NODE CLASSIFICATION ON RELATIONAL GRAPHS USING DEEP-RGCNS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Nagasai Chandra

March 2021

COMMITTEE MEMBERSHIP

TITLE: Node Classification on Relational Graphs using Deep-RGCNs

AUTHOR: Nagasai Chandra

DATE SUBMITTED: March 2021

COMMITTEE CHAIR: Franz Kurfess, Ph.D.
Professor,
Computer Science and Software Engineering Department

COMMITTEE MEMBER: Theresa Migler, Ph.D.
Assistant Professor,
Computer Science and Software Engineering Department

COMMITTEE MEMBER: Hisham Assal, Ph.D.
Lecturer,
Computer Science and Software Engineering Department

ABSTRACT

Node Classification on Relational Graphs using Deep-RGCNs

Nagasai Chandra

Knowledge Graphs are fascinating concepts in machine learning as they can hold usefully structured information in the form of entities and their relations. Despite the valuable applications of such graphs, most knowledge bases remain incomplete. This missing information harms downstream applications such as information retrieval and opens a window for research in statistical relational learning tasks such as node classification and link prediction. This work proposes a deep learning framework based on existing relational convolutional (RGCN) layers to learn on highly multi-relational data characteristic of realistic knowledge graphs for node property classification tasks. We propose a deep and improved variant, Deep-RGCNs, with dense and residual skip connections between layers. These skip connections are known to be very successful with popular deep CNN-architectures such as ResNet and DenseNet. In our experiments, we investigate and compare the performance of ***Deep***-RGCN with different baselines on multi-relational graph benchmark datasets, AIFB and MUTAG, and show how the deep architecture boosts the performance in the task of node property classification. We also study the training performance of ***Deep***-RGCNs (with N layers) and discuss the gradient vanishing and over-smoothing problems common to deeper GCN architectures.

ACKNOWLEDGMENTS

Thanks to:

My parents, Chandra Koteswar Rao and Chandra Venkata Laxmi, for their continued love, support and encouragement to pursue a master's degree.

Dhanunjay Vallamdas for motivating me to pursue education in the US and helping me make wise decisions (like choosing Cal Poly).

Professor Franz Kurfess for his inspiration, guidance, and great ideas.

Dr. Theresa Migler and Dr. Hisham Assal for their continued support and encouragement.

Andrew Guenther, for uploading this template.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

The general learning task in a machine learning algorithm is to learn a mapping from a given feature vector to an output prediction of some form. This output prediction could be a class label (classification), a regression score, or a latent vector. Euclidean data's feature vector can be embedded directly from an object alone, i.e., from pixels of an image or a set of words from a document. In contrast, in the non-Euclidean data structures, like graphs, an object's representation can contain its relationships with other items. These graphs have the advantage of holding valuable information about relations (i.e., edges) between individual entities (i.e., nodes), adding an extra dimension to the data. We can leverage this extra dimension for better results in various machine learning tasks. For example, one might recommend new friends to a user in a social network [64], classify a protein's role in a biological interaction graph [2], efficiently segment large point clouds [88], or predict a person's part in a collaboration network [90].

1.1 Knowledge Graphs

Knowledge graphs (KGs) are a specific type of graph. The edges in knowledge graphs are directed, i.e., the relations form an asymmetric triplet of the form subject-predicate-object and multi-relational, i.e., there are different edges for different types of ties. The phrase "knowledge graph" had gained most of its popularity when Google announced Google Knowledge Graph in 2012 [78], followed by further news of the de-

velopment of knowledge graphs by large-scale companies such as Amazon [44], eBay [69], Facebook [64], IBM [19], LinkedIn [71], Microsoft [77], Airbnb [12], and so on.

The increasing proliferation of non-Euclidean data in real-world setting has fueled knowledge graphs' popularity, leading to several (sometimes conflicting) definitions of the phrase "knowledge graph." In this thesis, we adopt a particular definition from Hogan et al. [35]. We view a knowledge graph as a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent these entities' relations. Figure 1.1 shows a sample knowledge graph where the nodes are entities and the edges are relations between the entities. KGs can be broadly distinguished based on their practical usage as open knowledge graphs and enterprise knowledge graphs. As explained by its name, open knowledge graphs are often published online, making their content accessible for the public good. Some examples of such large open KGs include DBPedia [48], Wikidata [87], Freebase [7], YAGO [34], etc. Enterprise KGs are typically developed and maintained internally in a company and applied for commercial use-cases. Some examples of prominent industries using enterprise KGs are Google [78](for web search); Amazon [21], Airbnb [12], and eBay [69] (for commerce); Facebook [64] and LinkedIn [71](for social networks); and companies like Bloomberg [56], Capital One [10], and Accenture [23] (for finance). The applications of such enterprise KGs include search [78], risk assessment [82], automation, personal agents [69], recommendations [[12], [21], [71]], advertising [71], business analytics [71], and more.

Despite such a comprehensive set of applications offered by the employment of KGs and the outstanding efforts invested in their creation and maintenance, even the most extensive knowledge bases (like DBPedia [48] and YAGO [34]) remain incomplete. This missing information in knowledge graphs harms the downstream applications and opens an opportunity for statistical relational learning (SRL) [43] research. SRL [43]

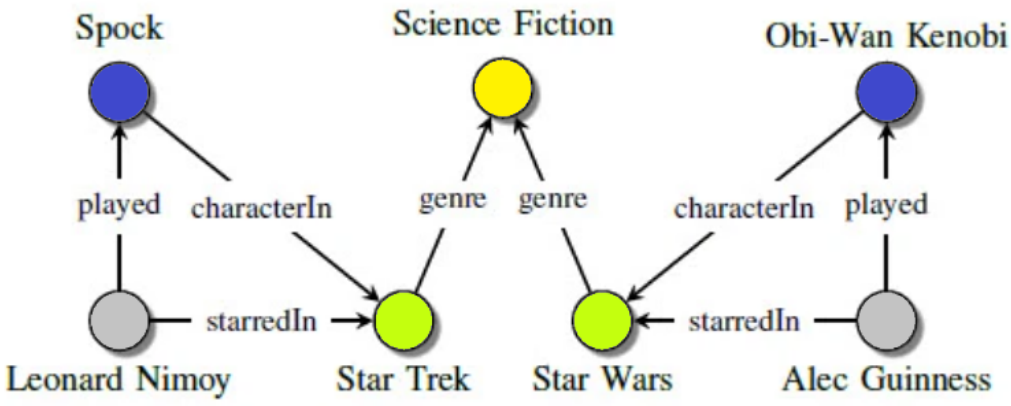


Figure 1.1: Knowledge Graph

is a subdiscipline of artificial intelligence and machine learning concerned with domain models that exhibit uncertainty and complex, relational structure. SRL emphasizes many canonical tasks that deal with graph structures, such as collective (or node) classification, link prediction, link-based clustering, knowledge base completion, etc.

1.2 Graph Convolutional Networks

In recent years, graph convolutional networks (GCNs) [41], a subset of convolutional graph neural networks (ConvGNNs) [[93], [16], [11]], have been gaining a lot of momentum, aiding in KB completion tasks. These GCNs operate directly on non-Euclidean graph data and are very promising for applications that depend on information modality. GCNs can be understood as a special case of a simple differentiable message-passing framework [26]. At each convolutional layer, the incoming information from neighboring nodes for each node is accumulated and passed through an element-wise activation function. Since the inception of GCNs, many GCN variants [[13], [29], [3], [1], [54], [94], [42]] were proposed, achieving state-of-the-art results in tasks such as graph classification, node classification, link prediction, and so on. An overview of such variants of GCN is provided in the Background and Related Works

sections. Despite such success of GCN frameworks, most of them do not credit the different types of relations available in realistic knowledge graphs, making them only suitable for graphs with a single or a low number of relation types. Schlichtkrull et al. [75] addressed this problem and proposed the R-GCN framework, which accumulates relation-specific transformed feature vectors of neighboring nodes through a normalized sum. The R-GCN framework is very similar to the GCN [41] architecture, except the relation-specific transformations introduced consider each edge's type and direction. The R-GCN model achieved state-of-the-art results in node classification and link prediction tasks for highly multi-relational graphs.

One may note an analogy between ConvGNNs and traditional convolutional neural networks (CNNs) [47] that have gained a lot of attention, particularly for machine learning tasks involving images. The core idea behind CNNs in image settings is to apply small kernels (or filters) over localized regions (sets of pixels) of an image using a convolution operator to extract features from that local region, together with all regions, leading to a feature map of the image. However, a fundamental difference in ConvGNNs for graphs is how regions of a graph are defined. Unlike the pixels of an image, nodes in a graph may have a varying number of neighbors. Convolutional kernels have been applied to data extracted from graphs in some previous works. For example, the ConvE [18] model wraps the source and relation vectors into matrices and concatenates them. A 2-D convolutional network then transforms the concatenated matrix into a feature map, which is finally used to make predictions of missing data.

One of the reasons behind the popularity of CNNs in ML tasks involving images is the ability to design and reliably train very deep CNN models [[31], [40], [37]]. For example, the ResNet-152 [31] architecture has 152 convolutional layers, with residual connections between the layers. However, this is not the same in GCNs, as most ar-

chitectures proposed so far involving graph convolution layers are shallow (not more than four layers). An obvious challenge of adding more graph convolution layers is the well-known vanishing gradient problem [33]. Back-propagating through these convolution layers leads to graph nodes' features converging to the same value. The same phenomenon occurs in deep stacks of convolution layers in CNN architectures as well. However, this problem has been alleviated by introducing residual [31], and dense [37] connections between the layers. In this work, we adopt the positive impact of dense connections and residual skip connections from CNN-based architectures such as DenseNet and ResNet and add this feature to the above-mentioned R-GCN architecture. One notable challenge/difference to notice is that a graph representation is specifically different from feature representations of images. To cope with these differences, we introduce a fusion block at the tail of the R-GCN layers. The fusion block is responsible for fusing the global semantic and multi-scale local features. The fusion is done by a concatenation operation of features from all R-GCN layers and passing the output through a 1 x 1 convolution layer followed by a max-pooling layer. We propose DenseR-GCN and ResR-GCN, deep architectures based on R-GCN layers with dense and residual connections, and evaluate their performance for node property classification tasks. Our proposed frameworks are different from R-GCN [75]. They comprise up to 14 layers of relational graph convolution layers with dense and residual connections to alleviate the vanishing gradient problem, while the R-GCN [75] model comprises only two relational graph convolution layers. We compare the results of different variants, namely, ResR-GCN-7, ResR-GCN-14, DenseR-GCN-7, DenseR-GCN-14, PlainR-GCN-2, PlainR-GCN-7, PlainR-GCN-14 on semi-supervised learning task for node classification on multi-relational knowledge graphs. We show how deeper architectures with residual and dense connections outperform R-GCN and deeper R-GCN concerning lower training losses and higher test accuracies.

