Vrije Universiteit Amsterdam

Bachelor Thesis

Selective Reduction of Dimensions for NodePiece Anchors

Author: Razvan Gabriel Olaru (2619927)

Supervisor: Michael Cochez
2nd reader: Nikos Kondylidis

A thesis submitted in fulfilment of the requirements for the VU Bachelor of Science degree in Computer Science

July 25, 2022

1 Abstract

The continuous innovative research around Knowledge Graphs in recent times has contributed to the complexity of Knowledge Graph Embedding models. A node can be represented by a multi-dimensional tensor and also the relationships between nodes. NodePiece is using an approach that hashes each node by K anchor nodes and M relation types in its relational context[7]. In this paper, we present a technique that selectively reduces the dimensionality of the tensors before they are fed to NodePiece[7]. Our approach includes clustering nodes based on degree centrality, computing new reduced dimensionalities and incorporating MLP architectures into the NodePiece encoder model. By applying dimensionality reduction to the embeddings vectors, we manage to significantly diminish the size of the embedding tensors, thus, Knowledge Graphs Embedding models can benefit from it when handling large datasets.

2 Introduction

One of the most prominent mathematicians of the 18th century, Leonhard Euler, proposed the first fundamental concept of graphs. He is frequently credited as the creator of Graph Theory for his work on the distinguished problem of the "Seven Bridges of Königsberg". The challenge was constructing a route through the city that would traverse each bridge exactly once[6].

Graph Theory studies mathematical structures used to model pairwise relations between objects. A graph is composed of *vertices* (also called *nodes*) and *edges* (also called *links*). A graph is an ordered pair G = (V, E), where V is the set of vertices and $E \subseteq \{\{x, y\} \mid x, y \in V \textit{ and } x \neq y\}$ is the set of edges[16].

Multiple types of graphs exist, such as weighted, undirected, and directed graphs (also called Digraphs). A weighted graph has numbers (weights) assigned to each edge. They can represent cost or lengths or others, depending on the underlying context. Throughout this paper, we will focus on directed weighted graphs. A directed edge, also known as an arc, is an ordered pair (x,y) or (y,x), with x and y being (the two connected) nodes. In a directed graph, the degree of a node has two variants: in-degree, which indicates the number of edges incident on it, and out-degree which refers to the number of links (going out edges). The degree of a node in a digraph is the sum of in-degree and out-degree.

Knowledge graphs are continuously getting more complex. An embedding is the numerical representation of an entity or a relation on a graph. **K**nowledge **G**raph **E**mbedding (KGE) are needed in order to transform a graph into a numerical state that can be used as input for machine learning algorithms[11]. Encoding the information contained in a KG in embedding vectors consisting of relations and nodes requires high-dimensional embedding vectors to achieve expressive representations. Using this type of vector significantly affects the model's time and space complexity, especially during training. The number of parameters increases fast, and KGE model becomes more complex as the embedding dimension expands. Another drawback is the amount of memory needed for storing these embeddings representations in memory. As a result, scalability issues arise, especially for large KGs. NodePiece is already significantly reducing the number of parameters of KGE models by representing nodes through a pre-selected set of anchor nodes instead of creating an embedding for every entity in the graph. A full NodePiece vocabulary is constructed afterwards using anchor nodes and relation types[7]. To address these downsides, a couple of experiments will be implemented to further reduce the dimensionality of the embedding vectors in NodePiece.

In Section 4, we will discuss the methods that were used to achieve our goal. Firstly, we computed the degree centrality of each node and used the result to split the KG into clusters. After this, we compute the new vector dimensions based on the number of anchors in each cluster. Then, we concatenate the anchors with the same length in batches and feed them to a **M**ulti-**L**ayer **P**erceptron (MLP). The output of the MLP will be a higher dimensional vector (according to the NodePiece requirements) that is fed to the NodePiece encoder.

3 Related work

Previous related work reveals two main approaches for KGE methods: embeddings-based and neural network methods. Multiple embedding spaces have been used to develop KGE models(Euclidian, hyperbolic, and complex space).

