

In addition to training data, we also need to define a model family to train. A common approach is to extend neural language models to model entities and relations. Neural language models learn a vector that provides a distributed representation of each word. They also learn about interactions between words, such as which word is likely to come after a sequence of words, by learning functions of these vectors. We can extend this approach to entities and relations by learning an embedding vector for each relation. In fact, the parallel between modeling language and modeling knowledge encoded as relations is so close that researchers have trained representations of such entities by using *both* knowledge bases *and* natural language sentences (Bordes *et al.*, 2011, 2012; Wang *et al.*, 2014a) or combining data from multiple relational databases (Bordes *et al.*, 2013b). Many possibilities exist for the particular parametrization associated with such a model. Early work on learning about relations between entities (Paccanaro and Hinton, 2000) posited highly constrained parametric forms (“linear relational embeddings”), often using a different form of representation for the relation than for the entities. For example, Paccanaro and Hinton (2000) and Bordes *et al.* (2011) used vectors for entities and matrices for relations, with the idea that a relation acts like an operator on entities. Alternatively, relations can be considered as any other entity (Bordes *et al.*, 2012), allowing us to make statements about relations, but more flexibility is put in the machinery that combines them in order to model their joint distribution.

A practical short-term application of such models is **link prediction**: predicting missing arcs in the knowledge graph. This is a form of generalization to new facts, based on old facts. Most of the knowledge bases that currently exist have been constructed through manual labor, which tends to leave many and probably the majority of true relations absent from the knowledge base. See Wang *et al.* (2014b), Lin *et al.* (2015) and Garcia-Duran *et al.* (2015) for examples of such an application.

Evaluating the performance of a model on a link prediction task is difficult because we have only a dataset of positive examples (facts that are known to be true). If the model proposes a fact that is not in the dataset, we are unsure whether the model has made a mistake or discovered a new, previously unknown fact. The metrics are thus somewhat imprecise and are based on testing how the model ranks a held-out set of known true positive facts compared to other facts that are less likely to be true. A common way to construct interesting examples that are probably negative (facts that are probably false) is to begin with a true fact and create corrupted versions of that fact, for example by replacing one entity in the relation with a different entity selected at random. The popular precision at 10% metric counts how many times the model ranks a “correct” fact among the top 10% of all corrupted versions of that fact.