NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Applied Mathematics and Informatics"

UDC 004.852

Research Project Report on the Topic:

Riemannian Optimization for Link Prediction Using Tensor Decompositions

Submitted by the Student:

group #БПМИ212, 3rd year of study   Sudakov Ilya Alexandrovich

Approved by the Project Supervisor:

Rakhuba Maxim Vladimirovich
Research Fellow
Faculty of Computer Science, HSE University

Moscow 2024

Contents

Abstract

It's well known, that a set of tensors of fixed rank is a riemanian manifold. This information can be applied for construction of effective methods using low-rank tensor approximation. It this project we are advised to modify methods of riemanian optimization on fixed tensor rank manifolds using regularixation. As an applicaton, we have to consider recommendation systems and knowledge graph completion.

Аннотация

Известно, что множество тензоров фиксированного ранга является римановым многообразием. Эту информацию можно использовать для построения эффективных методов поиска малоранговых тензорных приближений. В этом проекте предлагается модифицировать методы римановой оптимзиации на многообразии таккеровских тензоров для использования регуляризации. В качестве приложений необходимо будет рассмотреть рекомендательные системы, а также заполнение баз знаний.

Keywords

Riemannian manifold, Tucker decomposition, knowledge graph, optimization.

1 Introduction

There are a lot of information, that can be represented as subject, object and relation between them. Databases, containing data with that structure is called Knowledge Graph. Obviously, it's too costly to store the whole data, so there is a need in algorithms, that are able to restore and even predict relations, using just part of existing data. Newest, state of the art models use deep non-linear networks, but they are difficult to interpret. So there are still interest in using well researched tensor factorization models.

1.1 Knowledge graph

Let's consider knowledge graph, consisting of subjects, objects and relations between them. We will denote $\mathcal{E}$ and $\mathcal{R}$ as set of all entities and relations respectively presenting in a graph, and $(e_s, r, e_o) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ we will call triple. Also denote as $\mathcal{D} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ the set of all triples corresponding knowledge graph, note that $\mathcal{D}$ fully describes it. $\mathcal{D}$ can be presented as a 3-dimensional binary tensor $T \in \mathbb{R}^{n_r \times n_e \times n_e}$: $T_{e_s,r,e_o} = I[(e_s, r, e_o) \in \mathcal{D}]$ [24].

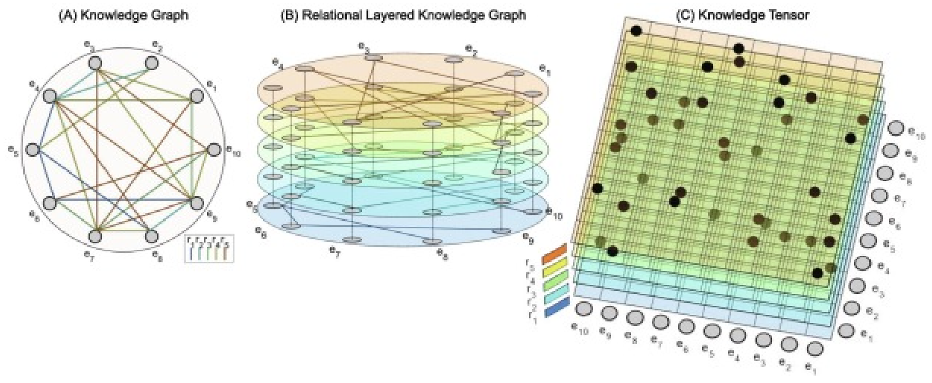

Figure 1.1: Example of Knowledge Graph and it's Tensor presentation.

As our task is to restore missed triples that not presenting in given data, we are given just part of knowledges, we denote it as $\mathcal{D}_{given}$, while set of missed triples as $\mathcal{D}_{missed}$.

1.2 Literature review

Several linear models have been previously suggested for link prediction:

- RESCAL [14] (Nickel et al., 2011) focuses on optimizing a scoring function that includes a bilinear product between subject and object entity vectors along with a full rank relation matrix. Despite being a powerful and expressive model, RESCAL is susceptible to overfitting due to its numerous parameters, which increase quadratically with the embedding dimension and the number of relations in a knowledge graph.

- DistMult [25] (Yang et al., 2015) is a variation of RESCAL with a diagonal matrix per relation to reduce overfitting. However, the linear transformation applied to entity embedding vectors in DistMult is limited.

- ComplEx [18] (Trouillon et al., 2016) extends DistMult into the complex domain by introducing complex conjugates for subject and object entity embeddings of the same entity, enabling the model to handle asymmetric relations.

- SimplE [11] (Kazemi and Poole, 2018) is based on Canonical Polyadic (CP) decomposition and alters CP by making subject and object entity embeddings dependent on each other.

Recent advancements in link prediction have been made with non-linear models:

- ConvE [9] (Dettmers et al., 2018) uses global 2D convolution on reshaped and concatenated subject entity and relation embedding vectors to achieve impressive results.