An initial method proposed in Euclidian space was TransE[2] , an energy-based model for learning low-dimensional embeddings of entities. It models the relation as a translation between head and tail entities and achieves a reduced number of parameters by learning only one vector for each entity and relation [17].

TransH[15] models a relation as a hyperplane with a translation operation on it and introduces relations such as reflexive, one-to-many, many-to-one and many-to-many. The model also uses these relations to reduce false-negative labels in training. As a result, it significantly improves TransE while preserving comparable scalability potential.

TransD[8] is a more fine-grained model that uses two vectors to represent the entity and relation object. The first vector represents the meaning of an entity, while the second vector is used to construct a mapping matrix dynamically. TransD uses fewer parameters and has no matrix-vector multiplication operations, making the model more scalable on large KGs. ComplEx[12], is using complex vectors to compute the Hermitian dot product, achieving antisymmetric relations while retaining the efficiency benefits of the dot product and measures validity by matching the semantics of relations and entities in the complex space.

Another approach is RotatE[10] which defines relations as rotations from source to target entity on a two-dimensional plane, achieving relations such as symmetry/anti-symmetry, inversion and composition. The model proposes a general technique: a self-adversarial negative sampling, which generates negative samples according to the current entity and relation embeddings and is used for efficiently training the RotatE. Furthermore, ATTH[3] uses an embedding method to combine hyperbolic reflections and rotations with attention to model complex relational patterns.
As opposed to embedding-based methods, various deep neural network approaches have been proposed. Neural Tensor Network (NTN)[4] suggested a feed-forward neural tensor network. Then ConvE[5] got compelling results using a Convolutional Neural Network (CNN) on two-dimensional embeddings. Another approach was InteractE[13], which significantly improved by increasing the number of interactions between entities and relations. The dimensionality of the embeddings is further increased as a result of encoding learned

information into embeddings. This is the notable downside of the embedding-based methods. In order to preserve descriptive representations, increasing the number of dimensions is desired. This constraint generates scalability issues on large KGs.

4 Methods

In this section, we will describe the steps used to reduce the dimensions for the anchor embeddings selectively. We start by computing the centrality of every node. Centrality metrics return a score that indicates how central/essential a node is in a graph. After that, we create clusters based on each node's centrality scores and compute the new dimension number for each anchor in each cluster. Next, the anchors with the same dimensions are concatenated and fed as an input to an MLP, and its output is used to feed the NodePiece encoder. The last step is the extraction of the required anchors, which is done by applying a mask on the MLP output that selects the anchor embeddings wrt. an inverted index list.

4.1 Degree Centrality

The first step in our approach is to quantify the importance of nodes in the input graph. In Graph Theory, centrality is crucial for detecting significant nodes in a graph by quantifying the usefulness of different nodes (how "central" is a node within the network). There are various types of centrality, each defining a node's significance from a unique angle and contributing valuable analytical data about a graph and its nodes.

According to the Degree Centrality metric, a node's importance in a graph is determined by its degree; the higher a node's degree, the more significant it is in the graph. Examining Figure 1, we can see that node a1 has an in-degree of 0 and out-degree of 2, while node a2 has an in-degree of 3 and out-degree of 1. As a consequence, node a2 is more important in our graph.

Therefore we apply degree centrality for all nodes in the graph, as defined in Equation 1. More precisely, it is defined as the number of incident edges upon a node a_i. The degree centrality can be standardized by dividing each score by $n-1$ where n is the total number of nodes.

$$\begin{aligned} D\left(a_1\right) &= \sum_{j=1}^{n} m\left(a_1, a_j\right) \ \textit{where} \\ & m\left(a_1, a_j\right) = 1 \ \textit{if there is a link between node } a_1 \textit{ and node } a_j \\ & m\left(a_1, a_j\right) = 0 \ \textit{otherwise} \end{aligned} \tag{1}$$

Other alternatives for calculating a centrality metric include eigenvector, closeness, betweenness and group centrality.

