MEKER: Memory Efficient Knowledge Embedding Representation for Link Prediction and Question Answering

Viktoriia Chekalina[1], **Anton Razzhigaev**[1,2], **Albert Sayapin**[1], **Evgeny Frolov**[1], **and Alexander Panchenko**[1]
[1]Skolkovo Institute of Science and Technology, [2]Artificial Intelligence Research Insitute (AIRI)

Abstract

Knowledge Graphs (KGs) are symbolically structured storages of facts. The KG embedding contains concise data used in NLP tasks requiring implicit information about the real world. Furthermore, the size of KGs that may be useful in actual NLP assignments is enormous, and creating embedding over it has memory cost issues. We represent KG as a 3rd-order binary tensor and move beyond the standard CP decomposition (Hitchcock, 1927) by using a data-specific generalized version of it (Hong et al., 2020). The generalization of the standard CP-ALS algorithm allows obtaining optimization gradients without a backpropagation mechanism. It reduces the memory needed in training while providing computational benefits. We propose a MEKER, a memory-efficient KG embedding model, which yields SOTA-comparable performance on link prediction tasks and KG-based Question Answering.

1 Introduction

Natural Language Processing (NLP) models have taken a big step forward over the past few years. For instance, language models can generate fluent human-like text without any problems. However, some applications like question answering and recommendation systems need correct, precise, and trustworthy answers.

For this goal, it is appropriate to leverage knowledge graphs (KG) (Bollacker et al., 2008; Rebele et al., 2016) a structured repository of essential facts about the real world. For convenience, the knowledge graph can be represented as a set of triples. A triple is two entities bound with relation and describes the fact. It takes the forms of $\langle e_s, r, e_o \rangle$, where e_s and e_o represent objects and subject entities, respectively.

For efficient use of information from KG, there is a need for the low-dimensional embedding of graph entities and relations. KG embedding models usually use a standard Neural Networks (NN) backward mechanism for parameter tuning, duplicating its memory consumption. Hence, existing approaches to embedding learning have substantial memory requirements and can be deployed only on small datasets under a single GPU card. Processing large KGs appropriate for the custom downstream task is a challenge.

There are several libraries designed to solve this problem. Framework LibKGE (Ruffinelli et al., 2020) allows the processing of large datasets by using sparse embedding layers. Despite the memory saving, sparse embedding has several limitations - for example, in the PyTorch library, they are not compatible with several optimizers. PyTorch-BigGraph (Lerer et al., 2019) operates with large knowledge graphs by dividing them into partitions - distributed subgraphs. Subgraphs need a place for storing, embedding models need modifications to work with partitions and perform poorly.

The main contribution of our paper is a memory-efficient approach to learning Knowledge Graph embeddings MEKER (Memory Efficient Knowledge Embedding Representation). It allows more efficient KG embedding learning, maintaining comparable performance to state-of-the-art models. MEKER leverages generalized canonical Polyadic (CP) decomposition (Hong et al., 2020), which allows a better approximation of given data and analytical computation of the parameters' gradient. MEKER is evaluated on a link prediction task using several standard datasets and large datasets based on Wikidata. Experiments show that MEKER achieves highly competitive results on these two tasks. To demonstrate downstream usability, we create a Knowledge Base Question Answering system Text2Graph and use embeddings in it. The system with MEKER embeddings performs better as compared to other KG embeddings, such as PTBG (Lerer et al., 2019).

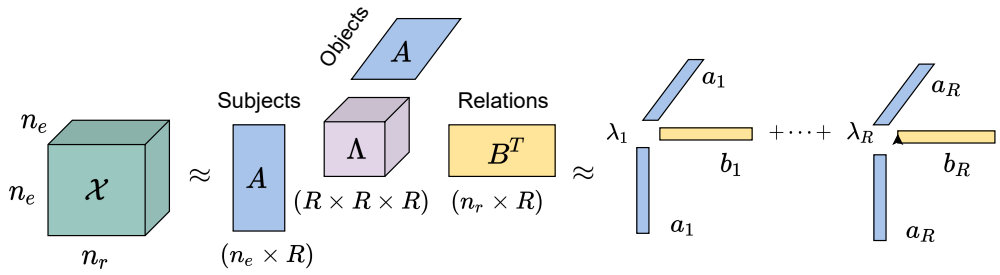

Figure 1: The CP decomposition scheme in the case of entity and relation KG embedding in MEKER. This is a binary 3-dimensional tensor $\mathcal{X}$ of knowledge graph facts that introduces objects, relations, and subjects indexes along the three axes. B contains relation embedding, while A represents entity vectors for the subject and object simultaneously. Λ is the diagonal core tensor, identity in our case.

2 Related Work

There are three types of approaches for learning KG embedding: distance-based, tensor-based, and deep learning-based models. The first group is based on the assumption of translation invariance in the embedding vector space. In model **TransE** (Bordes et al., 2013) relations are represented as connection vectors between entity representations. **TransH** (Wang et al., 2014) implies relation as a hyperplane onto which entities are being projected. **QuatE** (Zhang et al., 2019) extends the idea with hypercomplex space and represents entities as embeddings with four imaginary components and relations as rotations in the space.

