End-to-End Structure-Aware Convolutional Networks for Knowledge Base Completion

Chao Shang,1∗ Yun Tang,2 Jing Huang,2 Jinbo Bi,1 Xiaodong He,2 Bowen Zhou2
1Department of Computer Science and Engineering, University of Connecticut, Storrs, CT, USA
2JD AI Research, Mountain View, CA, USA
{chao.shang, jinbo.bi}@uconn.edu, {yun.tang, jing.huang, xiaodong.he, bowen.zhou}@jd.com

Abstract

Knowledge graph embedding has been an active research topic for knowledge base completion, with progressive improvement from the initial TransE, TransH, DistMult et al to the current state-of-the-art ConvE. ConvE uses 2D convolution over embeddings and multiple layers of nonlinear features to model knowledge graphs. The model can be efficiently trained and scalable to large knowledge graphs. However, there is no structure enforcement in the embedding space of ConvE. The recent graph convolutional network (GCN) provides another way of learning graph node embedding by successfully utilizing graph connectivity structure. In this work, we propose a novel end-to-end Structure-Aware Convolutional Network (SACN) that takes the benefit of GCN and ConvE together. SACN consists of an encoder of a weighted graph convolutional network (WGCN), and a decoder of a convolutional network called Conv-TransE. WGCN utilizes knowledge graph node structure, node attributes and edge relation types. It has learnable weights that adapt the amount of information from neighbors used in local aggregation, leading to more accurate embeddings of graph nodes. Node attributes in the graph are represented as additional nodes in the WGCN. The decoder Conv-TransE enables the state-of-the-art ConvE to be translational between entities and relations while keeps the same link prediction performance as ConvE. We demonstrate the effectiveness of the proposed SACN on standard FB15k-237 and WN18RR datasets, and it gives about 10% relative improvement over the state-of-the-art ConvE in terms of HITS@1, HITS@3 and HITS@10.

Introduction

Over the recent years, large-scale knowledge bases (KBs), such as Freebase (Bollacker et al. 2008), DBpedia (Auer et al. 2007), NELL (Carlson et al. 2010) and YAGO3 (Mahdisoltani, Biega, and Suchanek 2013), have been built to store structured information about common facts. KBs are multi-relational graphs whose nodes represent entities and edges represent relationships between entities, and the edges are labeled with different relations. The relationships are organized in the forms of (s, r, o) triplets (e.g. entity s = Abraham Lincoln, relation r = DateOfBirth, entity o = 02-12-1809). These KBs are extensively used for web search, recommendation and question answering. Although these KBs have already contained millions of entities and triplets, they are far from complete compared to existing facts and newly added knowledge of the real world. Therefore knowledge base completion is important in order to predict new triplets based on existing ones and thus further expand KBs.

One of the recent active research areas for knowledge base completion is knowledge graph embedding: it encodes the semantics of entities and relations in a continuous low-dimensional vector space (called embeddings). These embeddings are then used for predicting new relations. Started from a simple and effective approach called TransE (Bordes et al. 2013), many knowledge graph embedding methods have been proposed, such as TransH (Wang et al. 2014), TransR (Lin et al. 2015), DistMult (Yang et al. 2014), TransD ( Ji et al. 2015), ComplEx (Trouillon et al. 2016), STTransE (Nguyen et al. 2016). Some surveys (Nguyen 2017; Wang et al. 2017) give details and comparisons of these methods.

The most recent ConvE (Dettmers et al. 2017) model uses 2D convolution over embeddings and multiple layers of nonlinear features, and achieves the state-of-the-art performance on common benchmark datasets for knowledge graph link prediction. In ConvE, the embeddings of s and r are reshaped and concatenated into an input matrix and fed to the convolution layer. Convolutional filters of n × n are used to output feature maps that are across different dimensional embedding entries. Thus ConvE does not keep the translational property as TransE which is an additive embedding vector operation: e_s + e_r ≈ e_o ((Nguyen et al. 2017)). In this paper, we remove the reshape step of ConvE and operate convolutional filters directly in the same dimensions of s and r. This modification gives better performance compared with the original ConvE, and has an intuitive interpretation which keeps the global learning metric the same for s, r, and o in an embedding triple (e_s, e_r, e_o). We name this embedding as Conv-TransE.