4.2 K-means Clustering + Elbow Method

As a result of applying the previously-mentioned Degree Centrality metric, we obtain an importance measure for each node in the graph. The next step is to cluster nodes based on importance. Unsupervised machine learning techniques such as the k-means algorithm

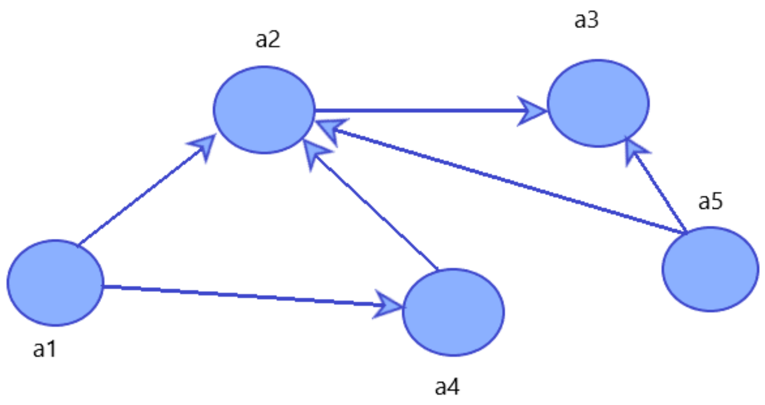


Fig. 1. A directed graph

are used to identify object clusters within a dataset. A cluster is a collection of data points that share similarities to other(s) in their cluster (and dissimilarities with the data points in the other clusters). The approach of the K-means algorithm is the following:

1. Partition the data in k pre-defined, distinct, and non-overlapping clusters
2. Reduce the sum of distances between each point and its corresponding cluster centroid
3. Iterates until there are no more changes to the centroids

A drawback of K-means is that the number of clusters (value of k) has to be predetermined. In order to address this issue, we use the Elbow method, which varies the number of clusters, and for each k, we are computing the WCSS (Within-Cluster Sum of Square).

$$WCSS = \sum_{C_k}^{C_n} \left(\sum_{d_i in C_i}^{d_m} distance(d_i, C_k)^2 \right)$$

Fig. 2. Within-Cluster Sum of Square formula

In order to avoid the issues caused by the random initialization of the K-means algorithm, we use the K-means++ method, which helps speed up convergence. The method works as follows:

1. Pick an initial centroid point C_1 randomly

2. Compute the distance from the selected centroid to all of the points in the dataset. The distance from the farthest centroid to a point x$_i$ can be computed using the following formula:

   $d_i \, = \, max_{(j:1 \, -> m)} \, ||x_i \, - \, C_j||^2, \, where \, m \, is \, the \, number \, of \, centroids \, alredy \, picked$

3. Make x$_i$ the new centroid that has maximum probability proportional to d$_i$
4. Repeat the steps until k-centroids are found

4.3 Getting reduced lengths

Given the previous two steps, we now have k clusters of nodes based on their centrality in the graph. Given that we want to apply dimensionality reduction on the anchors of the graph, we extract those and get k clusters, each of n_i number of anchors. The next step is to decide which approach to apply for selecting the new dimensionality. One straightforward method is to select one reduced dimensionality d' for all anchors, independent of their cluster. Another method is to pick different dimensions d'_i taking into account which clusters the anchors come from. The intuition behind this method is that if a cluster contains more anchors, then their overall impact on the graph is less important, while a cluster with just a few anchors should convey more information. As a result, we calculate the new anchor lengths using the formula in Equation 2. As a result, all anchors placed in the same cluster will result in the same reduced dimensionality.

$$d'(a_i) = \frac{d}{len(c(a_i))} \tag{2}$$

where $d'(a_i)$ represents the new reduced length of the current anchor, d is the initial embedding dimension, $c(a_i)$ is the cluster in which anchor a_i is placed. More precisely, $len(c(a_i))$ represents the total number of nodes in cluster $c(a_i)$.

Once we calculate the new embedding dimensions of the anchors, they will be further passed into an MLP as described in Section 4.4. To avoid redundant computational complexity, we concatenate the anchors based on their new reduced length. This will result in k matrices, each having the number of anchors within that cluster as rows and the new reduced dimensionality as columns.

