Cycle Consistency for Knowledge Graphs Representations

Nilesh Kulkarni Shengyi Qian Dandan Shan Richard Higgins
University of Michigan
{nileshk, syqian, dandans, relh}@umich.edu

ABSTRACT

We demonstrate an approach to learning representations for entity and relational embeddings in knowledge graphs by exploiting the cycles of a knowledge graph, and learning an inverse prediction model for identifying edges from nodes. We treat edges as operators and force the embedding space to be consistent with simple operations that allow for loop closure while still being predictive of the relations between nodes. Specifically, we evaluate our model on the FB15K-237 dataset and show how our approach significantly improves over the existing representation learning TransE model from Bordes *et al.* [2] on two evaluation metrics.

KEYWORDS

representation learning, cycle consistency

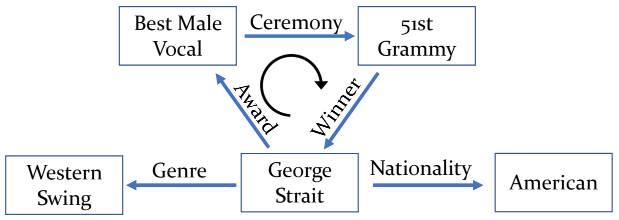


Figure 1: A subset of the graph showing a cycle and other relationships. The cycle is shown by the black arrow.

1 INTRODUCTION

Knowledge graphs (KGs) are a large repository of fact triples representing knowledge between entities (or words) represented as (*subject*, *relation*, *object*) also denoted as (h, r, t). Learning a good knowledge graph representation is crucial for many applications including web search, question answering, and playing games like jeopardy. Learning from knowledge graphs began with early work from Brachman *et al.* [3], which categorized core issues in knowledge graphs and facts. Since 1995, there have been various large scale projects on constructing knowledge graphs and encyclopedias [11] to help machines reason about the data. Since symbolic relations between entities can be stored in a graph, a lot of graph based approaches have been designed that allow one to query knowledge graphs. A typical tasks involve answering questions regarding a pair of entities in a graph, also referred to as the task of *link prediction.* These graph based methods fail to scale with an increasing number of fact triples, due to the data sparsity and computational inefficiency.

Challenges in scalability in KGs have been recently handled by learning representations for entities and relations on the knowledge graphs. Recent works in the past decade [2, 6] have shown representation learning for entities and relations scales well with the sparse nature of KGs. There have been significant improvements in link prediction performance by following a paradigm of learning a low-dimensional embedding space. Most existing representation learning methods [2, 6, 21] learn representations for entities just from fact triples, without even considering the reality that all entities are part of a large sparse graph. Other methods like Grove *et al.* [7] propose to learn representations by treating triples as part of graphs and then learning representations by performing random walks.

However, no methods on knowledge graphs use cycles on the graph to learn representation. Our key hypothesis and insight is that we can improve representation learning by imposing constraints related to the underlying graph structure of these knowledge bases. We show that our approach of using cycles draws inspiration for learning consistent representations and also from loop-closure ideas in Simultaneous Localization and Mapping (SLAM) [13]. Loop-closure enhances the robustness of the localization by imposing the constraint that the displacement of a mobile system is zero when the mobile system reaches the same location. We show that our first intuition to learn entity embedding that have no net change in representation while moving along a cycle of the graph show better empirical performance on the task of link prediction.

Representation learning methods treat relations as some kind of translation in embedding space [2, 21]. Our second intuition is loosely related to [14], where an agent in state s_0 takes an actions a to reach a state s_1. In [14], an internal curiosity model proposes to learn an inverse kinematic action prediction model for a just from state representations (s_0, s_1), in an unsupervised setting. Drawing parallels for a given triplet (h, r, t) we propose to learn a relation prediction model just from entity representations for (h, t). Such formulation, when coupled with the first intuition, results in learning better representations for entities.

To the best of our knowledge no other methods do these additions, and we show how these additional constraints help improve the performance of the TransE [2] model. Since our contributions are orthogonal to other trans* [2, 9, 19] models they can be applied to them as well. To summarize our key contributions are as follows.

1. Cycle Consistent Entity Representation. We show how incorporating a loop-closure, also referred to as learning cycle consistent representations, on entities in a knowledge graph helps improve performance on the task of link predictions for the Freebase 15K dataset [4].