Tensor-based models usually represent triples as a binary tensor and look for embedding matrices as factorization products. **RESCAL** (Nickel et al., 2011) employs tensor factorization in the manner of DEDICOM (Harshman et al., 1982), which decomposes each tensor slice along the relationship axis. **DistMult** (Yang et al., 2015) adapts this approach by restricting the relation embedding matrix to diagonal. On the one hand, it reduces the number of relation parameters, on the other hand, it losses the possibility of describing asymmetric relations. The **ComplEX** (Trouillon et al., 2016) represents the object and subject variants of a single entity as complex conjugates vectors. It combines tensor-based and translation-based approaches and solves the asymmetric problem. **TuckER** (Balazevic et al., 2019) uses Tucker decomposition (Tucker, 1966c) for finding representation of a knowledge graph elements. This work can also be considered a generalization of several previous link prediction methods.

Standard Canonical Polyadic (CP) (Hitchcock, 1927) decomposition in the link prediction task does not show outstanding performance (Trouillon et al., 2017). Several papers address this problem by improving the CP decomposition approach. **SimplIE** (Kazemi and Poole, 2018) states that low performance is due to different representations of subject and object entity and deploys CP decomposition with dependently learning of subjects and objects matrices. **CP-N3** (Lacroix et al., 2018) highlights the statement that the Frobenius norm regularizing is not fit for tensors of order more than 3 (Cheng et al., 2016) and proposes a Nuclear p-norm instead of it. Our approach also uses CP decomposition with enhancement. We consider remark from **SimplIE** and set the object and subject representations of one entity to be equals. At the same time, inside the local step of the CP decomposition algorithm, the matrices of subjects and objects consist of different elements and are different (see Appendix). In contradistinction to **CP-N3**, we do not employ a regularizer to improve training but change the objective. Instead of squared error, we use logistic loss, which is appropriate for one-hot data. We abandon the gradient calculation through the computational graph and count gradient analytically, which makes the training process less resource-demanding.

Approaches based on Deep Learning convolutions and attention mechanisms **ConvE, GAT, GAAT** (Dettmers et al., 2017; Nathani et al., 2019; Wang et al., 2020) achieve high performance in link prediction. Besides, they have their disadvantages - it necessitate more time and memory resources than other types of models and usually needs pre-training.

3 MEKER: Memory Efficient Knowledge Embedding Representation

Our approach to entity embeddings relies on generalized CP tensor decomposition (Hitchcock, 1927). Namely, R-rank CP decomposition approximates an N-dimensional tensor as a sum of R outer products of N vectors. Every product can also be viewed as a rank-1 tensor. This approximation is described by the following formula: $\mathcal{X} \approx \mathcal{M} = [|A, B, C|]$, where $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ is original data and $\mathcal{M} \in \mathbb{R}^{I \times J \times K}$ is its approximation. Factors have the following shape $A \in \mathbb{R}^{I \times R}$, $B \in \mathbb{R}^{J \times R}$, $C \in \mathbb{R}^{K \times R}$. The scheme of CP decomposition applied to the KG elements representation task is in Figure 1. We set matrix A equal to matrix C and simultaneously corresponding to subject and object entities.

3.1 Generalization of Canonical Poliyadic (CP) Decomposition

Following the determination of the approximation type, the next task is to find the parameters of the factor matrices that best match the ground truth data. Battaglino et al. (2018); Dunlavy et al. (2011) describe the most widely used CP decomposition algorithm, CP-ALS. The update rules for the factor matrices are derived by alternating between minimizing squared error (MSE) loss. Hong et al. (2020) demonstrates that MSE corresponds to Gaussian data and is a particular case of a more general solution for an exponential family of distributions. In general, the construction of optimal factors originates the minimization problem:

$$\begin{aligned} \min F(\mathcal{M}; \mathcal{X}) &\equiv \sum_{i \in \Omega} f(x_i, m_i), \\ f(x, m) &\equiv \log p(x | l^{-1}(m)), \end{aligned} \tag{1}$$

where f - elementwise loss function, Ω - set of indices of known elements of $\mathcal{X}$, l - link function, x_i and m_i - the i-th elements of $\mathcal{X}$ and $\mathcal{M}$, respectively. We also introduce $\mathcal{Y}$ - the tensor of derivatives of the elementwise loss with the same size as $\mathcal{X}$ and being filled by zeros for $i \notin \Omega$. The data in the sparse one-hot triple tensor has a Bernoulli distribution. The link function for Bernoulli is $l(\rho) = log(\rho/(1-\rho))$ and associated probability is $\rho = \exp(m)(1 - \exp(m))$ so the loss function and elements of the $\mathcal{Y}$ are defines as follows:

$$\begin{aligned} f(x_i, m_i) &= \log(1 + \exp m_i) - x_i m_i, \\ y(x_i, m_i) &= \frac{\partial f(x_i, m_i)}{\partial m_i} = \frac{\exp m_i}{1 + \exp m_i} - x_i. \end{aligned} \tag{2}$$

Hong et al. (2020) derives partial derivatives of F w.r.t. factor matrices and presents gradients G of it in a form similar to standard CP matrix update formulas:

$$\begin{aligned} G_A &= \mathcal{Y}_{[0]}(B \odot C)^{T\dagger}, \\ G_B &= \mathcal{Y}_{[1]}(A \odot C)^{T\dagger}, \\ G_C &= \mathcal{Y}_{[2]}(A \odot B)^{T\dagger}, \end{aligned} \tag{3}$$

where $\dagger$ - pseudo-inverse matrix, $\odot$ - Khartri-Rao operator, $\mathcal{X}_{[n]}$ - mode-n matricization, a reshaping of tensor $\mathcal{X}$ along the n axis. The importance of representation (3) is that we can calculate the gradients via an essential tensor operation called the matricized tensor times Khatri-Rao product (MTTKRP), implemented and optimized in most programming languages. Algorithm 1 describes the procedure for computing factor matrices gradients (3) in a Bernoulli distribution case (2).