1.3 Background Organization

The background section is organized as follows: a detailed introduction of deep CNN architectures ResNet [31] and DenseNet [37], a brief overview of some popular inductive techniques applied to knowledge graphs, mainly knowledge graph embeddings and graph neural networks, a summary of a few approaches involving spectral and spatial representations of graphs, and some techniques that employ attention mechanism that prioritizes features from essential nodes. These concepts are followed by a brief description of the GCN and R-GCN models. We further provide a quick review of R-GCN with its results for entity classification on multi-relational knowledge graphs such as AIFB, AM, MUTAG, and BGS datasets.

Related works will cover the variants of GCN approaches proposed and recent works revolving around Deep-GCNs for semantic segmentation in point clouds and node classification in protein-protein interaction graphs. These concepts are followed by an overview of state-of-the-art baselines on multi-relational RDF datasets, such as RDF2Vec embeddings [73], Weisfeiler-Lehman kernels (WL) [76], and hand-designed feature extractors (Feat) [67].

This thesis presents a brief review of graph representation learning techniques for multi-relational data. To the best of our knowledge, we are the first to investigate the behavior of residual and dense connections in combination with deep structures of R-GCN layers. We study that residual and dense connections significantly help in training and help alleviate the vanishing gradient and over-smoothing problems.

Chapter 2

BACKGROUND

2.1 Convolutional Neural Network

To better understand graph convolutional networks (GCNs) concepts, we summarize the well-known CNN and some popular deep CNN architectures, namely, ResNet [31] and DenseNet [37]. A convolutional neural network (CNN, or ConvNet) is a popular class of deep neural networks, most commonly applied in computer vision tasks. The concept of a CNN is analogous to neurons' connectivity pattern in the human brain and was inspired by the Visual Cortex organization. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A CNN is a multi-layer neural network (NN) architecture that contains convolutional layer(s) pursued by fully connected (FC) layer(s). Subsampling layers can exist between the convolution and fully connected layers. The primary building block of a CNN is a convolution layer. This layer determines the output of associated inputs in the receptive field. This output is achieved through kernels, which are convolved over the information data's height and width, computing the dot product between the input and filter values, building a 2-D activation map of that filter. With this convolution layer, a CNN can learn those filters quickly, which activate when a particular type of feature at some spatial position of the input is observed. Despite the advantages of using convolution layers, more layers lead to a phenomenon that commonly occurs while training recurrent neural networks, called the vanishing gradient problem, requiring us to use small learning rates with gradient descent. The vanishing gradient problem is a phenomenon where the gradient will quickly end up

being vanishingly small, effectively preventing the weight from changing its value. Many approaches were proposed to alleviate this problem, and we will summarize the popular methods, namely, residual skip connections and dense connections, in the following sub-sections

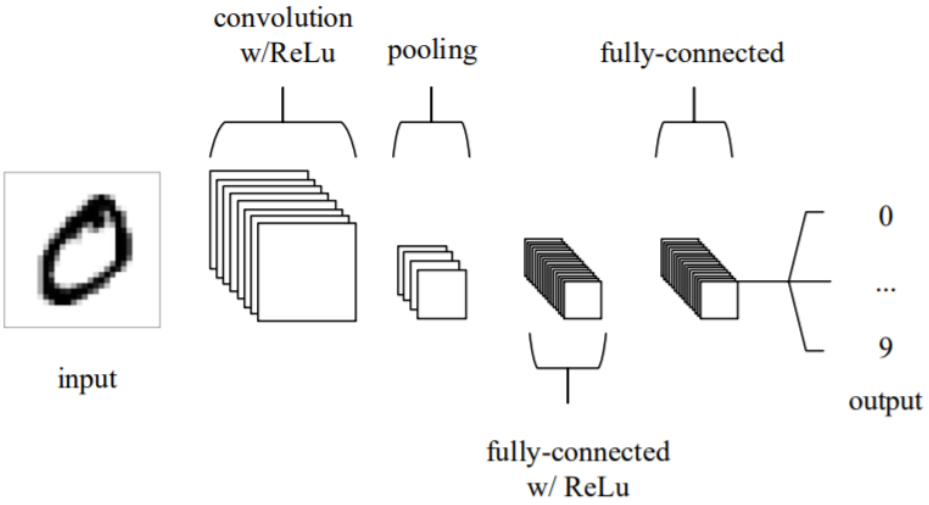


Figure 2.1: A simple 5 layer CNN architecture for MNIST classi cation [65]

Apart from the convolution layer, the CNN architecture can also include other types of layers:

A nonlinear function plays an important role. The main objective of a nonlinearity layer is to transform the input signal to the output signal, and that signal will be utilized as an input in the next layer. Some popular types of nonlinear functions are sigmoid, Tanh, ReLU, LeakyReLU, PReLU, and ELU.

CNN can have local or global subsampling (pooling) layers that add a neuron's outputs at one layer into an individual neuron in the following layer. Its main objective is to scale down the representation's spatial size to diminish the number of parameters and calculations in the model. It speeds up the calculations

and averts the problem of overfitting that the CNN architecture may cause. Max and average pooling layers are the most commonly used pooling layers.

Fully connected layers (FCs) are standard deep NNs, which seek to build the predictions from the activations to be used for classification or regression. Their principle is similar to conventional MLPs. This layer acquires the complete connections to every activation in the antecedent layer, and the activations can be calculated by using matrix multiplication followed by a bias offset.

Finally, the classification layer is used to guide the training process of a NN. Different loss functions may be suitable at this layer depending on the ML task, like, softmax, cross-entropy, etc.

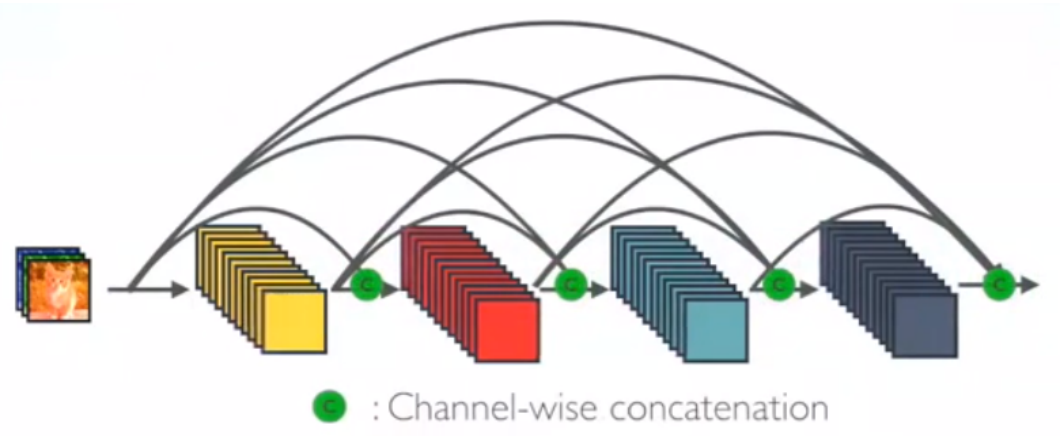


Figure 2.2: A representation of channel-wise concatenation in a DenseNet block [84]

2.1.1 Dense Connections

This architecture is proposed by Huang et al. [37], where every layer is coupled in a feed-forward manner. This model with L layers has $L(L+1)$=2 direct connections. It concatenates the output feature maps and incoming feature maps; therefore, each layer acquires collective knowledge from all previous layers. The dense connections

proposed in this architecture have significantly reduced the vanishing gradient problem, minimizing the number of parameters and reusing features. This architecture has attained state-of-the-art performance on several benchmark datasets, including CIFAR-100 [45], ImageNet [17], and SVHN [60]. However, the only drawback is that the dense connections utilize much extra memory for tensor concatenations. One way to overcome this memory issue is to strategically allocate shared memory [70] for intermediate concatenation results. The subsequent layers are allowed to overwrite the intermediate results of previous layers. These intermediate results are recalculated during backward propagation. This method could significantly solve the memory issues at the cost of computing time dedicated for re-calculations. We leave this approach of shared memory allocation for intermediate results for future works.

We adopt the naive implementation of dense connections to our architecture to alleviate the vanishing gradient descent problem in deep relational graph convolutional networks. We show the positive effect of dense connections by comparing the proposed deep neural network model, DenseR-GCN, with a bare block of an equal number of layers.

2.1.2 Residual Skip Connections

The ResNet is a similar kind of deep CNN architecture that uses residual skip connections between the convolution layers. The ResNet is an extremely deep model with 152 layers that makes advanced traces in categorization, localization, and detection with one magnificent model. The residual block in this architecture addresses the vanishing gradient problem caused by training a deep model by commencing identity skip connections so that layers can copy their inputs to the next layer. The idea is simple; each next layer should learn something new and divergent from the input given.

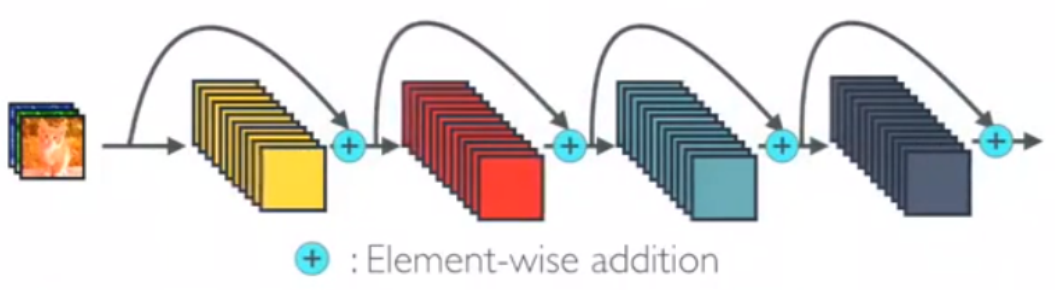


Figure 2.3: A representation of element-wise addition in a ResNet block [84]

He et al. [31] proposed this architecture for improved recognition of images. The ResNet has attained state-of-the-art results by reducing the top-1 error for the ImageNet benchmark dataset that contains 1000 classes. We implement residual skip connections between relational graph convolutional layer blocks and evaluate node classification performance on the multi-relational knowledge graphs AIFB and MUTAG. We call the deep variant with residual skip connections between R-GCN layers as ResR-GCN and compare it with a bare block of an equal number of R-GCN layers alongside other baselines.