- HypER [1] (Balazevic et al., 2019) simplifies convolutional modeling by utilizing a hypernetwork to generate 1D convolutional filters for each relation, focusing on extracting relation-specific features from subject entity embeddings.

Another linear model named TuckER model [2], coped with drawbacks of other linear models, it's based on Tucker decomposition of knowledge tensor. We will modify TuckER trying to beat it's result on some popular datasets.

2 Model

2.1 Tucker decomposition

Let denote tensor presentation of the knowledge graph as T, so it contains $n_e n_r^2$ numbers, when number of entities is too big, it's impossible to do computations with the whole tensor, and to store it such way due to memory limitations. To cope with these issues, usually used different tensor factorization, in our article we use Tucker decomposition - tensor analogue of SVD [5], because it was quite simple to implement certain operations that used while riemanian optimization.

Tucker decomposition [3] defined as:

$$T = W \times_1 E_s \times_2 R \times_3 E_o \tag{1}$$

where $\times_i$ indicating the tensor product along the i-th mode,
$W \in \mathbb{R}^{d_e \times d_r \times d_e}, E_s \in \mathbb{R}^{n_e \times d_e}, R \in \mathbb{R}^{n_r \times d_r}, E_o \in \mathbb{R}^{n_e \times d_e}$
It's multilinear rank defined as $mlrank(A) = (rank(A_{(1)}), rank(A_{(2)}), rank(A_{(3)}))$ where $A_{(k)}$ is matrix scan by k-th mode.

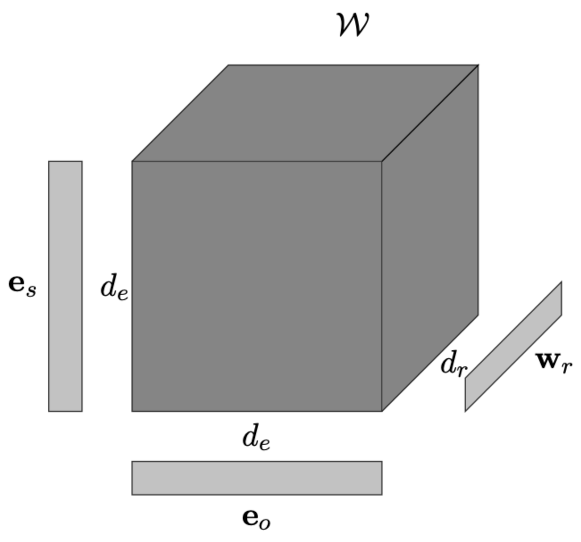

Figure 2.1: Tucker decomposition

We suppose, that T contains logits for each posible triple, that we convert to it's probability. Formally, to get prediction for triple (e_s, r, e_o), we will compute the probability of it applying sigmoid function to T_{e_s,r,e_o}. To get T_{e_s,r,e_o}, we can compute it using properties of Tucker decomposition.

$$a_{i,j,k} = \sum_{\alpha,\beta,\gamma=1}^{r_1,r_2,r_3} g_{\alpha\beta\gamma} u_{i\alpha} v_{j\beta} w_{k\gamma} \tag{2}$$

it's computational difficulty: $O(r_1, r_2, r_3)$

2.2 Riemanian manifold

Let's define manifold of tensors of fixed multilinear rank:

$$\mathcal{M}_k := \left\{ X \in \mathbb{R}^{n_e \times n_r \times n_r} : \text{mlrank}(X) = (d_e, d_r, d_r) \right\} \tag{3}$$

It is well known that $\mathcal{M}_k$ is a smooth manifold. When the objective function f is also smooth, problem is a smooth optimization problem, which can be solved by methods from Riemannian optimization.

2.3 Riemanian optimization

Let's consider generalization of standard unconstrained optimization, where the search space is $\mathbb{R}^n$, to optimization of a smooth objective function on a Riemannian manifold, named Riemanian optimization [19].

As we defined loss function, we can now define our optimization task:

$$\begin{aligned} \underset{X}{\text{minimize}} \quad & f(X) := \mathcal{L}(\mathrm{P}_\Omega(X), \mathrm{P}_\Omega(A)) \\ \text{subject to} \quad & X \in \mathcal{M}_k := \{X \in \mathbb{R}^{n_e \times n_r \times n_r} : \operatorname{mlrank}(X) = (d_e, d_r, d_r)\} \end{aligned}$$

Now, to implement optimizatoin algorithm, we need to use some important concepts:

Riemanian gradient $f\,(X_i)$ is a tangent vector ξ_i corresponding to direction of fastest increase of f, but restricted to $T_{X_i}M_k$ tangent space of X_i

If we have our tangent vector, we can not just add it to X_i, as $\xi_i + X_i$ don't have to $\in M_k$. So we must be able to map $X \in T_{X_i}M_k$ to $X \in M_k$

If we will use gradient descent with momentum, accumulated gradient doesn't belong to current tangent space, so we have to move it to current point and restrict to current tangent space. It is called vector transport.