2. Relation Predictive Models. We show how learning a predictive model conditioned on entity representation to predict the relations between entities helps to further improve performance on link prediction.
Code hosted at https://github.com/nileshkulkarni/cycleEmbedding

2 RELATED WORK

Many representation learning papers use neural networks and supervised learning techniques from the machine learning and computer vision communities. Separately, there is a large body of work on cycle consistency in the machine learning and computer vision communities [16, 20]. There has been a cross-pollination of ideas between the data mining and machine learning silos, but it does not appear to have extended to properly incorporating cycle consistency losses. In [12], the authors used a graph built from nearest visual neighbors in a dataset to enable a cycle consistency constraint. But this was thoroughly a computer vision paper, without explicit relation to the graph mining community. Furthermore, it wasn't thought of that this cycle consistency could generate node representations in a graph, but rather it was a simplistic method for generating image pairs.

2.1 Representation Learning

Node Representations. Node representation learning is a topic of recent interest, as the graph community has discovered its potential. Learning allows for better representations that are suitable for various downstream tasks. The core idea of any representation is to derive embeddings, usually in some kind of a euclidean space where the nodes follow some kind of ordering. For example, approaches like [7, 15] aim to develop embeddings for every node such that close nodes in the graph are also close in an embedding space, often referred to as the proximity embeddings. These embeddings are generally useful for solving semantic tasks, or understanding local community behaviours.

Most node embeddings are direct encoding approaches, where an embedding is calculated for node IDs and then retrieved as needed. Historically, the research community focused on methods that depended on matrix factorization. These methods were based on dimensionality reduction literature and included laplacian eigenmaps and inner-product methods. However, because large graphs create computational problems, newer methods for node embeddings shifted to focus on random-walk approaches. These random-walk approaches were valuable because they let one efficiently take an approximate measure of graph proximity. Notable random-walk methods include Deep Walk [15], node2vec [7], and struct2vec [17]. Other random walk methods include HARP: Extending random-walk embeddings via graph pre-processing, and GraRep.

Beyond random-walks, there are approaches that use encoder and decoder architectures to develop understandings of nodes. This series of methods includes the Large-scale Information Network Embeddings (LINE), a method for embedding neighborhoods which uses metrics related to KL-divergence as a loss function. This encoder-decoder approach led to a generalized problem formulation for encoders and decoders, with neighborhoods and autoencoders featuring prominently. This family of methods includes both the Deep Neural Graph Representations (DNGR) and Structural Deep Network Embeddings (SDNE) approaches. Recent works that have incorporated convolutional neural networks include Graph Convolutional Networks (GCNs), column networks, and the GraphSAGE algorithm [8].

Edge Representations. Edge representation learning has not been widely explored in the community as typically edge embeddings are just set as some kind of linear combination of node embeddings. One of the applications proposed by [1] is to use edge embeddings to learn a low dimensional factorization of node embeddings. In this work, the goal was that a representation between adjacent nodes should be similar and can be distilled down to an edge. The authors used it to reduce the number of parameters in their network. However, there is another school of thought that explores having edges with representations independent to that of nodes, with them instead encoding some kind of relation like between concepts in a knowledge graph [18].

2.2 Knowledge Graph Embedding

Knowledge graph embedding aims to embed entities and relationships of a knowledge base into low-dimensional spaces. Usually, one uses (entity, relation, entity) triplets to represent the knowledge as a graph. A one-hot vector is one simple way to do embedding for a graph, but it has scalability constraints when there are too many entities and relations. Thus, several graph embedding methods appear that help solve the data volume problem and try to further better represent the graph.

TransE. TransE is the first work in a series of translation embedding models. The basic idea of TransE is to make the sum of the head vector and relation vector close to the tail vector and use an L1 or L2 metric as the measurement of this distance. During training with negative sampling, it only uses a margin loss for supervision. This margin loss enforces that a valid or correct triplet has a lesser distance than an invalid or corrupted triplet. The limitation of TransE is that it only works well on one-to-one models.

TransH. TransH, short for translating on hyperplanes, is a follow-up improvement on the above translation embedding method. The basic idea of transH is to interpret the relation between entities through a projection operation on a new hyperplane. To realize this function, the relation embedding is separated into two parts, the norm vector (W) of the hyperplane and the translation vector (d) on the hyperplane. First, one projects the head vector and tail vector to the hyperplane with W, and then one makes sure the sum of the new head vector and translation vector d are close to the new tail vector. Its loss function and training method is the same as TransE. But TransH is suitable for one-to-many, many-to-one and many-to-many relations, while not increasing the complexity.