4.4 MLP

This project integrates an MLP model within the NodePiece encoder architecture. A Multilayer Perceptron is one of the most popular feed-forward neural network models. It is constructed from interconnected neurons transferring information to each other. Each MLP has at least 3 layers: input, output and one or multiple hidden layers. Each connection between layers has weights assigned to them, and each linear combination propagates to the next layer. Backpropagation is the technique that provides the MLP with the ability to iteratively adjust the weights in the network to minimize the cost function. In every iteration, the gradient of a loss function, for example, Mean Squared Error, is calculated across all input and output pairs. The iterative process is finished when the gradient has converged[1] (the newly computed gradient has not changed above a set threshold).

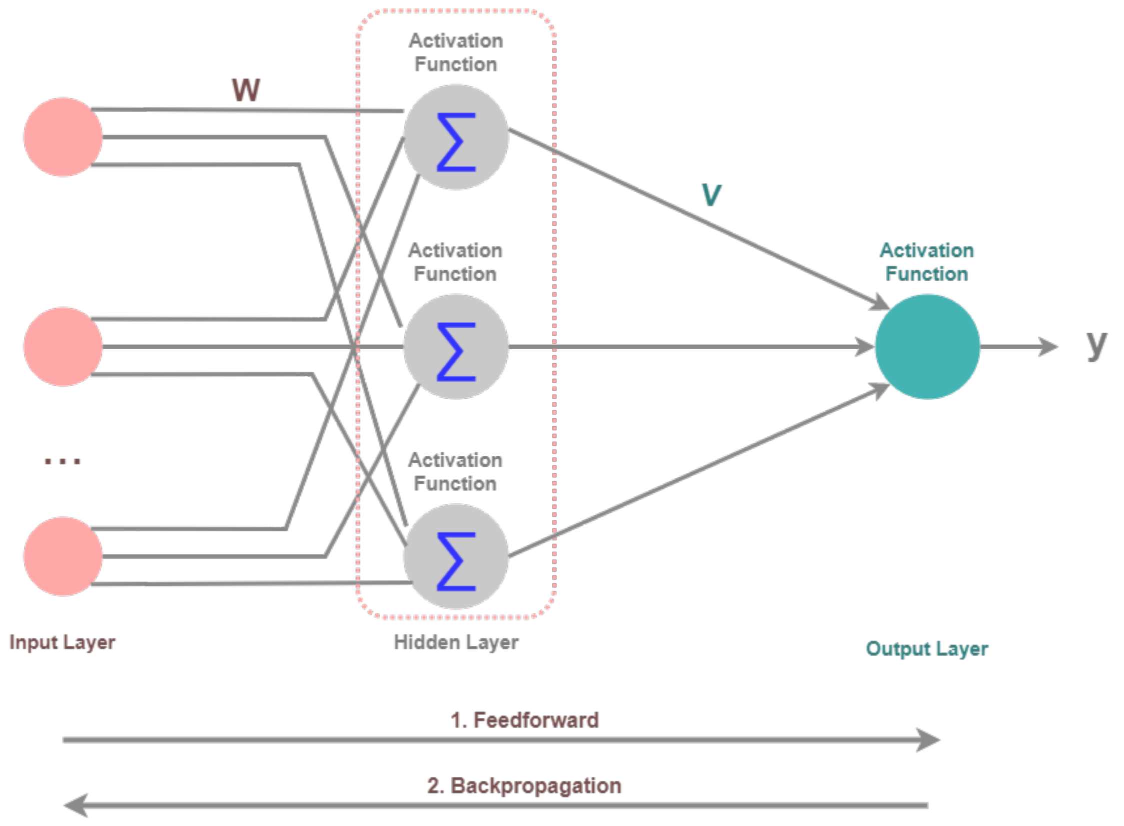


Fig. 3. Multi-Layer Perceptron

We use the MLP as a method to upscale lower dimensionality vectors. More precisely, the reduced lengths d' computed in Section 4.3 are used to calculate lower dimensionality embeddings in NodePiece. These embeddings must then be passed into the encoder of NodePiece, which expects a higher dimensionality input of size d. We use the MLP as a bridge to make the new embeddings usable. Therefore, the model receives as input embeddings of the new reduced shape and outputs the embeddings projected in the initial dimension. Given that there are k clusters, there are also k new reduced dimensionalities. This implies k MLP models with different input sizes must be created and trained. We tested our MLP with several values of hidden neurons: 32, 64, 128, 512 and choose 64 as the model obtains the best metric scores when using this number of hidden neurons.