2.2 Knowledge Graph Embeddings

Machine learning applications on knowledge graph data can broadly be divided into two types; applications of knowledge graph refinement (knowledge base completion tasks) and knowledge graphs for downstream applications such as information retrieval, question answering, recommender systems, etc. However, traditional machine learning techniques assume their input representations as numeric vectors, which are quite different from how knowledge graph representations are usually expressed. The concept of knowledge graph embeddings resolves this difference. The main goal of knowledge graph embedding techniques is to create a dense representation of the graph (i.e., embed the graph) in a low-dimensional, continuous vector space that can

be used for machine learning tasks. Typically, a graph's overall embedding is composed of an entity embedding for each available node and a relation embedding for each edge label. Figure 2.4 shows how embedding methods generate representations of the knowledge graph elements embedded in a vector space. These embedded vectors' overall goal is to abstract and preserve the knowledge graph's latent structures in hand. Another similar approach of using algorithms to detect communities or clusters, find central nodes and edges, etc., fall under graph analytics for unsupervised tasks. As our main objective in this thesis is to study (semi)-supervised node classification in graph data structures, we shall not divert the flow towards graph analytics concepts. We stick to self-supervised learning using knowledge graph embeddings and supervised learning techniques using graph neural networks in this section.

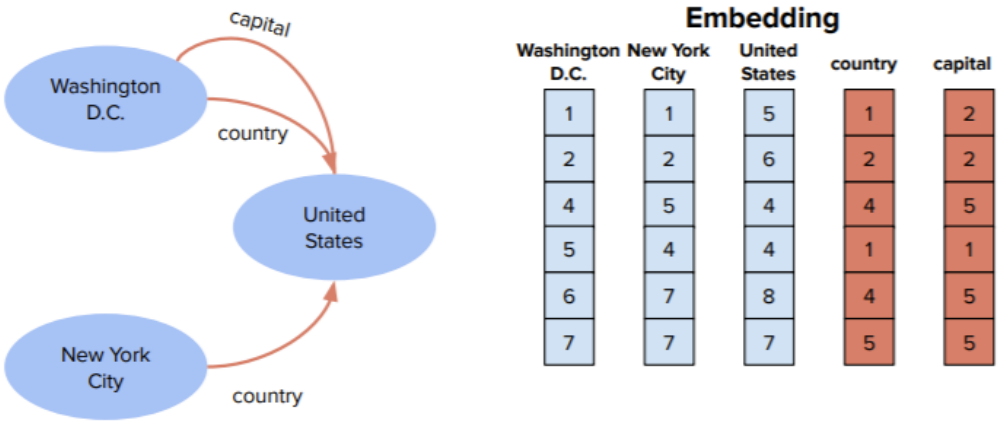


Figure 2.4: An example representation of knowledge graph embedding [66]

A wide range of knowledge graph embedding techniques have been proposed, and we provide a high-level introduction to some popular methods, as follows:

Translational models: These models interpret edge labels as transformations from subject nodes to object nodes. TransE 2013 [9] is the first kind of translational model, where the basic idea is to make the sum of the head vector and relation vector as close as possible with the tail vector. Either L1 norm

or L2 norm is used to measure the distance between the vectors. As an improvement to TransE, TransH 2014 [92] aims to deal with one-to-many/many-to-one/many-to-many relations and does not increase the model's complexity or make the training hard. TransH interprets a relation as a translating operation on a hyperplane. Unlike TransE and TransH, TransR 2015 [55] does not assume that entity and relation vectors exist in the semantic space, as each entity can have many aspects. Different relation pays attention to the different aspects of the entity. TransR models entities and relations in two distinct spaces, i.e., an entity space and multiple relation spaces (relation-specific entity spaces), and performs the translation in the corresponding relation space. TransD 2015 [38] uses two vectors to represent each entity and relation. The first vector represents the meaning for an entity or a relation, and the second vector (called projection vector) is used to construct mapping matrices. Recently in 2019, RotatE [80] proposed translational embeddings in complex space, which allows capturing more characteristics of relations, such as direction, symmetry, inversion, antisymmetry, and composition.

Tensor decomposition models: Another approach to derive graph embeddings is to apply methods based on tensor decomposition. The term tensor refers to a multidimensional numeric field that generalizes scalars, vectors, and matrices towards arbitrary dimensions. Tensor decomposition involves decomposing a tensor into lower-order tensors from which the original tensor can be recomposed (or approximated) by a fixed sequence of basic operations. These low-order or elemental tensors can be viewed as capturing latent factors underlying the information contained in the original tensor. Some of the popular approaches based on tensor decomposition are DistMult [95], RESCAL [63], HolE [62], ComplEx [83], SimplE [39], and TuckER [5].

Neural models: Some approaches use neural networks to learn embeddings with non-linear scoring functions for plausibility scores. One of the earliest proposed neural models was Semantic Matching Energy (SME) 2014 [8], which learns weights for two functions so that the dot product of the function results gives the embeddings. Some other neural models proposed are Neural Tensor Networks (NTN) 2013 [79], Multi-Layer Perceptron 2014 [20], ConvE 2018 [18] (convolutional kernels), HypER 2019 [4], etc.

Language models: Embedding techniques like word2vec [58] and GloVe [68] were initially explored to represent natural language within machine learning frameworks. These approaches compute embeddings for words based on large corpora of text such that words/tokens used in similar contexts have similar vectors. A significant contrast between graphs and text in natural language is that the text consists of arbitrary-length sequences of terms while a graph consists of an unordered set of sequences of three terms. Along these lines, RDF2Vec [73] was proposed. It performs random walks on the graph and records the paths as "sentences," which are then fed as input into the word2vec model. Another similar model, KGloVe [15], based on the GloVe model, has also gained some popularity. KGloVe uses a personalized PageRank technique to determine the most related nodes to a given node, whose results are then fed into the GloVe model.

Ontology/rule-based models: These approaches use a given set of rules or ontology to refine the predictions made by embeddings. One such technique is KALE [27]. This method computes entity and relation embeddings using TransE adapted to consider further rules using t-norm fuzzy logics.

For further discussion and examples of knowledge graph embedding methods, we refer the reader to Section 5.2 of a recently published survey, Knowledge Graphs [35], by

Hogan et al. 2021 and a recent study by Palmonari et al. [66], summarizing the state-of-the-art in the field of knowledge graph embeddings.

2.3 Supervised Learning with Graph Neural Networks (GNNs)

In this section, we give an overview of custom machine learning models adapted for graph-structured data. GNNs were introduced back in 2005, but they started to gain popularity in the last five years. A GNN builds a neural network based on a graph's topology, i.e., nodes are connected to their neighbors per the data graph. A model is typically learned to map input features of nodes to output features in a supervised manner. The nodes may be either manually labeled or taken from the knowledge graph. GNNs support end-to-end supervised learning for specific tasks; for instance, given a set of labeled examples, GNNs can be trained to classify elements of a graph or the graphs themself.

2.3.1 Recursive GNNs

A recursive form of GNNs that can process most of the practically useful types of graphs, e.g., acyclic, cyclic, directed, and undirected, was proposed by Scarselli et al. [74] in 2009. This architecture takes as input a graph where nodes and edges are associated with feature vectors that can capture node and edge labels, weights, etc. These feature vectors remain fixed throughout the process. Every node in the graph is also associated with a state vector, which is recursively updated, until a certain fixpoint, based on feature and state vectors from their respective neighboring nodes using a so-called transition function. Another parametric function called the output function is used to compute each node's final output based on its feature and state vector. This GNN model is flexible and can be adapted in various ways: we may

define neighboring nodes differently, for example, to include nodes for outgoing edges or nodes one or two hops away, or we may allow pairs of nodes to be connected by multiple edges with different vectors.

2.3.2 Convolutional GNNs

CNNs have gained popularity in machine learning tasks involving images [46]. The idea behind such CNN approaches is to apply small filters over localized regions of an image using a convolution operator to extract features from that local region. When applied to all local regions of images, such a convolution operator outputs a feature map for each image. As previously discussed, in GNNs, the transition function is applied over local input data regions. Following the same intuition of CNNs, a number of convolutional graph neural networks (or ConvGNNs) like GraphCNN [11], Semi-GCN [41], and Neural Networks for Graphs (NN4G) [57] were proposed. However, a key difference to note between CNNs and ConvGNNs is that nodes in a graph may have a varying number of neighbors, while pixels of an image have a fixed set of neighbors. To address this challenge, spectral representations [[11], [41]], spatial representations [[59], [26], [57]] and attention mechanisms [86] were employed. The spatial or spectral representations induce a more stable structure from the graph, while the attention mechanism is used to learn the nodes whose features are most important to the current node.

The first prominent research on spectral based ConvGNNS was presented by Bruna et al. [11], which developed a graph convolution based on the spectral graph theory. Since this time, there have been increasing improvements, extensions, and approximations on spectral-based ConvGNNs [[32], [16], [41], [49]].

NN4G [57] is the first work towards spatial-based ConvGNNs. The NN4G learns graph mutual dependency through a compositional neural architecture with independent parameters at each layer. The neighborhood of a node can be extended through incremental construction of the architecture. The model performs graph convolutions by summing up a node's neighborhood information directly.

Graph Attention Networks (GATs) [86] aim to learn the nodes whose features are most important to the current node. They assume the contributions of neighboring nodes to the central node to be neither identical nor pre-determined. Thus, they adopt an attention mechanism that can learn the relative weights between two connected nodes. GAT also performs multi-head attention to increase the model's expressive capability. While GAT assumes the contributions of attention heads are equal, a similar approach Gated Attention Network (GAAN) [96] introduces a self-attention mechanism that computes an additional attention score for each attention head.