3.2 Implementation Details

We use PyTorch (Paszke et al., 2019) to implement the MEKER model. We set the object and subject factors equal and correspond to matrix A for the decomposition of the one-hot KG triplet tensor. Sparse natural and reconstructed tensors are stored in Coordinate Format as a set of triplets (COO). We combine actual triples and sampled negative examples in batches, and process them. The corresponding pieces from the ground-truth tensor and current factor matrices are cut out for each batch. Then the pieces are sent to Algorithm 1 for the calculation of gradients of the matrix elements with appropriate indexes. Algorithm 2 describes the pseudocode of factorization KG tensor using GCP gradients.

We train the MEKER model using Bayesian search optimization to obtain the optimal training parameters. We use the Wandb.ai tool (Biewald, 2020) for experiment tracking and visualizations. The complete sets of tunable hyperparameters are in the Appendix. Table 2 shows the best combinations of it for the proposed datasets.

3.3 Baselines

As a comparison, we deploy related link prediction approaches that meet the following criteria:

1) it should learn KG embedding from scratch 2) it should report high performance 3) the corresponding paper should provide a runnable code. We use the Tucker, Hyper, ConvKB, and QuatE implementations from their respective repositories. For TransE, DistMult, ComplEx, and ConvE, we use LibKGE (Ruffinelli et al., 2020) library with the best parameter setting for reproducing every model. We run each model five times for each observed value and provide means and sample standard deviation.

Algorithm 1 GCP GRAD Bernuilli

Input: $\mathcal{X}$ ▷ Ground Truth Tensor
A, B, C ▷ Factor matrices

Output: F, G_A, G_B, G_C

$\mathcal{M} = \{A, B, C\}$ ▷ Model Restored tensor

$F = \sum_i f(x_i, m_i) = \sum log(1 + e_i^m) - x_i m_i$ ▷ Loss

$\mathcal{Y} = \sum_i \frac{\delta f(x_i, m_i)}{\delta m_i} =$ ▷ Derivative tensor
$= \sum \frac{1}{1+e^{(-m_i)}} - x_i$

$G_A = \mathcal{Y}_{[0]}(B \odot C)^{T\dagger}$. ▷ Element-wise gradient for A
$G_B = \mathcal{Y}_{[1]}(A \odot C)^{T\dagger}$ ▷ Element-wise gradient for B
$G_C = \mathcal{Y}_{[2]}(A \odot B)^{T\dagger}$ ▷ Element-wise gradient for C

Algorithm 2 Factorization of the KG tensor using GCP gradients

Input: $\mathcal{X}$ ▷ Ground Truth Tensor
Triplets ▷ List of triplets
LR ▷ learning rate
R ▷ Desired size of embeddings
N ▷ Number of epoch

Output: A, B ▷ Updated factor matrices

Initialize factor matrices $A \in \mathbb{R}^{R \times n_e}$, $B \in \mathbb{R}^{R \times n_r}$

for $i = 1 \ldots N$ **do**
  for $[\text{inds}_a, \text{inds}_b, \text{inds}_c]$ **in** Triplets **do**
    $\mathcal{X}_{batch} = \mathcal{X}[\text{inds}_a, \text{inds}_b, \text{inds}_c]$
    $g_a, g_b, g_c, loss =$
GCP_GRAD($\mathcal{X}_{batch}, A[\text{inds}_a], B[\text{inds}_b], A[\text{inds}_c]$)
    $A[\text{inds}_a].\text{grad} = g_a$
    $B[\text{inds}_b].\text{grad} = g_b$
    $A[\text{inds}_c].\text{grad} = g_c$
    UPDATE(A, B, LR)

4 Experiments on Standard Link Prediction Datasets

4.1 Experimental settings

The Link prediction task estimates the quality of KG embedding. Link prediction is a classification predicting if triple over graph elements is true or not. The scoring function $\Phi(e_s, rel, e_o)$ returns the probability of constructing a true triple. We test our model on this task using standard Link prediction datasets.

FB15k237 (Toutanova and Chen, 2015) is a dataset based on the FB15k237 adapted Freebase subset, which contains triples with the most mentioned entities. FB15k237 devised the method of selecting the most frequent relations and then filtering inversions from test and valid parts. The **WN18RR** (Bordes et al., 2013) version of WN18 is devoid of inverse relations. WN18 is a WordNet database that contains the senses of words as well as the lexical relationships between them. Table 3 shows the number of entities, relations, and train-valid-test partitions for each dataset used in the proposed work. As an evaluation, we obtain complementary candidates from the entity set for each pair entity-relation from each test triple and estimate the probability score of the received triple being true. The presence of a rising real supplement entity at the top indicates a hit. Candidate ranking is provided using a filtered setting, which was first used in (Bordes et al., 2013). In a filtered setting, all candidates who completed a true triple on the current step are removed from the set, except for the expected entity. We use Hit@1, Hit@3, Hit@10 as evaluation metrics. We also use mean reciprocal rank (MRR) to ensure that true complementary elements are ranked correctly.

4.2 Link Prediction

Table 1 shows the mean value of the experiment on small datasets for the embedding of size 200. The Hit@10 standard deviation for MEKER is 0.0034 for the FB15k237 dataset and 0.0026 for the WNRR18 dataset. Due to space constraints, the table with deviations from all experiments, comparable to Table 1, is in Appendix.

