LinkNBed: Multi-Graph Representation Learning with Entity Linkage (Appendix)

A Discussion and Insights on Entity Linkage Task

Entity linkage task is novel in the space of multi-graph learning and yet has not been tackled by any existing relational learning approaches. Hence we analyze our performance on the task in more detail here. We acknowledge that baseline methods are not tailored to the task of entity linkage and hence their low performance is natural. But we observe that our model performs well even in the unsupervised scenario where essentially the linkage loss function is switched off and our model becomes a relational learning baseline. We believe that the inductive ability of our model and shared parameterization helps to capture knowledge across graphs and allows for better linkage performance. This outcome demonstrates the merit in multi-graph learning for different inference tasks. Having said that, we admit that our results are far from comparable to state-of-the-art linkage results (Das et al., 2017) and much work needs to be done to advance representation and relational learning methods to support effective entity linkage. But we note that our model works for multiple types of entities in a very heterogeneous environment with some promising results which serves as an evidence to pursue this direction for entity linkage task.

We now discuss several use-case scenarios where our model did not perform well to gain insights on what further steps can be pursued to improve over this initial model:

Han Solo with many attributes (False-negative example). Han Solo is a fictional character in Star Wars and appears in both D-IMDB and D-FB records. We have a positive label for this sample but we do not predict it correctly. Our model combines multiple components to effectively learn across graphs. Hence we investigated all the components to check for the failures. One observation we have is the mismatch in the amount of attributes across the two datasets. Further, this is compounded by multi-value attributes. As described, we use paragraph2vec like model to learn attribute embeddings where for each attribute, we aggregate over all its values. This seems to be computing embeddings that are very noisy. As we have seen attributes are affecting the final result with high impact and hence learning very noisy attributes is not helping. Further, the mismatch in number of types is also an issue. Even after filtering the types, the difference is pretty large. Types are also included as attributes and they contribute context to relation embeddings. We believe that the skew in type difference is making the model learn bad embeddings. Specifically this happens in cases where lot of information is available like Han Solo as it lead to the scenario of abundant noisy data. With our investigation, we believe that contextual embeddings need further sophistication to handle such scenarios. Further, as we already learn relation, type and attribute embeddings in addition to entity embeddings, aligning relations, types and attributes as integral task could also be an important future direction.

Alfred Pennyworth is never the subject of matter (False-negative example). In this case, we observe a new pattern which was found in many other examples. While there are many triples available for this character in D-IMDB, very few triplets are available in D-FB. This skew in availability of data hampers the learning of deep network which ends up learning very different embeddings for two realizations. Further, we observe another patter where Alfred Pennyworth appears only as an object in all those few triplets of D-FB while it appears as both subject and object in D-IMDB. Accounting for asymmetric relationships in an explicit manner may become helpful for this scenario.

Thomas Wayne is Martha Wayne! (False-positive example). This is the case of abundance of similar contextual information as our model predicts Thomas Wayne and Martha Wayne to be same entity. Both the characters share a lot of context and hence many triples and attributes, neighborhood etc. are similar for of them eventually learning very similar embeddings. Further as we have seen before, neighborhood has shown to be a weak context which seems to hamper the learning in this case. Finally, the key insight here is to be able to attend to the very few discriminative features for the entities in both datasets (e.g. male vs female) and hence a more sophisticated attention mechanism would help.

In addition to the above specific use cases, we would like to discuss insights on following general concepts that naturally occur when learning over multiple graphs:

- **Entity Overlap Across Graphs.** In terms of overlap, one needs to distinguish between *real* and *known* overlap between entities. For the known overlap between entities, we use that knowledge for linkage loss function L_{lab}. But our method does not need to assume either types of overlap. In case there is no real overlap, the model will learn embeddings as if they were on two separate graphs and hence will only provide marginal (if any) improvement over state-of-art embedding methods for single graphs. If there is real overlap but no known overlap (i.e., no linked entity labels), the only change is that Equation (13) will ignore the term $(1 - b) \cdot L_{lab}$. Table 3 shows that in this case (corresponding to AUPRC (Unsupervised)), we are still able to learn similar embeddings for graph entities corresponding to the same real-world entity.

- **Disproportionate Evidence for entities across graphs.** While higher proportion of occurrences help to provide more evidence for training an entity embedding, the overall quality of embedding will also be affected by all other contexts and hence we expect to have varied entity-specific behavior when they occur in different proportions across two graphs

- **Ambiguity vs. Accuracy.** The effect of ambiguity on accuracy is dependent on the type of semantic differences. For example, it is observed that similar entities with major difference in attributes across graphs hurts the accuracy while the impact is not so prominent for similar entities when only their neighborhood is different.