2.4 Graph Convolution Network for Semi-supervised Learning on Graphs

In this section, we give an introduction of Graph Convolution Networks (commonly known as GCNs or Semi-GCNs) developed by Thomas Kipf and Max Welling in the paper *\Semi-Supervised Classi cation with Graph Convolutional Networks (2017)"* [41]. This work uses an efficient layer-wise propagation rule that is based on a first-order approximation of spectral convolutions on the graph. The main idea behind this approximation was inspired by signal/wave propagation. The information propagation in spectral GCNs can be thought of as a form of signal propagation along the network's nodes. Spectral GCNs generally make use of the Eigen-decomposition of graph Laplacian matrix to implement the information propagation. In such ap-

proaches, the Eigen-decomposition helps us understand the graph structure and helps classify nodes in graphs. The Semi-GCN model uses an adjacency matrix of the input graph in the forward pass enabling the model to learn the feature representations based on node connectivity. The resulting GCN can be seen as the first-order approximation of Spectral Graph Convolution in a message-passing network where the information is propagated along the neighboring nodes within the graph. A simplified form of forward pass in semi-GCN can be represented as:

$$H^{[i+1]} = \quad (W^{[i]}H^{[i]}A \ ) \tag{2.1}$$

where $H^{[i+1]}$ is the feature representation at layer $i+1$, A is the normalized version of adjacency matrix A, $W^{[i]}$ is the weights at layer i and is the non-linear activation function (like ReLu) used. The Semi-GCN model employs a renormalization trick, which is a form of symmetric normalization of adjacency matrix $A \quad ! \ D^{\ 1=2}AD^{\ 1=2}$ using the degree matrix (D) of the graph. In graph terminology, the degree matrix is a diagonal matrix containing information about each vertex's degree, i.e., the number of edges attached to each vertex. The degree matrix is generally used together with the adjacency matrix to construct the Laplacian matrix. This renormalization technique helps prevent numerical instabilities and vanishing/exploding gradients for the model to converge. So the final form of information propagation operation introduced by Semi-GCN is as follows:

$$H^{[i+1]} = \quad (W^{[i]}H^{[i]}D^{\ 1=2}AD^{\ 1=2}) \tag{2.2}$$

The Semi-GCN model outperformed other related methods on semi-supervised node classification tasks by a significant margin. The model was tested on Citeseer, Cora,

and Pubmed datasets, popular baselines for node classification tasks. With these experiments, the Semi-GCN model was shown to offer better predictive performance than previous spectral-based approaches such as ChebNet [30] and Spectral Network [11]. This thesis adopts an extension of the Semi-GCN model, R-GCN by Schlichtkrull et al. [75] for node classification for highly multi-relational graphs.

2.5 Relational Graph Convolutional Networks

Despite the enormous success of graph convolutional networks [41], they tend to fail or underperform in the cases of highly multi-relational graphs, as these models do not incorporate the edge features (direction and type). To address the issue of modeling multi-relational data and generalize the GCN framework to handle different relationships between entities in a knowledge base, R-GCN was proposed by Schlichtkrull et al. in the paper *"Modeling Relational Data with Graph Convolutional Networks [75]."* This work extends GCN that operates on local graph neighborhoods to large-scale relational data using multi-edge encoding to compute entities' embeddings. They also introduce parameter sharing techniques to apply R-GCNs to multigraphs with large numbers of relations. This section presents the back-bone of the R-GCN layer introduced by Schlichtkrull et al. [75] and discusses the regularization techniques used in their work.

The GCN, as discussed in the previous sections and similar methods such as, graph neural networks [74] and Neural FPs [22], can be understood as a particular case of a simple differentiable message-passing framework [26] which can be represented as:

$$h_i^{(l+1)} = \left(\sum_{m \in M_i} g_m(h_i^{(l)}, h_j^{(l)}) \right) \tag{2.3}$$

where $h_i^{(l)} \in \mathbb{R}^{d^{(l)}}$ is the hidden state of node v_i in the l-th layer of the neural network, with $d^{(l)}$ being the dimensionality of this layer's representations. Incoming messages of the form $g_m(i,j)$, where $j \in N_i$ (neighborhood of node i), are accumulated and passed through an element-wise activation function $\sigma(\cdot)$, such as the ReLu or LeakyReLu. M_i denotes the set of incoming messages for node v_i and is often chosen to be identical to the set of incoming edges. $g_m(\cdot,\cdot)$ is typically chosen to be a (message-specific) neural network-like function or maybe a linear transformation like $g_m(h_i,h_j) = Wh_j$ with a weight matrix W such as in Equation 2.2 proposed by Kipf and Welling (2017) [41].

Such a transformation has been very effective at encoding and accumulating features from local, structured neighborhoods and has led to significant improvements in semi-supervised node classification based on graphs [41] and graph classification [22]. The R-GCN model employs a simple but better propagation model similar to Equations 2.2 and 2.3 for calculating the forward-pass update for an entity denoted by v_i in a directed and labeled multi-graph.

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in R}\sum_{j\in N_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right) \tag{2.4}$$

where N_i^r denotes a set of all neighbor indices of node v_i under relation $r \in R$. $c_{i,r}$ is a problem-specific normalization constant that can either be learned or chosen in advance. This addition of variables at each accumulation of transformed feature vectors of neighboring nodes through a normalized sum allows the model to learn relation-specific features about the entities. Such relation-specific transformations are dependent on the type and direction of each edge in consideration. Similar to Semi-GCN [41], the R-GCN model adds a single self-connection of a special relation type to each node $v_i \in G$, to make sure the corresponding information of a node v_i

at layer l is not lost by the time it reaches layer $l+1$. We adopt this graph encoder model as a backbone to encode the representations of entities in highly multi-relational graphs. One of the main concerns of using such a relational graph encoder model is the rapid growth in the number of parameters with the number of relations present in the graph. Such growth in the number of parameters usually leads the model to over-fit very easily. To address this issue, the authors of [75] introduce effective parameter sharing and sparsity constraint techniques using basis- and block-diagonal-decomposition. We adopt the basis function decomposition technique to our proposed model DenseR-GCN.

In the basis decomposition regularization method, each $W_r^{(l)}$ is defined as a linear combination of basis transformations with coefficients. Only the coefficients depend on relation type r while the transformations are shared within each R-GCN layer.

$$W_r^{(l)} = \sum_{b=1}^{B} a_{rb}^{(l)} V_b^{(l)} \tag{2.5}$$

Where B is the number of bases employed and $a_{rb}^{(l)}$ are the coefficients at layer l. The R-GCN model's effectiveness in handling multi-relational graphs is evaluated on two fundamental statistical relational learning (SRL) tasks: entity classification (assigning types or categorical properties to entities) and link prediction (recovering missing triples in graphs). For the link prediction task, the R-GCN model is combined with a so-called decoder to exploit the representations encoded by the R-GCN model to predict labeled edges. A tensor factorization model like DistMult [95] or ComplEx [83] is used as the decoder. The use of the R-GCN layer has achieved competitive results on standard link prediction benchmarks, such as the FB15k-237 dataset, outperforming other baselines such as DistMult [95], HolE [62], and RESCAL [63]. For the entity classification task, the R-GCN model is used with softmax classifiers at each node in

the graph similar to [41]. These softmax classifiers take the R-GCN network's node representations and predict the labels while optimizing a cross-entropy loss. The R-GCN model is also evaluated for the entity classification task on graph datasets (AIFB, AM, MUTAG, and BGS) with many relation types. The results are compared against state-of-the-art classification results from RDF2Vec embeddings [73], Weisfeiler-Lehman kernels (WL) [76], and hand-designed feature extractors (Feat) [67]. More information about these baselines for entity classification tasks on multi-relational data is discussed in the Related Works section. The R-GCN model achieved state-of-the-art results on AIFB and AM baselines.

Chapter 3

RELATED WORKS

This section introduces the recent works that are related to graph learning techniques based on GCNs. Additionally, few approaches that explore the deep architectures of GCN and its variants are presented. Finally, we give a brief introduction to the previous works for semi-supervised node classification tasks on multi-relational graph datasets; namely, Weisfeiler-Lehman (WL) graph kernels [76], RDF2VEC embeddings [73], and Path Tree [85].

3.1 Graph Convolution Network Variants

As discussed, GCNs have gained much attention to generalize convolutions to the graph domain. This section reviews some of the GCN approaches categorized as spectral and spatial approaches based on the propagation type or aggregator employed. As such methods discussed in this section provide background knowledge and can be viewed as related works in node classification and knowledge representation domain, the reader may view this content as both background information and related works.

3.1.1 Spectral Variants

The spectral approaches work with a spectral representation of the graphs. They define graph convolutions by introducing filters from the perspective of graph signal processing based on graph spectral theory. One drawback of such approaches is that

it requires the entire graph to be processed simultaneously, which can sometimes be impractical for large graphs with billions of nodes and edges such as the social network graphs. This requirement of handling the entire graph makes it difficult to take advantage of the parallel processing power available today. Another disadvantage of such approaches is that they assume a fixed graph, thus generalizing poorly to new or different graphs. Some of the popular spectral approaches of graph convolutional networks are:

Bruna et al. proposed Spectral Network [11] in 2013. The convolution operation is defined in the Fourier domain by computing the Eigendecomposition of the graph Laplacian. This operation results in potentially intense computations and non-spatially localized filters.

ChebNet [30] suggests that a truncated expansion can approximate the convolution operation on eigenvalues in terms of Chebyshev polynomials. The ChebNet uses a K-localized convolution to define a convolutional neural network. This approach could remove the need to compute the Laplacian Eigenvectors.

GCN [41] limits the layer-wise convolution operation to K = 1 to alleviate the problem of overfitting on local neighborhood structures for graphs with extensive node degree distributions.

Adaptive Graph Convolution Network (AGCN) [53] learns a 'residual' graph Laplacian and adds it to the original Laplacian matrix. With this approach, AGCN is proven to be effective in several graph-structured datasets.

GGP [61] was proposed in 2018 by Ng et al. The authors present a Gaussian process-based approach (GGP) to solve the semi-supervised learning problems.

Michael Schlichtkrull et al. [75] extended the idea of graph convolutional networks to large-scale relational data. We adopt this R-GCN model's framework as the convolution aggregator for the semi-supervised node classification task.

3.1.2 Spatial Variants

Spatial (or non-spectral) approaches define convolutions directly on the graph, operating on spatially close neighbors. They formulate graph convolutions as aggregating feature information from neighbors. As they directly perform convolution in the graph domain by aggregating the neighbor nodes' information, they do not need the entire graph to be processed simultaneously. With sampling strategies, the computation with spatial approaches can be performed in a batch of nodes instead of the whole graph, improving training efficiency. Contrary to spectral approaches that assume fixed graphs leading to poor generalization to new or different graphs, spatial models perform graph convolution locally on each node, sharing weights across different locations and structures. This allows spatial approaches to generalize well for new graphs. Some of the popular spatial variants of GCNs are:

Neural FPs [22], proposed at NIPS 2015, uses different weight matrices for nodes with different degrees. However, this method cannot be applied to large-scale graphs with more node degrees.

Atwood et al. proposed the diffusion-convolutional neural networks (DCNNs) [3] that use transition matrices to define the neighborhood for nodes in DCNN.

Dual graph convolutional network (or DGCN) [97] jointly considers the local consistency and global consistency on graphs. It uses two convolutional networks to capture the local/global consistency and adopts an unsupervised loss to ensemble the networks.

Another popular spatial GCN variant is the GraphSAGE [29] framework. This framework generates embeddings by sampling and aggregating features from a node's local neighborhood.

3.2 Deep Architectures of Graph Convolutional Networks

This section gives an overview of deep architectures of graph convolutional networks that have recently gained much attention in many graph-based applications. A few works include deep GCNs for Graph-to-Sequence learning [28], graph learning on 3D Point Clouds for semantic segmentation [51], and graph learning on biological networks [50]. These methods employ different approaches to overcome the vanishing gradient challenges faced by deep stacks of graph convolution layers.