ConvE also does not incorporate connectivity structure in the knowledge graph into the embedding space. In contrast, graph convolutional network (GCN) has been an effective tool to create node embeddings which aggregate local information in the graph neighborhood for each node (Kipf and Welling 2016a; Hamilton, Ying, and Leskovec 2017a; Kipf and Welling 2016c; Pham et al. 2017; Shang et al. 2018).
GCN models have additional benefits (Hamilton, Ying, and Leskovec 2017b), such as leveraging the attributes associated with nodes. They can also impose the same aggregation scheme when computing the convolution for each node, which can be considered a method of regularization, and improves efficiency. Although scalability is originally an issue for GCN models, the latest data-efficient GCN, PinSage (Ying et al. 2018), is able to handle billions of nodes and edges.

In this paper, we propose an end-to-end graph Structure-Aware Convolutional Networks (SACN) that take all benefits of GCN and ConvE together. SACN consists of an encoder of a weighted graph convolutional network (WGCN), and a decoder of a convolutional network called Conv-TransE. WGCN utilizes knowledge graph node structure, node attributes and relation types. It has learnable weights to determine the amount of information from neighbors used in local aggregation, leading to more accurate embeddings of graph nodes. Node attributes are added to WGCN as additional for easy integration. The output of the decoder Conv-TransE becomes the input of the decoder Conv-TransE. Conv-TransE is similar to ConvE but with the difference that Conv-TransE keeps the translational characteristic between entities and relations. We show that Conv-TransE performs better than ConvE, and our SACN improves further on top of Conv-TransE in the standard benchmark datasets. The code for our model and experiments is publicly available 

Our contributions are summarized as follows:

• We present an end-to-end network learning framework SACN that takes benefit of both GCN and Conv-TransE. The encoder GCN model leverages graph structure and attributes of graph nodes. The decoder Conv-TransE simplifies ConvE with special convolutions and keeps the translational property of TransE and the prediction performance of ConvE;

• We demonstrate the effectiveness of our proposed SACN on the standard FB15k-237 and WN18RR datasets, and show about 10% relative improvement over the state-of-the-art ConvE in terms of HITS@1, HITS@3 and HITS@10.

Related Work

Knowledge graph embedding learning has been an active research area with applications directly in knowledge base completion (i.e. link prediction) and relation extractions. TransE (Bordes et al. 2013) started this line of work by projecting both entities and relations into the same embedding vector space, with translational constraint of \( e_s + e_r \approx e_o \). Later works enhanced KG embedding models such as TransH (Wang et al. 2014), TransR (Lin et al. 2015), and TransD (Ji et al. 2015) introduced new representations of relational translation and thus increased model complexity. These models were categorized as translational distance models (Wang et al. 2017) or additive models, while DistMult (Yang et al. 2014) and ComplEx (Trouillon et al. 2016) are multiplicative models (Sharma, Talukdar, and others 2018), due to the multiplicative score functions used for computing entity-relation-entity triplet likelihood.

The most recent KG embedding models are ConvE (Dettmers et al. 2017) and ConvKB (Nguyen et al. 2017). ConvE was the first model using 2D convolutions over embeddings of different embedding dimensions, with the hope of extracting more feature interactions. ConvKB replaced 2D convolutions in ConvE with 1D convolutions, which constrains the convolutions to be the same embedding dimensions and keeps the translational property of TransE. ConvKB can be considered as a special case of Conv-TransE that only uses filters with width equal to 1. Although ConvKB was shown to be better than ConvE, the results on two datasets (FB15k-237 and WN18RR) were not consistent, so we leave these results out of our comparison table. The other major difference of ConvE and ConvKB is on the loss functions used in the models. ConvE used the cross-entropy loss that could be sped up with 1-N scoring in the decoder, while ConvKB used a hinge loss that was computed from positive examples and sampled negative examples. We take the decoder from ConvE because we can easily integrate the encoder of GCN and the decoder of ConvE into an end-to-end training framework, while ConvKB is not suitable for our approach.

These embedding models achieved good performance for knowledge base completion in terms of efficiency and scalability. However, these approaches only modeled relational triplets, while ignoring a large number of attributes associated with graph nodes, e.g., ages of people or release region of music. Furthermore, these models do not enforce any large-scale connectivity structure in the embedding space, and totally ignore the knowledge graph structure. The proposed (SACN) handles these two problems in an end-to-end training framework, by using a variant of graph convolutional network (GCN) as the encoder, and a variant of ConvE as the decoder.

