Supplementary Information for SDE-KG: A Stochastic Dynamic Environment for Knowledge Graphs

Varun Ranganathan[1 [0000−0001−6865−3217]] and Natarajan Subramanyam[1 [0000−0002−8689−5137]]

PES University, Bangalore KA 560085, IN varunranga1997@hotmail.com, natarajan@pes.edu
http://www.pes.edu

Abstract. In this document, information is provided for the reproducibility of the experiments conducted in the paper. The codes that are used to perform the experiments are publicly available at github.com/varunranga/SDEKG-KGRL. Results for the experiments conducted are available at tinyurl.com/SDEKG-KGRL-Results.

1 Dependencies

1.1 Hardware dependencies

Multiple systems were used to train and test SDE-KG on various embedding methods and datasets. These experiments were conducted in a distributed manner across 4 systems, each with the following hardware specifications:

- CPU : 8th Generation Intel i7 Core Processor
- RAM : 16GB
- Hard disk space : 1TB

1.2 Software dependencies

Table 1 provides a list of software dependencies that are required to run the required codes. To run the codes available at the aforementioned link, please make sure the packages and softwares are installed on the system.

2 Executing code

To execute the main script 'main.py', use the 'python3' command to invoke the Python3 interpreter, and send the argument 'main.py' to execute statements from that script. Additional command line arguments must be provided to set hyperparameters and models that need to be trained. The command 'python3 main.py –help' will provide necessary information about the arguments that can

Package / Software	Version
Ubuntu	16.04
Python3	3.5
python3-tk	3.x
Cuda (To run on GPU)	9.0
Cudnn (To run on GPU)	7.4
Tensorflow	1.12
Tensorflow-gpu (To run on GPU)	1.12
Numpy	1.16
Tqdm	4.22
Matplotlib	2.0.2
Pandas	0.20.3
Pickle	(Available with Python3)
Argparse	(Available with Python3)
Pprint	(Available with Python3)
Math	(Available with Python3)

Table 1. Software dependencies for running provided codes.

be used to invoke the program. All generated results are available in 'SDEKG-KGRL Results.zip' compressed file in the given link. On extracting the zipped file, a 'Results' directory will be generated. The directory structure will follow this pattern: 'Results/[Dataset|Environment]_<Dataset>_<Embedding Method >'. The binary pickle files starting with 'Dataset' contain the required embeddings for components of the knowledge graph, along with the relevant information about the dataset. The binary pickle files starting with 'Environment' contain the trained SDE-KG environments and information regarding training and evaluation. To retrain SDE-KG models with the hyperparameters used in the experiments, the shell scripts '1.sh', '2.sh', '3.sh', and '4.sh' can be executed. Results for the experiments conducted are available at `tinyurl.com/SDEKG-KGRL-Results`. To view the contents of any pickle file, please use the 'view_contents.py' script with the short argument '-fn' or long argument '–filename' to load and display contents of the file. Alternatively 'Results/Results.xlsx' contains all the results consolidated in a spreadsheet.

3 Experimental Information

This section provides additional details for the reproducibility of experiments in the paper. Table 2 gives the dataset statistics, figure 1 shows the computational graph of SDE-KG for a simplified understanding. Algorithms 1 and 2 and their explanations give the detailed working of the training and evaluation procedure.

The small knowledge graphs chosen for the experiments were Countries, Kinship, UMLS as used by Rocktäschel and Riedel in [5]. The two large-scale knowledge graphs chosen are FB15K-237 and NELL995 as used by Xiong et al. in [6]. The Tensorflow [1] package was used to implement SDE-KG. The code used to create the embeddings was adopted from OpenKE [3]. The hyperparameters

used to train SDE-KG for the different types of datasets have been adopted from [2]. All models were trained using the Adam Optimizer [4].

Dataset	#entities	#relations	#facts	#degree	
				average	median
Countries	271	2	1,159	4.35	4
UMLS	135	46	6,529	38.63	28
Kinship	104	25	10,686	82.15	82
NELL-995	75,492	200	154,213	4.07	1
FB15K-237	14,541	237	272,115	19.74	14

Table 2. Statistics of various datasets used in experiments

3.1 Hyperparameters Used

- Countries
  - Embedding Size: 25
  - Batch Size: 256
  - Margin: 1.0
  - Learning Rate: 0.001
  - Patience: 50 Epochs
  - Sampling Type: Bernoulli
  - Number of Training Steps: 500,000
  - Walk Lengths: 1, 2, 3, 5
  - Number of Walks: 500
- UMLS
  - Embedding Size: 50
  - Batch Size: 256
  - Margin: 1.0
  - Learning Rate: 0.001
  - Patience: 50 Epochs
  - Sampling Type: Bernoulli
  - Number of Training Steps: 500,000
  - Walk Lengths: 1, 2, 3, 5
  - Number of Walks: 500
- Kinship
  - Embedding Size: 50
  - Batch Size: 512
  - Margin: 1.0
  - Learning Rate: 0.001
  - Patience: 50 Epochs
  - Sampling Type: Bernoulli
  - Number of Training Steps: 500,000
  - Walk Lengths: 1, 2, 3, 5