The best score belongs to QuatE (Zhang et al., 2019) model due to its highly expressive 4-dimensional representations. Among the remaining approaches, MEKER outperforms its contestants' overall metrics except for the Hit@10 - Tucker model surpasses MEKER for Fb15k237, ComplEX by LibKGE for WNRR18. In general, MEKER shows decent results comparable to strong baselines (Zhang et al., 2019; Balazevic et al., 2019). It is also worth noting that MEKER significantly improves MRR and Hit@1 metrics on freebase datasets, whereas on word sense, according to data, it has been enhanced in Hit@10.

Dataset	FB15k237				WNRR18			
Model	MRR	Hit@10	Hit@3	Hit@1	MRR	Hit@10	Hit@3	Hit@1
ConvKB (Nguyen et al., 2018)	0.2985	0.4785	0.3270	0.2296	0.2221	0.5074	0.3777	0.0347
HypER (Balazevic et al., 2018)	0.3423	0.5228	0.3774	0.2536	0.4653	0.5228	0.4774	0.4361
TuckER (Balazevic et al., 2019)	0.3455	0.5408	0.3899	0.2606	0.4654	0.5215	0.4784	0.4368
QuatE (Zhang et al., 2019)	**0.3614**	**0.5538**	**0.4014**	**0.2711**	**0.4823**	**0.5719**	**0.4955**	**0.4360**
CP-N3 (Lacroix et al., 2018)	0.3514	0.5294	0.3876	0.2646	0.4402	0.4858	0.4485	0.4207
LibKGE ConvE (Dettmers et al., 2017)	0.3367	0.5213	0.3682	0.2381	0.4282	0.5049	0.4492	0.3934
LibKGE TransE (Bordes et al., 2013)	0.3121	0.4962	0.3175	0.2195	0.2274	0.5189	0.3677	0.0516
LibKGE DistMult (Yang et al., 2015)	0.3331	0.5185	0.3673	0.2410	0.4505	0.5215	0.4634	0.4162
LibKGE ComplEx (Trouillon et al., 2016)	0.3390	0.5265	0.3724	0.2468	0.4752	0.5467	0.4809	0.4366
MEKER	0.3588	0.5393	0.3915	0.2682	0.4768	0.5447	0.4875	0.4371

Table 1: Link Prediction scores for various models on the FB15k237 and WN18RR datasets. The embedding size is 200. The winner scores are highlighted in bold font, and the second results are underlined.

Dataset	FB15k237	WN18RR
Optimizer	AdamW	AdamW
LR	0.01	0.009
Batch Size	156	128
L2 reg	0.001	0.0
Number of negative	6	8
Step of decay LR	3	15
Gamma of decay LR	0.8	0.6

Table 2: The best hyperparameters of the MEKER.

Dataset	#ents	#rels	Number of Triplets Train	Valid	Test
Fb15k237	14,541	237	$27.2{\cdot}10^4$	17,535	20,466
WN18RR	40,943	11	$8.6{\cdot}10^4$	30,034	3,134
Wiki4M	$4{,}316{\cdot}10^4$	1,245	$1{,}367{\cdot}10^4$	30,000	35,815
Wikidata5m	$4{,}594{\cdot}10^4$	822	$2{,}061{\cdot}10^4$	5,163	5,133

Table 3: Statistics of link prediction datasets.

4.3 Model efficiency in case of parameter size increasing

With a strong memory assumption, we can reduce the size of pre-trained MEKER embeddings by tenfold while losing only a few percent of performance.

Figures 2, 3 show MRR and Hit@1 scores for MEKER, TuckER, and ComplEX models at various embedding sizes. Each model approaches a constant value on both metrics around rank 100. For ranks 200 and 300, the performance difference between the three models is approximately consistent for both metrics, with MEKER scoring the highest on rank 20. It means that the number of MEKER parameters can be reduced while maintaining or improving quality. The quality loss is significant for other presented models.

4.4 Memory Complexity Analysis

The theoretical space complexity of models mentioned in the current work is shown in the right column of Table 4. In the context of the Link Prediction task, all approaches have asymptotic memory complexity $\mathcal{O}((n_e + n_r)d)$, which is proportional to the size of the full dictionary of KG elements, i.e. the embedding layer or look-up table. Other aspects of the proposed models are less significant: the convolutional layers are not very extensive. The implementation determines the amount of real memory used by the model during the training process. The Neural Network backpropagation mechanism is used to tune parameters in the most related work. Backpropagation in Figure 4 creates computational graph in which all model parameters are duplicated. It results in a multiplicative constant 2, insignificant in a small dictionary but becomes critical in a large one. To summarize, the following factors account for the decrease in MEKER's required memory:

1. In the MEKER algorithm gradients are computed analytically.

2. MEKER does not have additional neural network layers (linear, convolutional, or attention).

To measure GPU RAM usage, we run each considered embedding model on FB15k-237 into a single GPU and print peak GPU memory usage within the created process. The left column of a Table 4 demonstrates that MEKER has objective memory complexity that is at least twice lower than that of other linear approaches. This property reveals the possibility of obtaining representations of specific large databases using a single GPU card.