	FB15K-237				WN18RR			
#Hidden neurons	32	64	128	512	32	64	128	512
#P(M)	3.098	3.209	3.433	4.773	4.436	4.463	4.515	4.83
MRR	0.244	0.251			0.322	0.318		
Hits@10	0.41	0.416			0.471	0.478		

Fig. 4. Parameter count and metric scores of default NodePiece vs NodePiece + Variable Sized Embeddings with different amounts of hidden neurons. * results taken from (Galkin et al., 2021) [7]. #P is a total parameter count (millions).

4.5 Pseudocode

```
Integrate an MLP Sequential Layer into the NodePiece architecture
Create a NodePiece tokenizer with a set of specified hyperparameters
Get graph $G = (V, E)$ from the tokenizer
    Step 1: Calculate degree centrality of G
    Step 2: For k in $range(1, 10)$:
            Step 2.1: Apply k-means clustering wrt degree centrality
            Step 2.2: Calculate SSE
    Step 3: Apply Elbow Rule to select the best k
    Step 4: For each cluster c:
            Step 4.1: For each anchor $a_i \in c$ compute its lower dimensionality:

                      $d'(a_i) = d/int(len(c(a_i)))$                          (3)

    Step 5: For each cluster c and its reduced dimensionality d'
            Step 5.1: Change NodePiece's Embedding Vector from higher dimensionality d
                      to lower dimensionality $d'$
            Step 5.2: Use the output of the Embedding vector as input into the MLP layer
            Step 5.3: Extract the anchors corresponding to cluster c
            Step 5.4: Use the output of the MLP as input into the NodePiece Encoder
                      Encoder
    Step 6: Analyze how lowering the embedding dimensions of the anchors affects the
            performance of NodePiece
```

5 Experiments

5.1 Hyperparameters

The Idea: Analyze how different dimensionality reductions affect the performance of the NodePiece model. In this section, we will present the experiments conducted given the methods previously mentioned. They are conducted given multiple sets of hyperparameters as presented in Table 1. The training is run on the RTX Titan 24GB VRAM from the Lisa Cluster provided by SurfSara.

5.2 Intermediate Results

The first step is to apply the *degree_centrality*() method which returns one centrality score for each node in the graph. Given that the graph does not have self-loops, the scores are in the range $s_i \in [0, 1]$. In the next step, we apply K-means clustering from *sklearn* on the centrality scores. For finding the most relevant number of clusters, we apply the Elbow Rule as described in Section 4.2. More precisely, we run the K-means algorithm for $k \in [1, 10)$, calculate SSE (Sum of Squared Errors) and consider the first value of k for which its difference compared to the previous iteration is less than 0.0005. The best number of clusters in this example is $k = 4$. Therefore, the clustering algorithm is applied once more for the optimal $k = 4$. It is important to note that the chosen k is data-dependent.

embedding dimension	200
number of anchors	1000 / 500 / 30
k shortest paths	100 / 50 / 20
pool	cat / trf
loop	lcwa / slcwa
epochs	50 / 100 / 200 / 350
batch size	128 / 256 / 512
learning rate	0.0005
loss	bce / nssal
dropout	0.1

Table 1. Hyperparameters

After applying the K-means clustering algorithm, we extract only the anchors. When using the first experimental setup from Table 1, the anchors alone are placed in 3 clusters. More precisely, given the number of anchors 30, the algorithm results in 3 clusters with the number of anchors being $\{5, 10, 19\}$. When adding them the number of anchors exceeds the hyperparameters with value 30, that is because the model adds padding nodes.