Li et al. 2019 [51] introduced the concept of DeepGCNs by adopting residual skip connections, dense connections, and dilated convolutions techniques from successful deep CNN architectures to alleviate the difficulty of training, which is the primary problem impeding GCNs to go deeper. Their model with 56 GCN layers with residual connections has shown significant improvements over existing techniques for semantic segmentation of 3D point cloud data using such techniques.

Li et al. 2019 [50] extended this work to graph learning on biological networks. They evaluate the performance of deep GCN architectures with different GCN variants such as EdgeConv [91], GraphSAGE [29], and GIN [94]. Another GCN variant using a max aggregator called Max-Relative GCN (MRGCN) is proposed. The implementation of MRGCN has shown promising results on the protein-protein interaction graph dataset. Their architecture with 14 and 28 GCN layers supplemented with residual connections has achieved state-of-the-

art results for the ModelNet40 point cloud classification task. A deep architecture based on 28 and 56 layers of their proposed Max-Relative GCN (MRGCN) convolution operators complemented with residual connections achieves state-of-the-art results for the PPI node classification task. Our work is closely related to Li et al. [50], which uses residual/deep connections with MRGCN layers. We explore the functionalities of residual/dense connections in between R-GCN layers and evaluate their performance on learning on multi-relational graphs.

Li et al. 2020 [52] propose differentiable generalized aggregation functions to unify various message aggregation operations (like mean, max, sum aggregators). They also introduce a novel MsgNorm normalization layer to boost the performance of networks with under-performing aggregation functions. A combination of such techniques and a pre-activation version of deep GCN with residual connections has achieved state-of-the-art results for graph learning on several Open Graph Benchmark (OGB) datasets [36]. These datasets include node classification baselines, namely, ogbn-proteins and ogbn-arxiv, and graph classification baselines, namely, ogbg-molhiv and ogbg-ppa datasets.

Guo et al. [28] work uses dense connections between GCN layers for transducing graph structures to sequences for text generation. Such a deep GCN architecture with dense connections can integrate local and non-local features to learn a better structural representation. They evaluate different deep architectures' performance with residual/dense connections, GCNs with layer aggregations, and graph attention for graph-to-sequence learning. The model proposed outperforms the state-of-the-art neural models significantly on AMR-to-text generation and syntax-based neural machine translation tasks.

GCNII model was recently proposed by Chen et al. in the paper *\Simple and Deep Graph Convolutional Networks"* [14]. This work overcomes the over-

smoothing problem caused in deep GCN architectures by extending the vanilla GCN model with initial residual and identity mapping techniques. The initial residual connection is an alternative to the residual connection that ensures that each node's final representation retains at least a fraction of the input layer features. The model also employs an identity mapping technique that has shown success in ResNet architecture. The identity mapping essentially ensures that a deep stack of GCN layers achieves at least the same performance as its shallow version does. The combination of such techniques has outperformed several state-of-the-art models (like GCN [41], GAT [81]) for semi- and full-supervised tasks on baselines like Cora, Citeseer, Pubmed, Chameleon, Cornell, Texas, Wisconsin, and PPI datasets.

3.3 Multi-Relational Graph Learning

Various approaches have previously aimed to perform representation learning on multi-relational graphs. Schervashidze et al. 2011 [76] proposed the Weisfeiler-Lehman (WL) graph kernels using a quick feature extraction scheme based on the Weisfeiler-Lehman test of isomorphism on graphs. This feature extraction method maps the original graph to a sequence of graphs, whose node attributes capture topological and label information. A family of efficient kernels for large graphs with discrete node labels is defined based on such graph sequences. The kernels proposed in this work outperformed state-of-the-art approaches on several benchmark data sets for graph classification in terms of accuracy and runtime. Ristoski and Paulheim proposed the RDF2VEC embeddings in 2016 [73]. The RDF2VEC extracts walks on labeled graphs, which are then processed using the Skip-gram model to generate entity embeddings for the subsequent entity or graph classification. The vector representations produced by RDF2VEC outperformed previous techniques on various

RDF graphs, including AIFB, MUTAG, and BGS. We compare our model with the results of some of these approaches for the entity classification task. More recently, in 2019, a different approach has been proposed by Vandewiele [85] to tackle representation learning for multi-relational graphs. This approach focuses on interpretability by building a decision tree of class-specific substructures to classify various entities within the knowledge graph.

In this section, we have covered various deep architectures of graph convolutional networks. We also introduced spectral and spatial variants of GCNs and some previous works that aimed to perform representation learning on multi-relational graphs. In the next chapter, we introduce the system architecture of our proposed model.

Chapter 4

SYSTEM DEVELOPMENT

Our model is primarily based on the R-GCN layers proposed by Schlichtkrull and Kipf [75] for modeling multi-relational knowledge graphs to predict the missing information in knowledge bases like missing properties of entities or relations between entities. As discussed in the introduction and background sections, training deep neural networks based on graph convolution operators, such as R-GCN, GCN, or EdgeConv layers, is prone to over-smoothing or vanishing gradient problems. We adopt residual connections and dense connections and explore deep structures' behavior to overcome this. The relational graph convolutional network (R-GCN) operator uses relation-specific transformations that depend on the type and direction of an edge and accumulate the transformed feature vectors of neighboring nodes through a normalized sum. At each layer, this operation occurs in parallel for every node in the graph, and the output features are then passed on to the next layer through a suitable activation function (we use ReLu). To avoid R-GCN layers' weights to overfit for rare relations, we use basis decomposition to regularize the weights at each R-GCN layer. We provide an overview of this regularization approach in the background section. We use a constant pre-defined number of filters (or hidden nodes) at each convolution layer. Our main contribution in this work is to extend the impact of using residual connections and dense connections in deep learning to multi-relational graph data for improved multi-relational reasoning, especially for higher degree nodes.

Graph De nition. A directed and labeled multi-graph G is represented as $G = (V; E; R)$, where V is the set of unordered nodes (entities) and R is a s et of labeled

edges (relations), such that if there is an edge $(v_i; r; v_j)$ 2 E where $v_i; v_j$ 2 V, then vertex v_i is connected to a vertex v_j with an edge of label r, where r 2 R.

4.1 Architecture

Our model, *Deep*-RGCN, can be broken down into three components (Figure 4.1), the convolution, the fusion block, and the prediction block. The convolution block consists of R-GCN layers. We compare the behaviors of several variants of this convolution block, depending on the number of R-GCN layers used and the type of connections employed (if any) between the layers.

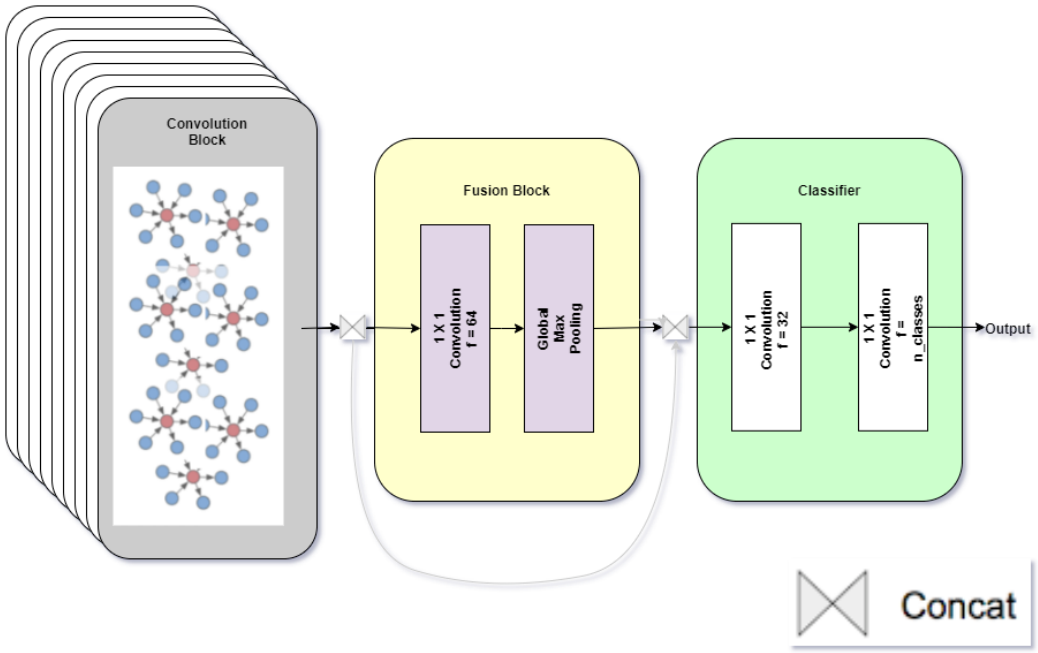


Figure 4.1: Proposed architecture

4.1.1 Convolution Block

This component of our model comprises of a series of predefined number of R-GCN layers. The forward propagation of an R-GCN layer as proposed by Schlichtkrull et al. [75] can be represented as follows:

$$h_i^{(l+1)} = \left(\sum_{r \in R} \sum_{j \in N_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)} \right) \tag{4.1}$$

where $h_i^{(l)} \in R^d$ is the hidden representation of node v_i at layer l and N_i^r denotes the set of neighbor indices of node v_i under relation $r \in R$. This R-GCN operation implies that every relation r has its own linear transformation matrix W_r. Figure 4.2 shows how an update of a single graph entity embedding is calculated in the R-GCN model. The activations (d-dimensional vectors) from neighboring nodes (dark blue circles) are gathered and then transformed for each relation type individually (for both incoming and outgoing edges). The resulting representation is accumulated in a (normalized) sum and passed through a ReLu activation function. This per-node update is computed in parallel with shared parameters across the whole graph. We choose the task-dependent normalization constant $c_{i,r}$ to be $\frac{1}{|N_i^r|}$. σ is the activation function used at each layer. We use basis decomposition as an effective weight sharing technique at each convolution operation to regularize the weights of different relations.

$$W_r^{(l)} = \sum_{b=1}^{B} a_{rb}^{(l)} V_b^{(l)} \tag{4.2}$$

Using this regularization, only the coefficients $a_{rb}^{(l)}$ depend on the type of relation, while the transformation $V_b^l \in R^{d \times d}$ are shared across the layer. Our proposed model extends R-GCN's work using 7, 14, and 28 R-GCN layers with residual skip con-

nections and dense connections between the layers. The R-GCN model proposed by Schlichtkrull et al. [75] only employs two R-GCN layers with basis- and block-diagonal-decomposition regularization technique for shared weights.