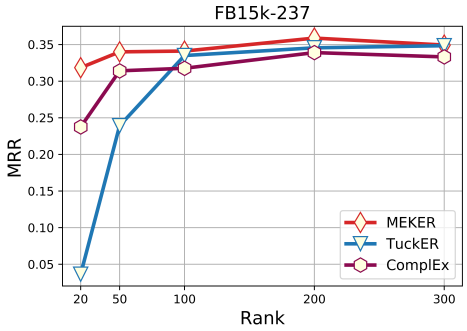

Figure 2: MRR score in dependence of embedding ranks

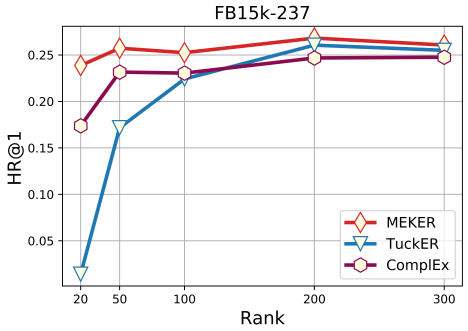

Figure 3: Hit@1 score in dependence of embedding ranks

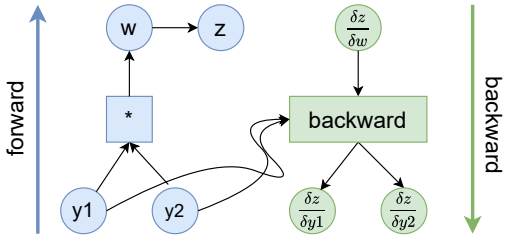

Figure 4: The scheme of the augmented computational graph of the Neural Network.

Model	GPU Memory Usage, MB	Theoretical Approximation of Space Complexity
TuckER	357	$2 \cdot ((n_e + n_r + c \cdot lin) \cdot d)$
HypER	208	$2 \cdot ((n_e + n_r + c \cdot lin) \cdot d)$
ConvKB	3 563	$2 \cdot ((n_e + n_r) \cdot d + c \cdot conv)$
ConvE	229	$2 \cdot ((n_e + n_r) \cdot d + c \cdot conv)$
ComplEX	252	$2 \cdot (n_e + n_r) \cdot d$
DistMult	174	$2 \cdot (n_e + n_r) \cdot d$
QuatE	2 367	$2 \cdot 4 \cdot (n_e + n_d + c \cdot lin)$
CP (N3)	138	$2 \cdot (n_e + n_r) \cdot d$
MEKER	79	$((n_e + n_r) \cdot d)$

Table 4: Memory, reserved in the PyTorch Framework during the training process and theoretical approximation of given implementations' complexity. On the FB15k237 dataset, we train 200-size representations with a batch size of 128. Lin denotes the number of output features in a linear layer, $conv$ denotes the size of convolutional layer parameters. The constant c represents the number of different layers.

5 Experiments on Large-Scale KG Datasets

5.1 Experimental settings

To test the model on large KG, we employ two WikiData-based datasets. The first English dataset, **Wikidata5m** (Wang et al., 2021)[1], is selected due to the presence of related works and reproducible baseline (Ruffinelli et al., 2020). This dataset is created over the 2019 dump of WikiData and contains of elements with links to informative Wikipedia pages. Our experiments use the transductive setting of Wikidata5m - triplet sets to disjoint across training, validation, and test.

The second English-Russian dataset is formed since its suitability for the NLP downstream task. We leverage KG-based fact retrieval over Russian Knowledge Base Questions (RuBQ) (Rybin et al., 2021) benchmark. This benchmark is a subset of Wikidata entities with Russian labels. Some elements in RuBQ are not covered with Wikidata5m, so we created a link-prediction **Wiki4M** dataset over RuBQ. We select triples without literal objects and obtain approximately 13M triples across 4M entities (see Table 3). Wiki4M also fits the concept of multilingualism is intended to be used in a cross-lingual transfer or few-shot learning.

5.2 Link Prediction

We embed the datasets for ten epochs on a 24.268 Gb GPU card with the following model settings: LR $2.5 \cdot 10^{-4}$, increasing in 0.5 steps every 10 epoch, batch size 256, number of negative samples 4 for Wiki4M and 2 for Wikidata5m.

As a comparison, we use the PyTorch-BigGraph large-scale embedding system (Lerer et al., 2019). PyTorch-BigGraph modifies several traditional embedding systems to focus on the effective representation of KG in memory. We select ComplEX and TransE and train graphs for these embedding models, dividing large datasets into four partitions. With a batch size of 256, the training process takes 50 epochs.

We also deploy LibKGE (Ruffinelli et al., 2020) to evaluate TransE and ComplEX approaches. For

[1] https://deepgraphlearning.github.io/project/wikidata5m

Model	MRR	Hit@1	Hit@3	Hit@10	Memory, GB	Storage, GB
English: Wikidata5m dataset						
PTBG (ComplEX)	0.184	0.131	0.210	0.287	45.15	9.25
PTBG (TransE)	0.150	0.091	0.176	0.263	43.64	9.25
LibKGE sparse (TransE)	0.142	0.153	0.211	0.252	33.29	0.00
LibKGE sparse (ComplEX)	0.202	**0.160**	0.233	0.316	**21.42**	0.00
MEKER (ours)	**0.211**	0.149	**0.238**	**0.325**	22.27	0.00
Russian: Wiki4M dataset						
PTBG (ComplEX)	0.194	0.141	0.212	0.293	42.83	9.25
LibKGE sparse (TransE)	0.183	0.126	0.191	0.275	26.75	0.00
LibKGE sparse (ComplEX)	0.247	0.196	0.275	0.345	**20.22**	0.00
MEKER (ours)	**0.269**	**0.199**	**0.303**	**0.410**	21.04	0.00

Table 5: Unfiltered link prediction scores for MEKER and PyTorch-BigGraph approaches for Wiki4M and Wikidata5m datasets and memory needed in leveraging every model. Storage means additional memory demanded for auxiliary structures. Batch size 256. Here "RAM" is GPU RAM or main memory RAM if GPU limit of 24 GB is reached. *Sparse* means sparse embeddings. Models without *sparse* mark employ dense embeddings matrix.

