Improving Breadth-Wise Backpropagation in Graph Neural Networks Helps Learning Long-Range Dependencies

Denis Lukovnikov [1] **Asja Fischer** [1]

Abstract

In this work, we focus on the ability of graph neural networks (GNNs) to learn long-range patterns in graphs with edge features. Learning patterns that involve longer paths in the graph, requires using deeper GNNs. However, GNNs suffer from a drop in performance with increasing network depth. To improve the performance of deeper GNNs, previous works have investigated normalization techniques and various types of skip connections. While they are designed to improve *depth-wise* backpropagation between the representations of the same node in successive layers, they do not improve *breadth-wise* backpropagation between representations of neighbouring nodes. To analyse the consequences, we design synthetic datasets serving as a testbed for the ability of GNNs to learn long-range patterns. Our analysis shows that several commonly used GNN variants with only depth-wise skip connections indeed have problems learning long-range patterns. They are clearly outperformed by an attention-based GNN architecture that we propose for improving both depth- and breadth-wise backpropagation. We also verify that the presented architecture is competitive on real-world data.

1. Introduction

The ability of graph neural networks (GNNs) to capture long-range dependencies in graphs with edge features heavily depends on their depth: at least K GNN layers are needed to capture information that is K hops away. However, GNNs suffer from decreasing performance when the number of layers is increased. Recently, several works have identified and investigated different possible causes of this decrease in performance: (1) over-fitting (Vashishth et al., 2020), (2) over-smoothing (Li et al., 2018; Chen et al., 2019; Zhao & Akoglu, 2020; Rong et al., 2019; Yang et al., 2020), (3) over-squashing (Alon & Yahav, 2020), and (4) possible vanishing gradient (Hochreiter & Schmidhuber, 1997; Pascanu et al., 2013; He et al., 2016) problems (Li et al., 2019; 2020; Rahimi et al., 2018).

Our contributions in this paper are threefold. First, we identify design choices of GNN architectures that may *limit their ability to learn long-range patterns* (Section 3). A simple but effective approach to avoid vanishing gradients and to reduce over-smoothing is the use of skip connections, as for example implemented in Residual GCNs (RGCNs) (Li et al., 2019), Jumping Knowledge Networks (Xu et al., 2018), and Gated Graph Neural Networks (GGNN) (Li et al., 2015). However, this approach focuses on improving backpropagation only depth-wise, that is between representations of the same node in successive layers. However, as we shall discuss, *they do not improve breadth-wise backpropagation*, that is between representations of one node to the representations of its neighbours in the previous layer. In addition, we point out that the commonly used uniformly weighted sum and max/min pooling aggregators can lead to *exponentially decaying gradients*, which is related to over-squashing. Finally, we argue that some commonly used techniques to take into account edge types or other edge features may also lead to poor backpropagation behavior, along with over-parameterization.

Second, in Section 4, we develop a GNN architecture for predictions on graphs with edge features and multi-relational graphs, with a focus on avoiding the identified problems. The presented architecture relies on attention-based aggregation to reduce exponential gradient decay, and ensures effective breadth-wise backpropagation by adopting breadth-wise residual connections. The resulting architecture is similar to Transformers (Vaswani et al., 2017) but simpler while allowing to seamlessly integrate edge features.

Third, we propose two synthetic datasets that challenge the ability of GNNs in learning long-range patterns. The first task is designed to measure the ability of GNNs to learn simple rules over multiple hops on a simple chain while the second tests the models on semi-supervised node classification on trees. We use these datasets to perform

[1]Ruhr University Bochum, Bochum, Germany. Correspondence to: Denis Lukovnikov <denis.lukovnikov@rub.de>, Asja Fischer <asja.fischer@rub.de>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

an empirical analysis of commonly used GNNs and the proposed architecture. In addition, we perform experiments on the ZINC and OGBG-CODE2 datasets to study real-world performance.

We hope that the presented work draws attention to the importance of breadth-wise backpropagation in GNNs and will facilitate the development of better architectures.

2. Background: Message Passing Networks

Many popular GNNs can be formulated within a message passing (MP) framework (Gilmer et al., 2017). Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with node set $\mathcal{V}$ and a set of directed edges $\mathcal{E}$, where an edge from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ with a feature vector $\mathbf{r}$ is specified by a triple $(u, v, \mathbf{r})$. Note that the feature vector $\mathbf{r}$ may contain only one element specifying an edge type r, in which case the triple specifying the edge can also be written as (u, v, r), and that it is not specified in the case of GCNs without edge features.

A GNN maps each node $v \in \mathcal{V}$ onto representation vectors $\mathbf{h}_v^1, \ldots, \mathbf{h}_v^K$ 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 the K layers of the GNN. In the encoding process, relational GNNs also take into account the edge types between the nodes in the graph $\mathcal{G}$. In the following sections, we work with the message passing framework where a single GNN layer/step is decomposed into a three-step process:

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

where $\mu(\cdot)$ computes a "message" along a graph edge $(u, v, \mathbf{r})$ depending on the neighbour's representation $\mathbf{h}_u^{(k-1)}$ and the edge features $\mathbf{r}$, $\gamma(\cdot)$ aggregates the incoming messages into a single vector, and $\phi(\cdot)$ computes a new representation for node v based on the aggregated messages and it's previous state. $\mathcal{E}(\cdot, v)$ denotes the set of all edges in $\mathcal{G}$ that end in v. After subsequently applying Eq. 1 K times 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 common message function which is for example used by RGCNs and GGNNs is a relation-specific linear transformation $\mu_{\text{MM}}(\mathbf{h}_u, \mathbf{r}) = \mathbf{W}_r\mathbf{h}_u$, where $\mathbf{W}_r$ is a $\mathbb{R}^{D\times D}$ parameter matrix associated with the edge type r specified in $\mathbf{r}$. The GGNN implements the update function $\phi(\cdot)$ as a gated recurrent unit (GRU) (Cho et al., 2014): $\mathbf{h}_v^{(k)} = \text{GRU}(\overline{\mathbf{h}}_v^{(k-1)}, \mathbf{h}_v^{(k-1)})$, where the GRU's *input* argument is $\overline{\mathbf{h}}_v^{(k-1)} = \gamma(\{\mu(\mathbf{h}_u^{(k-1)}, \mathbf{r})\}_{(u,v,\mathbf{r})\in\mathcal{E}(\cdot,v)})$, the vector for the aggregated neighbourhood of v. Some examples of aggregation functions are a (normalized) sum, a pooling operator (such as elementwise maximum), and attention (Veličković et al., 2018).

3. Motivation: Breadth-wise Learning

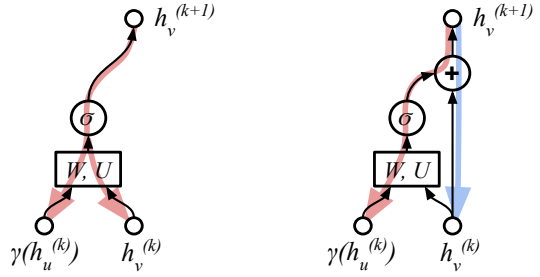
Many approaches have recently been proposed to mitigate vanishing gradients and oversmoothing in GNNs, using techniques such as residual connections (Li et al., 2019; 2020), gated skip connections (Li et al., 2015) and "jumping knowledge" connections (Xu et al., 2018), However, these techniques address only backpropagation in the depth of the network (i.e. vertically towards lower-level features of the same node). As we will now discuss, this is not sufficient for learning long-range patterns since breadth-wise backpropagation (i.e. horizontally towards the lower level features of the neighbours) in such GNNs may still result in learning problems. Another potential contributor to breadth-wise backpropagation problems is the *exponential gradient decay* associated with some commonly used aggregators. In addition, the presence of edge features (1) requires additional model extensions, which may exacerbate the issue, and (2) may prevent us from using dilated convolutions or similar techniques (Li et al., 2019; Abu-El-Haija et al., 2019; Klicpera et al., 2019).

3.1. Depth-wise backpropagation

Consider a simple (R)GCN [1] of the general form $\mathbf{h}_v^{(k)} = \sigma(\mathbf{W}^{(k)}\mathbf{h}_v^{(k-1)} + \sum_{(u,v)\in\mathcal{E}} \mathbf{W}^{(k)}\mathbf{h}_u^{(k-1)}))$, where σ is a non-linear activation function. The classic argument points out that functions of this form will suffer from vanishing or exploding gradients because of the stacking of linear transformations ($\mathbf{W}$) and non-linearities (σ). The backpropagation path from the top-level features $\mathbf{h}_v^{(K)}$ of a node to its initial features $\mathbf{h}_v^{(0)}$ involves the derivative of $\sigma(\mathbf{W}^{(K)}(\sigma(\mathbf{W}^{(K-1)}\ldots(\sigma(\mathbf{W}^{(1)}\mathbf{h}_v^{(0)})\ldots)))$. With many layers (i.e., in the case that K is large), the gradient magnitude might diminish at every step because of the multiplication with the derivative of the activation function. Additionally, depending on the values of the weights $\mathbf{W}$, multiplication with $\mathbf{W}$ may cause the gradients to either vanish or explode. This can become especially problematic when sharing weights between layers, which is one strategy to combat over-parameterization in deeper networks.

As shown for recurrent and feed forward neural networks, these problems can be mitigated by the choice of activation function (e.g. a ReLU), by using good initializations for the weights and by using normalization layers (Ioffe & Szegedy, 2015; Ba et al., 2016). LSTMs (Hochreiter & Schmidhuber, 1997), GRUs, ResNets (He et al., 2016), Highway Networks (Srivastava et al., 2015), and DenseNets (Huang et al., 2017) offer alternative solutions by introducing some form of skip connections. This ensures that the gradient to deeper layers does not vanish, and also that the gradient

[1] Note that while we use the this simple form of a GCN for clarity, the presented discussion also fully applies for RGCNs.



(a) Without skip connections. (b) With skip connections.

Figure 1. Effect of skip connections.

w.r.t. features (or states) in different layers (or timesteps) is similar. The latter, we believe, could lead to more effective weight sharing between nodes and layers. He et al. (2016) point out that the improvement they obtain is not due to gradient magnitudes but explained by the fact that modeling the residual error is an easier task for the network.

These approaches can also be applied to improve training of GCNs. For example, the residual GCN (ResGCN) (Li et al., 2019), introduces skip-connections into the update equation of simple GCNs: $\mathbf{h}_v^{(k)} = \mathbf{h}_v^{(k-1)} + \sigma(\mathbf{W}^{(k)}(\mathbf{h}_v^{(k-1)} + \sum_{(u,v)\in\mathcal{E}} \mathbf{h}_u^{(k)}))$. Transformers also employ skip connections both to skip over the attention as well as the 2-layer MLP. GGNNs and MPNNs (Gilmer et al., 2017) use a GRU-based update function that implements a gated connection to previous node features.