The next step is calculating the reduced dimensionality per anchor. Assuming the first 3 anchors are from clusters $\{c3, c1, c3\}$ and the last one from cluster $c2$ where the 3 clusters $\{c1, c2, c3\}$ have lengths $[10, 19, 5]$ where the length of a cluster is represented by the number of anchors in it, then Equation 4 displays the resulting lower dimensionality embedding space.

$$\begin{aligned} d'(a_1) &= int(200/5) = 40 \\ d'(a_2) &= int(200/10) = 20 \\ d'(a_3) &= int(200/5) = 40 \\ &\dots \\ d'(a_n) &= int(200/19) = 10 \end{aligned} \tag{4}$$

where $d'(a_i)$ represents the new reduced length of the current anchor.

5.3 Datasets

Due to limited time and computational resources, we experimented with two datasets, namely FB15k-237 and WN18RR, with a vocabulary size of 15k and 40k respectively.

FB15K-237 The initial setting of the model for this dataset was 1000 anchor nodes with 200 dimensions each. In Table 2 we can observe different dimensions being assigned to clusters and also the number of embedding tensors for each newly computed dimension. The majority of the anchor nodes had their dimensions reduced 10x and only 6 out 1000 anchors had their dimensions unaffected by our selective reduction technique.

FB15K-237			WN18RR		
Cluster ID	Embeddings	New reduced dimension	Cluster ID	Embeddings	New reduced dimension
0	206	1	0	243	1
1	271	2	1	150	2
2	88	3	2	91	3
3	51	4	3	20	10
4	85	5	-	-	-
5	148	6	-	-	-
6	62	7	-	-	-
7	28	8	-	-	-
8	18	12	-	-	-
9	13	16	-	-	-
10	10	20	-	-	-
11	8	25	-	-	-
12	6	67	-	-	-
13	4	100	-	-	-
14	6	200	-	-	-

Table 2. FB15K-237 and WN18RR Datasets

WN18RR The initial setting of the model for this dataset was 500 anchor nodes with 200 dimensions each. In Table 2 we can observe different dimensions being assigned to clusters and also the number of embedding tensors for each newly computed dimension. All of the anchor nodes had their dimensions reduced 20x by our selective reduction technique.

Table 3 compares the results of NodePiece-RotatE together with our selective reduction technique with the standard NodePiece-RotatE model. We used the same hyperparameters as the default implementation. The overall number of parameters is the same and the Mean Reciprocal Rank and Hits@10 are similar to the original NodePiece-RotatE model while the dimensionality of the embedding tensors is significantly lower, around 10 to 20 times lower.

	FB15K-237				WN18RR			
	#P(M)	MRR	Hits@1	Hits@10	#P(M)	MRR	Hits@1	Hits@10
NodePiece-RotatE	3.2*	0.256*	-	0.42*	4.4*	0.403*	-	0.515*
NodePiece-RotatE + Variable Sized Embeddings	3.23	0.25	0.169	0.416	4.41	0.318	0.22	0.478

Table 3. Metric results. * results taken from (Galkin et al., 2021) [7]. #P is a total parameter count (millions).