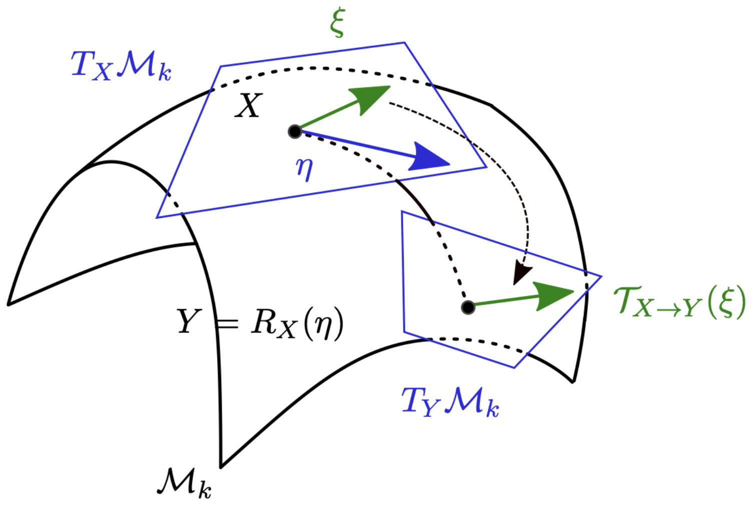

Figure 2.2: Vector transport

2.4 Algorithm

Algorithm 1 Riemanian optimization

```
Require: initial iterate $T_0 \in \mathcal{M}_k$, tangent vector $\mu_0 = 0$
  for $i = 1, 2, ...$ do
     Compute the gradient:
     $\xi_i \leftarrow \text{grad } f(T_i)$
     Compute a conjugate direction:
     $\mu_i \leftarrow -\xi_i + \beta_i \mathcal{T}_{T_{i-1}}(\mu_{i-1})$
     Update $T_i$:
     $T_i \leftarrow R_{T_i}(\mu_i)$
  end for
```

3 Implementation

3.1 Riemanian package

To implement training of TuckER model with riemanian optimization of loss, we use python package tucker-riemopt [15], that provides special functions corresponding to Tucker representation of tensors.

There implemented the following classes and it's methods:

- `Tucker`: class for tucker representation of tensor
- `from_dense`: to get tucker representation of dense tensor
- `to_dense`: to get dense representation of tucker tensor
- `flat_inner`: to calculate inner product of two given tucker tensors
- `round`: to find best (in MSE error sense) approximation of tensor on given manifold
- `norm`: to compute the Frobenius norm of tucker representation
- `TangentVector`: factorized representation of tangent vector in space of tensors
- `construct`: to get tucker representation of tangent vector
- `grad`: to compute the gradient of given scalar function at given point X
- `project`: to project given X from one tangent space to another
- `add`: to calculate sum of two given tangent vectors, starting from same point

- `norm`: to compute the Frobenius norm of tangent vectors

The methods listed above also provided for tensors with symmetric factors, which turned out to be more useful.

3.2 Loss evaluation

In order to compute the loss, we first organize the data into pairs consisting of a subject and a relation. We then predict which objects are associated with this relation for each subject. By comparing the predicted probabilities with the actual ground truth values, we determine the loss.

3.3 Score evaluation

For every test triple, we create several candidate triples by combining the test entity-relation pair with all entities in the dataset. We then rank these candidate triples based on their scores. We follow the filtered setting, where all known true triples are excluded from the candidate set except for the current test triple. Our evaluation metrics are commonly used in link prediction research: mean reciprocal rank (MRR) [22] and hits@k [21], where k can be 1, 3, or 10. Mean reciprocal rank calculates the average of the reciprocal of the mean rank assigned to the true triple across all candidate triples. Hits@k measures the proportion of times a true triple is ranked among the top k candidate triples.

3.4 Model implementation

Using `tucker_riemanian` package and `Torch` [8] backend, we implemented:

- `RTuckER`: analogue for original model, which doesn't have Batchnorm [16] and Dropout [17] layers due to riemanian context
- `SGD`: class providing riemanian SGD optimization
- `Adam`: class providing riemanian Adam optimization
- `Data`: class for processing datasets
- `get_loss_fn`: function that returns needed loss function for given batch
- `Experiment`: class for training and evaluating model

- `evaluate`: method for calculating metrics of model quality

The code for training model with such algorithm is provided in github.

4 Experiments

4.1 Datasets

To test our models we used classic link prediction datasets:

- FB15k [4]: collection of knowledge base relation triples and textual references to pairs of Freebase entities. It comprises 592,213 triplets involving 14,951 entities and 1,345 relationships. FB15K-237 is a modified version of the original dataset where reverse relations have been eliminated due to the discovery that a significant number of test triplets could be derived by reversing training triplets.