GCNs were first proposed in (Bruna et al. 2013) where graph convolutional operations were defined in the Fourier domain. The eigendecomposition of the graph Laplacian caused intense computation. Later, smooth parametric spectral filters (Henaff, Bruna, and LeCun 2015; Defferrard, Bresson, and Vandergheynst 2016) were introduced to achieve localization in the spatial domain and improve computational efficiency. Recently, Kipf et al. (Kipf and Welling 2016b) simplified these spectral methods by a first-order approximation with the Chebyshev polynomials. The spatial graph convolution approaches (Hamilton, Ying, and Leskovec 2017a) define convolutions directly on graph, which sum up node features over all spatial neighbors using adjacency matrix.

GCN models were mostly criticized for its huge memory requirement to scale to massive graphs. However, (Ying et al. 2018) developed a data efficient GCN algorithm called PinSage, which combined efficient random walks and graph convolutions to generate embeddings of nodes that incorporated both graph structure as well as node features. The experiments on Pinterest data were the largest application of deep graph embeddings to date with 3 billion nodes and

1https://github.com/JD-AI-Research-Silicon-Valley/SACN
WGCN layers builds an entity/node embedding matrix. For decoder, $e_s$ and $e_r$ are fed into Conv-TransE. The output embeddings are vectorized and projected, and matched with all candidate $e_o$ embeddings via inner products. A logistic sigmoid function is used to get the scores.

18 billion edges (Ying et al. 2018). This success paves the way for a new generation of web-scale recommender systems based on GCNs. Therefore we believe that our proposed model could take advantage of huge graph structures and high computational efficiency of Conv-TransE.

**Method**

In this section, we describe the proposed end-to-end SACN. The encoder WGCN is focused on representing entities by aggregating connected entities as specified by the relations in the KB. With node embeddings as the input, the decoder Conv-TransE network aims to represent the relations more accurately by recovering the original triplets in the KB. Both encoder and decoder are trained jointly by minimizing the discrepancy (cross-entropy) between the embeddings $e_s + e_r$ and $e_o$, to preserve the translational property $e_s + e_r \approx e_o$. We consider an undirected graph $G = (V, E)$ throughout this section, where $V$ is a set of nodes with $|V| = N$, and $E \subseteq V \times V$ is a set of edges with $|E| = M$.

**Weighted Graph Convolutional Layer**

The WGCN is an extension of classic GCN (Kipf and Welling 2016b) in the way that it weighs the different types of relations differently when aggregating and the weights are adaptively learned during the training of the network. By this adaptation, the WGCN can control the amount of information from neighboring nodes used in aggregation. Roughly speaking, the WGCN treats a multi-relational KB graph as multiple single-relational subgraphs where each subgraph entails a specific type of relations. The WGCN determines how much weights to give to each subgraph when combining the GCN embeddings for a node.

The $l$-th WGCN layer takes the output vector of length $F^l$ for each node from the previous layer as inputs and generates a new representation comprising $F^{l+1}$ elements. Let $h_i^l$ represent the input (row) vector of the node $v_i$ in the $l$-th layer, and thus $H^l \in \mathbb{R}^{N \times F^l}$ be the input matrix for this layer. The initial embedding $H^1$ is randomly drawn from Gaussian. If there are a total of $L$ layers in the WGCN, the output $H^{L+1}$ of the $L$-th layer is the final embedding. Let the total number of edge types be $T$ in a multi-relational KB graph with $E$ edges. The interaction strength between two adjacent nodes is determined by their relation type and this strength is specified by a parameter $\{\alpha_t, 1 \leq t \leq T\}$ for each edge type, which is automatically learned in the neural network.

Figure 1 illustrates the entire process of SACN. In this example, the WGCN layers of the network compute the embeddings for the red node in the middle graph. These layers aggregate the embeddings of neighboring entity nodes as specified in the KB relations. Three colors (blue, yellow and green) of the edges indicate three different relation types in the graph. The corresponding three entity nodes are summed up with different weights according to $\alpha_t$ in this layer to obtain the embedding of the red node. The edges with the same color (same relation type) use the same $\alpha_t$. Each layer has its own set of relation weights $\alpha_t$. Hence, the output of the $l$-th layer for the node $v_i$ can be written as follows:

$$h_i^{l+1} = \sigma \left( \sum_{j \in N_i} \alpha_t g(h_i^l, h_j^l) \right),$$

where $h_i^l \in \mathbb{R}^{F^l}$ is the input for node $v_i$, and $v_j$ is a node in the neighbor $N_i$ of node $v_i$. The $g$ function specifies how to incorporate neighboring information. Note that the activation function $\sigma$ here is applied to every component of its input vector. Although any function $g$ suitable for a KB embedding can be used in conjunction with the proposed frame-