ComplEX model training, we use the best parameter configuration from the repository, for TransE, we obtain a set of training parameters by greed search. The learning rate for TransE is 0.5, decaying in factor 0.45 every 5 step and train model in 100 epochs. In both cases, we use sparse embedding in the corresponding model setting and batch size of 256. Models from both wrappers that did not fit in 24 GB, we train on the CPU.

Embedding sets yielded by we these experiments we then test on the link prediction task. We provide scoring without filters because the partition-based setup of PyTorch-Biggraph does not support filtering evaluation. Tables 5 shows that MEKER significantly improves the results of PyTorch-Biggraph models across all proposed metrics. The ComplEX model with sparse embedding, fine-tuned by LibKGE, gives results almost approaching the MEKER and exceeding the Hit@1 in Wiki4M. The right part of Tables 5 shows that the baseline approaches consume twice as much memory as MEKER, but sparse ComplEX slightly improves memory consumption. TransE does not give such significant results as ComplEX.

5.3 Knowledge Base Question Answering (KBQA)

In this section, to further evaluate the proposed MEKER embeddings we test them in an extrinsic way within on a KBQA task on two datasets for English and Russian.

5.3.1 Experimental Setting

We perform experiment with two datasets: for English we use the common dataset SimpleQuestions (Bordes et al., 2015) aligned with Wiki4M KG[2] (cf. Table 3), and for Russian we use RuBQ 2.0 dataset (Rybin et al., 2021) which comes with the mentioned above Wiki4M KG (cf. Table 3). RuBQ 2.0 is a Russian language QA benchmark with multiple types of questions aligned with Wikidata. For both SimpleQuestions and RuBQ, for each question, an answer is represented by a KG triple.

For training we use a training set of SimpleQuestions for verification we use a test set of SimpleQuestions and RuBQ 2.0 dataset for English and Russian, respectively. These Q&A pairs provide ground truth answers linked to exact this version of KG elements.

More specifically, in these experiments, we test answers to 1-hop questions which are questions corresponding to one subject and one relation in the knowledge graph, and takes their object as an answer.

We want to leverage the KBQA model, which can process questions both in English and Russian. To measure the performance of a KBQA system, we measure the accuracy of the retrieved answer/entity. This metric was used in previously reported results on SimpleQuestions and RuBQ. If the subject of the answer triple matches the reference by ID or name, it is considered correct.

5.3.2 KBQA methods

The key idea of the KBQA approaches is mapping questions in natural language to the low-dimensional space and comparing them to graph elements' given representation. In **KEQA** (Huang

[2] https://github.com/askplatypus/wikidata-simplequestions

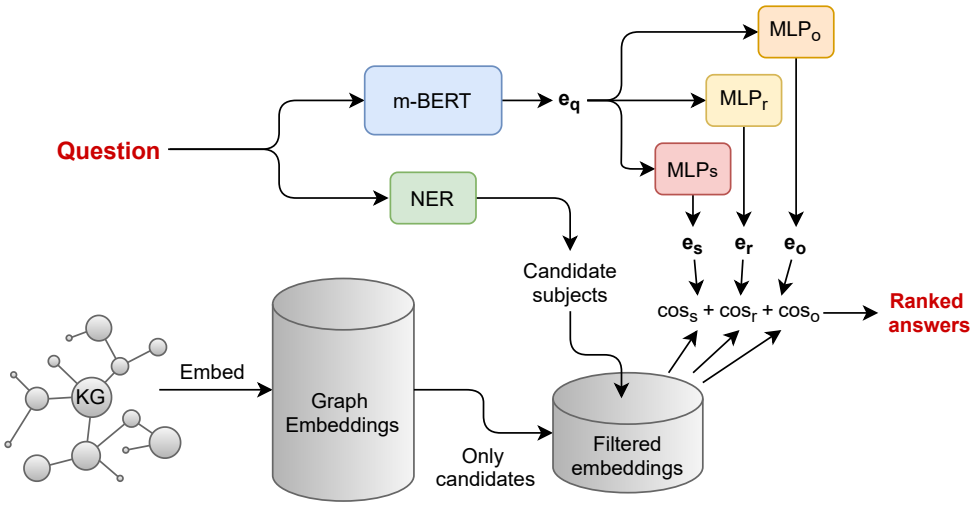

Figure 5: Text2Graph method used in our experiments: 1-Hop QA pipeline. First, we take original entity and relation embeddings. The question is embedded using m-BERT. This embedding is then processed by MLP, yielding candidate representations of an object, relation, and subject. The sum of the subject, relation, and object cosines is the final score of triple candidates.

et al., 2019) LSTM models detect the entity and predicates from the question text and project it further into the entity and predicate embedding spaces. The closest subject in terms of similarity to the entity and predicate embeddings is selected as the answer.

