ is the message sent along an edge $j \to i$ labeled by the relation r. Please note that a node i may receive multiple messages from a node j, and that those messages could contain the same relation.

The scores are normalized over all messages that node i receives:

$$\alpha_{\boldsymbol{\mu}_x}^{(h)} = \frac{s_{\boldsymbol{\mu}_x}^{(h)}}{\sum_{\boldsymbol{\mu}_{x'}} s_{\boldsymbol{\mu}_{x'}}^{(h)}} \tag{16}$$

The normalized scores are used to compute a summary:

$$\mathbf{z}_i^{(h,k)} = \sum_{\boldsymbol{\mu}_x} \mathbf{v}_j^{(h,r)} \alpha_{\boldsymbol{\mu}_x}^{(h)} \quad , \tag{17}$$

where r is the relation of $\boldsymbol{\mu}_x$ and j is the source node id and i is the target node id of message $\boldsymbol{\mu}_x$.

The updated representation for node i is then the concatenation over all heads, fed through the activation function σ:

$$\mathbf{z}_i^{(k)} = \sigma\big([\mathbf{z}_i^{(0,k)}, \ldots, \mathbf{z}_i^{(H,k)}]\big) \quad . \tag{18}$$

We experiment with a ReLU, as well as with a linear σ.

A.2 NOTES ON RELATION-AWARE MESSAGE FUNCTIONS.

Many different message function implementations are possible that take into account the relation connecting a node to a neighbour. The default message function used in earlier literature is simply a matrix multiplication:

$$\mu_{\mathrm{MM}}(\mathbf{h}_u, r) = \mathbf{W}_r \mathbf{h}_u \tag{19}$$

This can easily lead to overparameterization and leads to difficulties implementing the GNNs in a both memory-efficient and computationally efficient way.

In CompGCN, Vashishth et al. (2020) propose the following function:

$$\mu_{\mathrm{CompGCN}}(\mathbf{h}_u, r) = \mathbf{W}_{\lambda(r)} \phi(\mathbf{h}_u, \mathbf{a}_r) \ , \tag{20}$$

where $\mathbf{W}_{\lambda(r)}$ is one of the three matrices: one for forward relations, one for reverse and one for self-edges, and $\phi(\mathbf{h}_u, \mathbf{a}_r)$ is a composition function that combines the neighbour state and relation vector. For the composition function, the authors explore functions inspired by knowledge graph embedding literature: subtraction ($\mathbf{h}_u - \mathbf{a}_r$), multiplication or circular-correlation.

While this provides an alternative solution to over-parameterizing relations, and enables more efficient implementation, it does not improve gradient behavior since the backpropagation paths to the lower node features as well as to distant nodes still contain linear transformation and non-linearities.

To improve backpropagation, we can make use of (gated or residual) skip connections. The residual version would have the following general form:

$$\mu_{\mathrm{Res}}(\mathbf{h}_u, r) = \mu_{\mathrm{X}}(\mathbf{h}_u, r) + \mathbf{h}_u \ , \tag{21}$$

where μ_{X} is some function. Similarly to transformers and ResNets, we choose to use a two-layer MLP, which is more expressive than μ_{MM}, and unlike the composition-based message functions in Vashishth et al. (2020), can effortlessly incorporate additional edge features that are not based on the relation embeddings.

We chose the gated skip connection described in the text over the simple residual one because it enables better control of the additive contribution of the current layer and the residual version did not perform better in our early experiments. We leave a further investigation of residuals to replace the gated skip connections for future work.

A.2.1 A NOTE ON INTERCHANGEABILITY OF RELATIONS

Note that the simple choice of adding a relation vector, similarly to CompGCN's subtraction version, might pose problems since it loses information about which relation was associated with which neighbour:

$$(\mathbf{h}_u + \mathbf{a}_r) + (\mathbf{h}_w + \mathbf{a}_r) = (\mathbf{h}_w + \mathbf{a}_r) + (\mathbf{h}_u + \mathbf{a}_r) \ . \tag{22}$$

This is undesireable since it limits the expressive power of the network when using a sum aggregator. It is not clear to us how much this would affect models using attention-based aggregation, however, we prefer to avoid this since the attention-based aggregator may behave like a uniform sum, especially throughout the early stages of training.