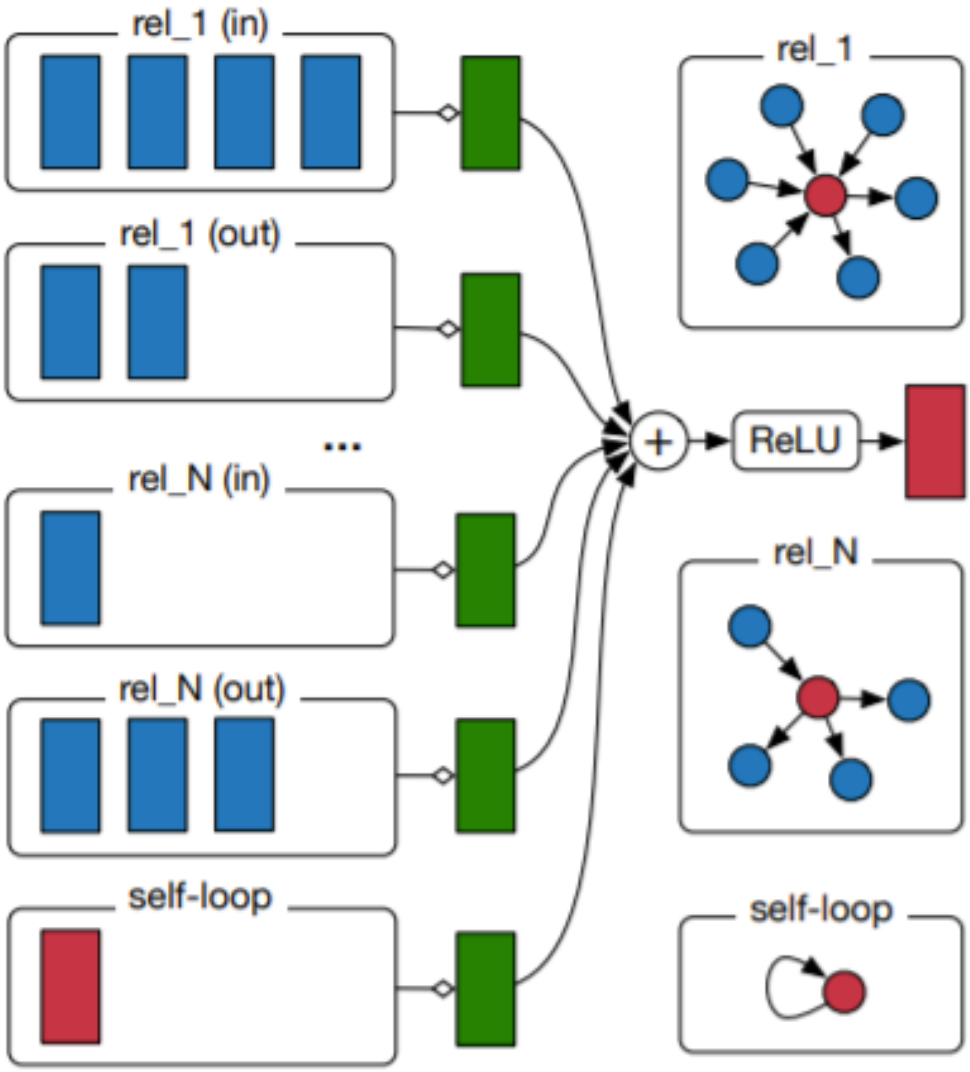


Figure 4.2: Feature update of a single entity in the network during an R-GCN operation [75]

4.1.2 Fusion Block

The fusion block is typically used to fuse the global and multi-scale local features. It takes as input the extracted vertex features from each of the convolution block's

R-GCN layers and concatenates them. These concatenated features are then passed through a 1 1 convolution layer followed by a global max-pooling layer. The global max-pooling layer is responsible for summarizing each node's features in the network aggressively. It aggregates the vertex features of the whole graph into a single global feature vector, which in return is concatenated with the feature of each vertex from all previous R-GCN layers, helping the fusion of global and local information.

4.1.3 MLP Prediction Block

Our proposed architecture's final component is a multi-layer perceptron that applies two MLP layers to each node's fused features in the network to predict its category. These MLP layers can be seen as 1 1 convolution operations on the features. For the entity classification task, we follow Schlichtkrull et al. 2017 [75], and Kipf and Welling et al. 2016 [41]; we use a softmax activation per node on the output of the final MLP layer for semi-supervised classification of entities. We ignore the unlabeled nodes while minimizing the following cross-entropy loss L using a negative log-likelihood function on the softmax output of all available labeled nodes.

$$L = \sum_{i \in V} \sum_{k=1}^{K} x_{ik} \log h_{ik}^{(L)} \tag{4.3}$$

Where V is the set of labeled nodes in the network, while h_{ik}^{L} is the k-th entry of the final layer output for the i-th labeled node and x_{ik} is the corresponding ground-truth label.

In the experiments with different architecture variants, we leave the fusion and MLP prediction blocks unchanged. The convolution block, however, varies in the number of R-GCN layers used and the type of (residual/dense) connections used in between the

layers (if any). We optimize our architecture while training by minimizing the cross-entropy loss on the output of the prediction block. We summarize the components of our system architecture, Deep-RGCN, with dense or residual skip connections. In the next chapter, we discuss the dataset details and specifications of the experiment setup.

Chapter 5

EXPERIMENTAL DESIGN

In this work, we consider the task of semi-supervised classification of entities in a knowledge base. To infer the type of an entity (e.g., person or company) in a network, a successful model needs to reason about its relations with other entities.

5.1 Datasets

We analyze our proposed frameworks, ResR-GCN and DenseR-GCN, on Resource Description Framework (RDF) format [72] benchmark datasets: AIFB and MUTAG. The relations in these datasets need not necessarily encode the presence, or absence, of a specific feature for a given entity. The exact statistics of the datasets used are shown in Table 5.1, which includes the number of entities, relation types, edges, and classes along with train/validation/test splits. In each of these datasets, the targets to be classified are properties of a group of entities represented as nodes.

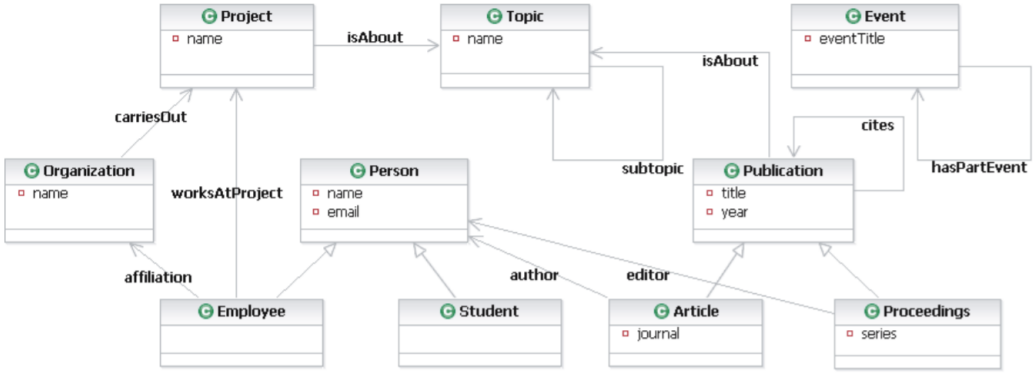


Figure 5.1: An ontology portion of an AIFB subgraph [6]

Table 5.1: Dataset statistics

Dataset	AIFB	MUTAG
Task	Transduction	Transduction
Entities	8285	23644
Relations	45	23
Edges	29043	74227
Avg. edges per node	3.50	3.13
Labeled nodes	176	340
Classes	4	2
Train nodes	112	217
Validation nodes	28	55
Test nodes	36	68

The AIFB dataset describes the AIFB research institute (Institute for Applied Informatics and Formal Description Methods) at the Karlsruhe Institute of Technology regarding its staff, their research interests, and contributions. This dataset describes the inter-relationships between persons (i.e., Professors, Students), research topics, projects, publications, affiliations, etc. Figure 5.1 shows the main classes and properties used in the semantic web for research communities ontology in AIFB. The node classification task is to predict the research group that any person is affiliated with. The MUTAG dataset is distributed as an example dataset for the DL-Learner toolkit. It contains information about complex molecules that are potentially carcinogenic, given by the *isMutagenic* property. Predicting whether a molecule in a molecular graph is carcinogenic or not is essentially a binary classification problem. It is important to note that MUTAG is a dataset for molecular graphs, which was later converted to RDF format, where relations either indicate atomic bonds or merely the presence of a particular feature. MUTAG is also a popular benchmark dataset for graph classification. Following the work of Schlichtkrull et al. 2017 [75], we remove the relations that were used to create entity labels: *employs* and *a liation* for AIFB and *isMutagenic* for MUTAG.

5.2 Baselines

As a baseline for our experiments, we compare the variants of our model against the original work of Schlichtkrull [75], RDF2Vec embeddings [73] and a more recent technique, Path Tree [85], that builds a decision tree of class-specific substructures to classify various entities within the knowledge graph. We also analyze the performances of ResR-GCN and DenseR-GCN concerning the number of hidden units used for convolution layers and present the results in the appendix section.

This chapter so far has outlined information about the AIFB and MUTAG benchmark datasets and baselines that we use to compare our results. The results obtained are discussed in detail in the next chapter.

Chapter 6

RESULTS

Our experiments analyze the effectiveness of node representations resulting from our proposed architecture for semi-supervised entity classification.

6.1 Semi-supervised Classi cation

Our entity classification approach is quite different from the formulation of Schlichtkrull et al. [75], in which a softmax activation per node is added directly on the output of the R-GCN layers, and cross-entropy loss is minimized. In our approach, fusion and prediction blocks are added after the final R-GCN layer, and cross-entropy loss is minimized on the softmax activation of the prediction block's output. The fusion block comprises a 1 1 convolution layer and a global max-pooling layer. The fusion block is responsible for concatenating all the vertex features resulted from each of the R-GCN layers. These concatenated features are then passed through the convolution and pooling layer to result in a single global vector for the prediction block. The output of the prediction block is passed through a softmax activation per node to predict their classes.

