Chapter 3

Multi-relational data and knowledge graphs

In Chapter 2 we discussed several approaches for learning low dimensional embeddings of nodes. The majority of these approaches were so-called *shallow embedding* approaches, where we learn a unique embedding for each node. In this chapter we will continue our focus on shallow embedding methods, and we will introduce techniques to deal with multi-relational graphs.

Knowledge graph completion Most of the methods we review in this chapter were originally designed for the task of knowledge graph completion. In knowledge graph completion, we are given a multi-relational graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the edges are defined as tuples $e = (u, \tau, v)$ indicating the presence of a particular relation $\tau \in \mathcal{T}$ holding between two nodes. Such multi-relational graphs are often referred to as *knowledge graphs*, since we can interpret the tuple (u, τ, v) as specifying that a particular "fact" holds between the two nodes u and v. For example, in a biomedical knowledge graph we might have an edge type $\tau = \texttt{TREATS}$ and the edge $(u, \texttt{TREATS}, v)$ could indicate that the drug associated with node u treats the disease associated with node v. Generally the goal in knowledge graph completion is to predict missing relations in the graph, i.e., relation prediction, but there are also examples of node classification tasks using multi-relational graphs [Schlichtkrull et al., 2017].

In this chapter we will provide a brief overview of embedding methods for multi-relational graphs, but it is important to note that a full treatment of knowledge graph completion is beyond the scope of this chapter. Not all knowledge graph completion methods rely on embeddings, and we will not cover every variation of embedding methods here. We refer interested readers to Nickel et al. [2016] for a complimentary review of the area.

3.1 Reconstructing multi-relational data

As with the simple graphs discussed in Chapter 2, we can view embedding multi-relational graphs as a reconstruction task. Given the embeddings $\mathbf{z}_u$ and $\mathbf{z}_v$ of two nodes, our goal is to reconstruct the relationship between these nodes. The complication compared to the embedding methods of the previous chapter is that we now have to deal with the presence of multiple different types of edges.

To address this complication, we augment our decoder to make it multi-relational. Instead of only taking a pair of node embeddings as input, we now define the decoder as accepting a pair of node embeddings *as well as a relation type*, i.e., $\text{DEC} : \mathbb{R}^d \times \mathcal{R} \times \mathbb{R}^d \rightarrow \mathbb{R}^+$. We can interpret the output of this decoder, i.e., $\text{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v)$, as the likelihood that the relation (u, τ, v) exists in the graph.

To give a concrete example, one of the simplest and earliest approaches to learning multi-relational embeddings—often termed RESCAL—defined the decoder as [Nickel et al., 2011]:

$$\text{DEC}(u, \tau, v) = \mathbf{z}_u^\top \mathbf{R}_\tau \mathbf{z}_v, \tag{3.1}$$

where $\mathbf{R}_\tau \in \mathbb{R}^{d \times d}$ is a learnable matrix specific to relation $\tau \in \mathcal{R}$. Keeping things simple with this decoder, we could train our embedding matrix $\mathbf{Z}$ and our relation matrices $\mathbf{R}_\tau, \forall \tau \in \mathcal{R}$ using a basic reconstruction loss:

$$\mathcal{L} = \sum_{u \in \mathcal{V}} \sum_{v \in \mathcal{V}} \sum_{\tau \in \mathcal{R}} \|\text{DEC}(u, \tau, v) - \mathcal{A}[u, \tau, v]\|^2 \tag{3.2}$$

$$= \sum_{u \in \mathcal{V}} \sum_{v \in \mathcal{V}} \sum_{\tau \in \mathcal{R}} \|\mathbf{z}_u^\top \mathbf{R}_\tau \mathbf{z}_v - \mathcal{A}[u, \tau, v]\|^2, \tag{3.3}$$

where $\mathcal{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{R}| \times |\mathcal{V}|}$ is the adjacency tensor for the multi-relational graph. If we were to optimize Equation 3.2, we would in fact be performing a kind of *tensor factorization*. This idea of factorizing a tensor thus generalizes the matrix factorization ideas discussed in Chapter 2.