Figure 1: An illustration of our end-to-end Structure-Aware Convolutional Networks model. For encoder, a stack of multiple WGCN layers builds an entity/node embedding matrix. For decoder, $e_s$ and $e_r$ are fed into Conv-TransE. The output embeddings are vectorized and projected, and matched with all candidate $e_o$ embeddings via inner products. A logistic sigmoid function is used to get the scores.
work, we implement the following function:

\[ g(h_i^l, h_j^l) = h_j^l W^l, \]  

where \( W^l \in \mathbb{R}^{f_l \times f_{l+1}} \) is the connection coefficient matrix and used to linearly transform \( h_i^l \) to \( h_j^l+1 \in \mathbb{R}^{f_{l+1}} \).

In Eq. (1), the input vectors of all neighboring nodes are summed up but not the node \( v_i \) itself, hence self-loops are enforced in the network. For node \( v_i \), the propagation process is defined as:

\[ h_i^{l+1} = \sigma(\sum_{j \in \mathcal{N}_i} \alpha_i^l h_j^l W^l + h_i^l W^l). \]  

The output of the layer \( l \) is a node feature matrix: \( H^{l+1} \in \mathbb{R}^{N \times f_{l+1}} \), and \( h_i^{l+1} \) is the \( i \)-th row of \( H^{l+1} \), which represents features of the node \( v_i \) in the \((l+1)\)-th layer.

The above process can be organized as a matrix multiplication as shown in Figure 2 to simultaneously compute embeddings for all nodes through an adjacency matrix. For each relation (edge) type, an adjacency matrix \( A_t \) is a binary matrix whose \( ij \)-th entry is 1 if an edge connecting \( v_i \) and \( v_j \) exists or 0 otherwise. The final adjacency matrix is written as follows:

\[ A^l = \sum_{t=1}^{T} (\alpha_t^l A_t) + I, \]  

where \( I \) is the identity matrix of size \( N \times N \). Basically, the \( A^l \) is the weighted sum of the adjacency matrices of subgraphs plus self-connections. In our implementation, we consider all first-order neighbors in the linear transformation for each layer as shown in Figure 2:

\[ H^{l+1} = \sigma(A^l H^l W^l). \]  

**Node Attributes.** In a KB graph, nodes are often associated with several attributes in the form of \( \text{(entity, relation, attribute)} \). For example, \((s = \text{Tom}, r = \text{people.person.gender}, a = \text{male})\) is an instance where gender is an attribute associated with a person. If a vector representation is used for node attributes, there would be two potential problems. First, the number of attributes for each node is usually small, and differs from one to another. Hence, the attribute vector would be very sparse. Second, the value of zero in the attribute vectors may have ambiguous meanings: the node does not have the specific attribute, or the node misses the value for this attribute. These zeros would affect the accuracy of the embedding.

In this work, the entity attributes in the knowledge graph are represented by another set of nodes in the network called attribute nodes. Attribute nodes act as the “bridges” to link the related entities. The entity embeddings can be transported over these “bridges” to incorporate the entity’s attribute into its embedding. Because these attributes exhibit in triplets, we represent the attributes similarly to the representation of the entity \( o \) in relation triplets. Note that each type of attribute corresponds to a node. For instance, in our example, gender is represented by a single node rather than two nodes for “male” and “female”. In this way, the WGCN not only utilizes the graph connectivity structure (relations and relation types), but also leverages the node attributes (a kind of graph structure) effectively. That is why we name our WGCN as a structure-aware convolution network.

**Conv-TransE**

We develop the Conv-TransE model as a decoder that is based on ConvE but with the translational property of TransE: \( e_s + e_r \approx e_o \). The key difference of our approach from ConvE is that there is no reshaping after stacking \( e_s \) and \( e_r \). Filters (or kernels) of size \( 2 \times k \), \( k \in \{1, 2, 3, \ldots\} \), are used in the convolution. The example in Figure 1 uses \( 2 \times 3 \) kernels to compute 2D convolutions. We experimented with several of such settings in our empirical study.