We created a simple approach **Text2Graph** which stems from the **KEQA** and differs from the original work in improved question encoder, entity extractor, additional subject embedding space and simplified retrieval pipeline. The Algorithm 3 describes the procedure of projecting the input question to graph elements. The multilingual-BERT (Devlin et al., 2019) model encodes the input question, and all word vectors are averaged into a single deep contextualized representation e_q. This representation then goes through three MLPs jointly learning candidate embeddings of an object, relation, and subject. We minimize MSE between predicted embeddings and the corresponding KGE model's embeddings. The appropriateness score of every fact in KG is a sum of cosine similarity between MLP outputs and ground truth model representation for every element in the triple. The triple with the highest score is considered to be an answer. The scheme is trained using an AdamW optimizer with default parameters for 10 epochs.

5.4 Baselines

5.4.1 RuBQ 2.0

We compare our method to several QA approaches compatible with questions from this benchmark.

Algorithm 3 Text2Graph question projection algorithm

Input: $\mathcal{Q}$, $\mathcal{G}$, **E**,
text encoder M_{enc},
projection modules: M_s, M_r, M_o,
Subject Candidates Extractor: NER
Output: answer $\langle o_a, r_a, s_a \rangle$

$e_q = M_{enc}(\mathcal{Q})$
Initialize answers-candidates list with empty list **A**=[]
Initialize scores list with empty list **S**=[]
Initialize entities-candidates list with empty list **C**=[]
for entity **in** $\mathcal{G}$ **do**
  if entity.name **in** NER($\mathcal{Q}$) **then**
    C.append(entity)
for entity **in C do**
  for relation **in** entity.relations **do**
    s = entity.id
    r = relation.id
    o = entity[r]
    triple = $\langle s, r, o \rangle$
    A.append(triple)
    $\mathbf{e_s}$ = **E**[s]
    $\mathbf{e_r}$ = **E**[r]
    $\mathbf{e_o}$ = **E**[o]
    $\mathbf{y_s} = M_s(e_q)$
    $\mathbf{y_r} = M_r(e_q)$
    $\mathbf{y_o} = M_o(e_q)$
    $score = \cos(\mathbf{e_o}, \mathbf{y_o}) + \cos(\mathbf{e_r}, \mathbf{y_r}) + \cos(\mathbf{e_s}, \mathbf{y_s})$
    S.append($score$)
ind = argmax(**S**)
$\langle s_a, r_a, o_a \rangle$ = **A**[ind]
return $\langle s_a, r_a, o_a \rangle$

KBQA Model	Embedding Model	Accuracy 1-Hop
DeepPavlov	-	30.5 ± 0.04
SimBa	-	32.3 ± 0.05
QA-En	-	32.3 ± 0.08
QA-Ru	-	30.8 ± 0.03
Text2Graph	PTBG (ComplEX) Wiki4M	48.16 ± 0.05
Text2Graph	PTBG (TransE) Wiki4M	48.84 ± 0.06
Text2Graph	MEKER Wiki4M	49.06 ± 0.06

Table 6: Comparison of the Text2Graph system with the various KG embeddings with existing solutions (QA-Ru, QA-En, SimBa) on RuBQ 2.0 benchmark.

KBQA Model	Embedding Model	Accuracy 1-Hop
KEQA	TransE FB5M	40.48 ± 0.10
Text2Graph	PTBG (TransE) Wikidata5m	59.97 ± 0.15
Text2Graph	MEKER Wikidata5m	61.81 ± 0.13

Table 7: Comparison of the Text2Graph system with the various KG embeddings with existing embedding-based solution on the SimpleQuestions benchmark.

QAnswer[3] is a rule-based system addressing questions in several languages, including Russian. **SimBa** is a baseline presented by RuBQ 2.0 authors. It is a SPARQL query generator based on an entity linker and a rule-based relation extractor. KBQA module of **DeepPavlov Dialogue System Library** (Burtsev et al., 2018) also based on query processing.

5.4.2 SimpleQuestions

Simple Question is an English language benchmark aligned with FB5M KG - the subset of Freebase KG. Its train and validation parts consist of 100k and 20k questions, respectively. As a baseline solution we employ **KEQA** (Huang et al., 2019). We realign answers from this benchmark to our system, which is compatible with Wikidata5m. Not all of the questions from FB5M have answers among Wiki4M, that is why we test both systems on a subset of questions whose answers are present in both knowledge graphs.

5.4.3 Experimental Results

We compare the results of the Text2Graph with PTBG embeddings versus MEKER embedding and baseline KBQA models. Results on the RuBQ 2.0 dataset are shown in Table 6. Text2Graph outperforms baselines. Using MEKER embeddings instead of the PTBG version of ComplEX and TransE demonstrates slightly better accuracy.

Table 7 presents results on the SimpleQuestions dataset. As Huang et al. (2019) model uses FB5M KG and Text2Graph uses Wikidata5m KG we test both models on the subset of questions, which answers are present in both knowledge graphs for a fair comparison. Our model demonstrates superior performance and regarding the comparison within different embeddings in a fixed system, MEKER provides better accuracy of answers than TransE embeddings on the SimpleQuestions benchmark.

6 Conclusion

We propose MEKER, a linear knowledge embedding model based on generalized CP decomposition. This method allows for the calculation of gradient analytically, simplifying the training process under memory restriction. In comparison to previous KG embedding linear models (Balazevic et al., 2019), our approach achieves high efficiency while using less memory during training. On the standard link prediction datasets WN18RR and FB15k-237, MEKER shows quite competitive results.