Loss functions, decoders, and similarity functions In Chapter 2 we discussed how the diversity of methods for node embeddings largely stem from the use of different decoders (DEC), similarity measures ($\mathbf{S}[u, v]$), and loss functions ($\mathcal{L}$). The decoder gives a score between a pair of node embeddings; the similarity function defines what kind of node-node similarity we are trying to decode; and the loss function tells us how to measure the discrepancy between the output of the decoder and the ground truth similarity measure.

In the multi-relational setting we will also see a diversity of decoders and loss functions. However, nearly all multi-relational embedding methods simply define the similarity measure directly based on the adjacency tensor. In other words, all the methods in this chapter assume that we are trying to reconstruct immediate (multi-relational) neighbors from the low-dimensional embeddings. This is due to the difficulty of defining higher-order neighborhood relationships in multi-relational graphs, as well as the fact that most multi-relational embedding methods were specifically designed for relation prediction.

3.2 Loss functions

As discussed above, the two key ingredients for a multi-relational node embedding method are the decoder and the loss function. We begin with a brief discussion of the standard loss functions used for this task, before turning our attention to the multitude of decoders that have been proposed in the literature.

As a motivation for the loss functions we consider, it is worth considering the drawbacks of the reconstruction loss defined in Equation 3.2. There are two major problems with this loss. The first issue is that it is extremely expensive to compute. The nested sums require $O(|\mathcal{V}|^2|\mathcal{R}|)$ operations, and this computation time will be prohibitive in many large graphs. Moreover, since many multi-relational graphs are *sparse*—i.e., $|\mathcal{E}| << |\mathcal{V}|^2|\mathcal{R}|$—we would ideally want a loss function that is $O(|\mathcal{E}|)$. The second problem is more subtle. Our goal is to decode the adjacency tensor from the low-dimensional node embeddings. We know that (in most cases) this tensor will contain only binary values, but the mean-squared error in Equation 3.2 is not well suited to such a binary comparison. In fact the mean-squared error is a natural loss for *regression* whereas our target is something closer to *binary classification on edges*.

Cross-entropy with negative sampling

One popular loss function that is both efficient and suited to our task is the *cross-entropy loss with negative sampling*. We define this loss as:

$$\mathcal{L} = \sum_{(u,\tau,v)\in\mathcal{E}} -\log(\sigma(\text{DEC}(\mathbf{z}_u,\tau,\mathbf{z}_v))) - \gamma\mathbb{E}_{v_n\sim P_n(\mathcal{V})}\left[\log\left(\sigma\left(-\text{DEC}(\mathbf{z}_u,\tau,\mathbf{z}_{v_n})\right)\right)\right] \tag{3.4}$$

where σ denotes the logistic function, $P_{n,u}(\mathcal{V})$ denotes a "negative sampling" distribution over the set of nodes $\mathcal{V}$ (which might depend on u) and $\gamma > 0$ is a hyperparameter. This is essentially the same loss as we saw for node2vec (Equation 2.12), but here we are considering general multi-relational decoders.

We call this a *cross-entropy* loss because it is derived from the standard binary cross-entropy loss. Since we are feeding the output of the decoder to a logistic function, we obtain normalized scores in $[0, 1]$ that can be interpreted as probabilities. The term

$$\log(\sigma(\text{DEC}(\mathbf{z}_u,\tau,\mathbf{z}_v))) \tag{3.5}$$

then equals the log-likelihood that we predict "true" for an edge that does actually exist in the graph. On the other hand the term

$$\mathbb{E}_{v_n\sim P_n(\mathcal{V})}\left[\log\left(\sigma\left(-\text{DEC}(\mathbf{z}_u,\tau,\mathbf{z}_{v_n})\right)\right)\right] \tag{3.6}$$

then equals the expected log-likelihood that the correctly predict "false" for an edge that does not exist in the graph. In practice the expectation is evaluated

using a Monte-Carlo approximation and the most popular form of this loss is

$$\mathcal{L} = \sum_{(u,\tau,v)\in\mathcal{E}} -\log(\sigma(\text{DEC}(\mathbf{z}_u,\tau,\mathbf{z}_v))) - \sum_{v_n\in\mathcal{P}_n(u)} \left[\log\left(\sigma\left(-\text{DEC}(\mathbf{z}_u,\tau,\mathbf{z}_{v_n})\right)\right)\right], \tag{3.7}$$

where $\mathcal{P}_{n,u}$ is a (usually small) set of nodes sampled from $P_{n,u}$.