3.2. Breadth-wise backpropagation

While the techniques described above improve depth-wise backpropagation, they do not address breadth-wise backpropagation (i.e., backpropagation towards the neighbouring nodes). For an example let us consider a single layer of an ResCGN illustrated in Fig. 1b, where blue arrow indicates the depth-wise backpropagation path towards previous node features of the same node, and the red arrow the breadth-wise path towards previous node features of the neighbours. It is clear that while the blue path preserves the gradient, the red one suffers from the same problems as in the case without the skip connection (shown in Fig. 1a). Therefore, performing backpropagation from the top-level features $\mathbf{h}_v^{(K)}$ of a node v to the $(K-L)$-level features $\mathbf{h}_u^{(K-L)}$ of a node u that is L hops away involves the derivative of $\sigma(\mathbf{W}^{(K)}(\sigma(\mathbf{W}^{(K-1)}\dots(\sigma(\mathbf{W}^{(K-L)}\mathbf{h}_u^{(K-L)})\dots)))$, and therefore may suffer from poorly behaving gradients, especially when using weight-sharing in a deep network. A similar argument can be provided for the GGNN (see Appendix E), which connects $\mathbf{h}_v^k$ with $\mathbf{h}_v^{k-1}$ using a GRU, since there the previous representations of neighbouring states are fed through non-linear gates before they are integrated in $\mathbf{h}_v^k$.

While proper initialization, the use of normalization layers, and non-saturating nonlinearities may maintain good gradient magnitudes, the training of the network can still be improved by implementing some form of breadth-wise skip connections. In addition, as we will experimentally show, the use of *only depth-wise* skip connections can actually *hurt* performance when the task requires learning patterns containing distant nodes.

Exponential gradient decay in aggregation. Note that the breadth-wise gradient issues are further exacerbated when aggregating incoming messages by averaging [2], which diminishes the gradient towards a neighbour by a factor equal to the number of incoming edges $\text{in\_deg}(n)$. This leads to a gradient magnitude that is exponentially decaying in the path distance between nodes and can quickly vanish when having many high-degree nodes on the path. This drop is also related to the issues of oversmoothing and over-squashing (Alon & Yahav, 2020). Also note that with max-pooling, only an average $1/\text{in\_deg}(n)$ fraction of the neurons has a non-zero gradient in the initial stages of training. It is not clear how max-pooling evolves throughout training, which we leave as a question for future work. Attention-based aggregation is similar to mean aggregation in the initial stages of training but can evolve to assign a weight close to one to a neighbouring node. When this happens, the gradient magnitude w.r.t. the attended neighbour is not as badly affected as when using a mean. In other words, attention-based aggregation can evolve to improve backpropagation to the selected nodes.

Using edge features. Different ways of embedding relations and incorporating edge features have been proposed. The message function μ_{MM} of RGCNs and GGNNs is given by a multiplication with a relation-specific matrix. CompGCN (Vashishth et al., 2020) and VR-GCN (Ye et al., 2019) make use of vector-based relation parameterization, and explore composition functions inspired by graph embedding learning methods (Bordes et al., 2013; Yang et al., 2014; Nickel et al., 2016). Edge features are important for many GNNs applications, but can pose additional challenges.

Firstly, the function used to incorporate edge features could hinder learning long-range patterns. For example, μ_{MM} (defined earlier in Section 2), which is used by several GNNs (Li et al., 2015; Schlichtkrull et al., 2018; Brockschmidt, 2019), performs a linear transformation and may lead to badly behaving gradients due to repeated multiplication of the transformation matrices in a path to a distant node. Note that this can also be the case for GNNs with vector-parameterized relations, such as CompGCN, whose

[2] Frequently, mean aggregation is used instead of a simple sum in order to stabilize training by normalizing.

message function is $\mu_{\mathrm{CompGCN}-\psi} = W_\lambda \psi(\mathbf{h}_u, \mathbf{a}_r)$, where ψ is a composition function (e.g. elementwise difference or product) and $\mathbf{a}_r$ is the embedding vector corresponding to the relation r.

Secondly, the way of representing relations may lead to over-parameterization. For example, μ_{MM} defines a trainable matrix for every relation, leading to $\mathcal{O}(d^2)$ parameter complexity. To resolve this, Schlichtkrull et al. (2018) proposes decomposing the relation matrices. Here, the vector-based relation parameterizations used by CompGCNs and VR-GCNs are advatigous, since they have $\mathcal{O}(d)$ parameter complexity and can still be decomposed.

Finally, it is not clear whether and how dilated graph convolutions (Li et al., 2019) and similar techniques (Abu-El-Haija et al., 2019; Klicpera et al., 2019) can be used with edge features. These techniques introduce new edges between non-neighbouring nodes if they are close in a graph (e.g. a path of certain length exists between them). This reduces the number of message passing layers necessary to reach a distant node. However, it is unclear whether and how to efficiently take into account the edge features present on such paths, and what kind of additional modeling limitations such techniques could have.

4. Residual Graph Attention Networks

To demonstrate that the ability to model long-range patterns indeed is improved when taking the issues discussed in the previous section into account, we develop a GNN architecture that addresses both the breadth-wise backpropagation issue as well as the problem of exponential breadth-wise gradient decay. The proposed architecture, that we refer to as Residual Relation Graph Attention Network (ResR-GAT), follows the general principle of using additive, skip-connected feature updates both depth-wise (vertically) and breadth-wise (horizontally).