- FB15k-237 [9]: dataset designed for link prediction, derived from FB15k. While FB15k includes 1,345 relations, 14,951 entities, and 592,213 triplets, many of these triplets are inverses leading to information leakage from training to testing and validation sets.

- WN18 [4]: dataset containing 18 relations extracted from WordNet covering around 41,000 synsets, resulting in 141,442 triplets.

- WN18RR [4]: link prediction dataset built from WN18, a subset of WordNet. WN18 consists of 18 relations and 40,943 entities. However, several text triples are generated by inverting triples from the training set. Hence, the WN18RR dataset is established to prevent test leakage with inverse relations in the evaluation dataset.

	entities	relations	triplets
FB15k	14,951	1,345	592,213
FB15k-237	14,541	237	310,116
WN18	40,943	18	141,442
WN18RR	40,943	11	93,003

Table 4.1: Dataset statistics.

4.2 First approach

To start our exploration, we tried to implement TuckER model without sharing factor, that made model more expressive, but more prone to overfitting, futhermore we are deprived of

Batchnorm and Dropout layers. There was taken multilinear rank (200, 200, 200) for dataset FB15k-237 and rank (30, 200, 200) for dataset WN18, other datasets weren't used for this experiment. We took negative log-likelyhood as loss, as in original paper. The simple SGD [23] was taken as an optimizer with constant learning rate = 1e9 and batch size = 1024 for 200 epochs and with constant learning rate = 1e9 and batch size = 1024 for 100 epochs, but recieved results were much worse than original TuckER (see table).

Then we tried to implement same model, but using shared factor for entities, so model became simpler and trained more quickly. Obtained results were already similar to our baseline. That's why we decided not to use assymetric model.

setup	hits@1	hits@3	hits@10	mrr
WN18	0.599	0.736	0.837	0.654
FB15k-237	0.143	0.241	0.336	0.202
WN18 + sharing	0.687	0.752	0.814	0.731
FB15k-237 + sharing	0.214	0.319	0.452	0.293

Table 4.2: First approach.

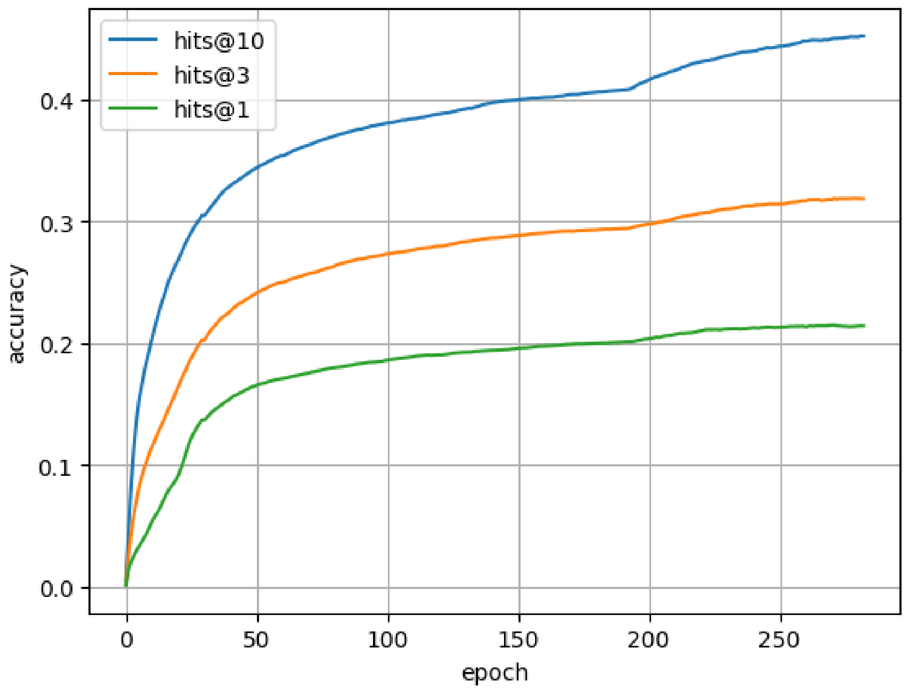

Figure 4.1: Training on FB15k-237

To observe overfitting of our model, we trained original model and compared score metrics at same loss. It turned out, that for WN18 dataset, at loss = 0.0015 our model had metrics hits@1 = 0.439, hits@3 = 0.631 , hits@10 = 0.785 while original model at loss = 0.0017 had metrics hits@1 = 0.599, hits@3 = 0.736, hits@10 = 0.837.

So in our next approach, we tried to add some regularization technics.

4.3 Second approach

We decided to use Stochastic Gradient Descent with momentum to speed up the training process. This change led to a significant improvement in training convergence rates, making the training process approximately two times faster.

We tried to use SGD on FB15k-237 dataset with previous setup, where

$W \in \mathbb{R}^{200\times200\times200}$

$R \in \mathbb{R}^{1,345\times200}$

$E1 = E2 \in \mathbb{R}^{14,951\times200}$

Without any regularization we got quite bad result, see table 4.3.

