# DistTransH: A Scalable Knowledge Graph Embedding by Translating on Hyperplanes

*Lab Report: Distributed Big Data Analytics, WS2017-2018*

*Master of Science, Universität Bonn, Germany*

Supervised by:

Dr.Hajira Jabeen, Gezim Sejdiu

Xhulja Shahini:
S6xhshah@uni-bonn.de

Ardit Meti
S6armeti@uni-bonn.de

Abstract

A knowledge graph is a multi-relational graph composed of entities as nodes and relations as different types of edges. An instance of edge is a triplet of fact (head, relation, tail) (denoted as (h, r, t)). One of the biggest problems related to knowledge graphs is that data can scale up to very large amounts, but yet they are incomplete, which leads to performance degradation of AI applications.

A solution proposed by Antoine Bordes.et al [2], is to embed entities and relationships of multi-relational data in low-dimensional vector spaces, so as to simplify the manipulation while preserving the inherent structure of the KG. This model, called TransE is easy to train and also contains a reduced number of parameters which means that it can scale up to very large databases. TransE is e very simple model, and yet it has proven to be very powerful, but when it comes to dealing with reflexive, one-to-many, many-to-one, and many-to-many relationships, it does not perform well. A new model, called TransH has been proposed by Zhen Wang et al. [1]. This model overcomes the problem of TransE by modelling the relations as a hyperplane, together with a translation operation on it by maintaining almost the same model complexity of TransE.

## 1. Problem Definition

### 1.1 Introduction

Knowledge graphs are often incomplete, thus one of the most challenging tasks related to knowledge graphs is graph completion using link prediction. There have been many solutions proposed for doing this. A model which has proven to be very powerful is embedding the entities and relations in a vector space. This model proposed by Bordes et al. [2] is called TransE (Translating Embeddings for Modeling Multi-relational Data). Although the performance of this model is very good for one to one relationship graphs, it performs poorly when dealing with one-to-many, many-to-one, and many-to-many relationships.

In this lab we have implemented a method named translation on hyperplanes (TransH) using Scala. TransH is an extension of TransE model that overcomes the issue of one-to-many, many-to-one, and many-to-many relationships. TransH interprets a relation as a translating operation on a hyperplane.

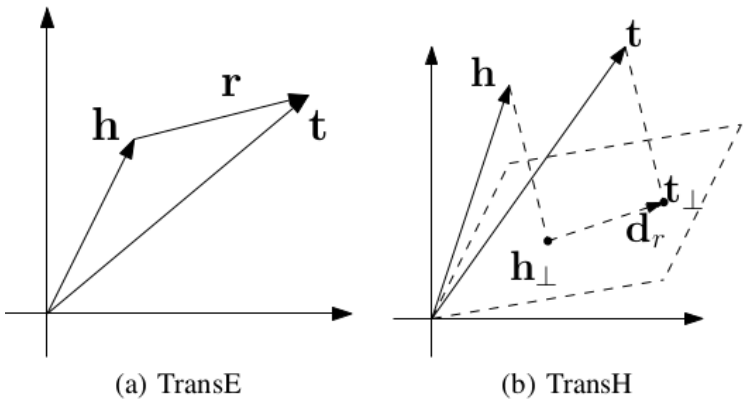


Figure 1. TransE and TransH

### 1.2 TransH Algorithm

In this model every relation is represented by a norm vector $w_r$ and a translation vector $d_r$ on the Hyperplane. For every existing triplet (h, r, t), the head h and the tail t are projected into the hyperplane $w_r$ and their projections h' and t' are expected to be connected by the translation vector $d_r$ with low error. The score function F is expected to be high for incorrect triplets and vice versa.

$$f_r(\mathbf{h}, \mathbf{t}) = \|(\mathbf{h} - \mathbf{w}_r^\top \mathbf{h}\mathbf{w}_r) + \mathbf{d}_r - (\mathbf{t} - \mathbf{w}_r^\top \mathbf{t}\mathbf{w}_r)\|_2^2.$$

Figure 2. Score function

This model is used for Link prediction, triplet classification and relational facts extraction.

