-

NANYANG TECHNOLOGICAL UNIVERSITY

CZ4071 Network Science

Project 2 – Modelling Relational Data with Graph Convolutional Networks

Chai Yinn (U1821733B)

Glendon Thaiw (U1821713F)

Jervis Chan (U1821352H)

Larry Lee (U1821668J)

1. Problem Studied in Paper

In this paper, the authors study and tackle the problem of incomplete knowledge graphs. Knowledge graphs have been widely used in artificial intelligence, including natural language processing and information retrieval. Knowledge graphs are open in nature, implying that they are incomplete and have defects. Despite much manpower dedicated to maintaining knowledge bases, even the largest such as DBPedia, have missing information which affects their practical utilisations and downstream applications.

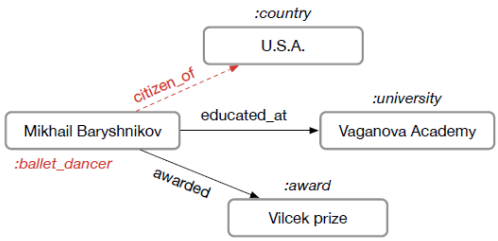

Figure 1. Knowledge Graph Fragment

Figure 1 illustrates an example of missing information within the knowledge bases. The node label and edge depicted in red are instances of incomplete information that can be predicted. Knowledge graphs store nodes and edges as triples in the form (subject, relation, object).

The missing pieces of information can be derived through the relations of its neighbouring nodes. For instance, if Mikhail Baryshnikov attended the Vaganova Academy, it can be predicted that the Mikhail Baryshnikov node can be labelled as a ballet dancer and that the triple (Mikhail Baryshnikov, citizen of, USA) should exist within the graph.

Two fundamental tasks can be used to increase completeness of knowledge graphs: Entity classification and link prediction.

The authors of the paper tackle the problem of retrieving such missing information through the development of new algorithms for entity classification and link prediction using relational Graph Convolutional Networks (rGCN).

2. Challenges

In recent years, there has been much attention given to the task of completing knowledge graphs. For entity classification, while previous works have seen success in entity classification for more generic types such as ‘Person’ and ‘Dog’, there is still a gap in more specific entity classification types. Entity classification for specific types is more challenging as it requires more learned and concise information.

The link prediction problem has also been previously explored, with many techniques such as decomposition-based and path-based methods proposed. However, most of these methods involve too many parameters, resulting in high computational complexity thus poor efficiency and scalability. With the continuous and dynamic expansion of knowledge graphs, the challenge of the link prediction problem is to derive an algorithm that will be able to run efficiently on evergrowing graph sizes, while maintaining accuracy.

3. Real-World Applications

Entity classification can help to identify categories that objects fall into. It can segment an entity into classes and subclasses. For instance, entity classification can help to determine the

taxonomic rank in biological classification. Link prediction can be used to predict missing links of incomplete data, such as completing interactions between chemicals and proteins and their effects. Link prediction can also be used to predict future possible links in knowledge graphs. For instance, it can predict if 2 people on Facebook know each other, and increase the accuracy of friend recommendations.

In general, increasing the completeness of knowledge graphs will enhance many uses and applications. Knowledge graphs are currently used for applications such as data governance, automatic fraud detection and insider trading. With a more complete knowledge graph, the applications can extend to uses such as enabling venture capital funds to determine new investment opportunities. They require a more complete knowledge graph that presents more robust historical information of startup data, deals, trends and funds, to predict new opportunities for portfolio optimisation.

4. Contributions Made in Paper

1) The authors are the first to implement rGCNs on modelling relational multi-graphs for the purpose of entity classification and link prediction.

2) The authors developed 2 regularisation techniques to reduce the number of parameters derived from rGCN to alleviate overfitting of data.

3) Tensor factorisation models such as DistMult, can be improved by passing in rGCN node representations instead of a single, real-valued vector.

5. Related Work

The paper's rGCN is an extension of the Graph Convolutional Network (GCN) model derived from *Semi-supervised Classification Using Graph Convolutional Networks (Kipf and Welling, 2017)*. The original GCN model was built for local graph neighbourhoods, and the authors adapted it to run on large-scale relational knowledge graphs.

$$h_i^{(l+1)} = \sigma\left(\sum_{m \in \mathcal{M}_i} g_m(h_i^{(l)}, h_j^{(l)})\right)$$

Equation 1. Simple Message Passing Formula

Equation 1 is the simple message passing equation for neural networks, where the representation of a node after one message pass is made up of the sums of the node's neighbours. In *Kipf and Welling, 2017*, g_m was chosen to be a simple weight multiplication.