In addition, we created a Text2Graph — KBQA system based on the learned KB embeddings to demonstrate the model's effectiveness in NLP tasks. We obtained the required representations using MEKER on the Wikipedia-based dataset Wiki4M for questions in Russian and on Wikidata5m for questions in English. Text2Graph outperforms baselines for English and Russian, while using MEKER's embeddings provides additional performance gain compared to PTBG embeddings. Furthermore, our model's link prediction scores on Wiki4M and Wikidata5m outperform the baseline results. MEKER can be helpful in question-answering systems over specific KG, in other words, in systems that need to embed large sets of facts with acceptable quality.

All codes to reproduce our experiments are available online.[4]

Acknowledgements

The work was supported by the Analytical center under the RF Government (subsidy agreement 000000D730321P5Q0002, Grant No. 70-2021-00145 02.11.2021).

[3] https://www.qanswer.eu

[4] https://github.com/skoltech-nlp/meker

References

Ivana Balazevic, Carl Allen, and Timothy Hospedales. 2019. TuckER: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5185–5194, Hong Kong, China. Association for Computational Linguistics.

Ivana Balazevic, Carl Allen, and Timothy M. Hospedales. 2018. Hypernetwork knowledge graph embeddings. *CoRR*, abs/1808.07018.

Casey Battaglino, Grey Ballard, and Tamara G. Kolda. 2018. A practical randomized CP tensor decomposition. *SIAM J. Matrix Anal. Appl.*, 39(2):876–901.

Lukas Biewald. 2020. Experiment tracking with weights and biases. Software available from wandb.com.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. pages 1247–1250.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale simple question answering with memory networks.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 2787–2795, Red Hook, NY, USA. Curran Associates Inc.

Mikhail Burtsev, Alexander Seliverstov, Rafael Airapetyan, Mikhail Arkhipov, Dilyara Baymurzina, Nickolay Bushkov, Olga Gureenkova, Taras Khakhulin, Yuri Kuratov, Denis Kuznetsov, Alexey Litinsky, Varvara Logacheva, Alexey Lymar, Valentin Malykh, Maxim Petrov, Vadim Polulyakh, Leonid Pugachev, Alexey Sorokin, Maria Vikhreva, and Marat Zaynutdinov. 2018. DeepPavlov: Open-source library for dialogue systems. In *Proceedings of ACL 2018, System Demonstrations*, pages 122–127, Melbourne, Australia. Association for Computational Linguistics.

Hao Cheng, Yaoliang Yu, Xinhua Zhang, Eric Xing, and Dale Schuurmans. 2016. Scalable and sound low-rank tensor learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 1114–1123, Cadiz, Spain. PMLR.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2017. Convolutional 2d knowledge graph embeddings. *CoRR*, abs/1707.01476.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Daniel M. Dunlavy, Tamara G. Kolda, and Evrim Acar. 2011. Temporal link prediction using matrix and tensor factorizations. *ACM Trans. Knowl. Discov. Data*, 5(2):10:1–10:27.

Richard A. Harshman, Paul E. Green, Yoram Wind, and Margaret E. Lundy. 1982. A model for the analysis of asymmetric data in marketing research. *Marketing Science*, 1(2):205–242.

F. L. Hitchcock. 1927. The expression of a tensor or a polyadic as a sum of products. *J. Math. Phys*, 6(1):164–189.

David Hong, Tamara G. Kolda, and Jed A. Duersch. 2020. Generalized canonical polyadic tensor decomposition. *SIAM Review*, 62(1):133–163.

Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, page 105–113, New York, NY, USA. Association for Computing Machinery.

Seyed Mehran Kazemi and David Poole. 2018. Simple embedding for link prediction in knowledge graphs. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 4289–4300, Red Hook, NY, USA. Curran Associates Inc.

Timothee Lacroix, Nicolas Usunier, and Guillaume Obozinski. 2018. Canonical tensor decomposition for knowledge base completion. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2863–2872. PMLR.

Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *Proceedings of the 2nd SysML Conference*, Palo Alto, CA, USA.

Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. 2019. Learning attention-based embeddings for relation prediction in knowledge graphs. *CoRR*, abs/1906.01195.

Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. 2018. A novel embedding model for knowledge base completion based on convolutional neural network. In *Proceedings of the 16th Annual Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT), pages 327–333.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 809–816, Madison, WI, USA. Omnipress.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Thomas Rebele, Fabian M. Suchanek, Johannes Hoffart, Joanna Biega, Erdal Kuzey, and Gerhard Weikum. 2016. YAGO: A multilingual knowledge base from wikipedia, wordnet, and geonames. In *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part II*, pages 177–185.

Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. 2020. You CAN teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*.

Ivan Rybin, Vladislav Korablinov, Pavel Efimov, and Pavel Braslavski. 2021. Rubq 2.0: An innovated russian question answering dataset. In *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings*, volume 12731 of *Lecture Notes in Computer Science*, pages 532–547. Springer.

Kristina Toutanova and Danqi Chen. 2015. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, Beijing, China. Association for Computational Linguistics.

Théo Trouillon, Christopher R. Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. 2017. Knowledge graph completion via complex tensor factorization. *J. Mach. Learn. Res.*, 18(1):4735–4772.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Eric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2071–2080, New York, New York, USA. PMLR.

L. R. Tucker. 1966c. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31:279–311.

Rui Wang, Bicheng Li, Shengwei Hu, Wenqian Du, and Min Zhang. 2020. Knowledge graph embedding via graph attenuated attention networks. *IEEE Access*, 8:5212–5224.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. 2021. Kepler: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, page 1112–1119. AAAI Press.

B. Yang, Wen tau Yih, X. He, Jianfeng Gao, and L. Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575.

Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. 2019. Quaternion knowledge graph embeddings. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.