### 1.3 Constructing negative labels

An additional feature of TransH is the way that negative triplets are created. In TransE negative triplets are created by replacing the head/tail of a correct triplet with a randomly selected entity. TransH instead assumes that relations are not 1-to-1 so it proposes a new simple trick to reduce the chance of false negative labeling.

Giving a higher chance to replacing the head entity if the relation is one-to-many or the tail entity if the relation is many-to-one, will reduce the probability of creating false negative triplets.

### 1.4 Evaluation

To perform evaluation TransH model replaces the heads (then repeat for tails) of correct triplets with every other entity in E in the knowledge graph and then compute the similarity using score function Figure 2. All the scores are ranked in ascending order and the rank of the correct answer is retrieved to perform later the Mean rank score and the proportion of ranks not larger than 10 score (denoted as Hits@10).

## 2. Approach

To implement the solution, we have used Scala programming language with Spark framework.

The overview of our solution of TransH model is as follows:

We read the input data specifically the FB15k dataset also used in the paper. The training data is read using the Triples() class of SANSA. We start with a randomly initialized tensor that will contain the normal vectors w for each relation then we normalize the tensor. In the same time, we initialize the embeddings for the entities and the relations. After completing the initialization part, we start the training procedure which will train the weights in such a way that for each correct triplet (h, r, t) the projections h' and t' will be connected by the translation vector $d_r$ with low error and the incorrect triplets, generated by replacing the head/tail with a randomly selected entity from entities E in the knowledge graph, the distance of the projections h' and t' should be far from the translation vector dr. We calculate the loss

function Figure 3. and if the loss is greater than zero, we calculate the gradient of the error and we update the parameters to optimize the result. In our model we have used stochastic gradient descent. While doing this we have to keep in mind the constraints predefined in the paper Figure 4. We repeat the training procedure for 1000 epochs and then we write the result in three different files, respectively e.txt, r.txt and wR.txt.

$$\mathcal{L} = \sum_{(h,r,t)\in\Delta} \sum_{(h',r',t')\in\Delta'_{(h,r,t)}} [f_r(\mathbf{h},\mathbf{t}) + \gamma - f_{r'}(\mathbf{h}',\mathbf{t}')]_+$$

Figure 3. Loss function

$$\forall e \in E, \|\mathbf{e}\|_2 \leq 1, \text{//scale} \quad (1)$$
$$\forall r \in R, |\mathbf{w}_r^\top \mathbf{d}_r| / \|\mathbf{d}_r\|_2 \leq \epsilon, \text{//orthogonal} \quad (2)$$
$$\forall r \in R, \|\mathbf{w}_r\|_2 = 1, \text{//unit normal vector} \quad (3)$$

Figure 4. Constrains

**3. Implementation**

The dataset used for this model is FB15k. It contains the following files:

i. Entity2Id.txt - containing entities and their respective IDs
ii. Relation2id.txt – containing relations and their respective IDs
iii. Train.txt - containing circa 480000 triplets (entity, entity, relation)
iv. Test.txt - containing circa 59000 triplets (entity, entity, relation)

Parameters that we have used are same as those proposed by the paper [2]

i. Margin: 1
ii. Batch size: 1200
iii. Learning rate: 0.005
iv. Length of embedding vectors (k): 20.

To implement our solution, we have created the following files:

i. TransHMain.scala
ii. TransH.scala
iii. EvaluateTransH.scala

In TransHMain we read the training data into a Dataset[IntegerTriples] and we instantiate the TransH model using the predefined algorithms.

TransH class extends the Models.scala from Sansa-ML-Spark. We used the Tensor datastructure from BigDl library to store the embeddings of entities, relations and normal vectors $w_r$. TranH class contains the following methods:

i. Run() - Here is where the distribution of the algorithm happens and we perform the online stochastic gradient descent to perform optimization.
ii. NormalizeTensor() - normalizes every row of the tensor that takes as input. We use it for the relations tensor and the normal vectors w
iii. ForceOrthogonality() - this functions tries to force orthogonality between the embeddings of the entity/relation vectors with the normal vectors w.
iv. Dist() - implements the score function, as given in the paper Formula 2
v. OptimizeTuple() - this function optimizes the parameters/embeddings for a triplet and re-apply constraints

EvaluateTransH class is used for evaluation purposes. We have followed the same evaluation technique as TransE, performing mean rank and Hits@10 for each testing triple. The Evaluation is also implemented in a distributed way using spark's mapPartition function.