A note on negative sampling The way in which negative samples are generated can have a large impact on the quality of the learned embeddings. The most common approach to define the distribution $P_{n,u}$ is to simply use a uniform distribution over all nodes in the graph. This is a simple strategy, but it also means that we will get "false negatives" in the cross-entropy calculation. In other words, it is possible that we accidentally sample a "negative" tuple $(u,\tau,v_n) \in \mathcal{E}$ that actually exists in the graph. Some works address this by removing such false negatives.

Other variations of negative sampling attempt to produce more "difficult" negative samples. For example, some relations can only exist between certain types of nodes. (A node representing a person in a knowledge graph would be unlikely to be involved in an edge involving the `MANUFACTURED-BY` relation). Thus, one strategy is to only sample negative examples that satisfy such type constraints. Sun et al. [2019] even propose an approach to select challenging negative samples using an adversarial network.

Note also that without loss of generality, we have assumed that the negative sampling occurs over the second node in the edge tuple. That is, we assume that we draw a negative sample by replacing the *tail* node v in the tuple (u,τ,v) with a negative sample v_n. Always sampling the tail node simplifies notation but can lead to biases in multi-relational graphs where edge direction is important. In practice it can be better to draw negative samples for both the *head* node (i.e., u) and the tail node (i.e., v) of the relation.

Max-margin loss

The other popular loss function used for multi-relational node embedding is the margin loss:

$$\mathcal{L} = \sum_{(u,\tau,v)\in\mathcal{E}} \sum_{v_n\in\mathcal{P}_{n,u}} \max(0, -\text{DEC}(\mathbf{z}_u,\tau,\mathbf{z}_v) + \text{DEC}(\mathbf{z}_u,\tau,\mathbf{z}_{v_n}) + \Delta). \tag{3.8}$$

In this loss we are again comparing the decoded score for a true pair compared to a negative sample—a strategy often termed *contrastive estimation.* However, rather than treating this as a binary classification task, in Equation 3.8 we are simply comparing the direct output of the decoders. If the score for the "true" pair is bigger than the "negative" pair then we have a small loss. The Δ term is called the margin, and the loss will equal 0 if the difference in scores is at least that large for all examples. This loss is also known as the *hinge loss*.

Name	Decoder	Relation Parameters
RESCAL	$\mathbf{z}_u^\top \mathbf{R}_\tau \mathbf{z}_v$	$\mathbf{R}_\tau \in \mathbb{R}^{d \times d}$
TransE	$-\|\mathbf{z}_u + \mathbf{r}_\tau - \mathbf{z}_v\|$	$\mathbf{r}_\tau \in \mathbb{R}^d$
TransX	$-\|g_{1,\tau}(\mathbf{z}_u) + \mathbf{r}_\tau - g_{2,\tau}(\mathbf{z}_v)\|$	$\mathbf{r}_\tau \in \mathbb{R}^d, g_{1,\tau}, g_{2,\tau} \in \mathbb{R}^d \to \mathbb{R}^d$
DistMult	$< \mathbf{z}_u, \mathbf{r}_\tau, \mathbf{z}_v >$	$\mathbf{r}_\tau \in \mathbb{R}^d$
ComplEx	$\text{Re}(< \mathbf{z}_u, \mathbf{r}_\tau, \mathbf{z}_v >)$	$\mathbf{r}_\tau \in \mathbb{C}^d$
RotatE	$-\|\mathbf{z}_u \circ \mathbf{r}_\tau - \mathbf{z}_v\|$	$\mathbf{r}_\tau \in \mathbb{C}^d$

Table 3.1: Summary of some popular decoders used for multi-relational data.

3.3 Multi-relational decoders

So far we have only discussed one possible multi-relational decoder, the so-called RESCAL decoder:

$$\text{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v) = \mathbf{z}_u^\top \mathbf{R}_\tau \mathbf{z}_v. \tag{3.9}$$

In this approach, we associate a trainable matrix $\mathbf{R}_\tau \in \mathbb{R}^{d \times d}$ with each relation. However, one limitation of this approach—and a reason why it is not often used in practice—is its high computational and statistical cost for representing relations. There are $O(d^2)$ parameters for each relation in RESCAL, which means that relations require an order of magnitude more parameters to represent, compared to entities.