This leads to the following condition that we impose on the message function: The message function μ must be chosen such that

$$\mu(\mathbf{a}, b) + \mu(\mathbf{c}, d) \neq \mu(\mathbf{a}, d) + \mu(\mathbf{c}, b) \tag{23}$$

iff $a \neq c$ and $b \neq d$.

The gated message function μ_{GCM} proposed earlier in our work satisfies this condition.

A.3 BACKPROPAGATION TO DISTANT NODES IN GATED GRAPH NEURAL NETWORKS

The GGNN (Li et al., 2015) implements the update function $\phi(\cdot)$ based on Gated Recurrent Units (GRUs) (Cho et al.):

$$\mathbf{r}^{(k)} = \sigma(W_r \overline{\mathbf{h}}_v^{(k-1)} + U_r \mathbf{h}_v^{(k-1)} + b_r) \tag{24}$$

$$\mathbf{z}^{(k)} = \sigma(W_z \overline{\mathbf{h}}_v^{(k-1)} + U_z \mathbf{h}_v^{(k-1)} + b_z) \tag{25}$$

$$\hat{\mathbf{h}}_v^{(k)} = \tanh(W \overline{\mathbf{h}}_v^{(k-1)} + U(\mathbf{h}_v^{(k-1)} \odot \mathbf{r}^{(k)})) \tag{26}$$

$$\mathbf{h}_v^{(k)} = (1 - \mathbf{z}^{(k)}) \odot \mathbf{h}_v^{(k-1)} + \mathbf{z}^{(k)} \odot \hat{\mathbf{h}}_v^{(k)} \quad , \tag{27}$$

where $\overline{\mathbf{h}}_v^{(k-1)} = \gamma(\{\mu(\mathbf{h}_u^{(k-1)}, r)\}_{(u,v,r) \in \mathcal{E}(\cdot,v)})$ is the vector representing the aggregated neighbourhood of node v.

Consider a node classification task, where the top-level state $\mathbf{h}_v^{(K)}$ of node v is fed into a classifier that produces a loss $\mathcal{L}_v$. To compute the gradient $\nabla_{\mathbf{w}} \mathcal{L}_v = \frac{\partial \mathcal{L}_v}{\partial \mathbf{w}}$, where $\mathbf{w}$ are the parameters of the model, we apply the chain rule as follows (here we ignore other nodes and only take into account only node v's contribution to the gradient):

$$\nabla_{\mathbf{w}} \mathcal{L}_v = \frac{\partial \mathcal{L}_v}{\partial \mathbf{w}} = \sum_{k=0}^{K} \frac{\partial \mathcal{L}_v}{\partial \mathbf{h}_v^{(k)}} \frac{\partial \mathbf{h}_v^{(k)}}{\partial \mathbf{w}} \tag{28}$$

with

$$\frac{\partial \mathcal{L}_v}{\partial \mathbf{h}_v^{(k)}} = \frac{\partial \mathcal{L}_v}{\partial \mathbf{h}_v^{(K)}} \prod_{j=k+1}^{K} \frac{\partial \mathbf{h}_v^{(k)}}{\partial \mathbf{h}_v^{(k-1)}} \tag{29}$$

Given the gated update equations of the GRU, the partial derivatives $\frac{\partial \mathbf{h}_v^{(k)}}{\partial \mathbf{h}_v^{(k-1)}}$ of the updated node representation w.r.t. the previous node representation are:

$$\frac{\partial \mathbf{h}_v^{(k)}}{\partial \mathbf{h}_v^{(k-1)}} = (1 - \mathbf{z}) \odot \mathbf{I} + \mathbf{z} \odot \frac{\partial \hat{\mathbf{h}}_v^{(k)}}{\partial \mathbf{h}_v^{(k-1)}} + \frac{\partial (-\mathbf{z})}{\partial \mathbf{h}_v^{(k-1)}} \odot \mathbf{h}_v^{(k-1)} + \frac{\partial \mathbf{z}}{\partial \mathbf{h}_v^{(k-1)}} \odot \hat{\mathbf{h}}_v^{(k)} \tag{30}$$