### 3.1 SANSA integration

We have used SANSA functions for reading our input data. Further, we have extended the Models.scala class of SANSA-ML-Spark, same class that has been used also for TransE model implementation. In this class the embeddings of the tensors for the entities and relations are initialized. It also contains side functions like the function for corrupting triplets and the function for sampling data into subsets.

## 4. Evaluation

We follow the same evaluation protocol as the paper. We calculate mean Ranks and hit@10. For each testing triple we corrupt the head with all other possible entities and calculate the dissimilarity scores for all those corrupted triples and rank them in ascending order. The rank for the test tuple would be the index that the correct test triple has in that sorted list. We then calculate the average rank of all testing triples and the proportion of ranks not larger than 10 (hit@10). We do the same while corrupting the tail of testing tuples instead of the head.

Since we didn't have a way to filter the corrupted triples (to exclude them if we ended up creating an existing triple in the training data) we report only the "raw" setting.

We report the mean ranks and hit@10 for 100 and 500 triples as testing data:

| FB15k | | |
|---|---|---|
| | **MEAN RANK** | **HIT@10** |
| **TransH (unif.)** | 211 | 42.5 |
| **TransH (bern.)** | 212 | 47.5 |
| **Our implementation (100 triples)** | 5211 | 0 |
| **Our implementation (500 triples)** | 5385 | 0.3 |

Table 1. Link prediction results

This evaluation was done by using the trained embeddings with a learning rate equal to 0.001. We have tried to train the model with different learning rates: 0.01, 0.001, 0.005 and we report the training errors for the first 70 epochs and the time in seconds that it took to finish an epoch (other parameters were the same in all the training sessions: batch size = 1200, margin = 0.25, k =20).