ComplEX. In the complEX embedding, the Hermitian dot product of vectors with complex numbers involves the conjugate-transpose of one of the two vectors. As a result, the dot product is not symmetric any more. The antisymmetric relations can receive different scores according to the ordering of the entities involved. Thus, complex vectors can effectively capture antisymmetric relations while retaining the efficiency benefits of the dot product, which is linear in space and time complexity.

2.3 Cycle Consistency as Supervision

A general property of a representation is that it should be consistent with what other representations say for nodes, images, or videos. Cycle consistency employs the idea of using loop-closures to learn representations that are consistent with one another. In some-sense it imposes constraints on various predictions or parameters as part of generating self-supervision. This self-supervision can serve as a training signal in learning better representations. Computer Vision over the last 5 years has seen a lot of progress employing huge volumes of data and using unsupervised learning for constructing diverse representations across mediums (videos, images, and more) [5, 10, 20].

Cycles on unlabeled videos. Wang et. al.[20] uses cycle-consistency in time as a free supervisory signal to learn visual correspondence from unlabeled videos. The key idea is to obtain unlimited supervision for correspondence by tracking backward and then forward in the video, before using the inconsistency between starting and ending points as a loss function. The paper shows their method can generalize across a range of visual correspondence tasks including video object segmentation, keypoint tracking, and optical flow (all without any finetuning). The results are promising because the system does not need human annotations. The ensuing questions are two fold: (1) how does one apply this method to work on noisier data such as arbitrary YouTube videos [5]? (2) How does one close the gap in performance between self-supervised and supervised methods?

Cycles on Images. There has been a huge body of work on using cycles on images, the goal being to enable a consistent representation across images. In [12], the authors wish to identify similar but not identical image pairs for learning semantic feature representations. Their goal is that these close but not identical images could be useful as a signal for training a network to recognize semantically meaningful features. The paper argues that nearest neighbors are too similar for learning semantic information, and that using such neighbors is similar to using egomotion as a supervisory signal. The paper proposes an unsupervised and semisupervised method for mining image pairs and generating data to use in a downstream classification task. Their mining positive image pair technique involves looking for nearest neighbors of nearest neighbors that form a cycle. Their mining negative image pair technique comes from looking for geodesically distant pairs. They use a Siamese network to predict whether two images are semantically similar (same class) or different. Unfortunately, they show no real improvement over their baselines. [12].

3 METHOD

We propose an unsupervised learning approach to representation learning for nodes and edges in a graph. The key high-level idea is that nodes in a graph are related to each other by edges. Edges can be considered as operators between nodes, and hence allow us to connect or relate different nodes in the graph. Representations on the graph should be consistent with the graph itself. When we move along a cycle on the graph, we propose to use loop closure as a supervisory signal to learn node and edge embeddings.

1. **Notation** Throughout this paper we will refer to a graph $G = (V, E)$ as a graph with nodes V and edges represented as an edge list E. We define u_i as the i^{th} node in the graph. We define e_{ij} to represent a directed edge from node u_i to u_j. We use d_N to represent the dimension of the node embedding and d_E is the dimension of the edge embedding.
2. **Representation for Nodes.** We define $f_\theta(u_i)$ as the node embedding for the i^{th} node. These representations are a kind of lookup table, conditioned on the properties of the node. In a knowledge graph with words and concepts they would be similar to a word embedding, while in the case of an image based graph they would be image features extracted from a pre-trained network, or recovered using a function such as a convolutional network.

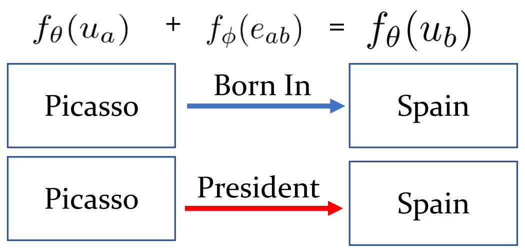


Figure 2: Examples showing the relation between entity embeddings and the relation embeddings that leads to an output embedding. The "Born In" edge is the positive edge while the "President" is the corrupted/negative edge. We sample such positive and negative edges to train with the L_{node}.