More popular modern decoders aim to use only $O(d)$ parameters to represent each relation. We will discuss several popular variations of multi-relational decoders here, though our survey is far from exhaustive. The decoders surveyed in this chapter are summarized in Table 3.1.

Translational decoders

One popular class of decoders represents relations as *translations* in the embedding space. This approach was initiated by Bordes et al. [2013]'s *TransE* model, which defined the decoder as

$$\text{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v) = -\|\mathbf{z}_u + \mathbf{r}_\tau - \mathbf{z}_v\|. \tag{3.10}$$

In these approaches, we represent each relation using a d-dimensional embedding. The likelihood of an edge is proportional to the distance between the embedding of the head node and the tail node, after *translating* the head node according to the relation embedding. TransE is one of the earliest multi-relational decoders proposed and continues to be a strong baseline in many applications.

One limitation of TransE is its simplicity, however, and many works have also proposed extensions of this translation idea. We collectively refer to these models as *TransX* models and they have the form:

$$\text{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v) = -\|g_{1,\tau}(\mathbf{z}_u) + \mathbf{r}_\tau - g_{2,\tau}(\mathbf{z}_v)\|, \tag{3.11}$$

where $g_{i,\tau}$ are trainable transformations that depend on the relation τ. For example, Wang et al. [2014]'s *TransH* model defines the decoder as

$$\text{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v) = -\|(\mathbf{z}_u - \mathbf{w}_r^\top \mathbf{z}_u \mathbf{w}_r) + \mathbf{r}_\tau - (\mathbf{z}_u - \mathbf{w}_r^\top \mathbf{z}_v \mathbf{w}_r)\|. \tag{3.12}$$

The TransH approach projects the entity embeddings onto a learnable relation-specific hyperplane—defined by the normal vector $\mathbf{w}_r$—before performing translation. Additional variations of the TransE model are proposed in Nguyen et al. [2016] and Ji et al. [2015].

Multi-linear dot products

Rather than defining a decoder based upon translating embeddings, a second popular line of work develops a multi-relational decoder by generalizing the dot product decoder from simple graphs. In this approach—often termed DistMult and first proposed by Yang et al. [2014]—we define the decoder as

$$\begin{aligned} \text{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v) &=< \mathbf{z}_u, \mathbf{r}_\tau, \mathbf{z}_v > & (3.13) \\ &= \sum_{i=1}^{d} \mathbf{z}_u[i] \times \mathbf{r}_\tau[i] \times \mathbf{z}_v[j]. & (3.14) \end{aligned}$$

Thus, this approach takes a straightforward generalization of the dot product to be defined over three vectors.

Complex decoders

One limitation of the DistMult decoder in Equation (3.13) is that it can only encode *symmetric* relations. In other words, for the multi-linear dot product decoder defined in Equation (3.13), we have that

$$\begin{aligned} \text{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v) &=< \mathbf{z}_u, \mathbf{r}_\tau, \mathbf{z}_v > \\ &= \sum_{i=1}^{d} \mathbf{z}_u[i] \times \mathbf{r}_\tau[i] \times \mathbf{z}_v[j] \\ &=< \mathbf{z}_v, \mathbf{r}_\tau, \mathbf{z}_u > \\ &= \text{DEC}(\mathbf{z}_v, \tau, \mathbf{z}_u). \end{aligned}$$

This is a serious limitation as many relations are directed and asymmetric.To address this issue, Trouillon et al. [2016] proposed augmenting the DistMult encoder by employing complex-valued embeddings. They define the *ComplEx* as

$$\begin{aligned} \text{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v) &= \text{Re}(< \mathbf{z}_u, \mathbf{r}_\tau, \bar{\mathbf{z}}_v) & (3.15) \\ &= \text{Re}(\sum_{i=1}^{d} \mathbf{z}_u[i] \times \mathbf{r}_\tau[i] \times \bar{\mathbf{z}}_v[j]), & (3.16) \end{aligned}$$

where now $\mathbf{z}_u, \mathbf{z}_v, \mathbf{r}_\tau \in \mathbb{C}^d$ are complex-valued embeddings and Re denotes the real component of a complex vector. Since we take the complex conjugate $\bar{\mathbf{z}}_v$ of the tail embedding, this generalization of the dot product can accommodate asymmetric relations.