We then added the L2 norm [10] of the model to the previous loss function, and the corresponding results improved significantly.

Additionally, we experimented with the L1 norm as suggested in [6]. Since the L1 norm is not unitary-invariant, we evaluated the L1 norm of the predictions batch due to the large size of the full tensor. This model trained very quickly but resulted in very low quality. We then attempted to change the loss from negative log-likelihood to the Frobenius norm, but this also yielded poor results during evaluation.

Subsequently, we tried using a pretrained model that was trained with the Frobenius norm loss and trained it with negative log-likelihood (and vice versa), but its quality deteriorated.

It turned out that training was too slow due to the small Riemannian gradient norm, so we experimented with artificially setting it to be equal to 1. With our default loss function, we achieved pretty good results. However, when we tried adding regularization to this approach, the quality decreased again.

setup	hits@1	hits@3	hits@10	mrr	logs
original TuckER	0.266	0.394	0.544	0.358	
default RTuckER	0.1935	0.2864	0.4079	0.2653	ref
RTuckER + $l_2(T)$	0.2190	0.3198	0.4475	0.2954	ref
RTuckER + $l_1(X)$	0.1651	0.2398	0.3428	0.2243	ref
RTuckER: $\|P_{Omega}(X - A)\|_F + l_1(X)$	0.1770	0.2674	0.3970	0.2497	ref
RTuckER: grad norm = 1	0.2215	0.3289	0.4661	0.3024	ref

Table 4.3: Regularization approaches.

4.4 Adam optimizer

After spending a lot of time on experiments with SGD, we read some [7] papers about riemaninan optimization, where the pseudo Adam [12] optimizer was suggested.

Adam calculates a unique learning rate for each parameter to adjust the step size adaptively, ensuring that the gradient direction remains accurate. In contrast, a single adaptive learning rate is assigned to each weight vector on the manifold. The algorithm for Adam is outlined in Algorithm 2.

Algorithm 2 Riemaninan Adam

```
Require: initial random state $y_0 \in \mathbb{R}^{n\times 1}$, initial zero state $y_0 \in \mathbb{R}^{n\times 1}$, initial scalar $v_0 = 0$
  for $i = 1, 2, ...$ do
    $\eta_t \leftarrow \eta\sqrt{1-\beta_2^t}/\left(1-\beta_1^t\right)$
    $g \leftarrow \partial f\left(y_{t-1}\right)/\partial y$
    $h \leftarrow g - \left(y_{t-1}^\top g\right) y_{t-1}$
    $m_t \leftarrow \beta_1 \cdot \tau_{t-1} + (1-\beta_1) \cdot h$
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot h^\top h$
    $d \leftarrow -\eta_t \cdot m_t/\sqrt{v_t + \epsilon}$
    $y_t \leftarrow \exp_{y_{t-1}}(d)$
    $\tau_t \leftarrow \mathrm{pt}_{y_{t-1}}\left(m_t; d\right)$
  end for
```

4.5 Shifted Binary Cross Entropy

It is a well-known that the datasets we work with contain a limited number of triplets in comparison to their maximum potential count. For instance, in the case of WN18RR, we have access to only 141,442 triplets while the theoretical maximum reaches approximately 3e10. Similarly, for FB15k-237, we are constrained to just 310,116 triplets despite the potential for around 5e10. When employing the standard BCEloss [20], our model is typically trained to assign high probabilities exclusively to the provided data points, consequently leading to lower probabilities being assigned to unseen test data.

To address this inherent bias, we have adopted a modified approach by transitioning from the conventional BCE loss function to a shifted BCE loss. This adjustment aims to mitigate the model's tendency to overfit on the training data by introducing a shift parameter that encourages a more balanced distribution of probabilities across both known and unseen instances:

$$\text{Shifted BCE}(\alpha) = -\text{mean}(y_i \log x_i + \alpha(1-y_i)\log\left(1-x_i\right)) \cdot \frac{2}{1+\alpha}$$