In this paper, the authors adapt this equation for a relational multi-graph.

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

Equation 2. Propagation Model for Relational Multi-graph

Instead of only taking into account the sum of node's neighbours in Equation 1, Equation 2 takes into account incoming and outgoing edges and the relation type. In the simple model, the feature vectors of the node's neighbours are summed but not the node itself. In order to ensure the node at layer l +1 is informed by the representation from layer l, a self-loop with a special relation type is added to the node itself in Equation 2.

The authors of the paper evaluate Equation 2 in parallel for each node to form a layer. By stacking multiple layers, the original simple message model is transformed into a rGCN.

In the model used for link prediction, the authors utilise DistMult from *Embedding Entities and Relations for Learning and Inference in Knowledge Bases (Yang et al., 2014)* as the decoder in the encoder-decoder approach. DistMult acts as a scoring function to predict whether a relation holds through Equation 3.

$$f(s, r, o) = e_s^T R_r e_o \,.$$

Equation 3. DistMult Scoring Function

The authors use the scoring function with the rGCN node representations instead of the parameter vectors to improve and compare the performance of their model. DistMult was chosen as it performs well on standard link prediction benchmarks and is used as a baseline for triple scores in the link prediction experiments.

6. Algorithms and Techniques

Firstly, the authors introduced techniques to adapt GCNs to be used on relational multi-graphs. As aforementioned, based on Equation 1, the authors adapted the simple messaging model of GCNs to rGCN as seen in Equation 2. They combined the network layers to form a rGCN model as shown below.

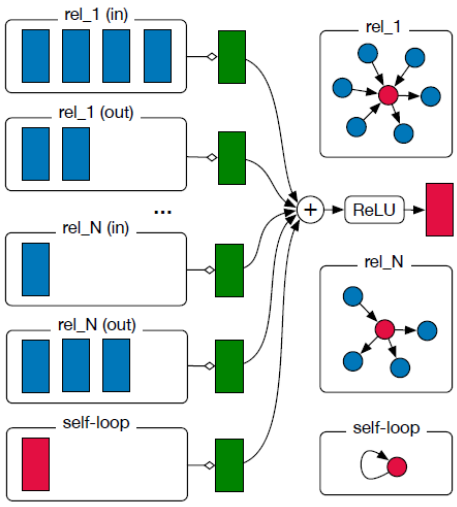

Figure 2. rGCN Model

Node representations are accumulated in parallel and passed to a ReLU activation function.
In addition to adapting the simple model to Equation 2, the authors developed 2 regularisation techniques for the rGCN. Applying Equation 2 on relational multi-graphs would result in the number of parameters growing fast with the size of the graph, which can result in overfitting. To overcome this problem, the authors proposed 2 techniques to regularise the weights of each network layer: basis-diagonal decomposition and block-diagonal decomposition.

6.1 Basis-diagonal Decomposition

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

Equation 4. Basis-diagonal Decomposition Formula

This equation decomposes the weight matrix from equation y into a linear combination of basis transformations, to enable effective weight sharing amongst relations with different types. Parameter updates are thus shared amongst frequent and rare relations, which reduces the chances of overfitting on rare relations.

6.2 Block-diagonal decomposition

$$W_r^{(l)} = \bigoplus_{b=1}^{B} Q_{br}^{(l)}.$$

Equation 5. Block-diagonal Decomposition Formula

A similar decomposition is applied for block-diagonal decomposition with matrices instead of basis vectors. It behaves as a sparsity constraint for relation types, resulting in latent features being more tightly coupled within sets of variables instead of across sets.
Both of these regularisation techniques enable the number of parameters to be lowered.

Secondly, the authors of the paper developed 2 algorithms that applied the rGCN framework onto entity classification and link prediction.

6.3 Entity Classification

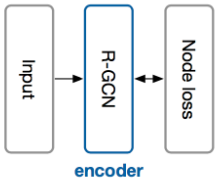

Figure 3. Entity Classification Framework

$$\mathcal{L} = -\sum_{i \in \mathcal{Y}} \sum_{k=1}^{K} t_{ik} \ln h_{ik}^{(L)}$$

Equation 6. Entity Classification Loss Formula

The rGCN model is run on the knowledge graph, after which a softmax classifier is used for each node on the output of the final layer. The softmax classifiers take in the rGCN output to predict the entity type labels for the nodes. The loss on labeled nodes are then evaluated and minimised using Equation 6, which are then used to train the model using gradient descent techniques.