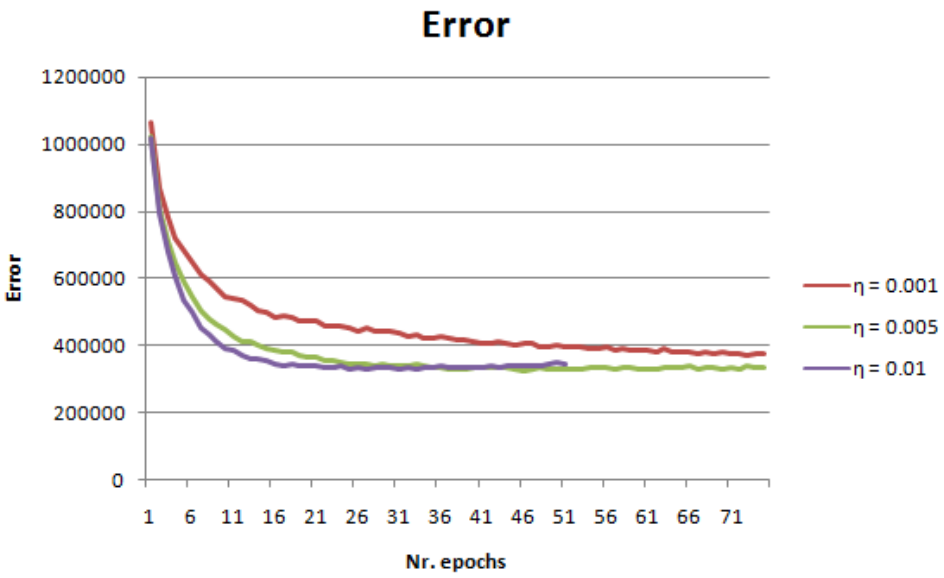


Figure 5. Training error per epoch during learning

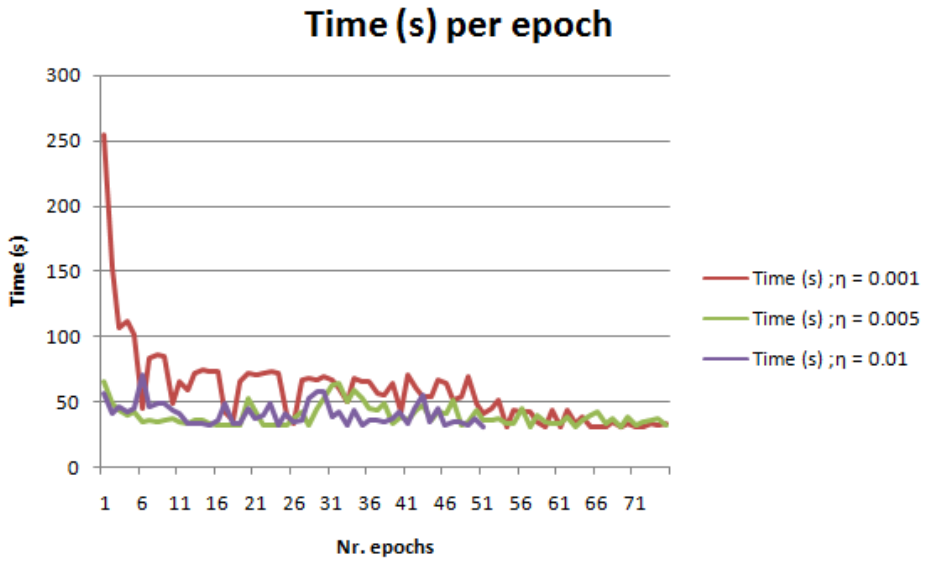


Figure 6. Time in seconds per epoch

**5. Future Work and Improvements**

TransH can generate corrupted triplets in two ways: raw and filtered. Raw means that the triplets are corrupted by replacing the head/tail of a triplet with a randomly chosen entity from the set of entities E in the knowledge graph, and it assumes that this triple is incorrect. Filtered works almost same, but after corrupting the triplet it checks if the triplet exists in the knowledge graph, so as to reduce the probability of a false incorrect triplets. Since we have implemented the solution in a distributed way, we have split the data into partitions and checking for the existence of a triplet is a challenge that we plan to solve in the future.

We also want to improve the performance of our model by implementing stochastic gradient descent with minibatches.

**6. Project Timeline**

Week1:

- Read the paper and understood the problem
- Divided the tasks

Week 2:

- Reviewed Scala/ Spark libraries
- Discussions regarding the approach of the solution

- Analyzed the algorithm and discussed how it would be implemented

Week 3:

- Researched previously done work on TransE
- Started implementing the algorithm using Scala

Week 4 - 6:

- Dealing with implementation
- Debugging and solving problems

Week 7:

- Optimizing the algorithm
- Started working on the report.

Week 8:

- Final work on the code, adding comments and making it more readable
- Finished the report

**7. Setting up the environment**

In order to run the program, you need Spark and Scala installed on your machine, or you can simply install Scala IDE for Eclipse.

- *Step 1:*

We have used the Maven template (forked from SANSA Template Maven spark https://github.com/SANSA-Stack/SANSA-Template-Maven-Spark.git). To run the project, you have to also download it's dependencies by running MVN clean package.

- *Step 2:*

Open Scala IDE and import this project as maven project. Go to:

- File > import
- Select Maven > Existing Maven Projects > Press Next

Browse to the project directory where you cloned/downloaded the project which contains the .pom file and press Finish

- *Step 3:*

Configure Scala Compiler to use Scala 2.11 bundle (dynamic)

- Go to project's properties by right-click on project > properties
- At Scala Compiler check `Use Project Settings` and select latest 2.11 bundle (dynamic)

- Press `Apply and close` and run the main Class > src/main/scala/TransHmain.scala
- After the parameters/embeddings are trained run Class > src/main/scala/EvaluateTransH.scala

## REFERENCES


[1]Zhen Wang , Jianwen Zhang , Jianlin Feng , Zheng Chen - *Knowledge Graph Embedding by Translating on Hyperplanes*

[2]Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán - Translating Embeddings for Modeling Multi-relational Data

[3]TransH model implementation in C++
https://github.com/thunlp/KB2E/blob/master/TransH/Train_TransH.cpp

[4]SANSA-ML
https://github.com/SANSA-Stack/SANSA-ML/tree/develop/sansa-ml-spark/src/main/scala/net/sansa_stack/ml/spark/kge/linkprediction

[5]BigDl library
https://bigdl-project.github.io/0.7.0/