3. **Representations for Edges**. We treat our edges as non-linear consistent operators that manipulate various node embeddings. Our graph has directed edges, with a source node u_s and target data u_t. We treat the embedding for the edge e_{st} as an operator embedding. We define $f_\phi(e_{st})$ to represent the embedding for this edge.

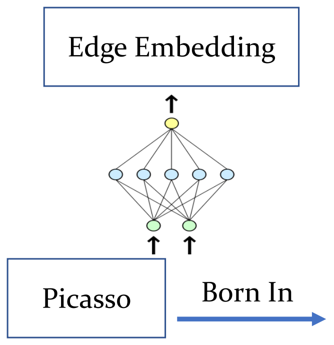


Figure 3: Our edge embeddings conditioned on the head node and the edge type corresponding to the triplet (h, r, t).

4. **Relation between edge and node embedding.** Nodes are connected by edges in the graph. But for the case of representation learning we would like to adopt the following relationship between edges and nodes. Consider g_γ to be an operator function that relates edges and nodes. We define below the relationship between u_s, u_t, e_{st}

$$g_\gamma[f_\theta(u_s), f_\phi(e_{st})] = f_\theta(u_t) \tag{1}$$

5. **Traversing a cycle**. We plan to learn representations consistent with Equation 1. Consider the smallest possible cycle traversal on a graph which involves using two nodes u_i and u_j, and their corresponding edges e_{ij}, e_{ji}. We represent, $u_i \longrightarrow e_{ij} \longrightarrow u_j$ as the *step* from u_i to u_j via e_{ij}. Now consider the following path, p, $u_i \longrightarrow e_{ij} \longrightarrow u_j \longrightarrow e_{ji} \longrightarrow u_i$ which also represents a cycle starting at u_i. A *path* is a sequence of steps as defined in Equation 2.

$$\begin{aligned} p &= [s_1, s_2, \ldots, s_l] \\ s_1 &= u_{i_1} \longrightarrow e_{i_1 i_2} \longrightarrow u_{i_2} \\ s_2 &= u_{i_2} \longrightarrow e_{i_2 i_3} \longrightarrow u_{i_3} \\ &\vdots \\ s_l &= u_{i_l} \longrightarrow e_{i_l i_{l+1}} \longrightarrow u_{i_{l+1}} \end{aligned} \tag{2}$$

There are multiple cycles that can exists starting at u_i. For brevity we represent C_{u_i} as the set of all cycles starting at u_i. Since every cycle is a path, using the definition in Equation 1 and Equation 2 we can now define how node representations change as we move along a path p, of length l, using an iterative definition of 1

$$\hat{u_j} = g_\gamma[f_\theta(u_{j-1}), f_\phi(e_{j-1,j})] \qquad \forall j \in [2, 3, \ldots, l, l+1] \tag{3}$$

We use $\hat{u_k}$ for any node k to denote the node representation obtained by using equation 1, and u_k represent the node representations obtained using the definition of node representation. Please note that the notation for u is slightly overloaded here. $\hat{u}$ represents a translated embedding as opposed to the entity u does.

6. **Grounding representations with auxiliary supervision.** There is an obvious solution to satisfy the cycle loss which converges all the node and edge representations to 0. To avoid learning such a representation we use the structure of the graph to ground our node representations. We believe that a good node representation should have the ability to predict various edges between graphs, and also be capable of predicting the type of edge. We use link-prediction as this supervision, along with predicting edge types. $h_\delta[f_\theta(u_i), f_\theta(u_j)]$ represents a probability distribution over edge types for given pair of node u_i & u_j

3.1 Optimization and Loss functions

1. **Node Translation Loss.** Now we can define the node translation loss using a margin loss. For a given triplet $(u_{i_1}, e_{i_1,i_2}, u_{i_2})$ in the dataset we sample a negative triplet by corrupting the edge e_{i_1,i_2} an example is shown in Figure 2. Now similar to the TransE model [2] we consider translating the the representation for node i_1 to get the representation for node i_2. Let $u_{\hat{i_2}+}$ denote the translation along the corrupted edge. We use a margin defined by Δ to create a node loss.

$$L_{\text{node}} = \max(0, \|f_\theta(u_{i_2}) - u_{\hat{i_2}+}\| + \Delta - \|f_\theta(u_{i_2}) - u_{\hat{i_2}-}\|) \tag{4}$$