Note that in the encoder of SACN, the dimension of the relation embedding is commonly chosen to be the same as the dimension of the entity embedding, so in other words, is equal to \( F^L \). Hence, the two embeddings can be stacked. For the decoder, the inputs are two embedding matrices: one \( \mathbb{R}^{N \times F^L} \) from WGCN for all entity nodes, and the other \( \mathbb{R}^{M \times F^L} \) for relation embedding matrix which is trained as well. Because we use a mini-batch stochastic training algorithm, the first step of the decoder performs a look-up operation upon the embedding matrices to retrieve the input \( e_s \) and \( e_r \) for the triplets in the mini-batch.

More precisely, given \( C \) different kernels where the \( c \)-th kernel is parameterized by \( \omega_c \), the convolution in the decoder is computed as follows:

\[
m_c(e_s, e_r, n) = \sum_{\tau=0}^{K-1} \omega_c(\tau, 0) \hat{e}_s(n + \tau \mid K) + \omega_c(\tau, 1) \hat{e}_r(n + \tau \mid K),
\]  

where \( K \) is the kernel width, \( n \) indexes the entries in the output vector and \( n \in [0, F^L - 1] \), and the kernel parameters \( \omega_c \) is trainable. \( \hat{e}_s \) and \( \hat{e}_r \) are padding version of \( e_s \) and \( e_r \), respectively. If the dimension \( s \) of kernel is odd, the first \( \lfloor K/2 \rfloor \) and last \( \lfloor K/2 \rfloor \) components are filled with 0. Here \( \lfloor \text{value} \rfloor \) returns the floor of \( \text{value} \). Otherwise, the first \( \lfloor K/2 \rfloor - 1 \) and last \( \lfloor K/2 \rfloor \) components are filled with 0. Other components are copied from \( e_s \) and \( e_r \) directly. As shown in Eq. (6) the convolution operation amounts to a sum of \( e_s \) and \( e_r \) after the one-dimensional convolution. Hence, it preserves the translational property of the embeddings of \( e_s, e_r \). The output forms a vector \( M_c(e_s, e_r) = \ldots \).
Table 1: Scoring function $\psi(e_s, e_o)$. Here $e_s$ and $e_r$ denote a 2D reshaping of $e_s$ and $e_r$.

<table>
<thead>
<tr>
<th>Model</th>
<th>Scoring Function $\psi(e_s, e_o)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransE</td>
<td>$</td>
</tr>
<tr>
<td>DistMult</td>
<td>$\langle e_s, e_r, e_o \rangle$</td>
</tr>
<tr>
<td>ComplEx</td>
<td>$\langle e_s, e_r, e_o \rangle$</td>
</tr>
<tr>
<td>ConvE</td>
<td>$f(\text{vec}(\text{concat}(e_s, e_r)) \ast \omega) W e_o$</td>
</tr>
<tr>
<td>ComKB</td>
<td>$\text{concat}(g([e_s, e_r, e_o] \ast \omega)) \beta$</td>
</tr>
<tr>
<td>SACN</td>
<td>$f(\text{vec}(M(e_s, e_r)) W e_o)$</td>
</tr>
</tbody>
</table>

Finally, the scoring function for the Conv-TransE method after the nonlinear convolution is defined as below:

$$\psi(e_s, e_o) = f(\text{vec}(M(e_s, e_r)) W) e_o, \tag{7}$$

where $W \in \mathbb{R}^{C \times F^L \times F^L}$ is a matrix for the linear transformation, and $f$ denotes a non-linear function. The feature map matrix is reshaped into a vector $\text{vec}(M) \in \mathbb{R}^{C \times F^L}$ and projected into a $F^L$ dimensional space using $W$ for linear transformation. Then the calculated embedding is matched to $e_o$ by an appropriate distance metric. During the training in our experiments, we apply the logistic sigmoid function to the scoring:

$$p(e_s, e_r, e_o) = \sigma(\psi(e_s, e_o)). \tag{8}$$

In Table 1, we summarize the scoring functions used by several state of the art models. The vector $e_s$ and $e_o$ are the subject and object embedding respectively, $e_r$ is the relation embedding, “concat” means concatenates the inputs, and “$\ast$” denotes the convolution operator.