In summary, the proposed model consists of (1) an attention-based neighbourhood aggregation mechanism similar to that used by Graph Attention Networks (GATs) (Veličković et al., 2018) and transformers (Vaswani et al., 2017) and (2) a residual message function that takes into account the edge features. Node-wise updates are optional. The resulting model is relatively simple. While it is similar to transformers, it has fewer parameters because some parts of the transformer architecture are omitted. In the following subsections, we discuss the different parts of the architecture in more detail.

4.1. Attention-based neighbourhood aggregation

As elaborated in Section 3.2, message aggregation by averaging can lead to decaying gradient magnitudes w.r.t. distant nodes. In addition, a simple, uniformly weighted sum aggregation 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 contributing distant nodes (Alon & Yahav, 2020). Since attention can learn to use a high weight for specific nodes, it can evolve to decrease the effect of exponential gradient decay towards relevant nodes. For this reason, we believe that using an attention mechanism is crucial to enable the learning of long-range patterns in graphs.

To implement the aggregation function $\gamma(\cdot)$, we adapt the scaled multi-head multiplicative attention mechanism of transformers. The per-head attention distributions are computed as described for transformers, but with two small differences: (1) the edge type embeddings are used directly in the keys when computing the attention, and (2) the linear transformations normally used to obtain value vectors are omitted. The latter follows the principle of using only additive operations to enable better long-range backpropagation through messages and node features. Compared to the transformer's attention block, the post-attention linear transformation and the residual connection skipping over the attention mechanism are omitted. Note that in order to take into account the lower-level features of a node when computing its new features, the model requires self-edges that connect each node with itself. See Appendix A for a more detailed description of the used attention mechanism.

4.2. Residual message function with edge features

To preserve the gradient in the message function, we choose an additive function. A simple choice of vector addition like μ_{Add} has the problem that under a non-weighted sum aggregator, the relation vectors are interchangeable, which limits the expressive power of the network. This leads us to the additional requirement of non-interchangeability of relations or edge features: $\forall \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \in \mathbb{R}^d : \mathbf{a} \neq \mathbf{c} \wedge \mathbf{b} \neq \mathbf{d} \iff \mu(\mathbf{a}, \mathbf{b}) + \mu(\mathbf{c}, \mathbf{d}) \neq \mu(\mathbf{a}, \mathbf{d}) + \mu(\mathbf{c}, \mathbf{b})$. The third requirement is vector-based parameterization of relation types to combat over-parameterization like in RGCN and GGNN. This also allows to easily replace the relation vectors with edge feature vectors without additional model changes.

We use the following two-layer residual message function, which fulfills these requirements and can model complex interactions between node and edge features:

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

$$\mathbf{u} = \mathbf{b}_\mathrm{B} + \mathbf{W}_\mathrm{B}\mathbf{z} \ , \tag{3}$$

$$\mu_{\mathrm{Res}}(\mathbf{h}_u, r) = \mathbf{h}_u + \mathbf{u} \ , \tag{4}$$

where $\mathbf{a}_r$ is an embedding vector associated with the edge type from u to v, and $\mathbf{W}_\mathrm{A}$ and $\mathbf{W}_\mathrm{B}$ are trainable weight matrices (with corresponding biases $\mathbf{b}_\mathrm{A}$ and $\mathbf{b}_\mathrm{B}$) that are

shared between different edge types. CELU (Barron, 2017) is used as the activation function. See also Appendix D.

4.3. Node update function

We experiment with using (1) no update function, i.e. the identity mapping $\phi_{\text{Id}}(\overline{\mathbf{h}}_v^{(k-1)}, \mathbf{h}_v^{(k-1)}) = \overline{\mathbf{h}}_v^{(k-1)}$ and (2) a gated unit which we refer to as Symmetrically Gated Recurrent Unit (SGRU). The SGRU is a modified GRU designed to improve breadth-wise backpropagation when used in a GNN by gating the input similarly to the current node state. See Appendix B for an elaborate explanation of the SGRU. Note that in both cases, we always ensure the node's lower-level features $\mathbf{h}_v^{k-1}$ are included among the incoming messages by using a self-edge of a special type.

4.4. Relation to transformers

The resulting architecture is similar to transformers and can be seen as a relation-aware adaptation of the transformer architecture for graphs. We therefore consider a transformer-based GNN architecture as one of the baselines in our experiments. A transformer can be formulated as a GNN on a fully connected graph as follows: (1) the attention mechanism is the neighbourhood aggregator and (2) the 2-layer MLP is the node-wise update function. We use the following message function in this baseline:

$$\mu_{\text{AddReLU}} = \text{LeakyReLU}(\mathbf{h}_u + \mathbf{a}_r) \ . \tag{5}$$

This is similar to Shaw et al. (2018)'s work on relative position encoding in transformers with the difference of the leaky ReLU activation function, which alleviates the issue of interchangeable relations mentioned previously and at the same time is non-saturating and thus helps to preserve gradient magnitude. Compared to this transformer-based GNN baseline, the proposed ResRGAT moves the 2-layer MLP from the nodes to the edges, using it as the message function, and has a slightly different attention mechanism, as described in Section 4.1. It has fewer parameters and a simpler form than the transformer while enabling better long-range pattern learning in graphs. While these changes are not necessary for transformers in normal settings because of their depth-wise residual connections and fully-connected graphs, we hope that our work will be useful for future work with sparse and/or relation-aware transformers (Child et al., 2019; Beltagy et al., 2020; Shaw et al., 2018).