    - Number of Walks: 500
- NELL-995
    - Embedding Size: 50
    - Batch Size: 128
    - Margin: 1.0
    - Learning Rate: 0.001
    - Patience: 50 Epochs
    - Sampling Type: Bernoulli
    - Number of Training Steps: 15,000,000
    - Walk Lengths: 1, 2, 3, 5, 10, 20
    - Number of Walks: 10,000
- FB15K-237
    - Embedding Size: 50
    - Batch Size: 256
    - Margin: 1.0
    - Learning Rate: 0.001
    - Patience: 50 Epochs
    - Sampling Type: Bernoulli
    - Number of Training Steps: 15,000,000
    - Walk Lengths: 1, 2, 3, 5, 10, 20
    - Number of Walks: 10,000

The explanation of the training algorithm for SDE-KG is as follows. In each training step, two random entities are chosen as the source and destination entities. A random relation is also chosen. To train SDE-KG to land on a valid entity available in the knowledge graph, a candidate entity must be chosen. Lines 8 to 31 of algorithm 1 help choose this candidate entity. Triplets with the chosen source entity are selected. These triplets are called candidate triplets. If none exist, the candidate entity is the target entity to achieve a larger jump. If candidate triplets exist, more robust candidate triplets by selecting those which contain the randomly selected relation. Tail entities of the candidate triplets are iterated over to check which of them is closest to the target. The source entity, target entity, candidate entity and relation are converted to their respective vector representations and are trained used to train SDE-KG.

Evaluation procedure of SDE-KG uses a random walk strategy. A random agent is used to generate paths of length L from a source entity to a destination entity by taking path available from the knowledge graph. A random source entity is chosen from a set of all head entities derived from the facts of the knowledge graph. A random triplet is selected from those triplets having the head entity as the randomly selected source entity. If no such triplets are available, the process is repeated to select another random source entity. For the first hop, the relation of the randomly selected triplet is stored. This process is iterated L times. The tail of the last randomly selected triplet is stored as the target. The metrics jump and distance to the nearest entity is calculated by lines 37 and 38 in algorithm 2. The nearest entity can then be compared with a candidate entity, which is selected as described in Algorithm 1. The distance of the predicted next-hop entity from the candidate entity, in comparison with all other entities can

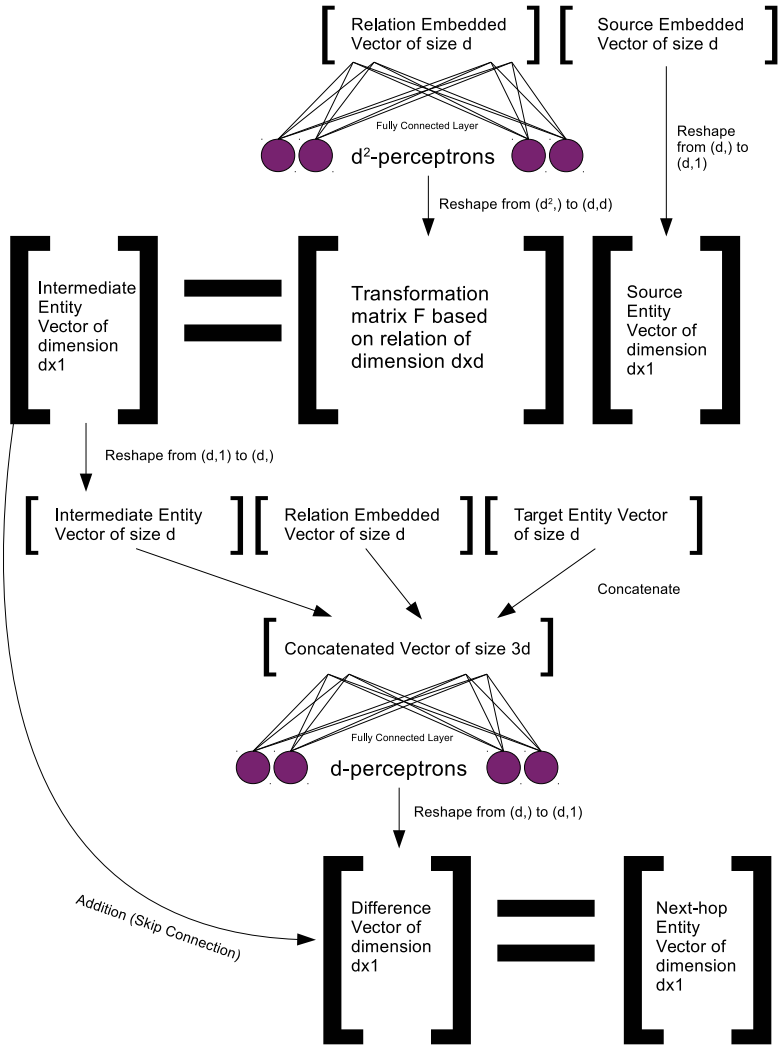

Fig. 1. SDE-KG Computation Graph. Inputs to the procedure are vector representations of the source entity, target entity and the relation. Output is the vector representation of the next-hop entity.