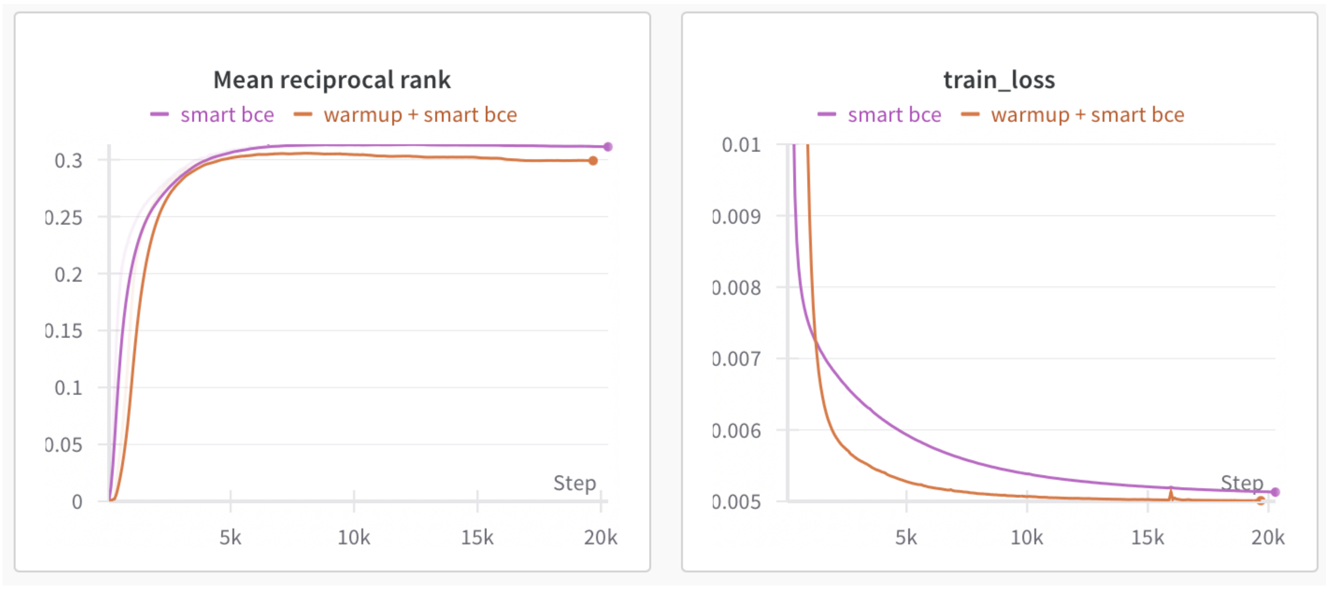

Figure 4.2: Warmup comparison

where we multiplied by $\frac{2}{1+\alpha}$ to keep the magnitude of loss.

4.6 Third approach

In our latest strategy, we implemented the Adam optimizer without artificially enlarging the gradient norm. This approach involved L2 regularization alongside a modified Binary Cross Entropy loss function with a novel shift parameter (set at $\alpha = 0.5$). The model's performance on the FB15k-237 dataset provided metrics: hits@1 $=$ 0.231, hits@3 $=$ 0.342, and hits@10 $=$ 0.480. Given these encouraging outcomes, we have decided to continue using the Adam optimizer with the adjusted BCE loss function for future iterations of our model.

4.7 Warmup learning rate

We also observed a high variance in the training loss at the beginning of training due to momentum in the Adam optimizer. That's why we tried to use a learning rate warmup [13] to make the training more stable. We used a linear schedule $[maxlr/100, maxlr]$ for 10 epochs. This made the training faster for the first epochs, but it didn't converge to our best result, so we didn't use warmup anymore.

An example of training a model with warmup is provided in the Figure 4.2.

5 Results

Finally, we trained the model with the following parameters:

dataset	rank	lr	regularization	batch size	decay rate
WN18RR	(200, 20, 200)	1e-1	1e-10	2048	0.999
FB15k-237	(200, 200, 200)	1e-1	1e-10	512	0.999