The closer $\mathbf{z}$ is to a zero vector, the closer the partial derivatives across layers $\frac{\partial \mathbf{h}_v^{(k)}}{\partial \mathbf{h}_v^{(k-1)}}$ get to the identity matrix $\mathbf{I}$ and the more the following holds:

$$\frac{\partial \mathcal{L}_v}{\partial \mathbf{h}_v^{(k)}} \approx \frac{\partial \mathcal{L}_v}{\partial \mathbf{h}_v^{(K)}} \quad , \tag{31}$$

which has the effect that all the GGNN layers are skipped. In practice, the gate $\mathbf{z}$ will take on different values throughout training, but the additive update still allows to retain some contribution of the original gradient $\frac{\partial \mathcal{L}_v}{\partial \mathbf{h}_v^{(K)}}$ deeper into the network.

On the other hand, consider a (K-1)-layer GGNN applied to a graph representing a sequence of length K. The sequence is represented as a graph according to the same rules as described in Section 4.1: the nodes are connected using only `next`-edges. Thus, only left-to-right propagation in the graph can be performed, and every node has only one incoming message, except the first, which has none. This is the setup that maximizes the ratio of graph diameter per number of nodes and edges.

When a (K-1)-layer GGNN is applied to such a graph of K nodes, there is only one backpropagation path from node K to node 0: $\frac{\partial \mathbf{h}_K^{(K)}}{\partial \mathbf{h}_0^{(0)}}$, which is also the longest backpropagation path possible in this

graph:

$$\frac{\partial \mathbf{h}_K^{(K)}}{\partial \mathbf{h}_0^{(0)}} = \prod_{k=1}^{K} \frac{\partial \mathbf{h}_k^{(k)}}{\partial \mathbf{h}_{k-1}^{(k-1)}} \tag{32}$$

Ignoring the used message function μ_{MM} and considering every node has only one neighbour, the neighbourhood aggregation vector is simply the neighbour node vector: $\overline{\mathbf{h}}_v^{(k-1)} = \mathbf{h}_{k-1}^{(k-1)}$.

Plugging this into the GGNN equations (Eqs. 24-27) gives:

$$\frac{\partial \mathbf{h}_k^{(k)}}{\partial \mathbf{h}_{k-1}^{(k-1)}} = \frac{\partial \mathbf{z}^{(k)} \tanh(W\mathbf{h}_{k-1}^{(k-1)})}{\partial \mathbf{h}_{k-1}^{(k-1)}} \tag{33}$$

$$= \tanh(W\mathbf{h}_{k-1}^{(k-1)}) \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{h}_{k-1}^{(k-1)}} + \mathbf{z}^{(k)} \frac{\partial \tanh(W\mathbf{h}_{k-1}^{(k-1)})}{\partial \mathbf{h}_{k-1}^{(k-1)}} \tag{34}$$

$$= \tanh(W\mathbf{h}_{k-1}^{(k-1)}) \frac{\partial \sigma(W_z\mathbf{h}_{k-1}^{(k-1)})}{\partial \mathbf{h}_{k-1}^{(k-1)}} + \mathbf{z}^{(k)} \frac{\partial \tanh(W\mathbf{h}_{k-1}^{(k-1)})}{\partial \mathbf{h}_{k-1}^{(k-1)}} \tag{35}$$

Both terms contain a partial derivative of the form $\frac{\partial \sigma(W\mathbf{h}_{k-1}^{(k-1)})}{\partial \mathbf{h}_{k-1}^{(k-1)}}$, which is essentially identical to the terms found in the backpropagation equations for a vanilla RNN.

As in vanilla RNNs (without any gating like in GRUs and LSTMs), these terms can cause vanishing gradients due to repeated multiplication with the derivative of a σ or $\tanh$ nonlinearity and vanishing or exploding gradients depending on the values of W (which also change throughout training and thus could potentially reach unstable values).