A related approach, termed *RotatE*, defines the decoder as rotations in the complex plane as [Sun et al., 2019]:

$$\text{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v) = -\|\mathbf{z}_u \circ \mathbf{r}_\tau - \mathbf{z}_v\|, \tag{3.17}$$

where $\circ$ denotes the Hadamard product. In Equation 3.17 we again assume that all embeddings are complex valued, and we additionally constrain the entries of $\mathbf{r}_\tau$ so that $|\mathbf{r}_\tau[i]| = 1, \forall i \in \{1, ..., d\}$. This restriction implies that each dimension of the relation embedding can be represented as $\mathbf{r}_\tau[i] = e^{i\theta_{r,i}}$ and thus corresponds to a rotation in the complex plane.

3.3.1 Representational abilities

One way to characterize the various multi-relational decoders is in terms of their ability to represent different logical patterns on relations.

Symmetry and anti-symmetry For example, many relations are *symmetric*, meaning that

$$(u, \tau, v) \in \mathcal{E} \rightarrow (v, \tau, u) \in \mathcal{E}. \tag{3.18}$$

In other cases, we have explicitly *anti-symmetric* relations that satisfy:

$$(u, \tau, v) \in \mathcal{E} \rightarrow (v, \tau, u) \notin \mathcal{E}. \tag{3.19}$$

One important question is whether or not different decoders are capable modeling both symmetric and anti-symmetric relations. DistMult, for example, can only represent symmetric relations, since

$$\begin{aligned} \text{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v) &=< \mathbf{z}_u, \mathbf{r}_\tau, \mathbf{z}_v > \\ &=< \mathbf{z}_v, \mathbf{r}_\tau, \mathbf{z}_u > \\ &= \text{DEC}(\mathbf{z}_v, \tau, \mathbf{z}_u) \end{aligned}$$

by definition for that approach. The TransE model on the other hand can only represent anti-symmetric relations, since

$$\begin{aligned} \text{DEC}(\mathbf{z}_u, \tau, \mathbf{z}_v) &= \text{DEC}(\mathbf{z}_v, \tau, \mathbf{z}_u) \\ -\|\mathbf{z}_u + \mathbf{r}_\tau - \mathbf{z}_v\| &= -\|\mathbf{z}_v + \mathbf{r}_\tau - \mathbf{z}_r\| \\ &\Rightarrow \\ -\mathbf{r}_\tau &= \mathbf{r}_\tau \\ &\Rightarrow \\ \mathbf{r}_\tau &= 0. \end{aligned}$$

Name	Symmetry	Anti-Symmetry	Inversion	Compositionality
RESCAL	✓	✓	✓	✓
TransE	✗	✓	✓	✓
TransX	✓	✓	✗	✗
DistMult	✓	✗	✗	✗
ComplEx	✓	✓	✓	✗
RotatE	✓	✓	✓	✓

Table 3.2: Summary of the ability of some popular multi-relational decoders to encode relational patterns. Adapted from Sun et al. [2019].

Inversion Related to symmetry is the notion of inversion, where one relation implies the existence of another, with opposite directionality:

$$(u, \tau_1, v) \in \mathcal{E} \rightarrow (v, \tau_2, u) \in \mathcal{E} \tag{3.20}$$

Most decoders are able to represent inverse relations, though again DistMult is unable to model such a pattern.

Compositonality Lastly, we can consider whether or not the decoders can encode compositionality between relation representations of the form:

$$(u, \tau_1, y) \in \mathcal{E} \wedge (y, \tau_2, v) \in \mathcal{E} \rightarrow (u, \tau_3, v) \in \mathcal{E}. \tag{3.21}$$

For example, in TransE we can accommodate this by defining $\mathbf{r}_{\tau_3} = \mathbf{r}_{\tau_1} + \mathbf{r}_{\tau_2}$. We can similarly model compositionality in RESCAL by defining $\mathbf{R}_{\tau_3} = \mathbf{R}_{\tau_2}\mathbf{R}_{\tau_1}$.

In general considering these kinds of relational patterns is useful for comparing the representational capacities of different multi-relational decoders. In practice, we may not expect these patterns to hold exactly, but there may be many relations that exhibit these patterns to some degree (e.g., relations that are symmetric $> 90\%$ of the time). Table 3.2 summarizes the ability of the various decoders we discussed to encode these relational patterns.