Table 5.1: Training paremeters

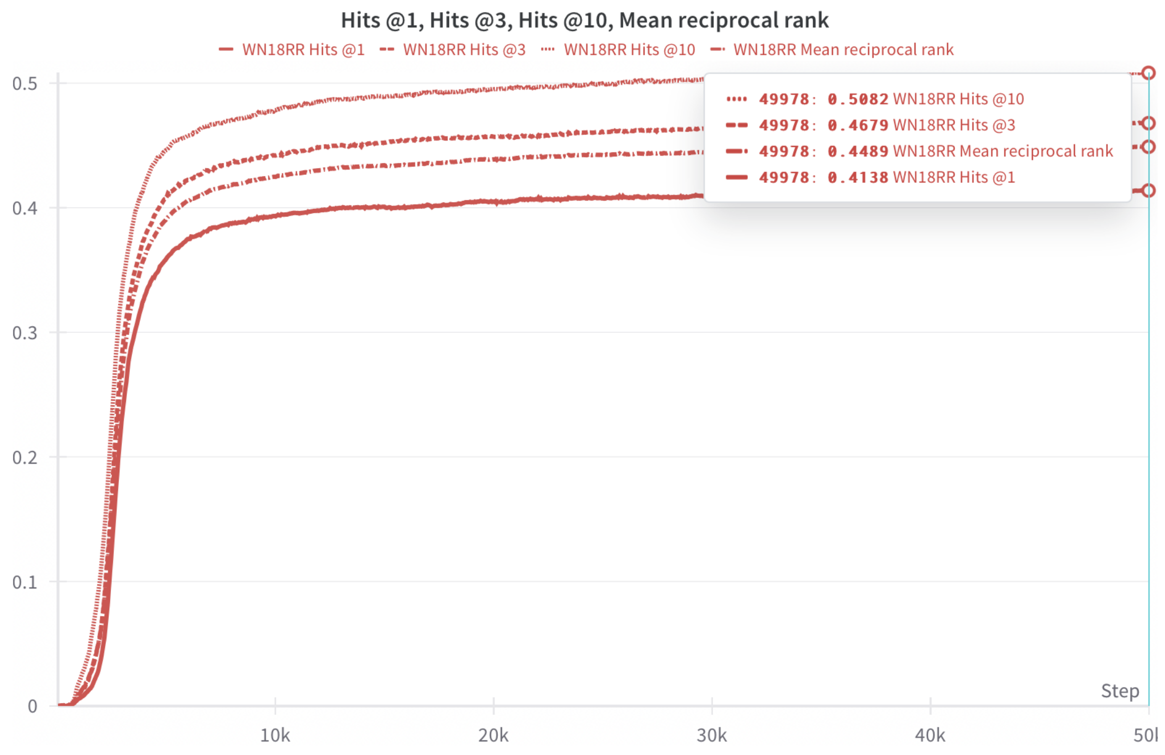

Figure 5.1: training WN18RR

We used the Adam optimizer with normalized gradients and ridge regularization. This setup achieved the best result, but it was still quite worse than the baseline.

The metrics during the training process for both datasets provided in 5.1 and 5.2

model	dataset	hits@1	hits@3	hits@10	mrr
RTuckER	WN18RR	0.414	0.468	0.508	0.449
TuckER	WN18RR	0.443	0.482	0.526	0.470
RTuckER	FB15k-237	0.231	0.342	0.481	0.313
TuckER	FB15k-237	0.266	0.394	0.544	0.358

Table 5.2: Achieved results

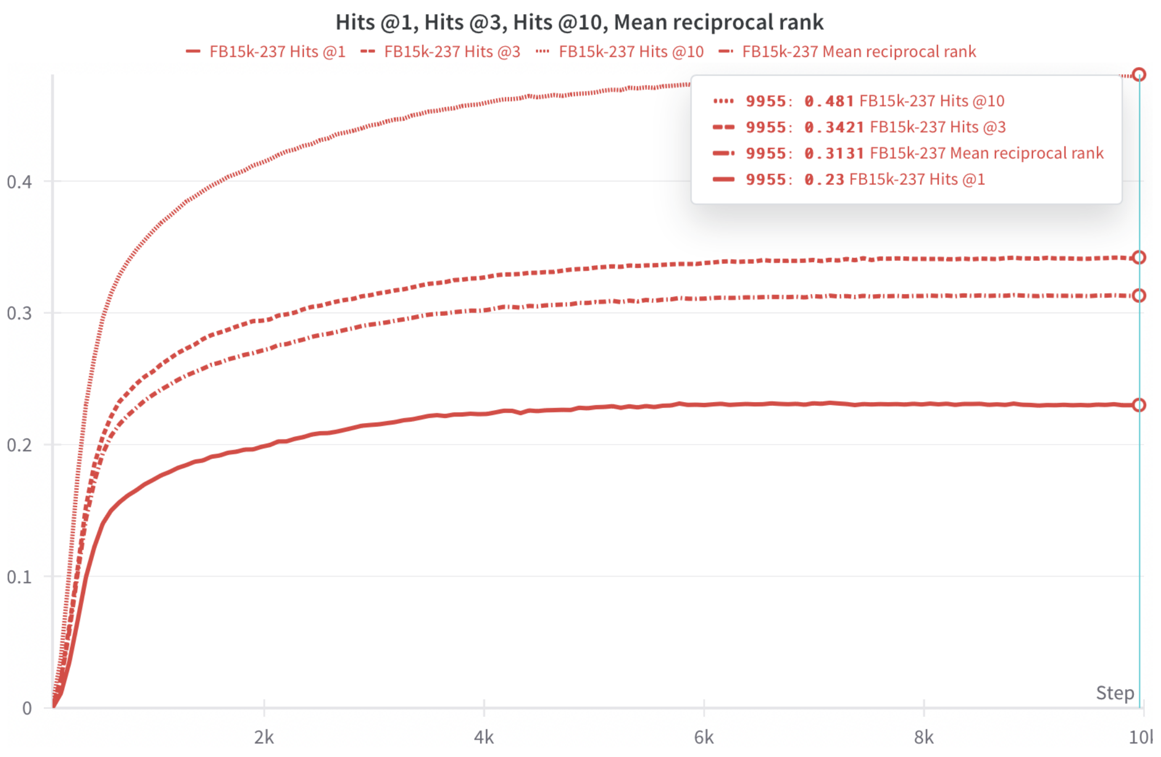

Figure 5.2: training FB15k-237

6 Future work

When considering model improvements, one approach is to explore asymmetric Tucker decomposition, where the dimensions of the entity and relation spaces (E_s and E_o) are not equal. This can potentially offer a more flexible representation that captures the underlying structure of the data more effectively. By adjusting the hyperparameters and implementing different schedulers for regularization and learning rate, it is possible to fine-tune the model for better performance and results.