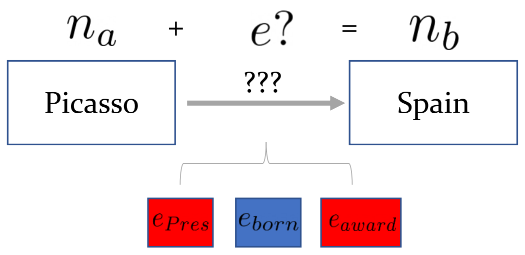
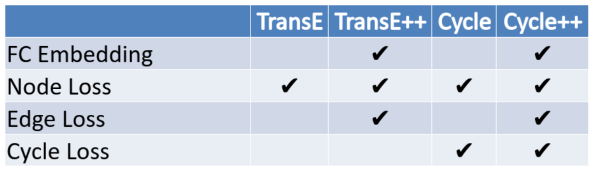


Figure 4: An example scenario of predicting an edge given a head and tail node and a classification task across the 237 possible relationships.



	TransE	TransE++	Cycle	Cycle++
FC Embedding		✓		✓
Node Loss	✓	✓	✓	✓
Edge Loss		✓		✓
Cycle Loss			✓	✓

Figure 5: A comparison between prior work TransE [2] and proposed variants showing an ablative analysis over the various loss functions.

2. **Cycle Consistency Loss.** Now we can define the cycle consistency loss using a margin loss. For a given cycle denoted by p with source node u_{i_1}. We consider a positive cycle and a corresponding corrupted cycle extracted from the dataset. A corrupted cycle is constructed by randomly corrupting one edge of a positive cycle. Consider $u_{\hat{0}+}$ as the representation for the source node after traversing the positive cycle and $u_{\hat{0}-}$ as the representation for the source node after traversing the negative cycle. . We denote Δ as the margin between the representation for the positive and the negative cycle. Figure 7 shows an example of the positive and negative cycle

$$L_{\text{cyc}} = \max(0, \|f_\theta(u_{i_1}) - u_{\hat{i_1}+}\| + \Delta - \|f_\theta(u_{i_1}) - u_{\hat{i_1}-}\|) \tag{5}$$

3. **Edge Prediction Loss.** For a given triplet $(u_{i_1}, e_{i_1,i_2}, u_{i_2})$ we predict the edge type between the u_{i_1} and u_{i_2} an example is shown in Figure 4. We add a cross entropy loss over the predictions:

$$\begin{aligned} L_{\text{edge}} &= -\log(h_\delta(f_\theta(u_{i_1}, f_\theta(u_{i_2}))[\text{gt}]; \\ &\quad \text{gt is index of the correct edge type} \end{aligned} \tag{6}$$

4. **Total Loss.** Our final loss function for optimization is defined as the sum of all the losses. In experiments we analyse the effect of various loss functions, and report results as an ablation.

$$L_{\text{total}} = L_{\text{node}} + L_{\text{edge}} + L_{\text{cyc}} \tag{7}$$

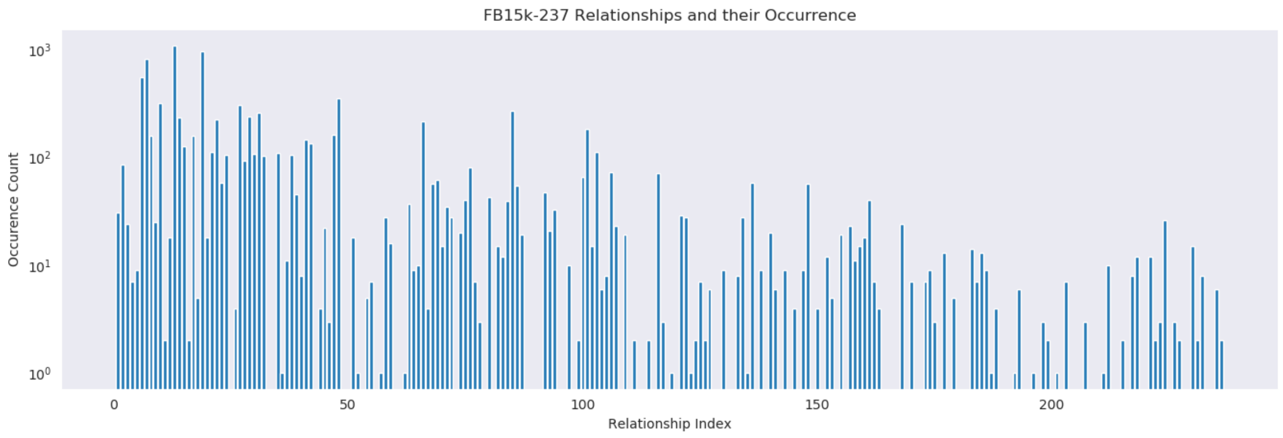


Figure 6: The occurrence of all relationships in the test dataset. Note the log scale and quite uneven distribution of relationships.