6.4 Link Prediction

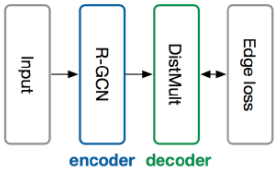

Figure 4. Link Prediction Framework

$$\mathcal{L} = -\frac{1}{(1+\omega)|\hat{\mathcal{E}}|} \sum_{(s,r,o,y) \in \mathcal{T}} y \log l\big(f(s,r,o)\big) + (1-y) \log\big(1 - l\big(f(s,r,o)\big)\big),$$

Equation 7. Link Prediction Loss Formula

The model built for link prediction uses an autoencoder. The encoder is the rGCN which produces the node representations, and the decoder is composed of the tensor factorisation model DistMult. The DistMult scoring function is used with the rGCN node representations to score triples. Negative sampling is used to randomly corrupt the object or subject to construct ω negative samples in order to train the model. Cross-entropy loss is minimised through Equation 7, to give the negative triples a lower score than the observable triples.

7. Connections with CZ4071

The knowledge graph introduced in the paper is an example of an uncertain graph learnt from the lecture. The basis of link prediction is determining edge existence probability and strength based on attributes, and entity classification is determining unknown attribute values.

From CZ4071, we learnt about Spectral Graph Partitioning, which utilises eigenvectors and eigenvalues of a matrix to minimise edges cut in the partitioning of the graph. These eigenvector and eigenvalue computations are very expensive, costing $O(N^3)$, especially for large knowledge-based graphs introduced in the paper. The paper introduces a method to reduce the computational cost of partitioning/clustering the graph by using a simplified first-order approximation of Spectral Graph Convolutions in a rGCN.

$$g_\theta \star x = U g_\theta U^\top x \,,$$

Equation 8. Original Equation for Spectral Graph Convolutions

Hammond et al. (2011) suggested that $g_\theta(\Lambda)$ can be well-approximated by Chebyshev polynomials up to K^{th} order.

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}) \,,$$

Equation 9. Chebyshev Polynomials

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L}) x \,,$$

Equation 10. New Equation After Substitution

After substituting with Chebyshev polynomials, the GCN becomes dependent on its K^{th} order neighbourhood. This reduces the complexity of evaluating the Spectral Convolution, reducing it to a linear computation, costing $O(|E|)$ and proportional to the number of edges. This algorithm introduced by the paper can be used to reduce computational complexity of Spectral Graph Partitioning by limiting the expression to maximum K steps away from the central node instead of calculating all nodes.

In CZ4071, Multilevel Graph Partitioning was introduced to reduce the Spectral Graph Partitioning computation costs. Multilevel Graph Partitioning is used to first coarsen the graph before implementing Spectral Graph Partitioning then uncoarsen it. This technique learnt can potentially be applied to the paper's algorithm to make the graph clustering even more efficient. The knowledge-based graph can be coarsened via maximal matching before the implementation of the rGCN, to reduce the number of nodes by at most half. This can help to reduce the time complexity of the clustering algorithm.

In the lectures, graph summary via compression is introduced. Nodes with similar neighbourhoods are collapsed into supernodes/clusters with edge corrections present. This paper utilises GCNs to form clusters. The N-layer GCN performs N propagation steps that combine the K^{th} order neighbourhood of each node. With labelling and multiple iterations, the GCN produces well-defined clusters.

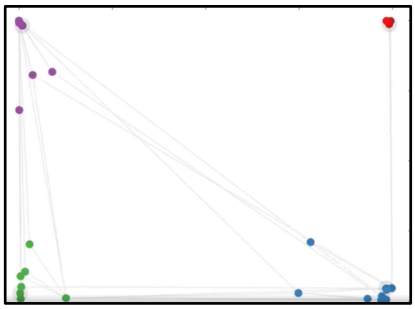

Figure 5. Clustering Derived from GCN

The GCN manages to form clusters similar to actual clusters indicated by colours. To make the clustering from lectures more robust, GCNs can be utilised to increase accuracy.