4.5. Graphs without edge features

The presented approach can be adapted to graphs without edge features. One possibility is adding self-edges and using "neighbour" and "self" relation types for neighbour and self-connections respectively. Alternatively, if self-edges are not used, it is necessary to use a node update function that also uses the previous node state. Neighbourhood dilation schemes can also be easily integrated. For example, a MixHop (Abu-El-Haija et al., 2019) variant can be implemented that uses different edge vectors for different hop distances. We leave the investigation of these modifications for future work.

5. Experimental Analysis

In this experimental analysis[3], we (1) challenge the ability of existing GNNs to model long-range patterns in graphs with edge features, (2) investigate how well the proposed method and its ablations work on the same tasks and (3) confirm that the proposed method is competitive with recent work on real-world tasks.

We test the following GNNs: (1) RGCN, (2) GGNN, (3) Relational Graph Attention Network (RGAT) (Busbridge et al., 2019), (4) GatedGCN (Bresson & Laurent, 2017; Dwivedi et al., 2020), and (5) a relational transformer "Rel. TM" (see Section 4.4), where we also experiment with weight sharing between all layers, which is indicated by "shared". RGCN is a variant of the Graph Convolutional Network (Kipf & Welling, 2016) that also models edge types (relations). Similarly to Schlichtkrull et al. (2018), in our implementation we use a mean aggregator across relation types. However, we do *not* decompose our relation matrices since we have only a small number of relations in our synthetic datasets. We implement our RGCN and GGNN baselines based on the code provided by the DGL framework. Note that in the GGNN, the parameters between layers are shared. We found several relational variants of the GAT in the literature (Busbridge et al., 2019; Brockschmidt, 2019; Sinha et al., 2019; Nathani et al., 2019) and chose to adapt the RGAT of Busbridge et al. (2019). In contrast to RGCN and GGNN, which aggregate using a (weighted) sum, RGAT computes the summation weights dynamically using attention, which ensure the weights sum up to one (see Appendix C for more details). GatedGCN implements a gating mechanism on the edges that is used to gate the transformerd neighbour states before aggregation. The gates are computed using a feedforward layer that takes the source and target node features, as well as the edge features of the previous layer, and then normalized using a softmax. While many other GNNs exist, we believe our selection reasonably covers a range of popular GNNs.

Weight sharing between all layers of ResRGAT is used in all synthetic experiments. The following ablations of our model (ResRGAT) are considered in the experiments: (1) "ResRGAT, val. transf." uses a linear layer to transform the value while using attention, like in transformers, (2) "ResRGAT, skip. att. " uses the skip connection to skip over

[3]Code is available at https://github.com/lukovnikov/resrgat.

the attention as in transformers, (3) "ResRGAT, no rel. cat. " does *not* concatenate the edge features directly in the key vector. Parameters are always shared between all layers of ResRGAT.

5.1. Conditional recall

In this experiment, we show that the baseline 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 deliberately design this task to measure how well the different GNNs can learn over a larger numbers of hops in a simple graph.

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 are 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". Twenty examples were randomly generated per output class for a total of 1220 examples and the data was split in 80/10/10 train/validation/test splits. See Appendix F for more details on the experimental setup.

The input sequences are transformed into graphs by (i) creating a node for every character of the sequence and (ii) adding edges labeled with the `next` type between every adjacent element in the sequence and the `self` type for self-edges linking the node to itself. 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. The number of layers L is always set to $N + 1$. The readout for prediction takes the representation of the last node in the sequence.

Results and discussion. The results for the different GNNs[4] and varying sequence lengths[5] N are shown in Table 1. While all existing GNNs that we tested reach acceptable accuracy for length 5, their accuracy severely suffers when the length is increased. In contrast, our model solves the task well for all lengths.

Surprisingly, methods using some form of skip connections (i.e. GGNN, Transformer, GatedGCN) severely underperform compared to the skip-less RGCN and RGAT. To verify that this is not only due to an increase in the number of layers, we modify the setup to verify that these models perform well in a deep (L=16) but short sequence (N=5) setup. Even though many layers are used, the task requires looking over at most 4 hops in the graph. As we can see from the results shown in last column of Table 1, the methods implementing depth-wise skip connections perform well while having the same number of layers as in the N=15 setting.

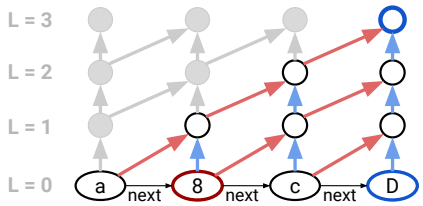


Figure 2. The input graph (ovals) for an example of the Conditional Recall task and the computational graph for 3 GNN layers. The final feature vector of the blue node is used for prediction and the red node specifies the desired output. Blue arrows are the depth-wise messages within the node and red arrows the breadth-wise messages to neighbours.

It is important to keep in mind that the computational graph of a GNN is different from that of an RNN applied to the same sequence. For RNNs, the "update distance" (i.e. the number of updates performed on the path) between the last state h_T of an RNN and an input token x_{T-k} (from k timesteps before) is k. In contrast, as illustrated in Figure 2, 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 case of an RGCN, the neighbours and the node itself are treated equally, while GGNNs provide a direct gated connection to previous node features, enabling easier depth-wise backpropagation. However, this does not improve backpropagation to distant nodes in the graph (see also Appendix E).

We also inspected the gradients w.r.t. the input vectors and experimented with different initializations for the edge type matrices of the baseline models. We found that the initialization indeed strongly affects the gradient magnitudes and also affects training behaviour. Because gradient magnitude w.r.t. the input doesn't appear to be the decisive factor, we believe that our proposed changes encourage more effective weight sharing across nodes and layers and provide a better training signal (He et al., 2016).