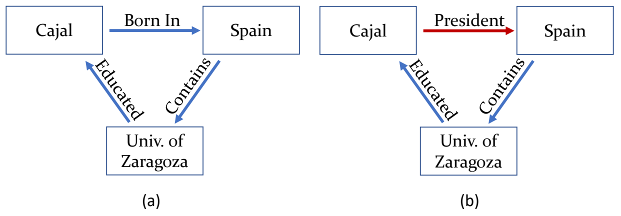


Figure 7: An example positive cyclic relationship (a) and corrupted cycle (b) use to compute the cycle loss.

5. **Training the network**. We jointly train all our parameters by combining the cycle and the edge prediction losses. To train the cycle loss we sample random cycles on the graph. We train the parameters $\theta, \phi, \gamma, \delta$, where all the functions are multi-layer perceptrons or lookup tables. Table 5 shows different methods that use different loss functions described.

4 DATA

We test our dataset on the FB15k-237 knowledge graph dataset. This knowledge graph contains 15K entities as nodes and 237 distinct relationship types connecting them. This leads to a count of 310K triplets of (head, relationship, tail). An example triplet might be that the Kingdom of Jordan's (head) form of government (relationship) is a Unitary Republic (tail).

The dataset contains pre-existing training, validation, and testing splits. We use these splits, but reduce them to a subset of nodes that are part of a cycle. This reduction leads to a set of 9195 nodes which we evaluate relationships from. In our results, we denote results obtained from the full dataset with an asterisk, while our results are reported for our subset contacting nodes in cycles. 9895 edges are used for link prediction.

We initially attempted to identify cycles of varying lengths for use in training. However, these cycles proved computationally difficult to find, so we limited the scope of our cycle's method to working on cycles of length 3. To find these cycles, we first sample an edge in the existing training graph. We then look at the tail node and see if it is the predecessor to any other nodes in the graph. We compare this list against the successor set of the original head node and take the intersection of these node lists to find nodes that form a triangle from head to tail to new node to head. Now that we have a cycle, we sample an edge that connects a tail node to the new node, as well as an edge that connects the new node to the head node.

We design two data loaders for our two relevant evaluations of interest. For the link prediction task, we can use the same dataloader structure as is used in training, albeit for a validation or testing task.

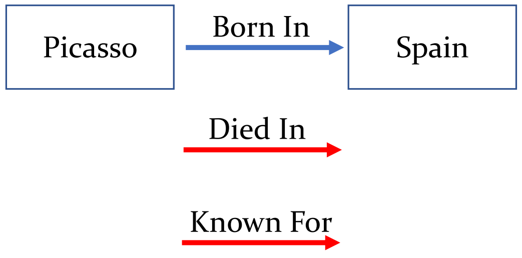


Figure 8: An example set of link predictions. The blue one is the correct relation and the red ones are the incorrect relations and an example top 3 are shown.

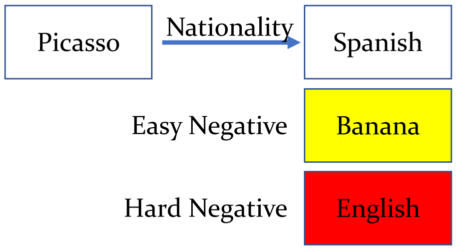


Figure 9: An example set of potential incorrect and correct triplets.

However, for the triplet classification task, we need to equally sample correct triplets alongside hard negative examples. It's clear why such a procedure is necessary, as distinguishing against the easy negative example in Figure 8 should be semantically trivial.

We use the existing hard negative vs. correct triplet dataset split as defined by [4].

5 EXPERIMENTS

We empirically evaluate our methods of learning node and edge representations on graphs. Following past works [2, 21] on knowledge graph embedding, we conduct our experiments on Freebase and evaluate the embedding using two tasks: link prediction and triplets classification. Our experiments aim to address the following question: Does cycle consistency help learning knowledge base embedding?