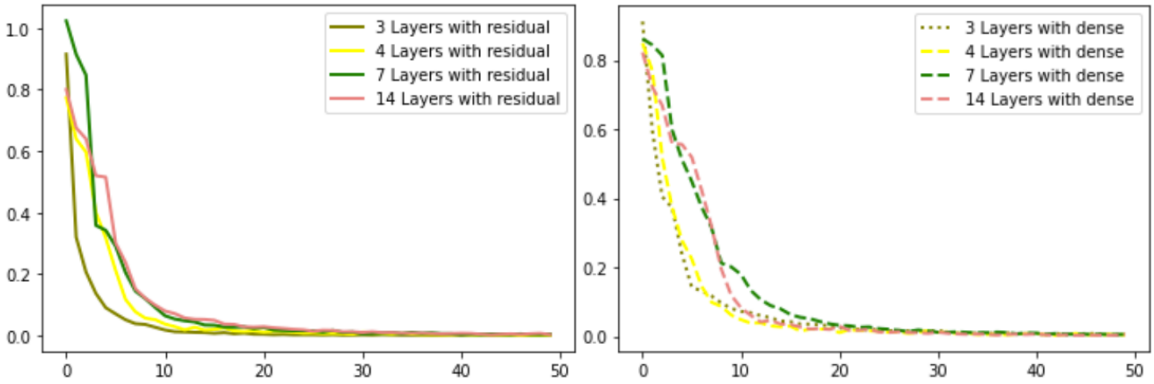
6.2 Implementation

We use the PyTorch Geometric [59] library in our implementation of the architecture and for retrieving the AIFB and MUTAG RDF datasets. We use the canonical train/test split provided in Ristoski et al. [72] while conducting the experiments

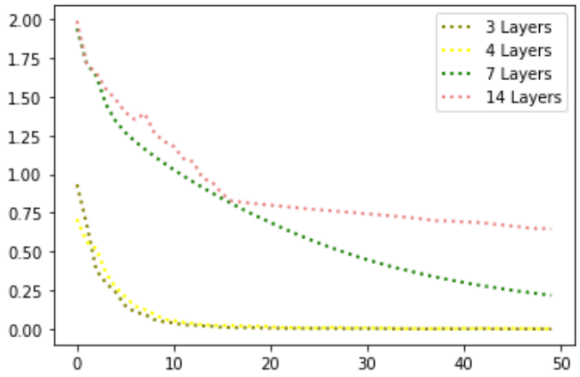
Table 6.1: Comparison of mean accuracies for 10 runs of proposed models with state-of-the-art baselines.

Model	AIFB		MUTAG	
RDF2Vec*	88.88	0.00	67.20	1.24
Path Tree*	89.44	2.08	73.82	5.61
R-GCN (Paper)*	95.83	0.62	73.23	0.48
R-GCN (Ours)	91.94	0.49	70.44	0.51
ResR-GCN-7	93.33	0.61	**74.56**	1.00
DenseR-GCN-7	**96.11**	0.74	**74.41**	0.48
ResR-GCN-14	95.28	0.72	**74.12**	1.00
DenseR-GCN-14	**96.11**	0.51	73.38	0.48

on both datasets. We use a 2-layer model with basis function decomposition and train with the Adam optimizer to implement R-GCN as in the original paper. We report the mean accuracy and standard error both from the original paper and our implementation using PyTorch Geometric library tools. Table 6.1 contains the entity classification results in accuracy (average and standard error over 10 runs) for RDF2Vec [73], Path Tree [85], R-GCN [75], and our proposed models (ResR-GCN and DenseR-GCN) with 7 and 14 layers. * represents the accuracies directly from published papers. **Bold** highlights models that outperform all state-of-the-art baselines for node property prediction. The hyperparameters for each dataset are identified by performing 5-fold cross-validation. The cross-validation experiments also show that the trained models generalize well. We use the Adam optimizer with a learning rate of 0.01 and a weight decay of $1e^{-5}$ for the MUTAG dataset. For the AIFB dataset, we observe better results without an $l2$-norm weight decay. Additionally, we use 30 transformations for basis decomposition for both datasets. The proposed networks are trained on NVIDIA Tesla V100 GPU (Google Colab Pro) with a batch size of 1. Furthermore, we use layer normalization for each of the R-GCN layers and a dropout of 0.2 at the prediction block's first MLP layer. In addition to studying the effect of residual and dense graph connections on AIFB and MUTAG graphs, we also investigate the influence of different hidden units (16, 32, 64) and layers (3,4,7,14).



(a) Residual (b) Dense



(c) No Residual/Dense

Figure 6.1: Training losses during 50 epochs for training on MUTAG using different architectures with (a) residual connections, (b) dense connections, (c) without any special connections

6.3 Discussion

Figure 6.1 presents the average training losses in 50 epochs for R-GCNs with different layers on the MUTAG dataset. We note how adding more layers translates to substantially higher loss (c). In contrast, using residual (a) and dense (b) connections alleviates this problem and results in consistent stability for all depths. The variants DenseR-GCN-7 and DenseR-GCN-14, with seven and fourteen densely connected R-GCN layers, respectively, in the backbone block, achieves an average accuracy of 96.11% on AIFB outperforming previous methods. Table 6.1 also shows that our models with dense and residual connections competitively perform compared to previous approaches. For unbiased comparison with original R-GCN [75], all the accuracies reported in the Table 6.1 are based on R-GCN layers with 16 hidden units. The MUTAG dataset experiments show improved accuracy over previous works, reaching up to 74.56% for a seven-layer architecture with residual connections.

These real-world graphs' enhanced accuracy represents that the trained model can make better predictions on unlabelled nodes. For instance, the labeled nodes in the AIFB dataset hold information about one out of four research groups. Using such a trained model, we could make predictions on unlabelled nodes with a level of confidence by modeling their neighbors and respective relation types. Similarly, for the MUTAG dataset, the architecture's goal is to learn to predict whether a molecule is a carcinogen or not by learning from the neighborhood information of labeled examples. In such a semi-supervised approach, the model trained on labeled data is used to predict labels on a batch of unlabeled nodes. Then these predicted labels are used to calculate the loss on unlabeled data. Finally, the labeled loss is combined with unlabeled loss and is backpropagated. Such a semi-supervised technique is also called pseudo-labeling.

We note a significant trade-off between a high number of layers and memory requirements for dense connections. For the graph datasets used in this work, the number of layers much higher than 14 requires very high GPU memory and more training time for the multi-layer feature concatenation operations. Because of this memory constraint, we limit the number of layers used to 14. Additionally, by employing K number of layers, the adequate context size for each node increases by the size of its K^{th}-order neighborhood with each additional layer. As a result, it may lead to overfitting with the increase in model parameters. Although the higher number of R-GCN layers shows promising effects on the AIFB and MUTAG datasets, such deep architectures may have memory limitations. R-GCN-based architectures with more than two layers may not be suitable for graphs with a very high number of nodes, like AM and BGS RDF graphs, because of higher memory requirements for training. We experimented on these large multi-relational graphs, AM and BGS, but couldn't complete the training with even three layers as they led to memory utilization errors. We consider not being able to scale our architectures to such large graphs with over a million entities as one of the significant limitations of deep-RGCNs. As discussed in the next chapter, parallel or shared memory allocation techniques can be explored as future work to overcome such memory issues.

Many previous works identified the set of problems caused in deep GCN architectures as over-smoothing and gradient vanishing problems. Over-smoothing is a phenomenon common in GCNs where after multi-layer graph convolution, Laplacian smoothing makes node representations more and more similar, eventually becoming indistinguishable. Some methods explore dynamic graph convolutions that allow the graph structures to change after each layer to alleviate over-smoothing. We find that the R-GCN operator's relation-specific transformations that depend on the type and direction of an edge automatically avoid the over-smoothing of features. Addition-

ally, the dense and residual skip connections reduce the vanishing gradient problem common in deep GCNs.

In this chapter, we've discussed our proposed architecture's performance and the positive results achieved by using residual and dense connections compared to deep R-GCN without any special skip connections.

Chapter 7

FUTURE WORK

As discussed in the previous sections, the R-GCN layers with more than two layers for graphs with a very high number of nodes and edges like AM and BGS requires a significant amount of memory for training. A potential direction for future work is to solve the memory problems caused in such large graphs or by dense connections in deep networks by implementing parallelized memory units or shared allocations [70] to store intermediate feature representations. By using shared memory allocations, the subsequent layers shall overwrite the intermediate results of previous layers. During back-propagation, these intermediate values can be re-calculated at the expense of computing time. This technique could allow a higher number of R-GCN layers with dense connections. Investigating the memory usages by replacing some dense blocks with residual blocks while keeping the feature dimensions compatible throughout the network could also help deal with the memory drawbacks of dense connections. Another interesting future work is to investigate the ability of node representations resulting from deep R-GCNs with dense and residual skip connections for link prediction and graph classification tasks. Deep GCNs are expected to become a powerful tool for processing graph-structured data in natural language processing, computer vision, and data mining. The form of summation over neighboring nodes and relation types can be replaced with a data-dependent attention mechanism like GATs Velickovic et al. [86] for better effectiveness. The number of layers of deep GCNs could be more of a personal choice depending upon the trade-off between training time, accuracy, and memory constraints.

Chapter 8

CONCLUSION

This thesis briefly reviews some of the existing techniques for modeling knowledge graphs and proposes a deep neural network architecture based on R-GCN layers, supplemented by a fusion and prediction block, for entity classification on multi-relational graphs: AIFB and MUTAG. Most proposed GCN architectures are shallow (less than four layers deep), as they are prone to over-smoothing and gradient vanishing problems. To alleviate these problems, we extend the popular techniques from CNN architectures, such as dense and residual skip connections, followed by adding a fusion and prediction block to the architecture. Our experiments show that the proposed architectures improve the accuracy over the two-layer R-GCN [75] and other baselines for node classification. We test the model's efficiency and generalization capability using a 5-fold cross-validation technique.

BIBLIOGRAPHY

[1] S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee. N-gcn: Multi-scale graph convolution for semi-supervised node classification. In *uncertainty in arti cial intelligence*, pages 841–851. PMLR, 2020.

[2] B. Angulo, Y. H. Roohani, and K. Shen. Graph embeddings of enriched protein-protein interaction (ppi) networks for identification of disease nodes. 2019.

[3] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. *arXiv preprint arXiv:1511.02136*, 2015.

[4] I. Balažević, C. Allen, and T. M. Hospedales. Hypernetwork knowledge graph embeddings. In *International Conference on Arti cial Neural Networks*, pages 553–565. Springer, 2019.

[5] I. Balažević, C. Allen, and T. M. Hospedales. Tucker: Tensor factorization for knowledge graph completion. *arXiv preprint arXiv:1901.09590*, 2019.

[6] S. Bloehdorn and Y. Sure. Kernel methods for mining instance data in ontologies. In *The Semantic Web*, pages 58–71. Springer, 2007.

[7] K. Bollacker, P. Tufts, T. Pierce, and R. Cook. A platform for scalable, collaborative, structured information integration. In *Intl. Workshop on Information Integration on the Web (IIWeb'07)*, pages 22–27, 2007.

[8] A. Bordes, X. Glorot, J. Weston, and Y. Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

[9] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenk o. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NIPS)*, pages 1–9, 2013.

[10] P. Branum and B. Sehon. Knowledge graph pilot improves data quality while providing a customer 360 view. In *Knowledge Graph Conference.(Invited talk)*, 2019.