In summary, the proposed SACN model takes advantage of knowledge graph node connectivity, node attributes and relation types. The learnable weights in WGCN help to collect adaptive amount of information from neighboring graph nodes. The entity attributes are added as additional nodes in the network and are easily integrated into the WGCN. Conv-TransE keeps the translational property between entities and relations to learn node embeddings for the link prediction. We also emphasize that our SACN has significant improvements over ComE with or without the use of node attributes.

### Experiments

#### Benchmark Datasets

Three benchmark datasets (FB15k-237, WN18RR and FB15k-237-Attr) are utilized in this study to evaluate the performance of link prediction.

**FB15k-237.** The FB15k-237 (Toutanova and Chen 2015) dataset contains knowledge base relation triples and textual mentions of Freebase entity pairs, as used in the work published in (Toutanova and Chen 2015). The knowledge base triples are a subset of the FB15K (Bordes et al. 2013), originally derived from Freebase. The inverse relations are removed in FB15k-237.

**WN18RR.** WN18RR (Dettmers et al. 2017) is created from WN18 (Bordes et al. 2013), which is a subset of WordNet. WN18 consists of 18 relations and 40,943 entities. However, many text triples obtained by inverting triples from the training set. Thus WN18RR dataset (Dettmers et al. 2017) is created to ensure that the evaluation dataset does not have inverse relation test leakage. In summary, WN18RR dataset contains 93,003 triples with 40,943 entities and 11 relation types.

**Data Construction**

Most of the previous methods only model the entities and relations, and ignore the abundant entity attributes. Our method can easily model a large number of entity attribute triples. In order to prove the efficiency, we extract the attribute triples from the FB24k (Lin, Liu, and Sun 2016) dataset to build the evaluation dataset called FB15k-237-Attr.

**FB24k.** FB24k (Lin, Liu, and Sun 2016) is built based on Freebase dataset. FB24k only selects the entities and relations which constitute at least 30 triples. The number of entities is 23,634, and the number of relations is 673. In addition, the reversed relations are removed from the original dataset. In the FB24k datasets, the attribute triples are provided. FB24k contains 207,151 attribute triples and 314 attributes.

**FB15k-237-Attr.** We extract the attribute triples of entities in FB15k-237 from FB24k. During the mapping, there are 7,589 nodes from the original 14,541 entities which have the node attributes. Finally, we extract 78,334 attribute triples from FB24k. These triples include 203 attributes and 247 relations. Based on these triples, we create the “FB15k-237-Attr” dataset, which includes 14,541 entity nodes, 203 attribute nodes, 484 relation types. All the 78,334 attribute triples are combined with the training set of FB15k-237.

#### Experimental Setup

The hyperparameters in our Conv-TransE and SACN models are determined by a grid search during the training. We manually specify the hyperparameters ranges: learning rate $\{0.01, 0.005, 0.003, 0.001\}$, dropout rate $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$, embedding size...
Table 3: Link prediction for FB15k-237, WN18RR and FB15k-237-Attr datasets.

<table>
<thead>
<tr>
<th>Model</th>
<th>FB15k-237 @10</th>
<th>FB15k-237 @3</th>
<th>FB15k-237 @1</th>
<th>WN18RR @10</th>
<th>WN18RR @3</th>
<th>WN18RR @1</th>
<th>MRR</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistMult (Yang et al. 2014)</td>
<td>0.42</td>
<td>0.26</td>
<td>0.16</td>
<td>0.24</td>
<td>0.49</td>
<td>0.44</td>
<td>0.39</td>
</tr>
<tr>
<td>ComplEx (Trouillon et al. 2016)</td>
<td>0.43</td>
<td>0.28</td>
<td>0.16</td>
<td>0.25</td>
<td>0.51</td>
<td>0.46</td>
<td>0.41</td>
</tr>
<tr>
<td>R-GCN (Schlichtkrull et al. 2018)</td>
<td>0.42</td>
<td>0.26</td>
<td>0.15</td>
<td>0.25</td>
<td>0.54</td>
<td>0.43</td>
<td>0.39</td>
</tr>
<tr>
<td>ConvE (Dettmers et al. 2017)</td>
<td>0.49</td>
<td>0.35</td>
<td>0.24</td>
<td>0.32</td>
<td>0.54</td>
<td>0.48</td>
<td>0.43</td>
</tr>
<tr>
<td>SACN using FB15k-237-Attr</td>
<td>0.55</td>
<td>0.40</td>
<td>0.27</td>
<td>0.36</td>
<td>0.54</td>
<td>0.48</td>
<td>0.43</td>
</tr>
</tbody>
</table>

Performance Improvement: 12.2% | 14.3% | 12.5% | 12.5% | 11.6% | 10.3% | 2.2%

\{100, 200, 300\}, number of kernels \{50, 100, 200, 300\}, and kernel size \{2 \times 1, 2 \times 3, 2 \times 5\}.