5.1 Experimental setup

Datasets. Freebase is a huge dataset of general facts. Following [2], we sample the subset FB15k-237, which contains 310K triplets with 15K entities and 237 relationships.

Baselines. We mainly compare our proposed method with TransE [2]. All four models we'd like to report are shown in Figure 5. To ensure fair comparison, they are the same models except loss functions are different. Cycle is TransE with cycle loss, and Cycle++ is TransE++ with cycle loss. If Cycle outperforms TransE and Cycle++ outperform TransE++, it means that our proposed approach is effective.

Training details. The embeddings sizes for our node embeddings are all 128. The TransE and Cycle model using embedding as parameter in the network while the TransE++ and Cycle++ models have 2 layer MLP to add non-linearity to the embeddings. The TransE and the cycle model do not have the edge prediction loss and trained only to minimize other losses. The TransE++ and Cycle++ model have the edge predictions loss, and the edge embedding is conditioned on the head node type. All our node embeddings are normalized to a hyper-sphere. We train all our networks with a batch size of 10K and sample one negative sample per example in the batch for the node and the cycle loss. All the networks are trained for 5000 epochs with stochastic gradient descent as the optimizer.

5.2 Link Prediction

Evaluation Protocol. Typical link prediction aims to complete a triplet (h, r, t) with h or t missing. Given (r, t), can the embedding predict h? Given (h, r), can the embedding predict t? Instead of top-1 accuracy, the metric focuses on ranking candidate entities in the knowledge base. One then reports HITS@K, which means an accurate prediction if the correct entity appears within the top-K candidates predicted by the model. We implement a variation of this link prediction task, where head and tail nodes are presented, and the model must rank relationships to determine whether a valid edge is predicted within the top K elements. All experimental link prediction is conducted on this metric. It is instead a link prediction task of choosing r given a (h, t) pair.

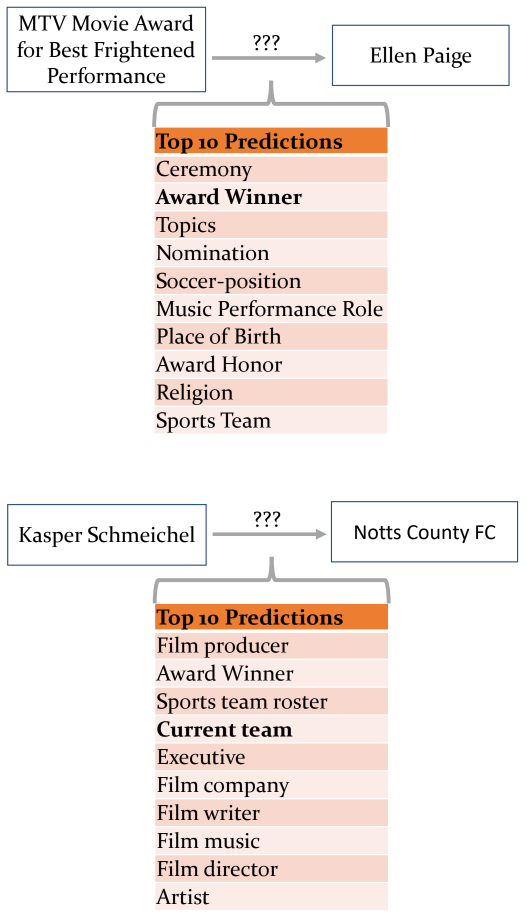


Figure 10: Two example link prediction qualitative results and the Top 10 results that our Cycle++ model predicted could be the relationships for each. The correct result is highlighted at number 2 and number 4 respectively. We use link prediction metrics to get quantitative performance for our models and report these results in Figure 1.

Method	HITS@10	HITS@5	Top-1
TransE	65.3%	60.6%	41.1%
Cycle	83.6%	71.3%	42.3%
TransE++	89.9%	84.8%	62.7%
Cycle++	**94.0%**	**90.2%**	**70.5%**

Table 1: Evaluation of Link Prediction for HITS@10, HITS@5, and Top-1 as described in Section 5.2

Results. Example qualitative results for link prediction are shown in Figure 10. We report performance of all the methods on HITS@10, HITS@5 and Top-1 in Table 1. The model with cycle loss outperforms corresponding TransE model without cycle loss significantly, which indicates that cycle consistency is very important.