To investigate how the number of examples affects the baselines, we also trained the RGCN and the GGNN with 50 examples of length $N = 15$ per class (instead of 20). We found improvements for both baselines (the weight-shared RGCN reached 53.9 ± 4.3 %, the GGNN 48.7 ± 7.9 %.) but both still severely underperform compared to ResRGAT

[4]We also experimented with weight sharing for RGCN.

[5]All examples have the same, specified length in one experiment.

Table 1. Conditional recall results. Average accuracy on the test set is reported. N is sequence length. L is the number of layers.

	N=5	N=10	N=15	N=5 (L=16)
RGCN	91.8 ± 2.4	53.5 ± 4.9	6.3 ± 8.9	47.5 ± 1.8
GGNN	69.7 ± 3.3	2.5 ± 1.8	3.3 ± 2.4	80.1 ± 1.4
RGAT	91.0 ± 1.2	75.1 ± 3.2	30.2 ± 21.0	41.0 ± 3.3
GatedGCN	81.2 ± 4.0	5.5 ± 1.0	4.4 ± 1.0	80.3 ± 1.8
Rel. TM	96.2 ± 2.2	5.5 ± 1.0	4.6 ± 1.9	94.8 ± 0.7
Rel. TM (shared)	93.7 ± 2.4	6.3 ± 2.4	6.0 ± 1.5	96.2 ± 1.7
GRU	93.7 ± 2.5	9.3 ± 3.7	4.6 ± 1.9	81.1 ± 3.5
ResRGAT	$\mathbf{99.2 \pm 0.7}$	$\mathbf{97.8 \pm 1.0}$	$\mathbf{95.4 \pm 3.4}$	98.6 ± 1.4
ResRGAT+SGRU	97.8 ± 1.0	94.8 ± 2.2	92.4 ± 2.7	99.5 ± 0.4
ResRGAT, skip. att.	98.1 ± 1.4	94.0 ± 1.0	33.3 ± 1.7	99.2 ± 1.2
ResRGAT, val. transf.	98.1 ± 0.8	94.5 ± 3.3	82.8 ± 9.0	$\mathbf{99.7 \pm 0.4}$
SGRU	94.8 ± 2.3	89.9 ± 2.0	84.4 ± 0.7	92.9 ± 4.0

(even when it's trained with less data) and the results heavily depend on the chosen hyperparameters.

Finally, we also experimented with GNNs simply using a GRU or a SGRU as the node-wise update function. [6] As reported in Table 1, the SGRU performs better than the GRU, and is thus more suitable for use in GNNs.

5.2. Tree max

In this experiment, we use a semi-supervised node classification task to evaluate the ability of a GNN to learn patterns over a large number of hops for many nodes simultaneously. 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 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))))*. Labeling all nodes with their correct output labels results in: *(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 G for an illustration of the graph for this example.

A total of 800 examples are generated, each containing one tree with a randomly chosen depth between 5 and 17. A 50/25/25% training/validation/test split is used. The largest generated trees contained more than 200 nodes. Generally, 17-layer GNNs are used in the experiments. Instead of training and evaluating with the expected labels of all nodes, we choose a more challenging semi-supervised setting. The labels of nodes that are closer than half of the depth of the tree to nodes containing their expected output label are omitted. Of the other nodes, only a random 50% retain their labels. Unlabeled nodes are not used in loss or accuracy computation. More details about the data and the experimental setup can be found in Appendix H.

Results and discussion. As shown in Table 2, the best node-level and graph-level[7] accuracies were obtained using the proposed ResRGAT model. Similarly to the observations made in the previous experiment, models with only depth-wise skip connections perform worse.

The ablation study further indicates that concatenating edge features to the key vector may not be necessary for these data. Removing the skip connection and value transformation appears to result in a larger improvement, which supports the discussion in Sections 3 and 4. Note that removing the skip connection and value transformation leads to a simpler model with fewer parameters.

5.3. Performance on ZINC

We aim to verify that our model does not lose performance compared to existing work on real data. To this end, we run experiments on the ZINC molecule dataset and use the

[6] We simply feed the GRU/SGRU with the previous features of the node as state and the features of the preceding node as input. Note there is only one or none neighbour in this data set and thus aggregation and message functions can be omitted.

[7] Graph-level accuracy is 100% for an example only if all labeled nodes in the graph have been classified correctly, and is 0% otherwise. Please note that since this is a semi-supervised task, only a fraction of the nodes is labeled and the graph-level accuracy takes into account only these nodes.

Table 2. Node and graph-level accuracies over the test set of the Tree Max task (higher is better). Top part: baselines. Middle part: ResRGAT as described in Section 4. Bottom part: ablations.

	Node-level	Graph-level
RGCN	63.4 ± 1.1	35.2 ± 3.1
GGNN	21.0 ± 2.4	7.7 ± 2.5
RGAT	45.6 ± 5.5	17.3 ± 4.6
GatedGCN	37.1 ± 9.3	16.8 ± 5.7
Rel. TM	23.2 ± 2.3	10.7 ± 1.0
Rel. TM (shared)	23.3 ± 2.4	10.8 ± 0.8
ResRGAT	$\mathbf{93.3 \pm 1.0}$	$\mathbf{83.8 \pm 2.1}$
ResRGAT+SGRU	92.6 ± 1.1	82.0 ± 1.1
ResRGAT, no rel. cat.	92.5 ± 1.3	81.7 ± 1.6
ResRGAT, skip. att.	91.3 ± 0.1	79.0 ± 1.1
ResRGAT, val. transf.	92.4 ± 0.9	81.8 ± 0.9

Table 3. Number of layers (#L), number of parameters (#P) and MAE (where lower values are better) on the test set of ZINC.