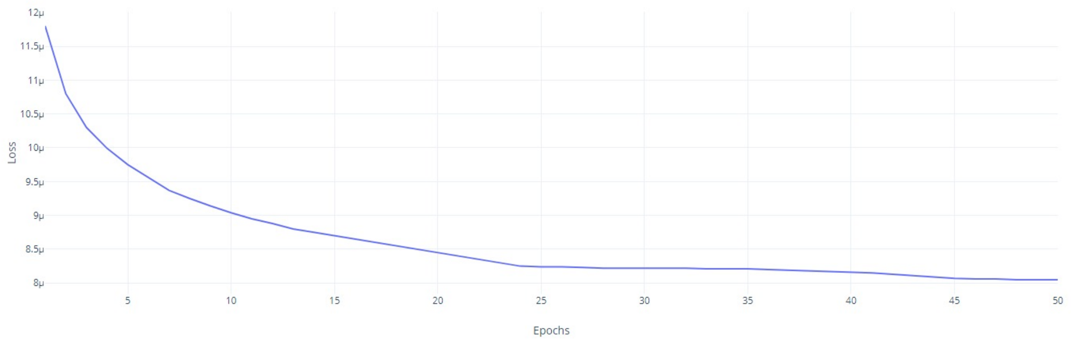


Fig. 5. NodePiece-RotatE BCE loss when training on FB15K-237

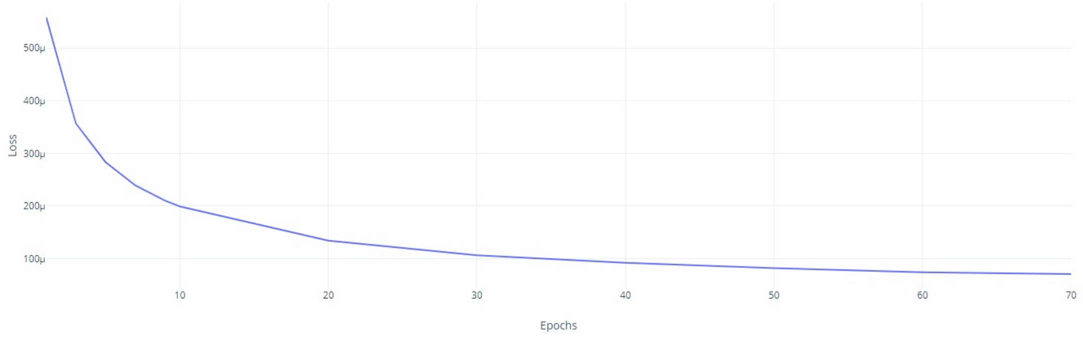
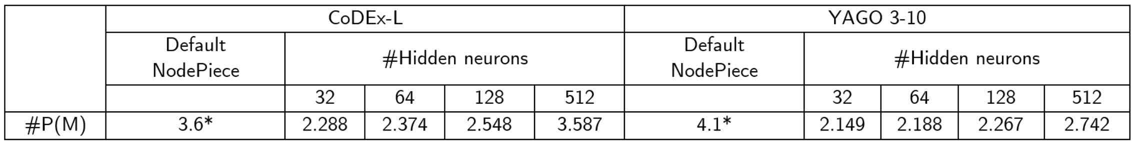


Fig. 6. NodePiece-RotatE NSSAL loss function when training on WN18RR

CoDEx-L & YAGO 3-10 The initial setting of the model for these datasets was 7000 and 10000 anchor nodes with 200 dimensions each. In Table 5 we can observe different dimensions being assigned to clusters and also the number of embedding tensors for each newly computed dimension. The majority of the anchor nodes had their dimensions reduced 10x by our selective reduction technique. Figure 7, presents the number of parameters of our model compared to the original NodePiece implementation. For CoDEx-L, the number of parameters declined to around 65% from its initial value and for YAGO 3-10, there is a drop of around 50%.

6 Future work

An idea for further improving the NodePiece architecture could be implementing the InterHT[14] approach regarding head and tail entities. The RotatE mechanism is representing the head and tail entities as deterministic vectors. This approach bounds the learning capability of models. InterHT proposes a novel distance-based method which can represent the head and tail entities by incorporating the information of the tail and head entities. More precisely, for the head entity, the original representation is multiplied with an auxiliary tail entity vector in order to obtain the final representation of the head entity. The



	CoDEx-L					YAGO 3-10				
	Default NodePiece	#Hidden neurons				Default NodePiece	#Hidden neurons			
		32	64	128	512		32	64	128	512
#P(M)	3.6*	2.288	2.374	2.548	3.587	4.1*	2.149	2.188	2.267	2.742

Fig. 7. Parameter count of default NodePiece vs NodePiece + Variable Sized Embeddings with different amount of hidden neurons. * results taken from (Galkin et al., 2021) [7]. #P is a total parameter count (millions).

CoDEx-L			YAGO 3-10		
Cluster ID	Embeddings	New reduced dimension	Cluster ID	Embeddings	New reduced dimension
0	6769	1	0	9720	1
1	98	3	1	179	2
2	52	4	2	94	3
3	29	7	3	8	25
4	21	10	4	3	200
5	10	20	-	-	-
6	6	34	-	-	-
7	12	50	-	-	-
8	3	67	-	-	-
9	2	100	-	-	-
10	2	200	-	-	-

Table 4. CoDEx-L and YAGO 3-10 Dataset

tail entity can be achieved in the same way as above. A further improvement of NodePiece would be to combine the InterHT approach with the NodePiece entity embeddings.