From the results of the paper's algorithm, the model tends to perform better for nodes with high degree and with a lot of contextual information. The paper's train-test set is randomly sampled and negatively sampled. The BFS-based sampling from the lecture could be used to derive the training set as it favours picking high degree nodes, which could potentially improve the performance of the model.

8. Implementation of Paper

As mentioned above, this paper aims to solve the problems of entity (node) classification and link prediction. In our implementation, we will be looking specifically at the problem of node classification. In the paper, the R-GCN model was used to perform node classification on the AIFB, MUTAG, BGS and AM datasets. The dataset was split into training and validation sets (in the ratio of 80% to 20%) and the accuracy in classifying the nodes in the validation sets was collected. The average accuracy over 10 rounds achieved by the paper is compared with other baseline models, such as Feat, WL and RDF2Vec, and the results are shown below. This paper is especially significant as the proposed R-GCN model performed better than the baselines and remains the best in node classification of AIFB and AM.

Model	AIFB	MUTAG	BGS	AM
Feat	55.55	77.94	72.41	66.66
WL	80.55	**80.88**	86.20	87.37
RDF2Vec	88.88	67.20	**87.24**	88.33
R-GCN	**95.83**	73.23	83.10	**89.29**

Figure 6. Experimental Results from Paper

We will implement the R-GCN model with StellarGraph library and try to replicate it on the AIFB dataset. Additionally, we will experiment on new datasets, the CORA dataset.

Implementation Details Used in Paper

Implementation Detail	**Value**
Number of Layers	2 layers (with 16 hidden units)
Num_bases	0 for AIFB, but B ∈ {0, 10, 20, 30, 40} based on validation set performance
Optimizer	Adam
Loss Function	Cross Entropy Loss
Number of Epochs	50
Learning Rate	0.01

Implementation Code

```
# 1.Loading Data
dataset = datasets.AIFB()
G, affiliation = dataset.load()

# 2.Input Preparation
train_targets, test_targets = model_selection.train_test_split(
    affiliation, train_size=0.8, test_size=0.2
)
generator = RelationalFullBatchNodeGenerator(G, sparse=True)
train_gen = generator.flow(train_targets.index, targets=train_targets)
test_gen = generator.flow(test_targets.index, targets=test_targets)

# 3.RGCN model creation and training
rgcn = RGCN(
    layer_sizes=[16, 16],
    activations=["relu", "relu"],
    generator=generator,
    bias=True,
    num_bases=0,
    dropout=0.5,
)
x_in, x_out = rgcn.in_out_tensors()
predictions = Dense(train_targets.shape[-1], activation="softmax")(x_out)
model = Model(inputs=x_in, outputs=predictions)
model.compile(
    loss="categorical_crossentropy",
    optimizer=keras.optimizers.Adam(0.01),
    metrics=["acc"],
)

history = model.fit(train_gen, validation_data=test_gen, epochs=50)

# 4.Evaluate
test_metrics = model.evaluate(test_gen)
print("\nTest Set Metrics:")
for name, val in zip(model.metrics_names, test_metrics):
    print("\t{}: {:0.4f}".format(name, val))
```

Figure 7. Code Snippet of R-GCN Implementation

As seen from the code snippet (Figure 7), the implementation is broken down into the following steps:

1. **Loading Data**
   The StellarGraph Library is inbuilt with commonly used datasets for graph problems. This includes the AIFB dataset and the CORA dataset. We will load the dataset, which will return us a Graph object and the labels used for classifying nodes.
2. **Input Preparation**
   We will split the labels into training and validation sets in the ratio of 80:20. Additionally, we put the graph through the StellarGraph RelationalFullBatchNode Generator which will supply the features array and the adjacency matrix to a full-batch Keras graph ML model.
   R-GCN Model Creation and Training
   We use StellarGraph to create a RGCN object. Here, we specify the number of RGCN layers and the num_bases used. Next, we add a softmax layer to transform the features created by the RGCN object into class predictions which we feed into a Keras Model. Lastly, we train the model on the generators.
3. **Evaluate**
   We evaluate the trained model on the validation set and calculate accuracy. This is done ten times and the average accuracy is recorded.

Results

The results from the implementations are shown below.

Model	AIFB	CORA
R-GCN	95.83	-
Implemented R-GCN	94.44	88.01

The implemented R-GCN model was able to achieve similar accuracy results as the paper. Additionally, the model was able to achieve 88.01% on the CORA dataset. This is a great result considering that other state-of-the-art models such as SSP achieved accuracy rates of 90.16%.