	#L	#P	Test MAE±s.d.
GatedGCN	4	106k	0.375 ± 0.003
	16	504k	0.282 ± 0.015
GatedGCN+PE	16	505k	0.214 ± 0.013
MPNN (sum)	4	∼100k	0.288 ± 0.002
MPNN (max)	4	∼100k	0.328 ± 0.008
PNA (no scalers)	4	∼100k	0.247 ± 0.036
PNA	4	∼100k	$\mathbf{0.188} \pm 0.004$
ResRGAT	4	105k	0.327 ± 0.016
	16	489k	0.314 ± 0.019
+SGRU	4	99k	$\underline{0.240} \pm 0.009$
	16	483k	$\underline{0.212} \pm 0.009$

same constraints as Dwivedi et al. (2020) [8], who experiment with several architectures and achieve the best results for a Gated GCN using edge features. ZINC contains 10k training, 1k validation, and 1k test examples. The task is graph regression w.r.t. the "constrained solubility" property of molecules, measured using mean absolute error (MAE).

We use three different seeds. In the 4-layer setup, weight sharing is not used. A partially weight-shared ResRGAT is used for the 16-layer experiments to reduce the total number of parameters while not using full weight sharing, i.e. eight unique layers are used, each repeated twice consecutively.

Results and discussion: The results, presented in Table 3, show that the SGRU variant of our model performs better than GatedGCN while using the same data and a comparable number of parameters. In fact, it's on par with the Gated GCN variant that in addition uses positional encodings. Comparison with the vanilla variant of our model shows that the SGRU-based node update is essential for the obtained performance. The model recently presented by Corso et al. (2020) (PNA) performs best. Note, however, that PNA uses multiple aggregators and to achieve results better than the ResRGAT explicitly uses degree information with multiple different scalers.

[8] https://github.com/graphdeeplearning/benchmarking-gnns

Table 4. Validation and test F1 on ogbg-code2 (higher is better). † DAGNN (Thost & Chen, 2021) can only be applied to DAGs. *: without edge features, results reported by Tailor et al. (2021).

	Validation F1±s.d.	Test F1±s.d.
GIN	0.1376 ± 0.0016	0.1495 ± 0.0023
GCN	0.1399 ± 0.0017	0.1507 ± 0.0018
GIN+VN	0.1439 ± 0.0020	0.1581 ± 0.0026
GCN+VN	0.1461 ± 0.0013	0.1595 ± 0.0018
EGC-M*	0.1464 ± 0.0021	0.1595 ± 0.0019
PNA*	0.1453 ± 0.0025	0.1570 ± 0.0032
MPNN-Max*	0.1441 ± 0.0016	0.1552 ± 0.0022
DAGNN†	$\mathbf{0.1607} \pm 0.0040$	$\mathbf{0.1751} \pm 0.0049$
ResRGAT	$\underline{0.1595} \pm 0.0027$	$\underline{0.1715} \pm 0.0013$
+SGRU	0.1551 ± 0.0013	0.1676 ± 0.0011

5.4. Performance on OGBG-CODE2

We also run experiments on the OGBG-CODE2[9] dataset provided as part of the Open Graph Benchmark initiative (Hu et al., 2020). We adapt the code of the OGB team[10], using our model instead of the baselines[11].

The OGBG-CODE2 dataset contains 453k examples consisting of Abstract Syntax Trees (AST) and the task is a form of code summarization, where the goal is to predict the method name description based on its AST. Performance is measured using the F1 score between predicted tokens and expected tokens. OGBG-CODE2's original project-based split contains 408k training, 23k validation, and 22k test examples such that the test set does not contain examples from *projects* of the examples observed during training.

We run our experiments with three different seeds and report the validation and test F1 scores, as well as their standard deviations over the seeds in Table 4. We use 10 layers,

[9] https://ogb.stanford.edu/docs/graphprop/#ogbg-code2

[10] https://github.com/snap-stanford/ogb/tree/master/examples/graphproppred/code2

[11] We also implement a custom batch sampler that packs less examples in a batch if they are too large, in order to run experiments more efficiently by enabling more constant memory use.

where every two layers share weights, and mean pooling to build the final graph representation vector. We compare the ResRGAT against the baselines from OGB, as well as the directed acyclic graph neural network (DAGNN) recently proposed by Thost & Chen (2021).

Results and discussion: The results, presented in Table 4, show that our model performs better than baselines but it is outperformed by the DAGNN. In contrast to ZINC, ResRGAT *without* SGRU performs better here. Note that DAGNN exploits an inductive bias that relies on the assumption that the input graph is a DAG, and is thus especially tailored for this type of datasets but not generally applicable to all graph structures. In contrast, the GCN and GIN baselines, and ResRGAT can be applied to any graph.