Here all the models use the WGCN with two layers. For different datasets, we have found that the following settings work well: for FB15k-237, set the dropout to 0.2, number of kernels to 100, learning rate to 0.003 and embedding size to 200 for SACN; for WN18RR dataset, set dropout to 0.2, number of kernels to 300, learning rate to 0.003, and embedding size to 200 for SACN. When using the Conv-TransE-alone model, these settings still work well.

Each dataset is split into three sets for: training, validation and testing, which is same with the setting of the original ConvE. We use the adaptive moment (Adam) algorithm (Kingma and Ba 2014) for training the model. Our models are implemented by PyTorch and run on NVIDIA Tesla P40 Graphics Processing Units. For the FB15k-237 dataset, the computation time of SACN for each epoch is about 1 minute. For the WN18RR, the computation time of SACN for one epoch is about 1.5 minutes.

Results

Evaluation Protocol Our experiments use the the proportion of correct entities ranked in top 1, 3 and 10 (Hits@1, Hits@3, Hits@10) and the mean reciprocal rank (MRR) as the metrics. In addition, since some corrupted triplets exist in the knowledge graphs, we use the filtered setting (Bordes et al. 2013), i.e. we filter out all valid triplets before ranking.

Link Prediction Our results on the standard FB15k-237, WN18RR and FB15k-237-Attr are shown in Table 3. Table 3 reports Hits@10, Hits@3, Hits@1 and MRR results of four different baseline models and two our models on three knowledge graphs datasets. The FB15k-237-Attr dataset is used to prove the efficiency of node attributes. So we run our SACN in FB15k-237-Attr to do the comparison with SACN using FB15k-237.

We first compare our Conv-TransE model with the four baseline models. ConvE has the best performance comparing all baselines. In FB15k-237 dataset, our Conv-TransE model improves upon ConvE’s Hits@10 by a margin of 4.1% , and upon ConvE’s Hits@3 by a margin of 5.7% for the test. In WN18RR dataset, Conv-TransE improves upon ConvE’s Hits@10 by a margin of 8.3% , and upon ConvE’s Hits@3 by a margin of 9.3% for the test. For these results, we conclude that Conv-TransE using neural network keeps the translational characteristic between entities and relations and achieve better performance.

Second, the structure information is added into our SACN model. In Table 3, SACN also get the best performances in the test dataset comparing all baseline methods. In FB15k-237, comparing ConvE, our SACN model improves Hits@10 value by a margin of 10.2%, Hits@3 value by a margin of 11.4%, Hits@1 value by a margin of 8.3% and MRR value by a margin of 9.4% for the test. In WN18RR dataset, comparing ConvE, our SACN model improves Hits@10 value by a margin of 12.5%, Hits@3 value by a margin of 11.6%, Hits@1 value by a margin of 10.3% and MRR value by a margin of 2.2% for the test. So our method has significant improvements over ConvE without attributes.

Third, we add node attributes into our SACN model, i.e. we use the FB15k-237-Attr to train SACN. Note that SACN has significant improvements over ConvE without attributes. Adding attributes improves performance again. Our model using attributes improves upon ConvE’s Hits@10 by a margin of 12.2% , Hits@3 by a margin of 14.3%, Hits@1 by a margin of 12.5% and MRR by a margin of 12.5%. In addition, our SACN using attributes improved Hits@10 by a margin of 19.9% , Hits@3 by a margin of 2.6%, Hits@1 by a margin of 3.8% and MRR by a margin of 2.9% comparing with SACN without attributes.

In order to better compare with ConvE, we also use the attributes into ConvE. Here the attributes will be treated as the entity triplets. Following the official ConvE code with default setting, the test result in FB15k-237-Attr was: 0.46 (Hits@10), 0.33 (Hits@3), 0.22 (Hits@1) and 0.30 (MRR). Comparing to the performance without the attributes, adding the attributes into the ConvE didn’t improve performance.