TuckER, as a model, applies regularization techniques like Batch Normalization and Dropout to prevent overfitting during training. To further enhance the model's generalization capabilities, one could consider introducing Riemannian-aware layers. These layers can help the model learn more complex geometrical structures, potentially leading to improved performance on tasks requiring a deeper understanding of relationships.

Additionally, experimenting with different rank configurations, different to proposed in TuckER could be beneficial.

This experiments can provide valuable insights into the optimal configuration for the tensor decomposition, potentially leading to enhanced performance on knowledge graph completion tasks or other related applications.

References

[1] Ivana Balažević, Carl Allen, and Timothy M Hospedales. "Hypernetwork knowledge graph embeddings". In: *Artificial Neural Networks and Machine Learning–ICANN 2019: Workshop and Special Sessions: 28th International Conference on Artificial Neural Networks, Munich, Germany, September 17–19, 2019, Proceedings 28*. Springer. 2019, pp. 553–565.

[2] Ivana Balažević, Carl Allen, and Timothy M Hospedales. "Tucker: Tensor factorization for knowledge graph completion". In: *arXiv preprint arXiv:1901.09590* (2019).

[3] Vineet Bhatt, Sunil Kumar, and Seema Saini. "Tucker decomposition and applications". In: *Materials Today: Proceedings* 46 (2021), pp. 10787–10792.

[4] 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* 26 (2013).

[5] Steven Brunton and J. Kutz. "Singular Value Decomposition (SVD)". In: Feb. 2019, pp. 3–46. ISBN: 9781108422093. DOI: 10.1017/9781108380690.002.

[6] Léopold Cambier and P-A Absil. "Robust low-rank matrix completion by Riemannian optimization". In: *SIAM Journal on Scientific Computing* 38.5 (2016), S440–S460.

[7] Minhyung Cho and Jaehyung Lee. "Riemannian approach to batch normalization". In: *Advances in Neural Information Processing Systems* 30 (2017).

[8] R. Collobert, K. Kavukcuoglu, and C. Farabet. "Torch7: A Matlab-like Environment for Machine Learning". In: *BigLearn, NIPS Workshop*. 2011.

[9] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. "Convolutional 2d knowledge graph embeddings". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[10] Trevor Hastie. "Ridge regularization: An essential concept in data science". In: *Technometrics* 62.4 (2020), pp. 426–433.

[11] Seyed Mehran Kazemi and David Poole. "Simple embedding for link prediction in knowledge graphs". In: *Advances in neural information processing systems* 31 (2018).

[12] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[13] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. “On the variance of the adaptive learning rate and beyond”. In: *arXiv preprint arXiv:1908.03265* (2019).

[14] Maximilian Nickel, Volker Tresp, Hans-Peter Kriegel, et al. “A three-way model for collective learning on multi-relational data.” In: *Icml*. Vol. 11. 10.5555. 2011, pp. 3104482–3104584.

[15] Ivan Peshekhonov. *tucker$_r$iemopt*. https://johanddc.github.io/tucker_riemopt/api.html. [Online; accessed 13-February-2024]. 2023.

[16] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. “How does batch normalization help optimization?” In: *Advances in neural information processing systems* 31 (2018).

[17] Vincent Tinto. “Dropout from higher education: A theoretical synthesis of recent research”. In: *Review of educational research* 45.1 (1975), pp. 89–125.

[18] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. “Complex embeddings for simple link prediction”. In: *International conference on machine learning*. PMLR. 2016, pp. 2071–2080.

[19] Bart Vandereycken. “Low-rank matrix completion by Riemannian optimization”. In: *SIAM Journal on Optimization* 23.2 (2013), pp. 1214–1236.

[20] Wikipedia. *Cross-entropy — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Cross-entropy&oldid=1221840853. [Online; accessed 12-May-2024]. 2024.

[21] Wikipedia. *Knowledge graph embedding — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Knowledge%20graph%20embedding&oldid=1192045377. [Online; accessed 11-May-2024]. 2024.

[22] Wikipedia. *Mean reciprocal rank — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Mean%20reciprocal%20rank&oldid=1218582630. [Online; accessed 11-May-2024]. 2024.

[23] Wikipedia. *Stochastic gradient descent — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Stochastic%20gradient%20descent&oldid=1222784950. [Online; accessed 11-May-2024]. 2024.

[24] Jianing Xi, Zhaoji Miao, Longzhong Liu, Xuebing Yang, Wensheng Zhang, Qinghua Huang, and Xuelong Li. “Knowledge tensor embedding framework with association enhancement for breast ultrasound diagnosis of limited labeled samples”. In: *Neurocomputing* 468 (2022), pp. 60–70.

[25] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. “Embedding entities and relations for learning and inference in knowledge bases”. In: *arXiv preprint arXiv:1412.6575* (2014).