6. Related Work

Li et al. (2019) 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., 2019), 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 to combat oversmoothing. The neighbourhood dilation proposed by Li et al. (2019) for point cloud semantic segmentation increases the receptive field of the network without requiring the information to pass through other nodes. Additionally, different normalization techniques (Zhou et al., 2020; Zhao & Akoglu, 2020; Li et al., 2020) have very recently been proposed that improve the training of deep GNNs. Works such as SSE (Dai et al., 2018) and IGNN (Gu et al., 2020) avoid backpropagation through deep networks by instead learning steady-state representations. AGGCN (Guo et al., 2019) transforms a graph into a fully connected graph and uses self-attention and thus, similarly to transformers (Vaswani et al., 2017), benefits from direct access to distant nodes. However, this leads to quadratic complexity in the number of nodes (due to attention in a fully connected graph). Nathani et al. (2019) propose a relational variant of GAT for link prediction in knowledge graphs. In contrast to previous work, we focus on improving the communication between neighbours in GNNs for graphs with edge features.

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 for every neighbour 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. DAGNN (Thost & Chen, 2021) generalizes the TreeLSTM and similarly to our model, uses attention-based aggregation and a GRU-based update. However, DAGNN is restricted to DAGs.

7. Conclusion

In this work, we show that several popular GNN architectures perform badly at tasks that explicitly challenge their ability to capture long-range patterns in graphs containing edge features. We present problems in breadth-wise backpropagation as the main reason behind this failure. We argue that these problems are exacerbated by exponential gradient decay due to uniform aggregation and certain approaches to propagate edge features. An experimental analysis using two synthetic tasks supports the claimed learning issues of existing methods. Then, we present a novel GNN architecture, that is developed applying well-known practices regarding long-range backpropagation both depth-wise and breadth-wise. The resulting architecture solves the synthetic tasks well, indicating that it is possible to capture long-range patterns that proved extremely challenging for several existing GNNs. In addition, it performs competitively on the real-world ZINC and OGBG-CODE2 datasets. We hope that the presented work draws attention to the importance of breadth-wise backpropagation in GNNs and will facilitate the development of better architectures in the future.

Acknowledgements

We thank all the reviewers for their helpful comments and suggestions. We also thank our colleagues Prof. Dr. Jens Lehmann, Gaurav Maheshwari, Priyansh Trivedi, Dr. Mikhail Galkin, and Rostislav Nedelchev for their help and illuminating discussions.

References

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 *International Conference on Machine Learning*, pp. 21–29, 2019.

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

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

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

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

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pp. 2787–2795, 2013.

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

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

Busbridge, D., Sherburn, D., Cavallo, P., and Hammerla, N. Y. Relational graph attention networks. *arXiv preprint arXiv:1904.05811*, 2019.

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *arXiv preprint arXiv:1909.03211*, 2019.

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

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder–decoder for statistical machine translation. pp. 1724–1734, 2014.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33, 2020.

Dai, H., Kozareva, Z., Dai, B., Smola, A., and Song, L. Learning steady-states of iterative algorithms over graphs. In *International conference on machine learning*, pp. 1106–1114. PMLR, 2018.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

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

Gu, F., Chang, H., Zhu, W., Sojoudi, S., and Ghaoui, L. E. Implicit graph neural networks. *arXiv preprint arXiv:2009.06211*, 2020.

Guo, Z., Zhang, Y., and Lu, W. Attention guided graph convolutional networks for relation extraction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 241–251, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1024. URL https://www.aclweb.org/anthology/P19-1024.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *CoRR*, abs/2005.00687, 2020. URL https://arxiv.org/abs/2005.00687.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

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

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

Klicpera, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems*, pp. 13354–13366, 2019.

Li, G., Muller, M., Thabet, A., and Ghanem, B. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9267–9276, 2019.

Li, G., Xiong, C., Thabet, A., and Ghanem, B. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Liang, X., Shen, X., Feng, J., Lin, L., and Yan, S. Semantic object parsing with graph LSTM. In Leibe, B., Matas, J., Sebe, N., and Welling, M. (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.

Nathani, D., Chauhan, J., Sharma, C., and Kaul, M. Learning attention-based embeddings for relation prediction in knowledge graphs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4710–4723, 2019.

Nickel, M., Rosasco, L., Poggio, T. A., et al. Holographic embeddings of knowledge graphs. In *AAAI*, volume 2, pp. 3–2, 2016.

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

Peng, N., Poon, H., Quirk, C., Toutanova, K., and Yih, W.-t. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics*, 5:101–115, 2017.

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

Rahimi, A., Cohn, T., and Baldwin, T. 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.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2019.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pp. 593–607. Springer, 2018.

Shaw, P., Uszkoreit, J., and Vaswani, A. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2074. URL https://www.aclweb.org/anthology/N18-2074.

Sinha, K., Sodhani, S., Dong, J., Pineau, J., and Hamilton, W. L. 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.

Song, L., Zhang, Y., Wang, Z., and Gildea, D. 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.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Tai, K. S., Socher, R., and Manning, C. D. 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.

Tailor, S. A., Opolka, F. L., Liò, P., and Lane, N. D. Adaptive filters and aggregator fusion for efficient graph convolutions, 2021.

Thost, V. and Chen, J. Directed acyclic graph neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=JbuYF437WB6.

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In Dy, J. G. and Krause, A. (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.

Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

Yang, C., Wang, R., Yao, S., Liu, S., and Abdelzaher, T. Revisiting" over-smoothing" in deep gcns. *arXiv preprint arXiv:2003.13663*, 2020.

Ye, R., Li, X., Fang, Y., Zang, H., and Wang, M. A vectorized relational graph convolutional network for multi-relational network alignment. In Kraus, S. (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.

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

Zhou, K., Dong, Y., Lee, W. S., Hooi, B., Xu, H., and Feng, J. Effective training strategies for deep graph neural networks. *arXiv preprint arXiv:2006.07107*, 2020.