A future work approach could be to experiment with the new selective reduction technique more thoroughly in the sense of exploring with more hyperparameters values and running longer training sessions.

As discussed above in Section 4, we used an approach that takes all the anchors and feeds them to the MLP in order to get a higher dimensional embedding as output. After this step, we extract the anchors that are requested from the higher dimensional vector and supply them to the encoder of NodePiece. Moreover, the hyper-parameters of the NodePiece-RotatE could be tuned in order to find a proper set that could further improve the performance of the model.

As further improvements, the Kretchmer method [9] can be used as a measure of centrality in order to find influential nodes in a graph, which has been used in social media networks like Twitter, Linkedin and Facebook. Besides the standard approach of degree centrality (measuring the in and out-degree of nodes), the Kretschmer method considers the relation weights based on the number of interactions.

7 Conclusion

In this project, we worked with anchor embeddings from NodePiece that tokenizes each node by K anchor nodes and M relation types in its relational context. Then, the resulting hash sequence is encoded through an MLP or a Transformer[7]. By applying our selective reduction method we observed that the dimensionality of the embedding tensors reduces by approximately 10-20x or even more depending on the shape and characteristics of the Knowledge Graph. The number of parameters of the model has not changed for smaller Knowledge Graphs while for large KGs such as CoDEx-L and YAGO 3-10 there is a significant drop in parameter count. The performance is relatively similar to the initial NodePiece model for the two datasets that were tested. We conclude that the selective reduction of dimensionality resembles a reliable method for improving NodePiece architecture.

References

1. C. Bento. Multilayer perceptron explained with a real-life example and python code: Sentiment analysis. *Medium*, Sep 2021.
2. A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2787–2795, 2013.
3. I. Chami, A. Wolf, D. Juan, F. Sala, S. Ravi, and C. Ré. Low-dimensional hyperbolic knowledge graph embeddings. In D. Jurafsky, J. Chai, N. Schluter, and J. R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 6901–6914. Association for Computational Linguistics, 2020.
4. D. Chen, R. Socher, C. D. Manning, and A. Y. Ng. Learning new facts from knowledge bases with neural tensor networks and semantic word vectors. In Y. Bengio and Y. LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.
5. T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2d knowledge graph embeddings. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1811–1818. AAAI Press, 2018.
6. V. Flovik. What is graph theory, and why should you care? *Medium*, Jan 2022.
7. M. Galkin, J. Wu, E. G. Denis, and W. L. Hamilton. Nodepiece: Compositional and parameter-efficient representations of large knowledge graphs. *CoRR*, abs/2106.12144, 2021.
8. G. Ji, S. He, L. Xu, K. Liu, and J. Zhao. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 687–696. The Association for Computer Linguistics, 2015.
9. Z. A. Rachman, W. Maharani, and Adiwijaya. The analysis and implementation of degree centrality in weighted graph in social network analysis. In *2013 International Conference of Information and Communication Technology (ICoICT)*, pages 72–76, 2013.
10. Z. Sun, Z. Deng, J. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.
11. R. Teja. Knowledge graph embedding-a simplified version! *Medium*, Mar 2022.
12. T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In M. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33nd International*

Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2071–2080. JMLR.org, 2016.

13. S. Vashishth, S. Sanyal, V. Nitin, N. Agrawal, and P. P. Talukdar. Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 3009–3016. AAAI Press, 2020.
14. B. Wang, Q. Meng, Z. Wang, D. Wu, W. Che, S. Wang, Z. Chen, and C. Liu. Interht: Knowledge graph embeddings by interaction between head and tail entities. *CoRR*, abs/2202.04897, 2022.
15. Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In C. E. Brodley and P. Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 1112–1119. AAAI Press, 2014.
16. Wikipedia contributors. Graph theory — Wikipedia, the free encyclopedia, 2022. [Online; accessed 24-July-2022].
17. S. Zhang, Z. Sun, and W. Zhang. Improve the translational distance models for knowledge graph embedding. *J. Intell. Inf. Syst.*, 55(3):445–467, 2020.