Convergence Analysis Figure 3 shows the convergence of the three models. We can see that the SACN (the red line) is always better than Conv-TransE (the yellow line) after several epochs. And the performance of SACN keeps increasing after around 120 epochs. However, the Conv-TransE has achieved the best performance after around 120 epochs. The gap between these two models proves the usefulness of structural information. When using the FB15k-237-Attr
Table 4: Kernel size analysis for FB15k-237 and FB15k-237-Attr datasets. "SACN+Attr" means the SACN using FB15k-237-Attr dataset.

<table>
<thead>
<tr>
<th>Model</th>
<th>Kernel Size</th>
<th>FB15k-237</th>
<th>Hits</th>
<th>MRR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>[0,100]</td>
<td>@10</td>
<td>@3</td>
</tr>
<tr>
<td>Con-TransE</td>
<td>2 x 1</td>
<td>0.504</td>
<td>0.357</td>
<td>0.234</td>
</tr>
<tr>
<td>Con-TransE</td>
<td>2 x 3</td>
<td>0.513</td>
<td>0.365</td>
<td>0.240</td>
</tr>
<tr>
<td>Con-TransE</td>
<td>2 x 5</td>
<td>0.512</td>
<td>0.361</td>
<td>0.239</td>
</tr>
<tr>
<td>SACN</td>
<td>2 x 1</td>
<td>0.527</td>
<td>0.379</td>
<td>0.255</td>
</tr>
<tr>
<td>SACN</td>
<td>2 x 3</td>
<td>0.536</td>
<td>0.384</td>
<td>0.260</td>
</tr>
<tr>
<td>SACN</td>
<td>2 x 5</td>
<td>0.536</td>
<td>0.385</td>
<td>0.261</td>
</tr>
<tr>
<td>SACN+Attr</td>
<td>2 x 1</td>
<td>0.535</td>
<td>0.384</td>
<td>0.260</td>
</tr>
<tr>
<td>SACN+Attr</td>
<td>2 x 3</td>
<td>0.543</td>
<td>0.394</td>
<td>0.268</td>
</tr>
<tr>
<td>SACN+Attr</td>
<td>2 x 5</td>
<td>0.547</td>
<td>0.396</td>
<td>0.268</td>
</tr>
</tbody>
</table>

Along the increasing of indegree scope, the average value of Hits@10 and Hits@3 will be increased. First for a node with small indegree, it benefits from aggregation of neighbor information from the WGCN layers of SACN. Its embedding can be estimated robustly. Second for a node with high indegree, it means that a lot more information is aggregated through GCN, and the estimation of its embedding is substantially smoothed among neighbors. Thus the embedding learned from SACN is worse than that from Conv-TransE. One solution to this problem would be neighbor selection as in (Ying et al. 2018).

Conclusion and Future Work

We have introduced an end-to-end structure-aware convolutional network (SACN). The encoding network is a weighted graph convolutional network, utilizing knowledge graph connectivity structure, node attributes and relation types. WGCN with learnable weights has the benefit of collecting adaptive amount of information from neighboring graph nodes. In addition, the entity attributes are added as the nodes in the network so that attributes are transformed into knowledge structure information, which is easily integrated into the node embedding. The scoring network of SACN is a convolutional neural model, called Con-TransE. It uses a convolutional network to model the relationship as the translation operation and capture the translational characteristic between entities and relations. We also prove that Con-TransE alone has already achieved the state of the art performance. The performance of SACN achieves overall about 10% improvement than the state of the art such as ConvE.

In the future, we would like to incorporate the neighbor selection idea into our training framework, such as, importance pooling in (Ying et al. 2018) which takes into account the importance of neighbors when aggregating the vector representations of neighbors. We would also like to extend our model to be scalable with larger knowledge graphs encouraged by the results in (Ying et al. 2018).

Acknowledgements

This work was partially supported by NSF grants CCF-1514357 and IIS-1718738, as well as NIH grants R01DA037349 and K02DA043063 to Jinbo Bi.
References


Lin, Y.; Liu, Z.; and Sun, M. 2016. Knowledge representa-
tion learning with entities, attributes and relations. ethnicity 1:41–52.

Mahdisoltani, F.; Biega, J.; and Suchanek, F. M. 2013. Yago3: A knowledge base from multilingual wikipedias. In CIDR.