Both the gains from the cycle consistency and inverse edge prediction model are significant and shows 30% improvement. We see the using cycles leads to representations that are consistent with the embedding space operations.

5.3 Triplet Classification

Evaluation Protocol. Triplet classification aims to predict if a triplet (h, r, t) is valid or not. It is a binary classification on triplets. We sample negative examples following the same policy as [2] to ensure the comparison is fair. For triplet classification, we report the accuracy on the test set. Random performance will be 50%, as a random model has a 50% chance to correctly guess triplet validity. For each of the models, we extract the median scores for valid and invalid triplets and then run a grid search at various increments between the two medians to choose a threshold for use in final evaluation. Then we report test results on this chosen threshold below in 2.

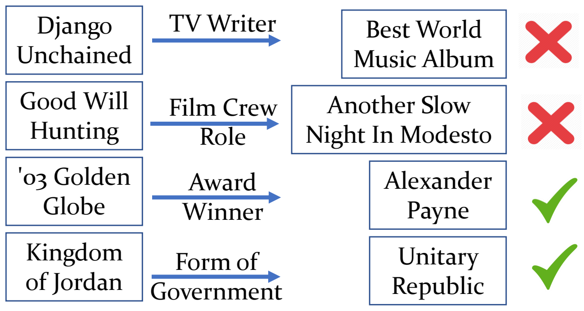


Figure 11: A set of incorrect and correct triplets and our Cycle++ model's correct prediction of these triplets.

Methods	Accuracy
TransE	56.29%
Cycle	59.65%
TransE++	64.17%
Cycle++	**64.97%**

Table 2: Evaluation of Triplet Classification.

Results. Example qualitative triplet classification results are shown in Figure 9. We also report accuracy in Table 2. Cycle++ outperforms TransE++, and Cycle outperforms TransE. Cycle++ and TransE++ performance is very similar.

6 CONCLUSIONS

We validated our hypothesis that the reverse kinematics and loop closure motivations from robotics transfer applicably to graph knowledge representation learning. The added benefits of working on a discrete structure (rather than more continuous real world terrain) led to state-of-the-art results for link prediction tasks. In future we would like to use the cycle and the inverse edge prediction model to show improvements on the representation methods in general. We only exploited tri-cycles on the knowledge graph as computing cycles longer than this was very expensive. Future work would change link prediction metrics and add longer cycles. This course project has been a valuable opportunity for us to with the semantically rich world of graphs and apply methods from our own fields to work here.

REFERENCES

[1] Sami Abu-El-Haija, Bryan Perozzi, and Rami Al-Rfou. Learning edge representations via low-rank asymmetric projections. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1787–1796. ACM, 2017.
[2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.
[3] Ronald J Brachman and Hector J Levesque. *Readings in knowledge representation*. Morgan Kaufmann Publishers Inc., 1985.
[4] Niel Chah. Freebase-triples: A methodology for processing the freebase data dumps. *arXiv preprint arXiv:1712.08707*, 2017.
[5] Weifeng Chen, Shengyi Qian, and Jia Deng. Learning single-image depth from videos using quality assessment networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.
[6] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.
[8] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.
[9] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 687–696, 2015.
[10] Nilesh Kulkarni, Abhinav Gupta, and Shubham Tulsiani. Canonical surface mapping via geometric cycle consistency. *arXiv preprint arXiv:1907.10043*, 2019.
[11] Douglas B Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.
[12] Dong Li, Wei-Chih Hung, Jia-Bin Huang, Shengjin Wang, Narendra Ahuja, and Ming-Hsuan Yang. Unsupervised visual representation learning by graph-based consistent constraints. In *European Conference on Computer Vision*, pages 678–694. Springer, 2016.
[13] Paul Newman and Kin Ho. Slam-loop closing with visually salient features. In *proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 635–642. IEEE, 2005.
[14] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.
[15] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.
[16] Stephen Phillips and Kostas Daniilidis. All graphs lead to rome: Learning geometric and cycle-consistent representations with graph convolutional networks. *arXiv preprint arXiv:1901.02078*, 2019.
[17] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394. ACM, 2017.
[18] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.
[19] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.
[20] Xiaolong Wang, Allan Jabri, and Alexei A Efros. Learning correspondence from the cycle-consistency of time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2566–2576, 2019.
[21] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Twenty-Eighth AAAI conference on artificial intelligence*, 2014.