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[12] S. Chang. Scaling knowledge access and retrieval at airbnb. airbnb medium blog., 2018. `https://medium.com/airbnb-engineering/scaling-knowledge-access-and-retrieval-at-airbnb-665b6ba21e95`.

[13] J. Chen, T. Ma, and C. Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[14] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735. PMLR, 2020.

[15] M. Cochez, P. Ristoski, S. P. Ponzetto, and H. Paulheim. Global rdf vector space embeddings. In *International Semantic Web Conference*, pages 190–207. Springer, 2017.

[16] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*, 2016.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[18] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI Conference on Arti cial Intelligence*, volume 32(1), 2018.

[19] D. Devarajan. Happy birthday watson discovery. ibm cloud blog., 2017. https://www.ibm.com/cloud/blog/announcements/happy-birthday-watson-discovery.

[20] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610, 2014.

[21] X. L. Dong. Building a broad knowledge graph for products. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 25–25. IEEE, 2019.

[22] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *arXiv preprint arXiv:1509.09292*, 2015.

[23] A. R. Ekpe Okorafor. The path from data to knowledge. accenture applied intelligence blog., 2019. https://www.accenture.com/us-en/insights/digital/data-to-knowledge.

[24] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[25] K. Fukushima. Neocognitron. *Scholarpedia*, 2(1):1717, 2007.

[26] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.

[27] S. Guo, Q. Wang, L. Wang, B. Wang, and L. Guo. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 192–202, 2016.

[28] Z. Guo, Y. Zhang, Z. Teng, and W. Lu. Densely connected graph convolutional networks for graph-to-sequence learning. *Transactions of the Association for Computational Linguistics*, 7:297–312, 2019.

[29] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.

[30] D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

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

[32] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[33] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[34] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. De Melo, and G. Weikum. Yago2: exploring and querying world knowledge in time, space, context, and many languages. In *Proceedings of the 20th international conference companion on World wide web*, pages 229–232, 2011.

[35] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres, et al. Knowledge graphs. *arXiv preprint arXiv:2003.02320*, 2020.

[36] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

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

[38] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao. Knowledge graph embedding via dynamic mapping matrix. 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)*, pages 687–696, 2015.

[39] S. M. Kazemi and D. Poole. Simple embedding for link prediction in knowledge graphs. *arXiv preprint arXiv:1802.04868*, 2018.

[40] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Arti cial Intelligence Review*, 53(8):5455–5516, 2020.

[41] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[42] J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[43] D. Koller, N. Friedman, S. Džeroski, C. Sutton, A. McCallum, A. Pfeffer, P. Abbeel, M.-F. Wong, C. Meek, J. Neville, et al. *Introduction to statistical relational learning*. MIT press, 2007.

[44] A. Krishnan. Making search easier: How amazon's product graph is helping customers find products more easily. amazon blog., 2018. `https://www.aboutamazon.com/news/innovation-at-amazon/making-search-easier`.

[45] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[46] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[47] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[48] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

[49] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109, 2018.

[50] G. Li, M. Müller, G. Qian, I. C. Delgadillo, A. Abualshour, A. Thabet, and B. Ghanem. Deepgcns: Making gcns go as deep as cnns. *arXiv preprint arXiv:1910.06849*, 2019.

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

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

[53] R. Li, S. Wang, F. Zhu, and J. Huang. Adaptive graph convolutional neural networks. In *Proceedings of the AAAI Conference on Arti cial Intelligence*, volume 32, 2018.

[54] R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. *arXiv preprint arXiv:1901.01484*, 2019.

[55] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the AAAI Conference on Arti cial Intelligence*, volume 29(1), 2015.

[56] E. Meij. Understanding news using the bloomberg knowledge graph. *Invited talk at the Big Data Innovators Gathering (TheWebConf).*, 2019.

[57] A. Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[58] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[59] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017.

[60] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[61] Y. C. Ng, N. Colombo, and R. Silva. Bayesian semi-supervised learning with graph gaussian processes. *arXiv preprint arXiv:1809.04379*, 2018.

[62] M. Nickel, L. Rosasco, and T. Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the AAAI Conference on Arti cial Intelligence*, volume 30, 2016.

[63] M. Nickel and V. Tresp. Tensor factorization for multi-relational learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 617–621. Springer, 2013.

[64] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor. Industry-scale knowledge graphs: lessons and challenges. *Communications of the ACM*, 62(8):36–43, 2019.

[65] K. O'Shea and R. Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[66] M. PALMONARI and P. MINERVINI. Knowledge graph embeddings and explainable ai. *Knowledge Graphs for Explainable Arti cial Intelligence: Foundations, Applications and Challenges*, 47:49, 2020.

[67] H. Paulheim and J. Fümkranz. Unsupervised generation of data mining features from linked open data. In *Proceedings of the 2nd international conference on web intelligence, mining and semantics*, pages 1–12, 2012.

[68] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[69] R. Pittman, A. Srivastava, S. Hewavitharana, A. Kale, and S. Mansour. Cracking the code on conversational commerce. ebay blog. *Library Catalog: www. ebayinc. com. 6th Apr*, 2017.

[70] G. Pleiss, D. Chen, G. Huang, T. Li, L. van der Maaten, and K. Q. Weinberger. Memory-efficient implementation of densenets. *arXiv preprint arXiv:1707.06990*, 2017.

[71] D. A. Qi He, Bee-Chung Chen. Building the linkedin knowledge graph. linkedin blog., 2016. `https://engineering.linkedin.com/blog/2016/10/building-the-linkedin-knowledge-graph`.

[72] P. Ristoski, G. K. D. De Vries, and H. Paulheim. A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In *International Semantic Web Conference*, pages 186–194. Springer, 2016.

[73] P. Ristoski and H. Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *International Semantic Web Conference*, pages 498–514. Springer, 2016.

[74] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[75] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[76] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[77] S. Shrivastava. Bring rich knowledge of people, places, things and local businesses to your apps. bing blogs., 2017. https://blogs.bing.com/search-quality-insights/2017-07/bring-rich-knowledge-of-people-places-things-and-local-businesses-to-your-apps.

[78] A. Singhal. Introducing the knowledge graph: things, not strings. google blog., 2012. https://www.blog.google/products/search/introducing-knowledge-graph-things-not/.

[79] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934. Citeseer, 2013.

[80] Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.

[81] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

[82] F. Tobin. Thomson reuters launches first of its kind knowledge graph feed allowing financial services customers to accelerate their ai and digital strategies. thomspon reuters press release, 2017. https://www.thomsonreuters.com/en/press-releases/2017/october/thomson-reuters-launches-first-of-its-kind-knowledge-graph-feed.html.

[83] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pages 2071–2080. PMLR, 2016.

[84] S.-H. Tsang. Review: Densenet—dense convolutional network (image classification). *en l nea].[consulta: 7 abril 2019]. Disponible en: https://towardsdatascience. com/review-densenetimage-classi cation-b6631a8ef803*, 2018.

[85] G. Vandewiele, B. Steenwinckel, F. Ongenae, and F. De Turck. Inducing a decision tree with discriminative paths to classify entities in a knowledge graph. In *SEPDA2019, the 4th International Workshop on Semantics-Powered Data Mining and Analytics*, volume 2427, pages 1–6, 2019.

[86] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[87] D. Vrandečić and M. Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[88] L. Wang, Y. Huang, Y. Hou, S. Zhang, and J. Shan. Graph attention convolution for point cloud semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10296–10305, 2019.

[89] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, et al. Deep graph library: Towards efficient and scalable deep learning on graphs. 2019.

[90] W. Wang, J. Liu, T. Tang, S. Tuarob, F. Xia, Z. Gong, and I. King. Attributed collaboration network embedding for academic relationship mining. *ACM Transactions on the Web (TWEB)*, 15(1):1–20, 2020.

[91] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[92] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI Conference on Arti cial Intelligence*, volume 28, 2014.

[93] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[94] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

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

[96] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294*, 2018.

[97] C. Zhuang and Q. Ma. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*, pages 499–508, 2018.

APPENDICES

Appendix A

ABLATION STUDY WITH DIFFERENT NUMBER OF HIDDEN UNITS

We also study the models' behavior for different hidden units' values (filters) used at each R-GCN layer. These performances for AIFB and MUTAG datasets are reported as accuracies in Table A.1 and Table A.2 respectively. A recent work by Li et al. [50], that proposed a deep architecture with max-relative graph convolution networks for node representation learning tasks, found that the network's width positively correlates with the network performance on the PPI node classification task. We find that residual and dense connections can help the deep networks converge well compared to the same model without any connections. Although we find that the higher the number of layers with such special connections results in better performance (accuracy), we could not determine a correlation between the network width and performance measured in accuracy for the node classification task. **Bold** highlights models that outperform all state-of-the-art baselines for node property prediction. Note that the '-' in Table A.2 denotes models that are not applicable due to memory limitations.

Table A.1: Ablation study on graph connections comparing mean accuracies obtained with di erent network widths on AIFB dataset.

Connections	# of filters	16	32	64
No Connections	PlainR-GCN-2	91.94	92.78	92.50
	PlainR-GCN-3	92.78	93.61	92.22
	PlainR-GCN-4	92.5	93.89	90.00
	PlainR-GCN-7	84.72	83.89	84.51
	PlainR-GCN-14	78.33	83.34	77.77
Residual	ResR-GCN-3	92.78	93.33	93.61
	ResR-GCN-4	92.78	93.61	93.89
	ResR-GCN-7	93.33	94.44	95.00
	ResR-GCN-14	95.28	95.56	**96.11**
Dense	DenseR-GCN-3	95.28	94.17	92.22
	DenseR-GCN-4	93.89	94.72	93.61
	DenseR-GCN-7	**96.11**	95.83	**96.11**
	DenseR-GCN-14	**96.11**	95.00	93.89

Table A.2: Ablation study on graph connections comparing mean accuracies obtained with di erent network widths on MUTAG dataset.

Connections	# of filters	16	32	64
No Connections	PlainR-GCN-2	70.44	69.56	67.94
	PlainR-GCN-3	71.91	68.24	69.56
	PlainR-GCN-4	69.56	70.44	68.24
	PlainR-GCN-7	65.15	66.34	66.62
	PlainR-GCN-14	64.41	65.29	65.88
Residual	ResR-GCN-3	71.18	71.18	71.62
	ResR-GCN-4	71.47	71.18	71.91
	ResR-GCN-7	**74.56**	72.21	73.38
	ResR-GCN-14	**74.12**	73.24	**75.15**
Dense	DenseR-GCN-3	70.88	72.21	72.06
	DenseR-GCN-4	71.03	70.88	71.62
	DenseR-GCN-7	**74.41**	72.35	73.24
	DenseR-GCN-14	73.83	-	-