Algorithm 1: Train SDE-KG

Input: Number of Training Steps N, Triplets from the Knowledge Graph T, Embeddings for all entities E_e, Embeddings for all relations E_r
all_entities = List of all entities;
all_relations = List of all relations;
for *step_count = 1 to N* **do**
    head_entity, target_entity = Choose two random entities from all_entities;
    relation = Choose random relation from all_relations;
    candidate_triplets = Empty List;
    foreach *triplet in T* **do**
        if *triplet.head == head_entity* **then**
            Add triplet to candidate_triplets;
        end
    end
    if *length of candidate_triplets == 0* **then**
        candidate_entity = target_entity;
    else
        better_candidate_triplets = Empty List;
        foreach *triplet in candidate_triplets* **do**
            if *triplet.relation == relation* **then**
                Add triplet to new_candidate_triplets;
            end
        end
        if *length of better_candidate_triplets > 0* **then**
            candidate_triplets = better_candidate_triplets;
        end
        candidate_entities = Empty List;
        foreach *triplet in candidate_triplets* **do**
            Add triplet.tail to candidate_entities;
        end
        target_entity_vector = Lookup target_entity vector from E_e;
        candidate_entity = min(candidate_entities, key: distance_to_target = $\sqrt{\Sigma_{i=0}^{d}(entity\_vector_i - target\_entity\_vector_i)^2}$)
    end
    head_entity_vector, target_entity_vector, candidate_entity_vector = Lookup head_entity, target_entity and candidate_entity vectors from E_e;
    relation_vector = Lookup relation vector from E_r;
    Update SDE-KG w.r.t $\Delta\mathcal{L}$;
end

Algorithm 2: Evaluate SDE-KG

Input: Random Walk Length L, Triplets from the Knowledge Graph T, Embeddings for all entities E_e, Embeddings for all relations E_r
source_entities = Empty Set;
foreach *triplet in T* **do**
    Add triplet.head to source_entities;
end
while *True* **do**
    current_walk_length = 0;
    source_entity = Choose a random entity from source_entities;
    relation = None;
    while *current_walk_length* $<$ *L* **do**
        candidate_triplets = Empty List;
        foreach *triplet in T* **do**
            if *triplet.head == source_entity* **then**
                Add triplet to candidate_triplets;
            end
        end
        if *length of candidate_triplets == 0* **then**
            break;
        end
        random_triplet = Choose random element from candidate_triplets;
        if *current_walk_length == 0* **then**
            relation = random_triplet.relation;
        end
        if *current_walk_length == L-1* **then**
            target_entity = random_triplet.tail;
        end
        source_entity = random_triplet.tail;
        current_walk_length += 1;
    end
    if *current_walk_length == L* **then**
        break;
    end
end
source_entity_vector, target_entity_vector = Lookup source_entity and target_entity vectors from E_e;
relation_vector = Lookup relation vector from E_r;
next_hop_entity_vector = SDE-KG(source_entity_vector, relation_vector, target_entity_vector);
jump = $\sqrt{\Sigma_{i=0}^{d} target\_entity\_vector_i - source\_entity\_vector_i} - \sqrt{\Sigma_{i=0}^{d} target\_entity\_vector_i - next\_hop\_entity\_vector_i}$;
distance_to_nearest_entity = $\min(\sqrt{\Sigma_{i=0}^{d} vector_i - next\_hop\_entity\_vector_i}$, iterator: *vector* in E_e);
Output: jump, distance_to_nearest_entity

assign a rank to the prediction. This rank can be used to observe the performance of SDE-KG in directing the agent towards the target entity via the required candidate entity.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). pp. 265–283 (2016)
2. Das, R., Dhuliawala, S., Zaheer, M., Vilnis, L., Durugkar, I., Krishnamurthy, A., Smola, A., McCallum, A.: Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. arXiv preprint arXiv:1711.05851 (2017)
3. Han, X., Cao, S., Lv, X., Lin, Y., Liu, Z., Sun, M., Li, J.: Openke: An open toolkit for knowledge embedding. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 139–144 (2018)
4. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
5. Rocktäschel, T., Riedel, S.: End-to-end differentiable proving. In: Advances in Neural Information Processing Systems. pp. 3788–3800 (2017)
6. Xiong, W., Hoang, T., Wang, W.Y.: Deeppath: A reinforcement learning method for knowledge graph reasoning. arXiv preprint arXiv:1707.06690 (2017)