In addition to the gradient-based argument outlined above, residual networks (as well as GRUs and LSTMs), benefit from better generalization abilities due to effects which were not formalized or deeply studied in the literature known to us. Intuitively, the argument for residual networks (He et al., 2016) is that the individual residual blocks can easily learn to *not* contribute if necessary, which enables to learn very deep architectures, and otherwise, they only need to model the function that models the *residual* error computed from the output of the previous layer. In addition, we believe that adding such skip connections enables more effective weight sharing across layers and nodes.

A.4 Conditional Recall Experimental Details:

In our experiments for this task, we randomly explore the following hyperparameter values for all compared methods: (1) dimensionality of node feature vectors in $[150, 300]$ (larger values can cause memory issues with the baselines), (2) dropout in $[0., 0.25, 0.5]$, (3) we set the dropout of embedding vectors to 0.1 (4) learning rate in $[0.001, 0.0005, 0.0001]$. For the baselines, we run for at least 200 epochs. For our GR-GAT, we reach high accuracy in 20 epochs and thus run only for a maximum of 50 epochs. Throughout all of the experiments, we use the Adam (Kingma & Ba, 2015) optimizer. We also use label smoothing with a factor 0.1 everywhere. For our experiments with task length 15, we use the best hyperparameters found for length 10 for all methods and also experiment with smaller dropout values.

For the final reported accuracies, we trained with the best hyperparameters and loaded the model with the best validation error and evaluated on the test set. This was repeated with three different seeds that were shared between the tested methods.

The initial node states are initialized by embedding the node type using a low-dimensional embedding matrix (dimensionality 20; for a vocabulary of 62 characters) and projecting the low-dimensional embeddings to the node state dimension using a learned transformation.

An example of a graph used in this task is given in Figure 4. The output in this example should be "8".

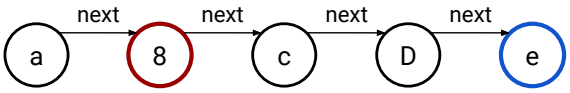

Figure 4: Example of input graph for the Conditional Recall task. The top-level state of the blue node is used for prediction. The red node specifies the desired output.

A.5 TREE MAX EXAMPLE

See Figure 5 on page 16.

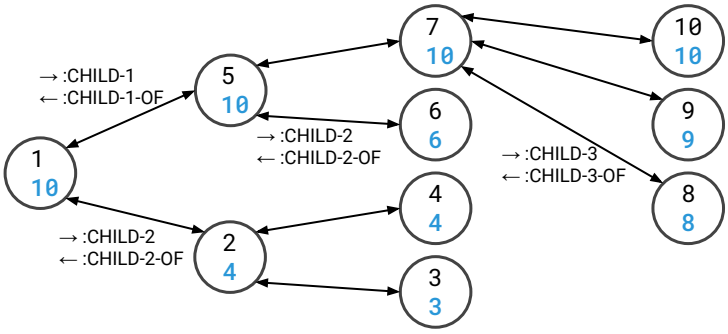

Figure 5: Example of input graph for the Tree Max task. The double-ended arrow between a node v and a node u represents two edges: one going from v to u and the other from u to v. Not every arrow is labeled for a clearer presentation. The labels on the arrows indicate edge labels: the forward arrow $\rightarrow$ corresponds to the edge label for going from the parent to the child and $\leftarrow$ for the reverse. The input labels are the black numbers and the output labels are the blue numbers.

A.6 TREE MAX EXPERIMENTAL DETAILS:

During data generation, we first randomly pick a tree depth between five and fifteen. Then, we generate a tree, choosing between 0, 2 or 3 children for each node until we reach the chosen tree depth. The largest trees in our generated data contained over 200 nodes. The generated dataset contained 800 examples; we used a 50/25/25 train/validation/test split.

For every model tested, we perform a hyperparameter search using a fixed random seed. For all models except our RGAT baseline, we used seventeen layers. Then we run the best hyperparameter setting with five different random seeds and report the test results in Table 2. [5]

We evaluated the trained models based on node-wise and graph-wise accuracy[6].