B Implementation Details

B.1 Additional Dataset Details

We perform light pre-processing on the dataset to remove self-loops from triples, clean the attributes to remove garbage characters and collapse CVT (Compound Value Types) entities into single triplets. Further we observe that there is big skew in the number of types between D-IMDB and D-FB. D-FB contains many non-informative type information such as $\#base.*$. We remove all such non-informative types from both datasets which retains 41 types in D-IMDB and 324 types in D-FB. This filtering does not reduce the number of entities or triples by significant number (less than 1000 entities filtered)

For comparing at scale with baselines, we further reduce dataset using similar techniques adopted in producing widely accepted FB-15K or FB-237K. Specifically, we filter relational triples such that both entities in a triple contained in our dataset must appear in more than k triples. We use $k = 50$ for D-FB and $k = 100$ for D-IMDB as D-IMDB has orders of magnitude more triples compared to D-FB in our curated datasets. We still maintain the overall ratio of the number of triples between the two datasets.

Positive and Negative Labels. We obtain 500662 positive labels using the existing links between the two datasets. Note that any entity can have only one positive label. We also generate 20 negative labels for each entity using the following method: (i) randomly select 10 entities from the other graph such that both entities belong to the same type and there exist no positive label between entities (ii) randomly select 10 entities from the other graph such that both entities belong to different types.

B.2 Training Configurations

We performed hyper-parameter grid search to obtain the best performance of our method and finally used the following configuration to obtain the reported results:
– Entity Embedding Size: 256, Relation Embedding Size=64, Attribute Embedding Size = 16, Type Embedding Size = 16, Attribute Value Embedding Size = 512. We tried multiple batch sizes with very minor difference in performance and finally used size of 2000. For hidden units per layer, we use size = 64. We used $C = 50$ negative samples and $Z = 20$ negative labels. The learning rate was initialized as 0.01 and then decayed over epochs. We ran our experiments for 5 epochs after which the training starts to convert as the dataset is very large. We use loss weights b as 0.6 and margin as 1. Further, we use $K = 50$

random walks of length $l = 3$ for each entity We used a train/test split of 60%/40% for both the triples set and labels set. For baselines, we used the implementations provided by the respective authors and performed grid search for all methods according to their requirements.

C Contextual Information Formulations

Here we describe exact formulation of each context that we used in our work.

Neighborhood Context: Given a triplet (e^s, r, e^o), the neighborhood context for an entity e^s will be all the nodes at 1-hop distance from e^s other than the node e^o. This will capture the effect of other nodes in the graph surrounding e^s that drives e^s to participate in fact (e^s, r, e^o). Concretely, we define the neighborhood context of e^s as follows:

$$\mathbf{N_c}(e^s) = \frac{1}{n_{e'}} \sum_{\substack{e' \in \mathcal{N}(e^s) \\ e' \neq e^o}} \mathbf{v^{e'}} \tag{1}$$

where $\mathcal{N}(e^s)$ is the set of all entities in neighborhood of e^s other than e^o. We collect the neighborhood set for each entity as a pre-processing step using a random walk method. Specifically, given a node e, we run k rounds of random-walks of length l and create the neighborhood set $\mathcal{N}(e)$ by adding all unique nodes visited across these walks.

Please note that we can also use $\max$ function in (1) instead of sum. $\mathbf{N_c}(e^s) \in \mathbb{R}^d$ and the context can be similarly computed for object entity.

Attribute Context. For an entity e^s, the corresponding attribute context is defined as

$$\mathbf{A_c}(e^s) = \frac{1}{n_a} \sum_{i=1}^{n_a} \mathbf{a_i^{e^s}} \tag{2}$$

where n_a is the number of attributes. $\mathbf{a_i^{e^s}}$ is the embedding for attribute i. $\mathbf{A_c}(e^s) \in \mathbb{R}^y$.

Type Context. We use type context mainly for relationships i.e. for a given relationship r, this context aims at capturing the effect of type of entities that have participated in this relationship. For a given triplet (e^s, r, e^o), we define type context for relationship r as:

$$\mathbf{T_c}(r) = \frac{1}{n_t^r} \sum_{i=1}^{n_t^r} \mathbf{v_i^{t'}} \tag{3}$$

where, n_t^r is the total number of types of entities that has participated in relationship r and $\mathbf{v_i^{t'}}$ is the type embedding that corresponds to type t. $\mathbf{T_c}(r) \in \mathbb{R}^q$.