We use a seventeen-layer networks unless otherwise indicated. Note that in our experiments, we use early stopping based on node-level classification accuracy on the validation set with patience set to ten and the minimum number of training epochs set to twenty. We reload the best model weights after training finishes and evaluate on the test set.

In our experiments, we randomly explore different combinations of different hyperparameter settings: dropout is selected from $\{0., 0.1, 0.25, 0.5\}$, dimensionality of node vectors is selected from $\{150, 300\}$. For RGAT, we experiment only with dimensionality of 100 due to memory problems. Learning rate is chosen from $\{0.001, 0.00075, 0.0005, 0.0001\}$. Since our *GR-GAT (SGRU) - edge matrix* ablation model showed unstable training behaviour, we used a learning rate of 0.00025.

[5]Note that the same five seeds are re-used for experiments for all models, and that every seed results in a different dataset being generated.

[6]The graph-level accuracy is 100% for an example only if all nodes in the graph have been classified correctly, and is 0% otherwise

For RGCN, we explore the following numbers of layers/steps: [4, 7, 12, 17]. For our RGAT baseline, we use [5, 7, 12, 17]. For our GGNN baseline, we use [7, 12, 17] Note that when the number of layers/steps is lower than 14, we can not achieve 100% accuracy on the task.

We perform a hyperparameter search using the validation portion and re-train on the entire training set (using the best configuration) with five different seeds and report the results in Table 2.

A.7 Entity Classification Experimental Details

We use the datasets as provided at `https://github.com/tkipf/relational-gcn`, removing some code for more memory efficient storage. The datasets contain graphs, where only a fraction of nodes carries a label that we should predict (we call these nodes labeled nodes). We train on the train portion as provided on the repository and test on the predefined test portion.

During experiments, we train with two different settings: (1) a fixed two-hop (subsampled) subgraph and (2) a fixed subsampled random walk subgraph.

In the first setting (two-hop subgraph), we retrieve for every labeled node that will be classified a two-hop, at most 6000 nodes (and 10000 edges) subgraph. During batching, these subgraphs are merged. Note that if the number of nodes (or edges) exceeds the maximum number of nodes (6000), we stop expanding. We experiments with the following hyperparameter ranges: (1) dropout from 0.25, 0.5, (2) embedding dropout from 0.25, 0.5, (3) dimension from 64, 96, (4) learning rate from 0.005, 0.001, 0.00025. We use at least 50 epochs and early stopping with patience set to 10 epochs to terminate validation runs earlier and save computation time. The number of heads in our GR-GAT method is set to four for all tried dimensions. During hyperparameter search, we run a random selection of all possible combinations with three different seeds, using a subset of the training set (80%) for training and another set (20%) for validation. After determining the best hyperparameters, we use them to train a final model on the entire training set and evaluate on the test set. The reported numbers use three different seeds.

In the second setting (random walk subgraph), we use random walks to generate the subgraph. For every labeled node, we precompute a subgraph of at most 2000 nodes and 3000 edges, using 5% reset probability in the random walk. During batching, these subgraphs are merged. Note that the subgraphs are precomputed once before training and fixed throughout, so the subgraphs do not get resampled at every epoch. We use the same set of seeds for the different methods we test to ensure that the methods have been tested on the same sets of data. We experiments with the following hyperparameter ranges: (1) dropout from 0., 0.1, 0.25, 0.5, (2) embedding dropout from 0., 0.1, 0.25, (3) dimension from 32, 64, (4) learning rate from 0.01, 0.005, 0.001, 0.00025. We use at least 50 epochs and early stopping with patience set to 10 epochs to terminate validation runs earlier and save computation time. The number of heads in our GR-GAT method is set to 4 for all tried dimensions. Since GR-GAT (Ident) reached near 100% training accuracy with the chosen hyperparameters after 15 epochs, we ran it for 20 epochs during final tests. During hyperparameter search, we randomly run all possible combinations with three different seeds, using a subset of the training set (80%) for training and another set (20%) for validation. After determining the best hyperparameters, we use them to train a final model on the entire training set and evaluate on the test set.

In both settings, we erase the initial features (embeddings) of the nodes for which we make predictions by setting them to the same learnable vector that is reused everywhere.